

Threat Modeling and Securing a Mobile Agent System for Disaster Response

by
Daniel Hein

A PhD Thesis
Presented to the Faculty of Computer Science and Biomedical Engineering,
Graz University of Technology (Austria),
in Partial Fulfillment of the Requirements for the PhD Degree

Assessors

Prof. Roderick Bloem, Ph.D. (Graz University of Technology, Austria)
Prof. Allan Tomlinson, Ph.D. (Royal Holloway University of London, UK)

November 2016



Institute for Applied Information Processing and Communications (IAIK)
Faculty of Computer Science
Graz University of Technology, Austria

Abstract

In this thesis we create threat models for disaster response, devise and implement a security solution for a PC based mobile agent system for supporting disaster response, use one of our threat models to evaluate the effectiveness of our security solution, and finally take first steps towards porting our security solution to mobile devices.

In disaster response, the use of information and communication technology promises to help saving lives. First, information and communication technology can support the decision making process in disaster response by facilitating information gathering and analysis. Second, it can speed up the implementation of these decisions through the use of new information-technology-based services. Disaster response imposes harsh conditions on information technological aids. One potential problem is a damaged or overloaded communication infrastructure. Mobile software agents are one approach that is particularly well suited to such conditions, because mobile agents can act autonomously on behalf of a human operator and they can migrate to different hosts. These abilities enable information gathering and analysis, or implementing a disaster response plan, while mitigating intermittent communication losses.

Securing a mobile agent system for disaster response poses a number of challenges. Given the nature of disaster response, only authenticated and authorized personnel should have access to the services provided by the mobile agents. Furthermore, only authorized mobile agents should be loaded into the system. In addition, the mobile agent platforms providing the common substrate for executing and migrating agents must ensure uninfluenced execution of the agents. Thus securing the mobile agent system translates to fulfilling the following requirements. First, security requires data integrity and authenticity on all data input and output, including requests to services and the agents themselves. Second, it requires confidentiality, data integrity and authenticity on all data transmitted within the system. Finally, it requires code integrity of the hosts comprising the agent system.

Towards refining and fulfilling these requirements we model the threats to a particular, existing mobile agent system for disaster response support, the Secure Agent Infrastructure. We then introduce two security mechanisms geared towards mitigating the identified threats and integrate these two security mechanisms with the Secure Agent Infrastructure. We then analyze the effectiveness of our mechanisms by contrasting their security properties with the identified threats.

The two security mechanisms we add are the Secure Docking Module and the

Trusted Docking Station. The Secure Docking Module is a hardware security module that helps authenticating and authorizing human personnel by providing protected storage for credentials. The Secure Docking Module only grants access to these credentials if the host computer can attest to its load-time code integrity. The Trusted Docking Station is a PC platform that provides a load-time integrity protected execution environment. The Trusted Docking Station is based on the acTvSM platform [Pir15]. The Trusted Docking Station uses the acTvSM platform to ensure and attest its load-time code integrity, thus helping to protect the uninfluenced execution of the mobile agents.

In an effort to make our security mechanisms available on smartphones and tablets, our final contribution is the Secure Block Device. The Secure Block Device is a software component for establishing a secure, efficient and easy to use confidential and authentic data store. As such, it is a core component of a secure mobile agent platform for smart mobile devices equipped with ARM TrustZone.

Our combined contributions cover key factors for enabling mobile agent systems for disaster response. Thus, we believe that our contributions can help to secure future disaster response activities.

Acknowledgements

This thesis is the culmination of many years of work during which the lives of numerous people entangled with mine. Science today is not the effort of singular persons but of groups of people collaborating. I want to thank the people I have collaborated with; without their contribution this thesis would not be what it is now. I also want to express my gratitude towards my friends and family. Without their support, I am not sure if I could have finished this thesis.

First and foremost I would like to thank Roderick Bloem for being an exemplary supervisor and mentor. Roderick always found the time for discussions and for giving qualified and valuable critique. Roderick, you have contributed significantly to my academic and personal growth; thank you for your guidance and support over the years.

I also would like to thank Allan Tomlinson for finding the time and agreeing to be the external reviewer of this thesis. Thank you so much.

I also want to extend my deepest gratitude towards Johannes Winter for always having time, for all the ideas we bounced around, for all the long talks and discussions, for helping me find the holes in my harebrained schemes without shooting them down, for all the help with anything ARM related, including our Secure Block Device experiments, and for being a true friend.

I am also thankful to Martin Pirker for his love of privacy and for his perspective on Trusted Computing. Furthermore, I would like to thank Ronald Tögl for his collaboration, sharing his vast knowledge on Trusted Computing, and helping me getting into the field. Also, I would like to thank Ronald and Martin for developing the acTvSM platform and their help in integrating it with the Secure Docking Module and the Trusted Docking Station.

I would also like to thank Michael Gissing, Nina Mocnik, and Thomas Kastner for helping me tame our zoo of virtual machines for the acTvSM platform. Furthermore, I would like to thank Andreas Fitzek for ANDIX OS and for helping me integrate the Secure Block Device.

I would also like to express my sincere gratitude to all my friends and colleagues from IAIK. I miss the interesting talks over coffee, the intellectually inspiring discussions, and your friendship. On a similar note, I want to thank the institute itself for providing the best and most positive working environment I have ever known.

I would also like to extend my deepest thanks towards the team at Café Zapo. Specifically, I would like to thank you for a warm and welcoming little island of calm, your unparalleled service, and for consistently providing the best food in the history of Cafés. I am at a loss of words to describe how much I miss you

guys!

I'd also like to thank my parents Sophie Raggam and Johann Hein. There is so much I want to thank you for: for bringing me into this world in the first place; for putting up with me; for supporting my drive to do stuff with these newfangled personal computers; for your never ending love and support, and also for looking after the kids when the work on this thesis tore away at my spare time. I am eternally grateful to you.

I also want to thank my father-in-law Franz Josef Klammingner for all the big and little things, from literally and figuratively helping keeping the house standing, to bringing my eldest to school, while I am writing these lines. Thank you so much.

I also want to acknowledge my kids, Leopold and Nepomuk, for missing out on their dad. Writing these lines feels a little bit like handing out these T-Shirts with texts like: my father wrote a thesis and all we got is a few lines in the acknowledgements. The sentiment is correct. These few lines are inadequate in every way. They neither express my deep love for you, nor can they make up for even one iota of Planck time not spent together. However, this thesis might never have been without you and I thank you for this.

My highest esteem and deepest gratitude go to my wife Karima Hein. There is so much I want to praise you for, and now after 300 pages I still cannot find the words; all the superlatives of the world seem insufficient. You have my gratitude and my heart for your love, for our two kids, for being there during thick and thin; for supporting me writing this thesis; for understanding when I had to work long hours; for everything. My life would be a bleak existence, and this thesis would never have taken form without you. For all this and much more my deepest, sincerest unending thanks.

Daniel Hein
Graz, October 2016

Table of Contents

Abstract	iii
Acknowledgements	v
List of Tables	xiii
List of Figures	xvii
Acronyms	xxi
1 Introduction	1
1.1 Motivation	1
1.1.1 The Mobile Software Agent Paradigm	4
1.1.2 Use of Smart Mobile Devices for Disaster Response	5
1.2 Goals	6
1.3 Problem Description	6
1.3.1 Threat Modeling	6
1.3.2 Threat Mitigation	7
1.3.3 Secure Data Storage	8
1.4 Contributions	8
1.4.1 Contribution 1 - Threat Modeling	8
1.4.2 Contribution 2 - The Trusted Docking Station and the Secure Docking Module	12
1.4.3 Contribution 3 - Secure Data Storage for TrustZone Sys- tems	13
1.5 Outline	14
2 Preliminaries	17
2.1 Agents	17
2.1.1 Properties of Agents	17
2.2 Multi-Agent Systems	19
2.2.1 Mobile Agent Systems	19
2.3 Mobile Agents Properties	20
2.4 Mobile Agent Security	20
2.4.1 Jansen and Karygiannis	21
2.4.2 Borselius	22
2.4.3 Bierman and Cloete	24

2.4.4	Discussion	25
2.5	Jini	26
2.5.1	Jini Services	27
2.5.2	Communication	27
2.5.3	Service Discovery	27
2.6	Secure Agent Infrastructure	30
2.6.1	Overview	30
2.6.2	Example Use Case	32
2.6.3	Distributed Secure Agent Platform	35
2.6.4	Properties of Secure Agent Infrastructure Agents	37
2.6.5	Security Architecture of the Secure Agent Infrastructure	38
2.6.6	Resilience against Temporary Communication Disruptions	38
2.6.7	Agent authenticity	38
2.6.8	Agent transport	38
2.6.9	Agent communication	39
2.6.10	Remarks	39
2.7	Security Goals	39
2.7.1	Confidentiality	39
2.7.2	Integrity	40
2.7.3	Availability	40
2.8	Threat Modeling	40
2.8.1	Definitions	40
2.8.2	Threats	41
2.8.3	The Microsoft Security Development Lifecycle Threat Modeling Methodology	43
2.8.4	The Microsoft Threat Modeling Tool Family	44
2.8.5	Threat Enumeration	45
2.9	Cryptographic Primitives	50
2.9.1	Cryptographic Hash Functions	51
2.9.2	Message Authentication Codes	53
2.9.3	Authenticated Encryption	53
2.10	Single-User Block Datastore Authentication	55
2.11	Attacks on Memory Authentication	57
2.12	Merkle Trees	62
2.12.1	Description	62
2.12.2	Merkle Trees and Data Integrity	62
2.13	Security Controllers	63
2.14	Trusted Platform Module	64
2.14.1	Introduction	64
2.14.2	Storing Platform Integrity Measurements	65
2.14.3	Secure Key Storage	66
2.14.4	Sealing	66
2.14.5	Remote Attestation	66
2.15	Intel Trusted eXecution Technology	67
2.15.1	Software Integrity Measurement	68

2.15.2	Late Launch	68
2.15.3	Load-Time Integrity	69
2.15.4	Security	70
2.16	The acTvSM Platform	71
2.17	ARM TrustZone	72
2.18	ANDIX OS	74
2.18.1	Inter-world Communication	76
2.18.2	Root of Trust for Storage	76
2.18.3	Sharing Resources with the Normal World	77
3	Related Work	79
3.1	Threat Modeling	79
3.1.1	STRIDE	79
3.1.2	Other Threat Elicitation Methodologies	83
3.2	Mobile Agents and Disaster Response	84
3.2.1	Mobile Agent System Security	85
3.3	Trusted Computing	86
3.4	Secure Data Storage	87
4	Threat Modeling Aspects of Disaster Response	91
4.1	Introduction	91
4.2	Secure Agent Infrastructure Model	93
4.3	Situational Awareness	96
4.3.1	Model Description	96
4.3.2	Threat Modeling Results	97
4.3.3	Mobile Agent System Level Model	102
4.4	Command and Control	104
4.4.1	Model Description	104
4.4.2	Threat Modeling Results	105
4.5	Distributed Secure Agent Platform Outpost	109
4.5.1	Overview	109
4.5.2	Description of the Model	110
4.5.3	Secure Agent Infrastructure Security Assumptions	111
4.5.4	Threat Modeling Results	113
4.6	Conclusions	117
4.6.1	Summary	117
4.6.2	High Level Threat Models	119
4.6.3	Using STRIDE-per-interaction for Threat Modeling a Mo- bile Agent Platform	119
4.6.4	Using STRIDE-per-interaction for Modeling High Level Processes in Disaster Response	120
4.7	Future Work	122
4.7.1	Risk Modelling	122
4.7.2	Enhanced Threat Modeling for the Distributed Secure Agent Platform Outpost	123

5	The Trusted Docking Station and the Secure Docking Module	125
5.1	Introduction	125
5.2	Objectives	127
5.3	Mode of Operation	128
5.4	Application	130
5.4.1	Communication Key Protection	130
5.4.2	Authentication Credential Protection	131
5.4.3	Agent Authorization Key Protection	131
5.5	Architecture	132
5.6	The Trusted Docking Station	135
5.6.1	The Outpost Appliance	135
5.6.2	The Distributed Secure Agent Platform Software	136
5.7	The Secure Docking Module	137
5.7.1	Overview	137
5.7.2	The Resource Access Protocol Setup	137
5.7.3	The Session Establishment Protocol	137
5.7.4	The Resource Access Protocol	139
5.7.5	Secure Docking Module Implementation	142
5.8	Performance Evaluation	143
5.9	Security Evaluation	145
5.9.1	Adversarial Model	145
5.9.2	Threat Model Mitigation	146
5.10	Conclusion	149
5.11	Future Work	150
6	Secure Block Device	151
6.1	Introduction	151
6.1.1	Motivation	152
6.1.2	Contribution	153
6.1.3	Properties of the Secure Block Device	153
6.1.4	Evaluation	155
6.2	The Secure Block Device Operation Principle	155
6.2.1	Creating a Secure Block Device	156
6.2.2	Opening a Secure Block Device	156
6.2.3	Using a Secure Block Device	156
6.2.4	Closing a Secure Block Device	160
6.3	The Secure Block Device Architecture	161
6.3.1	The Block Device Abstraction Layer	161
6.3.2	Cryptography Abstraction Layer	161
6.3.3	The Block Cache	162
6.3.4	The Merkle Tree	163
6.3.5	Secure Block Device API	165
6.4	Security Evaluation	165
6.4.1	Adversarial Model	165
6.4.2	Threat Model	167
6.4.3	Discussion	167

6.5	Experimental Evaluation	170
6.5.1	Code Size	170
6.5.2	Secure Block Device Block Cache Performance for Small Files	170
6.5.3	Experimental Setup	172
6.5.4	Impact of the Authenticated Encryption Scheme	173
6.5.5	Normal World Filesystem Encryption using the Secure Block Device	174
6.6	Future Work	177
6.7	Conclusion	178
7	Conclusions	179
7.1	Threat Modeling	179
7.2	The Trusted Docking Station and the Secure Docking Module . .	180
7.3	The Secure Block Device	181
7.4	Outlook	181
7.5	Final Words	183
A	Threat Modeling Agent Migration and Communication	185
A.1	Introduction	185
A.2	Threat Analysis Procedure	185
A.3	Distributed Secure Agent Platform Outpost Model	186
A.3.1	Structure	186
A.3.2	Modelled Workflows	188
A.3.3	Limitations of the Secure Agent Infrastructure Compared to a Generic Mobile Agent System	188
A.4	Secure Agent Infrastructure Security Assumptions	189
A.4.1	Secure Core Secure Agent Infrastructure	189
A.4.2	Secure Mobile Agents	189
A.4.3	Secure Communication Channel	189
A.4.4	Distributed Secure Agent Platforms and External Interac- tor	190
A.5	Agent Migration	190
A.5.1	Send Mobile Agent	190
A.5.2	Instantiate Agent	195
A.6	Agent – External Interactor Communication	200
A.6.1	Input	201
A.6.2	Output	206
A.7	Agent – Core Secure Agent Infrastructure Communication	208
A.7.1	MA \rightarrow DSAP	208
A.7.2	DSAP \rightarrow CSAI	212
A.7.3	CSAI \rightarrow DSAP	213
A.7.4	DSAP \rightarrow MA	215

B Security Evaluation	219
B.1 Introduction	219
B.2 Agent Migration	219
B.3 Agent - External Interactor Communication	222
B.4 Agent - Core Secure Agent Infrastructure Communication	227
C Threat Modeling the Secure Block Device	233
C.1 Introduction	233
C.2 Threat Model	234
C.3 Adversarial Model	238
C.4 List of Threats	238
D List of Publications And Cooperations	245
D.1 Journal Publications	245
D.2 Published Book Chapters	246
D.3 Publications in Conference and Workshop Proceedings	246
D.4 Relationship between Publications and Thesis	249
Bibliography	253
Index	271
Author Index	273

List of Tables

2.1	Description of the elements in the Microsoft Security Development Lifecycle threat modeling methodology’s diagramming system. The diagramming system is based on Data Flow Diagrams, so this description encompasses Data Flow Diagram elements, as well as the trust boundary element added by the threat modeling methodology. This table is based on a table in [Sho08].	44
2.2	The mapping of Data Flow Diagram (DFD) element types to specific threat categories in the STRIDE-per-element threat enumeration technique. The threat categories are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege. The question mark for the Data store under Repudiation signifies that data stores need to be treated differently, if they are used for authentic logging. In such a case repudiation threats need to be considered. This table is adapted from [Sho14].	46
2.3	The mapping of Data Flow Diagram (DFD) element types, information flow destinations, and interaction types to specific threat categories in the STRIDE-per-interaction threat enumeration technique. The threat categories are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege. This table is adapted from [Sho14].	47
2.4	An adapted excerpt of the report generated by the Microsoft Threat Modeling Tool 2016 for the system modeled by the Data Flow Diagram depicted in Figure 2.8. This table lists the 11 potential threats and their description as enumerated by the tool for the interaction <i>Input</i> between <i>External</i> and <i>Process 1</i> in Figure 2.8. These are only 11 of the 53 threats proposed for analysis by the tool for the overall system model.	48
4.1	A consolidated list of threats created by using the STRIDE-per-interaction threat enumeration method on the model depicted in Figure 4.2. The <i>Type</i> column uses the STRIDE threat categories. These are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege.	99

4.2	A consolidated list of threats created by using the STRIDE-per-interaction threat enumeration method on the model depicted in Figure 4.4. The <i>Type</i> column uses the STRIDE threat categories. These are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege.	106
4.3	Threats to agents migrating from the Core Secure Agent Infrastructure to a Distributed Secure Agent Platform Outpost	114
4.4	Threats to a Mobile Agent communicating with an External Interactor	115
4.5	Threats to a Mobile Agent communicating with the Core Secure Agent Infrastructure using the Distributed Secure Agent Platform’s communication facilities	116
5.1	The configuration parameters required by the Secure Docking Module’s session establishment protocol	138
5.2	The configuration parameters required by the Secure Docking Module’s resource access protocol	138
5.3	Trusted Docking Station/Secure Docking Module performance evaluation results. The access SDM resource action consists of the TPM_Quote2, quote verification, copy resource to host, and miscellaneous overhead table entries.	144
6.1	Secure Block Device application programming interface (API) . .	166
6.2	Threats to a Trusted Application when reading and writing data to and from an untrusted Datastore	168
6.3	Average SBD read/write times and throughput depending on Authenticated Encryption (AE) scheme	173
6.4	Compile times for compiling OpenSSL on Secure Block Device protected block devices. The Ramdisk row is the baseline, where we compiled OpenSSL on an Network Block Device backed by a Ramdisk, with no Secure Block Device in place. The next four lines give the results for a Network Block Device using the Secure Block Device and backed by a Ramdisk with different AE schemes. Finally the last two lines present the results for a Network Block Device that uses a Trusted Application (TA) to implement the Secure Block Device in the Secure World. The output of the Secure Block Device Trusted Application is again stored in a Ramdisk in the Normal World. These last two lines reflect the results of the setup depicted in Figure 6.6.	176
B.1	This table discusses how our security solution comprising the Trusted Docking Station and the Secure Docking Module mitigates the threats to agents migrating from the Core Secure Agent Infrastructure to a Distributed Secure Agent Platform Outpost .	220

B.2	This table contains a description if and how our Distributed Secure Agent Platform Outpost (DSAP Outpost) security solution consisting of the Trusted Docking Station and the Secure Docking Module mitigates threats to a Mobile Agent communicating with an External Interactor	222
B.3	Mitigation status of threats to a Mobile Agent communicating with the Core Secure Agent Infrastructure using the Distributed Secure Agent Platform’s communication facilities. Here we investigate how our DSAP Outpost security solution comprising the Trusted Docking Station and the Secure Docking Module mitigates threats pertaining to DSAP Outpost - Core Secure Agent Infrastructure communication.	228
C.1	Threats generated by the Microsoft Threat Modeling Tool 2016 for the second model (see Figure C.2) that have no direct representation in the third model	236
C.2	Threats generated by the Microsoft Threat Modeling Tool 2016 for the third model (see Figure C.3) that have no direct representation in the second model	236
C.3	List of potential threats, when a Trusted Application reads data from an untrusted Datastore.	238
C.4	List of potential threats, when a Trusted Application writes data to an untrusted Datastore.	241

List of Figures

1.1	The crisis management decision making process as illustrated by O’Neill et al. [OSZW12]	2
1.2	The Distributed Secure Agent Platform Outpost	10
2.1	The Mobile Agent System model used by Jansen and Karygianis [JK99] to discuss the security issues of Mobile Agent Systems. This model depicts an agent migrating from its home platform to another platform over a network that connects the platforms of the modeled Mobile Agent System. This specific version of the model is adopted from [JK99].	21
2.2	Jini lookup service discovery according to the Jini specification [Apa16b]. Here a service provider is looking for a <i>lookup service</i> to advertise its service.	28
2.3	A Jini service registering with a <i>lookup service</i> using the join protocol according to the Jini specification [Apa16b].	28
2.4	A Jini client looks up a specific service using the service proxy type and attributes. A copy of the service proxy is delivered to the client to enable communication between the client and the service provider [Apa16b].	29
2.5	A Jini client interacts with a service using the service proxy. This figure is adapted from [Apa16b].	30
2.6	This crisis mitigation workflow example illustrates how the components of the Secure Agent Infrastructure interact with crisis management personnel to help mitigating a swine flu epidemic. This example follows a single process that governs the distribution of more vaccine to the regional offices of the national health agency. In this example Mobile Agents communicate with users and retrieve information from otherwise incompatible legacy systems. The example is adopted from Emil Gatial’s dissertation [Gat10a].	33

2.7	The Distributed Secure Agent Platform component of the Secure Agent Infrastructure is a Jini service handling Mobile Agent transmission, client-agent communication and Mobile Agent execution [GBH10]. A newly instantiated Distributed Secure Agent Platform Service (DSAP Service) first uses Jini's discovery and join protocols to register with a Jini lookup service. Then clients, such as the Process Management Subsystem, can find the DSAP Service and request a service proxy to send agents to it and allow agents to communicate with the client through the service proxy and vice versa.	36
2.8	A Data Flow Diagram of a simple system illustrating all possible STRIDE-per-interaction interactions listed in Table 2.3. We have emphasized the Input data flow, as we use it in an example to detail how STRIDE-per-interaction works.	48
2.9	Spoofing attack on a Block Datastore	58
2.10	Splicing attack on a Block Datastore	59
2.11	Replay attack on a Block Datastore	60
2.12	Forking attack on a Block Datastore	61
2.13	A perfect binary Merkle Tree. The inner nodes are denoted by N_k , where $0 < k \leq 7$, and the leave nodes are denoted by L_i , where $0 < i \leq 8$. The inner nodes N_k of the tree are computed by applying a hash function h to the concatenation ($ $) of its immediate children.	62
2.14	Architecture of the ANDIX OS	75
4.1	Our high level model for gathering information and issuing commands in disaster response	95
4.2	A Data Flow Diagram modeling gathering intelligence for informing a command decision during the response phase of disaster response. We use this model as input to the STRIDE-per-interaction analysis using the Microsoft Threat Modeling Tool 2016.	97
4.3	A model of the information gathering process for situational awareness that incorporates Secure Agent Infrastructure components. This model exhibits a highly repetitive pattern, the Distributed Secure Agent Platform Outpost. The DSAP Outpost comprises a Distributed Secure Agent Platform hosting a Mobile Agent that interacts with an External Interactor.	103
4.4	A Data Flow Diagram modeling command and control during the response phase of a crisis. We use this model as input for a STRIDE-per-interaction threat enumeration using the Microsoft Threat Modeling Tool 2016.	104

4.5	A Data Flow Diagram modeling the Distributed Secure Agent Platform Outpost (DSAP Outpost). A DSAP Outpost is at the core of the Secure Agent Infrastructure typical process of sending a Mobile Agent to a DSAP to gather information from an external information system, or converse with a system user. Here we model both, the agent migration and the information exchange between the Core Secure Agent Infrastructure (CSAI) and the Mobile Agent (MA) running on the Distributed Secure Agent Platform (DSAP). We use this model as input to the STRIDE-per-interaction analysis using the Microsoft Threat Modeling Tool 2016.	110
5.1	Usage scenarios for TDS and SDM in conjunction with a DSAP Outposts	129
5.2	The Trusted Docking Station architecture. The red lines delineate Virtual Machine against other Virtual Machines and against the acTvSM Base System Virtual Machine Monitor. The black line separates hardware from software components.	132
5.3	Secure Docking Module session establishment protocol	139
5.4	Secure Docking Module resource access protocol	141
5.5	Hardware implementations of the Secure Docking Module: the security controller (front) and the assembled token prototype with USB interface in an epoxy sealed casing (back).	143
6.1	The component partitioning diagram for a Trusted Application that uses the Secure Block Device to store data. The diagram details which components reside in the Secure World and which in the Normal World. It also specifies which data is kept in RAM and which is persisted in a Block Datastore.	157
6.2	Block structure for a Secure Block Device with a block size of 2 KiB	159
6.3	Secure Block Device architecture	161
6.4	Dynamically growing a Merkle Tree	164
6.5	Secure Block Device (SBD) Block Cache Access Times for Small Files	171
6.6	Using the SBD for Normal World file system encryption	175
A.1	A Data Flow Diagram modelling the Secure Agent Infrastructure typical process of sending a Mobile Agent to a Distributed Secure Agent Platform to gather information from an external information system, or converse with a system user. Here we model both, the agent migration and the information exchange between the Core Secure Agent Infrastructure and the Mobile Agent running on the Distributed Secure Agent Platform. We use this model as input to the STRIDE-per-interaction analysis using the Microsoft Threat Modeling Tool 2016.	187

A.2	The part of our Data Flow Diagram model we use to analyze the threats to a Mobile Agent when it is being sent to an DSAP Outpost and instantiated there.	191
A.3	A bug called Feature	194
A.4	The part of our Data Flow Diagram model we use to analyze the threats to a Mobile Agent when interacting with an External Interactor. An External Interactor can be a human being, or an external information system.	200
A.5	The part of our Data Flow Diagram model we use to analyze the threats to a Mobile Agent when communicating with the Core Secure Agent Infrastructure using the DSAP's communication facilities.	209
C.1	The first model we considered for modeling a Trusted Application storing its data in an untrusted Datastore.	234
C.2	The second model we considered for modeling a Trusted Application storing its data in an untrusted Datastore in the Normal World.	235
C.3	The third model we considered for modeling a Trusted Application storing its data in an untrusted Datastore.	235

Acronyms

- AE** Authenticated Encryption. xiv, 50, 53, 54, 55, 87, 88, 154, 155, 156, 158, 160, 161, 165, 167, 169, 173, 174, 175, 177, 178, 181, 182
- CMAC** Cipher-based Message Authentication Code. 53, 155, 156, 159, 160
- DSAP** Distributed Secure Agent Platform. xviii, xx, 9, 10, 12, 19, 26, 27, 29, 30, 31, 34, 35, 37, 38, 94, 102, 109, 110, 109, 111, 112, 113, 118, 119, 120, 126, 127, 133, 134, 146, 185, 186, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 219, 222, 227, 228
- DSAP Service** Distributed Secure Agent Platform Service. xvii, 35, 132, 133, 135, 136, 143, 147, 149, 150
- DSAP Outpost** Distributed Secure Agent Platform Outpost. xiv, xv, xviii, xix, 8, 9, 10, 11, 12, 13, 14, 92, 93, 102, 109, 110, 109, 111, 112, 113, 117, 118, 119, 120, 122, 123, 124, 125, 126, 127, 128, 130, 131, 135, 136, 137, 143, 145, 146, 147, 148, 149, 150, 180, 181, 182, 185, 190, 200, 205, 219, 222, 227, 228
- IC** Integrated Circuit. 63
- IV** initialization vector. 54, 55, 158, 159, 161, 163, 167, 169
- MAC** Message Authentication Code. 40, 50, 52, 53, 54, 55, 159, 161, 173, 178
- NVRAM** Non-Volatile Random Access Memory. 64
- OCB** Offset Codebook Mode. 54, 55, 154, 161, 170, 173, 175, 177
- OS** operating system. 73, 74, 76, 132, 152, 174, 177
- Outpost Appliance** DSAP Outpost Application Virtual Machine. 132, 134, 135, 136
- PCR** Platform Configuration Register. 65, 66, 68, 69, 72, 85, 140, 141

SDM Secure Docking Module. 12, 13, 14, 37, 38, 63, 64, 85, 86, 87, 92, 118, 119, 126, 127, 130, 131, 132, 133, 134, 135, 136, 137, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 180, 181, 182, 185, 219, 222, 250

SIV Synthetic Initialization Vector. 54, 55, 154, 156, 161, 170, 173, 175

TDS Trusted Docking Station. 12, 13, 14, 37, 38, 64, 67, 71, 85, 86, 87, 92, 118, 119, 126, 127, 128, 130, 131, 132, 133, 134, 135, 136, 137, 139, 140, 141, 143, 144, 146, 147, 148, 149, 150, 151, 180, 181, 182, 185, 219, 222

TPM Trusted Platform Module. 52, 64, 65, 66, 67, 68, 69, 70, 71, 76, 85, 86, 128, 132, 133, 135, 136, 139, 140, 141, 143, 145

TXt Trusted eXecution Technology. 7, 12, 13, 64, 67, 68, 69, 70, 71, 72, 151

1

Introduction

In this thesis we model the threats to a Mobile Agent System for use in disaster response, and we introduce several security mechanisms to mitigate these threats. Using a Mobile Agent System for disaster response offers many advantages [HBG10], but disaster response also has a strong need for communication confidentiality, integrity and availability [OSZW12]. However, Mobile Agent System security has been identified as a major obstacle for the adoption of Mobile Agent Systems [Rot04]. In the sequel we will motivate the need for communication security in disaster response and also discuss the threats to security when using a Mobile Agent System. We will then state the problems we tackle in this thesis and conclude by outlining our contributions to the field.

1.1 Motivation

Crisis management for natural disasters can greatly benefit from modern information and communication technology. This is especially true for the response phase of crisis management, which is the most communication-heavy phase of crisis management [OSZW12]. Information exchange is essential when emergency plans are put into action and first responders perform rescue operations in real time to stabilize the crisis situation.

Security is essential for disaster response information and communication. A recent study conducted by O'Neill et al. [OSZW12] reviewing the security requirements of emergency services during disaster response documented the need for communication *availability*, *confidentiality* and *integrity*. The need for availability of communication is obvious, as first responders need to be able to exchange information in real time to coordinate their work and gain situational

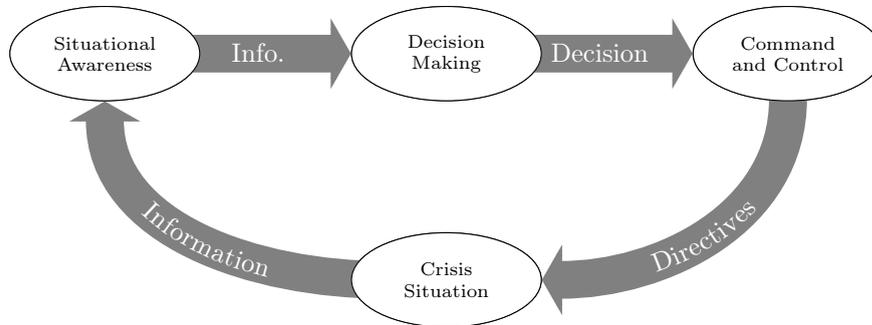


Figure 1.1: The crisis management decision making process as illustrated by O’Neill et al. [OSZW12]

awareness. The confidentiality and integrity of the information exchanges also play a major role. Note that integrity includes origin integrity or *authentication* and *non-repudiation*. Authentication is essential for *authorizing* access to confidential data or services. Finally, non-repudiation prevents decision makers from later repudiating command decisions.

To better illustrate the need for confidentiality and integrity, including authentication, authorization, and non-repudiation, in disaster response we introduce two examples for common processes during disaster response. The first process example demonstrates the need for security when gathering information about a crisis situation. As Figure 1.1 shows, *situational awareness* is a key element of the crisis management decision making process. Situational awareness is the perception, processing and comprehension of critical aspects of the environment. Situational awareness is critical to decision-makers in complex and dynamic areas such as disaster response. The second process we want to give an example for is *implementing a command decision*. Implementing a command decision is another key element of the crisis decision making process, and as we want to illustrate, it requires security.

We place our situational awareness and command decision implementation examples in the following fictional scenario. Our scenario is a wildfire that has already engulfed and subsequently destroyed a number of smaller settlements. The wildfire is now threatening a major city, that hosts a large chemical plant, which exacerbates matters. Emergency services have already responded, fire-fighters have begun fire suppression, emergency medical services treat injured people, and evacuation of settlements including parts of the threatened city has begun.

Situational Awareness Our first example highlights the importance of communication security for gaining situational awareness. Having first class field intelligence is essential for effective disaster response on the strategic, tactical and operational levels. In their study, O’Neill et al. note that 59% of all information exchanges during disaster response pertain to situational awareness.

This illustrates how important situational awareness is for disaster response and emphasizes the need for communication *availability*. On a strategic level meteorological, geological, and vegetation conditions can factor into a wildfire model for predicting its propagation, whereas the information that a number of people need to be evacuated from a structurally unsound building is important on both a tactical and, in more detail, on an operational level. From a security point of view all this information that helps building a better understanding of the disaster and its consequences needs to be authentic and its integrity intact, that is, we require communication *integrity*.

Confidentiality however is also a major requirement for gathering information. In our scenario the wildfire threatens a major city that hosts a chemical plant. The current production of the plant requires base chemicals that when handled incorrectly, for example, when the storage tanks are hit by a raging wildfire, can form a noxious gas cloud that can severely harm living beings in a wide area. Media work is an important aspect of disaster response [OSZW12]. One key aspect here is when, and how, information is disclosed to the public. The threat of a noxious gas cloud, when leaked, has the potential to induce a panic in the inhabitants of the major area around the threatened city, which in turn can severely hamper evacuation efforts.

Implementing a Command Decision The command decision implementation example concerns suppressing the wildfire. The wildfire in our scenario burns on a broad front and strategic and tactical commands decide where to best concentrate suppression efforts. Given a limited number of fire fighters and equipment available, and the sheer scope of the crisis, deciding to focus on one section of the wildfire means that suppression of other sections will degrade. This is in the nature of disaster response, the different level of commands have to decide how to best concentrate their efforts. But this means that someone with the right authority has to make these decisions. So, this example deals with the other end of the stick, the authority to issue commands. Based on the situational awareness strategic, tactical and operational command direct the disaster response operations and allocate resources to specific tasks. These resources can then not be used otherwise with potential ramifications for human lives. Therefore a clear command structure is important, and only entities who have the authority and consequently also shoulder the responsibility, are allowed to control these operations. From a security point of view we need authentication of authorized decision makers, the ability to prove their authority when a decision is implemented, and finally a mechanism for documenting these decisions. All these requirements call for *integrity* protection mechanisms and of course communication *availability*.

These two examples demonstrate the need for confidentiality, integrity, and availability in disaster response information exchanges and corroborate the findings of O'Neill et al. [OSZW12]. To reiterate, during a disaster the confidentiality, integrity and authenticity of all exchanged information needs to be protected, and when implementing a decision based on the gathered information,

the decision must have been made by a person who has the authority to do so. Furthermore, once a decision has been made, the responsible person must not be able to repudiate it. All the while, communication availability has to be retained otherwise there will not be any information exchange.

1.1.1 The Mobile Software Agent Paradigm

As is evident from the situational awareness and implementing a command decision examples, disaster response is a complex effort spanning three levels of command and numerous concurrent activities implementing plans prepared during the crisis preparation phase. Several independent research groups have pointed out the usefulness of the software agent paradigm for disaster response [CGHH89, SPJ⁺03, SMT⁺05, SPA⁺06, MMRM09]. Software agents are specialized programs implementing tasks on the behalf of persons. Software agents usually operate in Multi-Agent Systems, where they interact with other agents and systems. Multi-Agent Systems consist of a number of agent platforms that house and execute the software agents. In this work we focus on enhancing the security of a Mobile Agent System for supporting disaster response operations. Mobile Agent Systems are Multi-Agent Systems that use mobile software agents that are able to migrate between different execution platforms.

Mobile Agents are a distributed computing paradigm that emphasizes autonomy of software components. Mobile Agents are programs that are able to migrate their code and state between platforms. They can exercise an individual's or organization's authority, work autonomously toward a goal and socialize with other agents [JK99]. Mobile Agents localize problem solving, and are therefore well suited to operate in environments where intermittent network failures are to be expected. This makes Mobile Agents well suited for disaster response, which often happens under adverse conditions, and where constant, high bandwidth communication is not always a given. For example, Mobile Agents can replicate themselves to multiple platforms simultaneously to increase their chances of reaching their destination, or simply wait on any given platform until a link to their next destination becomes available.

One of the most prominent, if not the most prominent obstacle for adopting the Mobile Agent paradigm, is security [Rot04]. In a nutshell, the security concerns for Mobile Agent Systems arise from two of its fundamental properties. First, a platform owner in a Mobile Agent System executes potentially untrusted code, the Mobile Agents migrating to *his* or *her* platform. Second, an agent owner sends *her* or *his* agent to potentially untrusted remote agent platforms for execution. In addition to trusting the remote platform, there is also the fact that an agent will be executing in parallel with other agents, from, again, potentially untrusted sources. Also, in addition to the Mobile Agent platform, the remote platform might run other programs, which, again, might follow their own nefarious agenda. Finally, there is also the operator, or administrator of the remote platform to contend with. The fact that *your* agent might transport sensitive data of third parties to that platform only serves to exacerbate the problem.

Many researchers have studied the problem of Mobile Agent System security in depth [JK99, Jan00, BC02, Bor02]. Jansen et al. [JK99, Jan00] have categorized the threats pertinent to Mobile Agent System into four threat categories. These threat categories are a Mobile Agent threatening a Mobile Agent platform (agent versus platform), a Mobile Agent threatening other Mobile Agents (agent versus agent), a mobile agent platform threatening Mobile Agents (platform versus agent), and other threats (other). The general consensus is that the security mechanisms that counter the threats in platform versus agent category are the most immature [BC02, Bor02]. Borselius [Bor02] goes so far to state that

there seems to be no single solution to the security problems introduced by mobile agents unless trusted hardware is introduced,

while claiming that this is likely to prove too expensive. Bierman and Cloete [BC02] conclude that

it also seems that the creation of a trusted execution environment is the one measure that covers all the threats,

when referring to the platform versus agent threat category.

1.1.2 Use of Smart Mobile Devices for Disaster Response

We believe that disaster response can greatly benefit from using smartphones and tablets. There are a number of examples, where researchers propose to use smartphones to facilitate disaster response. Mitra and Poellabauer [MP12] show how smartphones can help paramedics to monitor the heart rate of multiple-patients. For this they use the acceleration sensors present in modern smartphones and the ability of smartphones to form mobile ad-hoc networks. Also Ishigaki et al. [IMIT13] developed a mobile radiation monitoring system and field tested it in Fukushima following the 2011 nuclear power plant incident. The mobile radiation monitoring system used cheap radiation sensors attached to smartphones as measurement equipment. As a final example, Thompson et al. [TWD⁺10] proposed the use of smartphones to detect car accidents and to provide situational awareness for emergency responders.

In addition to a smartphone's multitude of sensors and its ability to form mobile ad-hoc networks that can mitigate communication outages [DHK10], smartphones and tablets have significant computational power, memory and storage. Smartphones have had the technical capability to participate in a Multi-Agent System for years now (see for example Chan et al. [CRP08]). In fact, in 2008 Ughetti et al. [UTG08] demonstrated the use of the Java Agent Development Framework (JADE) Mobile Agent System in the Android smartphone OS.

However, as motivated above, disaster response requires strong security for data protection. When we started our research the platform security mechanisms we use as building blocks for the security solution we introduce in this thesis were still in their infancy on mobile devices. Hence, our security solution is geared towards PC platforms. However, over the years the security mechanisms for

mobile devices matured. The predominant platforms in the smartphone market are Android and Apple's iPhone using ARM System-On-Chips. Therefore, we started to investigate how we can use the security mechanisms available on ARM based systems to port our security solution.

1.2 Goals

Goals

The goals of this thesis are to

1. model the information security threats when using a Mobile Agent System to support disaster response,
2. evaluate the applicability of the threat modeling methodology we use (Data Flow Diagram models with STRIDE-per-interaction) for threat modelling disaster response activities and Mobile Agent Systems,
3. design and implement a security solution that mitigates all threats,
4. evaluate the effectiveness of our security solution (Goal 3) using the threat model developed for Goal 1,
5. make our security solution developed for Goal 3 available in a wide range of deployment scenarios.

1.3 Problem Description

In this thesis we address three distinct problems. First, we create threat models for using a Mobile Agent System for disaster response. Second, we devise a security solution that mitigates the threats we identify. Third, we adapt one of the key building blocks of our security solution for use with smartphones. This key building block is secure storage.

1.3.1 Threat Modeling

O'Neill et al. [OSZW12] have identified the assets in disaster response. The assets are the information exchanged between disaster response personnel to gather information and implement command decisions. Furthermore, O'Neill et al. have documented the need for communication confidentiality, integrity and availability. Given these assets and security goals, as a next step, we need

to identify the threats to these assets. Specifically, we want to investigate the threats to security, when parts of the aforementioned information exchanges are implemented using a Mobile Agent System, instead of by human personnel.

In general, we have to be very careful and meticulous when reasoning about security. We have already seen security goals such as confidentiality, integrity, availability, authenticity, authorization, and non-reputation arising from our two examples in Section 1.1. In any but the most trivial systems there is a multitude of opportunities to break any, if not all, of these goals, unless suitable security mechanisms are applied. Mobile Agent Systems are anything but simple and have an extensive set of security threats that need to be addressed to achieve an application's security goals. History is littered with examples of threats that were realized, because nobody thought about the threat, and hence no mitigation was in place. Therefore, we need a well-structured approach to comprehensively model the threats that arise from applying the Secure Agent Infrastructure to disaster response. The Secure Agent Infrastructure is the Mobile Agent System we use in this thesis.

1.3.2 Threat Mitigation

After identifying the threats during threat modeling, the second challenge we face in this thesis is mitigating these threats. Our threat modeling results are in line with the literature on Mobile Agent security that unanimously identifies a compromised or malicious agent platform as the most pertinent source of threats to a Mobile Agent System. In addition, our threat modeling efforts also identify the need to authenticate the external entities the Secure Agent Infrastructure interacts with and protect any communication with these external entities.

Therefore, we focus our attention on establishing the integrity of the agent platforms of the Secure Agent Infrastructure, thus mitigating the threats arising from a compromised agent platform. Similarly, we need to solve the problem of authenticating the external entities the Secure Agent Infrastructure interacts with.

The problem of establishing the integrity of the Secure Agent Infrastructure stratifies into being able to establish the integrity of the Mobile Agents and the agent platforms they are executed on. We investigate how to establish the integrity of the agent platforms. Establishing the integrity of an agent platform, where an agent platform is a networked computer running a specific set of software components, goes beyond the problem of merely identifying a machine. A single, minute piece of program code can change a reliable agent platform into a malicious host subverting Mobile Agents. Specifically in this thesis we investigate how we can provide a security mechanism that establishes the integrity of an agent platform, while still being usable and widely applicable.

Another aspect we investigate is how to authenticate the identity, and establish the physical presence of, disaster response personnel without access to an online central authority. This information is central to establishing the authority of a person and any agent acting on behalf of her.

1.3.3 Secure Data Storage

The third problem we investigate in this thesis is secure data storage. Secure data storage is a key component of our security solution developed for our second problem, threat mitigation. One of the goals for our security solution is wide applicability to support its adoption (see Goal 5). To this end, we base it on commercial off-the-shelf hardware security mechanisms. Specifically, we use Intel Trusted eXecution Technology (TXT) and a Trusted Platform Module. However, these particular hardware security mechanisms are only available for personal computers and laptops.

Smartphones and tablets are equally interesting agent platforms to connect to the Secure Agent Infrastructure and nowadays they provide similar, but not identical, hardware security mechanisms, such as the ARM TrustZone security extensions. Therefore, as part of Goal 5, we started planning a security solution for Secure Agent Infrastructure agent platforms based on ARM TrustZone. One of the key missing elements for our smartphone security solution was an efficient, flexible, and secure data storage that could be used in conjunction with ARM TrustZone.

The specific problem we tackle here is adding secure storage to ANDIX OS, an existing ARM TrustZone OS. ANDIX OS delegates the onus of secure storage to the Trusted Applications it executes in its Trusted Execution Environment. The Trusted Execution Environment is an execution environment isolated from normal applications, such as a mobile phone application. Our security solution called for a Trusted Application that could securely store and retrieve significant amounts of data efficiently, while maintaining data integrity and confidentiality.

1.4 Contributions

In this section we describe our contributions beyond the state of the art. Specifically, we present three contributions beyond the state of the art corresponding to the three problems identified in the previous section.

1.4.1 Contribution 1 - Threat Modeling

Our first contribution comprises a number of threat models for disaster response processes and disaster response supported by the Secure Agent Infrastructure. We use a systematic, semi-formal methodology to model our threats. This methodology uses Data Flow Diagrams to model systems and the STRIDE-per-interaction threat enumeration technique to generate threats based on the Data Flow Diagram model. STRIDE is an mnemonic for Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege, the six threat categories the STRIDE-per-interaction threat enumeration technique considers. The overall methodology was developed by Microsoft and is part of Microsoft's Security Development Lifecycle.

Contribution 1a - High Level Disaster Response Threat Models

The goal for this contribution is to determine the threats to the high level disaster response activities of gathering information and implementing command and control. O'Neill et al. [OSZW12] have identified the need for confidentiality, integrity and availability for disaster response communication pertaining to those two activities. However, we require concrete threats against disaster response communication, so we can determine adequate mitigation mechanisms. These high level models are the first step towards detailed threat models which we can use for the purpose of finding adequate mitigation mechanisms. Consequently, analysing and refining the high level models led to the discovery of the Distributed Secure Agent Platform Outpost (DSAP Outpost).

From O'Neill et al [OSZW12] we know that all disaster response communication falls into two broad categories. These categories are situational awareness and command and control. Based on this observation and the mode of operation of the Secure Agent Infrastructure we have created an overall model for gathering information (situational awareness) and implementing command and control in disaster response. From this model we derive two high level models for gathering information and command and control. For our high level models we avoid any implementation specific concepts. We use these models to establish the basic threats to our assets, the information exchanged during disaster response.

Our two high level models grant us valuable insights into the threats to situational awareness and command and control we need to consider during disaster response. Specifically, from threat modeling these two activities we have derived 41 threats. These 41 threats naturally cover threats to confidentiality, integrity and availability. One of the advantages of the threat list is that it also makes threats evident that are not immediately obvious from the three security goals of confidentiality, integrity and available. Examples for such threats are a responder repudiating having received a command, or an adversary maliciously changing information to determine the outcome of a command decision. Finally, by refining the high level model for situational awareness we identified the DSAP Outpost component. We develop security mechanisms for the DSAP Outpost as our second contribution (see Section 1.4.2).

Contribution 1b - Distributed Secure Agent Platform Outpost Threat Model

Our primary goal is to model the threats that arise from using the Secure Agent Infrastructure to facilitate disaster response. Therefore, we have refined the situational awareness high level model and included technical aspects of the Secure Agent Infrastructure. Our study of this model revealed a highly repetitive pattern. This pattern consists of a Distributed Secure Agent Platform (DSAP) communicating with the central components of the Secure Agent Infrastructure on the one hand (the Core Secure Agent Infrastructure), and an external entity such as a first responder or an external information system on the other hand.

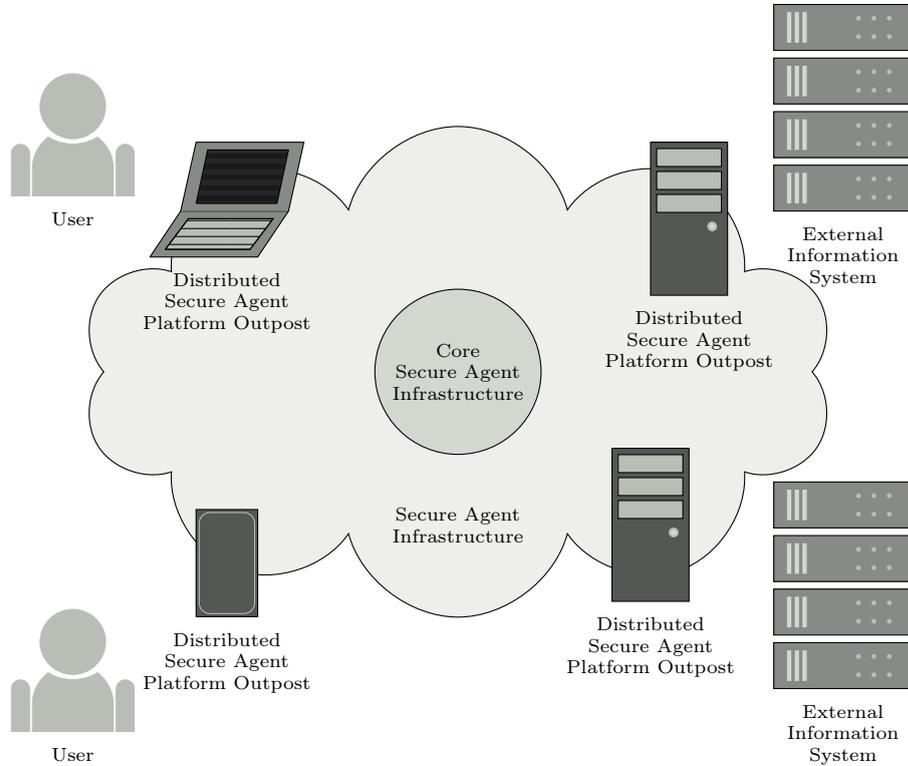


Figure 1.2: The Distributed Secure Agent Platform Outpost

The DSAPs are the agent platforms used by the Secure Agent Infrastructure at the interface points of the Mobile Agent System with external entities. We therefore termed this repetitive pattern the DSAP Outpost. We have depicted this situation in Figure 1.2.

When we analyzed our refined situational awareness model we observed that if we were to mitigate the threats to the DSAP Outposts, then we could mitigate a significant portion of the threats to the overall Secure Agent Infrastructure. Therefore, we decided to concentrate our research efforts on mitigating the threats to the DSAP Outposts. To this end, we have created an in-depth threat model of a DSAP Outpost. From this threat model we derived 54 threats to the DSAP Outpost. We use this threat list to validate the effectiveness of the Trusted Docking Station and the Secure Docking Module, the two security mechanisms we created and introduce in this thesis (see Section 1.4.2).

At the core of the DSAP Outpost lies the DSAPs. The DSAPs are a mobile agent platform. There exists a significant body of literature on the topic of mobile agent platform security. We compared our DSAP Outpost threat list with three works from the mobile agent platform security literature. These three works discuss threats to mobile agent platform security in general. We compared

our results with these works to verify our results. We observed that we identified 20 threats applicable to mobile agent platforms that were not previously recorded in literature. Most of these 20 threats are related to repudiation when a mobile agent platform interacts with external entities.

Contribution 1c - STRIDE Applicability

Case studies on the applicability of STRIDE-per-interaction are still rare. Therefore, we wanted to gauge the applicability of STRIDE-per-interaction for threat modeling high-level disaster response processes. Our results indicate that the Data Flow Diagrams used to model the disaster response activities are well suited to capture the information flows in disaster response. We base this observation on the fact that we consider almost all threats generated for our two high level threat models relevant. Note that our two high level models represent processes not necessarily implemented by an information processing system. Therefore, we needed to slightly adapt our models for the Microsoft Threat Modeling Tool 2016 based threat generation to generate pertinent threats.

Finally, because of the rarity of STRIDE-per-interaction case studies, we also documented our observations on using STRIDE-per-interaction for modeling agent mobility. Agent mobility is the capability of mobile agents to migrate between systems. Migrating an agent involves serializing the agent for transportation, sending it to another platform, deserializing it and then (re)starting its execution. As part of the DSAP Outpost model we have modeled receiving an agent, deserializing it and (re)starting its execution using a Data Flow Diagram. Data Flow Diagrams capture data flows and not complex processes. Nevertheless, when compared to literature, the results for threat modeling agent receipt, deserialization, and execution using a Data Flow Diagram, were consistent.

Threat Modeling Goals

A significant share of Contribution 1 is geared towards Goal 1, the creation of a threat model for a Mobile Agent System supporting disaster response. To achieve this goal we first model high level disaster response activities (Contribution 1a). We use these results as a basis to create a threat model that includes the Secure Agent Infrastructure components. From this model we choose to refine the DSAP Outpost component and create a specific model for it (see Contribution 1b).

We do not believe that we fully met Goal 1. We do not have a comprehensive set of threat models for all disaster response activities including all technical components of the Mobile Agent System. However, our threat modeling approach enabled us to identify the component that we believe to be most difficult to secure and create a detailed threat model for it. We use this model to evaluate the effectiveness of our security solution developed as Contribution 2.

We have achieved Goal 2, the evaluation of using STRIDE-per-interaction for use in disaster response and with Mobile Agent System. To this end, we have recorded our observations on using STRIDE-per-interaction (Contribution

1c). As there are no other STRIDE-per-interaction threat models, neither for disaster response activities, nor for Mobile Agent Systems, we compared our findings with the literature on Mobile Agent System security (Contribution 1b). We found 20 threats not previously documented in the body of literature we used for comparison.

1.4.2 Contribution 2 - The Trusted Docking Station and the Secure Docking Module

Our threat modeling efforts reveal the need to ensure the integrity and authenticity of the DSAP Outposts and the authenticity of the personnel using them. To this end, we design and implement a security solution that establishes the load-time integrity and authenticity of a DSAP Outpost, and the authenticity and presence of its user. This security solution comprises the Trusted Docking Station and the Secure Docking Module.

Contribution 2a - The Trusted Docking Station

Our TDS is an execution environment for the DSAP Outpost. The Trusted Docking Station (TDS) is based on the acTvSM platform [Pir15]. It provides evidence of the load-time integrity of the software it executes and it uses virtualization to isolate different partitions of the system to grant runtime confidentiality, integrity and availability protection. Our contribution here is that we have designed, implemented and evaluated a prototype of the TDS. To implement the TDS we adapted the Secure Agent Infrastructure DSAPs component to run on the acTvSM platform and to use the security features it provides. The Secure Agent Infrastructure is the Mobile Agent System we use. It was developed by researchers at the Slovakian Academy of Sciences [Gat10a, HBG10, GBH10, GBŠH11, BGH⁺11]. For the evaluation we have investigated both the performance and security of the TDS. The security evaluation uses the list of threats generated by our threat modeling efforts (see Section 1.4.1) to measure the effectiveness of our solution.

Contribution 2b - The Secure Docking Module

Our Secure Docking Module (SDM) addresses the two problems of establishing the load-time integrity of the TDS and identifying and establishing the presence of a user while ensuring the state of crisis. Our contributions here include a protocol for local platform configuration verification and the design and implementation of a physical prototype for the SDM. We also integrated the SDM with the TDS. We have evaluated the performance and the security of SDM in conjunction with the TDS. For the security evaluation we use the threats identified by our threat modeling efforts to measure the effectiveness of our solution.

Security Solution Goals

We have met Goal 3 within certain limits. We have designed and implemented a security solution comprising the TDS and the SDM for the DSAP Outpost. Our threat modelling efforts have unearthed 54 threats to a DSAP Outpost. Of these 54 threats, we mitigate all but three, which we consider out of scope (Goal 4). However, the majority of all threats (43/54) pertain to a compromised DSAP Outpost. Here our security solution only gives us load-time integrity and not runtime integrity. Thus our security solution is vulnerable to targeted, online attacks. In a targeted online attack an adversary exploits a runtime vulnerability to gain access to a TDS and the software it is hosting. However, our TDS offers load-time integrity protection on commodity off-the-shelf hardware equipped with Intel's Trusted eXecution Technology (TXT). Furthermore, the SDM enables user authentication and TDS load-time integrity verification with a commercially available Security Controller.

1.4.3 Contribution 3 - Secure Data Storage for TrustZone Systems

The Secure Block Device

Disaster response has strong data confidentiality and authenticity requirements (see Section 1.4.1). To help achieving these security goals, in Section 1.4.2 we propose using the SDM together with the TDS for establishing user authenticity and platform software load-time integrity. The TDS is based on the acTvSM platform and the acTvSM platform heavily relies on Intel's TXT. Intel TXT is a set of platform security extensions. With these security extensions it is possible to gather evidence of a platform's software load-time integrity. Furthermore, the platform can then create attested reports of this evidence. Finally, TXT supports security by isolation and secure key storage.

However, the majority of all smartphones is based on ARM platforms. We believe that with the advent of the ARM TrustZone security extensions for ARM based platforms and the GlobalPlatform standards for Trusted Execution Environments on mobile platforms, our disaster response security goals can also be achieved on ARM phones. We have investigated integrating DSAP Outposts based on smartphones into the Secure Agent Infrastructure, but were quickly limited by a lack of confidential and authentic storage for data at rest.

Storage that is both confidential and authentic is a core component of our acTvSM based TDS. ARM TrustZone based platforms allow for a number of potential mechanisms to provide secure storage. We have contributed the Secure Block Device, a software based solution that uses cryptography to achieve its security goals and relies on TrustZone to protect the cryptographic key while it is in use. The Secure Block Device is secure in the sense that, while the key is not compromised, the Secure Block Device provides data store confidentiality and authenticity. The Secure Block Device is easy to use, as it has a very simple API, modeled after the POSIX file abstraction, and it makes the use of

cryptography as transparent as possible. Furthermore, it is efficient, because it retains random access to the data it protects, allows for different cryptographic protection mechanisms, and implements a cache to alleviate the performance impact of using cryptography. Finally, it is flexible as it allows for different storage back ends and we believe it to be widely applicable beyond the scope we have developed it for. The details of the Secure Block Device in conjunction with a thorough evaluation can be found in Chapter 6.

Security Solution Deployment Goals

We were not able to fully meet Goal 5, the goal to make our security solution available in wide range of deployment scenarios. The main reason is that we have not yet ported our security solution to mobile platforms. However, we took a significant first step in extending ANDIX OS with the Secure Block Device. Furthermore, the TDS is based on commodity off-the-shelf hardware. Similarly, the SDM is implemented on a commercially available Security Controller. This allows for wide deployment on PC hardware.

1.5 Outline

This thesis follows a linear structure. After having motivated and outlined our research problems in this chapter, we introduce the necessary preliminaries for our three research areas in Chapter 2 and the pertinent related work in Chapter 3.

In Chapter 4, we create high level threat models for the disaster response typical processes of information gathering and command implementation. We then proceed to refine the information gathering model to include Secure Agent Infrastructure specific components. After identifying the highly repetitive pattern of a Distributed Secure Agent Platform Outpost (DSAP Outpost) we investigate the threats to a DSAP Outpost in detail. This threat modeling forms the basis for the security evaluation of the Trusted Docking Station (TDS) and the Secure Docking Module (SDM).

In Chapter 5 we introduce our security solution for the DSAP Outpost. Our security solution comprises the TDS and the SDM. We discuss our objectives for the security solution, its mode of operation and how it can be applied to protecting the Secure Agent Infrastructure. We then describe architecture and implementation of the TDS, the resource access protocol for the SDM, and the implementation of the SDM on a Security Controller. We conclude this chapter with a thorough performance and security evaluation. For the security evaluation we use the threats to the DSAP Outpost, which we have identified in Chapter 4, to gauge the effectiveness of our solution.

Chapter 6 is dedicated to the Secure Block Device, a secure, efficient, and flexible data store. We first outline our objectives for the Secure Block Device, and then proceed to threat model the Secure Block Device's use in a Trusted Execution Environment provided by using the ARM TrustZone security exten-

sions. We then proceed to describe the architecture of the Secure Block Device. We conclude our discussion of the Secure Block Device with a performance evaluation on an ARM based development board.

Finally, Chapter 7 summarizes our work, indicates the most important conclusions and identifies potential future work.

2

Preliminaries

2.1 Agents

In Chapter 4 we will investigate threat models for a specific Mobile Agent System, the Secure Agent Infrastructure. Furthermore, in Chapter 5 we introduce the Trusted Docking Station and the Secure Docking Module that together mitigate a number of threats specific to Mobile Agent Systems. Therefore, we introduce the concept of Mobile Agent Systems and the Secure Agent Infrastructure in this chapter. As a basis, here we start with discussing agents and their properties in general.

The goal of our discussion of agents is not to give a definitive definition of agents, as there is none, but to introduce properties associated with agents. In Section 2.6.4, we use these properties to classify the agents of the Secure Agent Infrastructure. We threat model components of the Secure Agent Infrastructure and introduce security mechanisms to mitigate these threats. Therefore the definition of the Secure Agent Infrastructure agents is the definition pertinent to us. Furthermore, in Section 2.4 we discuss and summarize selected literature on Mobile Agent security. More specifically, in Section 2.4.2 we summarize Borselius' investigation of the security implications of these agent properties.

2.1.1 Properties of Agents

We limit our discussion to agents that are software agents as opposed to, for example, human beings and robots. These computer system based agents have their roots in artificial intelligence research, but have since then entered mainstream computer science [WJ95]. There is no universally agreed definition for an agent [WJ95, JSW98]. There are however certain characteristics of agents

widely accepted in literature [Bor02]. These characteristics are documented in the works of Wooldridge and Jennings [WJ95, JSW98] and we discuss them presently. Note however that not all agents exhibit all characteristics. For example, not all agents are mobile.

Wooldridge and Jennings [WJ95] give two notions of agency a weak notion and a strong notion. The weak notion is weak in the sense that it is less restrictive and contentious. The weak notion of agency attributes agents with *autonomy*, *social ability*, *reactivity*, and *pro-activeness*. Wooldridge and Jennings's stronger notion of agency acknowledges the artificial intelligence roots of agents, and it emphasizes that in addition to the above four properties these agents are often conceptualized or implemented using concepts that are more usually applied to humans, such as the mentalistic notions of belief, intention, and obligation [WJ95]. In this thesis we are ultimately interested in securing data carried by Mobile Agent and therefore for us the weak notion of agency is sufficient.

Next to autonomy, social ability, reactivity and pro-activeness, Wooldridge and Jennings also attribute agents with *mobility*, *veracity*, *benevolence*, and *rationality*. Jennings et al. [JSW98] later present a refined list of these characteristics where they define agents to have *situatedness*, *autonomy*, and *flexibility*, where flexibility comprises responsiveness, pro-activeness, and social ability. We will now discuss these attributes as defined by Wooldridge and Jennings [WJ95] and Jennings et al. [JSW98].

Autonomy Autonomy is the ability to act without direct intervention of humans (or other agents). Agents should have control over their own actions and internal state [JSW98]

Situatedness Situatedness stresses that an agent is part of an environment. The agent receives sensory input from this environment and the agent can also change the environment with its actions.

Flexibility Flexibility encompasses responsiveness, pro-activeness, and social ability.

Responsiveness (Reactivity [WJ95]) Jennings et al. define responsiveness as follows: agents should perceive and respond in a timely fashion to changes that occur in the environment [JSW98].

Pro-activeness Agents do not simply act in response to the environment, they should also be able to exhibit goal-directed behaviour by taking the initiative [WJ95].

Social ability Agents should interact, when appropriate, with other artificial agents and humans in order to complete their own problem solving and to help others with their activities [JSW98].

Mobility is the ability to move around an electronic network [Whi94].

Veracity is the assumption that an agent will not knowingly communicate false information [Gal88].

Benevolence is the assumption that agents do not have conflicting goals, and that every agent will therefore always try to do what is asked of it [RG85].

Rationality is (crudely) the assumption that an agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved — at least insofar as its beliefs permit [Gal88].

All of these properties impact agent security and they will be discussed in Section 2.4.

2.2 Multi-Agent Systems

The Secure Agent Infrastructure we investigate in this thesis is a Mobile Agent System which is a form of Multi-Agent System. As with agents there is no unanimous, clear definition what constitutes a Multi-Agent System. According to Jennings et al. [JSW98] the term Multi-Agent System is used to refer to all types of systems composed of multiple (semi-) autonomous components. They define a Multi-Agent System as a system designed and implemented as several interacting agents. Similarly, Durfee and Lesser define a Multi-Agent System as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities and knowledge of each problem solver [DL89]. Also, Jennings et al. [JSW98] point out that Multi-Agent Systems are ideally suited to representing problems that have multiple problem solving methods, multiple perspectives, and/or multiple problem solving entities. They define the characteristics of a Multi-Agent System as follows:

- Each agent has incomplete information, or capabilities for solving the problem, thus each agent has a limited viewpoint;
- There is no global system control;
- Data is decentralized; and
- Computation is asynchronous.

We will revisit these characteristics, when we introduce the Secure Agent Infrastructure in Section 2.6.

2.2.1 Mobile Agent Systems

Mobile Agent Systems are a variant of Multi-Agent Systems that allow agent mobility. In a Mobile Agent System, agents can migrate between platforms. Typically Mobile Agent Systems support agent migration with platform features, as is the case with the Secure Agent Infrastructure, where the Distributed Secure Agent Platform (DSAP) component handles agent migration (see Section 2.6). In a Mobile Agent System the term agent platform refers to the execution environment in which agents operate. In the Secure Agent Infrastructure the DSAP component provides an agent platform.

According to Jansen and Karygiannis [JK99] the platform where an agent originates is called the *home* platform and normally is the most trusted environment for a Mobile Agent.

2.3 Mobile Agents Properties

White's definition that agent mobility is the ability to move around an electronic network [Whi94] is generic. We want to discuss different forms of agent mobility, based on what is actually migrated, and how many platforms are visited.

Mobile Agents are a form of mobile code. Carzaniga et al. [CPV07] consider two forms of code mobility for executing units. Executing unit is the general term for an entity that can execute code, for example a thread.

Strong code mobility allows executing units to move their code and execution state to a different platform.

Weak code mobility is the ability to migrate code to a different platform and automatically execute it, but no execution state is transferred.

So a Mobile Agent exhibits strong code mobility, if it is capable of suspending its execution state on one platform and move to another platform, where it resumes executing [JK99]. The Secure Agent Infrastructure Mobile Agents we consider in this thesis only support weak code mobility. These agents do not migrate their execution state (Section 2.6.4).

Mobile Agents can also be categorized by how many platforms they visit. Mobile Agents that can only migrate from their home platform to a single other agent platform are called single-hop agents. Multi-hop Mobile Agents can migrate to more than one platform. The Secure Agent Infrastructure Mobile Agents are single-hop agents.

2.4 Mobile Agent Security

One of the contributions of this thesis is a discussion of threat models for a specific Mobile Agent System, the Secure Agent Infrastructure (see Section 2.6), in Chapter 4. We wanted to ground our discussions of the threats to the Secure Agent Infrastructure in literature, therefore we selected three independent discussions of Mobile Agent security as reference base line. These three works are "Mobile Agent Security" by Jansen and Karygiannis [JK99], "Mobile Agent Security" by Borselius [Bor02], and "Classification of Malicious Host Threats in Mobile Agent Computing" by Bierman and Cloete [BC02]. We have selected these three sources, because they comprehensively discuss Mobile Agent security in general and not specific security aspects of particular Mobile Agent System.

Roth opines that the lack of security is one of the primary reasons why mobile agents have yet to be broadly adopted [Rot04]. In fact, according to Roth, almost thirty years after their inception, the only widespread incarnation

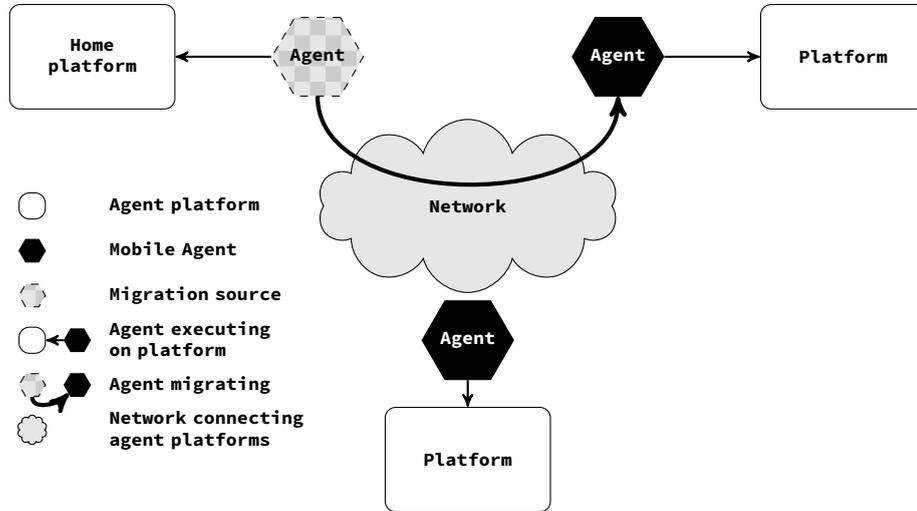


Figure 2.1: The Mobile Agent System model used by Jansen and Karygiannis [JK99] to discuss the security issues of Mobile Agent Systems. This model depicts an agent migrating from its home platform to another platform over a network that connects the platforms of the modeled Mobile Agent System. This specific version of the model is adopted from [JK99].

of mobile software agents is malware. Borselius similarly points out the need for securing agent systems for them to become adopted [Bor02].

2.4.1 Jansen and Karygiannis

A number of researchers have investigated Mobile Agent security in detail. Jansen and Karygiannis [JK99] and later Jansen alone [Jan00] have compiled a thorough discussion of Mobile Agent security. They have based their discussion on Mobile Agent System security on the model depicted in Figure 2.1. Their model considers a Mobile Agent migrating from its home platform to another platform through a network. In addition, the model considers that the previously migrated Mobile Agent is then executed by the receiving platform. We observe that the model does not show multiple agents interacting on the same platform. However, this is one of the threat categories considered by Jansen and Karygiannis.

Jansen and Karygiannis grouped their findings into four threat categories. Here we summarize their findings. We use both their joint work [JK99] and Jansen's solo paper [Jan00] as sources.

Agent-to-Platform This threat category comprises all threats arising from a Mobile Agent attacking the agent platform, either by exploiting a security vulnerability or by launching attacks on the agent platform, such as a Denial-of-Service attack. According to Jansen [Jan00] an incoming agent

has two lines of attack. These are unauthorized information access to platform information and the incoming agent using its authorized access in an unexpected and disruptive fashion. Unauthorized access to information may occur due to lack of adequate access control or weak authentication methods. It can enable a Mobile Agent to obtain confidential information, but also, with sufficient access to platform core services, to shut down or terminate the platform. Jansen and Karygiannis [JK99] note that Mobile Agents can always try for a Denial-of-Service attack by exhausting a platform's computational resources through excessively requesting platform services.

Agent-to-Agent A malicious agent has a number of options to attack another agent. Jansen and Karygiannis identify falsifying transactions, eavesdropping on communications, or interfering with another Mobile Agent's activity as threat categories. They list a Mobile Agent responding with tampered information, denying an answer to a request, masquerading as another agent, eavesdropping and modifying another Mobile Agent as potential threats.

Other-to-Agent Platform In this category Jansen and Karygiannis discuss external entities attacking a Mobile Agent System. They consider an external attacker disrupting, harming, or subverting the agent system. As potential methods they list attacking inter-agent and inter-platform communication, and they consider eavesdropping and tampering through masquerading, unauthorized access to the underlying platform, and Denial-of-Service as threats.

Platform-to-Agent A mobile agent is almost at the complete mercy of the mobile agent platform. Jansen states that a receiving agent platform can easily isolate and capture an agent and may attack it by extracting information, corrupting or modifying its code or state, denying requested services, or simply reinitializing or terminating it completely [Jan00]. The receiving platform can facilitate complete analysis and reverse engineering the Mobile Agent, and modify the agent to radically change its behaviour, for example turning it malicious. The receiving platform can also feed the agent incorrect information, withhold information, or change the agent internal representation of the information to corrupt it.

2.4.2 Borselius

Like Jansen and Karygiannis, Borselius has also discussed Mobile Agent security [Bor02]. However, instead of using a Mobile Agent platform migration model, Borselius bases his discussion of Mobile Agent security on the agent characteristics defined by Wooldridge and Jennings [WJ95] and Jennings et al. [JSW98] (see Section 2.1). We will now give a précis of his discussion of the impact of Mobile Agent characteristics on security.

Agent execution Under this heading Borselius discusses two aspects. First he raises the question of where access control decisions occur. Can the Mobile Agent decide if an incoming request is authentic and authorized, or can the Mobile Agent rely on the platform for this decision?

Second Borselius points out that the environment might need protection from the agent it interacts with. He cites the agent launching a Denial-of-Service attack against the agent platform as a potential problem.

Situatedness An agent is embedded in an environment with which it interacts. Borselius points out that the nature of the environment has an impact on security. If the environment is, for example, an open system like the Internet, a Mobile Agent might be well advised to check the authenticity of received information. Borselius also notes, that in limited environments no specific security measures might be necessary.

Autonomy Borselius documents that autonomy introduces serious security concerns. As example, he cites a Mobile Agent with the authority to buy or sell items. It should not possible for another party to coerce this agent into buying and selling. He also points out that a Mobile Agent should not be able to make commitments it cannot fulfill.

Communication Agents can exhibit social behavior and interact with other agents and humans. Borselius notes that this can have interesting implications for security. Borselius documents the need for confidentiality, data integrity, authentication, availability and non-repudiation in all inter-agent and human communication.

Mobility Borselius states the opinion that the ability of Mobile Agents to migrate to other platforms raises a number of security concerns. As Jansen and Karygiannis [JK99] before him he notes that Mobile Agents need protection from other agents and from the agent platform on which they execute. He also indicates the need to protect the agent platform against Mobile Agents and other parties that interact with the agent platform. He opines that the problems associated with the protection of hosts from malicious code are quite well understood.

He summarizes the attacks a malicious agent platform can perform on a Mobile Agent as follows: observation of code, data and flow control, manipulation of code, data and flow control – including the Mobile Agent’s itinerary, incorrect execution of code – including re-execution, denial of execution – in part or whole, masquerading as a different agent platform, eavesdropping on a Mobile Agent communications, manipulation of agent communications, and false system call return values.

Rationality, veracity, and benevolence Here, Borselius comments that on first glance these properties seem to be security relevant, but on closer inspection they are too abstract to serve as practical security concerns. He extracts the meaning of these three properties as: “Agents are well

behaved and will never act in a malicious manner.” He notes that a Mobile Agent System where this statement holds would be valuable, and points out numerous measures that can help to achieve this goal, but overall considers achieving this goal unlikely, unless the Mobile Agent System is under very strict control by a single authority.

Identification and authentication Borselius documents the importance of identification for authentication. He also notes that if identities are not permanent, security-related decisions cannot be based on a Mobile Agent’s identity. Finally, he points out that Mobile Agents might be used to protect the anonymity of their owners.

Authorization and delegation Here, Borselius again indicates that Mobile Agent act on the behalf of other entities and need to be granted access to information and resources. Borselius opines that existing mechanisms such as a Public Key Infrastructure can support this delegation.

2.4.3 Bierman and Cloete

Bierman and Cloete [BC02] restrict their discussion of Mobile Agent security to threats that arise from a malicious agent platform. They define four threat classes. These threat classes are integrity attacks, availability refusal, confidentiality attacks, and authentication risks. For each class they define up to three subclasses. We will concisely discuss these here.

Integrity attacks In this class Bierman and Cloete discuss both accidental and malicious tampering with an agent’s code, state, or data.

Integrity interference Bierman and Cloete define integrity interference as occurring when the executing host interferes with a Mobile Agent’s execution mission, but does not alter any information related to the agent. As an example they cite incorrect transmission of a Mobile Agent.

Information modification They define information modification to contain altering, corrupting, manipulating, deleting, misinterpreting, or incorrect execution of a Mobile Agent’s code, data, control flow, or status. They also include a host interfering with inter-agent communication.

Availability refusal Bierman and Cloete define availability refusal as preventing an authorized agent from accessing objects or resources to which it should have legitimate access. They have divided this threat class into Denial-of-Service, delay of service, and transmission refusal.

Denial-of-Service occurs when the agent platform executing the Mobile Agent denies requested resources, or when the agent platform bombards the Mobile Agent with so much information that the agent finds it impossible to complete its goals.

Delay of service occurs when the agent platform lets the Mobile Agent wait for a requested service. They note that the delay can have a negative effect on the mission of the agent.

Transmission refusal happens when an agent platform refuses to transmit the agent to the next target in its itinerary.

Confidentiality attacks In this class Bierman and Cloete discuss illegal access, potentially in conjunction with illegal removal, to a Mobile Agent's assets. They consider three subclasses, eavesdropping, theft, and reverse engineering.

Eavesdropping occurs when the agent platform spies on the Mobile Agent either by inspecting the Mobile Agent's data directly, or by listening in on inter-agent communication.

Theft takes place when the agent platform not just eavesdrops on information, but also removes the information. Bierman and Cloete point out that the stolen information could be the Mobile Agent itself.

Reverse engineering happens when a malicious host analyses a Mobile Agent in order to manipulate future or existing agents. Bierman and Cloete note that this allows the perpetrator to construct similar Mobile Agents.

Authentication risks In their final class Bierman and Cloete document the need for a Mobile Agent to correctly identify and authenticate an agent platform. According to them authentication risks fall into two subclasses. These are masquerading and cloning.

Masquerading is when an agent platform spoofs (see Section 2.8.2) another agent platform.

Cloning occurs when the agent platform clones a Mobile Agent. Bierman and Cloete point out that when a Mobile Agent carries authentication data this will lead to (unspecified) authentication problems.

2.4.4 Discussion

Jansen and Karygiannis, Borselius, and Bierman and Cloete all discuss Mobile Agent security in general, but using different approaches. Here we discuss how the different approaches relate to each other and to our approach. We do not go into detail on the results of our approach versus the other approaches, as this is part of Chapter 4.

Jansen and Karygiannis base their discussion of Mobile Agent security on the model depicted in Figure 2.1. Using this model they have listed numerous threats which they have classified into the agent-to-platform, agent-to-agent, other-to-agent, and platform-to-agent categories depending on which entity attacks and who the victim is. Based on the nature of the threats they list, we believe they

considered threats to authentication, authorization, non-repudiation, integrity in general, confidentiality, and availability.

Jansen and Karygiannis' approach to analysing Mobile Agent security is similar to our approach for modeling the threats. We both use a model, but whereas Jansen and Karygiannis' model is generic, ours is adapted to the Secure Agent Infrastructure. Furthermore, we base our threat modeling on STRIDE (see Section 2.8) using the Microsoft Threat Modeling Tool 2016 to create the model and a list of threats. We note however, that STRIDE generates threats in the Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege threat categories, which are the counterparts to the security violations considered by Jansen and Karygiannis.

Bierman and Cloete restrict their discussion of Mobile Agent security to the threats arising from a malicious agent platform, as they argue that these threats are the most difficult to counter. They do not declare if they use a model to elicit their threats. However, Bierman and Cloete state that they have based their analysis on the five network security concerns of integrity, availability, confidentiality, authentication, and non-repudiation. We note however that their threat classification does not contain any threats to repudiation, nor do they consider authorization. We also note that, again, the security concerns used by Bierman and Cloete are the counterparts to the STRIDE threat categories.

Borselius' approach differs significantly from ours and also from Jansen and Karygiannis' and Bierman and Cloete's approach. Borselius bases his discussion of Mobile Agent security on the properties of agents as listed in Section 2.1. This provides interesting insights into the nature of threats to Multi-Agent System and why these threats arise. The threats Borselius lists are the broadest in scope, but also the most generic. For example, he is the only one to mention the need for non-repudiation in all agent communication, but he does not detail why beyond this statement. Finally, Borselius' discussion has a strong focus on authentication and authorization. We believe this stems from his property based approach that makes it obvious that agents act on behalf of other entities and that their autonomous decisions can have significant impact on the physical world.

2.5 Jini

Jini is a distributed system toolkit written in the Java™ programming language. In the Secure Agent Infrastructure (see Section 2.6) Jini is used to implement the Distributed Secure Agent Platform (DSAP) component (see Section 2.6.3). The DSAP is central to the Secure Agent Infrastructure; therefore we discuss it here. Jini has many interesting properties, here we concentrate on those that are pertinent to the DSAP. Jini was initially developed by Sun Microsystems and it was later released as open source under the Apache license and is now maintained by the Apache foundation under the name Apache River¹.

¹<https://river.apache.org/>

2.5.1 Jini Services

According to the documentation of the Apache River project [Apa16a] River (Jini) is a toolkit to build distributed systems with. The core concept of Jini is the service. Any entity that can provide a service using Jini technology can be a Jini service. This definition includes computer systems, both in software and hardware, and even the environment or humans through the use of such a computer system. Jini specifies the manner a Jini service can be discovered by service consumers (clients). It also specifies the technical details how a service presents its interface to a client, and how a client can then communicate with the service. Other than that Jini imposes no limits on the nature of the service itself.

2.5.2 Communication

Jini uses Remote Procedure Calls as communication mechanism over IP based networks. Initially Jini was designed to rely only on Java Remote Method Invocation, but Jini now uses the JERI protocol [Apa16a]. Plain-SSL, HTTP, HTTPS, and Kerberos-TCP implementations of the JERI protocol exist, but for compatibility with Remote Method Invocation Jini still supports the JRMP protocol.

The DSAP uses Remote Method Invocation for communication. According to the Apache River project [Apa16b], Java Remote Method Invocation is a Java programming language-enabled extension to traditional remote procedure call mechanisms that allows objects, including their code, to traverse the network boundary.

2.5.3 Service Discovery

Jini's service discovery is based on *lookup services*. The lookup service is a central bootstrapping mechanism for the system and provides the major point of contact between the system and users of the system [Apa16b]. Service providers register with the lookup service to advertise their services and clients query lookup services to find specific services. Technically, the lookup service maps descriptions and the interface of a service to concrete service implementations.

A new service registers with a lookup service using the discovery and join protocol pair [Apa16b]. The discovery and join protocols are executed when a device providing at least one Jini service plugs into the system. The discovery protocol helps the joining service to find a lookup service and, once a suitable lookup service has been found, the device uses the join protocol to register with this lookup service.

Figure 2.2 illustrates the lookup service discovery process. When a new Jini service plugs into a specific Jini system it can use either a local network multicast to find a lookup service, or directly connect to a known, and potentially remote, service. Jini relies on UDP for multicast and TCP for direct connections to a lookup service. Jini also allows to defer lookup service discovery to a third party.

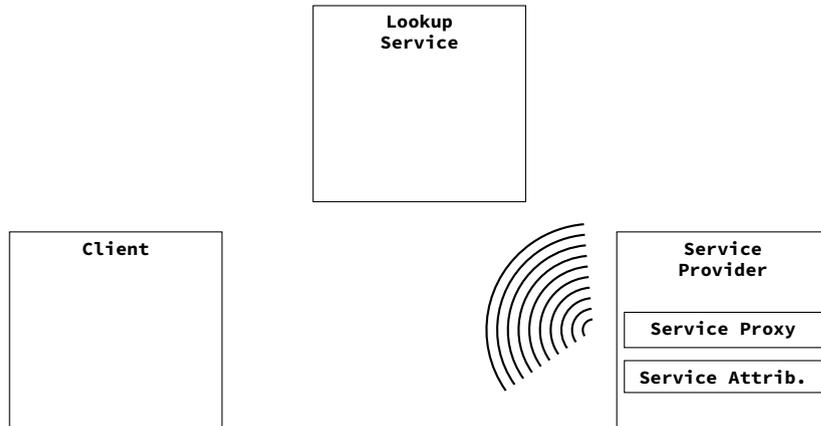


Figure 2.2: Jini lookup service discovery according to the Jini specification [Apa16b]. Here a service provider is looking for a *lookup service* to advertise its service.

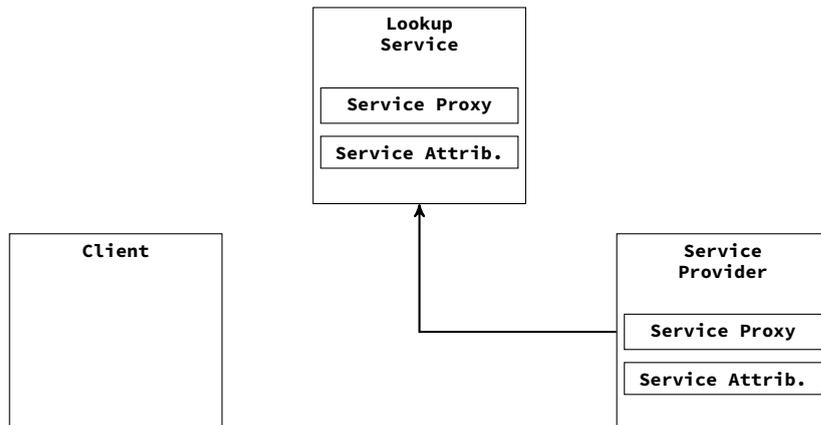


Figure 2.3: A Jini service registering with a *lookup service* using the join protocol according to the Jini specification [Apa16b].

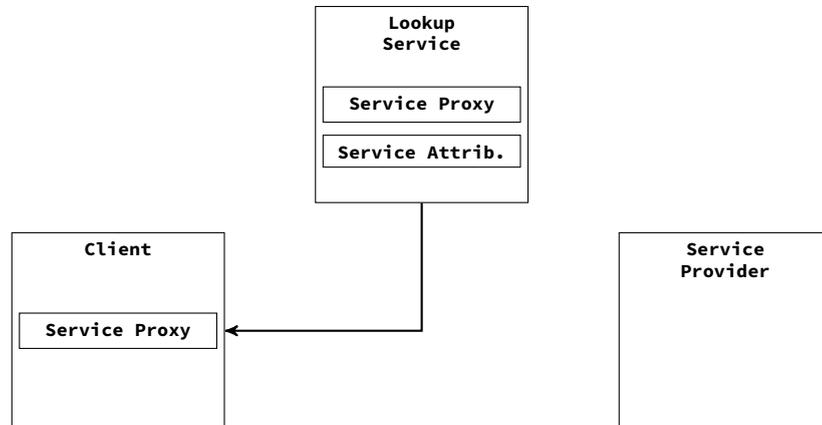


Figure 2.4: A Jini client looks up a specific service using the service proxy type and attributes. A copy of the service proxy is delivered to the client to enable communication between the client and the service provider [Apa16b].

The join protocol outlined in Figure 2.3 specifies how a service provider registers its service with a lookup service. When a service provider joins it transmits a service proxy and the service's attributes. The service proxy is a Java object adhering to the Java interface describing the service and it is transmitted using Java Remote Method Invocation.

Jini clients look for specific services by querying a lookup service. A query consists of two parts. First it specifies a Java interface the service must fulfill. Second the query contains a set of attributes the service must have. The lookup service searches its list of joined service providers. For a service provider to match the query, the service proxy object has to implement the requested Java interface and it has to have the right attributes. If an adequate service is registered with the lookup service, the lookup service transmits the service proxy object to the client (see Figure 2.4).

As depicted in Figure 2.5 Jini clients use the service proxies to interact with the actual service implementation by the service provider. Jini makes no stipulations about how the service proxy communicates with the service provider. However Jini provides facilities for allowing the service proxy and the service provider to communicate. One such facility are remote event listeners. Remote event listeners provide a way to synchronously notify the service implementation from the service proxy and vice versa. The DSAP uses this facility to implement agent communication.

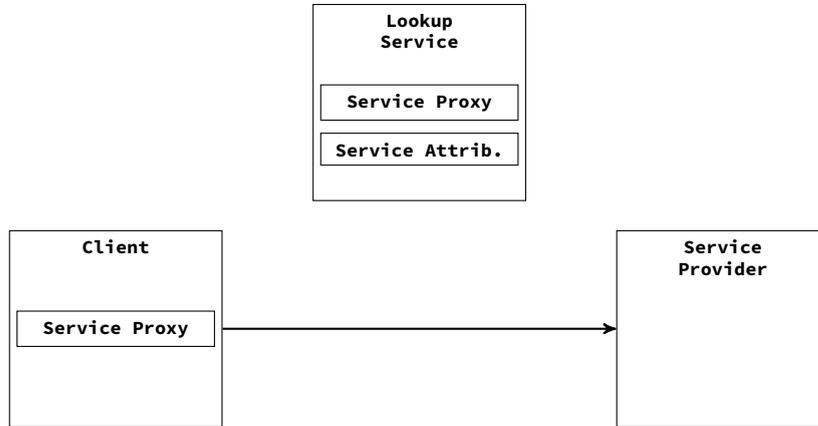


Figure 2.5: A Jini client interacts with a service using the service proxy. This figure is adapted from [Apa16b].

2.6 Secure Agent Infrastructure

2.6.1 Overview

The Secure Agent Infrastructure is a Mobile Agent System specifically developed for crises management in general and disaster response in particular by researchers at the Slovakian Academy of Sciences [Gat10a, HBG10, GBH10, GBŠH11, BGH⁺11]. Emil Gatial's Dissertation [Gat10a] contains a thorough discussion of the Secure Agent Infrastructure and is our main source for this section. The Secure Agent Infrastructure is the Mobile Agent System we create threat models for in Chapter 4, and for which we have designed and implemented the Trusted Docking Station and Secure Docking Module security solutions introduced in Chapter 5. In this section we will concentrate on describing those aspects of the Secure Agent Infrastructure that are pertinent to this thesis.

The goal of the Secure Agent Infrastructure is to support crises management personnel during a crisis. The Secure Agent Infrastructure is written in the Java programming language and its components are executed in a number of Java virtual machines. The architecture of the Secure Agent Infrastructure is divided in 5 principle components, the Process Management Subsystem, the Distributed Secure Agent Platform (DSAP), the Agent Repository, a Public Key Infrastructure repository and the Resource Lookup System. Of these 5 principle components the DSAP is the component most relevant to our work. Therefore we only outline the roles of the other components here, and explain how they interact through an example detailed in Section 2.6.2. We will then discuss the DSAP in more detail in Section 2.6.3.

Process Management Subsystem The Process Management Subsystem is the central control component of the Secure Agent Infrastructure Mobile Agent System. It governs crises management support by implementing

predefined workflows. These predefined workflows are stored in process templates. A process template specifies a list of activities that constitute the workflow. Process Management Subsystem users can instruct the Process Management Subsystem to start a workflow. The Process Management Subsystem then instantiates the process template for this workflow into a process object and executes the process object. The process object then performs all its activities and tracks the state of the activities.

For its role in crisis management and disaster response the Process Management Subsystem is a-priori set up with process templates and activities. During a crisis the Process Management Subsystem instantiates process templates and executes their activities. The Process Management Subsystem uses Mobile Agents to implement the activities. An example on how the Process Management Subsystem works to mitigate a crisis is detailed in Section 2.6.2.

Distributed Secure Agent Platform The DSAP is an agent platform implemented in Java as a Jini service. The Secure Agent Infrastructure comprises an arbitrary number of DSAPs. When a Process Management Subsystem process implements an activity it deploys one or more Mobile Agents to one or more DSAPs. The DSAPs execute these agents and provide communication facilities so that the Mobile Agents can send data back to the Process Management Subsystem process that spawned them. We discuss the DSAP in detail in Section 2.6.3.

Agent Repository The Agent Repository stores the Secure Agent Infrastructure's agents. Specifically, the Agent Repository stores the code of agents, either as plain class files, or in Jar files. Furthermore, it stores a certificate for each agent which is used to authenticate the Mobile Agent code.

Public Key Infrastructure repository The Public Key Infrastructure repository is a central storage for all agent platform certificates.

Resource Lookup System Situational awareness is a key success factor for crises management. The quality of the available intelligence directly influences the quality of decision making and thus crisis mitigation. Therefore, information retrieval is a central task of the Secure Agent Infrastructure. In a crisis the intelligence needed for situational awareness is often divided into many different domains and different stakeholders in these domains. Therefore, the Secure Agent Infrastructure needs a mechanism to find out which entities hold relevant information for a specific process. This is the task of the Resource Lookup System. The Resource Lookup System maps connected entities to the information they can provide.

In addition to these core components the Secure Agent Infrastructure also heavily relies on two agents.

Information Delivery Agent This purpose of this agent is to *retrieve* information from legacy systems that are not directly able to integrate into the

Mobile Agent System and deliver it to the Process Management Subsystem. The Information Delivery Agent uses exchangeable data connector subcomponents to interact with, and retrieve data from, any number of legacy systems. For example, one of the data connectors is a configurable relational database connector where the configuration specifies which data to retrieve and how.

User Communication Agent The User Communication Agent is designed for interacting with humans. The User Communication Agent can use different messaging mechanisms such as for example the Silentel² push-to-talk communication solution for first responders [BGH⁺11]. A User Communication Agent can directly send messages to a human user using the messaging mechanism, and, additionally, it can use a configurable form based interface to request structured information from the user.

2.6.2 Example Use Case

The authors of the Secure Agent Infrastructure like to illustrate how the different components of the Secure Agent Infrastructure interact to help mitigate a crisis through describing the following use case [Gat10a, HBG10, GBH10]. The scenario for the use case is a swine flu epidemic in a single country. The number of infected and sick people is steadily rising and the country's responsible government agency decides to up their warning level and initiate crisis management. This is where the Secure Agent Infrastructure comes in.

Figure 2.6 details the individual steps of a single process of the swine flu epidemic mitigation strategy and how the Secure Agent Infrastructure components and the crisis mitigation personnel work together in this process. Here follows a description of the individual steps.

1. The national health agency's Chief Officer responsible for handling an epidemic initiates the crisis mitigation process. For this she uses a User Communication Agent to inform the Process Management Subsystem about which kind of crisis it is, and how severe the crisis has become.
2. Based on the information received in the prior step the Process Management Subsystem compiles a list of mitigation plans and qualified personnel from its database. It sends this list to the Chief Officer, again using the User Communication Agent.
3. The Chief Officer peruses the list and selects the most appropriate mitigation plan and a responsible Crisis Officer to supervise the process. One process of the mitigation plan is to ensure that enough vaccine is available in the regional branches of the national health agency. This is the process we continue to follow here.

²<http://www.silentel.com/en/>

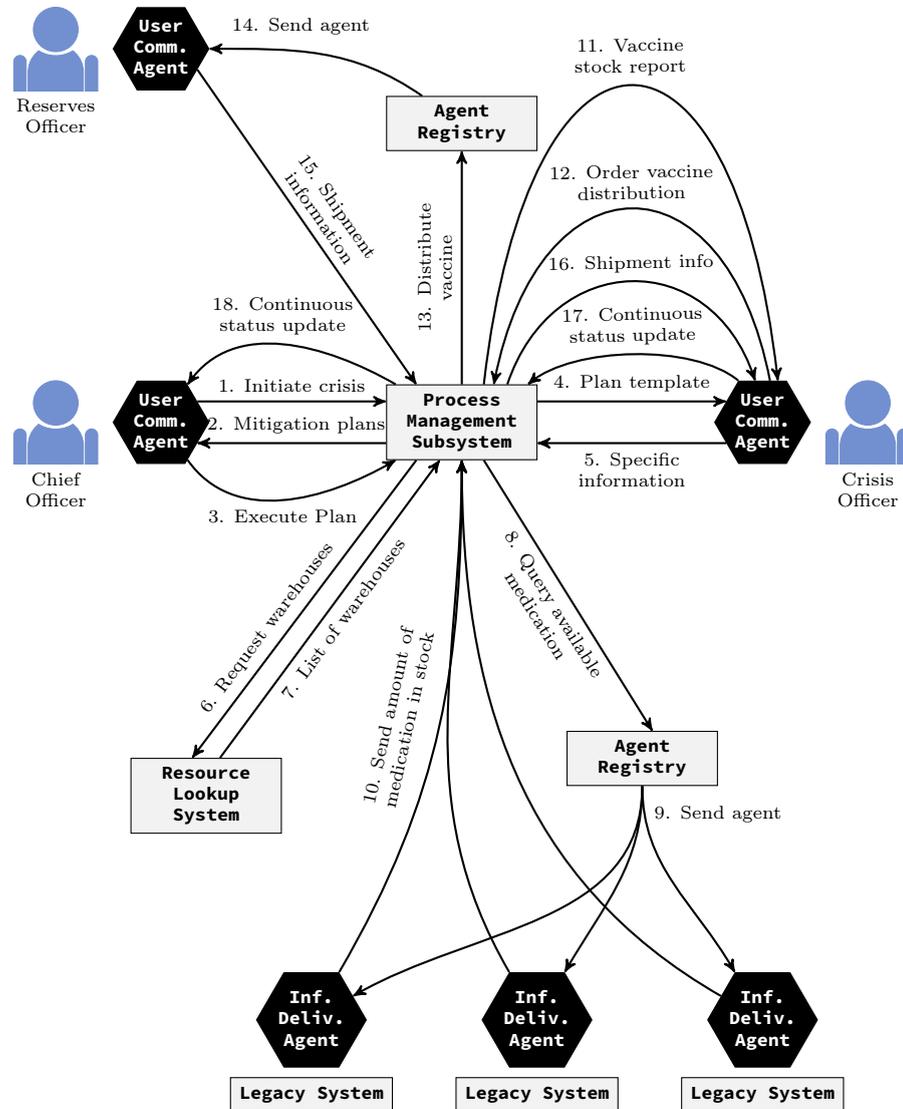


Figure 2.6: This crisis mitigation workflow example illustrates how the components of the Secure Agent Infrastructure interact with crisis management personnel to help mitigating a swine flu epidemic. This example follows a single process that governs the distribution of more vaccine to the regional offices of the national health agency. In this example Mobile Agents communicate with users and retrieve information from otherwise incompatible legacy systems. The example is adopted from Emil Gatia's dissertation [Gat10a].

4. With the overall strategy selected, the Process Management Subsystem contacts the responsible Crisis Officer, again using a User Communication Agent, to fill in the details. Specifically, here the Crisis Officer is asked to specify which vaccine to stock up on and how many people per 1000 must be vaccinated with this particular vaccine.
5. The Crisis Officer accepts to supervise the mitigation process and specifies the requested information.
6. Given this new information the Process Management Subsystem needs to determine how much vaccine is available in the individual regional branches of the health agency. For this the Process Management Subsystem contacts the Resource Lookup System and asks for a list of regional health agencies that can administer this vaccine.
7. The Resource Lookup System sends back a list of the appropriate regional health agencies to the Process Management Subsystem.
8. The Process Management Subsystem formulates the correct configuration for the database connector component of the Information Delivery Agents to query this information from the regional health agency information systems. It then contacts the Agent Repository to configure and release a number of Information Delivery Agents to perform this task.
9. The Agent Repository releases the Information Delivery Agents to query the regional health agencies information systems. As the regional health agencies use legacy systems not directly integrated with the Secure Agent Infrastructure, the agents migrate to a connector (a DSAP) in the regional health agencies computer systems.
10. Once the Information Delivery Agents have migrated to the DSAPs stationed in the regional health agencies, the agents query the amount of available vaccine and send this information back to the Process Management Subsystem.
11. The Process Management Subsystem waits a specified amount of time for all Information Delivery Agents to report back. After the deadline the Process Management Subsystem collates a list of vaccine stocks and sends it to the responsible Crisis Officer.
12. The Crisis Officer peruses the list and after careful consideration orders more vaccine to be distributed to the regional health agencies. Through a User Communication Agent the officer informs the Process Management Subsystem.
13. The Process Management Subsystem now knows how much vaccination per 1000 people is necessary, how many people a given regional agency is responsible for and how much vaccine each agency has in stock. Based on this information the Process Management Subsystem computes how

much vaccine to order and where to distribute it. The order request and the distribution scheme are sent to the Agent Repository together with a request to instantiate a User Communication Agent and send it to the national materials reserve that stocks more vaccine.

14. The Agent Repository instantiates a User Communication Agent configures it with the vaccine order request and distribution scheme and sends it to a DSAP in the national materials reserve to contact the appropriate Reserves Officer.
15. The Reserves Officer processes the order and through the User Communication Agent sends back status information on the shipments to the Process Management Subsystem.
16. The Process Management Subsystem informs the Crisis Officer about the status of the vaccine deliveries.
17. The Crisis Officer maintains the process status. Through the User Communication Agent the process status is relayed to the Process Management Subsystem.
18. The Process Management Subsystem keeps the Chief Officer up-to-date on the process status.

2.6.3 Distributed Secure Agent Platform

The DSAP software component is a Jini service (see Section 2.5.1) written in the Java programming language. A DSAP is capable of receiving a Mobile Agent and instantiating and executing the Mobile Agent. The DSAP also helps Mobile Agents to communicate with their Process Management Subsystem process by providing the necessary communication primitives. The individual DSAP instances are homogeneous; they use the same program code. Thus the DSAP is the substrate of the Secure Agent Infrastructure Mobile Agent System.

The DSAPs are built on Jini (see Section 2.5). Specifically, the individual DSAP instances are implemented as Jini services and they use Jini technology for service registration and communication. Figure 2.7 depicts how the DSAP Service use the Jini service discovery and join protocols (see Section 2.5.3) to register with a service registry (a Jini lookup service) and advertise their existence to potential clients, such as the Process Management Subsystem. When a Process Management Subsystem process wants to send an agent to a specific DSAP it can use the lookup service to find it. Then it fetches a service proxy object from the lookup service and uses that to send the Mobile Agent from the Agent Repository to the DSAP. Furthermore the client can send messages to and receive messages from a Mobile Agent using the service proxy object.

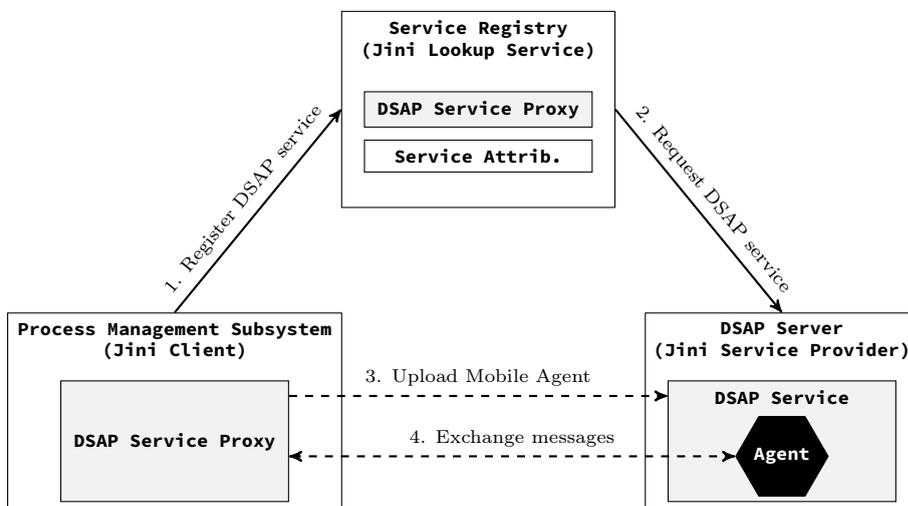


Figure 2.7: The Distributed Secure Agent Platform component of the Secure Agent Infrastructure is a Jini service handling Mobile Agent transmission, client-agent communication and Mobile Agent execution [GBH10]. A newly instantiated Distributed Secure Agent Platform Service (DSAP Service) first uses Jini’s discovery and join protocols to register with a Jini lookup service. Then clients, such as the Process Management Subsystem, can find the DSAP Service and request a service proxy to send agents to it and allow agents to communicate with the client through the service proxy and vice versa.

2.6.4 Properties of Secure Agent Infrastructure Agents

We have introduced properties associated with agency in Section 2.1. Gatial documents that of these agent properties the agents in use by the Secure Agent Infrastructure exhibit the following properties [Gat10a]:

Code mobility The agents exhibit weak code mobility and only support single-hop mobility (see Section 2.3). The agents are sent from the Agent Repository to a target DSAP where they are executed. However they do not migrate their execution state and they can travel to no additional hosts. Data is directly sent back to the Process Management Subsystem. Gatial points out that single-hop, weak code mobility was sufficient for the crises management use cases they implemented.

Autonomy The agents autonomously gather data and sent it to Process Management Subsystem process that caused the agents to be spawned.

Reactivity In some cases the agents perceive the context in which they operate and react accordingly. For example, agents can monitor the availability of resources and notify the entity requesting this information.

We observe that Gatial does not attribute the agents with *situatedness*. We believe this is due to the fact, that although the agents are part of an environment and interact with it, only the User Communication Agents change the environment, when interacting with humans. However, here the more or less just relay information to and from the Process Management Subsystem process that spawned them. A significant part of the “intelligence” of the Secure Agent Infrastructure is centralized in the Process Management Subsystem.

This lack of decentralized “intelligence” violates one of the properties of a Multi-Agent System we introduced in Section 2.2. Specifically, the property that states that there is no global system control. The Secure Agent Infrastructure exhibits the other three properties, namely that each agent has only incomplete information, that the data is decentralized and that the computation is asynchronous. However, if this makes the Secure Agent Infrastructure a Multi-Agent System in the classical sense is open for debate.

Gatial also makes no statement about *responsiveness*, *pro-activeness*, and *social ability*. We do believe that the agents are capable of perceiving and reacting in a timely fashion, therefore we believe the agents to be *responsive*. However, they exhibit little *pro-activeness* and they have no *social ability*.

Concerning *Veracity*, *Benevolence*, and *Rationality* we lean towards Borselius’ viewpoint that the gist of these properties is that “agents are well behaved and will never act in a malicious manner [Bor02].” To this end agents are certified by a trusted third party, before being introduced to the Secure Agent Infrastructure. See Section 2.6.7 for details.

2.6.5 Security Architecture of the Secure Agent Infrastructure

From the beginning the Secure Agent Infrastructure was designed to use an external security module, the Secure Docking Module (SDM) [SEC07]. We expanded the concept of the SDM into two parts: the Trusted Docking Station (TDS) and the SDM [HT09, ŠBH⁺09, HDFL09, KLL⁺09], and both concepts became an integral part of the Secure Agent Infrastructure. The design, implementation and evaluation of the SDM and the TDS are contributions of this thesis and will be detailed in Section 5. The objectives of the SDM/TDS combination include authenticating a DSAP and establishing its load-time integrity (see Section 2.15.3) in order to bind the use of cryptographic keys to a specific platform security policy.

The authors of the Secure Agent Infrastructure created security mechanisms that use the SDM and TDS combination. We consider these security mechanisms created by the Secure Agent Infrastructure author's external to this thesis and as prior art. These mechanisms are detailed in [Gat10a], and a brief description follows here. We supplemented the information in Gatial's dissertation by studying the source code of the DSAP [Gat10b]. Note that not all these mechanisms actually use the SDM and TDS combination.

2.6.6 Resilience against Temporary Communication Disruptions

One of the reasons the authors of the Secure Agent Infrastructure decided to use a Mobile Agent System for disaster response was its inherent resilience against temporary communication disruptions. Mobile Agents can overcome unstable network conditions, such as intermittent communication failures. Mobile Agents can autonomously use different communication paths, or wait for the network to be restored, and resume operation.

2.6.7 Agent authenticity

Agents are reviewed and certified by a trusted third party. The trusted third party reviews the agent's code. According to Gatial [Gat10a], by certifying an agent, the trusted third party "indicates that the agent code does exactly what its creator states it should do and that it does not contain any malicious code, which jeopardizes the host platform integrity". We assume that the agent code is signed upon certification and that this signature is checked before an agent is instantiated.

2.6.8 Agent transport

The Secure Agent Infrastructure encrypts agents during transmission. The Mobile Agents are encoded into an agent package consisting of the serialized agent object(s) and the agent class file(s). The Secure Agent Infrastructure uses a key

encapsulation scheme that encrypts a symmetric ephemeral session key created by the sending platform with the public key of the receiving platform that is retrieved from the Public Key Infrastructure repository. The agent package is encrypted using the ephemeral session key. The Secure Agent Infrastructure uses AES in Electronic Codebook Mode of operation for bulk encryption and RSA for key transport. The private decryption key of the receiving platform is protected using the SDM and TDS combination that only grants access to private key if the load-time integrity of the executing platform has been successfully established.

2.6.9 Agent communication

Agent communication is encrypted using plain AES in Electronic Codebook Mode of operation. Agent communication protection uses the same ephemeral key that was used during agent transport.

2.6.10 Remarks

The authors of the Secure Agent Infrastructure correctly identified the need for data protection and also identified cryptography as the right tool, but we do have some serious concerns with the design choices, such as the complete lack of integrity protection, and implementation details, such as using the Electronic Codebook Mode mode of operation. As authentic, integrity protected, and confidential communication is, at least in principle, achievable, and recent versions of Jini (Apache River see Section 2.5) support secure transport protocols such as HTTPS, for the purpose of this thesis, we assume that inter-platform communication is handled by a secure communication protocol.

2.7 Security Goals

In this section we discuss the three prominent security goals of confidentiality, integrity and availability; the CIA triad as they are sometimes called. Note that there are communities who interpret CIA as confidentiality, integrity and authenticity. We, however, follow Bishop's taxonomy of qualifying authenticity as origin integrity [Bis02].

2.7.1 Confidentiality

According to Bishop [Bis02] "confidentiality is the concealment of information or resources." In this thesis we are primarily concerned with data confidentiality. Data confidentiality is the property that access to certain data is restricted to authorized individuals, entities or processes. Cryptography and OS access control are two prominent security mechanisms that ensure data confidentiality. Next to data confidentiality there are other confidentiality related security goals such as hiding the existence of information and resources or hiding that certain

information or resources were accessed. These forms of confidentiality are out of scope for this thesis.

2.7.2 Integrity

Bishop [Bis02] states that “integrity refers to the trustworthiness of data or resources, and is usually phrased in terms of preventing improper or unauthorized change.” We are concerned with two different forms of integrity. Data integrity, which refers to the content of information, and origin integrity, which is concerned with the source of the data. We follow the terminology where origin integrity is authentication.

There are two main classes of integrity protection mechanisms. These are prevention mechanisms and detection mechanisms. Prevention mechanisms try to prevent unauthorized access to data and unauthorized modification. Bishop points out that it is important to differentiate between an unauthorized entity gaining access to data, versus an authorized entity changing the data in an unauthorized way. Bishop further states that the two cases require two completely different sets of controls to mitigate.

Detection mechanisms try to make integrity violations evident; they do not prevent unauthorized modifications. We are primarily concerned with detection mechanisms. For example, hash functions which we introduce in Section 2.9.1 can help detect data integrity violations, and Message Authentication Codes (MACs) (see Section 2.9.2) are geared towards detecting both data integrity and origin integrity violations.

2.7.3 Availability

Data, a resource, or a system is only of service if it is available for use. Therefore, the security goal of availability is to prevent an adversary from making data, a resource, or a system unavailable (see Denial-of-Service in Section 2.8.2).

2.8 Threat Modeling

Threat modeling is a central aspect of this thesis. For one, the threat modeling of the Secure Agent Infrastructure Multi-Agent System described in Section 2.6 is a central contribution of this thesis (see Chapter 4). Furthermore we use threat modeling to better illustrate the capabilities of the Secure Block Device in Chapter 6.

2.8.1 Definitions

As threat modeling is a very flexible term [Sho08], here we define threat modeling as it applies to this thesis. As a basis we first define the terms *security policy*, *security mechanism*, *adversary/attacker*, *vulnerability*, and *threat*, and their specific meaning herein.

Definition 2.1. A security policy is a statement of what is, and what is not allowed in a system [Bis02].

Security policies can be informal statements such as: “In this test you are not allowed to copy the results from any of your colleagues.”, but also mathematically formalized as a list of allowed and disallowed states [Bis02].

Definition 2.2. A security mechanism is a method, tool, or procedure to enforce a security policy [Bis02].

Often several security mechanisms have to work together to enforce a security policy.

Definition 2.3. An adversary, or attacker, is an entity motivated to violate a system’s security policy.

Definition 2.4. A vulnerability is a set of conditions that allows an attacker to violate an explicit or implicit security policy [Sea05].

Definition 2.5. A threat is the potential violation of the confidentiality, integrity, or the availability of an asset by an adversary exploiting a system vulnerability.

We could also conveniently define threats via the security policy. Here a threat is the potential violation of an element of the security policy. However, in our experience we define a system’s security policy, by first analysing the specific threats to that system. Hence, we use a definition that is based on the violation of the security goals of confidentiality, integrity and availability, and not on the violation of a security policy.

Definition 2.6. Threat modeling is the process of identifying specific threats to a system. The goal of this process is to identify, describe, rank, and mitigate these threats.

A threat model can also be applied to verify the effectiveness of security mechanisms in an implementation against the modeled threats.

2.8.2 Threats

In Chapter 4, we use STRIDE-per-interaction (see Section 2.8.5) for enumerating threats. STRIDE is an mnemonic for Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege, the six threats considered by the STRIDE threat enumeration approach. Here we describe these six threats. We use Matt Bishop’s Introduction to Computer Security [Bis02] as source.

Spoofing Spoofing or masquerading is the impersonation of one entity by another. It lures the victim into believing that the entity with which it is communicating is a different entity [Bis02]. For example, consider a Mobile Agent (see Section 2.1) that wants to migrate to a specific target agent

platform (see Section 2.2). If a malicious agent platform different from the target platform is able to masquerade as the target platform it spoofs the identity of the target agent platform. Spoofing is a threat to integrity, more specifically origin integrity or authenticity, and it is countered by authentication services.

Tampering Bishop calls this modification or alteration and describes it as an unauthorized change of information. A classical example for a tampering threat is the Man-in-the-Middle attack. Here an adversary reads incoming messages from a sender and sends modified versions to the recipient. Tampering is an integrity threat and can be countered with integrity services.

Repudiation Bishop discusses this threat as denial of receipt, as opposed to repudiation of origin. In denial of receipt an entity repudiates ever having received a specific information, whereas in repudiation of origin an entity denies creation of a specific piece of information. In all STRIDE-per-interaction threats we discuss in this thesis repudiation always refers to denial of receipt. For example, when a Mobile Agent migrates to a different agent platform, the receiving platform might repudiate ever having received the migrating agent. One potential motivation for realizing this particular threat could be that the receiving platform compromised the agent and wants to obfuscate this fact. This threat is affecting both availability and integrity and can be mitigated by a combination of availability and integrity mechanisms.

Bishop and STRIDE both cover repudiation of origin indirectly via spoofing. If it is possible to spoof the identity of an entity, this entity can always repudiate having sent anything. However, if this lack of origin integrity is mitigated with a suitable authentication mechanism, repudiation of origin is also mitigated.

Information Disclosure Information disclosure, or as Bishop calls it, snooping is the unauthorized interception of information. Bishop qualifies it as passive, that is, some entity is eavesdropping on communications or files. An example for information disclosure is a router on the Internet reading the packets it routes in addition to actually routing them. Information disclosure is a clear violation of confidentiality.

Denial-of-Service Here Bishop differentiates between two threats that are covered by the single Denial-of-Service category in STRIDE. These threats are delay and Denial-of-Service. Delay is a temporary inhibition of a service. For example, when an agent platform is executing a Mobile Agent so slow that the Mobile Agent cannot perform its task in a timely fashion, then this is a delay attack. Denial-of-Service is a long term inhibition of a service. To extend our example, here the malicious agent platform does not execute the agent at all, which is equivalent to executing the Mobile Agent with an infinite delay. Denial-of-Service violates availability and can be countered by availability mechanisms.

Elevation of Privilege Elevation of privilege, or privilege escalation is when an otherwise unauthorized entity gains access to privileged services or resources. Bishop does not consider this threat, however we consider elevation of privilege as a threat to authorization and thus a threat to confidentiality, integrity, availability.

A very prominent example of this is when a normal user of a system gains administrative privileges, or more specifically, if a process running as an unprivileged user is able to acquire administrative rights through an implementation attack, such as a buffer overflow with shell code injection. This process then can circumvent OS access control and can potentially read confidential data, corrupt data and thus violate integrity, or shutdown or slow down the system to violate availability.

2.8.3 The Microsoft Security Development Lifecycle Threat Modeling Methodology

Microsoft's Security Development Lifecycle threat modeling methodology is a practical approach for analysing software, usable by non-experts, and centered on Data Flow Diagrams and the STRIDE-per-element threat enumeration technique [Sho08]. Microsoft's Security Development Lifecycle has been adapted to use the STRIDE-per-interaction threat enumeration technique [Sho14, aT14, MTM15]. Threat modeling at Microsoft has been first documented in 1999 [KG99], and later became a part of Microsoft's Security Development Lifecycle [HL02, Sho08, Sho14]. The threat modeling methodology has evolved over the years [Sho08] and the following description is based on the latest iteration [Sho08, Sho14].

Microsoft's threat modeling methodology is a 4 step process with the goals of improving the security of designs, documenting the security design activity and teaching the people working on the process about security. The 4 steps of the threat modeling process are diagramming, threat enumeration, mitigation, and verification.

Diagramming The threat modeling methodology uses a diagramming system derived from standard Data Flow Diagrams to model the system under analysis. In addition to the standard Data Flow Diagram elements of *Process*, *Data store*, *Data flow* and *External entity*, the threat modeling methodology also defines trust boundaries, where the different sides of the boundary operate on different levels of privilege. We have listed the five Data Flow Diagram elements in use by Microsoft's threat modeling methodology in Table 2.1. According to Shostack [Sho08], Data Flow Diagrams were chosen for threat modeling, because they are easy to understand and because modeling the flow of data naturally reveals potential attack vectors for attacks on data and code confidentiality and integrity.

Threat enumeration The Microsoft Security Development Lifecycle supports both the STRIDE-per-element and the STRIDE-per-interaction technique to enumerate threats. Both techniques use the Data Flow Diagram model

Table 2.1: Description of the elements in the Microsoft Security Development Lifecycle threat modeling methodology’s diagramming system. The diagramming system is based on Data Flow Diagrams, so this description encompasses Data Flow Diagram elements, as well as the trust boundary element added by the threat modeling methodology. This table is based on a table in [Sho08].

Name	Representation	Definition	Examples
External entity		Entities outside of control	Humans, other systems, web sites
Process		A running program	Executables, components
Data flow		Flow of information	Function calls, Remote Procedure Call
Data store		Data at rest	Files, databases
Trust boundary		Different privilege levels	Process boundaries, machine boundaries

of the system created in the first step to generate a list of threats. The threats in this list fall into six categories. These six categories are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege. Together these six categories make up the STRIDE mnemonic and we detail these six threat categories in Section 2.8.2. We discuss how the STRIDE-per-element and STRIDE-per-interaction threat enumeration techniques work, and how they differ in Section 2.8.5.

Mitigation Microsoft’s Security Development Lifecycle considers four approaches to mitigation, in order of preference: redesign, use standard mitigations, such as access control, use unique mitigations (with caution), or accept the risk in accordance with policies [Sho08].

Validation Microsoft’s Security Development Lifecycle uses heuristics, such as graph analysis of the Data Flow Diagrams, for validating threat models [Sho08].

2.8.4 The Microsoft Threat Modeling Tool Family

The Microsoft Threat Modeling Tool family consists of tools for creating Data Flow Diagrams of systems, analyze the Data Flow Diagrams for automatic gener-

ation of potential threats, and creating reports. The Microsoft Threat Modeling tool family supports implementing the Microsoft Security Development Lifecycle. In this thesis we use the Microsoft Threat Modeling Tool 2014³ [aT14] and the Microsoft Threat Modeling Tool 2016⁴ [MTM15]. Both the Microsoft Threat Modeling Tool 2014 and 2016 use the STRIDE-per-interaction threat enumeration technique [aT14, MTM15].

2.8.5 Threat Enumeration

Here we discuss the STRIDE-per-element and STRIDE-per-interaction threat enumeration techniques. Both are variants of STRIDE. STRIDE was first documented by Loren Kohnfelder and Praerit Garg at Microsoft [KG99]. STRIDE is a mnemonic made up from the six different threat categories it considers. These threat categories are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege. According to Shostack [Sho14], the STRIDE framework and mnemonic were developed to help people developing software to identify the types of attacks that software tends to experience.

STRIDE-per-element

STRIDE-per-element is a semi-formal, prescriptive approach to threat enumeration that uses the model represented by the Data Flow Diagram of the system under analysis for enumerating potential threats that can occur for data flows originating or terminating in specific elements. STRIDE-per-element is based on STRIDE. STRIDE-per-element is build on the observation that certain threats are more prevalent with certain elements of the Data Flow Diagram [Sho14]. By limiting the threats to consider per element this approach makes it simpler to find threats. The threat category to Data Flow Diagram element association is depicted in Table 2.2. For every element in the Data Flow Diagram model of the system under analysis the threat modeler only considers those threats that have a tick in the respective STRIDE category. For example, for an external entity the threat modeler only considers Spoofing threats, when using STRIDE-per-element.

Shostack [Sho14] estimates that “When you have a threat per ✓ in the STRIDE-per-element table, you are doing reasonably well. If you circle around and consider threats against your mitigations (or ways to bypass them) you’ll be doing pretty well.”. We believe that Shostack wants to state that if a threat modeler follows the STRIDE-per-element methodology and can come up with at least one threat per tick in Table 2.2, then this is a good start. If the threat modeler then chooses mitigation mechanisms for these threats and considers threats against these mitigation mechanisms, the threat modeler will “be doing pretty

³<http://blogs.microsoft.com/cybertrust/2014/04/15/introducing-microsoft-threat-modeling-tool-2014/>

⁴<https://blogs.microsoft.com/cybertrust/2015/10/07/whats-new-with-microsoft-threat-modeling-tool-2016/>

Table 2.2: The mapping of Data Flow Diagram (DFD) element types to specific threat categories in the STRIDE-per-element threat enumeration technique. The threat categories are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege. The question mark for the Data store under Repudiation signifies that data stores need to be treated differently, if they are used for authentic logging. In such a case repudiation threats need to be considered. This table is adapted from [Sho14].

DFD element	S	T	R	I	D	E
External entity	✓		✓			
Process	✓	✓	✓	✓	✓	✓
Data flow		✓		✓	✓	
Data store		✓	?	✓	✓	

well[?]. Furthermore, we believe that Shostack’s statement is based on his personal experience. In Section 3.1.1 we discuss a descriptive study by Scandariato et al. [SWJ15] that evaluates the effectiveness of STRIDE-per-element.

STRIDE-per-interaction

STRIDE-per-interaction is an approach to threat enumeration that considers tuples of (*origin, destination, interaction*) and enumerates threats against these tuples [Sho14]. STRIDE-per-interaction was developed by Microsoft’s Larry Osterman and Douglas MacIver. According to Shostack [Sho14] STRIDE-per-interaction leads to the same number of threats as STRIDE-per-element, but the threats may be easier to understand with this approach. However, Shostack claims that STRIDE-per-interaction is too complex to use without having a reference, such as Table 2.3, at hand. We believe this makes the Microsoft Threat Modeling Tool family (see Section 2.8.4) that implements automatic STRIDE-per-interaction based threat generation especially helpful.

Table 2.3 illustrates how STRIDE-per-interaction maps (*origin, destination, interaction*) tuples to the STRIDE threat categories. Note that with STRIDE-per-interaction the term external entity has been replaced by external interactor. The first column of the table (Element) is the *origin* and the second column (Interaction) combines *destination* and *interaction*. For example, for a process that has an outbound data flow to a data store, the threat modeler has to consider the threats from the Spoofing and Information Disclosure threat categories. If the data flow also crosses a trust boundary, the threat modeler has to additionally deliberate tampering and repudiation threats.

We use an example adapted from [Sho14] to illustrate the STRIDE-per-interaction threat enumeration technique in conjunction with the Microsoft Threat Modeling Tool 2016. Figure 2.8 depicts a very simple system illustrating all of the possible interactions listed in Table 2.3. We have modeled this simple system using Microsoft Threat Modeling Tool 2016. The tool outlines 53 potential

Table 2.3: The mapping of Data Flow Diagram (DFD) element types, information flow destinations, and interaction types to specific threat categories in the STRIDE-per-interaction threat enumeration technique. The threat categories are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege. This table is adapted from [Sho14].

Element	Interaction	S	T	R	I	D	E
Process	has outbound data flow to data store.	✓		✓			
	sends output to another process.	✓	✓	✓	✓	✓	✓
	sends output to external interactor (code).	✓	✓	✓	✓		
	sends output to external interactor (human).			✓			
	has inbound data flow from a data store.	✓	✓			✓	✓
	has inbound data flow from a process.	✓	✓			✓	✓
	has inbound data flow from external interactor.	✓				✓	✓
Data flow	crosses machine boundary.		✓		✓	✓	
Data store	Process has outbound data flow to data store.	✓	✓	✓	✓		
	Process has inbound data flow from data store.			✓	✓	✓	
External	passes input to the process.	✓		✓	✓		
Interactor	gets input from process.	✓					

threats in the 37 (*origin, destination, interaction*) tuples-to-threat-categories-mappings listed in Table 2.3. In the interest of brevity we only excerpt the first 11 threats enumerated by the tool for the interaction *Input* from *External* to *Process 1* in Figure 2.8 and list them in Table 2.4.

As *External* is an external interactor and *Process 1* is a process, three lines from Table 2.3 apply. The first line is “Process has an inbound data flow from external interactor.” As the data flow crosses a trust boundary, the second is “Data flow crosses a machine boundary”. The third is “External Interactor passes input to the process”. Note that the lines “Process sends output to external interactor (code/human)” do not apply because here the external interactor sends data and not the process. The first line requires a threat modeler to check for spoofing, Denial-of-Service, and elevation of privilege threats. The second line mandates checking for tampering, information disclosure, and Denial-of-Service threats. Finally, the third line indicates that checking for spoofing, repudiation and Denial-of-Service threats is advisable. So for this (*origin, destination, interaction*) tuple, all six threat categories are pertinent, and some threat categories are indicated by several distinctive rules.

We also provide Table 2.4 as an example of the threat titles and threat descriptions the tool generates for its reports. We reprint the threat titles and threat descriptions verbatim in the table.

Shostack [Sho14] rates the effectiveness of STRIDE-per-interaction to find threats as follows: “When you have threat per ✓ in the STRIDE-per-interaction

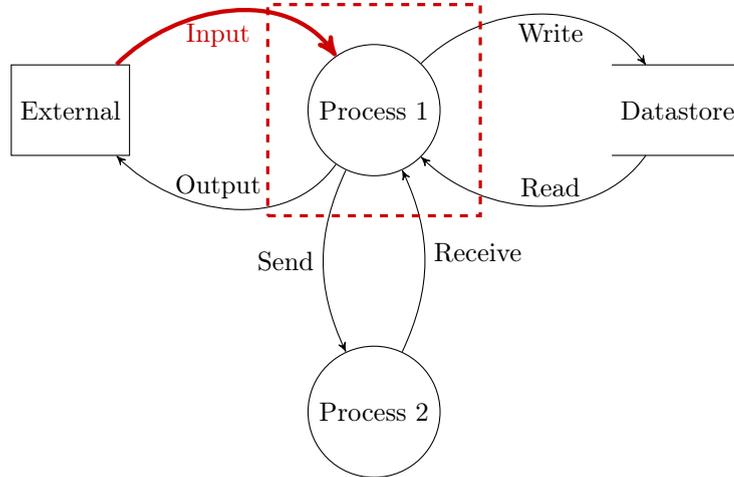


Figure 2.8: A Data Flow Diagram of a simple system illustrating all possible STRIDE-per-interaction interactions listed in Table 2.3. We have emphasized the **Input** data flow, as we use it in an example to detail how STRIDE-per-interaction works.

table, you are doing reasonably well. If you circle around and consider threats against your mitigations (or ways to bypass them) you’ll be doing pretty well.” We assume that this estimate is based on Shostack’s personal experience. In Section 3.1.1 we discuss a descriptive study by Williams and Yuan [WY15] that evaluates the effectiveness of the Microsoft Threat Modeling Tool 2014 that uses STRIDE-per-interaction.

Table 2.4: An adapted excerpt of the report generated by the Microsoft Threat Modeling Tool 2016 for the system modeled by the Data Flow Diagram depicted in Figure 2.8. This table lists the 11 potential threats and their description as enumerated by the tool for the interaction *Input* between *External* and *Process 1* in Figure 2.8. These are only 11 of the 53 threats proposed for analysis by the tool for the overall system model.

STRIDE	Threat / Description
✓	<p>Spoofing the Generic External Interactor External Entity</p> <p>External may be spoofed by an attacker and this may lead to unauthorized access to Process 1. Consider using a standard authentication mechanism to identify the external entity.</p>
✓	<p>Elevation Using Impersonation</p> <p>Process 1 may be able to impersonate the context of External in order to gain additional privilege.</p>

Table 2.4 – continued from previous page

S T R I D E	Threat / Description
✓	<p>Spoofing the Process 1 Process Process 1 may be spoofed by an attacker and this may lead to information disclosure by External. Consider using a standard authentication mechanism to identify the destination process.</p>
✓	<p>Potential Lack of Input Validation for Process 1 Data flowing across Input may be tampered with by an attacker. This may lead to a denial of service attack against Process 1 or an elevation of privilege attack against Process 1 or an information disclosure by Process 1. Failure to verify that input is as expected is a root cause of a very large number of exploitable issues. Consider all paths and the way they handle data. Verify that all input is verified for correctness using an approved list input validation approach.</p>
✓	<p>Potential Data Repudiation by Process 1 Process 1 claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data.</p>
✓	<p>Data Flow Sniffing Data flowing across Input may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations. Consider encrypting the data flow.</p>
✓	<p>Potential Process Crash or Stop for Process 1 Process 1 crashes, halts, stops or runs slowly; in all cases violating an availability metric.</p>
✓	<p>Data Flow Input Is Potentially Interrupted An external agent interrupts data flowing across a trust boundary in either direction.</p>
✓	<p>Process 1 May be Subject to Elevation of Privilege Using Remote Code Execution External may be able to remotely execute code for Process 1.</p>
✓	<p>Elevation by Changing the Execution Flow in Process 1 An attacker may pass data into Process 1 in order to change the flow of program execution within Process 1 to the attacker's choosing.</p>

Table 2.4 – continued from previous page

S T R I D E	Threat / Description
✓	<p>Cross Site Request Forgery</p> <p>Cross-site request forgery (CSRF or XSRF) is a type of attack in which an attacker forces a user’s browser to make a forged request to a vulnerable site by exploiting an existing trust relationship between the browser and the vulnerable web site. In a simple scenario, a user is logged in to web site A using a cookie as a credential. The user then browses to web site B. Web site B returns a page with a hidden form that posts to web site A. Since the browser will carry the user’s cookie to web site A, web site B now can take any action on web site A, for example, adding an admin to an account. The attack can be used to exploit any requests that the browser automatically authenticates, e.g. by session cookie, integrated authentication, IP whitelisting, etc. The attack can be carried out in many ways such as by luring the victim to a site under control of the attacker, getting the user to click a link in a phishing email, or hacking a reputable web site that the victim will visit. The issue can only be resolved on the server side by requiring that all authenticated state-changing requests include an additional piece of secret payload (canary or CSRF token) which is known only to the legitimate web site and the browser and which is protected in transit through SSL/TLS. See the Forgery Protection property on the flow stencil for a list of mitigations.</p>

2.9 Cryptographic Primitives

Cryptography provides one of the most powerful, if not the most powerful, sets of security mechanisms available to achieve the security goals, of Confidentiality, Integrity and Availability. Note that integrity includes origin integrity, commonly also known as authenticity. The primary field of application of cryptography is integrity, including authenticity, and confidentiality. As such we frequently use cryptography throughout this thesis. We expect you, esteemed reader, to understand the basics of symmetric and asymmetric cryptography, including for example cryptographic primitives such as AES or RSA. However, the Secure Block Device we introduce in Chapter 6 relies heavily on cryptographic hash functions, Message Authentication Codes (MACs) and Authenticated Encryption (AE). We therefore chose to briefly describe these three cryptographic primitives here.

2.9.1 Cryptographic Hash Functions

A hash function is any function that maps input data of arbitrary size to a fixed size output, the hash. A hash function has the following form:

$$h = H(x). \quad (2.1)$$

Here H designates a hash function, x is an input of arbitrary size, and h is the fixed size hash value, or simply hash.

A cryptographic hash function H is a hash functions that additionally fulfills the following three properties.

Pre-image resistance Given a hash h it should be computationally infeasible to find any input x such that $h = H(x)$.

Collision resistance It should be computationally infeasible to find two distinct inputs x_1, x_2 such that $H(x_1) = H(x_2)$.

Second pre-image resistance Given H and an input x_1 such that $h = H(x_1)$ it should be computationally infeasible to find a second input x_2 such that $H(x_1) = H(x_2)$.

Hash functions are widely used in computer security to achieve data integrity, secure password storage, short representatives of data (data identifier), etc.

Cryptographic Hash Functions and Data Integrity

Hash functions can compute short representatives of arbitrary amounts of data. These representatives can, for example, be used to determine data equality. Consider that you want to compare the content of two files f_x and $f_{x'}$. One way to compare the files would be to compare them bit by bit. However, with hash functions there is an alternative. You can use hash functions to test for equality by computing $h_x = H(f_x)$ and $h_{x'} = H(f_{x'})$ and then testing if $h_x = h_{x'}$. This can be useful if you want to test if a file was changed over time. Compute a hash $h_1 = H(f_x)$ at time t_1 and compare it with $h_2 = H(f_x)$ at time t_2 (now). If $h_1 \neq h_2$ the files have changed. However for the comparison to be meaningful you need to be sure that it is still your unmodified hash h_1 you use for the comparison. You need to be sure that h_1 is still authentic. You can achieve this by storing it in an authentic Datastore (see Section 2.10).

The advantage of this approach for data integrity is that instead of having to store the full data in an authentic memory, such as an authentic Datastore, it suffices to store the, typically significantly smaller, hash value in an authentic Datastore. Authentic data storage can, for example, be implemented with Security Controllers, where storage is at a premium. Thus cryptographic hash functions provide a computation time (recomputing the hash value) versus memory trade-off (storing hash versus storing an arbitrary amount of data).

Concerning the security of protecting data integrity using cryptographic hash functions the following holds: Due to the collision resistance property of a cryptographic hash function, it is computationally infeasible to find two arbitrary

different chunks of data that map to the same hash value. Furthermore, due to the second pre-image resistance property of cryptographic hash function given a specific chunk of data and corresponding hash value, it must be computationally infeasible to find a second chunk of data that maps to the same hash value. Thus cryptographic hash functions can prevent intentional forgeries of hash values, by manipulating the input to the hash function.

Note that cryptographic hash functions per se do not provide data authenticity. Authenticity arises from having proof corroborating the origin of data. Anyone having access to specific data can compute a hash of that data. Therefore, a hash value alone gives no indication of where data originated. To achieve this we need a more powerful cryptographic primitive, the MAC. Note however that MACs can be created by using cryptographic hash functions as underlying component.

Secure Hash Algorithm

The Secure Hash Algorithm is a family of cryptographic hash functions. It is published by the National Institute of Standards and Technology. The Secure Hash Algorithm family includes SHA-0, SHA-1, SHA-2, and SHA-3. Of these SHA-1 and SHA-2 are of interest to us.

SHA-1 SHA-1 is a 160-bit hash function designed by the National Security Agency. SHA-1 has the following form

$$h = \text{SHA-1}(x), \quad (2.2)$$

where h is the 160-bit (20 bytes) output value, and x is the input message with a maximum length of $2^{64} - 1$ bytes.

The SHA-1 function is cryptographically broken. Cryptographically broken means that theoretical weaknesses have been found, but no practical attack exists yet. Nevertheless, the cryptographic community and we believe it no longer advisable to use SHA-1. SHA-1 is the one hash function used by the Trusted Platform Module (TPM) 1.2 we introduce in Section 2.14.

SHA-2 SHA-2 is not a single hash function, but a family of related hash functions. We use the SHA-256 hash of the SHA-2 hash family for implementing the Merkle Tree (see Section 2.12) that protects the memory authenticity (see Section 2.10) of the Secure Block Device we introduce in Chapter 6.

SHA-256 is a 256-bit hash function designed by the National Security Agency. The SHA-2 function takes the following form

$$h = \text{SHA-256}(x), \quad (2.3)$$

where h is the 256-bit (32 bytes) output value, and x is the input message with a maximum length of $2^{64} - 1$ bytes.

We are not aware of any attacks, be they theoretical or practical, against SHA-2 in general and SHA-256 specifically that would render SHA-2 (cryptographically) broken.

2.9.2 Message Authentication Codes

A MAC is generated by a function M that accepts a secret key k , an input x of arbitrary size, and generates an output t of fixed size, the authenticity tag or MAC. A MAC function has the following form:

$$t = M(k, x). \quad (2.4)$$

Secure MAC functions must resist existential forgery under chosen-plaintext attacks: It must be computationally infeasible for an adversary to create a tuple (t, x) consisting of a tag t and a message x , that would validate under a certain key k , without being in possession of k . Furthermore, even given a MAC oracle that computes MACs for the adversary and that is in possession of the key k , the adversary must not learn anything from any (t, x) pairs created by the oracle that makes an existential forgery easier. Simply put, learning the key k from unlimited use of such a MAC oracle still has to be computationally infeasible.

MAC functions are used in computer security to achieve data authenticity; that is, data integrity and data origin integrity. Typically a message m is input into a MAC function with a secret key k_s . The message m and the output tag t_{snd} are then sent to a recipient who shares the same secret key k_s . The recipient then computes $t_{rec} = M(k_s, m)$ and compares the received t_{snd} with the freshly computed t_{rec} . If the two authentication tags match, the message content has not been tampered with and is originating with an entity that holds k_s .

Cipher-based Message Authentication Code

Cipher-based Message Authentication Code (CMAC) is a block cipher-based MAC function published by the National Institute of Standards and Technology [Dwo05]. We use the AES-CMAC algorithm described in [SPLI06] with the Secure Block Device we introduce in Chapter 6.

2.9.3 Authenticated Encryption

Authenticated Encryption schemes strive to achieve both data confidentiality and data authenticity simultaneously. AE schemes are a relatively new cryptographic approach that arose to alleviate the problems stemming from insecure compositions of existing confidentiality and integrity primitives [BN08]. AE schemes are semantically secure under a chosen plaintext attack. Therefore, an adversary cannot learn anything about the plaintext from its corresponding ciphertext. Furthermore, by definition AE schemes are secure against chosen ciphertext attacks, as they detect invalid ciphertext, and refuse to decrypt it.

An AE scheme consists of two functions (enc_{AE}, dec_{AE}) , enc_{AE} for encryption and dec_{AE} for decryption. These functions are:

$$\begin{aligned}(ct, t) &= enc_{AE}(k, m, ad) \\ m &= dec_{AE}(k, ct, ad, t),\end{aligned}$$

where m is a plaintext message, k a secret key, and ad is associated data. The AE encryption function enc_{AE} processes the plaintext m , the associated data ad and the key k , to compute the ciphertext ct and the authentication tag t . The associated data is not encrypted, but its integrity is protected by the AE scheme. The authentication tag t effectively acts as a MAC for the input of enc_{AE} . The decryption function dec_{AE} decrypts the cipher text and verifies the authenticity of the message m and the associated data ad using the key k . If either ad , t , or ct is modified, the decryption process will fail and the integrity violation will become evident.

Note that use of an initialization vector (IV) is not uniform across AE schemes. Many block cipher mode of operations use an IV to achieve ciphertext randomization, a property that requires that repeated encryptions of an identical plaintext yield different ciphertexts. An IV is a cryptographic nonce, a bit string of a certain size, where a specific valuation can only be used once. Some schemes, such as the Offset Codebook Mode (OCB) scheme used here, require an additional IV iv as input to both enc_{AE} and dec_{AE} . The iv is again protected by the AE scheme; if the iv changes the decryption will fail.

Finally, the effect of the associated data (ad) on the ciphertext is also non-uniform. In some AE schemes modifying ad , while holding all other inputs constant, will also change the ciphertext ct , whereas in other schemes, modifying ad will not change the ct . The tag t however must always change, if ad changes. For example, for the OCB AE scheme changing ad does not change the ct , only t , while for the Synthetic Initialization Vector (SIV) AE scheme modifying ad completely randomizes the ct and changes t .

Due to the non-uniformity of AE schemes it is difficult to generalize their security properties. We therefore discuss the security properties of the two AE schemes we use for the Secure Block Device, that is OCB and SIV, in the two following sections.

Offset Codebook Mode

OCB [RBB03] is an efficient single pass AE scheme. A single pass AE scheme only requires one block cipher operation per block of encrypted data. A potential drawback of OCB is that it is patented and its license⁵ restricts under which use cases it can be used without paying royalties. At the time of writing, the use of OCB implementations is free for Open-Source software, and non-military non-commercial, or non-military research purposes.

According to Rogaway et al. [RBBK01], OCB provides confidentiality under a chosen-plaintext attack and integrity under authenticity of ciphertexts. Together these two properties imply protection against malleability under a chosen-

⁵<http://www.cs.ucdavis.edu/~rogaway/ocb/license.htm>

ciphertext attack. In a chosen-ciphertext attack an adversary is able to obtain the decryption of multiple ciphertexts under an unknown key. However, even under this strong assumption, with OCB, the adversary cannot learn anything about the key k used, nor can he learn anything that makes a forgery of a (ct, t) tuple easier.

However, when using OCB, there is one caveat to consider. OCB is vulnerable under a collision attack [Fer02] that breaks the above security guarantees if an adversary obtains close to, or even more than, 2^{64} ciphertext blocks created under the same key k . Therefore Krovetz and Rogaway [KR14] stipulate that a single key should at most be used to create 2^{48} ciphertext blocks (4 petabytes of data). Furthermore, Krovetz and Rogaway emphasize the importance of never reusing the same IV, as obtaining two (ct, t) pairs for different plaintexts, but identical IVs and encrypted under the same key, allows the adversary to forge future (ct, t) pairs.

Synthetic Initialization Vector

SIV [RS06] is a two-pass AE scheme, meaning SIV requires two block cipher operations per block of encrypted data. For encryption the scheme first computes the MAC of the data and associated data, and then uses this MAC as IV for the encryption pass.

SIV is a Deterministic Authenticated Encryption construction [RS06] where the notion of security directly captures the amalgamation of privacy (confidentiality) and authenticity. In layman's terms an SIV encrypted ciphertext is indistinguishable from a random bit string of the same size, and without knowledge of the key an adversary will be unable to create an input to an SIV decryption (ct, t) that will decrypt successfully.

2.10 Single-User Block Datastore Authentication

In Chapter 6 we will introduce the Secure Block Device. The Secure Block Device is a software library that uses cryptography to establish certain security guarantees over data stored in an insecure storage location. In this section we introduce a number of definitions to reason about the security guarantees the Secure Block Device provides.

Here we define Single-User Block Datastore Authentication and the Single-User Authentic Block Datastore. Single-User Block Datastore Authentication is similar to memory authentication [ECG⁺09] and fork consistency [MS01, LKMS04], but we further abstract these notions to better fit our needs. Before defining Single-User Block Datastore Authentication and Single-User Authentic Block Datastore, we will first define the terms Datastore, Data Block, Block Datastore, Block Datastore Authentication, and Authentic Block Datastore. We base our definition of Block Datastore Authentication and Single-User Block Datastore Authentication on the definition of Memory Authenticity by Elbaz et al.

Definition 2.7. *Memory authentication* is defined as the ability to verify that the data read from memory by the processor (or by a specific application) at a given address is the data it last wrote at this address. [ECG⁺09]

Definition 2.8. A *Datastore* is a hardware component that provides a software interface for reading and writing data to a persistent storage. Datastores are of finite size s_S . Examples for Datastores are NVRAMs, block devices, hard disks, flat files, and databases.

Definition 2.9. A *Data Block* is a logical unit delimiting a contiguous area of memory. Within a Data Block the smallest addressable data unit is a byte. The size of a Data Block s_B is measured in bytes and we require $s_B \geq 1$ and s_B to be finite. Each byte in a Data Block is randomly accessibly, where the index $i \in \{0, \dots, s_B - 1\}$.

Definition 2.10. A *Block Datastore* is a Datastore whose only addressable and accessible unit is a Data Block of fixed size s_B . A Block Datastore of size s_S consists of n Data Blocks, where $s_S = n \cdot s_B$, and n , and thus also s_S , are finite. We index Data Blocks via their address DB_a , where $0 \leq a < n$.

Definition 2.11. A *Single-User Block Datastore* is a Block Datastore that is intended to be used by a single user. Specifically, all authorized read and write requests originate from the same source, the user, and are sequential in order.

Definition 2.12. *Block Datastore Authentication* is an authorized user's ability to verify that a Data Block DB_a read from a Block Datastore at address a is the Data Block last written at address a by any authorized user.

Definition 2.13. *Single-User Block Datastore Authentication* is the single authorized user's ability to verify that a Data Block DB_a read from a Single-User Block Datastore at address a is the Data Block last written at address a by

Note that Single-User Block Datastore Authentication has to hold even in the face of an adversary that can directly read from, or write to, the Single-User Block Datastore.

Definition 2.14. An *Authentic Block Datastore* is a Block Datastore where Block Datastore Authentication always holds. Similarly, a *Single-User Authentic Block Datastore* is an Single-User Block Datastore where Single-User Block Datastore Authentication always holds.

The Secure Block Device is a software library that implements a Single-User Authentic Block Datastore on top of an existing Single-User Block Datastore. The Secure Block Device uses cryptography to provide Single-User Block Datastore Authentication, even though the Single-User Block Datastore used for storage is vulnerable to attacks by an adversary. We outline these attacks on Single-User Block Datastore Authentication in the next section. We specifically also introduced the notion of an Authentic Block Datastore, in order to show why we can ignore forking attacks on a Single-User Authentic Block Datastore.

2.11 Attacks on Memory Authentication

The adversarial model on memory authentication is defined by the attacks an adversary can mount against memory authentication. We will later use the same attacks to define the adversarial model against our implementation of a Single-User Authentic Block Datastore, the Secure Block Device. We introduce the Secure Block Device in Chapter 6. The Secure Block Device implements a Single-User Authentic Block Datastore on top of a Single-User Block Datastore. We will discuss the adversarial model against the Secure Block Device in Section 6.4.1. Specifically, we will discuss what attacks the Secure Block Device has to mitigate in order to achieve Single-User Block Datastore Authentication although it relies on a Single-User Block Datastore that is potentially under the control of an adversary.

The attacks on memory authentication are *Spoofing*, *Splicing*, or *Relocation*, and *Replay*, which we will detail in the following. These attacks are well known attacks in the fields of memory authentication [ECG⁺09] and file system integrity [MS01]. The field of file system integrity also considers a 4th attack called *Forking*. Forking attacks will not be part of our adversarial model for our Single-User Authentic Block Datastore. However, we detail the attack here to highlight why it can be disregarded for any Single-User Authentic Block Datastore and consequently by the Secure Block Device.

Spoofing The adversary is able to pass off arbitrary data as an authentic Data Block in the Block Datastore.

As illustrated in Figure 2.9 the adversary partially or fully overwrites at least one data block with arbitrary data. In this example the adversary fully overwrites the Data Block DB_4 with arbitrary data. At a later point in time DB_4 is then read by an authorized user who accepts it as authentic.

Splicing or relocation The adversary is able to substitute an authentic Data Block DB_x with a different authentic Data Block DB_y from the same Block Datastore, where $x, y \in \{0 \dots n\} \wedge x \neq y$. This can be interpreted as a spatial permutation of the Data Blocks in a Block Datastore.

Figure 2.10 demonstrates a splicing attack. Here, the adversary fully reads the Data Block at address 7 (DB_7) and fully overwrites DB_4 with it. Later, an authorized user reads back DB_7 at index 4 and accepts it as authentic.

Replay The adversary is able to successfully substitute a previously recorded copy of a specific Data Block $DB_{x_{t1}}$ and play it off as the newest version of the Data Block $DB_{x_{t2}}$. A replay attack therefore is a temporal permutation of the Data Blocks in a Block Datastore.

In Figure 2.11 we depict a replay attack. First, the adversary obtains a copy of Data Block 4 ($DB_{4_{t1}}$) and stores it for later use. At some point in the future an authorized user of the Block Datastore updates the Data Block at address 4 with $DB_{4_{t2}}$. Now the adversary, who disagrees with this update, overwrites $DB_{4_{t2}}$ with his older copy $DB_{4_{t1}}$. Finally, next

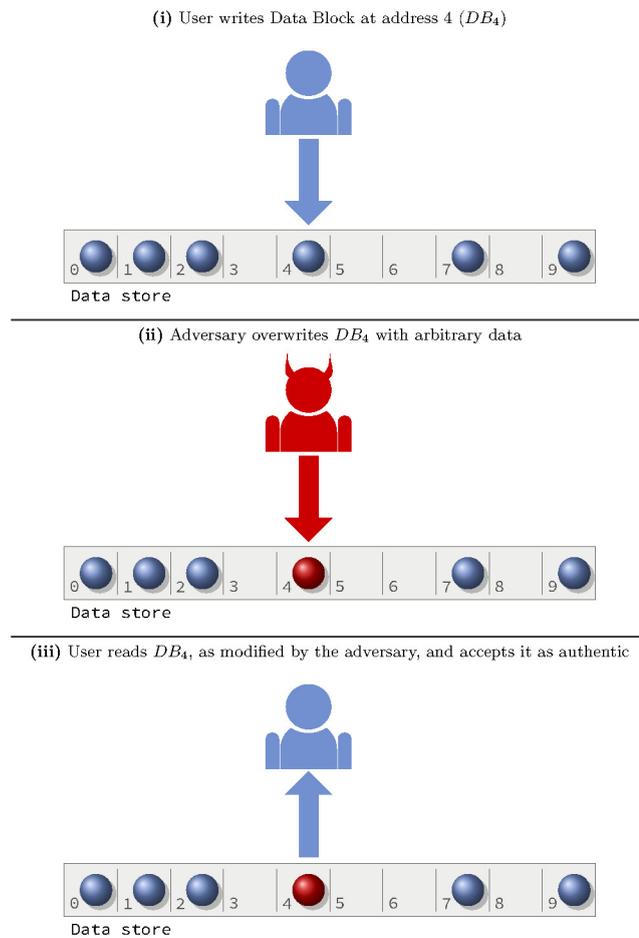


Figure 2.9: Spoofing attack on a Block Datastore

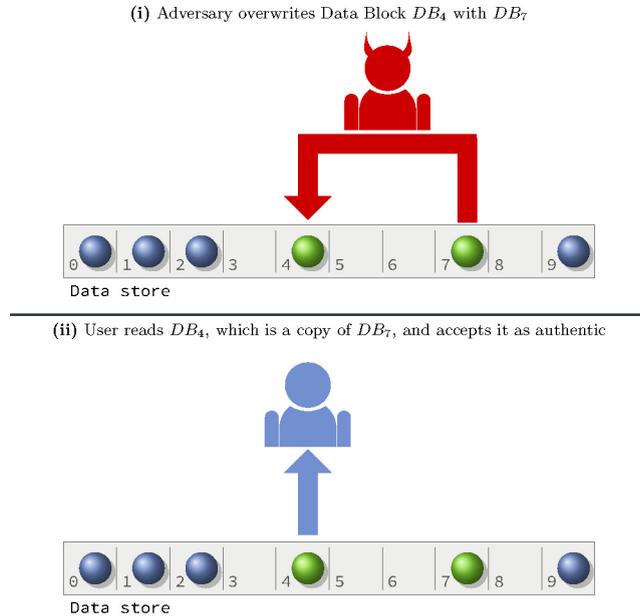


Figure 2.10: Splicing attack on a Block Datastore

time the authorized user reads the Data Block at address 4, she will read the outdated copy $DB_{4_{t_1}}$ and accept it as authentic.

Forking The adversary is able to present inconsistent copies of a Block Datastore to at least two different users. The adversary duplicates the state of the system, i.e., the Block Datastore, while the Block Datastore is in an authentic state. Then the adversary uses these different copies of the Block Datastore for different users. The adversary can then suppress updates by one user being visible to other users with different copies of the Block Datastore.

Figure 2.12 depicts an exemplary forking attack. In a first step the adversary creates two copies of an authentic Block Datastore: BDS_A and BDS_B . Next, when user Alice wants to use the Block Datastore, she actually interacts with BDS_A , her own copy of the data store. So, if Alice updated Data Block 4 (DB_4), she only updates BDS_A , whereas Bob's copy (BDS_B) remains untouched. Bob, who is under the misconception of sharing the same Block Datastore with Alice, later wants to read DB_4 . However, because the adversary presents Bob with BDS_B Bob reads an outdated version of DB_4 , while still accepting it as authentic.

An implementation of an Authentic Block Datastore has to mitigate the forking attack that arises from having multiple authorized users in the system. A Single-User Authentic Block Datastore only has a single authorized user and

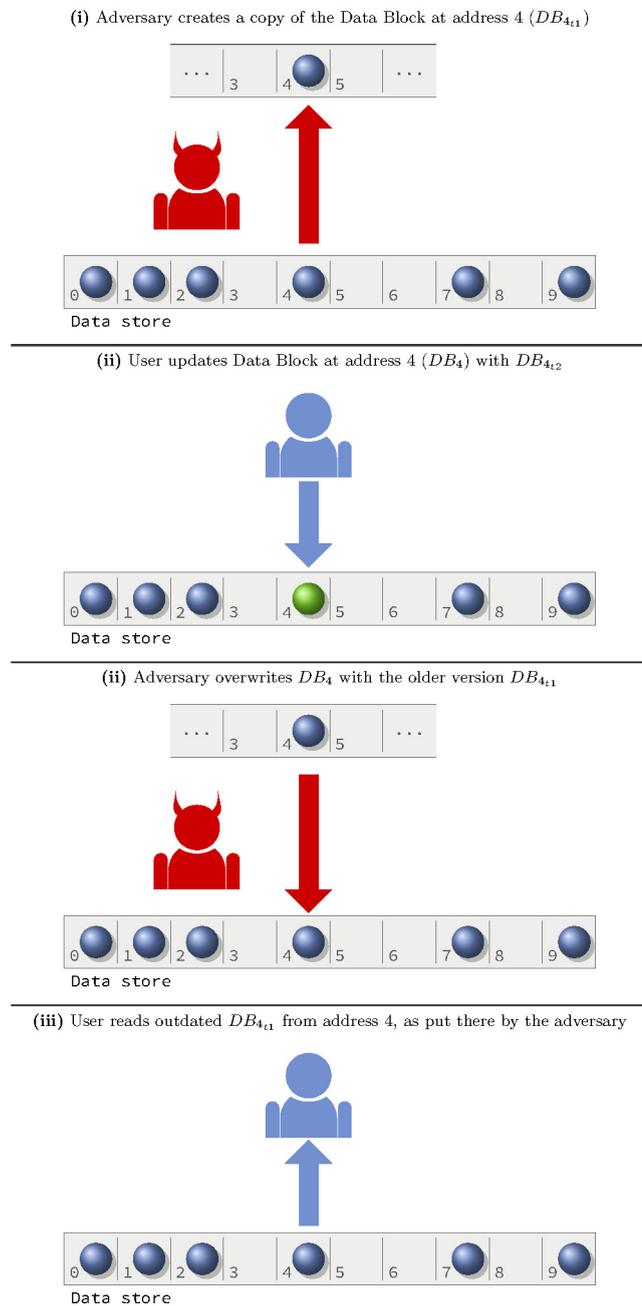


Figure 2.11: Replay attack on a Block Datastore

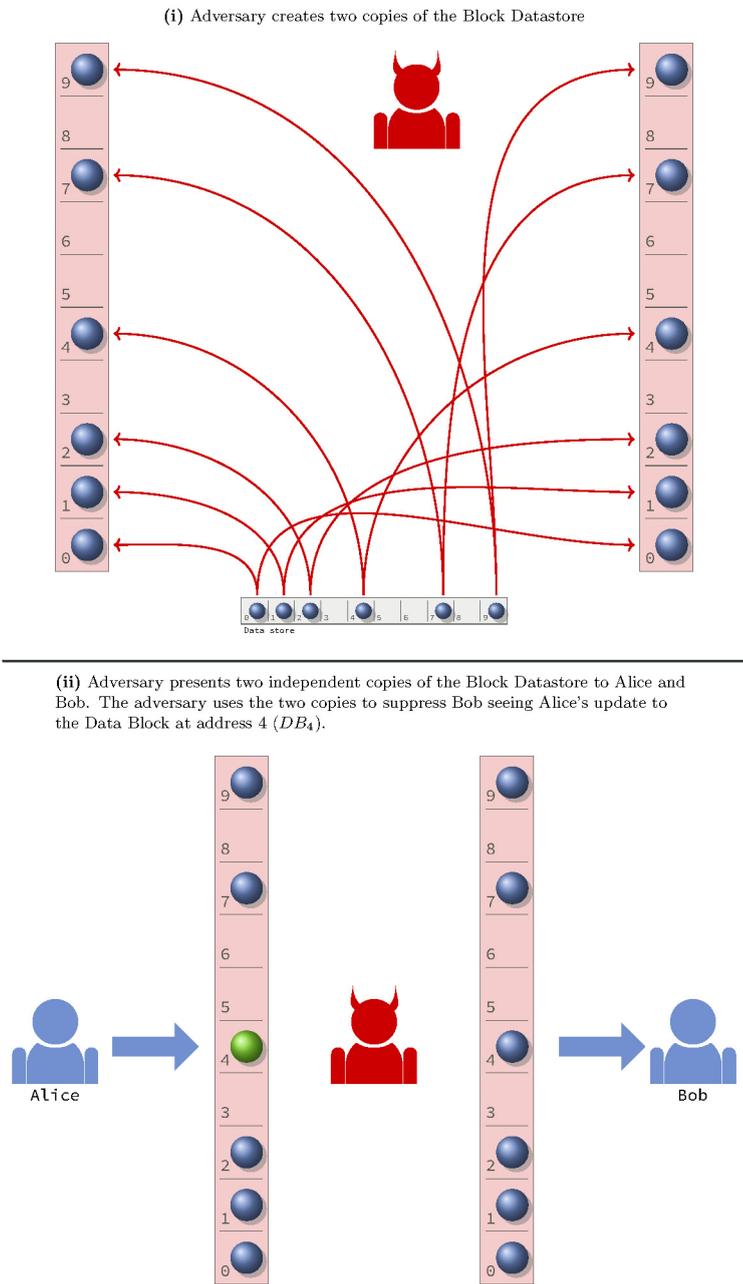


Figure 2.12: Forking attack on a Block Datastore

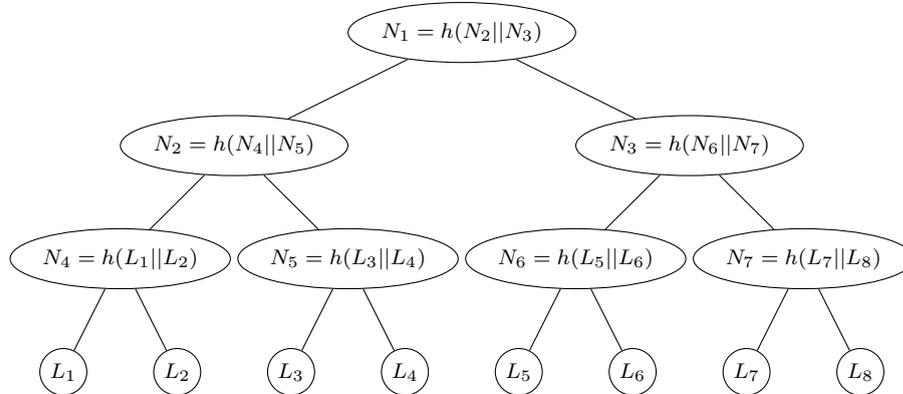


Figure 2.13: A perfect binary Merkle Tree. The inner nodes are denoted by N_k , where $0 < k \leq 7$, and the leaf nodes are denoted by L_i , where $0 < i \leq 8$. The inner nodes N_k of the tree are computed by applying a hash function h to the concatenation ($||$) of its immediate children.

can therefore disregard forking attacks.

2.12 Merkle Trees

2.12.1 Description

Merkle Trees [Mer80], or hash trees, were originally invented by Ralph Merkle as a digital signature scheme. Figure 2.13 depicts a perfect binary Merkle Tree. In a Merkle Tree each inner node corresponds to the hash of the concatenation of its child nodes. For example, in Figure 2.13, the inner node identified by the node index N_2 is computed by hashing the concatenation ($||$) of its two child nodes $N_2 = h(N_4||N_5)$. Due to this structure, and given that h is a cryptographically secure hash function, the root hash becomes a representative of the integrity of the overall tree, and by extension a representative to the data that is input on the leaf level. Put differently, if any of the input data on the leaf level, or any of the intermediate nodes change their value, the root hash value will also be different. This hash tree property is based on the collision resistance of a cryptographic hash function (see Section 2.9.1). Any alteration to the any number of hash tree nodes that keeps the root intact has been proven to yield an explicit collision for the underlying hash function [Cor05].

2.12.2 Merkle Trees and Data Integrity

In 1994, Blum et al. [BEG⁺94] proposed the use of Merkle Trees as a fast memory authenticity protection mechanism. The same idea can be adapted to Block Datastores and Block Datastore Authentication. Here, each Data Block of the Block Datastore is hashed and these hashes are used as leafs of a Merkle Tree,

and thus the Merkle Tree root hash becomes a representative of the data in the Block Datastore.

Similar to using a single hash function for data integrity, a Merkle Tree allows for a memory versus computation time trade-off. Instead of having to store the whole Block Datastore in an authentic memory to achieve Block Datastore Authentication, only the root hash needs to be stored in an authentic memory, such as an authentic Datastore. To verify Block Datastore Authentication one recomputes the Merkle Tree on the actual Block Datastore and compares the recomputed root hash with the stored one. The advantage of a Merkle Tree versus using a single hash to create a data representative is that at the expense of a little untrusted memory to store the internal nodes of the Merkle Tree, verification of the overall Block Datastore Authentication can be significantly sped up.

For example, to verify the integrity of a Data Block that was read from an untrusted source, one first recomputes the hash of the Data Block. Then one recomputes the hash values of all inner nodes lying on the path between the Data Block leaf node, and the root. Under the assumption that the used hash function is a cryptographically secure hash function, a modified Data Block block will lead to a different root hash, and thus the change becomes evident. On updating a Data Block, for example before sending it to an untrusted storage, the Merkle Tree must be updated accordingly.

2.13 Security Controllers

Security Controllers are discrete hardware modules that afford some protection against implementation attacks [MOP07]. Implementation attacks are attacks on the implementation of a security mechanism and not its underlying concept. For example, side-channel attacks use minute information leaks in cryptographic implementations to determine the cryptographic key. We use a Security Controller to implement the Secure Docking Module (SDM) introduced in Chapter 5. The following three paragraphs describing Security Controllers are taken almost verbatim from [LHW15].

A security micro-controller or Security Controller is a dedicated Integrated Circuit (IC) that provides a defined set of cryptographic operations. These cryptographic operations are carried out using credentials, typically cryptographic keys, persistently and securely stored inside the Security Controller. Thus a Security Controller protects the authenticity and confidentiality of cryptographic credentials in use and at rest. Security Controllers are designed to withstand two basic attack categories.

The first category, *local* attacks, involves an adversary physically attacking the Security Controller, either in an invasive or a non-invasive manner. Invasive attacks include, for example, bus-probing where an adversary probes a bus line to capture cryptographic key data. Semi-invasive attacks may for example use laser radiation to inject faults. Non-invasive attacks analyse side-channels such as computation time or power consumption to infer cryptographic credentials. A

Security Controller can provide tamper resistance against such attacks by using dual CPU concepts for integrity checks, encryption of bus communication, and a number of sensors to detect probing or manipulations [ABCS06, GR05].

The second category of attacks are *remote* attacks. Remote attacks are independent of the physical presence of an attacker and include protocol analysis and cryptanalysis. Protocol analysis aims at detecting flaws in the design of communication protocols with the Security Controller, while cryptanalysis tries to discover flaws in cryptographic primitives and schemes [ABCS06].

Finally, we discuss the Security Controller we use to implement the SDM (see Section 5.7). This text is a précis of the Security Controller description in this paper [HTP⁺12]. The Security Controller we use contains a dual CPU with on-chip ROM, RAM and Non-Volatile Random Access Memory (NVRAM). The dual CPU provides integrity checks and the Security Controller is further protected by full encryption of all data, including program code, while it is at rest and in use. Specifically, in this Security Controller decryption of data and code happens inside the CPU prior to execution and processing. In addition to these interesting security concepts the Security Controller sports a number of communication interfaces such as ISO7816, I2C, SPI, and USB, cryptographic accelerators for RSA, ECC, and AES, fast hashing for SHA-1 and SHA-256, and a true random number generator.

2.14 Trusted Platform Module

2.14.1 Introduction

The TPM is a core component for Intel Trusted eXecution Technology (TXT) and TXT in turn is at the heart of the acTvSM platform. The acTvSM platform underlies the Trusted Docking Station (TDS) introduced in Chapter 5. The Trusted Platform Module (TPM) is a general purpose computer component specified by the Trusted Computing Group⁶. There have been several iterations of the TPM specifications. The current TPM specification is for the TPM 2.0. However our work presented in this thesis uses the TPM version 1.2 [Tru07b]. The TPM 1.2 has been the dominant TPM standard for over a decade, and only recently, in 2014, it was superseded by TPM version 2.0.

A TPM 1.2 consists of a number of core components, such as an RSA engine, a key generator, a SHA-1 engine, a Random Number Generator, and volatile and non-volatile storage. The TPM uses these core components to implement high level security features such as cryptographically protected key storage (see Section 2.14.3).

The security features a TPM provides are similar in nature to what a smart card provides. However, where a smart card is usually bound to a user, the TPM is physically bound to a platform, that is, usually a general purpose computer. The fact that a TPM is physically bound to a host platform allows for a number

⁶<http://www.trustedcomputinggroup.org>

of interesting high-level security features, such as storing platform integrity measurements, sealing, secure key storage, and remote attestation. In the following we will detail those high-level security features of a TPM that are relevant to this thesis.

2.14.2 Storing Platform Integrity Measurements

A TPM can be used to keep track of its host platform software configuration measurements. In Section 2.15 we discuss how a platform can record a representation of the programs that have been loaded on it up to a certain point in time. We call this the platform software configuration. Without going into too much detail here, if the platform provides a storage for program measurements that cannot be tampered with after a measurement has been recorded, and if this platform can also ensure the integrity of the initial software configuration recording entity, then a track record of all programs loaded on a specific platform can be maintained (see Section 2.15.2 for details). This track record can then be used as evidence of the load-time integrity of the platform. The TPM provides a tamper-proof storage for program measurements with its Platform Configuration Registers (PCRs).

A platform's software configuration can be arbitrarily complex, consisting of arbitrary many individual component measurements. The TPM is a physical component with limited memory. Therefore, in general it is not able to hold a classical append-only log of all measurements. Here the authors of the TPM specification used the following strategy to limited the number of required PCRs. The TPM specification prohibits directly writing to a PCR. Instead a PCR can be extended. A PCR with index i in state t is extended with input x by setting

$$PCR_i^{t+1} = \text{SHA-1}(PCR_i^t || x).$$

Now a single PCR can hold an arbitrary amount of program measurements, while still guaranteeing the tamper-proof requirement for the program measurements. Note that the extend operation is the only way to alter the content of a PCR and that the initial value of all PCRs (PCR_i^0) are determined by the TPM specification.

Suppose an adversary wants to forge specific values in a set of PCRs. The legitimate PCR values are computed by loading a specific platform software configuration. Now, the adversary should not be able to achieve the same PCR values as the legitimate configuration, unless the same measurements are extended into the correct PCRs in the same order. The underlying assumption here is that the SHA-1 hash function is secure against 2nd pre-image attacks. Given a specific PCR value PCR_i^t , it will be computationally infeasible to find an input x' different from x , such that

$$PCR_i^{t+1} = \text{SHA-1}(PCR_i^t || x) = \text{SHA-1}(PCR_i^t || x').$$

2.14.3 Secure Key Storage

The TPM has an RSA engine for encrypting and decrypting data, as well as signing data. Also, the TPM can internally generate RSA keys. The TPM specification however only mandates TPM internal storage for two keys, the Endorsement Key and the Storage Root Key. Of those two the Storage Root Key is used to create a key hierarchy that enables cryptographically protected external storage of TPM created RSA keys. Any platform user that has access to the Storage Root Key usage secret that authorizes access to the Storage Root Key can create and use new keys in the TPM. To store these keys for later use, the TPM wraps, that is encrypts, these keys with the Storage Root Key and a usage secret supplied by the key creator. User created keys can again serve to wrap other user generated keys leading to a key hierarchy. A TPM supports a number of key types, such as the Attestation Identity Key type we will encounter later. Furthermore, TPM protected keys can have properties such as migratability, which specifies that a key can, under strictly limited conditions, be migrated into a different TPM. Not all key types can have all properties, for example, an Attestation Identity Key or the Storage Root Key can never have the migratable property.

2.14.4 Sealing

Another high-level feature of a TPM is that it can *seal* data, for example a symmetric key, to a platform by encrypting it with a non-migratable key. Data may be sealed to a specific set of PCR values. A TPM will only decrypt this data, if the PCRs hold the same values that were specified when the key was sealed. Thus, use of data can be restricted to a single PCR state of the TPM's host computer. In addition to only supporting sealing with non-migratable keys, the TPM also adds the so-called TPM proof to the sealed data blob. The TPM proof is TPM specific and it is created at the same time when a Storage Root Key is created. The TPM proof is added to prevent false key injection. False key injection is an attack where an adversary uses knowledge of the public Storage Root Key to try to inject a key into a TPM's key hierarchy. As a blanket protection mechanism, the authors of the TPM specification have added the TPM proof to all operations using a TPM key hierarchy key.

2.14.5 Remote Attestation

The TPM implements reporting of the current system software configuration and providing evidence of the integrity and authenticity of this measurement. This is called Remote Attestation. Remote Attestation is the process of reporting a subset of the PCRs to an interested party, while also providing sufficient evidence that this platform configuration is really maintained inside a TPM.

The TPM creates a proof for the integrity and authenticity of a set of PCRs by signing the PCR report together with a nonce supplied by the entity that is requesting the report. This process is called a Quote operation. One type of

TPM keys that can be used in a Quote operation are the Attestation Identity Keys. Attestation Identity Keys are non-migratable so they are always bound to a specific TPM. The nonce is essential to guarantee the freshness of the Quote operation. Presented with a Quote, the receiving entity requesting the Quote can decide whether it wants to interact with the quoting platform at all, and to what extent it wants to interact with it.

2.15 Intel Trusted eXecution Technology

Intel Trusted eXecution Technology (TXT) is at the heart of the acTvSM platform. The acTvSM platform underlies the Trusted Docking Station (TDS) introduced in Chapter 5. In this section we detail those aspects of Intel TXT relevant to this thesis. The primary aspects we are interested in are the TXT's ability to create isolated execution environments and providing a flexible way of measuring a platform's software configuration. TXT requires a Trusted Platform Module (TPM) version 1.2 (see Section 2.14) as part of the platform.

According to David Grawrock [Gra09] TXT is designed to use platform virtualization as isolation technique to separate different execution environments with different security policies. Here Intel[®] Virtual Machine eXtensions provide the basic support for virtualization in the CPU. These extensions can then be used by a Virtual Machine Monitor, or hypervisor, to create isolated partitions of the platform, where each partition is a virtual platform. The idea behind security by isolation is to execute software without it being hindered by other software on the same platform. For example, if one partition has been compromised by a computer virus, a different partition on the same physical platform should not be affected by this virus.

Next to enabling isolated partitions, TXT also aims to provide evidence that a platform is configured such that these isolated partitions are set up in accordance with a specific security policy. To create this evidence, TXT provides the facilities to measure those parts of the Trusted Computing Base that are implemented in software.

Definition 2.15. The Trusted Computing Base is all of the elements, technologies, and services that the user of a platform requires trust in [Gra09].

In addition to the ability to measure the software components of the Trusted Computing Base, TXT, or rather the platform's mandatory TPM can provide evidence of the authenticity of the measurements.

So far we have purposefully avoided detailing how to measure a platform's software components. We now discuss the approach proposed by TXT. The approach is called late launch and is based on measuring a platform's firm- and software components in a stepwise fashion as they get loaded in conjunction with using a Dynamic Root of Trust for Measurement.

Definition 2.16. A hardware or software mechanism that one implicitly trusts is a root of trust [Gra09].

We will first detail how the software measurement process works and then we describe the Dynamic Root of Trust for Measurement.

2.15.1 Software Integrity Measurement

The concept behind TXT's software measurement process is to establish the identity and thus the integrity of the software when it is loaded, but before it is executed. To establish a software's integrity, the software executables, its resources, configuration, etc., are hashed using a cryptographically secure hash function (see Section 2.9.1). The hash creates a representative of the software's integrity that is significantly smaller than the software itself, while relying on the collision resistance property of the hash function for uniqueness of the representative. This hash is then stored in a tamper-proof location. Only after the software measurement has been taken, and the measurement put into a protected memory, the software is actually executed. Because the hash of the software is in a tamper proof location that the software cannot overwrite, the software will not be able to hide its identity against anyone who can read the measurement track record. Here TXT mandates the use of a TPM for storing the measurement values in its Platform Configuration Registers (PCRs) (see Section 2.14.2). The hash changes when the software changes. Therefore the hash corroborates the integrity of the measured software component.

The software measurement process is transitive. For example, a platform's kernel gets loaded, measured, and then executed by the bootloader. Whereas the kernel then, e.g., loads, measures, and executes the platform initialization process. Thus a chain of trust from the first measuring entity to the lastly executed software is created.

Now the question arises, *quis custodiet ipsos custodes*⁷? Or rather when the measurement is done in software, and integrity of the whole process relies on the integrity of the software performing the measurement, how can we establish the integrity of the component that does the first measurement? This is where the late launch and the Dynamic Root of Trust for Measurement come into play. Note however, that in the end we have to trust the Dynamic Root of Trust for Measurement, as nobody watches that component.

The late launch is able to bring a platform in a well-defined state, where it can operate free of all outside interference and act as a Dynamic Root of Trust for Measurement to start building a chain of trust. The process of establishing the Dynamic Root of Trust for Measurement is impressively intricate, therefore we refer the esteemed reader to David Grawrock's book [Gra09] for the details. Nonetheless, we will briefly outline it here.

2.15.2 Late Launch

Conceptually the late launch starts a *secure boot* that ensures execution of a known-good Virtual Machine Monitor configuration. The secure boot is gov-

⁷Who will guard the guards themselves? In our case, who boots the bootloader?

erned by two policies, the Launch Control Policy, and the Verified Launch Policy. If at any point during the boot process either the Launch Control Policy or the Verified Launch Policy is violated or the system detects tampering, the boot process is immediately aborted. The system then enters TXT shutdown mode where it scrubs all potentially sensitive data from memory, before shutting down.

The late launch starts with the `GETSEC[SENTER]` CPU instruction to provide a well-defined system state. This CPU instruction leads to full platform lock down, where all but the initiating Central Processing Unit core are put to sleep. It then loads an Authenticated Code Module from the main memory into the CPU internal level 1 cache and locks down the chipset to prevent interference from any components on the system buses, including main memory, or platform extension cards. The one remaining active CPU then verifies the authenticity of the Authenticated Code Module. The Authenticated Code Module is digitally signed and the hash of the verification key is supplied by the platform's chipset. In addition to verifying the Authenticated Code Module's signature the CPU also measures the Authenticated Code Module.

After measuring the Authenticated Code Module, the CPU selectively rescinds the lock down on specific system components, most importantly it reactivates the TPM. Next to providing the PCR where the measurement of the Authenticated Code Module is stored, the TPM serves as access-controlled, autonomous storage for the Launch Control Policy and Verified Launch Policy, the two policies which are enforced by the late launch. The Launch Control Policy specifies the valid measurement of the secure boot loader to use, and the Verified Launch Policy specifies the measurement of the Virtual Machine Monitor to boot.

Now the CPU delegates control to the Authenticated Code Module that reactivates access to the main memory where it expects to find the secure boot loader. It then measures the secure boot loader, verifies it against the Launch Control Policy and stores its measurement in the TPM. If the secure boot loader measurement is different from the valid measurement specified by the Launch Control Policy, as with all errors the late launch will go into TXT shutdown mode.

After finishing its system setup tasks, loading, measuring and enforcing the Launch Control Policy on the secure boot loader the Authenticated Code Module hands over control to the secure boot loader. The secure boot loader further sets up the system, such that it can resume normal operation and loads, measures, and enforces the Verified Launch Policy on the Virtual Machine Monitor. The late launch terminates with delegating control the Virtual Machine Monitor.

2.15.3 Load-Time Integrity

We define load-time integrity as the security property established by performing the platform software measurement process described in Section 2.15.1. The platform software measurement process establishes what software is loaded on the platform. The counterpoint to load-time integrity is run-time integrity. Run-

time integrity concerns itself with the integrity of the software that is currently executed. These are two quite different concepts, as can be illustrated with the following simple example.

Consider a platform that provides load-time integrity. Suppose, this platform is compromised by an adversary. The adversary uses a security vulnerability to modify a component on the platform at runtime in a way that allows her to change the code of the component in memory, for example to enable remote code execution. A prominent example here is a buffer overflow with shell code injection. The platform is now compromised and its run-time integrity subverted. However, the load-time integrity of the platform is still unaltered.

Load-time integrity only creates evidence about a platform's software configuration by measuring at discrete points in time, that is, every time software is loaded. However, in general, we cannot make any assumptions on when a system will be compromised at run-time. Thus a time of check to time of use problem arises. Furthermore, load-time integrity only concerns itself with statements about software at rest, that its off-line, binary representation. As long as an adversary does not change the off-line software components a run-time attack might never be detected by measuring load-time integrity.

2.15.4 Security

To sum up David Grawrock's book on this issue [Gra09], and at the same time gravely oversimplify, the security objective of Intel TXT is to provide strong protection against software based attacks, where the user wants to protect her data. Furthermore, the goal of Intel TXT is:

... to force attackers to use hardware mechanisms to attack the platform. The Intel TXT design protects from attacks accessing memory, changing drivers, or changing the application. The attacker needs to use hardware access to gain access to protected memory [Gra09].

TXT even considers some hardware attacks and tries to provide mitigation mechanisms. For example, TXT provides limited mitigation against cold boot attacks. In a cold boot attack, an adversary literally freezes the main memory so that it retains its state, while it is put into a different computer, where the memory content is then read. As an other example, TXT also tries to mitigate against rogue CPU cores not joining in or prematurely leaving the TXT protected execution environment. However these mitigation mechanisms rely on the integrity of the PC chipset to be intact and not compromised.

However, Rutkowska et al. have investigated TXT and identified issues in the implementation [WRT09, WR09, WR11] allowing software attacks. Furthermore, one of the cornerstones of TXT's security is the TPM and the physical bond of the TPM with its host platform. Johannes Winter reported several very successful attacks on this bond, in the form of attacks on the Low Pin Count bus connecting the TPM to the host platform [WD13, Win14].

An Anecdote

The following is purely informal and not part of this thesis' canon, but some stories simply have to be told.

Quite a few years back two young students, one of the PhD ilk and the other crusading towards his master had a lively discussion about the security afforded by TXT. Lets call the PhD student SubjectD and the master student SubjectJ. SubjectD had just read David's book [Gra09], because he wanted to hold a lecture on the topic of TXT. SubjectJ held strong doubts about TXT's security against hardware attacks. SubjectD opposed that TXT did not want to protect against sophisticated hardware attacks, only simple ones, as according to the book [Gra09, p. 132]:

Intel TXT is designed to provide protection from simple hardware attacks. What is the definition of a simple hardware attack? An exploit based on turning off the power and removing the battery is a simple hardware attack. Going to the local electronic store, purchasing twenty dollars worth of parts, putting the parts together and defeating the Intel TXT protections is a simple hardware attack. The Intel TXT objective is to mitigate simple hardware attacks.

SubjectJ simply accepted the challenge, and while intermediate attacks needed quite a lot of time and sophisticated hardware, the final result was an attack subverting Intel TXT, performable by everyone who can read a schematic and open a computer's case. The attack only requires a piece of wire [WD13, Win14]. Even without inflation that is still well below twenty dollars. And the morale? Never underestimate the power of the motivated student, or alternatively, do not taunt Johannes, for he is subtle and quick to being motivated.

2.16 The acTvSM Platform

The acTvSM Platform [Pir15] uses Intel Trusted eXecution Technology (TXT) to provide isolated execution environments, or partitions, for software components with different security policies. It relies on TXT's late launch to securely boot (see Section 2.15.2) a fully measured Virtual Machine Monitor. The acTvSM Virtual Machine Monitor is a Linux system with Kernel-based Virtual Machine. Kernel-based Virtual Machine is a full virtualization solution that transmogrifies the Linux kernel into a Virtual Machine Monitor on platforms that support the Intel or AMD virtualization extensions. In addition to the Kernel-based Virtual Machine extensions in the Linux kernel, the acTvSM base system also provides the facilities to manage installed Virtual Machines. Each installed Virtual Machine, when started, is first measured into the Trusted Platform Module (TPM) (see Section 2.15.2) and then executed in its own partition, isolated against other partitions and the acTvSM base system. At rest, the individual Virtual Machines are cryptographically protected and the cryptographic key is sealed (see Section 2.14.4) to the platform.

One important feature for the Trusted Docking Station (TDS) introduced in Chapter 5 is the acTvSM's platform ability to patch the TPM through to one of the Virtual Machine it hosts. After booting up the base system, the acTvSM platform can be configured to relinquish control of the TPM and hand control over to exactly one of its Virtual Machines on boot of this Virtual Machine. After this TPM delegation the base system cannot measure any other Virtual Machine before starting them, thus effectively preventing starting further Virtual Machines.

As the acTvSM platform is based on Intel TXT it inherits TXT's security properties, as discussed in Section 2.15.4.

One of the key achievements of the acTvSM platform is that the platform integrity measurements become predictable so the Platform Configuration Register (PCR) values can be calculated *a priori* and data can be sealed to future trusted states after a planned configuration update.

A different point of note is the acTvSM platform's well-structured set of file systems. For instance, measurements of the base system are taken over its whole file system. To achieve measurement consistency on every boot the acTvSM platform uses a read-only file system that is comparable to a Linux Live-CD. Similar again to a Linux live system an in-memory file system is merged on top of the read only file system to form the runtime file system.

The Virtual Machines that house services and applications are stored in cryptographically protected logical volumes. Each volume has a set of key slots that allows assigning different access keys that are sealed to different trusted platform states.

2.17 ARM TrustZone

In Chapter 6 we introduce the Secure Block Device, a cryptographically secured Single-User Authentic Block Datastore. We evaluate the Secure Block Device in the context of ANDIX OS (see Section 2.18). ANDIX OS uses the ARM TrustZone security extensions. Hence we give a concise introduction to ARM TrustZone here.

The ARM TrustZone security extensions [WFM⁺07] are a relatively new addition to the field of hardware extensions for security by isolation. However, today many mobile phones and ARM application processors provide at least minimal support for ARM TrustZone. With ARM TrustZone a system can be split into two partitions. Conceptually this is a red/green system [Lam09], similar to NGSCB [PCEM04]. In ARM parlance the partitions are called the *secure world* and the *normal world*.

The secure world has higher privileges and controls the TrustZone security extensions. The secure world can isolate both memory and I/O against access from the normal world. To this end ARM TrustZone introduces a new secure state to the processor. Next to logically separating all major components inside the CPU, this state is also signalled via the system bus to all peripheral devices. Thus the peripherals can base access control decisions on the current state of

the system.

The secure world can logically isolate parts of the physical memory against access by the normal world. The access control is done by a TrustZone-aware memory controller that limits access to certain memory partitions based on the current system state. Thus only the secure world is able to access these isolated memory partitions. Depending on the system, the memory partitioning scheme can be fixed, or programmable at runtime.

For peripheral isolation, the secure world can force specific signals like hardware interrupts and exceptions to always trap into the secure world. A programmable TrustZone-aware interrupt controller regulates access to these signals based on a policy configured by the secure world. In addition ARM TrustZone specifies mechanisms to block normal world access to certain peripheral devices, where the access control policy is again configurable by the secure world.

The ARM TrustZone security extensions allow system partitioning schemes where even the less privileged normal world is able to run a rich operating system (OS), such as Linux, or Android. Here, the rich OS can be oblivious of a system running in the secure world, although a rich OS can greatly benefit from services provided in an isolated environment.

The GlobalPlatform association⁸ has established a set of standards for creating a Trusted Execution Environment that can be implemented using the ARM TrustZone security extensions. These public standards specify the software interface between the untrusted system partition, and the Trusted Execution Environment [Glo10], and the interface between the Trusted Execution Environment and its Trusted Applications [Glo11]. These standards propose a model where the Trusted Execution Environment harbors a set of Trusted Applications that can be used by applications in the normal world to perform security sensitive operations. A Trusted Execution Environment can use TrustZone to isolate itself from the normal world and also takes care of isolating the Trusted Applications from each other.

Furthermore, the Trusted Execution Environment can also take control of I/O devices such as the screen and the keyboard and isolate them against the normal world [Glo13]. The goal for such a trusted I/O path is to be both confidential and authentic. For example, this feature can be used for direct password input, without the normal world being able to eavesdrop, or change the password [LC14].

Finally, the Trusted Execution Environment and its Trusted Applications require platform resources such as memory. All resources permanently assigned to the Trusted Execution Environment are missing from the normal world. Therefore, a Trusted Execution Environment can impose resource restrictions such as limiting the memory, or off-line storage available to a Trusted Application.

⁸<http://www.globalplatform.org/aboutus.asp>

2.18 ANDIX OS

In Chapter 6 we introduce the Secure Block Device. The Secure Block Device is a software library that implements a Single-User Authentic Block Datastore (see Section 2.10). It relies on cryptography to achieve its security guarantees. We evaluate the Secure Block Device in the context of a Trusted Application (see Section 2.17). ANDIX OS [FAWH15, Fit14] is the OS that provides the Trusted Execution Environment in which we conduct our experiments.

ANDIX OS is a non-preemptive multitasking operating system (OS) developed by Fitzek et al. ANDIX OS is specifically designed to run and control the secure world partition of an ARM TrustZone platform. ANDIX OS can host a common off-the-shelf Linux as guest OS in the normal world and it provides a Trusted Execution Environment for multiple Trusted Applications based on the corresponding GlobalPlatform standards [Glo10, Glo11] (see Section 2.17).

ANDIX OS is designed for two different execution environments. First it runs on the low cost iMX53 Quick Start Board, a development board based on the ARM Cortex-A8 iMX53 processor by Freescale that provides full access to the processors security features. Second, ANDIX OS also runs on a modified version of the QEMU machine emulator⁹. The necessary extensions to QEMU were first introduced by Winter et al. [WWPT11]. ANDIX OS is conceived as a research OS and is available under a dual licensing scheme, where the GNU General Public License (GPL), Version 2.0 applies for research purposes. It can be freely downloaded¹⁰.

The objectives for ANDIX OS are a minimal Trusted Computing Base, security by isolation, and compatibility. ANDIX OS minimizes the Trusted Computing Base to reduce the risk of security flaws in the code. ANDIX OS limits the Trusted Computing Base to the hardware platform, the bootloader, and the secure world ANDIX kernel. The secure world ANDIX kernel uses the hardware platform's security features to implement memory and I/O isolation. The goal of this security by isolation mechanism is to protect the secure world against normal world software intrusions, even if the normal world is completely compromised. To achieve this goal ANDIX OS uses the ARM TrustZone security extensions to isolate the security kernel and its Trusted Execution Environment from the normal world. On the iMX53 Quick Start Board, ANDIX OS configures the processor's Central Security Unit, memory controllers, and Direct Memory Access controllers to protect the secure world memory against the normal world. ANDIX OS builds on the fact that TrustZone compatible Memory Management Units provide two separate sets of translation tables, one for the normal world and one for the secure world, to realize OS-level process isolation of the ANDIX security kernel against the Trusted Applications and the Trusted Applications against each other. Simultaneously, the Linux guest OS can use the independent normal world Memory Management Unit translation table to implement its own process and kernel isolation. Furthermore, ANDIX OS also

⁹http://wiki.qemu.org/Main_Page

¹⁰<http://andix.iaik.tugraz.at/>

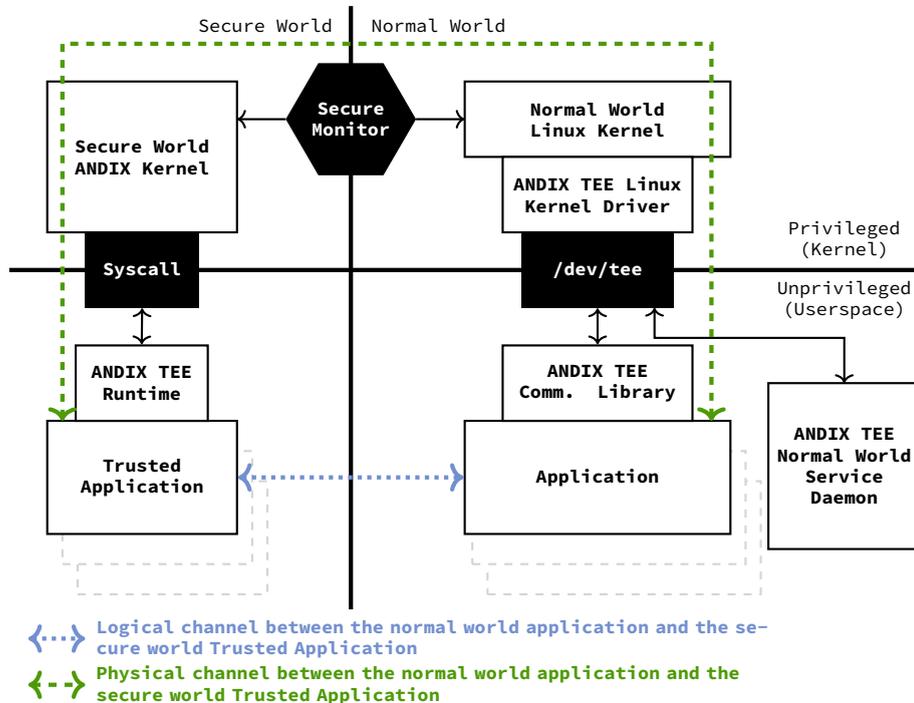


Figure 2.14: Architecture of the ANDIX OS

supports trusted I/O paths, where certain I/O devices are only available to the secure world. ANDIX OS isolates the peripheral devices by configuring a TrustZone enabled interrupt controller to send certain interrupts only to the secure world. Finally, ANDIX OS implements a subset of the GlobalPlatform TEE internal [Glo10] and client application programming interfaces [Glo11] with the aim of being source code compatible with alternate implementations of these standards. In addition, ANDIX OS exposes an alternative programming interface based on the standard C library (GNU Newlib¹¹).

Figure 2.14 illustrates the architecture of the ANDIX OS. The left side shows the secure world, whereas the normal world is to the right. The top part of the figure depicts the privileged kernel-mode, and the bottom the unprivileged user-mode. The secure world privileged mode (top left) houses the ANDIX kernel. The ANDIX kernel is the core of ANDIX OS. It is a small, monolithic kernel that sets up TrustZone, provides process isolation, scheduling, and communication facilities. The secure world unprivileged section is where the Trusted Applications reside. The normal world side of the figure reflects ANDIX OS hosting a Linux guest OS, where the Linux kernel operates in kernel-mode (privileged, top right) and the applications run as process in the unprivileged

¹¹<https://www.sourceware.org/newlib/>

user-mode (bottom right).

2.18.1 Inter-world Communication

ANDIX OS implements inter-world communication to enable normal world applications to communicate with and benefit from Trusted Applications running in the secure world. Figure 2.14 depicts both the [logical communication channel](#) and the [physical data path](#) between a normal world application and a Trusted Application in the secure world. ANDIX OS implements a Remote Procedure Call interface for inter-world communication. This interface uses well-defined control structures in shared memory for exchanging parameters and results. Remote Procedure Calls are always initiated in the normal world. Typically, a normal world application uses the `ANDIX TEE communication library` to start a call. The `ANDIX TEE communication library` uses the `/dev/tee` device provided by the `ANDIX TEE Linux Kernel Driver` to dispatch the call to the `Normal World Linux Kernel`. In response, the Linux kernel invokes a secure monitor call processor instruction that switches the processor into the *secure monitor mode*. The secure monitor call instruction can only be called when the processor is in a privileged mode, and the secure monitor mode is a special processor mode to enable controlled world switches and data sharing between worlds. From the secure monitor mode the ANDIX kernel forwards the call to the destination Trusted Application. The destination Trusted Application uses ANDIX OS's synchronous first-in first-out message queuing interface to wait for incoming requests. After having been initialized, the Trusted Application invokes the `Syscall` interface and the `Syscall` interface then blocks the Trusted Application until a call to this particular Trusted Application arrives. When a call arrives, the Trusted Application is woken up. It then processes the call and after completing it notifies the ANDIX kernel via the `ANDIX TEE Runtime`. In addition to the completion notification, the Trusted Application can send a response. The ANDIX kernel can then complete the secure monitor call handler and return to the normal world.

2.18.2 Root of Trust for Storage

ANDIX OS provides a symmetric encryption key as a *root of trust for storage*. One way of providing a secure root of trust for storage is to incorporate a Security Controller (see Section 2.13) or Trusted Platform Module (TPM) (see Section 2.14) into the hardware platform and use these components to securely store a root key. However, this approach is not portable across platforms that have no such facility.

Therefore, ANDIX OS implements this feature by not storing the root key at all. Instead, at boot ANDIX OS asks for a password over a trusted I/O path. This password is then converted into a key using the PBKDF2 key derivation function. Note that ANDIX OS can be easily extended to use a platform specific hardware based mechanisms, such as a TPM, as root of trust.

2.18.3 Sharing Resources with the Normal World

ANDIX OS was developed with a minimal Trusted Computing Base in mind and it delegates managing many resources, such as storage devices and network interfaces, to the normal world OS. Sharing resources between the secure world and the normal world is delicate with serious security and implementation ramifications to consider. ANDIX OS' strategy is to reuse existing normal world resources, such as complex device drivers, file-system drivers, and network stacks as much as possible. In this way ANDIX OS elegantly avoids the need for separate storage hardware or network interfaces for the secure world, and issues such as sharing SD card driver state between Linux in the normal world and its own kernel driver in the secure world. The downside is that it becomes impossible to guarantee that devices are always available to the secure world. For example, the normal world OS can simply ignore block read and write requests. We study the security ramifications of this approach specifically for data storage in Chapter 6.

ANDIX OS uses a simple, but effective mechanism to let Trusted Applications share resources with the normal world. Trusted Application operations usually acquire normal world resources as part of a normal world Remote Procedure Call request. Thus, ANDIX OS translates a Trusted Application's request for a specific resource into an Remote Procedure Call. This request is then dispatched to the `ANDIX TEE Normal World Service Daemon` (see Figure 2.14) which handles the request. The `ANDIX TEE Normal World Service Daemon` is a Linux background process that uses `/dev/tee` to wait for incoming resource requests from a Trusted Application. Once such a request is received it reads or writes to the specified resource and sends the result of this operation back to the waiting Trusted Application.

Persistent storage in ANDIX OS is shared with the normal world. Specifically, ANDIX OS' storage interface in the `ANDIX TEE Runtime` allows Trusted Applications to open files that are then mapped into a special container in the normal world by the `ANDIX TEE Normal World Service Daemon`. The Trusted Application can then read and write data from these files through the mechanism outlined above. Thus the ANDIX OS kernel dispenses with in-kernel support for file systems and storage device drivers.

3

Related Work

3.1 Threat Modeling

3.1.1 STRIDE

As part of the DeSecA project financed by Microsoft several different academic teams published a series of papers [DJPJ04b, GE04, BBF⁺04, Cha04, CWS⁺04] that discuss threat models for different aspects of the generic web application architecture presented in [DJPJ04a]. All five papers present case studies where the authors use STRIDE to enumerate and classify the threats they identified for a specific aspect of the overall architecture. In general the authors of these papers enumerate the threats and give guidance on how to mitigate them. They do not present any observation on how well STRIDE is suited for the threat modeling they undertake. Grimm and Eichstädt map the STRIDE threat categories to the confidentiality, integrity, availability, authenticity, and accountability security requirements. De Cock et al., the team that analyses the use of security tokens in web applications, has the most tenuous connection to STRIDE, as they do not even mention it in their paper.

In this thesis, one of our contributions is a case study for disaster response security where we create threat models of several aspects of disaster response using Data Flow Diagrams as models in conjunction with the STRIDE-per-interaction threat enumeration technique as implemented in the Microsoft Threat Modeling Tool 2016. We compare our results for one of our models, the agent platform threat model, with the comprehensive Mobile Agent System security literature.

Möckel and Abdallah [MA10] present another case study where they use

Declaration of Sources

This section is based on and reuses material from the following sources, previously published by the author:

- [HT09] Daniel M. Hein and Ronald Toegl. An autonomous attestation token to secure mobile agents in disaster response. In Andreas U. Schmidt and Shiguo Lian, editors, *Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec 2009, Turin, Italy, June 3–5, 2009, Revised Selected Papers*, volume 17 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 46–57. Springer, 2009.
- [HTK10] Daniel M. Hein, Ronald Toegl, and Stefan Kraxberger. An autonomous attestation token to secure mobile agents in disaster response. *Security and Communication Networks*, 3(5):421–438, 2010.
- [HTP⁺12] Daniel M. Hein, Ronald Toegl, Martin Pirker, Emil Gattal, Zoltán Balogh, Hans Brandl, and Ladislav Hluchý. Securing mobile agents for crisis management support. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing, STC '12*, pages 85–90, New York, NY, USA, 2012. ACM.
- [HWF15] Daniel M. Hein, Johannes Winter, and Andreas Fitzek. Secure block device – secure, flexible, and efficient data storage for ARM TrustZone systems. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20–22, 2015, Volume 1*, pages 222–229, 2015.

References to these sources are not always made explicit. The related work concerning using Mobile Agent Systems for disaster response and their security is adapted from [HT09, HTK10, HTP⁺12] and the related work on secure data storage is adapted from [HWF15]. The related work for our disaster response threat modeling efforts however is original to this thesis.

the Microsoft Security Development Lifecycle threat modeling tool to model the architectural designs of an e-banking application. The Microsoft Security Development Lifecycle threat modeling tool is an ancestor of the Microsoft Threat Modeling Tool family, as discussed in Section 2.8.4. Möckel and Abdallah illustrate and discuss the used threat modeling process and also contrast it to a second tool, the Threat Analysis & Modeling tool, also by Microsoft. They conclude that although threat modeling tools “are a valuable help, they require sensible input and interpretation at a later stage.”. They also conclude that threat modeling is complex for non-trivial systems. They claim that non-experts might prefer Threat Analysis & Modeling, as it provides more guidance, whereas software developers might prefer the Security Development Lifecycle threat modeling tool. Thus they propose to use a combination of both tools to satisfy different target groups. They also note that threat modeling needs to be acted upon to be useful to application security and conclude that threat modeling requires further study.

Based on our observations of our threat modeling efforts we agree with Möckel and Abdallah that threat modeling for all but the most trivial systems is complex and that the tool output requires interpretation. Also, as software developers we felt a certain affinity to the Microsoft Threat Modeling Tool 2016 as understanding systems by the data that is processed by the system comes naturally to us, which somewhat supports their conjecture that software developers might prefer the Security Development Lifecycle threat modeling tool. We did not contrast our findings to a second tool, however we compared our threats with results from Mobile Agent System security research.

We have found two independent proposals to use different methods of system modeling than Data Flow Diagrams. Johnstone [Joh10] remodels a well known threat modeling case study based on an Internet pet shop with Unified Modeling Language (UML) activity diagrams instead of Data Flow Diagrams. Johnstone then contrasts his results with the results of the Data Flow Diagram model. Johnstone notes that activity diagrams have more expressive power than Data Flow Diagrams and that there was no loss of precision compared to using Data Flow Diagrams in his case-study.

Schaad and Borozdin [SB12] have evaluated using the STRIDE threat enumeration technique in conjunction with software architecture diagrams. Software architecture diagrams are artefacts created early in the software development process. They have developed a tool for automated threat analysis of augmented software architecture diagrams and validated it by a) demonstrating it to architects at SAP and b) by evaluating the effectiveness of the tool with a specialized team of security auditors using it for multiple threat assessment sessions. Schaad and Borozdin state that the demonstration to the architects provided positive feedback and more automated analysis and countermeasure guidance was requested. In the three trial runs with the security auditor team, “roughly 20-25% of all potential threats were identified as applicable threats requiring follow-up.” Schaad and Borozdin noted that initial results need to be investigated further. The security auditors claimed the tool “to be extremely ef-

ficient in their work as it saves a lot of time by firstly supporting question based assessment and secondly by tight coupling with an issue tracker and Outlook.”

As opposed to Johnstone and Schaad and Borozdin we did not change the Microsoft Security Development Lifecycle’s threat modeling. For our case study, we accepted it as provided by the Microsoft Threat Modeling Tool 2016. We do note the usefulness of threat modelling in an early stage of software development and on an abstract level, when modelling disaster response processes.

Scandariato et al. [SWJ15] have assessed Microsoft’s STRIDE by performing a descriptive study involving 57 master students over three years. Their contribution is threefold. First, their study assesses STRIDE’s productivity, as it measures how many valid threats per hours are produced on average by the analysis. Second, their study evaluates the correctness of the analysis by investigating the average number of false positives. Finally, it determines the completeness of the analysis results by measuring the average number of false negatives. Scandariato et al. have conducted their study by letting students analyse a digital publishing system using Data Flow Diagrams for modeling and STRIDE-per-element in conjunction with the threat tree templates detailed by Howard and Lipner [HL06]. Scandariato et al. summarize the conclusion of their study as STRIDE being “not difficult to learn and execute, although it is relatively time consuming. Further, many threats go undetected during the analysis.”

In detail their conclusions are:

1. The STRIDE technique is not perceived as difficult but, with an average productivity of 1.8 threats per hour at best, the time cost is relatively large.
2. The average number of incorrect threats is low and corresponds to the 19-24% of the total amount of produced threats.
3. The average number of overlooked threats is very high and corresponds to the 64-69% of the total amount of threats.

Williams and Yuan [WY15] have evaluated the effectiveness of the Microsoft Threat Modeling Tool 2014 by comparing it with manual threat elicitation in a descriptive study. Their study was conducted with 20 graduate level students who were tasked to perform manual threat modeling on a mock online shopping platform. After the manual threat modeling the students had to threat model the same online shopping platform using the Microsoft Threat Modeling Tool 2014. Williams and Yuan’s observations are that students have difficulties correctly modeling the online shopping platform using Data Flow Diagrams. They also note that the threat modeling results of their students have improved from the manual assignment to the Microsoft Threat Modeling Tool 2014 assignment.

The objectives of our threat modeling efforts were to gain insight into the threats to disaster response processes and to establish threats to agent platforms in the Secure Agent Infrastructure Mobile Agent System in order to mitigate them. Also we have compared the threats to agent platforms we elicited with the literature on Mobile Agent System security to gauge the quality of our threat

list. As such we have conducted a case study and therefore we can only observe our results without laying claim to any generality. However, our agent platform model did elicit a significant number of threats known to literature in addition to a number of undocumented threats. Therefore, we find the high average number of overlooked threats in the study by Scandariato et al. interesting, and believe that this dichotomy requires further study. Furthermore, our personal observation on using STRIDE-per-interaction through the Microsoft Threat Modeling Tool 2016 tool is that, although it is simple to use on the face of it, understanding the intricacies of modeling and interpreting the resulting threats does require experience.

3.1.2 Other Threat Elicitation Methodologies

In this subsection we discuss alternative threat elicitation methodologies and argue why we preferred using the Microsoft Security Development Lifecycle approach through the Microsoft Threat Modeling Tool 2016. For a more comprehensive discussion of threat elicitation techniques we refer the reader to Tøndel et al. [TJM08] and Hussain et al. [HKA⁺14].

Misuse Cases, Abuse Cases and Abuser Stories

Misuse cases were developed by Sindre and Opdahl [SO05]. Alexander gives a concise introduction of misuse cases [Ale03]. Misuse cases are the negative form of use cases. System engineers can use misuse cases to document and analyze scenarios where a hostile actor threatens a system, from the point of view of such a hostile actor. According to Alexander misuse cases are identified by brainstorming.

Misuse cases are related to abuse cases [MF99]. McDermott and Fox define an abuse case “as a specification of a type of complete interaction between a system and one or more actors, where the results of the interaction are harmful to the system, one of the actors, or one of the stakeholders in the system.” McDermott and Fox use Unified Modeling Language (UML) use case diagrams to model abuse cases. Sindre and Opdahl [SO05] discuss the differences between misuse cases and abuse cases. Abuse cases were refined into abuser stories for agile development [Pee05], and Boström et al. demonstrate how to use them in Extreme Programming.

We have decided to use the Microsoft Security Development Lifecycle approach to threat elicitation because threats are elicited from a model of the system and the template based threat enumeration gives more guidance than free form brainstorming.

A Framework For Security Requirements Representation and Analysis

Haley et al. present a framework for security requirements elicitation and analysis [HLMN08]. In their framework the analyst constructs a context for a system using a problem-oriented notation and uses this context to represent security

requirements as constraints. The analyst then proceeds to validate the security requirements by developing explicit satisfaction arguments. These satisfaction arguments consist of two parts. The first is a formal outer argument that the system meets its security requirements. The second is a structured informal argument supporting the assumptions in the first argument. They evaluate their framework using a case study involving air traffic control technology. In their case study they use propositional logic to formalize the outer satisfaction argument.

The framework presented by Haley et al. provides a strong formal approach to security requirements elicitation but for now it lacks adequate tool support. Therefore, we decided to use the less formal approach provided by the Microsoft Security Development Lifecycle threat elicitation technique through the Microsoft Threat Modeling Tool 2016.

Attack Trees

“Attack trees represent attacks against a system in a tree structure. The root node represents the attack’s goal, and leaf nodes represent different ways of achieving that goal.” [Sch99]

We are interested in complementing our threat modeling efforts in Chapter 4 with threat trees (see Section 4.7.2). Threat trees add mitigated conditions to attack trees [SS04].

3.2 Mobile Agents and Disaster Response

Schurr et al. [SMT⁺05] introduce a Mobile Agent System that helps coordinating fire fights by facilitating vehicle management. This includes planning of routes, resource allocation and even deciding which fire to fight. They combine the agent system with sophisticated visualization technology to enable informed decisions by human personnel in a different location. In [SPA⁺06], Sadik et al. propose an Earthquake Management System, where Mobile Agents facilitate information retrieval and distribution through all stages of an earthquake scenario. According to Scerri et al. [SPJ⁺03] the use of Mobile Agents to coordinate heterogeneous teams of robots, agents, and humans provides the safest and most effective means for quick disaster response. A somewhat related topic is the use of Multi-Agent System in disaster response simulation [BMMV06]. This application uses agents to model disaster response personnel in a disaster simulation. Cohen et al. [CGHH89] have introduced the PHOENIX system for using planning agents in the domain of simulated forest fire management. Honda and Djordjevich [HCD08] introduce Mobile-FIRST a Mobile Agent based first responders training system. They present a case study where they have integrated their Mobile Agent based training system into a first responder training video game. In the ALADDIN project [RRM⁺08] Ramchurn et al. work on using Mobile Agents to coordinate first responders in disaster response.

3.2.1 Mobile Agent System Security

According to Jansen [Jan00], the Mobile Agent distributed computing paradigm suffers from a number of complex threats that go beyond those of the classic client-server architecture. We have extensively discussed Mobile Agent System security and corresponding threats in Section 2.4.

Wilhelm et al. [WSB99] propose a specialized, tamper-proof hardware module that provides the so-called trusted processing environment for an agent execution environment. The idea is that if a client trusts the trusted processing environment manufacturer, the trust could automatically be extended to a host with such an environment. We cover our work on the Trusted Docking Station (TDS) and the Secure Docking Module (SDM) in Chapter 5 and although it is similar in concept the TDS and the SDM are based on commercial off-the-shelf hardware products and dispense with the need for a specially developed hardware module.

Wu et al. and Shen and Wu [WSZ06, SW08] have considered using the Trusted Platform Module (TPM) for increasing the security of Mobile Agent Systems. Specifically, they propose an architecture for a system that uses remote attestation as evidence of the load-time integrity of a remote platform. However they present no implementation or evaluation of the system they propose. In contrast, our TDS and SDM based security solution uses local attestation and includes an evaluation of a concrete implementation.

In 2007, Balfe and Gallery [BG07] investigated how Trusted Computing in general, and the TPM especially, can enhance the security of Mobile Agent Systems. For their analysis they assumed the ubiquitous presence of agent platforms that gather evidence of their load-time integrity. Furthermore, they assume that these agent platforms can attest their load-time integrity to other platforms. Based on these assumptions they investigate four scenarios for Mobile Agent migration supported by Trusted Computing primitives. Gallery et al. [GNV09] improve one of the secure agent transfer protocols introduced in [BG07] with property-based configuration information about a platform's software. Gallery et al. operate under similar assumptions as Balfe and Gallery. Our work on the TDS and the SDM is focused on helping establishing the assumptions made by Balfe and Gallery and Gallery et al. Specifically, we concentrate on the creation of ubiquitously available platforms that can establish and attest their load-time integrity.

Pridgen and Julien [PJ06] introduce SMASH, a trust enhanced mobile agent system using SELinux and Trusted Computing mechanisms for security. SMASH is designed as an open and secure platform supporting multi-hop mobility. The Secure Agent Infrastructure is a closed system supporting single-hop mobility. Thus SMASH introduced additional complexity unneeded in the Secure Agent Infrastructure case, such as support for anonymous agents. Furthermore, as opposed to our work, SMASH lacks an in-depth investigation of its security properties.

3.3 Trusted Computing

The Enforcer platform [MSW⁺04] by Marchesini et al. was one of the first systems to benefit from the use of Trusted Computing mechanism, specifically the Trusted Platform Module (TPM). The Enforcer platform relies on the TPM to protect the load-time integrity of a central security policy enforcement engine. IBM's Integrity Measurement Architecture [SZJvD04] measures a Linux system during boot and operation and stores the measurements inside the Platform Configuration Register (PCR) of a TPM. These first generation trusted platforms demonstrate establishing load-time integrity, but rely on the OS for isolating platform applications with different security policies.

Second generation trusted platforms combine virtualization for isolating applications with the static root of trust for measurement of the TPM for establishing load-time integrity. Examples of second generation trusted platforms include [CGL⁺11, SMS⁺09]. One hampering factor for the adoption of second generation trusted platforms is the complexity and number of the integrity measurements. The chains-of-trust formed by the integrity measurements starting at the static root of trust included components such as the platform firmware and OS bootloader thus lengthening and complicating the chains.

Third generation trusted platforms use advances in personal computer hardware in conjunction with the TPM to enable a dynamic switch into a platform configuration where evidence of the platform's load-time integrity can be given. This dynamic switch eliminates the need to include components such as the platform firmware in the chain-of-trust for measurement, thus greatly reducing its complexity. The acTvSM platform by Martin Pirker [Pir15] we use as the base system for the Trusted Docking Station (TDS) introduced in Chapter 5 is such a third generation trusted platform. Another example is TrustVisor [MLQ⁺10] by McCune et al.

Establishing proof of a platform's load-time integrity is one side of the coin; the other is a human verifying said proof before disclosing any sensitive information to such a platform. To overcome the issue of a platform presenting forged load-time integrity proofs Parno [Par08] proposes using a local channel between a user and a platform's TPM. Similarly, McCune et al. [MPSvD07] propose to use an axiomatically trusted portable device to implement the load-time integrity proof verification for a user. McCune et al. note that such a device should be as simple as possible, and thus easy to understand and certify. They propose to use a USB device with LEDs to signal the load-time integrity verification status. The Secure Docking Module (SDM) we introduce in Chapter 5 is a concrete implementation of such a device. However, the SDM uses local attestation and a custom designed resource acquisition protocol to verify the host platform's load-time integrity. Furthermore, we use shared secrets to authenticate the user to the platform and the platform to the user, while also establishing the presence of the user to the platform.

The Lockdown platform by Vasudevan et al. [VPQ⁺12] is a third generation trusted platform that separates untrusted and trusted software in space and time. It statically splits its platform memory and hampers side channel attacks

by executing only either trusted or untrusted code within a given time frame. A micro-controller driven USB-token with green and red LEDs and direct software channel to the hypervisor indicates the status of the platform to the user.

At first glance Lockdown appears similar to our TDS and SDM security solution, both in architecture and applied technologies. However, Lockdown resides in a different point in the design space and has a different field of application. Lockdown has the advantage of being optimized for a minuscule Trusted Computing Base with a significantly smaller security-critical code base than the acTvSM platform. However, Vasudevan et al. report that Lockdown's switching-times between executing trusted and untrusted code are between 13 and 31 seconds. Thus Lockdown's low performance when switching between system states makes it unsuitable for disaster response scenarios. Furthermore, the Lockdown architecture disregards even simple hardware attacks on the state signalling token. We believe our robust and side-channel resilient SDM implementation can be used to even improve Lockdown.

3.4 Secure Data Storage

Cryptographically protected data storage is a versatile solution for protecting data at rest. Using cryptography it is possible to achieve data confidentiality and integrity even in the face of a storage that is under the complete control of an adversary. Recent years have seen a flurry of activity in the field of protecting data in multi-tenant systems such as the cloud [SS13, SvDJO12, YCZD13, YSK09]. Whereas the cloud has access to high performance computers and storage devices, and cloud solutions are often optimized for fast parallel access, we investigate the other end of the spectrum. With the Secure Block Device we investigate the question of providing versatile cryptographically protected storage for devices with limited capabilities.

The Secure Block Device (SBD) uses a number of well known techniques to achieve its goals. These techniques have been used in the past to achieve similar goals in different contexts, such as secure processors and secure remote storage. Examples include the use of Merkle-Trees for authenticated storage [ECG⁺09, LTM⁺00, SOD07, DK06, RPS06, RCPS07, MS01, GSMB03, SvDJO12, SS13, YCZD13], the use of Authenticated Encryption (AE) for authenticated storage [YSK09, KEAR09], and the use of block caches to improve performance for authenticated storage systems [SvDJO12, WH08]. We have designed and implemented the SBD to cover the middle ground between hardware solutions for secure processors [LTM⁺00, SOD07, DK06, RPS06, RCPS07] and solutions for secure remote storage [MS01, GSMB03, SvDJO12, SS13, YCZD13].

DroidVault [LHB⁺14] is a system to provide a secure data vault on Android devices. It uses the ARM TrustZone security extensions to isolate a custom small Trusted Computing Base (TCB) system from a non-secure, normal world OS. DroidVault provides code authentication, secure data processing, secure networking, and secure I/O. DroidVault also provides a secure storage based on AE, but without guaranteeing data freshness. In contrast the SBD is more flexible,

provides a cache for improving performance, allows fast random access to individual blocks, and guarantees data freshness. Although the SBD was developed for the ANDIX OS it can be used in any scenario where the cryptographic key and the integrity root hash can be protected.

The Trusted Language Runtime (TLR) [SRSW14] uses ARM TrustZone to isolate a .NET micro managed runtime, which has a mechanism to seal data to a managed application. Similarly, Kostianen et al. introduce a system for protecting On-board Credentials [KEAR09] in an isolated compartment. They also use AE to seal data. In contrast, the SBD provides a generic, scalable data protection mechanism. Again the SBD provides fast random block access to data backed by the Merkle-Tree and the block cache.

The problem of storing sensitive data in an untrusted data store is relevant to secure processors such as XOM [LTM⁺00] and AEGIS [SOD07], which need to protect the integrity and confidentiality of their RAM. Secure processors require special hardware and operate under strong resource restrictions compared to software. Nevertheless, they have significant common ground with the SBD. Secure processors [SOD07, DK06, RPS06] also use a small cache operating on the decrypted data to increase efficiency and reduce access times. Furthermore, they also use Merkle-Trees to ensure data integrity. Here, Rogers [RCPS07] et al. have introduced a method called Bonsai Merkle-Tree to reduce the actual memory consumption of this protection mechanism. This work inspired our use of special data blocks storing integrity tags to reduce the size of the Merkle-Tree.

Secure storage of sensitive data is also relevant to network file systems storing data on untrusted servers, such as SUNDR [MS01], SiRiUS [GSMB03], and IRIS [SvDJO12, SS13], the trusted cloud storage by Yang et al. [YCZD13], and the user-level network file system by Yun et al. [YSK09]. Here, the problem of having several clients concurrently working on the same data exacerbates the integrity protection problem, by introducing *forking* [MS01]. In a *forking* attack, the untrusted server presents data and integrity state copies to different clients. This is a problem we disregard for the SBD, as we do not allow file sharing between different Trusted Applications (TAs). Many of these systems are build on, or integrated into, the network file system (NFS). SiRiUS [GSMB03] is a NFS client extension that introduced fast random access through splitting files into blocks, and separating the integrity information, similar to the mechanism used by the SBD. Yun et al. have created a custom cryptographic scheme for their user-level network file system [YSK09]. In their scheme they integrate a Merkle-Tree with AE and proof the correctness of their approach. The SBD can support any number of custom cryptographic schemes through its cryptography abstraction layer. IRIS [SvDJO12, SS13] is a high performance cloud file system with a distributed architecture that aims at support non-blocking parallel access for a large number of concurrent users, and provides proofs of retrievability. The SBD is neither a file system, nor is it designed for massive parallel use. The SBD is a minimal TCB component for adding authenticity to any storage back end with fast random block access. VPFS [WH08] introduces a file system for use by a security kernel that uses an untrusted storage mechanism. Compared

to the SBD, VPFS is a file system and is integrated into the security kernel, whereas the SBD is a user space library and behaves like a block device. Yang et al. [YCZD13] propose using a custom designed pair of security chips, the S-P chips to prevent forking, without the need of client communication. The SBD is pure, minimal TCB software solution.

4

Threat Modeling Aspects of Disaster Response

Declaration of Sources

This section is new art and not based on prior publications by the author.

4.1 Introduction

Thanks to the work of O'Neill et al. [OSZW12] we are aware that confidentiality, integrity, and above all availability are essential security requirements in disaster response. We also learned from O'Neill et al. that the main assets in disaster response are the, as they call them, information exchanges between disaster response personnel. The assessment of the domain experts on which O'Neill et al. base their work attributes the highest security requirements to voice communication, because this is what disaster response personnel uses to establish situational awareness and implement commands.

With the introduction of the Secure Agent Infrastructure (see Section 2.6) parts of the voice communication of disaster response personnel are replaced with automated computer based information processing. The Secure Agent Infrastructure automates information collation and command implementation

using the Information Delivery Agent and the User Communication Agent Mobile Agents. Hence we need to add suitable security mechanisms to the Secure Agent Infrastructure to fulfill the security requirements of disaster response voice communication.

As the assets have been identified by O’Neill et al., the next step in a security-by-design process falls to us: identifying concrete threats to these assets [Tor05]. Identifying the pertinent threats to disaster response communication using the Secure Agent Infrastructure enables us to better gauge the requirements for suitable security mechanisms and evaluate the security performance of such a system’s implementation.

We use the Microsoft Threat Modeling Tool 2016 to model aspects of the Secure Agent Infrastructure and identify pertinent threats. The Microsoft Threat Modeling Tool 2016 uses Data Flow Diagrams as models and STRIDE-per-interaction for threat enumeration. As *first contribution* we model two disaster response activities. These two activities are gaining situational awareness and command and control. According to O’Neill et al. all disaster response communication falls into these two categories. To model these activities, we first created an abstract model of the Secure Agent Infrastructure. We have derived this model based on the mode of operation of the Secure Agent Infrastructure as described in Section 2.6.2 and the results described by O’Neill et al. [OSZW12]. Based on the Secure Agent Infrastructure model, we create a specific high level model for each activity. We use this specific model to analyse the basic security properties of the activity. To make the activities more tangible we supplement their description with examples based on the wildfire scenario we described in the introduction of this thesis. Our analysis of the threats on the high level models already gives us significant insights into potential security issues, which is in-line with results, such as by Schaad and Borozdin [SB12], who also report good results with using STRIDE in the early stages of software development. Specifically, we can extend the results from O’Neill et al. O’Neill et al. state that disaster response communication requires a high degree of confidentiality, integrity, and availability. We give 41 concrete threats to information gathering for situational awareness and command and control. These 41 threats detail problems such as repudiation of commands, tampering with information, etc. Thus they uncover security issues not immediately evident from the need for confidentiality, integrity, and availability. Finally, we believe our threat list to be relevant to future research into disaster response security, because it can serve as a basis for the risk evaluation of specific communication exchanges.

The analysis of the high level models, although it grants valuable insights, is inadequate to map the intricacies of a complex Mobile Agent System, such as the Secure Agent Infrastructure. We therefore increase the resolution of our model for the situational awareness example and create a more detailed model that captures the threats related to using Mobile Agents as information gathering and relaying mechanism. In this detailed model of the situational awareness example we identify a highly repetitive pattern, which we call the Distributed Secure Agent Platform Outpost (DSAP Outpost). The DSAP Outposts are agent plat-

forms situated at the interface points between the Secure Agent Infrastructure and all external entities interacting with the Secure Agent Infrastructure. Our *second and primary contribution* in this chapter is an in-depth threat analysis of a DSAP Outpost. Our efforts here have yielded a list of 54 threats to the DSAP Outpost. Note that we only summarize these 54 threats in this chapter and defer their in-depth discussion to Appendix A. We use this list of 54 threats to validate the effectiveness of the Trusted Docking Station (TDS) and the Secure Docking Module (SDM) (see Chapter 5). The TDS and the SDM are two security mechanisms we have developed for securing the DSAP Outpost as part of this thesis.

As part of our second contribution we have also compared the results of our STRIDE-per-interaction based threat modeling efforts with the literature on Mobile Agent security. Specifically, we have compared the threats we identified with the generic discussions of Mobile Agent System security by Jansen and Karygiannis [JK99, Jan00], Borselius [Bor02], and Bierman and Cloete [BC02]. Although these three sources comprehensively discuss threats to Mobile Agent Systems, we were still able to *identify new pertinent threats* not found in literature.

As a *third contribution*, we have also recorded our observations on using Data Flow Diagrams and STRIDE-per-interaction for threat modeling. We use this threat modeling methodology by using the Microsoft Threat Modeling Tool 2016. We have recorded our observations of using this threat modeling methodology in the context of disaster response in general and in the context of Mobile Agent System supported disaster response specifically. Our findings are twofold. First, we observe that due to the use of Data Flow Diagrams as models, STRIDE-per-interaction lends itself naturally to capture the threats to the information flows in disaster response. This is evident, because we consider almost all threats for our models enumerated by the Microsoft Threat Modeling Tool 2016 pertinent. This holds true for both the high level models and the detailed DSAP Outpost model. Second, even though we have used a standard data flow to represent the action of instantiating an agent, we were able to identify a number of relevant threats. We think this significant, as STRIDE-per-interaction per se does not support processes such as sending and instantiating a Mobile Agent.

4.2 Secure Agent Infrastructure Model

Here we create an abstract model of the Secure Agent Infrastructure. We base our model on the Secure Agent Infrastructure's mode of operation (see sec:prelim:sai:uc) and the findings described by O'Neill et al. [OSZW12]. We then use the Secure Agent Infrastructure model to create high level threat models for establishing situational awareness (Section 4.3) and dispatching a command during disaster response (Section 4.4).

O'Neill et al. have analyzed a disaster response scenario to establish who needs to talk with whom and what information is exchanged during crisis management. They have designed a disaster response scenario and broken the sce-

nario down into activities. An activity is described by the information exchange requirements it entails. According to O'Neill et al., an information exchange requirement describes the need of two or more actors in disaster response to exchange a piece of information. An information exchange requirement is a tuple $(src, dst, t, s, crit, c, other)$, where src is the source of the information, dst its destination, t the type, such as text, data, or voice, s the size of the information based on its type, $crit$ the information's criticality, c determines how confidential the information is, and finally $other$ is a set of miscellaneous analysis attributes. O'Neill et al. have captured and analyzed over 700 information exchange requirements for their disaster response scenario. Based on their results they state that these information exchange requirements fall into two categories. These categories are situational awareness and command and control.

The goal of the Secure Agent Infrastructure is to automate gathering information for situational awareness and to automate dispatching of commands. The Secure Agent Infrastructure component that implements this automation is the Process Management Subsystem (see Section 2.6). A Secure Agent Infrastructure user can request information from the Process Management Subsystem. The Process Management Subsystem uses Mobile Agents to gather the requested information from various sources, including human users and external information systems. It then collates this information and presents it to the user (see Section 2.6.2). The Process Management Subsystem can implement complex information gathering workflows, where several sources need to be queried sequentially, and/or in parallel. For example, the Process Management Subsystem can use information queried from a previous source, to refine future queries. Similarly, the Process Management Subsystem can dispatch commands for its users and track execution of these commands.

Based on the observation that communication in disaster response is centered around either gaining situational awareness or command and control and how the Secure Agent Infrastructure facilitates these processes we have come up with the abstract model depicted in Figure 4.1. In our model we have combined the Process Management Subsystem with the Agent Repository and the Resource Lookup System (see Section 2.6.1) into the Core Secure Agent Infrastructure. We have combined these three because they work closely together and we assume they are in the same security domain (cf. Section 4.5.3). The Core Secure Agent Infrastructure interfaces with users and external information systems using Mobile Agents running on Distributed Secure Agent Platforms (DSAPs).

In Figure 4.1 we have modeled a Commander requesting information from the Secure Agent Infrastructure (1. Request information). The Secure Agent Infrastructure gathers the information from disaster Responder(s) (2. Request information) and External Information System(s) (4. Request information). Note, that these request information steps can be parallel or sequential. For example, first the Secure Agent Infrastructure requests information from a disaster Responder and based on this information determines that it needs information from further sources, such as an External Information System. In Figure 4.1 we have depicted this. First the Secure Agent Infrastructure queries a disaster

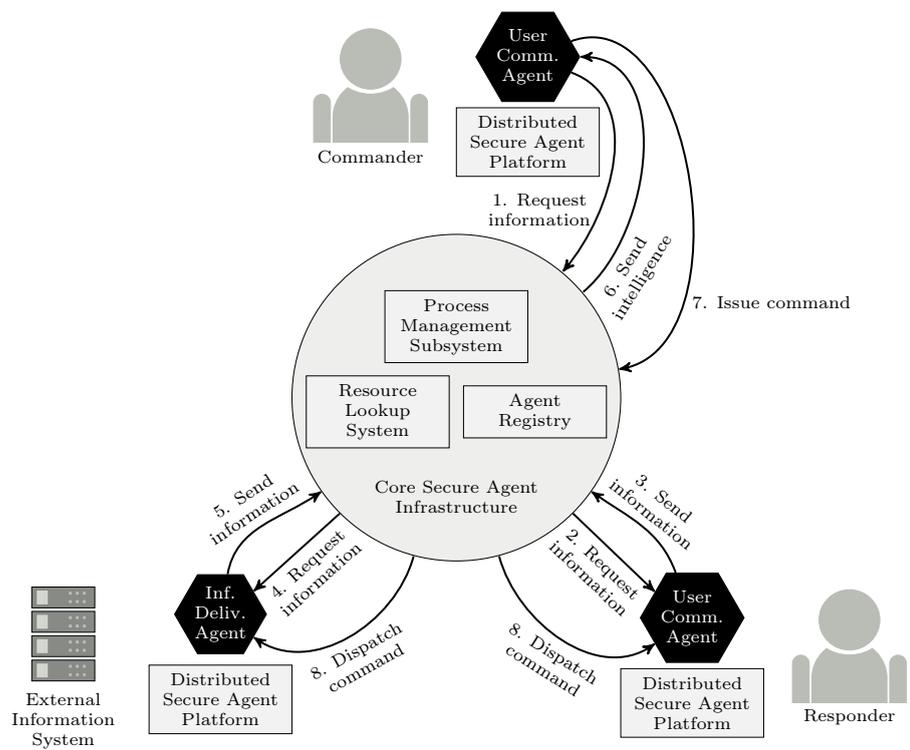


Figure 4.1: Our high level model for gathering information and issuing commands in disaster response

Responder (2. Request information) and receives this information (3. Send information). Then it queries an External Information System for further data (4. Request information) that it then receives (5. Send information). The Secure Agent Infrastructure collates the information, prepares a report, and sends it to the Commander (6. Send intelligence). Based on the received intelligence the Commander decides on a certain course of action and instructs the Secure Agent Infrastructure to implement this decision (7. Issue command). The Secure Agent Infrastructure then dispatches the command to the pertinent disaster Responders and External Information Systems (8. Dispatch command).

4.3 Situational Awareness

Situational awareness is a key factor in the crisis management decision making process, as the quality of the information available about the situation directly influences the quality of decision-making [OSZW12]. Here we model threats to the process of obtaining information during a crisis. To make the modeling process more tangible and allow us to supplement our analysis with concrete examples, we revisit the wildfire scenario from the introduction.

4.3.1 Model Description

We introduce this high level model as a simple abstraction to model the process of gathering information in the disaster response phase of crisis management. In the high level model we try to avoid concepts from information processing systems and concentrate on the actors involved. We do this to focus the attention on the overall security requirements for gathering information pertaining to situational awareness, and avoid to clutter these requirements with technical details. In addition this allows us to corroborate the findings of O’Neill et al. that confidentiality, integrity and availability are essential to crisis management communication using a different methodology.

We use the Microsoft Threat Modeling Tool 2016 to create our threat model. Our model is the Data Flow Diagram depicted in Figure 4.2. We derive our information gathering model from the overall model (see Figure 4.1). Specifically, we extracted the part where disaster Responders and External Information Systems send requested information to the Core Secure Agent Infrastructure, the Core Secure Agent Infrastructure prepares a report on the information for the Command, and it then sends the report to the Command. We model all disaster responders using a single external entity (Responder). Similarly, we model all external information systems as the single external entity External Information System. Finally, all entities that make decisions based on the gathered information are represented by the singular Command entity. In addition we replaced the Core Secure Agent Infrastructure with the single process Information Collation. This process models an entity collating the information to present it to a Command entity so that it can inform a decision. In this high level model we would have preferred to abstract this process in the sense that infor-

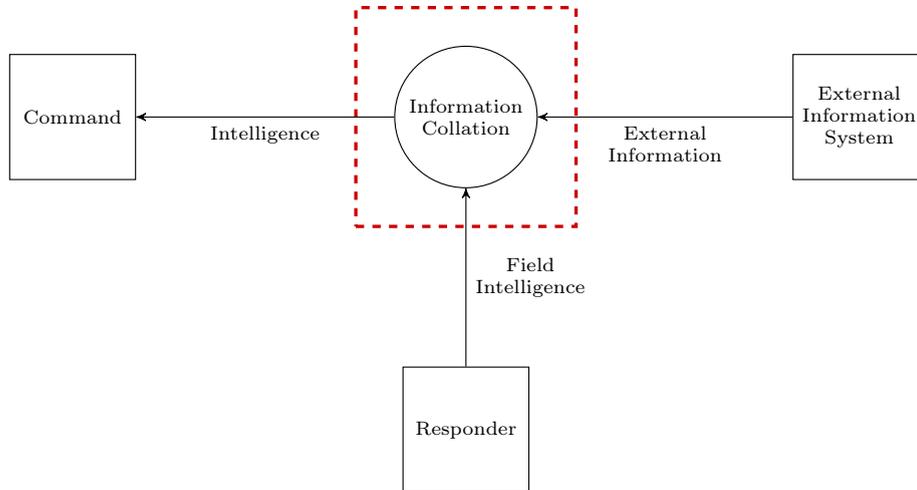


Figure 4.2: A Data Flow Diagram modeling gathering intelligence for informing a command decision during the response phase of disaster response. We use this model as input to the STRIDE-per-interaction analysis using the Microsoft Threat Modeling Tool 2016.

mation collation is done by the Command entity itself. However, we had to add the Information Collation process, because the threats generated by STRIDE-per-interaction from external entities interacting with each other are limited in nature (see Section 2.8.5).

An example realization of the model in our scenario would be operational commanders of fire fighter units reporting to their tactical command about the state of fire suppression in specific sections. Tactical command could supplement this information with maps fetched from an external information system. The tactical commander would use this information to decide where to focus fire suppression, which units need further support, etc. The model works equally well to describe several police officers reporting to operational command who they are evacuating in a specific area, while operational command checks the resident registry information system to verify evacuation completeness.

Our information gathering model does not include the Command entity requesting information. Therefore, our model does not generate threats, such as, the request for information never reaching a disaster responder or an external information system. So, why did we choose to omit requesting information? Because requesting information is a form of command and covered by our second high level model for command and control (see Section 4.4).

4.3.2 Threat Modeling Results

We used Microsoft Threat Modeling Tool 2016 to create the model depicted in Figure 4.2 and then let the tool generate a list of threats. The Microsoft Threat

Modeling Tool 2016 generates 25 threats for our model. We consolidated the list removing four threats pertaining to elevation of privilege attacks and manually added 4 threats concerning the *Intelligence* data flow (threat 22-25). Table 4.1 lists the 25 threats to this disaster response process we consider relevant.

The 25 threats of our consolidated list can be divided into two categories. The first category encompasses the threats pertaining to the *External Information* and *Field Intelligence* data flows. The second category consists of the threats to the *Intelligence* data flow. We will now briefly discuss the threats in those two categories and highlight why we think that these threats are pertinent.

External Information and Field Intelligence

In this first category we will look at pairs of threats. So, if we discuss “Spoofing the Information Collation Process” we actually discuss both Threat 01 and Threat 10. We take note of the threat numbers in the title of the threat pairs we discuss in the following description.

Also for this first category, we removed 2 threat pairs, that is, in total 4 threats, from the list of threats generated by STRIDE-per-interaction. These threat pairs were “Information Collation May be Subject to Elevation of Privilege Using Remote Code Execution” and “Cross Site Request Forgery”. Those two pairs refer to highly technical threats, and in this high level threat model we wanted to limit ourselves to the conceptual threats to situational awareness in disaster response. The effect of these two threat pairs is adequately modeled by the “Elevation Using Impersonation” and “Elevation by Changing the Execution Flow in Information Collation” threats.

Spoofing the Information Collation Process (01, 10) Here an adversary is able to impersonate the Information Collation process and this can lead to information disclosure to unauthorized recipients and Denial-of-Service. We consider this threat applicable in our scenario, based on the findings by O’Neill et al. [OSZW12] that state the importance of both confidentiality and availability in disaster response. By spoofing the Information Collation process the perpetrator can intercept information that is destined for a commander. If there is no further protection on the information, then the adversary can read the information, and if the adversary does not relay the information, it will be lost.

Spoofing the External Information System/Responder (02,11) An adversary is able to impersonate the External Information System, or a Responder. We consider this threat pertinent, because it enables an adversary to inject tampered data, or prevent the impersonated entity from sending information to the Information Collation process. In both cases the adversary might be able to change the decision made by Command according to his or her wishes, while implicating the spoofed External Information System or Responder.

Potential Lack of Input Validation for Information Collation (03,12) By not verifying the correctness of the incoming information the Information

Table 4.1: A consolidated list of threats created by using the STRIDE-per-interaction threat enumeration method on the model depicted in Figure 4.2. The *Type* column uses the STRIDE threat categories. These are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege.

ID	Type	Title
<i>External Information</i>		
01	S	Spoofing the Information Collation Process
02	S	Spoofing the External Information System
03	T	Potential Lack of Input Validation for Information Collation
04	R	Potential Data Repudiation by Information Collation
05	I	Data Flow Sniffing
06	D	Potential Process Crash or Stop for Information Collation
07	D	Data Flow <i>External Information</i> Is Potentially Interrupted
08	E	Elevation Using Impersonation
09	E	Elevation by Changing the Execution Flow in Information Collation
<i>Field Intelligence</i>		
10	S	Spoofing the Information Collation Process
11	S	Spoofing the Responder
12	T	Potential Lack of Input Validation for Information Collation
13	R	Potential Data Repudiation by Information Collation
14	I	Data Flow Sniffing
15	D	Potential Process Crash or Stop for Information Collation
16	D	Data Flow <i>Field Intelligence</i> Is Potentially Interrupted
17	E	Elevation Using Impersonation
18	E	Elevation by Changing the Execution Flow in Information Collation
<i>Intelligence</i>		
19	S	Spoofing of Command
20	R	Command Potentially Denies Receiving Data
21	D	Data Flow <i>Intelligence</i> Is Potentially Interrupted
<i>The following 4 threats were manually added</i>		
22	S	Spoofing of the Information Collation Process
23	T	Potential Lack of Input Validation for Information Collation
24	I	Data Flow Sniffing
25	E	Elevation by Changing the Execution Flow in Command

Collation process is susceptible to attacks that are based on injecting tampered information. In disaster response, it might not always be possible to verify the veracity and integrity of the incoming information. In this cases ensuring that the information comes from an authentic source and has not been tampered with becomes all the more important.

Potential Data Repudiation by Information Collation (04,13) According to O’Neill et al. [OSZW12] the objective of the disaster response phase is to: “. . . eliminate or limit the scope of arising crisis situations, which should provide victims with necessary help and neutralize sources of a threat.” The key observation with regard to repudiation here is that disaster response is about helping and protecting victims from further harm. If victims are put to further harm due to decisions based on bad intelligence anyone involved in information collation might have sufficient motivation to repudiate ever having collated the bad intelligence. Therefore we consider this threat applicable.

Data Flow Sniffing (05,14) Here an adversary is able to eavesdrop on the information sent to the Information Collation process. As previously established there is a strong need for confidentiality in disaster response, and therefore we regard this threat as applicable.

Potential Process Crash or Stop for Information Collation (06,15) The information collation fails at collating the information in time or at all. Of all three core security goals in disaster response, availability is held as the most important one (see O’Neill et al. [OSZW12]). If information collation fails to work in a timely fashion, the command decisions will be felled using degraded intelligence. Therefore we consider this threat pertinent.

Data Flow *External Information/Field Intelligence* is Potentially Interrupted (07,16) An external agent interrupts either the *External Information* or the *Field Intelligence* data flow to the information collation process. A realization of this threat allows an adversary to potentially selectively interrupt information gathering and thus, to a certain extend, control the output of the Information Collation process and therefore influence the decision by Command. We believe this threat to be relevant.

Elevation Using Impersonation (08,17) An adversarial information collation process might use information send by either the External Information System or a Responder to impersonate the sending entity. As Information Collation is actually the process that prepares information for command in our system, it is in an excellent position to implement this kind of impersonation. The Information Collation process might impersonate either sending entity to plant tampered or incriminating information and we consider this threat pertinent.

Elevation by Changing the Execution Flow in Information Collation (09,18) An adversary could pass (tampered) data into the Information Collation process to control its output and thus the Command decision to his or her choosing. As any threat that allows an adversary to control the Command decision we regard this threat as applicable.

Intelligence

In the second category we briefly discuss the threats to the *Intelligence* data flow from the Information Collation process to the Command. Here we manually added 4 threats pertaining to spoofing, information disclosure, tampering, and elevation of privilege (22-25). This was necessary, because STRIDE-per-interaction only generates limited threats for data flows that end in an External Interactor (see Section 2.8.5 for details). Note that we did not add an “Elevation Using Impersonation” threat (see for example Threats (08,17)), because there is no entity in our model that the Command entity could impersonate someone to, as there are no outgoing data flows in this model. Alternatively to manually adding the threats, we could have modeled Command as a process to get the full gamut of threats. However, conceptually Command is an External Interactor, and we would have then had to remove those threats that are inappropriate to an External Interactor.

Spoofing of Command (19) Here an adversary is able to impersonate the Command External Interactor. A realization of this threat can lead to information disclosure and Denial-of-Service (see Threats (01,10) above). We believe this threat to be even more pertinent than the Threats (01,10), because Command is send collated information from all External Information Systems and all Responders, instead of just one or a few. So for information disclosure an adversary that realizes this threat gains access to all command decision relevant information and by denying or delaying delivery of this information can seriously impact the decision making process.

Command Potentially Denies Receiving Data (20) Command decisions can endanger disaster victims (cf. Threats (04,13)). If Command makes a decision that harms victims, although available intelligence might have indicated a different decision, then a Commander might be sufficiently motivated to repudiate ever having received incriminating intelligence. Therefore this threat stands.

Data Flow *Intelligence* Is Potentially Interrupted (21) An external adversary interrupts the *Intelligence* data flow to the Command. Here the adversary can prevent relevant intelligence reaching Command and thus influencing command decisions. This threat is further aggravated if the adversary is capable of selectively targeting specific intelligence updates from the Information Collation process, as this would allow for even more control on the decisions made by Command. For these reasons we believe this threat to be relevant.

Intelligence - Manually Added Threats

Spoofing the Information Collation Process (22) An adversary is able to impersonate the Information Collation process. We consider this threat applicable, because here the adversary can inject tampered data directly into Command and also withhold or delay delivery of important information by the real Information Collation process. In both cases the adversary can exact control over the command decision.

Potential Lack of Input Validation by Command (23) As indicated in Threat 23 an adversary that is able inject tampered data directly into the Command is a potent threat. Therefore command should consider taking measures to verify the incoming information, or if this is not possible, at least ensure authenticity of its source.

Data Flow Sniffing(24) An adversary is able to eavesdrop on the information the Information Collation process sends to Command. As documented in Threat 19 an adversary that can realize this threat has access to sensitive information collated from a number of sources, which we believe exacerbates this threat compared to the Threats (05,14).

Elevation by Changing the Execution Flow in Command (25) An external adversary is able to inject tampered data into the *Intelligence* data flow (cf. Threat 22 and Threat 23). Such an adversary can potentially exact an alarming level of control over the command decision. Therefore we consider this a threat that requires mitigation.

4.3.3 Mobile Agent System Level Model

Description of the Model

The high level model is useful for identifying high level threats. However, it does not do the complexity of a Mobile Agent System based information processing system justice. To gain deeper insights into the threats that arise from using the Secure Agent Infrastructure Mobile Agent System to implement the information gathering process, we created a model that incorporates the Secure Agent Infrastructure components involved. This model is shown in Figure 4.3.

Two facts are immediately apparent when studying Figure 4.3. First the model's complexity has increased significantly and second there is a highly repetitive pattern of Distributed Secure Agent Platforms (DSAPs) hosting a Mobile Agent, such as the Information Delivery Agent or the User Communication Agent communicating with an external entity, such as an External Information System, or Responder. We call this repetitive pattern a DSAP Outpost, which is due to the fact that these DSAPs are often physically close to the external entity they communicate with.

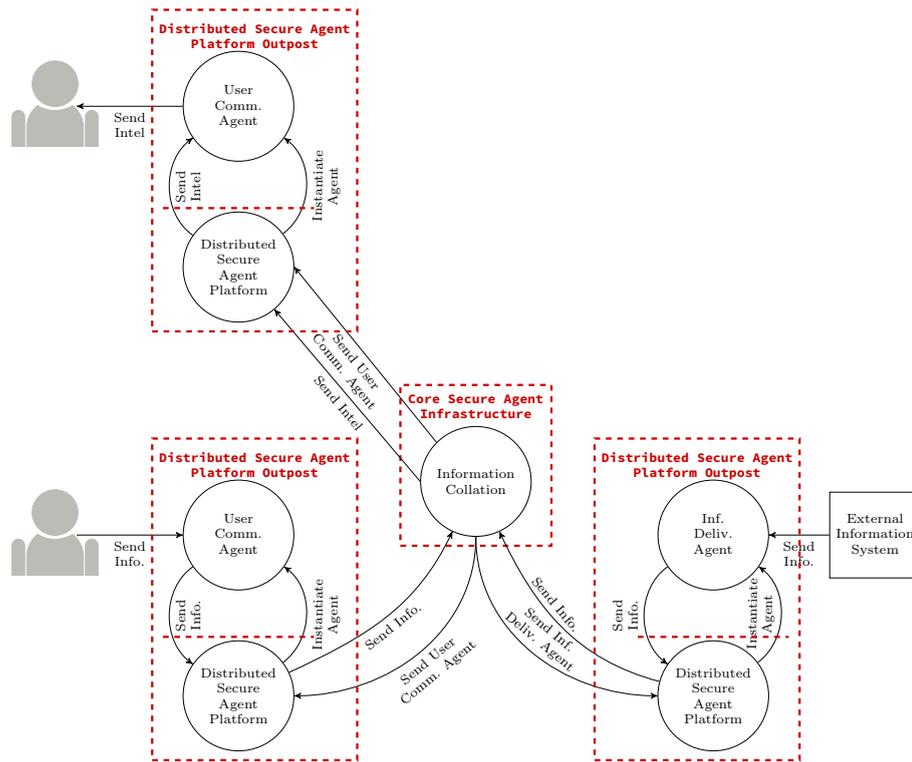


Figure 4.3: A model of the information gathering process for situational awareness that incorporates Secure Agent Infrastructure components. This model exhibits a highly repetitive pattern, the Distributed Secure Agent Platform Outpost. The Distributed Secure Agent Platform (DSAP Outpost) comprises a Distributed Secure Agent Platform hosting a Mobile Agent that interacts with an External Interactor.

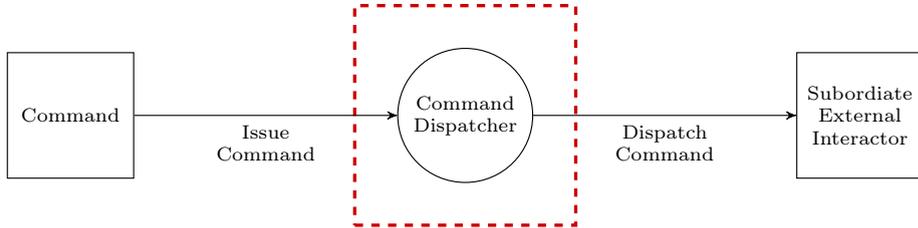


Figure 4.4: A Data Flow Diagram modeling command and control during the response phase of a crisis. We use this model as input for a STRIDE-per-interaction threat enumeration using the Microsoft Threat Modeling Tool 2016.

Results

We have modeled this Mobile Agent System level model using Microsoft Threat Modeling Tool 2016 and used the tool to generate a list of 163 threats in total. As expected, due to the repetitive nature of the model, the list of threats suffered the same problem. Having three almost identical DSAP Outpost in the model lead to a triplicate set of threats, where the only differences were the direction of the data flows and the quality of the information that was processed. Specifically, different information sources generate information with varying degrees of sensitivity. We will further discuss this in Section 4.6.2 As we cannot make any assumptions on the sensitive of information beyond the requirements outlined by O’Neill et al. [OSZW12], we saw little value in studying the DSAP Outposts thrice. Therefore, we decided to create a model that encompasses all aspects of a DSAP Outpost and analyze this model in detail. We have documented this in Section 4.5.

4.4 Command and Control

Here we model the process of dispatching commands during disaster response. Our first high level model was concerned with gathering data for informing a command decision. Here, we turn the table around and investigate the other direction, that is, how commands are implemented.

4.4.1 Model Description

We use a high-level representation for modeling command dispatching in disaster response. As with our first model (cf. Section 4.3) we try to avoid concepts from information processing systems to focus on the overall security requirements for command and control.

Figure 4.4 depicts the Data Flow Diagram we use as our model for the STRIDE-per-interaction threat enumeration. We derive this model from the overall model (see Figure 4.1). For this model we extract the parts that pertain

to command and control, such as issuing a command and dispatching a command. In this model we again use a single External Interactor (Command) to represent all personnel that make decisions. We also unite all External Interactors that implement commands in a single external entity, without distinguishing between human beings and information processing systems. Central to our model is the Command Dispatcher process. The Command Dispatcher process models the command propagating through the chain of command, before being dispatched to all entities implementing the command. This process can involve a number of personnel and information systems. The Command Dispatcher process is also the process that will be handled, at least in part, by the Core Secure Agent Infrastructure.

As an example, consider strategic command issuing an evacuation order for a residential area. For command implementation to be successful the command decision must be implemented through the different levels of the command hierarchy until it is translated into orders for operational personnel. It is important that command decisions reach the right people (availability) and that the command is faithfully processed and passed on (integrity). Confidentiality is also important for implementing command decisions. For example, orders like “expedite evacuation in sector five, because the wildfire is closing in significantly faster than expected” can lead to panicking in the affected civilian population when leaked.

4.4.2 Threat Modeling Results

We used Microsoft Threat Modeling Tool 2016 to analyze the model depicted in Figure 4.4 and let the tool generate a list of threats. The Microsoft Threat Modeling Tool 2016 generates 14 threats for our model. We consolidated the list removing two threats pertaining to elevation of privilege attacks and manually added 4 threats concerning the *Dispatch Command* data flow (threats 04-07). We have done this for the same reasons outlined in Section 4.3.2. Table 4.1 summarizes the 16 threats to this disaster response example we consider pertinent.

Dispatch Command

Spoofing of the Subordinate External Interactor (01) By realizing this threat an adversary can effect information disclosure and Denial-of-Service attacks. The gravity of these attacks depends on the number of subordinate entities the adversary is able to impersonate, and of course which position they hold. We consider this attack pertinent, as it endangers confidentiality and availability.

Subordinate External Interactor Potentially Denies Receiving Data (02) According to O’Neill [OSZW12], Disaster response encompasses helping victims and neutralizing threats. If victims are harmed because of human error, the involved humans might have sufficient motivation to repudiate ever having received an order, or to claim having received a different order. Therefore, we consider this threat pertinent.

Table 4.2: A consolidated list of threats created by using the STRIDE-per-interaction threat enumeration method on the model depicted in Figure 4.4. The *Type* column uses the STRIDE threat categories. These are Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege.

ID	Type	Title
<i>Dispatch Command</i>		
01	S	Spoofing of the Subordinate External Interactor
02	R	Subordinate External Interactor Potentially Denies Receiving Data
03	D	Data Flow <i>Dispatch Command</i> Is Potentially Interrupted
04	S	Spoofing the Command Dispatcher Process
05	T	Potential Lack of Input Validation for Subordinate External Interactor
06	I	Data Flow Sniffing
07	E	Elevation by Changing the Execution Flow in Subordinate External Interactor
<i>Issue Command</i>		
08	S	Spoofing the Command Dispatcher Process
09	S	Spoofing the Command External Interactor
10	T	Potential Lack of Input Validation for Command Dispatcher
11	R	Potential Data Repudiation by Command Dispatcher
12	I	Data Flow Sniffing
13	D	Potential Process Crash or Stop for Command Dispatcher
14	D	Data Flow <i>Issue Command</i> Is Potentially Interrupted
15	E	Elevation Using Impersonation
16	E	Elevation by Changing the Execution Flow in Command Dispatcher

Data Flow *Dispatch Command* Is Potentially Interrupted (03) This threat directly attacks the availability of a command. As with Threat 01 the severity of the impact depends on how many commands the adversary can interrupt and who the recipients are. An adversary being able to selectively interrupt command distribution can severely hamper the disaster response effort. As such this threat stands.

Spoofing the Command Dispatcher Process (04) By impersonating the command dispatcher an adversary can inject commands, or prevent that the affected subordinate entities receive commands issued to them. In line with the other threats to this data flow the severity depends again on whom the adversary is able to impersonate the command dispatcher process to, and how many subordinate entities the adversary is able to deceive. In all cases this threat violates availability and integrity and we consider it relevant.

Potential Lack of Input Validation for Subordinate External Interactor (05) Here an adversary directly injects a tampered command or tampers with a command during transmission. Again, the grievousness of the threat depends on against whom the adversary is able to realize this threat and how many subordinates the adversary can affect. An adversary that can issue or change commands to Subordinate External Interactors can seriously hamper disaster response, hence we consider this a threat.

Data Flow Sniffing (06) Here an adversary is able to eavesdrop on the information sent to at least one Subordinate External Interactor. According to O'Neill et al. [OSZW12] confidentiality is important to disaster response, therefore this threat is pertinent. As with the previous threats, threat severity depends on which concrete data flows the adversary can eavesdrop on, and how many.

Elevation by Changing the Execution Flow in Subordinate External Interactor (07) An adversary is able to tamper with a command in a way that alters the behavior of the Subordinate External Interactor. As with all threats to this data flow, this threat's severity depends on which concrete data flows, and how many of them, the adversary can tamper with. We consider this threat relevant.

Issue Command

Spoofing the Command Dispatcher Process (08) This process models the command propagating through the command hierarchy until it is broken down to a level where it is sent to the executive personnel, for example, first responders. Similar to Threat 01 this is a threat to availability and confidentiality. Here the severity depends on where in the chain of command the adversary is able to spoof the dispatcher process. If the adversary is, for example, able to

impersonate an entity close to a strategic decision maker, the adversary may significantly influence disaster response and potentially gain high value intelligence. Even at lower levels in the chain of command the threat is pertinent.

Spoofing the Command External Interactor (09) An adversary that can impersonate the Command External Interactor can directly inject tampered commands (see Threats 11,16) and can prevent the authentic command entity from sending commands (see Threats 13,14) to the Command Dispatcher Process. In addition to enabling a number of relevant threats, it can also implicate the real Command External Interactor. Hence we think it needs mitigation.

Potential Lack of Input Validation for Command Dispatcher (10) Here an adversary is able to inject tampered information into the Command Dispatcher Process and that includes issuing forged commands due to a lack of authenticity and integrity checking. Note that especially in large scale disaster response operations verifying the authenticity and integrity of incoming commands can be daunting. We consider this threat to be grievous. For reference, all integrity threats in our situational awareness discussion in Section 4.3 included changing the command decision outcome. Here the adversary can directly inject or tamper with a command. The impact of this threat depends on where in the chain of command this threat is realized. An adversary realizing this threat can effect a Denial-of-Service attack by injecting nonsensical commands that bind critical resources, she can send out commands that reveal sensitive information to an unauthorized audience, or otherwise compromise the integrity of the Command Dispatch Process.

Potential Data Repudiation by Command Dispatcher (11) Accountability is an important aspect of disaster response, as disaster response is concerned with preventing harm to disaster victims. If disaster victims are further harmed due to human error, an entity involved in dispatching commands might have sufficient motivation to repudiate ever having received a command or report having received a different command. We think this is a threat requiring mitigation.

Data Flow Sniffing (12) Here an adversary is able to eavesdrop on at least one command sent to the Command Dispatcher Process. As we have previously established the need for confidentiality in disaster response, we consider this threat applicable.

Potential Process Crash or Stop for Command Dispatcher (13) When the command dispatcher process fails to dispatch commands it is a clear violation of availability. Given the importance of command and control, this is a pertinent threat to disaster response.

Data Flow Issue Command Is Potentially Interrupted (14) An adversary that can interrupt commands coming from the Command External Interactor can exert control over command implementation. On the one hand interrupting commands is a clear violation of availability. On the other hand by selectively interrupting commands an adversary might be able to control the disaster response effort. Hence, we consider this threat relevant.

Elevation Using Impersonation (15) A compromised command dispatch process might use information received from the Command External Interactor to impersonate the Command External Interactor. This allows for injecting tampered commands, while incriminating the Command External Interactor (see Threat 09). Therefore we think this threat should be mitigated.

Elevation by Changing the Execution Flow in Command Dispatcher (16) An adversary passes tampered information into the Command Dispatcher process to control command implementation. The adversary might “simply” issue forged commands, or even try to change the structure of the Command Dispatcher Process, for example by reassigning personnel involved in the process. We believe this is a relevant threat requiring mitigation.

4.5 Distributed Secure Agent Platform Outpost

4.5.1 Overview

In this section we analyze what we call the Distributed Secure Agent Platform Outpost (DSAP Outpost). The DSAP Outpost is a highly repetitive pattern we found when modeling the disaster response activities implemented using the Secure Agent Infrastructure. A DSAP Outpost comprises a Distributed Secure Agent Platform (DSAP) that hosts Mobile Agents that interact with external entities such as first responders or information systems. The Mobile Agents use the DSAP to communicate with the central components of the Secure Agent Infrastructure. As the DSAP Outpost is an ubiquitous aspect of the Secure Agent Infrastructure we find it valuable to model it and perform an in-depth analysis of the threats it might be subjected to. For the same reason we decided to secure the DSAP Outpost with the two security mechanisms we introduce in Chapter 5. We use our results from this chapter to gauge the effectiveness of these two security mechanisms.

We first describe the Data Flow Diagram we use for threat modelling in Section 4.5.2. We then define our security assumptions in Section 4.5.3 and proceed with presenting our threat modeling results in Section 4.5.4. Note that the actual discussion of the identified threats is deferred to Appendix A. Note further, that Section 4.5.4 only presents the bare-bone results and that our findings based on these results are discussed as part of our conclusions (see Section 4.6.3).

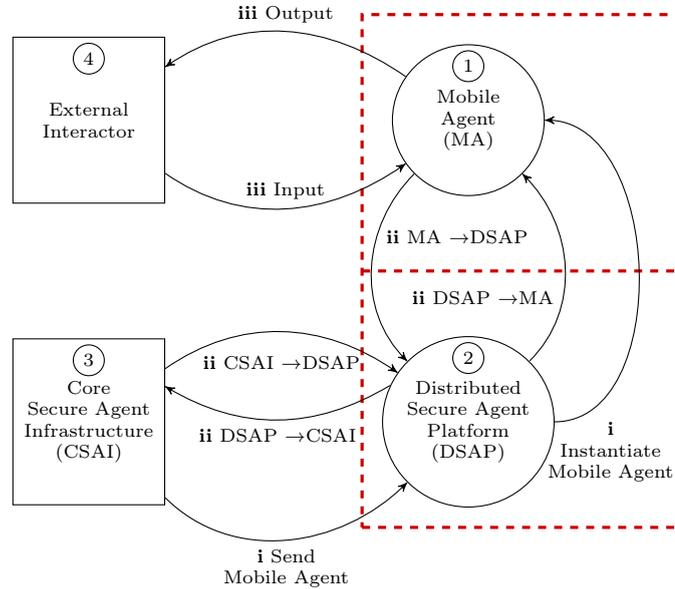


Figure 4.5: A Data Flow Diagram modeling the Distributed Secure Agent Platform Outpost (DSAP Outpost). A DSAP Outpost is at the core of the Secure Agent Infrastructure typical process of sending a Mobile Agent to a DSAP to gather information from an external information system, or converse with a system user. Here we model both, the agent migration and the information exchange between the Core Secure Agent Infrastructure (CSAI) and the Mobile Agent (MA) running on the DSAP. We use this model as input to the STRIDE-per-interaction analysis using the Microsoft Threat Modeling Tool 2016.

4.5.2 Description of the Model

We created our DSAP Outpost model using Microsoft Threat Modeling Tool 2016. Figure 4.5 shows our Data Flow Diagram model. Our model consists of four entities and eight data flows. Starting with the entities, first we have the *Core Secure Agent Infrastructure* (③). The Core Secure Agent Infrastructure is an External Interactor and it comprises a number of components, such as the Process Management Subsystem, the Agent Repository, and the Resource Lookup System (see Section 2.6 and Section 4.2). The next entity is the *DSAP* process (②) that handles agent migration, agent execution and agent communication with the Core Secure Agent Infrastructure. The *Mobile Agent* (①) itself is also represented as a process that is instantiated by the DSAP and communicates with the Core Secure Agent Infrastructure via the DSAP, and directly with the last entity, the *External Interactor* (④). The External Interactor subsumes the human users and external information systems the agent might communicate with.

We group the data flows into three groups.

- i **Agent Migration** consists of the *Send Mobile Agent* and *Instantiate Mobile Agent* data flows. We discuss this group in detail in Section A.5.
- ii **Agent – Core Secure Agent Infrastructure Communication** consists of the *CSAI → DSAP*, the *DSAP → CSAI*, the *MA → DSAP* and the *DSAP → MA* data flows. In Section A.7 we detail this group of information flows.
- iii **Agent – External Interactor Communication** comprises the *Input* and *Output* data flows (see Section A.6).

We assume a secure communication channel between the Core Secure Agent Infrastructure and the DSAP (see Section 4.5.3). Therefore we have configured our model so that the *CSAI → DSAP*, the *DSAP → CSAI*, and the *Send Mobile Agent* data flows provide confidentiality, integrity, and end-point authenticity.

4.5.3 Secure Agent Infrastructure Security Assumptions

Torr [Tor05] explicitly states that it is important to enumerate the assumptions made about a software component during design and development and that these assumptions should be verified at a later stage of the development cycle. Similarly, Haley et al. note the importance of making assumptions [HLMN08] explicit in security requirements engineering process. Haley et al. summarize that these assumptions arise from the analyst choosing what domains (real-world elements) to consider in security requirements engineering and thus defining system context. In the following, we present our assumptions on different domains of the Secure Agent Infrastructure and thus setting the stage for our threat modeling of the DSAP Outpost.

Secure Core Secure Agent Infrastructure

As described in Section 2.6, the Secure Agent Infrastructure comprises a number of core components, such as the Process Management Subsystem, the Agent Repository, and the Resource Lookup System. For our threat modeling activities we combine those components into the Core Secure Agent Infrastructure External Interactor, and we assume that these core components operate within a protected domain and will not violate their security policy, while being inviolable to outside attacks.

We define the *security policy of the Core Secure Agent Infrastructure* to be as follows.

- The Core Secure Agent Infrastructure performs its disaster response tasks according to specification.
- The Core Secure Agent Infrastructure will be permanently available. However, single network connections to the Core Secure Agent Infrastructure might intermittently fail or be interrupted.

- The Core Secure Agent Infrastructure will protect confidential information and will only disclose it to authorized entities.
- The Secure Agent Infrastructure will not degrade or corrupt the integrity of the information it processes.
- The Core Secure Agent Infrastructure will rigorously verify the integrity of incoming information to prevent corruption of the Core Secure Agent Infrastructure, or remote command execution on a component of the Core Secure Agent Infrastructure.

Our security policy is strict and places a strong burden on Core Secure Agent Infrastructure development, operation, and maintenance. However, we claim that the underlying assumption that the Core Secure Agent Infrastructure components and network will be developed, operated and maintained with a focus on security is sound. As such we focused our research efforts on those outlying DSAPs, which are operated in heterogeneous environments, and where we can make only limited assumptions on their operational environment.

Secure Mobile Agents

The Secure Agent Infrastructure assumes that Mobile Agents are vetted for their adherence to the Secure Agent Infrastructure security policy and only agents that check out are certified for use with the Secure Agent Infrastructure. For threat modeling activities we assume that Mobile Agents in the Core Secure Agent Infrastructure are inviolable to attacks. Only when a Mobile Agent is sent to a DSAP Outpost do we consider threats to it.

Secure Communication Channel

We assume that all data flows between the Core Secure Agent Infrastructure and the DSAP are communicated using an authentic, integrity protected, and confidential channel. We call a communication channel that provides authenticity, integrity, and confidentiality a secure communication channel. The Jini technology underlying the Secure Agent Infrastructure allows for a number of communication mechanisms that can be configured to provide a *secure communication channel* (see Section 2.5.2). For example, Jini supports HTTPS and Transport Layer Security (TLS). TLS with client certificate authentication using a cryptographically secure cipher suite can provide a secure communication channel. Furthermore, we assume that the implementation of the secure communication channel is free of security vulnerabilities, precluding for example buffer overflows that allow remote code execution, the use of insecure cipher suites, or misconfigured certificates.

Distributed Secure Agent Platforms and External Interactors

The DSAPs are the outposts of the Secure Agent Infrastructure. Mobile Agents migrate to DSAP Outposts to be physically close to the entities they interact

with. On the one hand these entities are human users interacting with the Secure Agent Infrastructure to fulfill their disaster response duties, on the other hand these entities are information systems providing useful services and information for disaster response.

There are few assumptions we can make on the security of these DSAP Outpost. In our setting concrete DSAP Outposts can range from first responders using their own private mobile smartphones to well maintained and protected, dedicated systems in both public and governmental organizations.

4.5.4 Threat Modeling Results

Here we present the threats we have derived from our threat model. For a discussion of our findings see Section 4.6.3. We use the Microsoft Threat Modeling Tool 2016 to model the DSAP Outpost and generating a list of threats. Our first attempt generated 72 threats to analyze. Specifying that the connections between the Core Secure Agent Infrastructure and the DSAP use an authenticated, confidential, and integrity protected communication protocol (see Section 4.5.3) reduced the number of threats to 64. We then performed an in-depth analysis of all 64 threats in the DSAP Outpost setting. We have documented this analysis in Appendix A.

In Appendix A we describe every threat, decide if it is pertinent, and we check if and how the threat is represented in the Mobile Agent System security literature. For this, we use the works by Jansen and Karygiannis [JK99, Jan00], Borselius [Bor02], and Bierman and Cloete [BC02] as references. See Section 2.4 for a more detailed discussion of the prior art in Mobile Agent security. Finally, if applicable, we also discuss how well STRIDE-per-interaction is suited to model a particular aspect of the DSAP Outpost.

We present our results concerning agent migration in Table 4.3, the results for Mobile Agent – External Interactor communication in Table 4.4, and the results for Mobile Agent – Core Secure Agent Infrastructure communication in Table 4.5. These tables list the 55 threats we believe relevant to a DSAP Outpost. Each table specifies the threat identifier (ID), the threat type (T), the threat title (Title), the threat identifier generated by the Microsoft Threat Modeling Tool 2016 tool that is used to identify the threat in Appendix A, and finally the novelty (N) of the threat. The ID is just an incremental number. The threat type can be one of the six STRIDE-per-interaction threat categories, that is, *S*poofing, *T*ampering, *R*epudiation, *I*nformation Disclosure, *D*enial-of-service, and *E*levation of privilege. The novelty column gives an indication how well this particular threat was represented in the Mobile Agent System security literature we analysed. Here \uparrow signifies a novel threat, that is at best mentioned in the literature as a generally desirable property, if at all. A \sim signifies that literature acknowledges the existence of the threat, but often in a different context. Finally, \downarrow signifies that this particular threat is well described in literature.

Table 4.3: Threats to agents migrating from the Core Secure Agent Infrastructure to a Distributed Secure Agent Platform Outpost

ID	T	Title	AID	N
<i>Send Mobile Agent</i>				
01	R	The Distributed Secure Agent Platform Repudiates Receipt of a Mobile Agent	58	↑
02	D	Potential Process Crash or Stop for Distributed Secure Agent Platform	59	↓
03	D	The Data Flow <i>Send Mobile Agent</i> Is Potentially Interrupted	60	↓
04	E	Compromised Distributed Secure Agent Platform Uses Received Information to Impersonate Core Secure Agent Infrastructure	61	↑
05	T	A Compromised Distributed Secure Agent Platform Implicates the Core Secure Agent Infrastructure to Have Sent Data to Compromise the Distributed Secure Agent Platform	63	↑
<i>Instantiate Agent</i>				
06	S	Compromised Distributed Secure Agent Platform Gains Full Access to a Mobile Agent	12	~
07	T	Compromised Distributed Secure Agent Platform Tampered with Mobile Agent	14	↓
08	I	Compromised Distributed Secure Agent Platform Eavesdrops (on) Mobile Agent	16	↓
09	D	Compromised Distributed Secure Agent Platform Enacts Process Crash or Stop for the Mobile Agent	17	↓
10	D	Compromised Distributed Secure Agent Platform Interrupts Data Flow <i>Instantiate Agent</i>	18	↓
11	E	Compromised Distributed Secure Agent Platform Impersonates Mobile Agent	20	↓

Table 4.4: Threats to a Mobile Agent communicating with an External Interactor

ID	T	Title	AID	N
<i>Input</i>				
12	S	Compromised Distributed Secure Agent Platform Spoofs the Mobile Agent Process	1	~
13	S	External Entity Spoofs Mobile Agent Process	1	↑
14	S	Compromised Distributed Secure Agent Platform Spoofs the External Interactor	2	~
15	S	External Entity Spoofs the External Interactor	2	↑
16	T	Potential Lack of Input Validation for the Mobile Agent	3	↑
17	T	Compromised Distributed Secure Agent Platform Tamperers With Mobile Agent Input	3	↑
18	R	Compromised Distributed Secure Agent Platform Compromises Mobile Agent to Repudiate Data	4	↑
19	I	External Entity Sniffs Data Flow	5	~
20	I	Compromised Distributed Secure Agent Platform Sniffs Data Flow	5	↓
21	D	Compromised Distributed Secure Agent Platform Enacts Process Crash or Stop for the Mobile Agent	6	↓
22	D	External Entity Interrupts Data Flow <i>Input</i>	7	~
23	D	Compromised Distributed Secure Agent Platform Interrupts Data Flow <i>Input</i>	7	↓
24	E	Cross Site Request Forgery	8	↑
25	E	Compromised Distributed Secure Agent Platform Uses Information Received By Mobile Agent to Impersonate External Interactor	9	↑
<i>Output</i>				
26	S	Compromised Distributed Secure Agent Platform Spoofs the External Interactor to Implicate External Interactor	23	↑
27	S	External Entity Spoofs the External Interactor To Gain Unauthorized Access	23	↑
28	S	Compromised Distributed Secure Agent Platform Spoofs the Mobile Agent	65	~
29	R	External Interactor Potentially Denies Receiving Data	24	↑
30	D	Data Flow <i>Output</i> Is Potentially Interrupted	25	~
31	T	Compromised Distributed Secure Agent Platform Tamperers With Mobile Agent Output	66	↓
32	I	Compromised Distributed Secure Agent Platform Sniffs Data Flow	67	↓
33	D	Compromised Distributed Secure Agent Platform Enacts Process Crash or Stop for the Mobile Agent	68	~

Table 4.5: Threats to a Mobile Agent communicating with the Core Secure Agent Infrastructure using the Distributed Secure Agent Platform’s communication facilities

ID	T	Title	AID	N
<i>MA → DSAP</i>				
34	S	Compromised Distributed Secure Agent Platform Spoofs the Mobile Agent Process	26	~
35	S	Spoofing the Distributed Secure Agent Platform Process	27	↓
36	T	Potential Lack of Input Validation for Distributed Secure Agent Platform	28	↓
37	R	Compromised Distributed Secure Agent Platform Repudiates Receiving Data	29	↑
38	I	Compromised Distributed Secure Agent Platform Sniffs Data Flow	30	↓
39	D	Compromised Distributed Secure Agent Platform Crashes or Stops	31	~
40	D	Compromised Distributed Secure Agent Platform Interrupts Data Flow <i>MA → DSAP</i>	32	↓
41	E	Compromised Distributed Secure Agent Platform Elevation Using Impersonation	33	↓
<i>DSAP → CSAI</i>				
42	D	Data Flow <i>DSAP → CSAI</i> Is Potentially Interrupted	39	~
<i>CSAI → DSAP</i>				
43	R	Potential Data Repudiation by Distributed Secure Agent Platform	40	↑
44	D	Potential Process Crash or Stop for the Distributed Secure Agent Platform	41	↑
45	D	External Entity Interrupts Data Flow <i>CSAI → DSAP</i>	42	~
46	D	Compromised Distributed Secure Agent Platform Interrupts Data Flow <i>CSAI → DSAP</i>	42	↓
<i>DSAP → MA</i>				
47	S	Compromised Distributed Secure Agent Platform Impersonates the Core Secure Agent Infrastructure	47	~
48	S	Compromised Distributed Secure Agent Platform Spoofs the Mobile Agent Process	48	↑
49	T	Potential Lack of Input Validation for the Mobile Agent	49	↑
50	R	Compromised Distributed Secure Agent Platform Modifies Mobile Agent to Enact Data Repudiation by the Mobile Agent	50	↑
51	I	Compromised Distributed Secure Agent Platform Sniffs Data Flow	51	↓

Table 4.5 – continued from previous page

ID	T	Title	AID	N
52	D	Compromised Distributed Secure Agent Platform En-acts Process Crash or Stop for the Mobile Agent	52	↓
53	D	Data Flow <i>DSAP</i> → <i>MA</i> Is Potentially Interrupted	53	~
54	E	Elevation Using Impersonation	54	↑

4.6 Conclusions

4.6.1 Summary

In this chapter we have created threat models for two high level processes pertaining to disaster response: situational awareness and command and control. We have modeled these processes, because they are instrumental to disaster response [OSZW12] and the communication exchanges taking place in these processes form the assets we want to protect. Given that we already know the importance of confidentiality, integrity and availability for these assets according to domain experts [OSZW12], we wanted to identify concrete threats to these assets. We used abstract, high level models first to gain insights into the threats, without cluttering our observations with technical details. We present our conclusions in Section 4.6.2.

Our primary objective for this chapter is to provide a list of threats we can use to evaluate the two security mechanisms forming our security solution we introduce in Chapter 5. For this purpose we created a detail level threat model that encompasses concepts of the Secure Agent Infrastructure for situational awareness. Using this model we have identified a highly repetitive pattern, the Distributed Secure Agent Platform Outpost (DSAP Outpost). The DSAP Outposts form the interface points between the Secure Agent Infrastructure and the outside world. We expect a significant number of DSAP Outposts in any deployment of the Secure Agent Infrastructure. Therefore, we decided to further investigate the threats to the DSAP Outpost. The threats found for the DSAP Outpost form the basis of the security evaluation of our security solution introduced in Chapter 5. In addition, we have also compared our findings with the comprehensive literature on Mobile Agent System security. We present our findings in Section 4.6.3 and the future work we have identified in Section 4.7.2.

We have threat modeled both the situational awareness process and the command implementation process using Data Flow Diagrams to create the models and STRIDE-per-interaction to enumerate the threats. The threat modeling was supported by the Microsoft Threat Modeling Tool 2016. We have gathered our observations and conclusions for both processes in Section 4.6.4 and identified future work in Section 4.7.1.

- By high level threat modeling disaster response activities we have estab-

lished 41 threats to gathering information (situational awareness) and command and control. See Section 4.6.2 for details.

- By refining the situational awareness high level model we identified the highly repetitive DSAP Outpost pattern. We observe that by mitigating threats to the DSAP Outpost we can mitigate a significant portion of the threats to situational awareness and command and control. Therefore, we decided to concentrate on securing the DSAP Outpost (see Section 4.6.2).
- We have created a threat model for the DSAP Outpost. This threat model yielded a list of 54 threats. We use this list to validate the effectiveness of the Trusted Docking Station (TDS) and the Secure Docking Module (SDM). The TDS and the SDM are two security mechanisms we introduce in Chapter 5. We have designed these mechanisms to secure the DSAP Outpost. See Section 4.6.3 for further discussion of this topic.
- The core of the DSAP Outpost is the Distributed Secure Agent Platform (DSAP). The DSAP is a mobile agent platform. We have compared our DSAP Outpost threat modeling results with the comprehensive existing literature on Mobile Agent System security. Of our 54 threats we consider 20 to be novel and not recorded in literature. We present further details in Section 4.6.3.
- We believe Microsoft's threat modeling methodology based on Data Flow Diagram models and STRIDE-per-interaction for threat enumeration to work well for modeling a Mobile Agent System based system. Specifically, the methodology also worked to capture threats to mobile code specific activities such as migrating Mobile Agents. We discuss this in Section 4.6.3.
- We also find Microsoft's threat modeling methodology to work well for capturing the threats to situational awareness and command and control. We base this finding on the fact we we considered almost all threats the Microsoft Threat Modeling Tool 2016 identified for the situational awareness and command and control high level models to be pertinent (see Section 4.6.4). Similarly, we find the methodology to work equally well for the communication aspects of the DSAP Outpost (see Section 4.6.3). We believe this to be rooted in the use of Data Flow Diagrams as a model.
- We observe that Microsoft's threat modeling methodology, as implemented in the Microsoft Threat Modeling Tool 2016, also works well for modeling activities that are not necessarily implemented using an information processing system. With minor manual adaptation of the generated threats we were able to use the Microsoft Threat Modeling Tool 2016 to create high level threat models that abstract almost all implementation aspects. We discuss this in Section 4.6.4.

4.6.2 High Level Threat Models

From O’Neill et al. we know that disaster response communication requires a high degree of confidentiality, integrity and availability [OSZW12]. Specifically, O’Neill et al. note that voice communication requires the highest level of confidentiality, integrity and availability relative to other information types, such as file transfer. With the use of the Secure Agent Infrastructure to support disaster response a portion of disaster response voice communication is replaced with automated computer based information processing. Consequently, the security requirements of voice communication apply to this automated information processing. Therefore, the Secure Agent Infrastructure has to fulfill these confidentiality, integrity and availability requirements.

For the Secure Agent Infrastructure to fulfill the confidentiality, integrity and availability requirements we have to equip it with adequate security mechanisms that mitigate threats to these security goals. However, the results of O’Neill et al. only name security goals, not threats. Therefore, to identify adequate mitigation mechanisms we first need to identify the threats to disaster response communication. Based on the finding of O’Neill et al. that all disaster response communication either pertains to situational awareness or command and control and the function of the Secure Agent Infrastructure we have developed an overall model for threat modeling disaster response communication (see Section 4.2). We have then derived two independent models for gathering information (situational awareness) and command and control. Using Microsoft Threat Modeling Tool 2016, and with manual addition of four threats, we have come up with 41 threats to information gathering and command and control that the Secure Agent Infrastructure needs to mitigate. By further refining the situational awareness high level model to include aspects of the Secure Agent Infrastructure we have identified a repetitive pattern, the DSAP Outpost. The DSAP Outpost is a key element of the all disaster response communication handled by the Secure Agent Infrastructure. Therefore, by mitigating threats to the DSAP Outpost we automatically mitigate a significant portion of the threats to information gathering and command and control.

4.6.3 Using STRIDE-per-interaction for Threat Modeling a Mobile Agent Platform

Our threat modeling efforts have yielded a list of 54 pertinent threats to the DSAP Outpost. We will use this list of 54 threats to analyze the effectiveness of a security solution we devised and which we present in Chapter 5. Towards this we note the overabundance of threats related to a compromised DSAP. Of our 54 threats 43 are directly related to a compromised DSAP. Our solution comprises the TDS and the SDM, and it is geared towards mitigating these threats.

We also compared our findings with the Mobile Agent System security literature. Overall we have identified 54 pertinent threats. Of these 54 threats, we consider 20 as being novel, 14 as being somewhat represented in literature, and 20 as being well known and described in literature. The main reason for the

novel threats is the lack of discussion on repudiation and impersonation threats in our literary sources. A second reason being that only Jansen and Karygianis use a model based approach, and even they only model entities internal to the Mobile Agent System. Borselius does consider external communication, but only in the broadest sense. Bierman and Cloete only study threats that originate from a malicious agent platform. We take the 20 threats we consider novel as an indication that even a well-founded generic discussion of the topic of Mobile Agent security is no substitute for a detailed threat modeling of the concrete system. We believe this is due to the topic being too varied for one discussion to cover all potential security threats in a Multi-Agent System.

Concerning the use of STRIDE-per-interaction to threat modeling the DSAP Outpost we note two interesting facts. First, we used STRIDE-per-interaction to model the Mobile Agent System specific concept of mobile code. Specifically, we used the *Instantiate Mobile Agent* data flow to represent deserialization and instantiation of the Mobile Agent. With this interpretation of the data flow we left the confines of what STRIDE-per-interaction was developed for. Nonetheless, we were able to identify six pertinent threats using STRIDE-per-interaction with the *Instantiate Agent* data flow. We even gained more insight into this particularly well studied area of Mobile Agent System security, than by a pure literary study.

Second, STRIDE-per-interaction was well suited to analyse the Mobile Agent to Core Secure Agent Infrastructure and External Interactor communications. The Data Flow Diagram based model lends itself naturally to represent these communications (see also Section 4.6.2). Also the analysis of the threats pertaining to Mobile Agent communication gave us in-depth understanding of what specific harm a compromised DSAP can inflict on the overall Secure Agent Infrastructure. This knowledge is valuable, because literature is unanimous in identifying a malicious agent platform as the threat agent that is the most difficult to mitigate.

4.6.4 Using STRIDE-per-interaction for Modeling High Level Processes in Disaster Response

Case studies on the applicability and effectiveness of threat modeling using Microsoft's Security Development Lifecycle methodology based Data Flow Diagram models with STRIDE-per-interaction for threat enumeration are still scarce (see Section 3.1.1). Therefore, in addition to deriving the threats we want to mitigate, we also recorded our observations on using the Microsoft Threat Modeling Tool 2016 for establishing these threats. Here we discuss our findings on how well this threat modeling methodology worked for high level disaster response communication in general, and for modeling activities that are not necessarily implemented by software, but for example by human beings.

We find that STRIDE-per-interaction's Data Flow Diagram models are well suited to map the information gathering and command implementation processes in disaster response. Data Flow Diagrams naturally capture the flow of informa-

tion between the data sources, such as external information systems and disaster responders, and the command that has to use this information to decide upon the best disaster mitigation strategy. Conversely, a Data Flow Diagram catches the process of command implementation equally well. As the information flows for gathering information and command implementation are exactly the assets we want to protect, we found almost all threats the tool generates to the data flows pertinent.

However, there are two caveats to using STRIDE-per-interaction. First of all there are artifacts that arise from using a tool created for analysing software to a process where we make as of yet no assumption on how it is implemented. For example the process Information Collation in Figure 4.2 could be conducted by a human being. The effect of this “misuse” is that some threats generated using STRIDE-per-interaction are really only applicable to information systems. Here these threats are “ $\{\text{Process}\}$ May be Subject to Elevation of Privilege Using Remote Code Execution” and “Cross Site Request Forgery”, where $\{\text{Process}\}$ is a variable for an actual process, such as Information Collation or Command Dispatcher. We are not aware of any way to do a privilege escalation through remote code execution on, for example, a human being, however we consider feeding a human wrong data to force a specific outcome possible. Therefore we specifically consider threats of the sort “Elevation by Changing the Execution Flow in $\{\text{Process}\}$ ” pertinent.

On the other hand, some of the threats that STRIDE-per-interaction generates can be naturally adapted to a general process. For example the threat of “Elevation Using Impersonation” is specifically targeted at the impersonation facilities of Microsoft’s Windows OS family, where a software process can take on the guise of another entity to change its access rights (cf. Section A.5.1). However, the underlying principle also holds if you replace the actors with people and the impersonation token, with a password, or even simpler and more applicable to disaster response, a caller name.

Another artifact arising from using STRIDE-per-interaction on an abstract level is that we had to use processes for tasks that are potentially done by humans. This is due to the fact that STRIDE-per-interaction generates different threats depending on the type of the origin or endpoint of a data flow (see Section 2.8.5). If we would not have used a process for information collation in Figure 4.2 and instead would have directly connected the External Interactors we would have only gotten 4 threats.

The second caveat of using STRIDE-per-interaction is related to generalizing data flows. Specifically, we stated that the External Interactors Responder and External Information System are representative of all Responders and External Information Systems connected to the Information Collation process. Furthermore, we have unified all commanding entities under a single Command External Interactor and all executive entities under the single Subordinate External Interactor in Figure 4.4. Now, as we generalized the meaning of the data flows to these entities, we see generated threats in a general context. Although this gives us an overview of the threats we will need to consider, we lack those details that

allow tailoring a security solution.

For example, a fire fighter operational commander giving a situation report to his or her tactical command might have less need for confidentiality, than an assessment that the chemical factory threatened by the wildfire currently holds a stock of chemicals that could potentially create a large scale poisonous cloud capable of killing people, if the wildfire should damage or destroy the chemical tanks. So given the threat list in Table 4.1 and Table 4.2 we know what threats we will have to mitigate, but we have little information that allows us to gauge the strength of the mitigation mechanisms we need for specific information exchanges. For this we need a risk analysis that analyses all instances of potential information exchanges for situational awareness and command implementation in disaster response. We will discuss this further in Section 4.7.1.

4.7 Future Work

We want to highlight two areas for further research. The first is risk modeling for the two high level threat models for situational awareness and command implementation. The second area is enhancing the existing threat model for the Distributed Secure Agent Platform Outpost (DSAP Outpost) we introduced in Section 4.5.

4.7.1 Risk Modelling

We have investigated two high level models relevant to disaster response. The first modeled data flows for gaining situational awareness to inform a command decision and the second modeled implementing command decisions. We intentionally kept these models at a high level of abstraction, and still their analysis revealed a number of pertinent threats.

One aspect we abstracted in our models is the sensitivity of the information dealt with. Although our analysis revealed a number of pertinent threats, what we did not consider is the impact of a threat. To analyze the impact of a threat we need to consider who exchanges information and how important this information is for the overall disaster response. With this information we can then put threats we have identified into context and evaluate their impact using a risk modeling methodology, such as Microsoft DREAD [Sho14].

For the purpose of the risk modeling we propose the following approach, which is based on the approach by O'Neill et al. [OSZW12]. First, develop a concrete disaster response scenario that encompasses a significant portion of disaster response activities. One possible example for this would be to expand the wildfire scenario we use in this thesis, or use the scenario of O'Neill et al. Second, break the scenario down into activities, where each activity maps to a communication exchange between at least two disaster response personnel. Third, and here we deviate from the approach by O'Neill et al., map these activities to the simple threat models we have created for situational awareness and command implementation. Fourth, and here we rejoin the approach of O'Neill

et al., gather a group of domain experts for disaster response. Finally, and here we deviate again from the approach of O’Neill et al., select a risk assessment approach, such as for example Microsoft’s DREAD [Sho14], and evaluate the threats to the activities mapped to the threat models we have introduced in this chapter with the group of domain experts.

We believe that the outcome of this risk modeling will be an increased understanding of which areas of disaster response communication require stronger security mechanisms, and where simpler and cheaper, but also more widely deployable security mechanisms can be used. In Chapter 5 we introduce a security solution, consisting of multiple security mechanisms, that is geared towards wide deployability using COTS hardware, while providing more than “just” communication security. We believe that by risk modeling the disaster response activities as outlined above we could gather concrete evidence on where our security solution can be used, and where stronger mechanisms are in order.

4.7.2 Enhanced Threat Modeling for the Distributed Secure Agent Platform Outpost

We have created a detailed threat model for the DSAP Outpost in Section 4.5. For this model our threat modeling efforts have yielded 54 concrete threats against the DSAP Outpost. Although our model is detailed, it is by necessity generic. We believe that given the ubiquity of DSAP Outposts in the Secure Agent Infrastructure, there will be many different physical manifestations of the DSAP Outpost, thus establishing a variety of system contexts. Haley et al. [HLMN08] state that “System context can have a profound effect on security goals and security requirements.” For example, a first responder might use her private mobile phone as a DSAP Outpost, whereas a DSAP Outpost installed at an infrastructure facility such as a hospital might run on dedicated server hardware. Having different physical manifestations and varying operational environments introduces new threats or limits already identified threats. Let us consider two examples.

First, a first responder’s private mobile phone might simply get stolen, which potentially gives an adversarial entity physical access to the DSAP Outpost. Physical attacks are a difficult class of attacks to protect against. Protection against physical attacks often requires special equipment, such as Security Controllers (see Section 2.13). While a private phone getting stolen is not an unlikely event, someone burglarizing a hospital to steal the DSAP Outpost from the hospital’s server room is somewhat more difficult to envision.

For the second example consider a DSAP Outpost stationed in a hospital and directly connected to the hospital’s information infrastructure. In Section 4.5 we have identified the threat of an external information system being spoofed by an external entity (Threat 15, “External Entity Spoofs the External Interactor”). If in our example the external entity is the hospital’s information infrastructure and the DSAP Outpost is directly connected to it via a cable, then spoofing it seems less likely than for example, if the DSAP Outpost and its corresponding

external entity are in physically separate locations and connected via insecure networks.

We think that it would be worthwhile to study different physical manifestations and operational environments of DSAP Outposts and model the threats accordingly. We believe that the Data Flow Diagram and STRIDE-per-interaction approach we used for our threat models so far only provides limited support for identifying threats pertaining to the operational environment or a particular physical manifestation. Therefore, we propose to conduct this study using attack trees [SSSW98] to supplement our DSAP Outpost threat model. Attack trees are another method for enumerating threats and attack trees help reasoning about threats in a formal and methodical way. Attack trees describe the security of a system based on attacks to the system. The root node of the tree represents the attack goal and subtrees decompose the problem of implementing this attack into different strategies which are represented as leaf nodes. For example an attack tree root node could be “Unauthorized access to emergency services communication network” and a leaf node could be “Steal smart phone with DSAP Outpost”. Similar to the risk modeling approach we outlined in Section 4.7.1 attack tree modeling is best done with domain experts, such as disaster responders, mobile phone security experts, and external information system operators.

We expect the attack tree threat enumeration based on our DSAP Outpost model to grant deeper insights into what threats are applicable in which operational scenario and for what DSAP Outpost physical manifestation. This information can help tailor security solutions to specific classes of operational environments or DSAP Outpost physical manifestations.

5

The Trusted Docking Station and the Secure Docking Module

5.1 Introduction

In Chapter 4 we established the threats to a Distributed Secure Agent Platform Outpost (DSAP Outpost). The DSAP Outposts are the interface points between the Secure Agent Infrastructure on the one hand and human users and external information systems on the other. We established that the overwhelming majority of threats to a DSAP Outpost (43/54) stem from a compromised DSAP Outpost. Here our results are in line with the literature [JK99, Jan00, BC02, Bor02]. A compromised DSAP Outpost can attack the Mobile Agents it executes. These Mobile Agents interact with human users and external information systems to facilitate disaster response. Therefore, we have developed a security solution that mitigates exactly these threats.

As noted in Chapter 4 the security requirements for a specific DSAP Outpost depend on the sensitivity of the information processed by that particular DSAP Outpost. Therefore we regard a one size fits all solution as impractical, as we believe high security solutions for all entities in disaster response will simply be too expensive. Disaster response is a complex process involving both governmental and non-governmental organizations. For example a fire department in a big city might be run by the municipal administration. During disaster response this fire department would work with utility companies, such as a privately owned power company. This fire department would also work with non-governmental organizations like the Red Cross or smaller fire fighter organizations relying on

Declaration of Sources

This section is based on and reuses material from the following sources, previously published by the author:

- [HT09] Daniel M. Hein and Ronald Toegl. An autonomous attestation token to secure mobile agents in disaster response. In Andreas U. Schmidt and Shiguo Lian, editors, *Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec 2009, Turin, Italy, June 3–5, 2009, Revised Selected Papers*, volume 17 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 46–57. Springer, 2009.
- [HTK10] Daniel M. Hein, Ronald Toegl, and Stefan Kraxberger. An autonomous attestation token to secure mobile agents in disaster response. *Security and Communication Networks*, 3(5):421–438, 2010.
- [HTP⁺12] Daniel M. Hein, Ronald Toegl, Martin Pirker, Emil Gattal, Zoltán Balogh, Hans Brandl, and Ladislav Hluchý. Securing mobile agents for crisis management support. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing, STC '12*, pages 85–90, New York, NY, USA, 2012. ACM.

References to these sources are not always made explicit. In particular parts of the Secure Docking Module resource access protocol description, the evaluation, and the conclusion are adapted from [HTP⁺12].

volunteer work. In such a setting, we believe it unlikely that a single universal high-security solution can be deployed. Therefore we geared our DSAP Outpost security solution towards providing a significant degree of security, while using commercial off-the-shelf components. Our idea was to provide a solution that can be widely deployed, even on otherwise privately or commercially used hardware, while providing sufficient security for a wide range of disaster response use cases.

In this chapter we introduce our security solution for protecting DSAP Outposts. This security solution consists of two components, the Trusted Docking Station (TDS) and the Secure Docking Module (SDM). The TDS is a commercial off-the-shelf personal computer system that uses Trusted Computing functionality to provide a load-time integrity protected execution environment for the Distributed Secure Agent Platform (DSAP) and the Mobile Agents it hosts (see Section 2.15.3). The TDS is based on the acTvSM platform [Pir15] (see Section 2.16). The SDM is a small hardware security token capable of verifying the load-time integrity of a TDS, while also establishing the presence and iden-

tity of a human user. The SDM is an access control security mechanism and provides the *resource access protocol*. The resource access protocol mediates access to protected credentials, such as cryptographic keys, pinned certificates, or name/password combinations. The SDM only grants access to these credentials if the load-time integrity of the platform requesting them and the presence and identity of an authorized user have been established.

Our contribution is fivefold. *First* we have designed the SDM, including the cryptographic resource access protocol that establishes the load-time integrity of its host platform using Trusted Computing functionality and the presence of a human user using a shared secret. *Second* we have implemented the SDM comprising the resource access protocol and the required support facilities. We have implemented the SDM using a Security Controller (see Section 2.13). *Third* we have integrated the DSAP with the SDM. *Fourth*, we have integrated the DSAP with the acTvSM platform to create the TDS. *Finally*, we have evaluated the performance and the security of the TDS/SDM security solution.

5.2 Objectives

The threats revealed by our threat modeling efforts (cf. Chapter 4) can be divided into two categories. The first category contains threats related to a compromised Distributed Secure Agent Platform (DSAP). The second category comprises threats clustered around an adversary impersonating an External Interactor, such as a human user or an information system. Of the 54 threats we identified to a Distributed Secure Agent Platform Outpost (DSAP Outpost), 43 are related to a compromised DSAP. The remaining threats pertain to an adversary impersonating an External Interactor and the follow up threats that arise, such as information disclosure, injection of tampered data, and Denial-of-Service.

We have devised a security solution that mitigates both of these threat categories. The security solution consists of the Trusted Docking Station (TDS) paired with a Secure Docking Module (SDM). In conjunction, these two components mitigate the threats posed by a compromised DSAP and threats that arise from failure to authenticate the involved parties.

Besides mitigating the threats identified in Chapter 4, our objectives for our security solution were wide applicability, usability, and availability.

Applicability Given our disaster response setting and the heterogeneous landscape of disaster response organizations we wanted to develop a solution that could be used even by volunteers on their private hardware, as long as this hardware provides some supporting security technologies.

Usability Given its use in disaster response we wanted to create an unobtrusive solution. We assume there is little time during disaster mitigation for users to react to security related alarms or prompts.

Availability Finally, above all, the system has to be available. From our point of view wide applicability and usability are just other aspects of availability. Although, availability is considered in the threats we mitigate, this deserves further discussion. Availability of communication, and in our case by extension to the Secure Agent Infrastructure, is key to successful disaster response. For example, one of the key selling points of a Mobile Agent System for disaster response, is that Mobile Agents can mitigate temporary communication outages. Therefore, when designing our security solutions we have provided fallback modes that allow operating a DSAP Outpost, even in the face of potential security policy violations.

5.3 Mode of Operation

The key idea of our security solution is to limit access to the Secure Agent Infrastructure to platforms that can provide evidence of their load-time integrity and authenticity and, if applicable, of the authenticity of their users. To achieve this goal we bind access to the Secure Agent Infrastructure to cryptographic keys, and make those keys only available to platforms and users that can provide evidence of their integrity and authenticity.

We distinguish two modes of operation for our security solution. Which mode of operation to use depends on the environment where our security solution is deployed. Specifically, the key difference is if a human being is involved or not. We use Figure 5.1 to illustrate our discussion of the two modes of operation.

In Chapter 4 we elaborated that we want to protect the DSAP Outposts and that there are two general usage scenarios for DSAP Outposts.

The first scenario is when a DSAP Outpost is attached to an external information system. The idea here is that even when the connection to the Core Secure Agent Infrastructure is temporarily unavailable, an Mobile Agent running on the DSAP Outpost can finish its tasks and wait for the connection to be reestablished. The key observation here is that in this mode there is no human personnel involved. The Mobile Agent migrates to the DSAP Outpost, performs its tasks involving the external information system the DSAP Outpost is attached to, and finally sends its results home to the Core Secure Agent Infrastructure. Therefore, for this scenario, we propose to use the Trusted Docking Station (TDS) alone in conjunction with *sealing* (see Section 2.14) to protect the access credentials.

In this scenario, when the TDS goes online to provide a DSAP Outpost to the Secure Agent Infrastructure, the cryptographic keys that protect access to the Secure Agent Infrastructure will be released to the TDS, iff the DSAP Outpost is in a specific software configuration and the Trusted Platform Module (TPM) (see Section 2.14) that provides this security mechanism is present. Concerning availability, we do not make any assumptions on whether the DSAP Outpost is permanently online. Thus, if a particular DSAP Outpost is offline during disaster response, it might be that the system is simply not booted, or that the unsealing operation failed. In both cases, the Core Secure Agent Infrastructure will detect

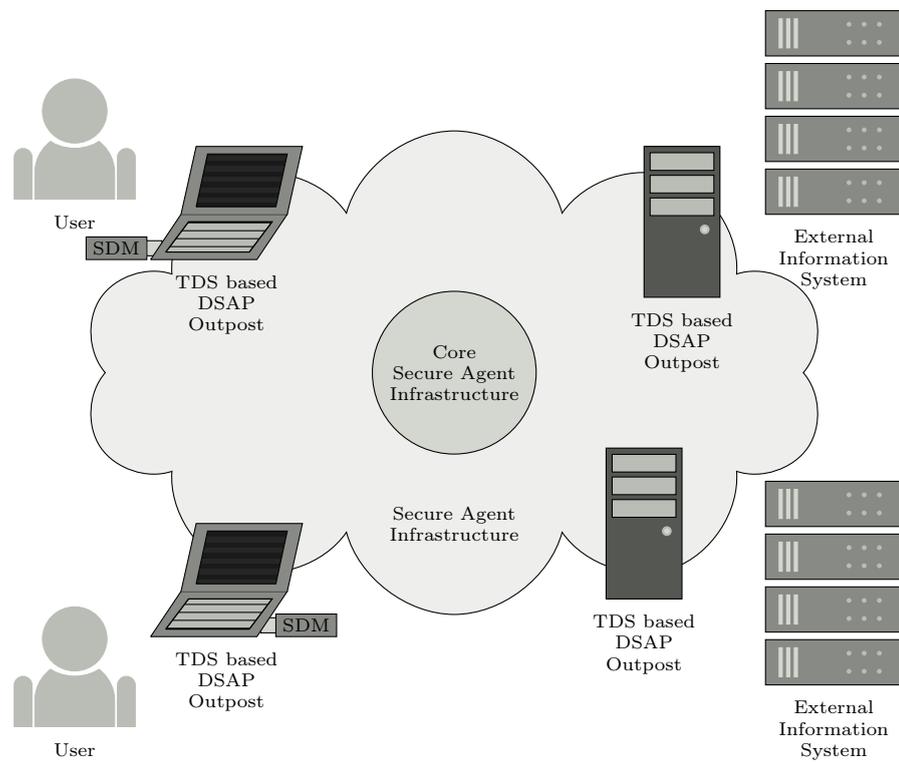


Figure 5.1: Usage scenarios for TDS and SDM in conjunction with a Distributed Secure Agent Platform Outposts (DSAP Outposts)

that this particular DSAP Outpost is not operational, and it can use other means to establish communication with the DSAP Outpost operator. For example the Core Secure Agent Infrastructure can flag disaster response personnel to simply call the DSAP Outpost operator and obtain the information from that particular external information system via voice communication.

The second scenario we consider is when the DSAP Outpost is used by a human user, for example a first responder in the field. Here we propose to use the TDS in conjunction with the Secure Docking Module (SDM). The SDM acts as an access control mechanism that establishes the load-time integrity of the user's platform, communicates the fact that the load-time integrity has been established to the user, and requires the user to enter a shared secret to proof his or her presence. Thus, the user's DSAP Outpost will only start operating, if there is sufficient evidence of user presence and authenticity and platform integrity and authenticity. For availability, we planned for a fallback mode in which the user can use the DSAP Outpost with a set of secondary cryptographic keys. These keys are only bound to the user and not to the TDS's load-time integrity. Thus the user can decide to use the system, even if its integrity is compromised. On the other hand the Core Secure Agent Infrastructure can detect that the user's system is in fallback mode on connection and apply a different security policy when communicating with this user's DSAP Outpost. Finally, to make the system unobtrusive, there is the option that the user only has to authenticate herself to the SDM once, when she boots her TDS. Note that this can happen, for example, while the user is still on her way to the disaster response operation and thus does not impact disaster response operation itself.

5.4 Application

Our security solution comprising the Trusted Docking Station (TDS) and, optionally, the Secure Docking Module (SDM) controls access to credentials, such as keys or passwords. Here, we discuss how this capability can be used by describing *three applications* for our security solution in the context of protecting a Distributed Secure Agent Platform Outpost (DSAP Outpost). These three applications are *communication key protection*, *authentication credential protection*, and *agent authorization key protection*. We have prototyped one of these applications, the agent authorization key protection, for the performance evaluation (see Section 5.8).

5.4.1 Communication Key Protection

In Section 4.5.3 we stated that we assume a secure communication channel between the Core Secure Agent Infrastructure and a DSAP Outpost. In Section 2.6.10 we discussed that a secure communication channel should be easy to add to the Secure Agent Infrastructure by using Apache River's (formerly Jini, see Section 2.5) new support for secure communication protocols such as

HTTPS. If HTTPS is used, then the private authentication key of a DSAP Outpost could be protected by our security solution.

Our security solution provides two protection mechanisms for the private DSAP Outpost authentication key. The first protection mechanism is sealing (see Section 2.14) the key to the TDS. The second protection mechanism is storing the key in the SDM. Both mechanisms provide authentic and confidential credential storage where access is bound to a platform's software configuration. The SDM mechanism additionally provides a key-to-user binding and can verify the presence of a user through password verification.

Both mechanisms can also be used for storing credentials that require only authentic storage. In the HTTPS setting this applies to the credentials that ensure the Core Secure Agent Infrastructure server authenticity. Specifically, our mechanisms can protect the root certificates of the Public Key Infrastructure (see Section 2.6) that authenticates the Secure Agent Infrastructure components.

5.4.2 Authentication Credential Protection

The second application is protecting the authentication credentials used for authenticating External Interactors, such as external information systems or disaster response personnel, to the TDS, and vice-versa. The SDM was specifically developed for authenticating human users to the TDS, and vice-versa, in addition to ensuring the presence of the human user. We have discussed this particular application of our security solution in Section 5.3. This leaves the communication between a DSAP Outpost and an external information system to consider. Our security solution lends itself to protect the authentication credentials for this communication. For example, if the external information system supports it, TLS with client certificate authentication can be used to establish a secure communication channel with mutual authentication. Again, the private authentication key of the DSAP Outpost and the public verification key of the external information system could be protected by either sealing them or by storing them in the SDM.

5.4.3 Agent Authorization Key Protection

The third application we discuss here is agent authorization. Here we want to ensure that only an authorized DSAP Outpost is able to execute a particular Mobile Agent. Mobile Agents can contain valuable Intellectual Property (IP) and additional credentials otherwise not available on a specific DSAP Outpost. Therefore, in addition to a secure communication channel, Mobile Agents are encrypted for a specific DSAP Outpost using public key cryptography. The capability of decrypting this agent is bound to a specific DSAP Outpost software configuration. This binding is ensured by either sealing the decryption key to the TDS or storing it on the SDM. This is also the application we implemented for the performance evaluation of our security solution described in Section 5.8.

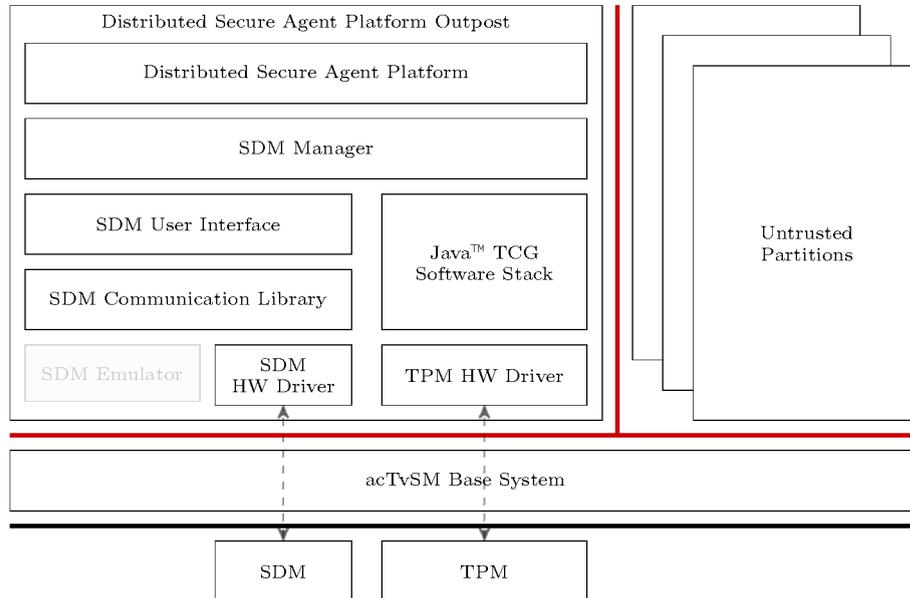


Figure 5.2: The Trusted Docking Station architecture. The red lines delineate Virtual Machine against other Virtual Machines and against the acTvSM Base System Virtual Machine Monitor. The black line separates hardware from software components.

5.5 Architecture

We have prototyped our security solution using the acTvSM platform [Pir15] to implement the Trusted Docking Station (TDS) and extend it further with the Secure Docking Module (SDM). The acTvSM platform uses Intel's TXT (see Section 2.15) to create isolated software execution environments. Furthermore, the acTvSM platform ensures the load-time integrity (see Section 2.15.3) of the base platform and selected compartments software configuration. Finally, the acTvSM platform can provide evidence of this load-time integrity to a party external to the acTvSM platform.

Our architecture depicted in Figure 5.2 stipulates running the Distributed Secure Agent Platform Service (DSAP Service) in an isolated acTvSM platform compartment. These compartments are implemented as application Virtual Machines. We call these application Virtual Machines Virtual Appliances. For every application executed on the acTvSM platform, the platform launches a Virtual Appliance complete with its own operating system (OS) executing the application.

We have named the DSAP Service Virtual Appliance the DSAP Outpost Application Virtual Machine (Outpost Appliance). The Outpost Appliance consists of DSAP Service itself and the TDS and SDM support components. The

TDS support components consist of the jTSS¹ and the Trusted Platform Module (TPM) hardware driver (TPM HW Driver). The SDM support components consist of the SDM Manager that uses the SDM User Interface and the jTSS² to communicate with the SDM and the TPM. The SDM User Interface relies on the SDM Communication Library to implement the actual communication to the SDM. The SDM Communication Library implements a plug-in system for actual SDM implementations. We have implemented two SDM implementations, an SDM Emulator written in software and Security Controller based hardware implementation of the SDM.

We now describe each of these components and how they interact in detail.

The Distributed Secure Agent Platform Service The DSAP Service implements the Distributed Secure Agent Platform (DSAP) component as described in Section 2.6.3. Here we briefly reiterate its key properties. The DSAP Service is written in Java and implements the execution environment for Mobile Agents. The DSAP Service supports agent migration, in the sense that it can receive and then execute Mobile Agents. It also provides communication services that allow the Mobile Agents to exchange messages with their home platform, that is, the Process Management Subsystem in the Core Secure Agent Infrastructure (see Section 2.6 and Section 2.6.2 for details). The DSAP software directly integrates with the SDM Communication Library to use the SDM and the jTSS TCG software stack to communicate with the TPM.

The SDM Manager The SDM Manager is a software component written in Java for administering the SDM. It integrates with the SDM Communication Library to exchange messages with the SDM and it relies on the jTSS TCG software stack to use the TPM. The SDM protects resources and only releases these resources to the TDS when the resource access protocol, as described in Section 5.7.4, is executed correctly. To execute the protocol successfully, it requires a number cryptographic keys and shared secrets to be set up in advance. The SDM Manager is the component that implements this setup.

The jTSS Java™ TCG Software Stack The jTSS TCG software stack is a pure Java implementation of the TCG Software Stack (TSS) Specification [Tru07a]. The jTSS software stack provides a standardized interface to use a TPM. Our security solution uses the TPM to record and report the TDS software configuration. The jTSS provides the functionality to setup and use platform configuration reporting. For example, the TPM signs a platform software configuration report using an Attestation Identity Key. This Attestation Identity Key has to be created before it can be used for this purpose. This is a task implemented using the jTSS. In addition the jTSS is also used to create the platform software configuration reports.

¹<http://trustedjava.sourceforge.net/index.php>

²<http://trustedjava.sourceforge.net/index.php>

The SDM User Interface The SDM User Interface is a command line utility that uses the SDM Communication Library and the jTSS software stack to execute the resource access protocol. We have implemented this command line utility for test purposes and as fallback SDM interface for software that does not support the Java™ interface.

The SDM Communication Library The SDM Communication Library is the heart of all SDM software support components. To its users, such as the SDM Manager or the DSAP software, it provides a simple application programming interface (API) for using the resource access protocol to obtain a protected resource, or administering the SDM.

Communication with the SDM is session based. In general, a session needs to be established before commands can be sent to the SDM. The SDM sessions are cryptographically protected. The session establishment protocol is detailed in Section 5.7.4. The only commands that can be sent to an SDM without a session are those commands that initialize a freshly minted SDM to the point where it can establish a cryptographically protected session. Afterwards, these commands are disabled. Furthermore, the SDM supports two different kinds of sessions, a user session and an administration session. A user session grants access to the resource access protocol, whereas an administration session actually allows to configure an SDM. In this thesis we will restrict ourselves to the user session and using the SDM. SDM administration has been investigated by Danner et al. [DH10, DHK10].

Once the TDS has established a session with the SDM, the SDM can receive commands. The SDM Communication Library translates high level commands sent via the API into binary data structures that an actual SDM implementation can interpret. These commands are then sent to an SDM implementation. The SDM Communication Library supports different SDM implementations via a plug-in system. The plug-in system is configured at runtime.

The SDM Emulator We developed the SDM Emulator as an executable specification of the SDM in software. We used the SDM Emulator to specify the SDM and to implement and integrate the components that require an SDM before we had an actual SDM hardware implementation. The SDM Emulator only appears in Figure 5.2 to illustrate the SDM Communication Library's ability to support different SDM implementations. We strongly suggest not to integrate and use the SDM Emulator into a production Outpost Appliance.

The SDM Security Controller Implementation We use the SDM to authenticate a user, and part of the authentication is the fact that the user actually possesses an SDM. We have implemented a prototype of the SDM using a commercially available Security Controller. We discuss the implementation in detail in Section 5.7.5.

5.6 The Trusted Docking Station

The Trusted Docking Station (TDS) is based on the acTvSM platform [Pir15], which we described in Section 2.16. The acTvSM platform enforces integrity guarantees on itself as a software platform, as well as, the applications and the services it hosts. In addition to these integrity guarantees it leverages strong hardware-based memory isolation to separate applications into partitions. We operate the DSAP Outpost Application Virtual Machine (Outpost Appliance) in one of these partitions. We use the acTvSM platform as available on this website³.

5.6.1 The Outpost Appliance

The Outpost Appliance is a Virtual Appliance based on the Outpost Appliance image. The Outpost Appliance image comprises all software components necessary to operate the Distributed Secure Agent Platform Outpost (DSAP Outpost). We have discussed the architecture of the TDS as a whole, and the DSAP Outpost in particular, in Section 5.5. The Outpost Appliance image is a read-only virtual machine image based on a customized Debian GNU/Linux major version 5 (Lenny). We included Java-support and all the components described in Section 5.5, with the marked exception of the SDM emulator. All DSAP Outpost software components are implemented in Java™. In order to create the read-only virtual machine image using Debian tools we also packaged the Secure Docking Module (SDM), jTSS, and the Distributed Secure Agent Platform Service (DSAP Service) components itself as packages compatible with the Debian package manager.

Furthermore, we configured the acTvSM base system to forward the Trusted Platform Module (TPM) to the Outpost Appliance during runtime, so that the DSAP Outpost can use the TPM's integrity reporting features. In addition, we also configured the acTvSM base system to forward the Security Controller implementation of the SDM to the Outpost Appliance. This enables the TDS to use the SDM. Our Security Controller based prototype of the SDM provides a USB interface that allows sending and receiving ISO7816 Application Protocol Data Units to and from the SDM prototype. Finally, we configured the acTvSM base platform to automatically launch the Outpost Appliance on boot.

With our setup we can install the acTvSM base platform and the Outpost Appliance in parallel to any operating system that can be booted by the GNU GRUB boot loader⁴ used by the acTvSM system. Whenever the user reboots her device she can choose if she wants to go into disaster response mode, or boot her normal OS.

Although the main Outpost Appliance image itself is read-only, the DSAP Outpost needs storage for data created at runtime, for example for the Secure Docking Module Authentication Key (SAK) (see Section 5.7.4) or for temporary

³<http://trustedjava.sourceforge.net/index.php?item=actvsm/readme>

⁴<https://www.gnu.org/software/grub/>

files created by Mobile Agents. These files are persistently stored in a cryptographically protected logical volume managed by the acTvSM base platform. The decryption key for this volume is sealed (see Section 2.14) to the base platform software configuration. This writeable logical volume is merged with the Outpost Appliance read-only image at runtime by the acTvSM base platform. To the DSAP Outpost the process of overlaying a writeable file system on top of the read-only Outpost Appliance image is completely transparent.

Note that the Secure Block Device we introduce in Chapter 6 provides a cryptographically protected Datastore. We specifically developed the Secure Block Device, because we identified the need for a Single-User Authentic Block Datastore (see Section 2.10) when implementing a Mobile Agent System for disaster response on a platform running ANDIX OS (see Section 2.18).

When the acTvSM base platform boots, it measures the integrity of all software involved in the boot process up to, and including, itself. Once the boot process of the acTvSM base platform is finished, we have configured the acTvSM base platform to automatically load the Outpost Appliance image. This image is also measured before it is started and the measurement stored inside the platform's mandatory TPM. However, the writeable overlay image is not measured.

5.6.2 The Distributed Secure Agent Platform Software

In collaboration with the developers of the Secure Agent Infrastructure, we have modified a version of the DSAP Service to take advantage of our TDS and SDM based security solution. Specifically, we have rewritten the DSAP Service to use cryptographic keys provided by either the TDS alone or by the TDS and SDM in combination. The authors of the Secure Agent Infrastructure had already fitted the Secure Agent Infrastructure with a facility to encrypt Mobile Agents for a specific target DSAP Service before sending the Mobile Agent to this DSAP Service. The Mobile Agent encryption is done using an asymmetric key encapsulation scheme with a symmetric bulk encryption key (see Section 2.6.8). In conjunction with an underlying secure communication channel (see Section 4.5.3 and Section 2.6.10), encrypting agents for a specific DSAP Service using asymmetric cryptography to identify a particular DSAP Service can act as an access control scheme. In this scheme only DSAP Services that can actually decrypt their agents can partake in the disaster response effort. Access to the decryption key is protected either by sealing it to the TDS using the TPM, or protecting it with an SDM, depending on the deployment scenario. We have implemented protecting a DSAP Service's Mobile Agent decryption key using the TDS in combination with the SDM to evaluate our security solution. In addition, we strongly advocate adding a secure communication channel to the DSAP Outpost and using the TDS/SDM to protect the authentication key for this secure communication channel. We discuss this further in Section 5.11.

5.7 The Secure Docking Module

5.7.1 Overview

The Secure Docking Module (SDM) is a pluggable security module that protects authentication credentials, such as cryptographic keys. As described in Section 5.4, a Distributed Secure Agent Platform Outpost (DSAP Outpost) can use this functionality for several applications. We have implemented and evaluated the agent authorization key protection application detailed in Section 5.4.3.

The SDM only releases protected resources if the DSAP Outpost running on the Trusted Docking Station (TDS) is in an authorized software configuration. The SDM verifies the software configuration. In addition, the SDM also provides authentication of the entity using the device, for example a first responder. The SDM verifies all these conditions as part of its *resource access protocol*.

We describe the resource access protocol in Section 5.7.4. For a protocol run to succeed, the protocol requires a number of preconfigured parameters to be set up on both the SDM and the DSAP Outpost hosting the SDM. We detail this setup in Section 5.7.2. Finally, we describe our implementation of the SDM in Section 5.7.5

5.7.2 The Resource Access Protocol Setup

The SDM implements a number of protocols for communication with its DSAP Outpost host platform. We investigate two in Section 5.7.4, the titular resource access protocol and the *session establishment protocol*. The session establishment protocol establishes a session between the host platform and the SDM. The session establishment protocol needs to be run, before the resource access protocol can be executed. Once a session is established, an arbitrary number of resource access protocol runs can be executed as part of the established session.

We have detailed the configuration parameters that need to be set up before the first run of the session establishment protocol in Table 5.1. Furthermore, we describe the configuration parameters that need to be set up before the resource access protocol can be run in Table 5.2. The use of these configuration parameters is described in the next section.

The question of gathering, collating, and managing the SDM and host platform configurations has been treated by Danner et al. [DH10, DHK10].

5.7.3 The Session Establishment Protocol

The purpose of the session establishment protocol is to mutually authenticate the TDS and SDM to each other and derive an ephemeral session key to encrypt their communication. The goal of the encryption is to prevent leaking the resources stored on the SDM when sending them to the TDS. The session establishment protocol is a version of the Needham-Schroeder-Lowe [NS78, Low95] protocol for mutual authentication and ephemeral key derivation. The exact protocol is illustrated by Figure 5.3.

Table 5.1: The configuration parameters required by the Secure Docking Module's session establishment protocol

Shorthand	Name	Description
H_I	Host identifier	A string identifying the host platform to the SDM
S_I	SDM identifier	A string identifying the SDM to the host platform
SAK	SDM Authentication Key	A 2048-bit RSA key pair that authenticates the SDM to a host platform
HAK	Host Platform Authentication Key	A 2048-bit RSA key pair that authenticates the host platform to the SDM

Table 5.2: The configuration parameters required by the Secure Docking Module's resource access protocol

Shorthand	Name	Description
ID_R	Resource identifier	A string identifying the resource to release to the host platform
R_I	Platform Configuration Register indices	The set of Platform Configuration Register indices that need to be quoted by the host platform to release the resource
AIK	Attestation Identity Key	A 2048-bit RSA key pair that authenticates a TPM Quote to the SDM
A_I	Authentication identifier	A pre-shared secret demonstrating the host platform's authorized state to its user
PWD	User password	A pre-shared secret authenticating the user to the SDM

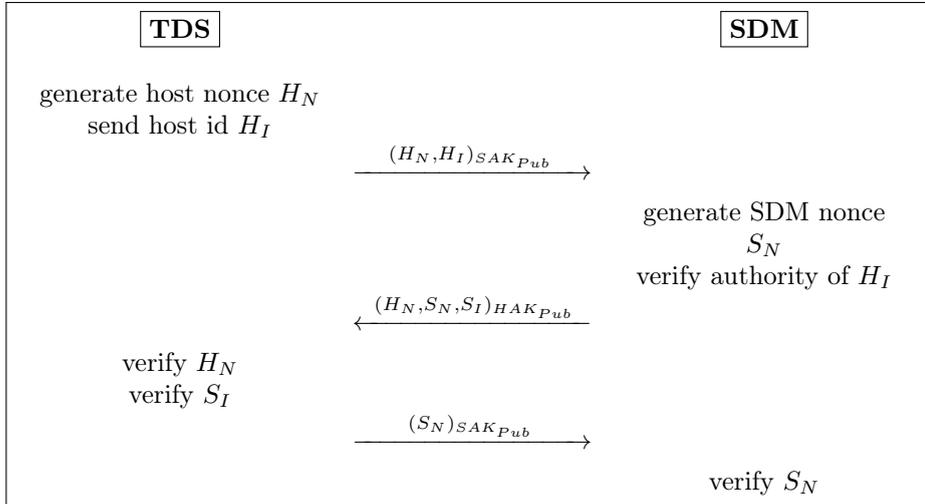


Figure 5.3: Secure Docking Module session establishment protocol

The protocol uses two pre-shared 2048-bit RSA keys to authenticate both parties and to create and exchange the ephemeral AES session key. These keys are the Secure Docking Module Authentication Key (SAK) and the Host Platform Authentication Key (HAK). Every SDM has a unique SAK that is generated by the SDM upon first-time initialization. The private part of the SAK never leaves the SDM. Therefore, the SAK can be used in a challenge response protocol to corroborate the identity of the SDM. Every TDS has a unique identification key, the HAK.

The authentication and key exchange protocol works as follows. The TDS randomly generates a nonce H_N . The nonce and the TDS identifier H_I are encrypted with the SDM's SAK, and communicated to the SDM. The SDM decrypts the package, using the TDS identifier H_I to lookup the Host platform's public HAK, generates a nonce S_N , and encrypts it, together with the SDM's identifier S_I using the HAK. The TDS in turn decrypts the message, verifies the nonce H_N , and verifies the SDM identity S_I . If both values are correct, the TDS has now established the identity of the SDM. Both parties now independently derive the session key by computing the SHA-1 hash of both nonces and using the lower 16 bytes as Advanced Encryption Standard (AES) session key. The session key is used with AES in Cipher Block Chaining (CBC) mode of operation to encrypt the communication between the TDS and the SDM.

5.7.4 The Resource Access Protocol

The SDM provides two critical security services to the Secure Agent Infrastructure and Secure Agent Infrastructure users. It authenticates platform users and it authenticates a platform and its platform software configuration, without re-

quiring online access to any Secure Agent Infrastructure services. The SDM authenticates a user through possession of the SDM and knowledge of a shared secret stored in the SDM. We call this shared secret the user shared secret. The SDM authenticates the platform and its software configuration by verifying Trusted Platform Module (TPM) attested platform configuration reports. The SDM signals the fact that platform authentication and software configuration verification was successful by releasing a second shared secret, the platform authentication identifier, to the platform. The platform can then present the platform authentication identifier to the user as proof of its identity and software configuration. The SDM only releases resources it protects to the host platform, iff the platform software verification and user shared secret verification are completed successfully. In the Secure Agent Infrastructure, every SDM is paired with a TDS. Thus all SDM host platforms are TDSs.

The SDM's resource access protocol first verifies the TDS' software configuration by verifying the TPM attested platform software configuration report. For this, the SDM's resource access protocol integrates the TPM's remote attestation protocol (see Section 2.14) for verifying a platform's configuration. The reason for first establishing the TDS' platform configuration and thus by extension the TDS' load-time integrity is that it provides evidence about the existence of a trusted I/O path to the user (see Section 2.15). The trusted I/O path is used to show the platform authentication identifier to the user and request the user shared secret from the user. After verifying the TPM's attested platform software configuration report, the SDM's resource access protocol sends the platform authentication identifier to the host and then awaits the user password. With the user password the SDM verifies the presence of a human user. The SDM only discloses the requested resource, if the shared secret is correct. We present the protocol for this process in Figure 5.4.

The SDM is a passive device. The SDM does not initiate a resource access protocol run and, even while the protocol is executed, it will only react to protocol messages send by the TDS in a request/response fashion. Thus, the SDM resource access protocol is always started by the TDS by sending a host identifier H_I and a resource identifier ID_R to the SDM. However, communication with the SDM is session based and to execute the resource access protocol the host platform first needs to establish a session with the SDM. The session establishment protocol provides a confidential channel between the SDM and the host platform.

The TPM remote attestation protocol is implemented in the TPM_Quote2 operation of the TPM. The TPM_Quote2 takes a set of Platform Configuration Register (PCR) indices R_I and a nonce n as input. It generates a compound hash CH_{PCR} by hashing the contents of the PCRs indicated by R_I . The TPM_Quote2 command then signs CH_{PCR} together with the nonce n . This $(n, CH_{PCR})_{AIK_{pri}}$ message is also called TPM Quote. The TPM signs this report using a private Attestation Identity Key created and protected by the TPM. The Attestation Identity Key is an RSA key pair, where the private key never leaves the TPM unencrypted.

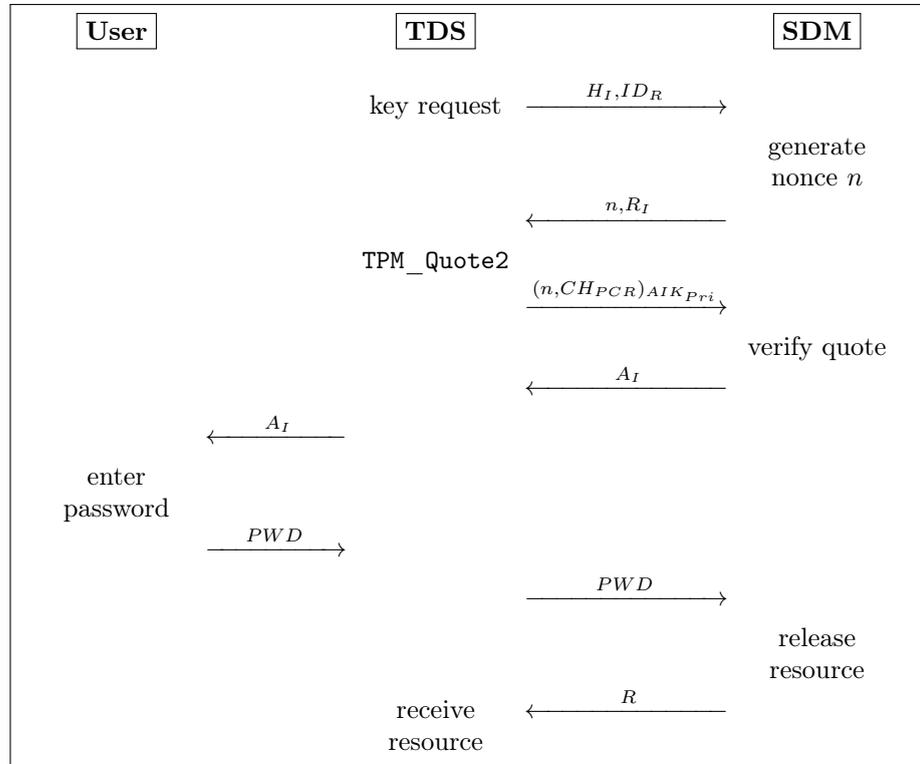


Figure 5.4: Secure Docking Module resource access protocol

Upon receiving a resource access request consisting of the H_I and R_{ID} tuple, the SDM retrieves the PCR indices R_I associated with (H_I, R_{ID}) from its internal data store and generates a nonce n . The SDM then responds to the resource access request by sending the (R_I, n) tuple to the TDS. The TDS then executes the `TPM_Quote2` operation with the SDM sent R_I and n as input. The TDS then forwards the result of the `TPM_Quote2` operation to the SDM.

The SDM verifies TPM Quotes in a 3 step process. First the SDM verifies the signature on the report using the Attestation Identity Key for H_I . Second the SDM ensures that the nonce is the same as the one sent to the TPM in the previous response. Finally, the SDM tests if the CH_{PCR} is contained in an internal database of authorized compound hash values for H_I . The SDM only proceeds with the resource access process, if all 3 checks succeed. If all 3 checks succeed the SDM has been given sufficient evidence (a) that the platform configuration originates in an authentic TPM, through the Attestation Identity Key signature, (b) that the TPM Quote is fresh and not a replay, through the nonce n , and (c) of the platform configuration itself, via the compound hash of the PCRs CH_{PCR} .

In the next step, the SDM authenticates the user. For this the SDM first

sends an authentication identifier A_I to the TDS. The authentication identifier is a predetermined shared secret between the user and the SDM, and unknown to the TDS. A TDS in an authorized software configuration must immediately delete the shared secret after displaying it to the user. Because the SDM immediately terminates the resource access process, if the TDS's platform software configuration is not authorized, the secrecy of A_I is protected. The TDS's ability to display the shared secret demonstrates the TDS's authorized software configuration to the user. If the TDS software configuration is not authorized, the TDS cannot present the shared secret to the user, because the SDM will not disclose it. Failure to produce the shared secret signals the user that the TDS might be compromised.

Upon being prompted for the authentication identifier A_I the user then enters her password PWD to authenticate herself. The password is then transmitted to the SDM for validation. If the password is correct, the SDM will grant access to the resource R .

5.7.5 Secure Docking Module Implementation

Previous proposals by Fournaris [Fou10] and Hein [HTK10] called for a specialized hardware implementation of the SDM. However, we implement the SDM functionality in software on a commercially available Security Controller (SC). This SC implements an architecture that is highly resilient to implementation attacks.

The SDM functionality is implemented as software on the Security Controller. The SDM functionality is divided in communication interface, cryptographic cores (RSA, AES, SHA-1), data management, and protocol implementation.

The SDM communicates with the host platform using the ISO7816 interface. Although the SDM supports a number of physical communication interfaces, the smart card interface was chosen for its simplicity and inherent compatibility with Java. The SDM uses custom Application Protocol Data Units to exchange data with the host platform.

The SDM chip natively supports the AES, SHA-1 and RSA cryptographic algorithms required to implement the SDM functionality.

The SDM provides resources, such as cryptographic keys and authorization credentials. These resources are protected by the resources access protocol. Execution of the access protocol requires cryptographic keys, authorized platform configurations, authentication identifiers and the protected resource itself. The SDM uses the on-chip NVRAM to store this information in a simple file system structure. The data management component is capable of supporting up to 4 host platforms with up to 16 KiB of resources per host. Each resource can have up to 20 authorized platform configurations, one user shared secret, and one authentication identifier.

The SDM protocols for session establishment, administration and resource access are implemented as software modules running on the main CPU of the SDM chip.

Our hardware prototypes of the SDM combine the security controller with a USB smart card reader. We have sealed both components in an epoxy filled casing for physical protection. Figure 5.5 depicts the security controller in the front and the assembled prototype in its epoxy sealed casing in the back.



Figure 5.5: Hardware implementations of the Secure Docking Module: the security controller (front) and the assembled token prototype with USB interface in an epoxy sealed casing (back).

5.8 Performance Evaluation

Our security solution for protecting the Distributed Secure Agent Platform Outpost (DSAP Outpost) comprising the Trusted Docking Station (TDS) and Secure Docking Module (SDM) incurs overheads. Specifically, certain operations require more time and additional user interaction. The TDS running a DSAP Outpost is intended to be booted up once in case of a crisis, and then stay active during the course of disaster response. For this reason operations can be split up into two categories, those that are performed once, such as platform boot up, and those that can be performed repeatedly, such as requesting an SDM protected resource. The SDM protected resource we use for our performance evaluation is the agent authorization key (see Section 5.4.3).

Table 5.3 presents the time required by the security architecture related agent platform operations. *Boot TDS platform* is the time required to boot up the integrity protected TDS platform. *Start DSAP Service* is the time necessary to boot the integrity protected DSAP Outpost service on the TDS. In comparison, *Start DSAP Service* is the time required to start the Distributed Secure Agent Platform Service (DSAP Service) without integrity protection. The overhead for starting the integrity protected DSAP Service is the time required to boot the TDS added to the time required to start the integrity protected DSAP

Table 5.3: Trusted Docking Station/Secure Docking Module performance evaluation results. The access SDM resource action consists of the TPM_Quote2, quote verification, copy resource to host, and miscellaneous overhead table entries.

Action	Recurring	Time	[unit]
Boot TDS platform	×	66.5	s
Start DSAP Outpost service	×	57.25	s
Start DSAP Service	×	16.75	s
Establish SDM session	×	2181	ms
Access SDM resource	✓	9354	ms
TPM_Quote2	✓	2410	ms
Quote verification	✓	694	ms
Copy resource to host	✓	6144	ms
Miscellaneous overhead	✓	106	ms

Service on the TDS platform, minus the time required for directly starting the DSAP Service. Although the integrity protection incurs a significant overhead of 107s, this price is only paid once every boot. The DSAP Outpost should only be booted once per crisis. The majority of the overhead is incurred by the time required to measure the software components of the platform. This time is dominated by the throughput of the platform’s mass storage media. The measurements for the above evaluations were taken on a HP EliteBook 8440p with a 2.67GHz Intel Core i7 CPU, 4 GiB RAM and an Infineon 1.2 Trusted Platform Module (TPM).

Establish SDM session is the time required to establish an encrypted session with the SDM. Ideally, a session is established only once during disaster response. If the user of the device changes, so does the SDM. In this case the session needs to be reestablished. *Access SDM resource* is the time required to retrieve a resource protected by the SDM using an existing SDM session. Of the close to 10s required to retrieve a 4 KiB resource, the bulk of the time is consumed by the TPM_Quote2 operation and the physical copying of the requested resource from the SDM to the TDS⁵. Verification of the TDS platform configuration, and sending back the authorization identifier only requires 694ms.

We have evaluate the performance of the TDS and the SDM in the context of authorizing Mobile Agents. Here the decryption key for the Mobile Agent is protected by the SDM. We consider two options for using the SDM in this application. The first is to retrieve the SDM protected key once after boot and store it in TDS memory while the TDS is turned on. The second option is to retrieve the SDM protected Mobile Agent authorization key every time the user is sent a Mobile Agent. The first option is less obtrusive, only requiring the user to enter her password once. The second option can better mitigate time of check

⁵We expect that alternative buses also supported by the security controller, such as the USB interface, would greatly reduce the copying time.

to time of use problems, as the platform load-time integrity measurements are taken more often.

In both cases, of the total time required to authorize a Mobile Agent with an SDM protected resource, only the first 3[s] are immediately apparent to the user. This time is required to compute the `TPM_Quote2`, for the SDM to verify it, and send back the authorization identifier. At this point the user inputs her shared secret. Actually retrieving the protected resource from the SDM is handled in a background process. The time delay incurred by authorizing the agent in the background is acceptable, because agents operate asynchronously. Mobile agent operations are typically long running, and the user is notified by the agent, when a specific agent finishes its task. The measurements for this SDM evaluation were taken on a HP dc7900 desktop PC with a 3GHz Intel Core 2 Duo CPU, 4GiB RAM and an Infineon 1.2 TPM.

5.9 Security Evaluation

In this section we discuss how our security solution mitigates the threats to the Distributed Secure Agent Platform Outpost (DSAP Outpost) we identified in Chapter 4. To this end we first complement our threat model with an adversarial model. Once we have established the adversarial model we will investigate which threats our security solution mitigates given our adversarial model. We detail the evaluation of how our security solution mitigates the threats identified in Chapter 4 in Appendix B. Here, we summarize these results in Section 5.9.2.

5.9.1 Adversarial Model

Our adversarial model comprises non-targeted remote network and local software attacks. In the following we will describe our definitions of non-targeted attacks, remote and local attacks, network attacks, and local software attacks.

We define a non-targeted attack as an attack where the adversary is not especially singling out a specific attack target, but opportunistically attacks systems that are exposed to his or her attack. For example, we consider the threats of a trojan horse software on a first responder's laptop or a compromised external information system. Equally, we want to protect against attackers that scour the networks for vulnerable systems to turn into nodes in their botnet. We specifically exclude Advanced Persistent Threats. An Advanced Persistent Threats is an entity that brings considerable skill, time, and money to the table to attack a specific target. Our security solution is geared towards wide applicability. It is not suitable for high security applications that have to withstand an Advanced Persistent Threat.

A remote attack is an attack where the attacker is not physically present when attacking. Its counterpart is the local attack. Here the adversary is physically present when performing the attack. A network attack is when the adversary uses a specific network protocol, or set of network protocols, to implement an attack. A example for a remote network attack is a SYN flood. Here the attacker

floods a specific host with TCP SYN packets to create enough half-open connections, such that the attacked host does not accept incoming connections from legitimate users any more. Thus the adversary implements a Denial-of-Service attack.

In a software attack an adversary uses a software program to exploit an implementation fault in the target platform's software to realize a threat. An iconic example here is a buffer overflow in a kernel's system call interface leading to an elevation of privilege. For another example consider the adversary exploiting an implementation fault, such as an integer overflow or a format string attack, to execute code in a program that is executed with root privileges. This attack is again an instance of an elevation of privilege attack. Network attacks can also use implementation faults to achieve similar attack goals. For example a network attack could exploit an implementation fault in a network server to execute code as the targeted network server.

We define a local software attack as a software attack where the attacking software is executed by the local user. This does not necessarily mean that the user wanted to attack the system. For example, consider a user that downloaded a shiny piece of new, free software from the Internet that promised to make money for him. When this user executes the program he realizes that this nice program has encrypted his private files. Although the software makes money, it is not he who benefits, but the blackmailers who will give him the decryption key to his private files. For a small fee of course. One has to live you know . . .

Other reasons for a local attack include the user actually wanting to get root access to his device. Modern phones often disallow root access for security reasons. This is a sensible default for many users, but also restricts functionality. As certain mobile phone manufacturers do not provide a legitimate way to root their devices, users who want to root their mobile phone actually need to use a software attack on the OS to achieve root access.

In our adversarial model we consider the legitimate users trusted, that is, we expect them to follow security policy during disaster response. However, if a user uses her private phone or laptop as a DSAP Outpost, we believe the user should be able to freely use her device as she sees fit, except during disaster response. Furthermore, we exclude social attacks, where an authorized user is coerced into attacking the Secure Agent Infrastructure.

Finally, we also consider the operators of the external information systems that are connected to a DSAP Outpost to follow the Secure Agent Infrastructure security policy.

5.9.2 Threat Model Mitigation

Here we investigate how our security solution consisting of the Trusted Docking Station (TDS) and the Secure Docking Module (SDM) mitigates the threats against an DSAP Outpost we identified in Chapter 4. For this we have considered each individual threat and determined if we consider the threat mitigated and how. We detail our results in Appendix B. Here we present a summary our these results.

Limitations

Load-Time Integrity The acTvSM platform and thus our TDS only provides load-time integrity, not run-time integrity. Software is measured before it is executed, it is not monitored during execution. This is a possible attack vector for a time of check to time of use attack (cf. Section 2.15.3). An adversary might successfully use a vulnerability in the base platform, or the Distributed Secure Agent Platform (DSAP) application Virtual Machine, to take control over the DSAP Outpost. If the platform is already up and running, the attacker might gain access to protected resources, because the attack will not show up in the platform software integrity report. This attack is mitigated by the self-repairing nature of the underlying platform. At the next boot the attack will either be detected, or the platform will boot into an authorized state.

In the following discussion if we claim that a threat is mitigated through load-time integrity, we mean that the threat is alleviated and not eliminated. An adversary might still be able to perform a run-time attack and realize a particular threat.

Mitigation of Compromised Distributed Secure Agent Platform Outposts

All in all we have identified 54 threats to a DSAP Outpost in Chapter 4. Of these 54 threats we mitigate 43 using our security solution that ensures the load-time integrity of the DSAP Outpost. These 43 are all related to a compromised DSAP Outpost. Therefore we want to summarize this mitigation here.

Mobile Agents are executed by the Distributed Secure Agent Platform Services (DSAP Services) in the DSAP Outposts. A compromised DSAP Service has full control over Mobile Agent execution and Mobile Agent data. Such a DSAP Service could therefore masquerade as any Mobile Agent executed by it. Thus, a malicious DSAP Service could leak security sensitive information and tamper with or disrupt disaster response.

To prevent a malicious DSAP Service from masquerading as an agent, the DSAP Service must be denied access to the credentials used by Mobile Agents to perform their disaster response tasks. Credentials are invariably stored in the SDMs or sealed to the TDSs. To access these credentials, the accessing platform must be in an authorized platform configuration, otherwise the TDS cannot unseal the data and the SDM will deny access to the credentials. The assumption is that only a TDS in an unauthorized software configuration would misuse the credentials. While this assumption holds, the credentials are secure.

When an agent is transmitted it can transport additional credentials that it requires to perform its tasks. Therefore, the agent must be protected against receipt by a compromised DSAP Service. Agents in transit are encrypted with the public RSA key of the receiving platform. The corresponding private decryption key is either sealed to the TDS or stored on an SDM. If the receiving DSAP Service is in an authorized configuration, it will be able to unseal the decryption key or retrieve it from the SDM. With this decryption key it can decrypt the Mobile

Agent code and its credentials. Again the assumption is that only a platform in an unauthorized configuration would misuse this agent and its credentials, for example, for obtaining classified information or disrupting disaster response. Again, while this assumption holds, the credentials are secure.

While all of the above assumptions hold true, the credentials are secure within the boundaries of the TDSs. Thus, the integrity of the overall Secure Agent Infrastructure is supported.

Mitigations Based on Secure Communication

Four of the 54 threats pertaining to end point authenticity can be mitigated by using secure communication channels between a DSAP Outpost and the external information systems it is connected to. Here the credentials used to authenticate the communication end points, that is, the DSAP Outpost on the one hand and the external information system on the other, can be protected by our security solution. For this our security solution allows the authentication credentials to be sealed to a security policy conforming authorized software configuration.

Two of the four threats pertaining to the authenticity of the DSAP Outpost or an External Interactor are mitigated specifically by the SDM. Here the SDM fulfills two roles. First it establishes the software configuration of the TDS it is attached to. The authorized software configuration is proven to the user by a shared secret that the SDM only releases to the TDS if it is in an authorized software configuration. Thus the user knows she is dealing with an authentic, policy conforming DSAP Outpost. Second, the SDM establishes the authenticity and presence of a user to the DSAP Outpost. Here the possession of the SDM is a proof of authenticity of the user. Also, once the policy conformance of the software configuration of the TDS has been established, the user has to proof her presence by entering a second shared secret, different to the one used to corroborate the software configuration of the TDS. Thus the DSAP Outpost is given evidence of the authenticity and presence of the user.

Mitigation of Temporary Communication Disruptions

Another four of the 54 threats we have identified are related to an external entity interrupting the communication between the Core Secure Agent Infrastructure and the DSAP Outpost or the DSAP Outpost and an External Interactor. These threats are mitigated by the use of Mobile Agents (see Section 2.6.6).

Out of Scope Threats

Three threats are out of scope of our solution. These threats are Threat 19, where a human External Interactor is eavesdropped upon by another human, Threat 22 where a human is interrupted from using the DSAP Outpost because of an external event, and Threat 29 where an External Interactor repudiates having received information or a command. The final threat can be mitigated

using auditing, and to facilitate this our security solution provides a cryptographically secured storage and a secure communication channel to the Secure Agent Infrastructure.

5.10 Conclusion

We have introduced our security solution for the Distributed Secure Agent Platform Outposts (DSAP Outposts) of the Secure Agent Infrastructure comprising the Trusted Docking Station (TDS) and the Secure Docking Module (SDM). The TDS provides an execution platform that measures the load-time integrity of the software it executes and provides virtualization based security by isolation as a runtime protection mechanism. The SDM is a user authentication and platform verification module that mutually authenticates the DSAP Outpost and its human user to each other, while also establishing that the software configuration of the DSAP Outpost is security policy conforming.

We have designed and implemented both the TDS and the SDM. Furthermore, we have based our prototypical implementation of the TDS on the acTvSM platform [Pir15] and we have implemented the SDM using a Security Controller (SC). We have ported the Distributed Secure Agent Platform Service (DSAP Service) to run on the TDS and to use the TDS's sealing and the SDM's credential protection capabilities to mitigate the threats to the DSAP Outpost.

Finally, we have evaluated our security solution. We have evaluated both the performance of the security solution and its security. The performance evaluation of our additions to the DSAP Outpost shows that the incurred overhead is negligible for disaster response scenarios.

The security evaluation indicates that although our DSAP Outpost is still vulnerable to targeted, online attacks, our platform offers load-time integrity protection and user authentication with commodity-off-the-shelf hardware components. We have evaluated the effectiveness of our approach against the 54 threats to a DSAP Outpost identified in Chapter 4. Our security solution mitigates the 43 related to a compromised DSAP Outpost. Furthermore, it provides the basic mechanisms for counteracting the four threats to authentication, of which two are mitigated by the SDM. Finally, only three threats are not mitigated because they are out of scope of our security solution.

Security and availability are key factors for Mobile Agent System (MAS) to gain acceptance as disaster response support tools. We hope that our work protecting the DSAP Outposts helps to increase the acceptance of Mobile Agent for disaster response so that future disaster response operations can leverage the power of Mobile Agents to prevent further harm to humans beings enduring a disaster situation.

5.11 Future Work

Currently, the Secure Docking Module (SDM) protects the communication with the Trusted Docking Station (TDS) using AES in Cipher Block Chaining (CBC) mode of operation. The CBC mode of operation provides strong confidentiality [Rog11] when operated with a random IV, as is the case for the SDM and TDS communication. However, it provides no protection against malleability attacks [Rog11]. Consequently, the confidentiality of the communication between the SDM and the TDS is protected, but the integrity is not. Although we have no concrete attack in mind, the potential for an adversary to modify the plaintext of the communication in any meaningful way by modifying the ciphertext worries us. Therefore, we propose to upgrade the communication protection to use Authenticated Encryption, as introduced in Section 2.9.3.

Although the SDM supports user authentication through possession of the security module, its security is still limited by the fact that the protected resources, serving for authentication and authorization, are released to the host platform. One solution to alleviate this problem in the future is to use the host platform verification to actually protect cryptographic operations implemented on the SDM, such as challenge response authentication protocols, or on chip signature creation and verification.

For our prototypical adaptation of the Distributed Secure Agent Platform Service (DSAP Service) we have only used the TDS and the SDM to protect the Distributed Secure Agent Platform Outpost (DSAP Outpost) specific agent authorization key (see Section 5.4). However, as our security evaluation has indicated it would be beneficial to also protect the authentication credentials for secure communication with External Interactors. Also, although we have presupposed a secure communication link between the Core Secure Agent Infrastructure and the DSAP Outposts, it has not yet been implemented. As described in Section 5.4 we believe this can be implemented by choosing a secure communication protocol for Apache River (formerly Jini), which is underlying the DSAP Service.

Finally, as our performance evaluation showed, the bulk of the time required to use the SDM's resource access protocol is allocated by transferring the resource from the SDM to the TDS. The main reason for this is our choice of the ISO7816 interface for communication with the SDM. The Security Controller we used to implement the SDM also provides a Serial Peripheral Interface and a USB interface. Both of these interfaces can be significantly faster than the ISO7816 interface. Therefore, we propose to switch to one of these two interfaces as future work.

6

Secure Block Device

6.1 Introduction

Declaration of Sources

This section is based on and reuses material from the following sources, previously published by the author:

- [HWF15] Daniel M. Hein, Johannes Winter, and Andreas Fitzek. Secure block device – secure, flexible, and efficient data storage for ARM TrustZone systems. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20–22, 2015, Volume 1*, pages 222–229, 2015.
- [BHRS15] Roderick Bloem, Daniel M. Hein, Franz Röck, and Richard Schumi. Case study: Automatic test case generation for a secure cache implementation. In *Tests and Proofs – 9th International Conference, TAP 2015, Held as Part of STAF 2015, L’Aquila, Italy, July 22–24, 2015. Proceedings*, pages 58–75, 2015.

References to these sources are not always made explicit. In particular the description of the block cache is adapted from [BHRS15] and the Secure Block Device details are based on [HWF15].

6.1.1 Motivation

We believe that disaster response can greatly benefit from using smartphones and tablets. However disaster response requires strong security for data protection (see Chapter 4 and O’Neill [OSZW12]). Therefore, we have developed the security solution comprising the Trusted Docking Station (TDS) and the Secure Docking Module (SDM) we introduce in Chapter 5. Our security solution is based on the acTvSM platform. The acTvSM platform in turn is based on Intel Trusted eXecution Technology (TXT) (see Section 2.15). However, the predominant platforms in the smartphone market are Android and Apple’s iPhone using ARM System-On-Chips. Neither of these platforms supports TXT. Therefore, we started to investigate how we can use the security mechanisms available on ARM based systems to build a platform with similar security properties to the acTvSM platform. Our first effort in this direction is a storage mechanism that uses cryptography to protect the confidentiality and integrity of data, the Secure Block Device (SBD).

There are a number of examples, where researchers propose to use smartphones to facilitate disaster response. Mitra and Poellabauer [MP12] show how smartphones using their acceleration sensors and their ability to form mobile ad-hoc networks can be used to help paramedics to monitor the heart rate of multiple-patients. Also Ishigaki et al. [IMIT13] developed a mobile radiation monitoring system and field tested it in Fukushima following the 2011 nuclear power plant incident. The mobile radiation monitoring system used cheap radiation sensors attached to smartphones as measurement equipment. As a final example, Thompson et al. [TWD⁺10] proposed the use of smartphones to detect car accidents and to provide situational awareness for emergency responders.

In addition to a smartphone’s multitude of sensors that allow for a number of interesting use cases in disaster response, and its ability to form mobile ad-hoc networks that can mitigate communication outages [DHK10], smartphones and tablets have significant computational power, memory and storage. Smartphones have the technical capability to participate in a Multi-Agent System for years now (see for example Chan et al. [CRP08]). In fact, in 2008 Ughetti et al. [UTG08] demonstrated the use of the Java Agent Development Framework (JADE) Mobile Agent System in the Android smartphone OS.

Having established the utility and capability of using smartphones and tablets for disaster response we have to tackle the question of security. As the disaster response scenario for Mobile Agent Systems mandates strong security requirements for data protection (see Chapter 4 and O’Neill [OSZW12]) we are interested in security mechanisms that go beyond standard operating system (OS) capabilities. We estimate that with the advent of the ARM TrustZone security extensions (see Section 2.17) and the de-facto dominance of ARM based platforms for smartphones a strong security by isolation mechanism is now widely deployed in smartphones. Recently, a number of OSs supporting the ARM TrustZone security extensions and the GlobalPlatform standards for Trusted Execution Environments, such as ANDIX OS (see Section 2.18), have arisen. This enables us to look into building a Mobile Agent System that uses Trusted

Applications in conjunction with the SDM (see Section 5) for protection of sensitive data. One of the first requirements we established for such a Mobile Agent System was the need for authentic and confidential data storage.

ANDIX OS takes minimizing its Trusted Computing Base to the limit by delegating the onus of secure storage to its Trusted Applications. ANDIX OS choose this design on the assumption that not every Trusted Application might need fully blown, fast random block access for its confidential and authentic data. Many Trusted Applications can fulfill useful roles without needing a secure storage with fast random access. A Trusted Application might not have any storage requirements at all. One such example is VeriUI [LC14], a component that protects a user's credentials while they are input. Another example is DroidVault [LHB⁺14] that requires secure storage, but not data freshness. Finally, there are applications that need secure storage, but not fast random block access [KEAR09]. Thus by not enforcing secure storage with data freshness and fast random access inside the ANDIX OS kernel, ANDIX OS can still be the foundation of many Trusted Applications, while minimizing its Trusted Computing Base.

6.1.2 Contribution

We have developed the Secure Block Device (SBD) to provide data confidentiality and authenticity for Trusted Applications running on ANDIX OS. We have created the SBD as part of an effort to establish basic security building blocks for implementing a Mobile Agent System for smartphones. The SBD is an *easy to use, efficient, flexible, and widely applicable C* software library for adding data confidentiality and authenticity, including data freshness, to a variety of storage systems. We want to emphasize that the SBD is not a file system. The SBD wraps storage systems with a block device like interface. However, a file system can use the SBD as a storage back end wrapper and thus achieve data confidentiality and authenticity.

6.1.3 Properties of the Secure Block Device

Security

The SBD provides data confidentiality and data authenticity, including data freshness, for data at rest. The SBD uses cryptographic mechanisms to achieve these data security goals. The security goals hold against an adversary who has access to the cryptographically protected data, but not the cryptographic access credentials. In conjunction with a Trusted Execution Environment, the security goals also hold against an adversary that can read, modify and drop any SBD read and write messages between the Trusted Execution Environment and the Normal World, but who is unable to directly access the Secure World memory or Secure World devices. We do however explicitly deny our adversary the ability to perform local implementation attacks (see Section 2.13) and local and remote side-channel attacks such as cache-timing at-

tacks [AES15, LYG⁺15, SP13, GSM15]. Resilience against side-channel attacks depends on implementation details, such as the quality of the implementation of the cryptographic mechanisms and ANDIX OS's cache management. Therefore, we consider it out of scope. See Section 6.4.1 for details on our adversarial model.

The SBD uses Authenticated Encryption (AE) combined with a Merkle Tree to achieve its security objectives. The AE scheme is selectable. Currently the SBD supports Offset Codebook Mode (OCB) and Synthetic Initialization Vector (SIV). Additional schemes can be easily added, and can even take advantage of platform specific cryptographic hardware. The SBD's only security requirement is that the user of the library is able to provide confidential and authentic storage for a symmetric cryptographic key and a single integrity value. As dedicated secure storage in hardware is usually at a premium, the SBD reduces the problem of confidential, and authentic data storage of arbitrary amounts of data to storing a single integrity value and cryptographic key. We discuss the security properties of the SBD in Section 6.4.

Ease of use

The SBD has a simple to use application programming interface (API) (see Section 6.3.5) and the use of cryptography is as transparent as possible. The SBD's API is modelled after the well-known Portable Operating System Interface (POSIX) file abstraction as specified for the C standard library; a user can use the SBD as he or she would use a file. Furthermore, the SBD's API only requires cryptographic credentials when a new SBD is created, or an existing SBD is opened. For all other operations, such as reading or writing data, the cryptography itself is completely transparent.

Efficiency

The SBD has three efficiency mechanisms. The first is fast random block access. The second is a block cache (cf. Section 6.3.3) and the third a cryptography abstraction layer (cf. Section 6.3.2). Fast random block access is complicated when using cryptography to achieve data security goals. Therefore, we need to combine two mechanisms, an AE scheme and a Merkle Tree, to achieve it. See Section 6.2.3 for details. The last two efficiency mechanisms aim at mitigating the impact of cryptography on performance. The block cache caches decrypted data, thus on a cache hit, no cryptographic operations are performed. The cryptographic abstraction layer allows the user of the SBD to decide which AE scheme to use at SBD instantiation time.

Flexibility

The SBD works with a variety of storage back ends, such as block devices or ANDIX OS's file abstraction. To this end the SBD only requires its back end to provide address based block read and block write functions, such as POSIX compatible implementations of the `pread` and `pwrite` functions.

Applicability

The SBD is an add-on module for Trusted Applications executing in the Secure World user space. To achieve this goal the SBD is implemented as a software library. If a Trusted Application requires a confidential and authentic storage, it can use the SBD library. Additionally, the SBD can be linked to any application with similar properties. To allow widespread use, the SBD is published under an open source license¹.

6.1.4 Evaluation

Security

In Section 6.4, we evaluate the security of the SBD by threat modeling the use case where the SBD is used to add a confidential and authentic data storage to a Trusted Application using the Microsoft Threat Modeling Tool 2016. We then argue how the SBD mitigates all threats to data confidentiality, authenticity, and freshness. Finally, we fully disclose and discuss how our SBD uses the Merkle Tree, AE, and Cipher-based Message Authentication Code (CMAC) primitives to achieve the data confidentiality, authenticity, and freshness goals even in the face of an adversary that can perform **spoofing**, **splicing**, and **replay** attacks (see Section 2.11).

Performance

The SBD was designed to provide a secure persistent storage to Trusted Applications (TAs) as part of an effort to build a Mobile Agent System for smartphones. In lieu of said Mobile Agent System and to thoroughly put both ANDIX OS and the SBD through its paces, we extended the idea of providing secure storage not only to a Trusted Application, but also as a service for the Normal World. We have implemented a Trusted Application providing a simple block storage interface to the Normal World. The Trusted Application stores the block using the SBD mechanism. In the Normal World we used the Linux Network Block Device kernel infrastructure to integrate our secure storage solution. Thus, we have developed a TrustZone protected block device. We evaluate the performance of the SBD in this context.

6.2 The Secure Block Device Operation Principle

The Secure Block Device (SBD) is a software library that adds confidentiality and authenticity, including data freshness, to arbitrary block storage devices. The application programming interface (API) allows reading, and writing arbitrary sized byte buffers, at arbitrary offsets, to and from the SBD. The SBD applies a protection mechanisms based on a selectable Authenticated Encryption

¹https://www.iaik.tugraz.at/content/research/opensource/secure_block_device/

(AE) scheme (see Section 2.9.3), the Cipher-based Message Authentication Code (CMAC) MAC (see Section 2.9.2), and a Merkle Tree (SHA-256) to achieve data confidentiality, authenticity, and freshness.

6.2.1 Creating a Secure Block Device

When a client, such as a Trusted Application, first initializes a new SBD it needs to provide a back end Datastore and a symmetric master key MK . This symmetric master key authorizes later use of the SBD and its confidentiality and authenticity must be protected by the client. The SBD uses MK with the Synthetic Initialization Vector (SIV) S2V pseudo random function construction to derive two internal symmetric keys. These keys are the *AE key* AE_k and the *CMAC key* $CMAC_k$. They are stored in the SBD header encrypted under the master key, when the SBD is closed.

6.2.2 Opening a Secure Block Device

A client can open a previously created SBD by providing the master key MK and the master authenticity tag M_t representing the authenticity of the overall SBD. We detail the computation of M_t in Section 6.2.3. It is returned to the SBD's client upon closing the SBD and the client has to ensure the authenticity of M_t when supplying it. Upon opening an SBD MK is used to decrypt the two internal symmetric keys AE_k and $CMAC_k$. $CMAC_k$ and M_t are used to verify the overall authenticity of the SBD. The root hash of the Merkle Tree is the representative for the data authenticity and freshness of the complete SBD. When opening an existing SBD all management blocks (see Section 6.2.3) are read, their CMACs computed, and the Merkle Tree rebuilt. The Merkle Tree root hash is then compared with the client specified reference value M_t . If the root hash of the rebuilt Merkle Tree matches the specified reference value, the SBD open operation succeeds; otherwise it fails, because the integrity of the SBD has been compromised. If rebuilding the Merkle Tree and its verification succeeds the SBD can ensure that all unauthorized changes to the data stored in the SBD become evident on reading the data from the untrusted back end store.

6.2.3 Using a Secure Block Device

The SBD splits incoming read and write requests from the client into fixed size² Data Blocks. This is done to allow scalable random access. Random access in combination with cryptographic protection is difficult, because the naive solution of encrypting and authenticity protecting the full SBD is prohibitively slow. If the SBD were to use the naive solution, it would have to decrypt and verify the authenticity of its whole data just to read a single bit. Therefore, to allow for efficient random access, the SBD splits the data into blocks of fixed size.

²block size is a compile time parameter

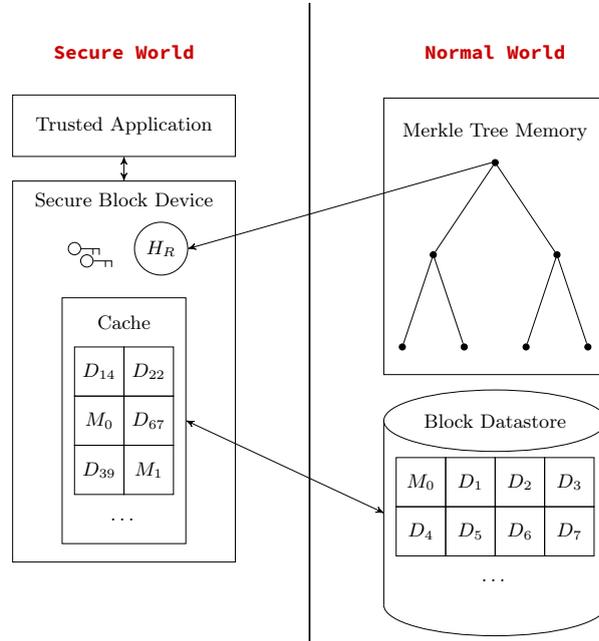


Figure 6.1: The component partitioning diagram for a Trusted Application that uses the Secure Block Device to store data. The diagram details which components reside in the Secure World and which in the Normal World. It also specifies which data is kept in RAM and which is persisted in a Block Datastore.

Block processing complicates achieving our overall confidentiality and authenticity, including data freshness, objectives for the SBD. We need to ensure data freshness for all blocks, the confidentiality and authenticity of Data Blocks, and the authenticity of management blocks (see below) and the header block. We use a Merkle Tree for the data freshness of all blocks and the AE to protect the Data Blocks and the CMAC for the management and header blocks. We will now explain how these three mechanisms, the Merkle Tree, the AE and the CMAC, mesh together to achieve the overall security objectives by explaining how Data Blocks are processed when written or read.

However, before we detail how Data Blocks are read and written we first outline how the SBD components we need for our discussion are partitioned with respect to the Secure World and the Normal World (see Section 2.17). Furthermore, we detail where these components store their data. Specifically, we discuss which data is stored in RAM and which in a persistent Block Datastore.

Figure 6.1 depicts how we partitioned our components and where they store their data. Here, a Trusted Application uses the SBD to store its data. The SBD maintains a pair of cryptographic keys and the root hash H_R in Secure World RAM. The root hash is the root of the Merkle Tree. The other Merkle

Tree nodes and its leaves are conceptually stored in Normal World RAM. Note that the SBD performs all Merkle Tree computations in the Secure World, it just stores the Merkle Tree in the Normal World. The SBD uses a cache component (see Section 6.3.3) to speed up access to blocks of data D_x , where x is the Data Block index. Next to Data Blocks the cache also stores management blocks (see below) M_y , where y is the index of the management block. All these blocks are cached in Secure World RAM. When the cache evicts a block it writes it to a persistent Block Datastore in the Normal World. Conversely, when the cache loads a Data Block it loads it from the persistent Block Datastore in the Normal World.

Writing a Data Block

The SBD block processing works as follows. A Data Block (D_x , where x is the block index) is written to the back end store when it is evicted from the SBD block cache. First, D_x is encrypted and an authentication tag is computed using an AE scheme (enc_{AE})

$$(E_x, T_x) = enc_{AE}(AE_k, D_x, \{x, IV\}), \quad (6.1)$$

where E_x is the encrypted and authenticity protected Data Block with index x , T_x is the corresponding authentication tag, AE_k is the AE key, D_x is the Data Block to encrypt, x is the block index and IV is the initialization vector (IV). We use a global counter as IV, when the AE scheme supports an IV; otherwise we input the global counter and the block index x as associated data (see Section 2.9.3).

Why add a global counter? When applying an AE scheme to individual blocks, identical plaintext blocks will encrypt to identical ciphertext blocks. We use an SBD global counter to prevent this. Every time a Data Block is encrypted using an AE scheme we supply the current value of the global counter as IV to the AE operation, and then we increment the global counter. This ensures that the IV is unique for every encryption and therefore each ciphertext is different. We use a 96-bit (12 bytes) global counter for the SBD. This allows for 2^{96} write operations, before the SBD's AE key must be changed and the whole SBD reencrypted (see Section 6.6).

For every Data Block D_x we need to store the IV and its T_x in addition to the protected data E_x . We have decided to group this information in a special block type, the management block. Management blocks are stored interleaved with Data Blocks in the back end store. The lower part of Figure 6.2 illustrates the block structure of an SBD with a block size of 2 KiB. The first block, with block index 0 is the header block (HDR). The header block stores, amongst other data, the SBD version, and the encrypted AE_k and $CMAC_k$. The second block with the block index 1 is a management block (M_0). Management blocks store the IVs (IV) and authenticity tags (T_x) for a specific number of Data Blocks. We have assigned a fixed size of 16 bytes to the IVs and authenticity tags, although we currently only use 12 bytes for the IV. Consequently, the number of Data

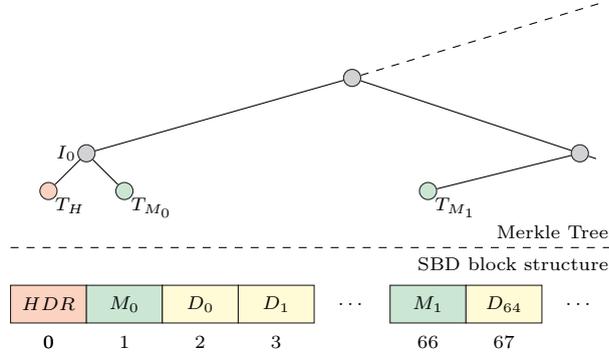


Figure 6.2: Block structure for a Secure Block Device with a block size of 2 KiB

Blocks covered per management block solely depends on the SBD block size. In our example with a block size of 2 KiB, each management block can store 64 (IV, T_x) tuples. Hence the next 64 blocks are Data Blocks, being followed by the management block for the next 64 Data Blocks etc.

The management block M_y corresponding to our DB_x is authenticated using CMAC

$$T_{M_y} = \text{CMAC}(\text{CMAC}_k, M_y) \quad (6.2)$$

where T_{M_y} is the Message Authentication Code (MAC), CMAC is the CMAC function, CMAC_k is the CMAC key and M_y is the management block to protect. T_{M_y} is then inserted in the Merkle Tree, the tree updated and the root hash recomputed. Finally, both the management block and the Data Block get written to the persistent storage.

We use a binary Merkle Tree to protect against replay attacks. The upper part of Figure 6.2 depicts the subtree of the Merkle Tree pertaining to the visible management blocks. The leftmost leaf in the subtree T_h is the CMAC of the header block HDR , whereas T_{M_0} and T_{M_1} are the CMACs of the first and the second management block respectively. As detailed in Section 2.12, the inner nodes are the hashes of the concatenated children. For example,

$$I_0 = \text{SHA-256}(T_h || T_{M_0}), \quad (6.3)$$

where $\text{SHA} - 256$ is the SHA-256 hash function, T_h is the left child (the CMAC of the header), T_{M_0} is the right child (the CMAC of the first management block), and $||$ is the concatenation operation. The advantage of using a Merkle Tree in our scenario is that, with the marked exception of the root hash, the Merkle Tree can be maintained in the Normal World, that is, in untrusted memory. This is advantageous, because the Merkle Tree can grow to significant size.

We could have directly stored the Data Block authentication tags (T_x) in the leaves of the Merkle Tree to protect their freshness. However, this strategy creates a Merkle Tree of significant size. For example, the Merkle Tree for a 2 GiB

SBD with a block size of 2 KiB would require a minimum of 2^{21} hashes. Given that the output size of the SHA-256 function is 32 bytes (2^5), the memory size of the Merkle Tree would amount to a minimum 64 MiB. To reduce the Merkle Tree memory requirements, instead of directly storing the block authentication tags in the Merkle Tree, they are stored in management blocks, which in turn are stored in the persistent storage, when not in cache. Given the above example, where each management block stores 64 IVs and authentication tags, the size of the Merkle Tree is reduced to 2^{15} hashes, or a minimum of 1 MiB of memory. Although most of the Merkle Tree can be maintained in the Normal World, we are operating on mobile device, where memory still is a precious commodity. Thus, we believe this design decision justified.

Reading a Data Block

When the client application requests data from the SBD, and the corresponding Data Blocks are not yet in cache, the SBD will request the Data Blocks from the untrusted back end Datastore. For every encrypted and authenticity protected block E_x read from the back end Datastore the SBD ensures it has the corresponding management block M_y loaded into the block cache. If M_y is not yet in the cache, the SBD reads it from the back end Datastore and computes T_{M_y} , the CMAC of M_y . The SBD then recomputes the root hash of the Merkle Tree with the leaf value supplied by T_{M_y} . If the newly computed root hash matches the root hash maintained by the SBD in the Secure World, M_y is accepted; otherwise it is rejected and the SBD signals an integrity error to the client.

If the SBD accepts M_y it starts processing E_x . The SBD decrypts and checks the authenticity of E_x by applying

$$D_x = dec_{AE}(AE_k, E_x, x, IV, T_x), \quad (6.4)$$

where D_x is the decrypted and authentic Data Block with the block index x , dec_{AE} is the authenticated decryption function, AE_k is the key, E_x the protected block, x the block index, IV the IV and T_x the authenticity tag. Both, IV and T_x are read from M_y . The decryption function dec_{AE} will only succeed if the correct key is used and x , IV , and T_x match the encrypted block E_x . Thus if E_x has been modified by an adversary, dec_{AE} will fail. The SBD returns the requested data to the client if all required management block verification and decryption operations succeed.

6.2.4 Closing a Secure Block Device

When an SBD is closed, the current Merkle Tree root hash maintained in the Secure World is returned to the client as new M_t . The client must protect M_t against modification, as it is the representative for the data authenticity and freshness of the complete SBD. If the SBD is not closed correctly, for example, because ANDIX OS crashes, the M_t of the client and the one recomputed by the SBD from its data upon opening might not match. In this case the client has to manually check the consistency of the data.

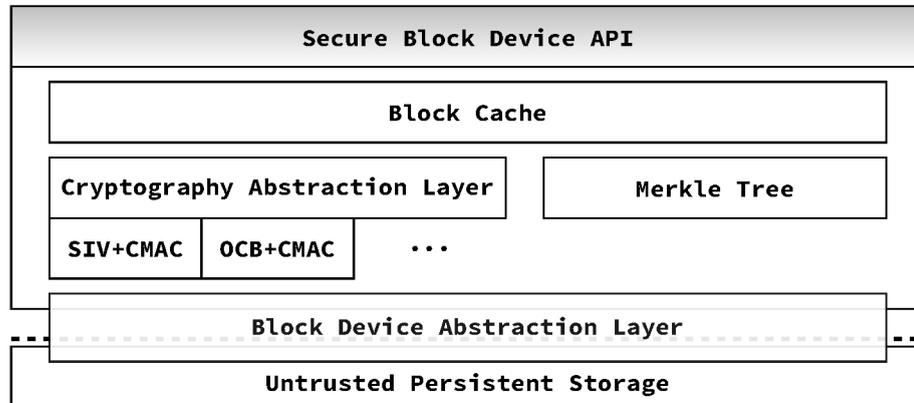


Figure 6.3: Secure Block Device architecture

6.3 The Secure Block Device Architecture

Our architecture for the Secure Block Device (SBD) is depicted in Figure 6.3. The block device abstraction layer abstracts the implementation details of the underlying block device. The actual SBD functionality requires the block cache, the cryptography abstraction layer, including the corresponding Authenticated Encryption (AE) scheme and cryptographic functions implementations, and the Merkle Tree. Finally, the SBD interface provides an easy to use C POSIX file like interface to its users.

6.3.1 The Block Device Abstraction Layer

The SBD requires only limited support from the underlying system. The SBD requires functions to read and write index addressable Data Blocks of a fixed, compile-time configurable size. The block device abstraction layer hides the details of the underlying implementation by providing a unified interface to the SBD. This abstraction enables using both the Portable Operating System Interface (POSIX) `pread` and `pwrite` C standard functions, as well as the ANDIX OS kernel block device abstraction as back ends for the SBD.

6.3.2 Cryptography Abstraction Layer

The SBD is designed to support different implementations for the required cryptographic functions. The cryptography abstraction layer facilitates integration of new AE schemes and Message Authentication Code (MAC) functions into the SBD. This layer hides the details of the underlying AE scheme, and MAC function implementation behind a unified interface. The interface allows for using different keys for the AE scheme and the MAC computation, if necessary.

Currently, the SBD supports two different AE schemes. These schemes are Offset Codebook Mode (OCB) [RBB03] and Synthetic Initialization Vec-

tor (SIV) [RS06]. Notably, the two schemes treat the 96-bit global counter differently. The OCB scheme uses it as initialization vector (IV), and the SIV scheme treats it as associated data. This is secure, because in SIV a change in the associated data also changes the cipher text. As is to be expected the OCB schemes performs significantly better than the SIV scheme, as OCB is a single-pass scheme, whereas SIV needs to process the data twice. Thus, selection of the appropriate AE scheme allows a trade-off between efficiency and a more liberal software license, as OCB is patented³.

6.3.3 The Block Cache

The block cache is an efficiency mechanism. When the SBD writes data to the back end Datastore, this data is encrypted and its authenticity protected. Conversely, when the SBD reads data it has to reverse the encryption and verify the data authenticity. Both ways, this is computationally significantly more costly than simply reading and writing unprocessed data. In addition, ANDIX OS uses the Normal World for data storage, which is expensive, due to the inter-world communication overhead. The performance is further degraded by using management blocks to reduce the size of the Merkle Tree. Every time a Data Block is updated, its authentication tag is updated in the corresponding management block, which is then written. To minimize these costs the SBD uses a block cache. The block cache retains a configurable number of Data Blocks⁴ in unencrypted and unprotected form for quick access in RAM.

The block cache is a write-back, write-allocate cache. The block cache lives in the Secure World Trusted Application. Thus, depending on the workload, it can significantly reduce the number of cryptographic operations, world, and context switches. The Secure World block cache stores the plaintext of the actual data and management blocks. As management blocks contain data pertinent to a range of Data Blocks, the cache component gives them preferential treatment over pure Data Blocks. Also, to read and write a specific Data Block, the corresponding management block needs to be in the cache. This is an invariant that must hold for the SBD to work, and the cache has partial responsibility for ensuring it. A management block is evicted from the cache, iff no corresponding Data Blocks are left in the cache, and it is the least recently used cache element. On the other hand, if a Data Block is evicted, the Data Block itself, and the corresponding management block are stored. This strict policy minimizes data integrity loss in case of a system crash. We have systematically tested the cache component using test cases automatically generated from a formal cache model [BHRS15].

The block cache component provides a very simple interface towards the SBD. The interface comprises a call back to the SBD for writing a dirty block before evicting it from the cache, a call back predicate function to test if given management block is responsible for a given Data Block, and a request Data

³<http://www.cs.ucdavis.edu/~rogaway/ocb/license.htm>

⁴SBD cache size is a compile time parameter

Block function. The request Data Block function either provides access to the requested block data, if it is cached, or a free cache slot. The SBD can use the free cache slot for storing the block data it now reads from the persistent data storage back end. Before the data is stored in the free cache slot, the SBD also decrypts it and verifies its integrity.

6.3.4 The Merkle Tree

The Merkle Tree component is a dynamic implementation of a binary Merkle Tree data structure optimized for use with the SBD. One of the objectives for the SBD is to keep its memory footprint small, as it is targeted for embedded devices running ANDIX OS. To this end we have implemented our Merkle Tree to grow and shrink in size depending on the size of the SBD. This capability limits memory usage, while still allowing the SBD to grow in size. If the SBD consists only of a few blocks, the memory footprint of the Merkle Tree is also small. However, if the SBD grows, the Merkle Tree grows with it as needed.

How Does Your Garden Grow?

To detail how the Merkle Tree grows dynamically, we first explain how the SBD grows. An SBD instance starts with a size of zero and grows as needed. In other words, the current size s of the SBD and the current number of Data Blocks n in the SBD are determined by the highest index written to so far. For example, if the highest index written to so far is 133121 and the block size $s_B = 2048$ (2 KiB), then $s = 131121$ and $n = 65$. When the client application writes beyond the last allocated Data Block, the SBD automatically grows the back end Datastore to accommodate the new data. For the Merkle Tree the interesting case is when the SBD has to create at least one new management block in the process.

As explained in Section 6.2, management blocks store (IV, CMAC) tuples for Data Blocks. Each management block is responsible for a number of Data Blocks, depending on the chosen block size of the SBD. As an invariant, the SBD always creates all necessary management blocks to store the (IV, CMAC) tuples for all n Data Blocks. For example, if the block size is $s_B = 2048$, then each management block holds 64 (IV, CMAC) tuples. Put differently, in our example the SBD needs one management block per 64 Data Blocks. Let us further assume that the SBD has not yet been written to and is empty. Now the client application writes a byte to index 133121. By dividing through the block size the SBD determines it needs to write the byte to 65th Data Block. Here the SBD creates two new management blocks, M_0 and M_1 . M_0 stores (IV, CMAC) tuples for the Data Blocks $DB_x, 0 \leq x < 63$ and M_1 covers $DB_x, 64 \leq x < 127$. These two new management blocks now need to be added to the Merkle Tree.

Figure 6.4 illustrates how our binary Merkle Tree implementation grows as the SBD grows. Our Merkle Tree implementation manages the tree in levels. Memory is allocated per level and not for the tree as a whole. Each level is

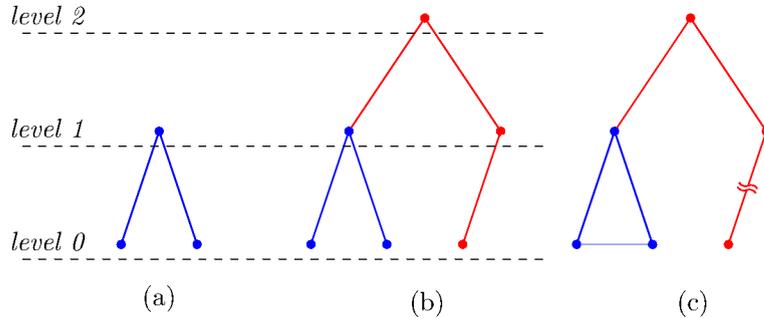


Figure 6.4: Dynamically growing a Merkle Tree

managed as a dynamically allocated, exponentially growing array of hash values. So for Figure 6.4 (a), our Merkle Tree maintains two arrays, L_0 for level 0 and L_1 for level 1. When a single new leaf is added (see Figure 6.4 (b)), our implementation doubles the size of the L_0 array to four elements and creates a new L_2 array to hold the new root hash. The L_1 array is not grown yet. Why can we get away without growing L_1 ? Because without a fourth node on level 0, we keep the value of the new inner node on level 1 equal to the new leaf value. Figure 6.4 (c) shows how the Merkle Tree generally grows from a perfect tree to a tree with a new level, where the old tree is now the right subtree.

Our level based exponential growth has slightly better memory usage characteristics than exponentially growing the memory for the whole tree. Basically, when the tree grows over an exponential boundary, we get one node comparatively cheaply. For example, consider a tree where $n_M = 1024$. Here n_M is the number of management blocks, which is identical to the number of leaves of the tree. One additional node ($n_M = 1025$) will cause L_0 to grow to hold an additional 1024 hash values. If we were to grow the memory for the whole tree exponentially, we would have had to allocate space for 2048 hash values immediately. The advantage of the level based tree growth strategy rapidly diminishes. To continue our example, for $n_M = 1040$ our strategy already allocates memory for 1985 hash values, which is already close to the maximum of 2048.

The Merkle Tree in Untrusted Memory

One of the advantages of using a Merkle Tree for memory authentication (see Section 2.10) is that with the exception of the root node the whole tree can be maintained in untrusted memory. Even if an adversary modifies a tree node, the modification of the tree node will be detected upon reading a management block that is a descendant of that tree node, unless the adversary is able to find a collision in the underlying hash function. Thus Merkle Trees make tampering with their structure evident.

We also designed the Merkle Tree used by the SBD to be maintained in untrusted memory. However, currently ANDIX OS lacks the ability to efficiently

use untrusted memory. Therefore, our Merkle Tree is currently maintained in Secure World memory.

6.3.5 Secure Block Device API

Our design goals for the SBD application programming interface (API) were twofold. First, the SBD should be easy to use. Second the underlying, cryptography based security mechanism should be as transparent as possible. We have modelled the SBD API after the POSIX file API for the C standard library. The SBD provides the following commands `sbd_open`, `sbd_lseek`, `sbd_read`, `sbd_write`, `sbd_pread`, `sbd_pwrite`, `sbd_fsync`, and `sbd_close`. With the exception of the `sbd_open` and `sbd_close` the function signatures mirror their POSIX counterparts. The behavior of the SBD functions is also modelled after their POSIX counterparts based on the description POSIX Programmer's Manual.

The SBD security mechanisms are completely transparent during SBD operation. Only the SBD open and SBD close operations require special attention. When opening an SBD, the caller must specify the master key MK (cf. Section 6.2) and a root hash reference value M_t . The master key is used for key derivation and key storage. The root hash reference value is used to ensure the integrity of the SBD. Conversely, if the SBD is closed this value is returned for safe-keeping by the caller. Therefore, we claim that the SBD API is easy to use, as it is based on a very well known and widely adopted file abstraction. Furthermore, the security mechanism is completely transparent, with the sole exception of opening and closing an SBD, and here, the additional interaction is kept at the minimum.

6.4 Security Evaluation

The security evaluation we present herein is based on Appendix C. Specifically, the threat model we describe in Section 6.4.2 is a summary of Section C.2 and Section C.4.

6.4.1 Adversarial Model

We analyse the security properties of the Secure Block Device (SBD) against an adversary that has full access to the cryptographically protected data at rest, but not to the Authenticated Encryption (AE) key and the Merkle Tree root hash. This adversary can read and modify all data while it is at rest and record operations on the data over the full lifetime of the SBD store, but cannot break the cryptography.

Specifically, when the SBD runs in a Trusted Application, the following assumptions about our adversary hold. Our adversary is restricted to software attacks. Our adversary has full privileges on the Normal World side, but can only access the Secure World and the Trusted Applications therein using the

Table 6.1: Secure Block Device API

Function	Description
<code>sbd_open</code>	Creates a new, or opens an existing SBD and associates it with an SBD handle. It requires specification of a master encryption key, and, when opening an existing device, a root hash reference value.
<code>sbd_lseek</code>	Moves the read/write offset of the SBD to the specified position
<code>sbd_read</code>	Tries to read the requested number of bytes at the current SBD read/write offset into the given buffer.
<code>sbd_write</code>	Tries to write the specified number of bytes from the given buffer to the SBD at the current read/write offset.
<code>sbd_pread</code>	Tries to read the requested number of bytes at the specified SBD offset into the given buffer.
<code>sbd_pwrite</code>	Tries to write the specified number of bytes from the given buffer to the SBD at the specified offset.
<code>sbd_fsync</code>	Tries to flush the internal SBD block cache and signals ANDIX OS to flush its virtual block device interface.
<code>sbd_close</code>	Tries to re-encrypt the SBD symmetric keys. It also returns the root hash reference value.

Remote Procedure Call interface provided by ANDIX OS. In the Normal World the adversary has full control over all processes, can access all memory, and can fully read and write all files, including the Datastore used by our Trusted Application. We do however deny our adversary the ability to perform software side-channel attacks to obtain the cryptographic keys used by the SBD, as we consider this to be out of scope for this work.

Thus our adversarial model encompasses an adversary that can perform **spoofing**, **splicing**, and **replay** attacks, as discussed in Section 2.11. Our adversary cannot perform a **forking** attack, because there is only a single user for the SBD, the Trusted Application that is using it.

6.4.2 Threat Model

We have created a threat model for the use case where a Trusted Application that runs in the ANDIX OS Trusted Execution Environment stores sensitive data using ANDIX OS' storage mechanism. The storage mechanism of ANDIX OS relies on the Normal World to store data (cf. Section 2.18.3), which we consider insecure. We detail our threat modeling efforts for this use case in Appendix C.

In Table 6.2 we enumerate the threats from our threat model we consider pertinent. The ID column shows the unique number of the threat. The T column contains the type of the threat according to Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege. The title column displays the name of the threat and the AID is the identifier of the threat used in Appendix C. The final column M contains a check mark if we consider the threat to be mitigated by the SBD.

In comparison with the list of threats in Appendix C we have removed three threats that do not apply, because we consider the Trusted Application trusted and secure by definition. These three threats are

- 4 Potential data repudiation by the Trusted Application,
- 5 Potential process crash or stop for the Trusted Application,
- 8 The Trusted Application may be subject to elevation of privilege using remote code execution.

6.4.3 Discussion

The SBD is intended to be a Single-User Authentic Block Datastore and uses the well known combination of a Merkle Tree in conjunction with an AE scheme to achieve Single-User Block Datastore Authentication, while retaining fast random access (cf., for example, [YSK09]). We argue that using the SBD with its data confidentiality, authenticity and freshness guarantees completely mitigates all spoofing, tampering, information disclosure and elevation of privilege based on data corruption threats, that is, Threats 01, 02, 03, 04, 07, 08, 10, 11, and 15.

Table 6.2: Threats to a Trusted Application when reading and writing data to and from an untrusted Datastore

ID	T	Title	AID	M
<i>Reading data</i>				
01	S	Spoofing of Datastore	1	✓
02	I	Weak access control for the Datastore	2	✓
03	I	Data Flow Sniffing	2a	✓
04	S	Spoofing the Trusted Application	3	✓
05	D	Data flow <i>Read</i> from the Datastore to the Trusted Application is potentially interrupted	6	
06	D	Datastore inaccessible	7	
07	E	Elevation by changing the execution flow in the Trusted Application	9	✓
<i>Writing Data</i>				
08	S	Spoofing of the Datastore	10	✓
09	D	Potential excessive resource consumption for Trusted Application or Datastore	11	
10	S	Spoofing the Trusted Application	12	✓
11	T	The Datastore could be corrupted	13	✓
12	R	The Datastore denies writing data, although the data was potentially written	14	
13	I	Data flow sniffing	15	✓
14	D	The data flow from the Trusted Application to the Datastore is potentially interrupted	16	
15	D	Datastore inaccessible	17	

Here we detail how the SBD combines the AE scheme with the Merkle Tree to achieve its security goals. To allow for efficient random access, the SBD splits the data into blocks of fixed size. Each block is then processed using an AE scheme. The AE scheme encrypts and authenticates each block individually. Block processing complicates both protecting confidentiality and authenticity. Concerning confidentiality, when applying an AE scheme to individual blocks, identical plaintext blocks will encrypt to identical ciphertext blocks. We use an SBD global counter to prevent this. Every time a Data Block is encrypted using an AE scheme we supply the current value of the global counter as initialization vector (IV) to the AE operation, and then we increment the global counter. This ensures that the IV is unique for every encryption and therefore each ciphertext is different. We use a 96-bit (12 bytes) global counter for the SBD. This allows for 2^{96} write operations, before the SBD's AE key must be changed for this property to hold.

Protecting the overall data authenticity of the SBD is even more complicated. The Single-User Authentic Block Datastore threat model therefore allows for **spoofing**, **splicing**, and **replay** attacks on authenticity (cf. Section 2.11).

Spoofing The adversary is able to pass off arbitrary data as a valid Data Block. Spoofing is effectively prevented by the AE scheme. The AE operation generates an authentication tag that is verified upon authenticating decryption. If the Data Block, or the authentication tag is modified by an adversary, the authenticating decryption operation will fail.

Splicing, or relocation The adversary is able to substitute a valid block with a different valid block of the same SBD. We prevent splicing by adding the block index number as associated data to the AE operation and also through the use of a Merkle Tree.

Replay The adversary is able to successfully substitute a previously recorded copy of a specific Data Block as the newest version of the Data Block. We prevent replay attacks using a Merkle Tree. The Merkle Tree efficiently maps the authenticity of each individual block and their spatial and temporal dependency to a single representative, the Merkle Tree root hash. If the adversary replays an old block on a read operation, the root hash verification will fail, thus this attack is prevented.

We do not think it necessary to protect the confidentiality on the management blocks, because they only contain the IVs and authenticity tags for the Data Blocks. The IVs contain a serial number that allows an adversary to identify in which order the blocks have been written. However, this is an ability we have already conceded to the adversary in our adversarial model. So, the adversary will learn nothing more. The authenticity tags of the Data Blocks are created by the AE when protecting a Data Block and by definition must not reveal anything about the protected data. Finally, all non-public information in the header block is individually encrypted.

The SBD provides no data availability guarantees. In our threat model an adversary can easily mount a Denial-of-Service (DoS) attack, see threats 05, 06, 09, 14, and 15. As the Merkle Tree root hash is stored in a secure storage, and

therefore assumed to be secure against this adversary, such a DoS attack will always be detectable, but not preventable.

The final threat in our list to discuss is Threat 12. Threat 12 is a repudiation threat, where the Datastore repudiates ever having written data sent by the SBD. If the Datastore at one point acknowledges writing specific data, then this fact is documented in the Merkle Tree root hash. If the Datastore at a later point repudiates such a write operation, the root hash will be evidence to the contrary. On the other hand, if the Datastore is sent data for writing and it simply retains it while repudiating the receipt, then the SBD will have no way to detect this attack. However, we cannot think of a use for repudiating the receipt of encrypted and authenticity protected data other than denial of service.

6.5 Experimental Evaluation

All experiments were conducted on a iMX53 Quick Start Board⁵, equipped with a 1 GHz ARM Cortex-A8 processor.

6.5.1 Code Size

The Secure Block Device (SBD) adds very little to the Trusted Computing Base. Excluding the implementations of the cryptographic algorithms, the SBD C library has 2949 lines of code (LoCs)⁶, where 2029 LoCs implement the SBD core and 920 LoCs implement the Merkle Tree. Adapting the SBD to use ANDIX OS's storage abstraction requires an additional 47 lines of code. The adapters to use the reference implementations of Offset Codebook Mode (OCB) and Synthetic Initialization Vector (SIV) require 160 LoCs and 152 LoCs respectively. For comparison, the optimized reference implementation of OCB has 1207 LoCs.

6.5.2 Secure Block Device Block Cache Performance for Small Files

Profiling the stand-alone SBD code showed that a significant amount of computation time is spent performing the cryptographic operations. ANDIX OS adds an additional layer of cost to file read/write operations. Therefore we added the block cache component (cf. Section 6.3.3). The potential SBD block cache performance benefits depend on a number of factors including the cache size, the Trusted Application's access pattern, and the Normal World storage back end. To measure the impact of the block cache on the overall SBD performance we developed a performance test Trusted Application that can repeatedly read and write chunks of data, either with sequential or with random access patterns.

Figure 6.5 illustrates typical block access times that were measured with the performance test Trusted Application for a small file that almost fits into

⁵http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=IMX53QSB

⁶Determined with the `cloc` tool (<http://cloc.sourceforge.net>)

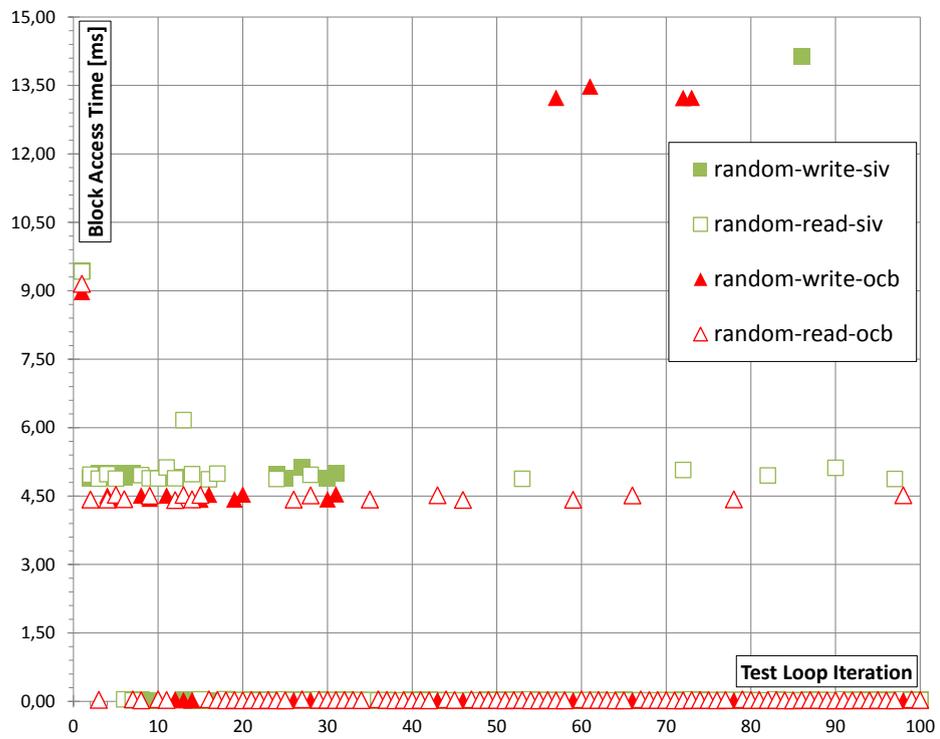


Figure 6.5: SBD Block Cache Access Times for Small Files

the block cache. For this experiment, measurements were done in a loop, with each test loop iteration consisting of a single block read or write operation to a random block position within the file. The horizontal axis of the graph gives the number of the test loop iteration. The block cache was configured for a total size of 32 KiB, organized as 16 blocks of 2 KiB bytes. The size of the test file was 32 KiB. With this choice of parameters most of the test file (up to 15 Data Blocks⁷) can be held within the block cache. The vertical axis shows the time, in milliseconds, needed to perform a single SBD block read or write operation from within the measurement Trusted Application. These times include all overheads related to SBD management, cryptography, and access to the Normal World back-end storage.

During the initial phase of each read and write test-series in this experiment, most of the read and write operations are clustered around the 4.5 ms mark. Here the cache misses are dominant, and the block cache is successively filled with data and management blocks. Block read and write access times during this phase mainly depend on the selected cryptography back-end, and the Normal World back-end storage. After roughly 20–30 test loop iterations, the cache has been filled, and the cache misses start to drop. This is evident from the solid line of data points on the 0 ms mark. Note that although mostly the red triangles (Δ) are visible, this is due to them occluding all other symbols. For cache hits the access time only depends on the SBD cache logic in the Secure World, and is typically in the order of tens of microseconds. Cache misses on block reads are in the same order as during the initial phase. Cache misses on block writes are commonly more expensive, and can be observed as spikes starting around test loop iteration 60 in Figure 6.5. When evicting a Data Block from the block cache, the SBD logic has to update the corresponding management block.

The SBD block access times for cache misses shown in Figure 6.5 are largely dominated by the time required to read from, or write to the Normal World. The current prototype version of the ANDIX OS interface to the Normal World file system, which is used by the SBD as storage backend, only supports synchronous, in-order scheduling of I/O operations. A more flexible implementation of the ANDIX OS Normal World interface, with support for asynchronous operations and Normal World I/O coalescing is part of currently ongoing work. We expect the block access times to significantly decrease, once the revised ANDIX OS Normal World interface is completed.

6.5.3 Experimental Setup

For the following two experiments we compile OpenSSL on an SBD mounted using the Linux Network Block Device (NBD) infrastructure. We argue this is a suitable benchmark as compiling is a common task and OpenSSL is a reasonably sized example. The Linux NBD kernel infrastructure is used to access block-devices, like for example disk partitions, over a (network) socket connection. Once an NBD device has been configured, it appears to the Normal World user-

⁷A hot cache always contains at least one management block.

Table 6.3: Average SBD read/write times and throughput depending on AE scheme

AE scheme	SBD		Throughput	
	read [ms]	write [ms]	read [MiB/s]	write [MiB/s]
None	0.443	0.477	11.191	6.690
OCB (AES)	1.214	1.167	4.078	2.705
SIV (AES)	1.656	1.575	2.990	2.045
AES+HMAC	2.393	2.080	2.073	1.497
OS	0.028	0.068	69.256	28.818

space as an ordinary block device. Filesystems stored on Linux NBD devices can be mounted with the standard Linux `mount` command. To setup a Linux NBD device for use, it has to be connected to an NBD server via a network socket. Once a Linux NBD device has been connected to its server, the Linux kernel starts to forward block read, and write operations over the network socket to the server.

The SBD instance we use for all experiments is backed by a Linux Ramdisk to ensure comparable results and provides 56 MiB of storage space. The 56 MiB are partitioned into 28672 Data Blocks and 224 management blocks at a block size of 2 KiB. Prior to each compilation, we create a new SBD and completely zero it out. We do this to eliminate spurious measurement artefacts, when the SBD grows in size. We also format the NBD device residing on top of the SBD with the ext2 file system. The cache size is again 32 KiB, with 16 blocks of 2 KiB each.

6.5.4 Impact of the Authenticated Encryption Scheme

A key performance factor of the SBD is the choice of Authenticated Encryption (AE) scheme. To evaluate its impact on the SBD performance, we compiled OpenSSL thrice for each supported AE scheme. To better compare the AE schemes and eliminate the effects of inter-world communication, we do *not* use a Trusted Application running in ANDIX OS. For this experiment all operations are performed on Linux. We measured the average time and amount of data processed for each `sbd_i_read` and `sbd_i_write` operation triggered by the compilation process. On average over all 12 compilation runs each compilation process required 2046 read and 31176 write operations. Table 6.3 records our results. Note that we omit standard deviations as they are all \ll 5% of the measurement values in magnitude.

Table 6.3 identifies four AE schemes: None, OCB, SIV, and AES+HMAC. For each of the schemes it records the average time required for an SBD read and write operation and the throughput. The “None” scheme in the first row does not use any AE scheme for encrypting the data. It only maintains the Merkle Tree over the management blocks, which in this case only contains all-

zero authentication tags. For comparison with the supported OCB and SIV AE schemes, we also implemented AES+HMAC, an encrypt-then-Message Authentication Code (MAC) construction using HMAC-SHA256 and AES in CBC mode of operation. As expected, OCB outperforms SIV, and SIV outperforms the HMAC+AES construction.

The goal of this experiment is to measure the impact of the AE scheme, therefore we use the “None” scheme as a baseline for the experiment. We evaluate the impact of the SBD in general in Section 6.5.5. However, we also recorded the runtime and throughput of the OS read and write operations used by the SBD to actually read and write data to the underlying Ramdisk. We present these values for reference in the OS row of Table 6.3.

6.5.5 Normal World Filesystem Encryption using the Secure Block Device

Although it is primarily intended for use in secure-world, the SBD framework can be employed to provide transparent filesystem encryption services to the Normal World operating system (OS). We developed a small Trusted Application prototype that provides an interface to an SBD instance to the Normal World that we integrate with the NBD infrastructure. The architecture of the experimental setup is depicted in Figure 6.6. The secure-world part (left side) mainly consists of our ANDIX OS operating system kernel, and a dedicated SBD Trusted Application. On the Normal World side we use a combination of the standard Linux NBD kernel infrastructure, the ANDIX OS TEE kernel driver, and a user-space daemon acting as NBD server. Similar to existing Linux kernel facilities, like encrypted loopback devices (`cryptoloop`) and the crypto device mapper `dm-crypt`, our proposed SBD based architecture provides full disk encryption at the block device level. However, our SBD Trusted Application never exposes encryption keys to the Normal World, thus providing key confidentiality, even when the Normal World kernel is compromised.

We again compiled OpenSSL to measure the performance of the SBD Trusted Application. During compilation of the OpenSSL library, the compiler toolchain produces a mixture of short-lived temporary files, and long-lived output (object) files. Writing of these output files is intermixed with read access to source files. Most of these file accesses can be efficiently cached by the Normal World Linux file-system caches, thus minimizing the number of block reads and writes to the underlying SBD instance. To establish a base-line, we first ran the entire build on the plain Linux Ramdisk, without using the SBD. For the actual performance measurements with the SBD, we ran the OpenSSL builds on ext2 file-systems residing on the (virtual) network block devices protected by the SBD instances.

Table 6.4 summarizes the measured performance results. We want to note that we measured the performance differently to Table 6.3. In Table 6.3 we measured the average SBD read/write times and throughput, instead of overall time to finish the compilation as we do in Table 6.4. The reason is that we believe for an application benchmark overall completion time is the most important

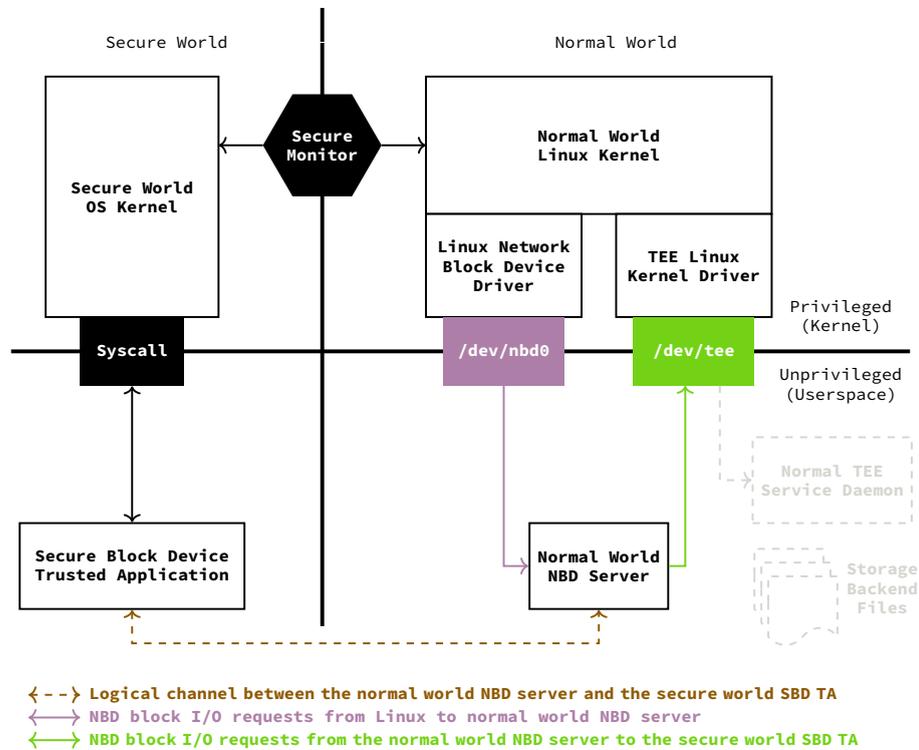


Figure 6.6: Using the SBD for Normal World file system encryption

factor.

The first row of Table 6.4 shows build times when building on a plain Ramdisk, without using any SBD device at all. Rows 2-5 give the times for building on an SBD device running directly on Linux, using the “None”, OCB, SIV, and HMAC+AES AE schemes respectively. The last two rows show results for SBD Trusted Application-backed block devices, using the “None” and OCB AE scheme. We omitted evaluating the SBD Trusted Application with SIV and HMAC+AES, as we have previously established that OCB is the fastest supported AE scheme.

The “Unpack” column shows the time required to unpack the OpenSSL source tarball. During this step a large number of new files is created, and the overall process is dominated by I/O activity. During the “Configure” phase, which is shown in the second column, the OpenSSL configure scripts are executed, and several small files and symbolic links are created. The “Compile” step shows the times for building the OpenSSL cryptography library itself. During the compile step, different parts of the compiler toolchain are called, and a number of temporary and output files are created, while most of the phase is dominated by computation. To somehow account for the impact of Linux file-system caches,

Table 6.4: Compile times for compiling OpenSSL on Secure Block Device protected block devices. The Randisk row is the baseline, where we compiled OpenSSL on an Network Block Device backed by a Randisk, with no Secure Block Device in place. The next four lines give the results for a Network Block Device using the Secure Block Device and backed by a Randisk with different AE schemes. Finally the last two lines present the results for a Network Block Device that uses a Trusted Application (TA) to implement the Secure Block Device in the Secure World. The output of the Secure Block Device Trusted Application is again stored in a Randisk in the Normal World. These last two lines reflect the results of the setup depicted in Figure 6.6.

Device	Unpack		Configure		Compile		Total	
	[s]	[1]	[s]	[1]	[s]	[1]	[s]	[1]
Randisk	2.494	1.000	20.062	1.000	968.527	1.000	991.083	1.000
SBD None	6.295	2.524	20.797	1.037	1161.713	1.199	1188.805	1.200
SBD OCB	11.538	4.626	20.939	1.044	1168.959	1.207	1201.435	1.212
SBD SIV	14.384	5.768	21.036	1.049	1171.213	1.209	1206.633	1.217
SBD HMAC+AES	19.188	7.694	21.391	1.066	1175.790	1.214	1216.369	1.227
SBD TA None	192.480	77.177	29.230	1.457	1449.670	1.497	1671.380	1.686
SBD TA OCB	210.580	84.435	30.390	1.515	1519.840	1.569	1760.810	1.777

we explicitly dropped the read caches⁸ before each test step and requested write-back of cached data via the `sync` command to the underlying block devices after each step. The `unpack`, `configure`, and `compile` columns show the sum of the time required for the actual command, and the following `sync` command for writing back the Linux file-system caches.

The results from Table 6.4 support the performance data, and the bottlenecks that we identified earlier in Section 6.5.2, and in Section 6.5.4. Most time in this experiment is spent when synchronously writing back blocks from the SBD to the Normal World back-end storage, as evidenced by the factor of 30.58 between using the SBD “None” and the an SBD Trusted Application “None” variants in the I/O heavy “Unpack” step. This significant factor is due to ANDIX OS copying each block of memory four times for inter-world communication and ANDIX OS only processing a single block at a time, instead of coalescing multiple blocks into a single bulk operation (see Section 6.6). Nevertheless, the total time difference between an unencrypted SBD Trusted Application, and the SBD Trusted Application using the OCB scheme is only 5.35 percent for the overall experiment.

At first glance the overall performance for I/O intensive tasks when using the SBD Trusted Application seems abysmal. This is evidenced by the factor of 77.177 between using a plain Ramdisk with no protection and the SBD in a Trusted Application providing full disk encryption and authenticity on the block device level. However, this factor is put into perspective by two observations. First, the total factor for compiling OpenSSL on a plain Ramdisk and using the SBD Trusted Application is just 1.777. We believe this factor to be reasonable. Second, note that the primary objective of our SBD framework is to provide a lightweight, simple-to-use mechanism for realizing potentially large amounts non-volatile storage with authenticity and confidentiality in Trusted Applications where the backing store for each SBD is handled by the Normal World OS. As such, we believe the SBD is useful to Trusted Applications and efficient enough for all Trusted Applications that do not require unceasing Datastore access. By using the SBD as a disk encryption utility we intentionally put the SBD through its paces to assess the limits of its applicability. Finally, we hope to alleviate much of the Trusted Application specific performance impact, with a more efficient ANDIX OS block access layer, as discussed in Section 6.6

6.6 Future Work

As mentioned in Section 6.5.2 and Section 6.5.5 ANDIX OS’s Normal World block access layer is currently one of the main performance limiting factors of the Secure Block Device (SBD) Trusted Application. It only supports synchronous, in-order scheduling of I/O operations. As future work for ANDIX OS, an implementation with support for asynchronous operations and Normal World I/O coalescing is planned. We expect the block access times to decrease significantly with a revised ANDIX OS Normal World block access layer.

⁸`echo 3 > /proc/sys/vm/drop_caches`

Currently, the SBD has no support for changing the Authenticated Encryption (AE) key AE_k . However, as pointed out in Section 2.9.3, Offset Codebook Mode (OCB) should not be used to create more than 2^{48} ciphertext blocks. Therefore, we want to add automated support for key changing to the SBD to mitigate any problems associated with overusing keys.

Another topic for future work is actually storing the bulk of the Merkle Tree in Normal World memory. As indicated in Section 6.3.4, we currently use precious Secure World memory to store the entire Merkle Tree. This is less than optimal, and as soon as ANDIX OS supports efficient access to Normal World memory we want to remedy this fact. On a related note, we currently use SHA-256 for our Merkle Tree. In the future, not only do we want to make the actual hash configurable, we also want to support Merkle Trees based on MAC functions, such as CMAC. MAC functions for Merkle Trees can have the advantage of smaller tag lengths and faster computation. Finally, as Rogers et al. [RCPS07] have shown it is sufficient to only protect the counter values instead of the complete management blocks with a Message Authentication Code (MAC). This is also something we want to retrofit into the SBD.

6.7 Conclusion

In an effort to implement a secure Mobile Agent System for disaster response that supports mobile platforms such as tablets and smartphones, we propose the Secure Block Device (SBD). The SBD is a flexible, and scalable mechanism to provide data confidentiality, integrity, and freshness for data at rest, using a Merkle Tree in conjunction with Authenticated Encryption (AE). Secure storage for data at rest is a fundamental building block for a secure Mobile Agent platform. Here, the SBD is a minimal Trusted Computing Base solution for applications that require fast and secure random block access to data, but do not require a fully fledged file system. It was designed for Trusted Applications running in an ARM TrustZone based Trusted Execution Environment, where the Trusted Execution Environment itself only provides storage mechanisms for cryptographic keys. It is easy to use, as the cryptographic operations for achieving the data security goals are as transparent as possible. It is flexible, and efficient in that it supports different AE schemes, and uses a block cache. Finally, the SBD is an open source C library that can be easily integrated into existing applications and useful even beyond its intended target as building block in a Mobile Agent platform. To demonstrate the SBD's applicability, we implemented a suitably well performing protected long term storage system.

7

Conclusions

In this section we summarize the conclusions of our contributions. These conclusions are a summary of the more detailed conclusions in the individual contribution chapters. We close with an outlook and a final remark.

7.1 Threat Modeling

As a first contribution we have created threat models of several aspects of disaster response. We have composed all our threat models with the Microsoft Threat Modeling Tool 2016. The Microsoft Threat Modeling Tool 2016 uses Data Flow Diagrams as models and STRIDE-per-interaction for threat enumeration.

First, we have threat modeled two high level processes pertaining to disaster response: situational awareness and command implementation. These two processes are critical to disaster response and our efforts have identified concrete threats to the information exchanges performed as part of these processes.

Our first observation here is that STRIDE-per-interaction's Data Flow Diagram models are well suited to map processes in disaster response. Data Flow Diagrams naturally capture the flow of information between entities involved in disaster response for gaining situational awareness and implementing commands. These information flows are the assets we want to protect, hence we found all threats generated by this methodology pertinent. We identified some minor caveats when using this methodology on a high abstraction level, such as the tool generating threats that are only applicable to information systems or having to use processes for entities such as people.

Our second observation concerns the abstractness of our models. Our two models for situational awareness and command implementation generalize the

many data flows that are part of these processes. For example, a single External Interactor is used to represent all first responders. Thus, although we believe the list of generated threats to grant valuable insights, we cannot qualify risks associated with a particular threat. For this the individual instances of the data flows need to be studied.

In addition to modeling the situational awareness and command implementation processes we have also created a detailed threat model that encompasses central aspects of the Secure Agent Infrastructure. Using this model we have identified a highly repetitive pattern, the Distributed Secure Agent Platform Outpost (DSAP Outpost). The DSAP Outposts are the agent platforms at the interface points of the Secure Agent Infrastructure with the outside world. We investigated the threats to the DSAP Outpost in detail and compared our results with the literature on Mobile Agent System security.

Our threat model of the DSAP Outpost yielded 54 pertinent threats. We used these 54 threats to analyze the effectiveness of our security solution comprising the Trusted Docking Station (TDS) and the Secure Docking Module (SDM). Of the 54 threats, 43 were related to a compromised agent platform. Therefore, we geared our security mechanisms towards ensuring the integrity of the agent platform (see Section 7.2). We also compared our findings with the Mobile Agent System security literature. Overall, we have identified 54 pertinent threats. Of these 54 threats, we consider 20 as being novel, 14 as being represented in literature to some degree, and 20 as being well known and described in literature. The main reasons for the novel threats is the lack of discussion on repudiation and impersonation threats in our literary sources and their limited scope. However, we concur with literature that a compromised agent platform is the most relevant source of threats to a Mobile Agent System. Our conclusion is that a literature study cannot replace threat modeling the actual system.

Finally, we want to document that the Data Flow Diagram and STRIDE-per-interaction based threat modeling methodology worked well to identify threats to the communication between the DSAP Outpost and the Core Secure Agent Infrastructure and the External Interactors, respectively. In addition, we used this methodology to model agent transfer and instantiation, and it yielded new insights into this well studied area of Mobile Agent System security.

7.2 The Trusted Docking Station and the Secure Docking Module

To mitigate the threats identified by our Distributed Secure Agent Platform Outpost (DSAP Outpost) threat model we have introduced our security solution comprising the Trusted Docking Station (TDS) and the Secure Docking Module (SDM). The TDS provides an execution platform that measures the load-time integrity of the software it executes and provides virtualization based security by isolation as a runtime protection mechanism. The SDM is a user authentication and platform verification module. It mutually authenticates the DSAP Outpost

and its human user to each other, while also establishing that the software configuration of the DSAP Outpost conforms to a given security policy.

We have designed, implemented, and evaluated both the TDS and the SDM. Our TDS implementation is based on the acTvSM platform [Pir15] and our SDM uses a commercially available Security Controller. We have evaluated both, the performance of the security solution and its security. Our conclusion concerning the performance of our security solution is that the incurred overhead is negligible for disaster response scenarios. The result of our security evaluation is that our DSAP Outpost is still vulnerable to targeted, online attacks. In a targeted online attack an adversary exploits a runtime vulnerability to gain access to a TDS and the software it is hosting. However, our TDS offers load-time integrity protection on commodity off-the-shelf hardware equipped with Intel's TXT. Furthermore, the SDM enables user authentication and TDS load-time integrity verification with a commercially available Security Controller. Within these limits our security solution mitigates all but three of the 54 threats identified by our threat modeling efforts.

7.3 The Secure Block Device

We have developed the Secure Block Device as a first key component for a Distributed Secure Agent Platform Outpost (DSAP Outpost) security solution based on the ARM TrustZone security extensions. The Secure Block Device is a flexible, and scalable software solution to provide data confidentiality and authenticity for a Single-User Block Datastore using a Merkle Tree and Authenticated Encryption (AE). The Secure Block Device is a minimal Trusted Computing Base solution for applications that require fast and secure random block access to data, but do not require a fully fledged file system. It was designed for Trusted Applications running in an ARM TrustZone based Trusted Execution Environment, where the Trusted Execution Environment itself only provides storage mechanisms for cryptographic keys. It is easy to use, as the cryptographic operations for achieving the data security goals are as transparent as possible. It is flexible, and efficient in that it supports different AE schemes, and uses a block cache. Finally, the Secure Block Device is an open source C library that can be easily integrated into existing applications. We have evaluated the Secure Block Device by implementing a suitably well performing long term storage system. This long term storage system would be a core component of our ARM TrustZone based mobile DSAP Outpost security solution.

7.4 Outlook

With our threat modeling efforts we have created models for high level disaster response processes and the Secure Agent Infrastructure, the particular Mobile Agent System we use. We believe this to be a valuable contribution to the fields of threat modeling, mobile agent security, and disaster response se-

curity. However, as part of our research we identified areas for further study. *First*, we believe that detailed risk modeling using the high level threat models we created could help to identify those aspects of disaster response that need high security solutions and those where lesser security solutions suffice. We think this worthwhile, as we don't believe that a blanket high security solution can be realistically deployed. *Second*, concerning our threat model for the Distributed Secure Agent Platform Outpost (DSAP Outpost), we would like to further investigate the threats to particular physical manifestations in specific operational environments. The DSAP Outposts are the interface points of the Secure Agent Infrastructure and potential realizations range from first responder's mobile phones to servers in data centers. Given this diversity, we believe that there exists a number of disjoint, concrete threats. Furthermore, we believe it would be interesting to investigate these threats using the attack tree threat modeling methodology.

Security and availability are key factors for Mobile Agent Systems to gain acceptance as disaster response support tools. We hope that our work protecting the DSAP Outposts helps to increase the acceptance of Mobile Agents for disaster response so that future disaster response operations can leverage the power of Mobile Agents to prevent further harm to humans beings enduring a disaster situation. However, our security solution is far from perfect. There is still a number of implementation and research issues to tackle. *First* of all we would like to change the protection of the communication channel between the Secure Docking Module (SDM) and the Trusted Docking Station (TDS) to use Authenticated Encryption (AE) instead of mere encryption. Thus we want to prevent potential malleability attacks on the encrypted data. *Second*, we would like to change the SDM's transmission interface from using a smart card interface via USB to directly using USB or SPI to increase throughput and remove data transmission as a limiting factor. *Third*, the security of the DSAP Outpost could further benefit from the TDS/SDM providing additional security services, such as end-point authentication for a secure communication protocol between the DSAP Outpost and the Core Secure Agent Infrastructure. *Fourth*, we believe the security of the SDM is limited by it disclosing authentication material to the TDS. Although this solution is very versatile concerning the used authentication mechanisms, we consider investigating how the SDM can provide flexible authentication services, without the need to disclose credentials, to be a fruitful research topic.

With the Secure Block Device we have realized one of the key components of a security solution for DSAP Outposts for mobile devices based on the ARM TrustZone security mechanisms. However, the Secure Block Device is just a first step in this direction and there is significant need for further research on how to create a high quality security solution. Nevertheless, given the ubiquity of smart mobile devices and their potential utility to disaster response, we believe this research effort would be beneficial.

7.5 Final Words

As final remark I would like to freely quote Mikko Hyppönen, Chief Research Officer of F-Secure from his talk at the TrustCom 2015 conference:

... [at the beginning of the talk] I have some good news and some bad news for you. The good news are you will always have a job in computer security. ... [at the end of his talk listing all the threats we currently face and will face in the future] The bad news are, you will always have a job in computer security.



Threat Modeling Agent Migration and Communication

A.1 Introduction

In this appendix we model the threats to agent migration and communication in the Secure Agent Infrastructure and discuss each threat in detail. This appendix is a companion piece to Section 4.5 where we discuss the overall results, while here we investigate the individual threats. Specifically in this appendix, we analyse the Distributed Secure Agent Platform Outpost (DSAP Outpost). The DSAP Outpost is the template of an outlying Distributed Secure Agent Platform (DSAP) that receives a Mobile Agent. This agent at the DSAP Outpost then contacts an External Interactor using the DSAP as execution platform and potentially sends results back to the Core Secure Agent Infrastructure. The DSAP Outpost is a highly repetitive pattern in the Secure Agent Infrastructure. Therefore we developed security mechanisms to protect it. These mechanisms, the Secure Docking Module (SDM) and the Trusted Docking Station (TDS) were introduced in Chapter 5. We have used the threats we discuss in this appendix to evaluate the effectiveness of these security mechanisms.

A.2 Threat Analysis Procedure

Here we create a threat model for agent migration and communication in the Secure Agent Infrastructure. We use the following procedure. We model a Distributed Secure Agent Platform Outpost (DSAP Outpost). The DSAP Outpost is an outlying component of the Secure Agent Infrastructure that is placed

physically close to an external entity the Secure Agent Infrastructure wants to communicate with. We create this model using the Microsoft Threat Modeling Tool 2016 introduced in Section 2.8.4. In the Microsoft Threat Modeling Tool 2016 we create a Data Flow Diagram of the DSAP Outpost and the tool uses STRIDE-per-interaction (see Section 2.8.5) to generate a list of threats.

We then discuss each threat individually. First, we give the threat title as generated by the Microsoft Threat Modeling Tool 2016 including its threat number as assigned by the tool. Note that due to our dividing of the threats into logical groups, the threat numbers are not necessarily consecutive. Second, we briefly describe the threat. This description is based on the threat description generated by the Microsoft Threat Modeling Tool 2016. In a few cases we directly copy parts or the whole of the description, but primarily we adapt the description and put it into the context of the Secure Agent Infrastructure Mobile Agent System. Third we establish if we think this threat to be pertinent in our disaster response setting. Fourth we discuss if and how the threat is represented in literature. For this, we use the works by Jansen and Karygiannis [JK99, Jan00], Borselius [Bor02], and Bierman and Cloete [BC02]. See Section 2.4 for a more detailed discussion of the prior art in Mobile Agent security. Finally, if applicable, we discuss how well the Data Flow Diagram model and the STRIDE-per-interaction generated threat apply to a specific aspect of the Secure Agent Infrastructure Mobile Agent System.

A.3 Distributed Secure Agent Platform Outpost Model

A.3.1 Structure

Our model consists of four entities and eight data flows. Starting with the entities, first we have the *Core Secure Agent Infrastructure* External Interactor. The Core Secure Agent Infrastructure interactor comprises a number of components, such as the Process Management Subsystem, the Agent Repository, and the Resource Lookup System (see Section 2.6). The next entity is the *Distributed Secure Agent Platform (DSAP)* process that handles agent migration, agent execution and agent communication with the sending Secure Agent Infrastructure component. The *Mobile Agent* itself is also represented as a process that is instantiated by the DSAP and communicates with the Core Secure Agent Infrastructure via the DSAP, and directly with the last entity, the *External Interactor*. The External Interactor subsumes human users and external information systems the agent might communicate with.

We group the data flows into three groups. These groups are *agent migration*, *agent – Core Secure Agent Infrastructure communication*, and *agent – External Interactor communication*. The first, the agent migration group consists of the *Send Agent* and *Instantiate Agent* data flows. The second group, the agent – Secure Agent Infrastructure communication group consists of the *SAI → DSAP*, *DSAP → SAI*, *MA → DSAP*, and the *DSAP → MA* data flows. The

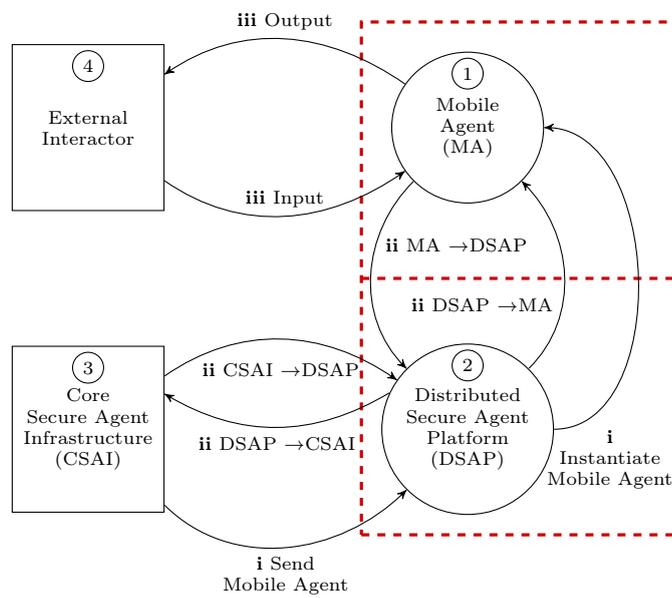


Figure A.1: A Data Flow Diagram modelling the Secure Agent Infrastructure typical process of sending a Mobile Agent to a Distributed Secure Agent Platform to gather information from an external information system, or converse with a system user. Here we model both, the agent migration and the information exchange between the Core Secure Agent Infrastructure and the Mobile Agent running on the Distributed Secure Agent Platform. We use this model as input to the STRIDE-per-interaction analysis using the Microsoft Threat Modeling Tool 2016.

final agent – External Interactor communication group comprises the *Input* and *Output* data flows. We discuss these groups separately in Section A.5 (Agent Migration), Section A.7 (Agent Core Secure Agent Infrastructure Communication) and Section A.6 (Agent External Interactor communication). Our main reason for this taxonomy is to break up the long list of semi-associated threats and put them into logically related categories. For our model we assume that the SAI → DSAP, the DSAP → SAI, and the Send Mobile Agent data flows are protected using a secure communication protocol (see Section A.4). Therefore, we configure these data flows as source and destination authenticated, providing confidentiality, and providing integrity.

A.3.2 Modelled Workflows

On the one hand the model represents the Core Secure Agent Infrastructure sending a Mobile Agent (an Information Delivery Agent) to a DSAP to gather information from an external information system and delivering this information back to the sending component in the Core Secure Agent Infrastructure. First the Information Delivery Agent is sent to the DSAP using the *Send Mobile Agent* data flow. Then the DSAP deserializes the Information Delivery Agent and instantiates it over the *Instantiate Agent* data flow. Then the Information Delivery Agent queries the External Interactor, for example a database, via the *Output* data flow. It receives its response via the *Input* data flow, processes the data and sends back its results to the Core Secure Agent Infrastructure via the *MA → DSAP* and *DSAP → CSAI* data flows.

On the other hand the model represents the Core Secure Agent Infrastructure sending a User Communication Agent to a DSAP to interact with a human user. Here, the Core Secure Agent Infrastructure uses the User Communication Agent to send status updates or ask the user for command decisions. First the User Communication Agent is sent out to the DSAP over the *Send Mobile Agent* data flow. Then the DSAP deserializes the User Communication Agent and instantiates it over the *Instantiate Agent* data flow. Next the User Communication Agent starts communication with the user via the *Input* and *Output* data flows. The User Communication Agent sends updates to the Core Secure Agent Infrastructure, which we modeled using the *MA → DSAP* and *DSAP → CSAI* data flows. Finally, the User Communication Agent can receive updates from the Core Secure Agent Infrastructure. We modeled this with the *CSAI → DSAP* and *DSAP → MA* data flows.

A.3.3 Limitations of the Secure Agent Infrastructure Compared to a Generic Mobile Agent System

The Secure Agent Infrastructure has two properties that greatly simplify threat modeling, when compared to a generic Mobile Agent System. First, the Secure Agent Infrastructure only supports single-hop mobility. The Core Secure Agent Infrastructure sends out an agent to DSAP, and the Mobile Agent will not migrate any further. This precludes any threats that arise from Mobile Agent

freely migrating between agent platforms. One example of such threat is that of another agent, or an agent platform compromising an agent and then sending it to the next agent platform.

Second the Secure Agent Infrastructure does not support inter-agent communication. Although a DSAP can host several agents in parallel, the DSAP provides no facilities for agents to directly communicate with each other. Secure Agent Infrastructure agents only communicate with the Core Secure Agent Infrastructure and External Interactor. This precludes all attacks arising from agent-to-agent communication.

A.4 Secure Agent Infrastructure Security Assumptions

A.4.1 Secure Core Secure Agent Infrastructure

For this threat modelling activity we assume that the Core Secure Agent Infrastructure adheres to a security policy that, in layman's terms, precludes the Core Secure Agent Infrastructure from misbehaving. In a little more detail, the Core Secure Agent Infrastructure will not act against its specification and its security is inviolable to attacks. See Section 4.5.3 for a more formal specification of Core Secure Agent Infrastructure security.

A.4.2 Secure Mobile Agents

The Secure Agent Infrastructure assumes that Mobile Agents are vetted for their adherence to the Secure Agent Infrastructure security policy and only agents that check out are certified for use with the Secure Agent Infrastructure. For threat modelling activities we assume that Mobile Agents in the Core Secure Agent Infrastructure are inviolable to attacks. Only when a Mobile Agent is sent to a Distributed Secure Agent Platform (DSAP) do we consider threats to it.

A.4.3 Secure Communication Channel

We assume that all data flows between the Core Secure Agent Infrastructure and the DSAP are communicated using an authentic, integrity protected, and confidential channel. We call a communication channel that provides authenticity, integrity, and confidentiality a secure communication channel. The Jini technology underlying the Secure Agent Infrastructure allows for a number of communication mechanisms that can be configured to provide a secure communication channel (see Section 2.5.2). For example, Jini supports HTTPS and Transport Layer Security (TLS). TLS with client certificate authentication using a cryptographically secure cipher suite can provide a secure communication channel. Furthermore, we assume that the implementation of the secure communication channel is free of security vulnerabilities, precluding for example buffer

overflows that allow remote code execution, the use of insecure cipher suites, or misconfigured certificates.

A.4.4 Distributed Secure Agent Platforms and External Interactor

The DSAPs are the outposts of the Secure Agent Infrastructure. Mobile Agents migrate to DSAPs to be physically close to the entities they interact with. On the one hand these entities are human users interacting with the Secure Agent Infrastructure to fulfill their crisis management duties, on the other hand these entities are information systems providing useful services and information for crisis management.

There are few assumptions we can make on the security of these Distributed Secure Agent Platform Outposts (DSAP Outposts). In our setting concrete DSAPs can range from first responders using their own private mobile smart-phones to well maintained and protected dedicated systems in both public and governmental organizations.

A.5 Agent Migration

Figure A.2 depicts the part of the Distributed Secure Agent Platform Outpost (DSAP Outpost) model relevant to agent migration. We have decided to model agent migration using two data flows. First, the agent is sent from the Core Secure Agent Infrastructure to a Distributed Secure Agent Platform (DSAP). This is modelled by the data flow *Send Mobile Agent*. Then the DSAP instantiates the agent and starts executing it. We use a single data flow (*Instantiate Agent*) to model this procedure. This is outside of what a normal data flow models within STRIDE-per-interaction, nevertheless we think it works reasonably well.

A.5.1 Send Mobile Agent

This data flow models the Core Secure Agent Infrastructure sending out an agent to a DSAP for execution.

58. (Repudiation) Potential Data Repudiation by the Distributed Secure Agent Platform

The DSAP claims that it did not receive the agent from the Core Secure Agent Infrastructure. We believe this is a pertinent threat. For example, a compromised DSAP might want to cover its tracks. Let us assume this compromised DSAP wants to prevent access to the service it is associated with. Instead of plainly refusing to grant access to the associated service, the compromised DSAP claims never to have received any agents that would consume the service and thus obfuscating the Denial-of-Service attack.

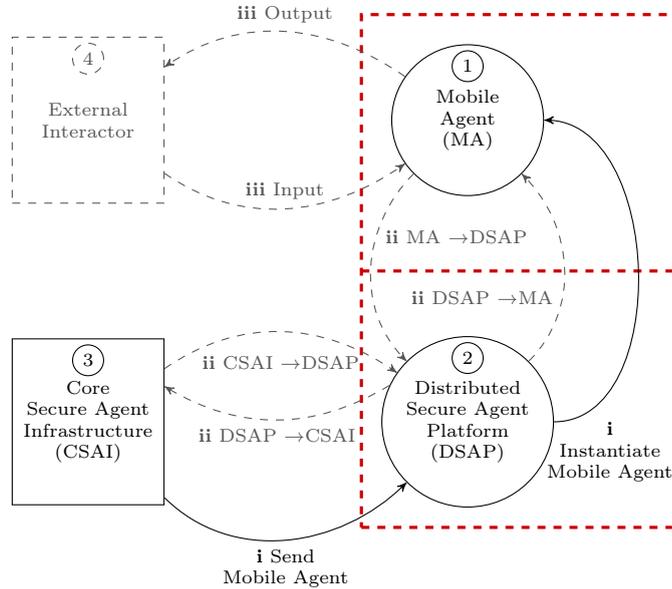


Figure A.2: The part of our Data Flow Diagram model we use to analyze the threats to a Mobile Agent when it is being sent to an DSAP Outpost and instantiated there.

Interestingly, of our three main sources for agent security, that is, Jansen and Karygiannis [JK99], Bierman and Cloete [BC02], and Borselius [Bor02] (see Section 2.4) lack this particular threat. While Jansen and Karygiannis mention repudiation in the context of agents interacting with other agents, and Borselius points out the importance of non-repudiation in all agent communications, all three Mobile Agent security discussions lack this particular threat.

59. (Denial Of Service) Potential Process Crash or Stop for Distributed Secure Agent Platform

The DSAP crashes, halts, stops or runs slowly; in all cases violating an availability metric. Given the nature of the disaster response use case we consider for agent migration, our assessment is that this threat is very pertinent. In fact, one of the advantages of using a Mobile Agent System in a disaster response context is this particular paradigm's resilience to intermittent network failures. We believe it is important to plan for this threat and provide for alternative means of communication, such as the plain old voice call.

Jansen and Karygiannis discuss Denial-of-Service multiple times in distinct threat categories. They discuss Mobile Agents attacking each other (agent-to-agent threat category), the hosting agent platform (agent-to-platform), and the agent platform delaying or denying services to the agent (platform-to-agent). The first two categories do not apply to this threat. The third threat category

discusses the platform attacking agents, but not the platform being unavailable, when an agent is transmitted. Finally, Jansen and Karygiannis do discuss an external (w.r.t. the Multi-Agent System) adversary attacking the availability of the agent platform in their other-to-agent threat category. This threat is closest to the threat we discuss here. Borselius points out the importance of availability in communication, and acknowledges that a malicious agent platform can deny executing a Mobile Agent. Finally, Bierman and Cloete identify three threat subclasses for Denial-of-Service, which they call availability refusal. These three threat subclasses are Denial-of-Service, delay of service, and transmission refusal. Whereas the Denial-of-Service and delay of service subclasses discuss Denial-of-Service threats to executing a Mobile Agent, the transmission refusal category documents that a malicious agent platform can refuse to send a Mobile Agent to the next agent platform. Threats to executing a Mobile Agent will be treated in more detail in Threat 17.

The execute relationship between the DSAP and the Mobile Agent is one of the areas that is difficult to model using a Data Flow Diagram and STRIDE-per-interaction and this Denial-of-Service threat here is a symptom. However, this is compensated by considering the Mobile Agent during operation, that is, while it interacts with an External Interactor or communicates with the Core Secure Agent Infrastructure.

60. (Denial Of Service) The Data Flow *Send Mobile Agent* Is Potentially Interrupted

An external agent interrupts data flowing across a trust boundary in either direction. Targeted interruption of the Mobile Agent transfer can prevent the Mobile Agent from fulfilling its objectives. As we assume an authenticated, encrypted, and integrity protected connection between the Core Secure Agent Infrastructure and the DSAP, the interruption will be detected at least at the sending Core Secure Agent Infrastructure side and the Core Secure Agent Infrastructure can then react to this problem by e.g. trying a different route to the target, using a different communication carrier, etc.

As with Threat 59, our assessment is that this threat corresponds with Borselius' need for availability and Jansen and Karygiannis' Denial-of-Service threats in their other-to-agent threat category. Furthermore, our threat here is somewhat related to Bierman and Cloete's transmission refusal threat category.

61. (Elevation Of Privilege) Elevation Using Impersonation

Microsoft Threat Modeling Tool 2016 states that a DSAP may be able to impersonate the context of a Core Secure Agent Infrastructure in order to gain additional privilege. Microsoft's Windows operating systems have a specific impersonation mechanisms. Impersonation in Windows allows an entity to gain all the access rights of another entity, by use of an impersonation token. The impersonation token is a piece of data and as such transferable. To give an example in our setting, if a process or user in the Core Secure Agent Infras-

tructure generates an impersonation token and sends it to a DSAP, this DSAP will inherit all access rights from the generating entity. We believe that Window's impersonation mechanism is the reason why Microsoft Threat Modeling Tool 2016 generates this threat. We make no assumptions about using specific Microsoft products, so is the threat pertinent at all?

No, this particular threat is not pertinent, because the Secure Agent Infrastructure does not support any form of impersonation. However this threat touches on a security relevant property of the Secure Agent Infrastructure and we want to use the mantle of this threat to discuss it. And therefore we also let the threat stand as pertinent.

In the Secure Agent Infrastructure all components operate with the authority of the Core Secure Agent Infrastructure. The security of the Secure Agent Infrastructure is based on the assumption that no one can inject an agent into the Secure Agent Infrastructure or impersonate any component in the Secure Agent Infrastructure. The security assumptions that the Core Secure Agent Infrastructure is secure (see Section A.4.1) and that the communication is secure (see Section A.4.3) preclude impersonation of any Core Secure Agent Infrastructure component and thus also injection of an unauthorized agent from the Core Secure Agent Infrastructure domain. However, we make no such assumptions about the security of the DSAP, hence the DSAP could violate the Secure Agent Infrastructure security policy, due to, for example, outside interference by an adversary.

The Mobile Agent security literature is deeply concerned with a compromised platform interfering with Mobile Agent execution, and if the Secure Agent Infrastructure were using a mechanisms equivalent to Microsoft's impersonation tokens, then eavesdropping by a compromised platform would be a pertinent problem. This is documented by Borselius in his discussion of agent mobility, by Jansen and Karygiannis in their discussion of the platform-to-agent threat category, and also by Bierman and Cloete who classify this as confidentiality attack. Although all three of our literature sources discuss the threat of an agent platform eavesdropping on a mobile agent, none of our sources points out the threat of an agent platform using this information to impersonate the agent.

We want to discuss this threat as generated by STRIDE-per-interaction when threat modeling the Secure Agent Infrastructure. Before we started to analyse the Secure Agent Infrastructure, we investigated agent migration in a generic Mobile Agent System using a similar model. In a first iteration we did not identify this as a potential threat for a Mobile Agent System. Only when we analysed the threats to the Secure Agent Infrastructure, we realized that this threat might be a problem. The reason for this is that it depends on the actual Mobile Agent System implementation, and how this system handles authorization, if this is a threat. We take this as a weak indication that STRIDE-per-interaction is more useful when modelling the threats for concrete systems, because when put in context, we found it much easier to actually *see* the threat.



Figure A.3: A bug called Feature

62. (Elevation Of Privilege) The Distributed Secure Agent Platform May be Subject to Elevation of Privilege Using Remote Code Execution

The Core Secure Agent Infrastructure may be able to remotely execute code for DSAP. We consider the Core Secure Agent Infrastructure as secure and therefore preclude a compromised Core Secure Agent Infrastructure sending anything but certified Mobile Agent to any DSAP. We therefore disregard the threat.

Elevation of privilege by exploiting a remote code execution vulnerability is a common threat that is the source of a plethora of security issues. However, in our setting this is the proverbial feature, and not a bug (see Figure A.3). The numerous publications on securing Mobile Agent System, including this thesis, are a very good indication of the problems stemming from turning a security threat into a feature. In brief, the trick here is to mitigate all the other threats in this list, and all the threats we have not found, to make this feature safe to use.

Jansen and Karygiannis have their own threat category (agent-to-platform) that discusses how a malicious agent can interfere with the normal operation of an agent platform. Specifically they consider agents impersonating other agents, agents performing Denial-of-Service attacks on the hosting agent platform, and how agents can try to get unauthorized access to sensitive data and services on the agent platform. Borselius concedes that hosts need to be protected from agents and other parties that can communicate with an agent platform, but also claims that the problems associated with the protection of hosts from malicious code are quite well understood.

63. (Elevation Of Privilege) Elevation by Changing the Execution Flow in the Distributed Secure Agent Platform

An adversary may pass data into the DSAP in order to change the flow of program execution within the DSAP to the adversary's choosing. As we assume a confidential, integrity protected and authentic connection between the sending Core Secure Agent Infrastructure and the receiving DSAP an outside adversary should not be able to inject any data into the DSAP. Furthermore, we consider the Core Secure Agent Infrastructure as secure and therefore preclude a compromised Core Secure Agent Infrastructure exploiting the target DSAP. For these reasons we disregard this particular threat.

The discussion of this threat however indicates the following issue. A compromised DSAP might try to make it look like it was compromised by the Core Secure Agent Infrastructure in order to cover the tracks of the real attack vector used. This we consider a threat.

Jansen and Karygiannis warn against unauthorized access against agent platforms by conventional attack scripts that subvert the OS. Borselius documents that hosts need to be protected from other parties that can communicate with the platform and claims that the problems associated with the protection of hosts from malicious code are quite well understood.

Literature does not consider an agent platform implicating a sending platform.

64. (Elevation Of Privilege) Cross Site Request Forgery

Cross Site Request Forgery is a web browser specific elevation of privilege attack. As we do not use a web browser to neither send nor receive the Mobile Agent we disregard this threat here.

A.5.2 Instantiate Agent

This data flow represents the deserialization and instantiation of the Mobile Agent previously sent by the Core Secure Agent Infrastructure. With this interpretation of the data flow we leave the confines of what STRIDE-per-interaction was developed for, so here we take some liberties in reinterpreting the threats.

12. (Spoofing) Spoofing the Distributed Secure Agent Platform Process

The DSAP may be spoofed by an adversary and this may lead to unauthorized access to Mobile Agent. Our security assumptions specifically preclude sending a Mobile Agent anywhere, but to an authentic DSAP, by mandating a secure communication mechanism. However, our adversarial model allows for an adversary to subvert an authentic DSAP. In this case the adversary would gain full access to the Mobile Agent. As such we consider this threat very pertinent.

This particular threat is an enabling threat, because if it is realized, it empowers the adversary to sniff the Mobile Agent code and data, tamper with

the agent code and data, and decide not to execute the agent at all, or execute it in a way that violates its availability constraints. The threats resulting from a compromised platform gaining access to a Mobile Agent are well documented in literature by all our primary sources. Also the threat of an agent platform to masquerade as another agent platform, a threat we preclude based on our security assumptions, is described by Jansen and Karygiannis in their platform-to-agent threat category, and by Bierman and Cloete in their threat class authentication risks, subclass Masquerading. However an authentic platform becoming compromised is not explicitly documented.

Although the intent of the STRIDE-per-interaction model that generates this threat is to model one process communicating with another, and not one process instantiating the other, this threat is accurate in our context, with only minimal reinterpretation.

13. (Spoofing) Spoofing the Mobile Agent Process

The Mobile Agent may be spoofed by an adversary and this may lead to information disclosure by DSAP. The STRIDE-per-interaction model's intent is to model DSAP sending data to a Mobile Agent. In this interpretation the Mobile Agent is sending data to itself. Our interpretation is that the DSAP instantiates the Mobile Agent. Our security assumptions state that the Core Secure Agent Infrastructure only introduces vetted and certified agents into the system and that a conforming DSAP will only receive Mobile Agents over a secure channel. Therefore, we disregard this threat.

The threat of an agent masquerading as another agent is well documented in literature. Jansen and Karygiannis discuss this threat under the heading Masquerading in their agent-to-platform threat category. They point out that a masquerading agent might be able to gain unauthorized access to resources, or misbehave under another agents identity to shift blame. Borselius briefly discusses identification, authentication and authorization of agents. He points out that authentication is often fundamental to secure communication.

14. (Tampering) Potential Lack of Input Validation for Mobile Agent

Data flowing across Instantiate Agent may be tampered with by an adversary. As with Threat 13 our security assumptions preclude injection of a non-policy conforming Mobile Agent into a conforming DSAP. However, we allow for a compromised DSAP to tamper with the Mobile Agent while it is instantiated. A compromised DSAP might compromise a Mobile Agent to shift blame. We consider this threat pertinent.

The threat of a tampered, or malicious agent wrecking havoc in a Mobile Agent System is well documented in literature. Jansen and Karygiannis pay tribute to this with their discussion of threats in the agent-to-agent, agent-to-platform, and other-to-agent platform threat categories. Also Borselius concedes that hosts need to be protected from agents and other parties that can communicate with the platform.

The threat of a compromised DSAP tampering with the Mobile Agent is one of the core threats inherent in the Mobile Agent paradigm. Jansen and Karygiannis extensively discuss this threat under the heading Alteration in their platform-to-agent threat category. They note that a Mobile Agent arriving at an agent platform exposes its code, state and data to the platform. Therefore, the agent platform can modify the agents code, state and data. They especially emphasize the threat of an agent platform maliciously modifying a Mobile Agent and sending it on to further agent platforms. This is a threat we can disregard, because the Secure Agent Infrastructure only supports single-hop mobility. Bierman and Cloete categorize security issues pertaining to modifying a Mobile Agent as integrity attacks, where they distinguish between interference and information modification. They state that integrity interference occurs when the executing host interferes with the Mobile Agent's execution mission, but does not alter any information related to the agent. However, they cite transmitting the agent incorrectly as an example, which we would categorize as a clear alteration of the agent. In their information modification they document the threats of an agent platform altering, corrupting, manipulating, deleting, misinterpreting, or incorrectly executing a Mobile Agent's code, data, control flow, or status. They also point out that an agent platform can interfere with the interaction between different agents, and can alter the communication between them for its own benefit. Finally, Borselius also points out that due to agent mobility, the agent platform can observe and modify agent code, data, and state.

15. (Repudiation) Potential Data Repudiation by the Mobile Agent

The Mobile Agent claims that it did not receive data from a source outside the trust boundary. The STRIDE-per-interaction model's intent is to model DSAP sending data to a Mobile Agent. In this interpretation the Mobile Agent is sending data to itself. Our interpretation is that the DSAP instantiates the Mobile Agent. We cannot make any sense of this threat in neither interpretation, therefore we treat it as an artifact of using STRIDE-per-interaction outside its intended purpose and ignore this threat.

16. (Information Disclosure) Data Flow Sniffing

Data flowing across *Instantiate Agent* may be sniffed by an adversary. As a Secure Agent Infrastructure Mobile Agent can contain, or relay sensitive information we consider this threat pertinent. The threat to sensitive information relayed by the agent will be discussed in the agent communication section (cf. Threat 1, Threat 5, Threat 30, and Threat 51). The Secure Agent Infrastructure can send out agents that contain sensitive information such as credentials, or where the code itself contains valuable Intellectual Property. Our security assumptions preclude information leakage during transport, but a compromised DSAP can eavesdrop this sensitive information. Therefore, we consider this a pertinent threat.

Literature considers this one of the core threats inherent in Mobile Agent

Systems. Jansen and Karygiannis point out that Mobile Agents exacerbate the problem of eavesdropping as agent platforms have full access to a Mobile Agent's code, state and data and can directly eavesdrop this information. Furthermore, they indicate that agent platforms might already learn sensitive information by just observing the agent and that agent platforms can listen in on all agent communications. Borselius likewise points out that due to mobility, the executing platform can observe code, data and flow control, and can eavesdrop agent communication. Bierman and Cloete discuss confidentiality attacks, where they consider eavesdropping, theft, and reverse engineering. They classify eavesdropping as passively listening in on the agents information or intercommunication. They specify theft to include removing the eavesdropped information from the attacked agent, and finally, their interpretation of reverse engineering considers analysing a Mobile Agent's data and state in order to manipulate future or existing agents.

17. (Denial Of Service) Potential Process Crash or Stop for the Mobile Agent

The Mobile Agent crashes, halts, stops or runs slowly; in all cases violating an availability metric. Our security assumptions only allow for vetted and certified Mobile Agents to be introduced by the Core Secure Agent Infrastructure. In a security policy conforming DSAP this precludes the Mobile Agent itself from violating its own availability constraints. Furthermore, availability violations because of external interactions or home platform communication are treated in the respective threats (cf. Threat 6, Threat 25, Threat 31, Threat 39, Threat 41, Threat 42, and Threat 52). A compromised DSAP however can easily prevent the Mobile Agent from being instantiated or executed at all, and it could execute it in a way that violates the Mobile Agent's availability constraints, for example by giving the agent too little compute time to perform its task. We consider this is a pertinent threat.

All three of our primary sources document the threat of an agent platform denying or delaying services to a Mobile Agent. Borselius specifically mentions incorrect execution of code and denial of execution in his discussion of mobility. Jansen and Karygiannis extensively discuss Denial-of-Service in their platform-to-agent threat category, indicating that an agent platform can ignore agent requests, introduce unacceptable delays for critical tasks, or simply not execute the agent. They also describe agent lifelock, where a compromised platform continuously creates more work for an agent it hosts, such that this agent can never complete its task. Finally, Bierman and Cloete discuss availability refusal in general and Denial-of-Service and delay of service specifically, pointing out the same problems as Jansen and Karygiannis.

18. (Denial Of Service) Data Flow *Instantiate Agent* Is Potentially Interrupted

An external entity interrupts data flowing across a trust boundary in either direction. In our disaster response setting, disrupting the instantiation of a Mobile Agent can seriously hamper the crisis mitigation procedure. Therefore, we consider this a pertinent threat. In our model, the only entity capable of doing the deed is a compromised DSAP. As such, we consider this a special subclass of Threat 17.

19. (Elevation Of Privilege) Elevation Using Impersonation

The Mobile Agent may be able to impersonate the context of DSAP in order to gain additional privilege. As discussed in Threat 61 it is inherent in the Secure Agent Infrastructure that every DSAP is granted the authority of the Core Secure Agent Infrastructure. Furthermore, the Secure Agent Infrastructure does not really differentiate authorization between a DSAP and a Mobile Agent executing in this DSAP. Therefore it is system inherent that the Mobile Agent can impersonate the DSAP. However, our security assumptions prevent the introduction of compromised agents on a security policy conforming DSAP. A DSAP may be compromised so that it allows modified Mobile Agents to be loaded and executed, and these agents would then operate with the authority of the Core Secure Agent Infrastructure. We consider this threat as pertinent.

This threat is also considered in literature. Jansen and Karygiannis detail that an agent that has access to a platform and its services without having the proper authorization can harm other agents and the platform itself. Borselius states that authorisation and delegation are important issues in Multi-Agent System.

20. (Elevation Of Privilege) The Mobile Agent May be Subject to Elevation of Privilege Using Remote Code Execution

The DSAP may be able to remotely execute code for the Mobile Agent. This threat points at one of the core security issues with Multi-Agent System. The Mobile Agent is executed by the DSAP and the DSAP can change the code, data, state of the Mobile Agent. A compromised DSAP can disrupt the workings of the Secure Agent Infrastructure, therefore we consider this threat pertinent. We have discussed it in detail under Threat 14.

21. (Elevation Of Privilege) Elevation by Changing the Execution Flow in the Mobile Agent

An adversary may pass data into the Mobile Agent in order to change the flow of program execution within the Mobile Agent to the adversary's choosing. We consider this a special variant of Threat 20.

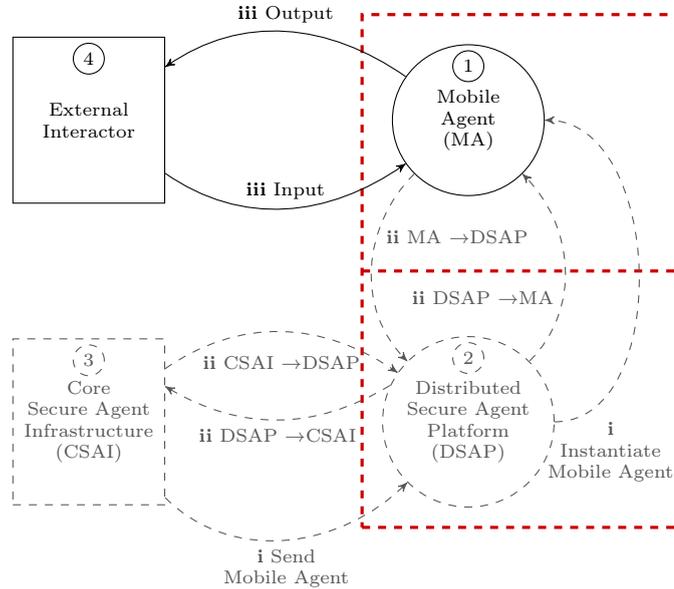


Figure A.4: The part of our Data Flow Diagram model we use to analyze the threats to a Mobile Agent when interacting with an External Interactor. An External Interactor can be a human being, or an external information system.

22. (Elevation Of Privilege) Cross Site Request Forgery

Cross Site Request Forgery is a web browser specific elevation of privilege attack. As we do not use a web browser to instantiate the Mobile Agent we disregard this threat here.

A.6 Agent – External Interactor Communication

This section discusses the threats to a Mobile Agent, when interacting with an External Interactor. Figure A.4 depicts our Data Flow Diagram model of the Secure Agent Infrastructure interacting with an External Interactor through the use of a Mobile Agent executing in a Distributed Secure Agent Platform Outpost (DSAP Outpost). Here we only consider the *Input* and *Output* data flows between the Mobile Agent and the External Interactor. The External Interactor can be a human, or an information system. The Secure Agent Infrastructure uses User Communication Agents for communicating with external human interactors, and Information Delivery Agents for retrieving information from external information systems. See Section 2.6 for details on these Mobile Agents.

A.6.1 Input

1. (Spoofing) Spoofing the Mobile Agent Process

A Mobile Agent may be spoofed by an adversary and this may lead to information disclosure by the External Interactor. We consider this a pertinent threat. Here we want to distinguish two use cases. First, if the Distributed Secure Agent Platform (DSAP) and consequently the Mobile Agent operate on a mobile device associated with a human user, then we need to ensure that the user only inputs its data into an authentic User Communication Agent. Second, if the Mobile Agent interacts with another information system, then we still need to authenticate the Mobile Agent against the external system, but we can use different credentials, such as cryptographic keys, for authentication. Also, we want to distinguish between two threats. First a truly external entity spoofs the Mobile Agent, and second a compromised DSAP spoofs the Mobile Agent.

A truly external entity spoofing the Mobile Agent is a pertinent threat. A realization of this threat enables eavesdropping on sensitive information, tampering with information, injecting false information, Denial-of-Service and, if the External Interactor is an information system, a number of elevation of privilege attacks. Given our disaster response scenario this can seriously impact crisis mitigation performance.

Concerning a compromised DSAP, our security assumptions only allow the introduction of security policy conforming agents, so we preclude the threat of one Mobile Agent masquerading as another Mobile Agent. Also all our Mobile Agent and, by necessity, also the DSAP have the same authority, the authority of the Core Secure Agent Infrastructure, thus we need to consider a compromised DSAP masquerading as the agent. Here the compromised DSAP can then repudiate ever having received any information from the External Interactor (cf. Threat 4), tamper with this information (cf. Threat 3), eavesdrop the information (cf. Threat 5), and delay or discard the information (cf. Threat 6 and Threat 7).

Jansen and Karygiannis discuss Mobile Agents masquerading as other Mobile Agents and the need for authenticating agents in order to authorise their use of platform services correctly in the agent-to-platform threat category. Borselius acknowledges the need for authenticity in agent to environment communication when discussing the communication property of agents. Furthermore, he also establishes the need for agent authentication and authorisation. None of our sources specifically considers a compromised platform spoofing an Mobile Agent, but all three acknowledge a compromised platform's ability to eavesdrop and manipulate agent communication.

Our literature sources do not consider an external entity spoofing a mobile agent in order to gain unauthorized access to an External Interactor.

As we are now investigating the inner workings of the Secure Agent Infrastructure, specifically how Mobile Agents are used to propagate information, we gain detailed threat intelligence for these Secure Agent Infrastructure specifics. As will become more and more apparent in the discussion of the following threats

this helps us to gain an in-depth understanding of threats that is superior to what can be gleaned from the generic discussions in the Multi-Agent System security literature. This holds even though the Multi-Agent System literature is as comprehensive as it is. Furthermore, given the threat discussions so far, and considering the following discussions, we observe that none of our Multi-Agent System security sources covers all threats we discuss here. We take this as proof that a literature study is no substitute for a detailed threat analysis of the particular system.

2. (Spoofing) Spoofing the External Interactor

The External Interactor may be spoofed by an adversary and this may lead to unauthorized access to Mobile Agent. Again we consider this is pertinent threat as it enables Denial-of-Service (cf. Threat 6 and Threat 7), injection of tampered information (cf. Threat 3), and elevation of privilege attacks (cf. Threat 9, Threat 10, and Threat 11), while implicating the spoofed entity. As with Threat 1, if this threat is mitigated using authentication, then again both the need to authenticate human users and automatic information systems need to be considered.

In addition to an adversarial external entity spoofing an authentic external entity we also consider a compromised DSAP masquerading as the External Interactor. Here the compromised DSAP can then tamper with this information (cf. Threat 3) and withhold information as Denial-of-Service (cf. Threat 6 and Threat 7). For reference a compromised DSAP modifying agent data, state, and code has been discussed in Threat 14.

Jansen and Karygiannis consider unauthorized access to the DSAP in the sense that an agent platform needs to be protected against attacks on the underlying system, but they do not consider an external entity masquerading as an authorized external entity. Borselius establishes the need for authenticity in agent to environment and human communication as a generic property of all agent communications. All three sources acknowledge a compromised platforms ability to manipulate agent communication.

3. (Tampering) Potential Lack of Input Validation for the Mobile Agent

Data flowing across *Input* may be tampered with by an adversary. We consider this a threat that requires mitigation as it is a threat that potentially enables information disclosure (cf. Threat 5) Denial-of-Service (cf. Threat 6 and Threat 7), and elevation of privilege attacks (cf. Threat 9, Threat 10, and Threat 11). We consider two different threat agents here. First a truly External Interactor can tamper with the information before it reaches the Mobile Agent. Second, a compromised DSAP might introduce tampered data via this channel. A compromised DSAP modifying agent data, state, and code has been discussed in Threat 14, but this threat adds the additional perspective of implicating the External Interactor (cf. Threat 2).

Borselius emphasizes the need for integrity in communication to protect against manipulation while the data is in transit. He also points out the need for authenticity, confidentiality and availability when an agent communicates with the environment. However, none of our sources considers the injection of tampered data by an authenticated entity, nor do they specifically consider a compromised agent platform using this an environmental input channel to mask tampering with agent data.

4. (Repudiation) Potential Data Repudiation by the Mobile Agent

The Mobile Agent claims that it did not receive data from a source outside the trust boundary. We consider this a pertinent threat, especially with the disaster response background. The crisis mitigation processes needs to be auditable, in case of claims of damages or injuries caused by wanton negligence.

Again, based on our security assumptions, the Core Secure Agent Infrastructure only introduces certified security policy conforming agents into the system, and we consider the Core Secure Agent Infrastructure’s Mobile Agent dispatch mechanism secure. However, a compromised DSAP can modify a Mobile Agent, or even tamper with local logs, so that it repudiates ever having received the information.

Borselius’ mentions non-repudiation as a desired property for all agent communication. Jansen and Karygiannis mention repudiation only in the context of agent-to-agent communication.

5. (Information Disclosure) Data Flow Sniffing

Data flowing across *Input* may be sniffed by an adversary. We consider this a pertinent threat. In our discussion of this threat we want to differentiate three cases. First the External Interactor is a human and another human is eavesdropping the input. Second, the External Interactor is an information system and another entity is eavesdropping on the connection between the information system and the Mobile Agent. Third, the External Interactor is either a human or an information system and a compromised DSAP eavesdrops the information.

We have already discussed the literature on the threat of a compromised agent platform eavesdropping on the agent data in Threat 16. The problem of an external entity listening in on Mobile Agent communication is mentioned by both Jansen and Karygiannis and Borselius.

This threat actually comprises the three threats enumerated above. We could have made these threats more explicit by modelling both a human and a non-human External Interactor. Furthermore we could have highlighted the role of the platform by routing the communication with the External Interactor through the platform. We chose to model the External Interactor as a single interactor directly interacting with the Mobile Agent to avoid having all significant number of duplicate threats cluttering the analysis. Furthermore, we believe that an attack tree analysis would better resolve individual threats, than actually complicating the model and generated more threats.

6. (Denial Of Service) Potential Process Crash or Stop for the Mobile Agent

The Mobile Agent crashes, halts, stops or runs slowly; in all cases violating an availability metric. In our disaster response setting this can lead to important information, not reaching its destination and we consider suitable mitigation essential. Our security assumptions preclude the Mobile Agent itself violating its own availability constraints, but as documented in Threat 17 a compromised DSAP can realize this threat easily. See Threat 17 for a discussion of this threat in literature.

7. (Denial Of Service) Data Flow *Input* Is Potentially Interrupted

An external agent interrupts data flowing across a trust boundary in either direction. Given that a realization of this threat can hamper crisis mitigation, we consider this threat pertinent. Here we consider three distinct threats. First, if the DSAP is associated with an external information system, then this information system might be compromised, leading to interruptions of the data flow. Second if the DSAP is coupled to a human External Interactor, this human might be interrupted from interacting with the Mobile Agent. Third, the DSAP itself is compromised and interrupts the data flow.

Jansen and Karygiannis consider both Denial-of-Service due to a compromised agent platform in their platform-to-agent category, and external entities disrupting the agent platform, but not a Mobile Agent, in their other-to-agent platform category. Borselius indicates the need for availability in all agent communication and Bierman and Cloete document a compromised agent platform denying or delaying a service.

8. (Elevation Of Privilege) Cross Site Request Forgery

Cross-site request forgery (CSRF or XSRF) is a type of attack in which an adversary forces a user's browser to make a forged request to a vulnerable site by exploiting an existing trust relationship between the browser and the vulnerable web site. The User Communication Agent (see Section 2.6) supports user communication via web technologies by using its own web server. In this case the user inputs data into a form and sends it to the web server that then forwards the data to the User Communication Agent. As the web server is under control of the User Communication Agent Mobile Agent and the User Communication Agent is under control of the DSAP this is a pertinent threat. Even though we assume the User Communication Agent to conform to the security policy, the DSAP might be compromised.

Our literature sources do not specifically consider cross-site request forgery as a threat in the context of Multi-Agent System. Borselius however notes that the environment might also need certain protection from the agents it hosts. Jansen and Karygiannis document that Mobile Agent need to be authorized before they are given access to agent platform service.

Cross-site request forgery is a very specific threat and it is not surprising that it is not represented in literature. We have to admit it is somewhat surprising to us that the Secure Agent Infrastructure actually has a context where this threat can materialize.

9. (Elevation Of Privilege) Elevation Using Impersonation

The Mobile Agent may be able to impersonate the context of External Interactor in order to gain additional privilege. As has been pointed out earlier we assume that our Mobile Agents will conform to the security policy. If however the DSAP is compromised, and the external entity discloses information that allows impersonation, then this is a concrete threat. Therefore, we qualify this threat as pertinent.

Especially for this threat, threat mitigation and threat risk evaluation have to be done in the context of concrete use of a particular DSAP Outpost. Specifically, the potential for damage depends on what information users, or external informations systems, disclose. Relevant realizations of this threat could pertain to users disclosing access credentials to external systems, or external information systems disclosing privacy sensitive information, such as social security numbers, allowing the impersonation of human beings.

For this threat and also Threat 10 and Threat 11 we consider a compromised DSAP as a potential threat agent. A compromised DSAP can use these elevation of privilege attacks to, not only to access sensitive information, but also imply the Mobile Agent in having a hand in these attacks.

We have discussed the literature about the agent platform eavesdropping on the Mobile Agent in the context of impersonation in Threat 61. None of our literary sources discuss a Mobile Agent impersonating an external entity. However, Jansen and Karygiannis and Borselius point out that agents act on behalf of a person, organization, or other agents. Borselius specifically identifies the need for a mechanism to transfer rights, but does not discuss the security issues inherent in such an approach.

10. (Elevation Of Privilege) The Mobile Agent May be Subject to Elevation of Privilege Using Remote Code Execution

The External Interactor may be able to remotely execute code for the Mobile Agent. Our security assumptions preclude this threat. The threat of a compromised DSAP executing code in the context of the Mobile Agent has been discussed in Threat 14 and Threat 20.

11. (Elevation Of Privilege) Elevation by Changing the Execution Flow in the Mobile Agent

An adversary may pass data into the Mobile Agent in order to change the flow of program execution within the Mobile Agent to the adversary's choosing. We treat this as a subclass of Threat 10.

A.6.2 Output

Here the Mobile Agent sends information to a source external to the Secure Agent Infrastructure. This External Interactor can be a human, or an information system. The Secure Agent Infrastructure uses User Communication Agents for communicating with external human interactors, and Information Delivery Agents for retrieving information from information systems. See Section 2.6 for details on these Mobile Agents.

Threat 65, Threat 66, and Threat 67 are special in the sense that we manually added them in. STRIDE-per-interaction does not consider eavesdropping and tampering on data flows going out to an External Interactor. Also STRIDE-per-interaction does not consider impersonating of the sending entity, the Mobile Agent in our case. However due to the nature of a Mobile Agent System, we need to consider a compromised DSAP realizing these threats on an Mobile Agent sending data to an External Interactor.

23. (Spoofing) Spoofing of the External Interactor External Destination Entity

The External Interactor may be spoofed by an adversary and this may lead to data being sent to the adversary's target instead of the External Interactor. We consider this a pertinent threat that mandates mitigation. Again we want to distinguish between a human generic interactor and an external information system, as we expect these to require different mitigation mechanisms. Furthermore, we want to distinguish two sub-threats. First, a truly external entity spoofs the External Interactor and second a compromised DSAP spoofs the External Interactor. In the first case, we are dealing with information disclosure. The second case is more complicated. By the spoofing the External Interactor, the DSAP leads the agent to believe it has delivered its output to the External Interactor, whereas in reality, the DSAP received it. We consider the DSAP eavesdropping Mobile Agent information in Threat 67, the DSAP tampering with Mobile Agent data in Threat 66, and the DSAP performing a Denial-of-Service attack versus the agent in Threat 68. In the disaster response setting, the DSAP not delivering information, or delivering tampered information has serious implications for accountability in case of injury or damages due to misinformation or lack of information.

Borselius points out that authentication is an important property in all agent communication. Jansen and Karygiannis list masquerading agents and agent platforms as threat to Mobile Agent in their agent-to-agent and platform-to-agent threat categories, but they do not consider an external entity deceiving a Mobile Agent. All three sources acknowledge a compromised platforms ability to manipulate agent communication.

24. (Repudiation) The External Interactor Potentially Denies Receiving Data

The External Interactor claims that it did not receive data from a process on the other side of the trust boundary. As with Threat 4 auditability is an important property for crisis management to, not to put to fine a point on it, hold people accountable for their actions, if things go south. We therefore consider this a pertinent threat that mandates mitigation.

Only Borselius mentions non-repudiation as a desirable property in all agent communication. Jansen and Karygiannis mention repudiation only in the context of agent-to-agent communication.

25. (Denial Of Service) Data Flow *Output* Is Potentially Interrupted

An external agent interrupts data flowing across a trust boundary in either direction. Again we consider this threat pertinent, as this threat can impact situational awareness, or delivery of commands, which can seriously disrupt crisis mitigation. Here we again distinguish between an external information system and a human interactor. We have discussed the case of a compromised DSAP interrupting a Mobile Agent in Threat 18.

Again, only Borselius documents the need for communication availability as an important property in all agent communication. Jansen and Karygiannis do consider the importance of availability in the context of agents communicating with other agents and agent platforms, but not when agents communicate with truly external entities.

65. (Spoofing) Compromised Distributed Secure Agent Platform Spoofs the Mobile Agent

A compromised DSAP can spoof the Mobile Agent and this may lead to the injection of tampered information (cf. Threat 67), or a violation of availability constraints (cf. Threat 68).

We have discussed the literature on the threat of a compromised agent platform impersonating a Mobile Agent in Threat 13 and Threat 1.

66. (Tampering) Compromised DSAP Tampers With Mobile Agent Output

Data flowing across *Output* may be tampered with by an adversary. We consider this this threat pertinent in our disaster response setting. In this setting, the DSAP delivering tampered information has serious implications for accountability in case of injury or damages due to misinformation or lack of information.

We have discussed the literature on the threat of a compromised agent platform tampering with agent data in Threat 14.

67. (Information Disclosure) Compromised Distributed Secure Agent Platform Sniffs Data Flow

Data flowing across *Output* may be sniffed by a compromised DSAP. We consider this threat pertinent, as the Secure Agent Infrastructure deals with sensitive information during disaster response.

We have discussed the literature on the threat of a compromised agent platform eavesdropping on the agent data in Threat 16.

68. (Denial Of Service) Compromised Distributed Secure Agent Platform Enacts Process Crash or Stop for the Mobile Agent

A compromised DSAP executes the Mobile Agent in a way so that it crashes, halts, stops, or runs slowly; in all cases violating an availability constraint. In our disaster response setting this can lead to important information not reaching its destination in a timely fashion or at all. Therefore we consider this threat pertinent.

We have discussed the literature on the threat of a compromised agent platform interfering with the correct execution of a Mobile Agent to inflict a Denial-of-Service attack in Threat 17. Concerning disrupting the communication with external entities, only Borselius documents the need for communication availability as an important property in all agent communication. Jansen and Karygiannis do consider the importance of availability in the context of agents communicating with other agents and agent platforms, but not when agents communicate with truly external entities. All three sources acknowledge a compromised platforms ability to manipulate agent communication.

A.7 Agent – Core Secure Agent Infrastructure Communication

This section discusses the threats to a Mobile Agent when communicating with the Core Secure Agent Infrastructure. Figure A.4 illustrates our Data Flow Diagram model of a Mobile Agent communicating with the Core Secure Agent Infrastructure through the use of the Distributed Secure Agent Platform (DSAP)'s communication facilities. Here we only consider the $MA \rightarrow DSAP$, the $DSAP \rightarrow CSAI$, the $CSAI \rightarrow DSAP$, and the $DSAP \rightarrow MA$ data flows between the Mobile Agent and the Core Secure Agent Infrastructure. See Section 2.6 for details on the Secure Agent Infrastructure Mobile Agent communication concept.

A.7.1 MA \rightarrow DSAP

26. (Spoofing) Spoofing the Mobile Agent Process

The Mobile Agent may be spoofed by an adversary and this may lead to unauthorized access to the DSAP. As our security assumptions only allow certified

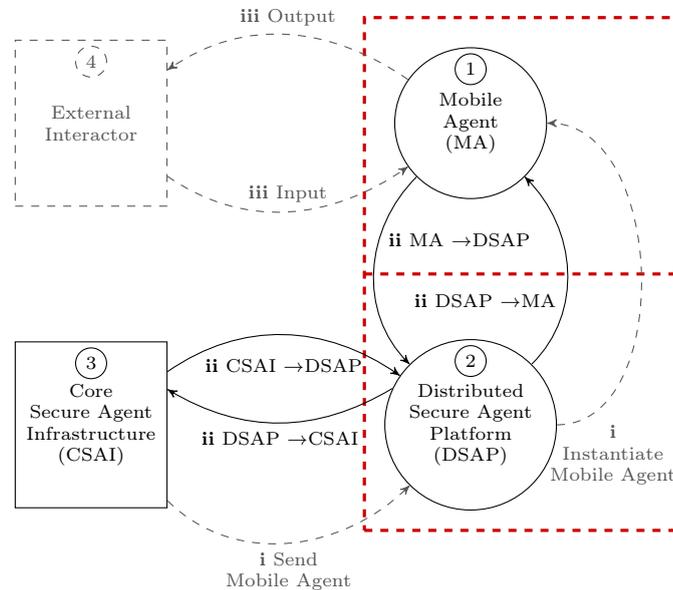


Figure A.5: The part of our Data Flow Diagram model we use to analyze the threats to a Mobile Agent when communicating with the Core Secure Agent Infrastructure using the DSAP’s communication facilities.

agents that adhere to the security policy that leaves a compromised DSAP spoofing the Mobile Agent. A compromised DSAP can use this capability to fake the origin of information for example to hide her tracks, or implicate an External Interactor. As such we consider this threat pertinent.

Jansen and Karygiannis document the threat of agents masquerading as other agents when communicating both with other agents and the agent platform in their agent-to-platform and agent-to-agent threat categories. They also consider an agent platform helping an agent in masquerading its identity under the other-to-agent category. However, they do not discuss an agent platform impersonating an agent. However, they do consider an agent platform compromising a Mobile Agent and sending the compromised agent to other remote platforms. Borselius discusses the need for identification and authentication of agents in agent platforms, but not a compromised platform masquerading as an agent. However, Borselius also points out that an agent platform can modify a Mobile Agent’s code, data, and state. Finally, Bierman and Cloete consider a host masquerading as another host to lure in Mobile Agents as authentication risk, in their subcategory of masquerading. They also document an agent platform’s ability to compromise Mobile Agents, but again they do not specifically mention an agent platform impersonating an agent.

27. (Spoofing) Spoofing the Distributed Secure Agent Platform Process

The DSAP may be spoofed by an adversary and this may lead to information disclosure by the Mobile Agent. In our model, the only entity that could spoof the DSAP versus the Mobile Agent is a compromised DSAP. There is the potential problem of an adversary stealing a Mobile Agent and running it in his own execution environment that is different from any authentic DSAP. However, given our security assumptions the only way to obtain a Mobile Agent is through compromising an authentic DSAP. As the Mobile Agent itself might contain valuable Intellectual Property, or transport sensitive information, being able to execute the agent in an isolated environment that is observable and controllable by an adversary might help said adversary to extract information from the agent or to reverse engineer it. Therefore we believe this threat worth considering.

Jansen and Karygiannis point out that the code, data and state of a visiting agent is transparent to the agent platform and that an agent platform can compromise the agent before sending it on. Borselius similarly indicates that the agent platform can observe and manipulate code, data and flow control of Mobile Agent. Bierman and Cloete have their own reverse engineering threat subclass where they consider an adversary that observes an agent in order to facilitate reverse engineering.

28. (Tampering) Potential Lack of Input Validation for Distributed Secure Agent Platform

Data flowing across $MA \rightarrow DSAP$ may be tampered with by an adversary. In our model the only entity that could perform this tampering is a compromised DSAP (cf. Threat 14). This however is a very pertinent threat given our disaster response scenario. We have already discussed the threat of a compromised DSAP tampering with the data of a Mobile Agent in Threat 14. However this threat offers a new Secure Agent Infrastructure specific viewpoint on what a compromised DSAP can do, therefore we let it stand.

All our three primary literary sources indicate this threat. See Threat 14 for details.

29. (Repudiation) Potential Data Repudiation by the Distributed Secure Agent Platform

The DSAP claims that it did not receive data from a source outside the trust boundary. As we discussed in Threat 4, Threat 23, and Threat 24, repudiation is a serious threat in a disaster response scenario. We therefore consider this threat pertinent. A compromised DSAP can easily implement this threat, while at the same time complicating mitigation.

Jansen and Karygiannis point out that an agent platform can tamper with all agent communication, but they do not specifically discuss repudiation in this context. Borselius describes non-repudiation as a desirable property in all

agent communication. Similarly, Bierman and Cloete do not discuss a platform repudiating having received agent messages. All three sources acknowledge a compromised platform's ability to manipulate agent communication.

30. (Information Disclosure) Data Flow Sniffing

Data flowing across the $MA \rightarrow DSAP$ data flow may be sniffed by an adversary. The only realization of this threat permissible in our model is through a compromised DSAP. We consider this threat pertinent. The Secure Agent Infrastructure deals with sensitive information, and disclosing sensitive information to unauthorized entities is not an option.

All our three primary sources document the threat of an agent platform eavesdropping information from their agents. See Threat 16 for a discussion.

31. (Denial Of Service) Potential Process Crash or Stop for the Distributed Secure Agent Platform

The DSAP crashes, halts, stops or runs slowly; in all cases violating an availability metric. The network of DSAPs forms the execution fabric for all Mobile Agent and thus they are an essential component of the Secure Agent Infrastructure, without which the Secure Agent Infrastructure will not be able to perform its disaster response support role. This is a pertinent threat. In this case whatever information the Mobile Agent has gathered will not be forwarded to the Core Secure Agent Infrastructure which can have a negative impact on situational awareness and hamper dispatching commands. In our model, a possible realization of this threat is again a compromised DSAP.

Jansen and Karygiannis document the threat of agents performing Denial-of-Service attacks on an agent platform (agent-to-platform threat category), as well as external entities attacking the agent platform (other-to-agent platform). Borselius points out the need for availability when discussing agent communication.

32. (Denial Of Service) Data Flow $MA \rightarrow DSAP$ Is Potentially Interrupted

An external agent interrupts data flowing across a trust boundary in either direction. In our model the only entity able to interrupt this flow is a compromised DSAP. This is a pertinent threat as a compromised platform could suppress specific information or commands to change the crisis mitigation process.

See Threat 7 for a discussion of the literature.

33. (Elevation Of Privilege) Elevation Using Impersonation

The DSAP may be able to impersonate the context of the Mobile Agent in order to gain additional privilege. This is a pertinent threat. We have already discussed this threat at length in Threat 61 and in Threat 20.

34. (Elevation Of Privilege) The Distributed Secure Agent Platform May be Subject to Elevation of Privilege Using Remote Code Execution

The Mobile Agent may be able to remotely execute code for the DSAP. We assume that all Mobile Agents adhere to the security policy and disregard this threat.

Interestingly, Jansen and Karygiannis do not explicitly consider this a threat in their agent-to-platform threat category. They do consider agents masquerading as other agents and agents performing Denial-of-Service attack against an agent platform. The closest security violation they consider is unauthorized access by Mobile Agents to platform services and resources through lack of proper access control. Borselius points out that hosts need to be protected from agents and from other parties that communicate with the platform, and he claims that the problems associated with the protection of hosts from malicious code are quite well understood.

35. (Elevation Of Privilege) Elevation by Changing the Execution Flow in Distributed Secure Agent Platform

An adversary may pass data into the DSAP in order to change the flow of program execution within the DSAP to the adversary's choosing. Again we assume that all Mobile Agents adhere to the security policy and disregard this threat. For a discussion of the literature see Threat 34.

36. (Elevation Of Privilege) Cross Site Request Forgery

Cross Site Request Forgery is a web browser specific elevation of privilege attack. As we do not use a web browser to send data from a Mobile Agent to the DSAP we disregard this threat.

A.7.2 DSAP \rightarrow CSAI**37. (Spoofing) Spoofing of the Core Secure Agent Infrastructure External Destination Entity**

The Core Secure Agent Infrastructure may be spoofed by an adversary and this may lead to data being sent to the adversary's target instead of the Core Secure Agent Infrastructure. We assume the use of a secure communication protocol between the DSAP and the Secure Agent Infrastructure that provides mutual authentication. Therefore, we disregard this threat.

Borselius points out that authentication is an important property in all agent communication. Jansen and Karygiannis list masquerading agents and agent platforms as threat to Mobile Agent in their agent-to-agent and platform-to-agent threat categories, but they do not consider an external entity deceiving a Mobile Agent. All three sources acknowledge a compromised platforms ability to manipulate agent communication.

38. (Repudiation) The Core Secure Agent Infrastructure Potentially Denies Receiving Data

The Core Secure Agent Infrastructure claims that it did not receive data from a process on the other side of the trust boundary. We assume that the Core Secure Agent Infrastructure adheres to the security policy. Therefore we disregard this threat.

Only Borselius points out the need for non-repudiation in all agent communication. Jansen and Karygiannis mention repudiation only in the context of agent-to-agent communication.

39. (Denial Of Service) Data Flow *DSAP* → *CSAI* Is Potentially Interrupted

An external agent interrupts data flowing across a trust boundary in either direction. This is a pertinent threat, as an adversary can seriously hamper crisis mitigation by selectively suppressing the data flow (cf. Threat 32). As we assume a secure communication protocol the interrupted will be detected by the sending DSAP.

Again, only Borselius documents the need for communication availability as an important property in all agent communication. Jansen and Karygiannis do consider the importance of availability in the context of agents communicating with other agents and agent platforms, but not when agents communicate with truly external entities.

A.7.3 CSAI → DSAP**40. (Repudiation) Potential Data Repudiation by Distributed Secure Agent Platform**

DSAP claims that it did not receive data from a source outside the trust boundary. We consider this a pertinent threat, as a compromised DSAP can easily realize this threat. As discussed before (cf. Threat 29, Threat 24, Threat 23, Threat 38 and Threat 4) auditability is a very important in a disaster response setting. For example, repudiating having received certain messages can implicate other entities, or help hide the tracks of an adversary.

Again Borselius' mention of non-repudiation as a desired property for all agent communication is the only pertinent occurrence of this threat in literature. Jansen and Karygiannis mention repudiation only in the context of agent-to-agent communication.

41. (Denial Of Service) Potential Process Crash or Stop for the Distributed Secure Agent Platform

The DSAP crashes, halts, stops or runs slowly; in all cases violating an availability metric. We consider this a pertinent threat. Aside from under specifying

the equipment hosting the DSAP or severe misconfiguration we consider a compromised DSAP as a primary realization of this threat. If the DSAP violates availability constraints this can prevent or delay delivery of important information, such as information pertaining to situation awareness and commands. For reference, we have previously discussed a DSAP violating availability constraints in Threat 59 and Threat 17.

Jansen and Karygiannis primarily discuss how a malicious agent platform can interfere with Mobile Agent, including delaying and denying agent execution or agent service requests. They also mention that the agent platform can tamper with all agent communication. Borselius establishes the need for availability for all agent communications, but does not single out communication with agent platforms that violate availability constraints. Finally, Bierman and Cloete discuss denial of service, delay of service and transmission refusal in the context of an agent platform interfering with a Mobile Agent. They do not consider external parties communicating with an agent platform.

42. (Denial Of Service) Data Flow *CSAI* → *DSAP* Is Potentially Interrupted

An external agent interrupts data flowing across a trust boundary in either direction. This is a pertinent threat, as a adversary can seriously hamper crisis mitigation by selectively suppressing the data flow (cf. Threat 60, Threat 32 and Threat 39). As we assume a secure communication protocol the interrupted will be detected by the sending Core Secure Agent Infrastructure. Again we consider a compromised DSAP as one potential realization of this threat.

See, for example, Threat 25, Threat 32, or Threat 39 for a discussion of pertinent literature.

43. (Elevation Of Privilege) Elevation Using Impersonation

The DSAP may be able to impersonate the context of Secure Agent Infrastructure in order to gain additional privilege. We consider this a pertinent threat that we have discussed in great detail in Threat 61.

44. (Elevation Of Privilege) The Distributed Secure Agent Platform May be Subject to Elevation of Privilege Using Remote Code Execution

The Core Secure Agent Infrastructure may be able to remotely execute code for DSAP. We assume the Core Secure Agent Infrastructure to adhere to the security policy, and due to our use of a secure communication protocol we also preclude other entities injecting code. If the Core Secure Agent Infrastructure wants to remotely execute code on the DSAP it just needs to send the appropriate agent. Therefore we disregard this threat.

45. (Elevation Of Privilege) Elevation by Changing the Execution Flow in the Distributed Secure Agent Platform

An adversary may pass data into the DSAP in order to change the flow of program execution within the DSAP to the adversary's choosing. In this case, we consider this threat to be a special variant of Threat 44 and disregard it.

46. (Elevation Of Privilege) Cross Site Request Forgery

Cross Site Request Forgery is a web browser specific elevation of privilege attack. As we do not use a web browser send data from the Core Secure Agent Infrastructure to the DSAP we disregard this threat.

A.7.4 DSAP → MA**47. (Spoofing) Spoofing the Distributed Secure Agent Platform Process**

The DSAP may be spoofed by an adversary and this may lead to unauthorized access to Mobile Agent. In our model, the only entity able to spoof an authentic DSAP is a compromised DSAP. However there is a seconds aspect to consider. Here the DSAP relays information from the Core Secure Agent Infrastructure. Therefore, the DSAP can impersonate the Core Secure Agent Infrastructure versus the Mobile Agent. We consider this threat pertinent. This threat has been discussed in the context of a compromised DSAP impersonating an External Interactor in Threat 2. As in Threat 2 a compromised DSAP that realizes this threat can inject tampered information into the Mobile Agent (cf. Threat 49) and withhold information as Denial-of-Service. See Threat 2 for a discussion on the literature.

48. (Spoofing) Spoofing the Mobile Agent Process

The Mobile Agent may be spoofed by an adversary and this may lead to information disclosure by the DSAP. In our model, the only entity able to spoof an authentic Mobile Agent is a compromised DSAP. This is due to our assumption that only certified and security policy adhering Mobile Agents can enter the Secure Agent Infrastructure. A compromised DSAP might use this capability to claim having delivered information to the intended receiver, when in fact it has not. See also Threat 1.

Jansen and Karygiannis discuss Mobile Agents masquerading as other Mobile Agents and the need for authenticating agents in order to authorise their use of platform services correctly in the agent-to-platform threat category. Borselius acknowledges the need for authenticity in agent to environment communication when discussing the communication property of agents. Furthermore, he also establishes the need for agent authentication and authorisation. None of our sources specifically considers a compromised platform spoofing an Mobile Agent,

but all three acknowledge a compromised platform's ability to eavesdrop and manipulate agent communication.

49. (Tampering) Potential Lack of Input Validation for the Mobile Agent

Data flowing across the *DSAP* \rightarrow *MA* data flow may be tampered with by an adversary. In our model, the only entity able to tamper with this data at this juncture is a compromised DSAP. This threat has been detailed in Threat 14 and Threat 3.

50. (Repudiation) Potential Data Repudiation by the Mobile Agent

The Mobile Agent claims that it did not receive data from a source outside the trust boundary. As argued in Threat 4, based on our assumption a Mobile Agent cannot repudiate any messages, unless it is tampered with by a compromised DSAP. We consider this a threat as any kind of non-repudiation is an important security property in a disaster response setting (cf. Threat 40, Threat 29, Threat 24, Threat 23, and Threat 4).

51. (Information Disclosure) Data Flow Sniffing

Data flowing across *DSAP* \rightarrow *MA* may be sniffed by an adversary. We consider this a pertinent threat, because, as we have pointed out in Threat 30, the Secure Agent Infrastructure is operating with sensitive information. The only entity capable of realizing this threat in our model is again a compromised DSAP. We have already discussed the details and literature research for this threat in Threat 16 and Threat 5.

52. (Denial Of Service) Potential Process Crash or Stop for the Mobile Agent

The Mobile Agent crashes, halts, stops or runs slowly; in all cases violating an availability metric. A User Communication Agent or a Information Delivery Agent violating their availability constraints can lead to important information not reaching an External Interactor or the Core Secure Agent Infrastructure in time. Therefore we consider this threat pertinent. As detailed in Threat 6 our security assumptions preclude the Mobile Agent from violating its own availability constraints. However as described in Threat 6, Threat 59, and Threat 17 a compromised DSAP can realize this threat. A compromised DSAP performing a Denial-of-Service attack has been discussed in Threat 59 and Threat 17.

53. (Denial Of Service) Data Flow *DSAP* \rightarrow *MA* Is Potentially Interrupted

An external agent interrupts data flowing across a trust boundary in either direction. In our model, the only external agent that can realize the threat is

a compromised DSAP. This is a pertinent threat, as a compromised DSAP can selectively suppress data flows to influence the crisis mitigation process. See Threat 7, Threat 32, Threat 33, and Threat 39 for a discussion.

54. (Elevation Of Privilege) Elevation Using Impersonation

The Mobile Agent may be able to impersonate the context of the DSAP in order to gain additional privilege. As we established in Threat 19 and Threat 61 every Mobile Agent and the DSAP operate with the authority of the Core Secure Agent Infrastructure. Furthermore, the only entity in our model capable of realizing this threat is a compromised DSAP. The threat of a compromised DSAP impersonating a Mobile Agent has been discussed in Threat 20. This specific instance of this impersonation threat adds the additional dimension of implicating the Mobile Agent, as the compromised DSAP can claim delivery of the information send by the Core Secure Agent Infrastructure, when in fact, it never delivered the information to the agent.

55. (Elevation Of Privilege) The Mobile Agent May be Subject to Elevation of Privilege Using Remote Code Execution

The DSAP may be able to remotely execute code for the Mobile Agent. This is pertinent threat. We have discussed the details in Threat 14 and Threat 20.

56. (Elevation Of Privilege) Elevation by Changing the Execution Flow in the Mobile Agent

An adversary may pass data into the Mobile Agent in order to change the flow of program execution within Mobile Agent to the adversary's choosing. As the along this flow is forwarded from the Core Secure Agent Infrastructure and we assume that the Core Secure Agent Infrastructure adheres to the security policy, the only adversary in our model that can realize this threat is a compromised DSAP. This is a pertinent threat that we see as a special variant of Threat 55.

57. (Elevation Of Privilege) Cross Site Request Forgery

Cross Site Request Forgery is a web browser specific elevation of privilege attack. As we do not use a web browser to send data from the DSAP to the Mobile Agent we disregard this threat.

B

Security Evaluation

B.1 Introduction

One of the contributions of this thesis is the evaluation of the security of our security solution for Distributed Secure Agent Platform Outposts (DSAP Outposts). We have introduced our security solution in Chapter 5 and it comprises the Trusted Docking Station (TDS) and the Secure Docking Module (SDM). For this evaluation we have considered each of the threats against a DSAP Outpost that we have identified in Chapter 4. For each threat we evaluated if the threat is mitigated and, if it is mitigated, by what mechanism. We detail our evaluation here and we discuss the results of our evaluation in Section 5.9.

To break up the long threat evaluation tables we have split them into the same three categories we have used in Appendix A. These categories are *agent migration*, *agent - External Interactor communication*, and *agent - Core Secure Agent Infrastructure communication*. These categories, and also the threats we discuss here, are based on the model depicted in Figure A.1.

B.2 Agent Migration

Here we discuss threats to agent migration and instantiation. Of the eleven threats in this category, ten are mitigated by our security solution by ensuring the load-time integrity of the Distributed Secure Agent Platform Outpost (DSAP Outpost). The remaining one threat, Threat 03, is mitigated by using the Mobile Agent paradigm.

Table B.1: This table discusses how our security solution comprising the Trusted Docking Station and the Secure Docking Module mitigates the threats to agents migrating from the Core Secure Agent Infrastructure to a Distributed Secure Agent Platform Outpost

ID	T	Title
<i>Send Mobile Agent</i>		
01	R	The Distributed Secure Agent Platform Repudiates Receipt of a Mobile Agent
		By ensuring the load-time integrity of the Trusted Docking Station (TDS), only a policy conforming Distributed Secure Agent Platform (DSAP) will be loaded. We assume that the security policy forbids repudiation of receiving a Mobile Agent. Thus load-time integrity mitigates this threat.
02	D	Potential Process Crash or Stop for Distributed Secure Agent Platform
		Our security solution does not increase availability in the face of a hardware fault. In fact, by adding the Secure Docking Module (SDM) component we introduce a new hardware component that can fail. However, by enforcing load-time integrity, we can enforce that only DSAP implementations that do not threaten their own availability and are devoid of known security vulnerabilities are loaded. Also, in the face of compromised load-time integrity, we have planned for a fallback mechanism that allows the user to boot and use the platform despite compromised integrity. Overall, we consider this threat mitigated.
03	D	The Data Flow <i>Send Mobile Agent</i> Is Potentially Interrupted
		Using Mobile Agents provides mitigation of this threat against an adversary that can only opportunistically interrupt the <i>send Mobile Agent</i> data flow. If the connection is interrupted, the Mobile Agent can be resend at a later point in time. Our security solution does not provide any protection against an adversary that permanently disconnects the DSAP Outpost. However, as our adversarial model includes only opportunistic adversaries, we consider this threat mitigated.

Table B.1 – continued from previous page

ID	T	Title
04	E	Compromised Distributed Secure Agent Platform Uses Received Information to Impersonate Core Secure Agent Infrastructure
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost. Load-time integrity ensures that only security policy conforming DSAP Outposts are loaded. We assume that the security policy forbids this impersonation threat. A run-time attack might still temporarily realize this threat (cf. Section 5.9.2).
05	T	A Compromised Distributed Secure Agent Platform Implicates the Core Secure Agent Infrastructure to Have Sent Data to Compromise the Distributed Secure Agent Platform
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
<i>Instantiate Agent</i>		
06	S	Compromised Distributed Secure Agent Platform Gains Full Access to a Mobile Agent
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
07	T	Compromised Distributed Secure Agent Platform Tamper with Mobile Agent
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
08	I	Compromised Distributed Secure Agent Platform Eavesdrops (on) Mobile Agent
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
09	D	Compromised Distributed Secure Agent Platform Enacts Process Crash or Stop for the Mobile Agent
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
10	D	Compromised Distributed Secure Agent Platform Interrupts Data Flow <i>Instantiate Agent</i>
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.

Table B.1 – continued from previous page

ID	T	Title
11	E	Compromised Distributed Secure Agent Platform Impersonates Mobile Agent
This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.		

B.3 Agent - External Interactor Communication

In this section we discuss threats to the communication between a Mobile Agent and an External Interactor. Of the 22 threats in this category 14 are mitigated by our security solution enforcing load-time integrity of the Distributed Secure Agent Platform Outpost (DSAP Outpost). These 14 threats are: 12, 14, 17, 18, 20, 21, 23, 24, 25, 26, 28, 31, 32, and 33. Four of the threats (13, 15, 19, 27) can be mitigated using the credential protection offered by our security solution in conjunction with a secure communication channel to the External Interactor (see Section 5.4). Two of the aforementioned threats (15, and 27) are specifically mitigated by the Secure Docking Module (SDM) establishing the presence and authenticity of a DSAP Outpost user. Two threats pertaining specifically to an external adversary interrupting communication flows (22, and 30) are mitigated by using the Mobile Agent paradigm. This mitigation holds under the assumption that the interruptions are not permanent. Threat 16 is mitigated by auditing the Mobile Agents and the Secure Agent Infrastructure only introducing audited agents into the system (see Section 2.6.7).

Three threats are out of scope of our solution. These threats are Threat 19, where a human External Interactor is eavesdropped upon by another human, Threat 22 where a human is interrupted from using the DSAP Outpost because of an external event, and Threat 29 where an External Interactor repudiates having received information or a command. The final threat can be mitigated using auditing, and to facilitate this our security solution provides a cryptographically secured storage and a secure communication channel to the Secure Agent Infrastructure.

Table B.2: This table contains a description if and how our DSAP Outpost security solution consisting of the Trusted Docking Station and the Secure Docking Module mitigates threats to a Mobile Agent communicating with an External Interactor

ID	T	Title
<i>Input</i>		

Table B.2 – continued from previous page

ID	T	Title
12	S	Compromised Distributed Secure Agent Platform Spoofs the Mobile Agent Process
		<p>This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost. We want to use this threat to point out the role of the SDM when the External Interactor is a human user. When the user boots the Trusted Docking Station (TDS) on her device, the TDS needs to obtain the credentials it uses to communicate with the Secure Agent Infrastructure. As in this case these credentials are bound to the user, the credentials are stored on the SDM. When the TDS requests these credentials using the SDM's resource access protocol, the SDM first establishes if the TDS has a software configuration that enforces the Secure Agent Infrastructure security policy. It then demonstrates this to the user by releasing a shared secret to the TDS that the TDS displays. Thus the user now knows that the TDS is in a policy conforming software configuration. If the Distributed Secure Agent Platform (DSAP) is compromised, the user will not be presented with this secret and can react accordingly.</p>
13	S	External Entity Spoofs Mobile Agent Process
		<p>This threat is mitigated by using strong credentials for authentication of the Mobile Agents, such as cryptographic keys. By binding these keys to the load-time integrity of the TDS either through sealing or through using the SDM a security policy that prevents key disclosure can be enforced. Thus, this threat is mitigated, as an adversarial external entity cannot gain access to these keys. However, an adversary might still be able to successfully perform a run-time attack against the DSAP Outpost that has access credentials to gain access to the key material.</p>
14	S	Compromised Distributed Secure Agent Platform Spoofs the External Interactor
		<p>This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.</p>

Table B.2 – continued from previous page

ID	T	Title
----	---	-------

15	S	External Entity Spoofs the External Interactor
----	---	---

Here we want to distinguish two cases. If the External Interactor is an information system, then this threat can be mitigated using strong credentials, such as cryptographic keys, for authentication of the External Interactor in conjunction with a secure communication channel (see Section 5.4). For example, by using TLS with client certificate authentication to connect the DSAP Outpost with the External Interactor the authenticity of both end-points can be corroborated. To mitigate this threat the operator of the External Interactor has to ensure the security of the External Interactor's private authentication key. On the TDS side, load-time integrity can enforce using the correct and authentic public verification key for establishing the External Interactor's identity.

If the spoofed External Interactor is a human than the SDM ensures the authenticity and the presence of the human user. First, the user has to have the SDM, which is a prove of authenticity by something a user possesses. Furthermore, the user has to proof his presence by presenting a shared secret to the SDM. See Section 5.7 for details.

16	T	Potential Lack of Input Validation for the Mobile Agent
----	---	--

The Secure Agent Infrastructure only introduces audited and authorized agents into the system (cf. Section 2.6.7). Before an agent is cleared for participation in the Secure Agent Infrastructure, a group of independent auditors validates the functionality of the agent and its adherence to the Secure Agent Infrastructure security policy. If the auditors authorize the agent, it is then digitally signed by the auditors. Signed agents are stored with their signature in the Agent Repository. Before an agent is instantiated the Secure Agent Infrastructure verifies the signature on the agent code. The auditing and signing process is out of scope of this work, but if it is secure this threat is mitigated under the assumption that the Secure Agent Infrastructure security policy requires thorough input validation in all Mobile Agents.

17	T	Compromised Distributed Secure Agent Platform Tamperers With Mobile Agent Input
----	---	--

This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.

Table B.2 – continued from previous page

ID	T	Title
18	R	Compromised Distributed Secure Agent Platform Compromises Mobile Agent to Repudiate Data
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
19	I	External Entity Sniffs Data Flow
		Here we distinguish two cases. First, the external entity is human and another human eavesdrops on the information input by the first human. This particular threat is out of scope. Second, the external entity is an information system and an adversary eavesdrops the data on the connection between the DSAP Outpost and the information system. To mitigate this threat our TDS/SDM security solution provides credential storage that is bound to the DSAP Outpost software configuration. So if the external information system supports a cryptographically secured communication channel, such as TLS, our security solution can mitigate this threat.
20	I	Compromised Distributed Secure Agent Platform Sniffs Data Flow
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
21	D	Compromised Distributed Secure Agent Platform Enacts Process Crash or Stop for the Mobile Agent
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
22	D	External Entity Interrupts Data Flow <i>Input</i>
		Here we distinguish two particular realizations of this threat. First, if the External Interactor providing input is a human and the human is interrupted by an external event, then we consider this out of scope. Second, if the External Interactor is an information system and an external entity interrupts the data flow, then similar to Threat 03, the use of Mobile Agents mitigates temporary connection interruptions.
23	D	Compromised Distributed Secure Agent Platform Interrupts Data Flow <i>Input</i>
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.

Table B.2 – continued from previous page

ID	T	Title
24	E	Cross Site Request Forgery
		As the only entity in our model that can perform this cross site request forgery is a compromised DSAP, this threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
25	E	Compromised Distributed Secure Agent Platform Uses Information Received By Mobile Agent to Impersonate External Interactor
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
<i>Output</i>		
26	S	Compromised Distributed Secure Agent Platform Spoofs the External Interactor to Implicate External Interactor
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
27	S	External Entity Spoofs the External Interactor To Gain Unauthorized Access
		For this threat we distinguish two cases. First, if the External Interactor is an information system, then a secure communication channel in conjunction with sealing the authentication credentials to a TDS software configuration that is security policy conforming mitigates this threat.
		For the second case, when the External Interactor is a human user, the SDM mitigates this threat by ensuring the human user's presence and authenticity (see Threat 15 and also Section 5.7).
28	S	Compromised Distributed Secure Agent Platform Spoofs the Mobile Agent
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
29	R	External Interactor Potentially Denies Receiving Data
		This threat is out of scope of our security solution. However, the TDS provides facilities to create a tamper proof log, such as a cryptographically protected storage and a secure communication connection to the Core Secure Agent Infrastructure. These facilities can be used to implement a log that can record delivery of information.

Table B.2 – continued from previous page

ID	T	Title
30	D	Data Flow <i>Output</i> Is Potentially Interrupted
		Similar to Threat 03 and Threat 23 this threat is mitigated by using Mobile Agents. If the output to the External Interactor is interrupted, unless the interruption is permanent, the Mobile Agent can simply output the information again.
31	T	Compromised Distributed Secure Agent Platform Tamers With Mobile Agent Output
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
32	I	Compromised Distributed Secure Agent Platform Sniffs Data Flow
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
33	D	Compromised Distributed Secure Agent Platform Enacts Process Crash or Stop for the Mobile Agent
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.

B.4 Agent - Core Secure Agent Infrastructure Communication

Here we discuss the mitigation of the threats to the Distributed Secure Agent Platform Outpost (DSAP Outpost) communicating with the Core Secure Agent Infrastructure. In total we discuss 21 threats. Of these 21 threats, 19 threats are mitigated by our security solution by enforcing load-time integrity on the DSAP Outpost. The only two threats not mitigated by our security solution enforcing load-time integrity are Threat 42 and Threat 45. These two threats pertain to an external entity interrupting the Mobile Agent - External Interactor communication and are mitigated by using the Mobile Agent paradigm.

Table B.3: Mitigation status of threats to a Mobile Agent communicating with the Core Secure Agent Infrastructure using the Distributed Secure Agent Platform’s communication facilities. Here we investigate how our DSAP Outpost security solution comprising the Trusted Docking Station and the Secure Docking Module mitigates threats pertaining to DSAP Outpost - Core Secure Agent Infrastructure communication.

ID	T	Title
<i>MA → DSAP</i>		
34	S	Compromised Distributed Secure Agent Platform Spoofs the Mobile Agent Process
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
35	S	Spoofing the Distributed Secure Agent Platform Process
		Here we consider the threat of an adversary running a potentially compromised Distributed Secure Agent Platform (DSAP) to investigate and reverse engineer a Mobile Agent. In our threat model, the only way for an adversary to obtain a Mobile Agent is to compromise a legitimate DSAP in the first place. Then the adversary can use the compromised DSAP to harvest agents and analyse them. This is a targeted attack and therefore outside of our adversarial model. Nonetheless, by enforcing load-time integrity on the DSAP Outpost, the threat of a compromised DSAP service is mitigated.
36	T	Potential Lack of Input Validation for Distributed Secure Agent Platform
		The Secure Agent Infrastructure only introduces certified agents into the system. We assume the Secure Agent Infrastructure security policy prohibits tampering with data sent through the DSAP to the Core Secure Agent Infrastructure. Thus in our threat model, the only entity that can realize this threat is a compromised DSAP Outpost. This however is again mitigated by enforcing its load-time integrity.
37	R	Compromised Distributed Secure Agent Platform Reputates Receiving Data
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.

Table B.3 – continued from previous page

ID	T	Title
38	I	Compromised Distributed Secure Agent Platform Sniffs Data Flow
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
39	D	Compromised Distributed Secure Agent Platform Crashes or Stops
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
40	D	Compromised Distributed Secure Agent Platform Interrupts Data Flow <i>MA</i> → <i>Distributed Secure Agent Platform</i>
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
41	E	Compromised Distributed Secure Agent Platform Elevation Using Impersonation
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
<hr/> <i>DSAP</i> → <i>CSAI</i> <hr/>		
42	D	Data Flow <i>DSAP</i> → <i>CSAI</i> Is Potentially Interrupted
		As with Threat 30, Threat 23, and Threat 03 the use of Mobile Agents mitigates problems arising from temporary communication interruptions.
<hr/> <i>CSAI</i> → <i>DSAP</i> <hr/>		
43	R	Potential Data Repudiation by Distributed Secure Agent Platform
		Only a compromised DSAP can realize this threat in our threat model. Thus, this threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.

Table B.3 – continued from previous page

ID	T	Title
44	D	Potential Process Crash or Stop for the Distributed Secure Agent Platform
		There are several potential realizations of this threat. We exclude the case of an under-specified hardware platform as out of scope. In this case the device used to execute the DSAP Outpost has too little computational resources to operate the DSAP Outpost in a timely fashion. We mitigate a compromised DSAP Outpost from violating availability by enforcing its load-time integrity.
45	D	External Entity Interrupts Data Flow <i>CSAI</i> → <i>DSAP</i>
		As with Threat 42, Threat 30, Threat 23, and Threat 03 the use of Mobile Agents mitigates problems arising from temporary communication interruptions. The Core Secure Agent Infrastructure can resend messages to the DSAP when the connection is reestablished.
46	D	Compromised Distributed Secure Agent Platform Interrupts Data Flow <i>CSAI</i> → <i>DSAP</i>
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
<i>DSAP</i> → <i>MA</i>		
47	S	Compromised Distributed Secure Agent Platform Impersonates the Core Secure Agent Infrastructure
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
48	S	Compromised Distributed Secure Agent Platform Spoofs the Mobile Agent Process
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
49	T	Potential Lack of Input Validation for the Mobile Agent
		In our threat model, the only entity that can realize this threat is a compromised DSAP Outpost. Thus, this threat is mitigated by enforcing load-time its integrity; see Section 5.9.2 and Threat 04 for details.

Table B.3 – continued from previous page

ID	T	Title
50	R	Compromised Distributed Secure Agent Platform Modifies Mobile Agent to Enact Data Repudiation by the Mobile Agent
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
51	I	Compromised Distributed Secure Agent Platform Sniffs Data Flow
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
52	D	Compromised Distributed Secure Agent Platform Enacts Process Crash or Stop for the Mobile Agent
		This threat is mitigated by enforcing the load-time integrity of the DSAP Outpost; see Section 5.9.2 and Threat 04 for details.
53	D	Data Flow <i>DSAP</i> → <i>MA</i> Is Potentially Interrupted
		Only a compromised DSAP can realize this threat in our threat model. Thus, this threat is mitigated by enforcing its load-time integrity; see Section 5.9.2 and Threat 04 for details.
54	E	Elevation Using Impersonation
		In our threat model, the only entity that can realize this threat is a compromised DSAP. Thus, this threat is mitigated by enforcing its load-time integrity; see Section 5.9.2 and Threat 04 for details.

C

Threat Modeling the Secure Block Device

C.1 Introduction

This appendix is a companion piece to Chapter 6. Here we create a threat model for the Secure Block Device in its primary use case, as a secure storage component for a Trusted Application running on ANDIX OS. As detailed in Section 2.18 ANDIX OS uses the Normal World OS to store the data of Trusted Applications. In line with the ARM TrustZone concept, we do not trust the Normal World OS to protect the confidentiality and integrity of the data it stores for Trusted Applications. Therefore, we want to model the threats to the Secure Block Device Trusted Application use case using the Microsoft Security Development Lifecycle threat modeling methodology as implemented in the Microsoft Threat Modeling Tool 2016.

The goal of our threat modeling efforts is to establish a list of threats for a Trusted Application that uses the Secure Block Device for data protection. We then use this list of threats to evaluate the effectiveness of the Secure Block Device. We detail the security evaluation of the Secure Block Device using this list of threats in Section 6.4. In addition to generating the threat list for the Secure Block Device's security evaluation we also record our observations on using the Microsoft Threat Modeling Tool 2016.

In this appendix we first discuss the creation of our threat model. Then we give an adversarial model. Finally, we conclude this chapter with a list of threats and their description.

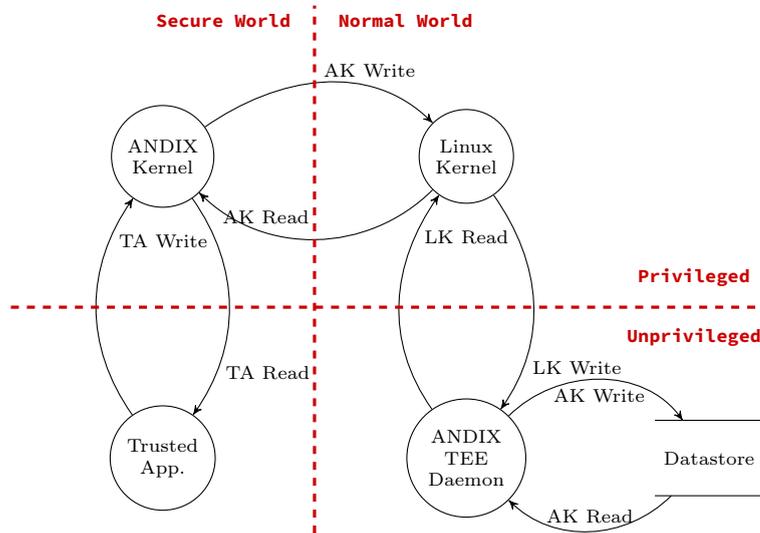


Figure C.1: The first model we considered for modeling a Trusted Application storing its data in an untrusted Datastore.

C.2 Threat Model

The key for successful modeling is striking the right balance between abstraction and accuracy. In this regard threat modeling is not different. We have considered three different models for modeling the threats for our use-case. We have modelled all three models using the Microsoft Threat Modeling Tool 2016 (cf. Section 2.8.4). We based our first model (Figure C.1) on ANDIX OS' storage architecture modeling all five major components in the storage system. The Microsoft Threat Modeling Tool 2016 generated a list of 70 potential threats for this model. Naturally, a significant portion of the threats related to communication between the different components. For example the analysis pointed out threats such as a message ostensibly coming from the Linux Kernel triggering a potential code execution vulnerability in the ANDIX Kernel. These threats offer important insights for securing the ANDIX architecture and implementation, but can be collapsed into threats more pertinent to our secure storage use-case. For instance, any threat pertaining to an adversary being able to execute code with the privilege level of the Linux Kernel, potentially allows such an adversary to read, modify and delete all message exchanged between the Trusted Application and the Datastore.

Here, we want to focus on analysing the secure storage use-case. Therefore we consider analysing the security of the Linux Kernel and the ANDIX Kernel out of scope. As a consequence we decided to remove these components from our model.

Figure C.2 illustrates our second model. The Trusted Application resides

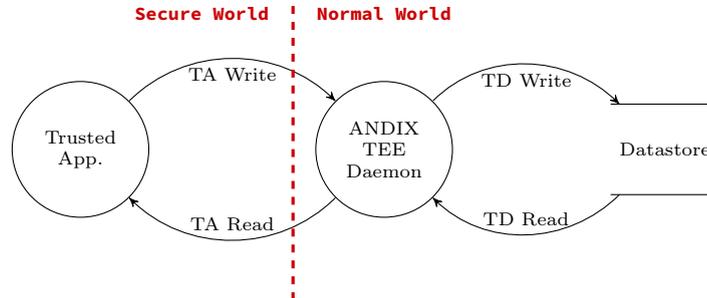


Figure C.2: The second model we considered for modeling a Trusted Application storing its data in an untrusted Datastore in the Normal World.

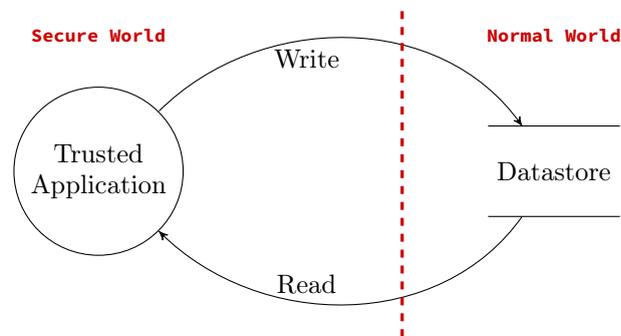


Figure C.3: The third model we considered for modeling a Trusted Application storing its data in an untrusted Datastore.

in the Secure World on the left side, whereas the ANDIX TEE Daemon and the Datastore run in the Normal World to the right. By removing the ANDIX Kernel and the Linux Kernel from the model, we also removed the distinction between a privileged (kernel) and unprivileged (application) mode. However, we included the ANDIX TEE Daemon in this model to study its contribution to the list of threats. In this model the Trusted Application communicates directly with the ANDIX TEE Daemon which in turn uses the Datastore to load and store data. The Microsoft Threat Modeling Tool 2016 generated 26 threats for this model.

Finally, we also analysed the effect of removing the ANDIX TEE Daemon from the model. Figure C.3 depicts the third model. The Microsoft Threat Modeling Tool 2016 generates 17 potential threats for this model. Here we performed a threat by threat comparison between the 26 threats generated for the second model and the 17 threats of the third. A key observation of this comparison is that both threat lists contained threats that were unique to that list and pertinent to the overall threat model. We list these threats in Table C.1 for the second model and in Table C.2 for the third model. For instance, the

Table C.1: Threats generated by the Microsoft Threat Modeling Tool 2016 for the second model (see Figure C.2) that have no direct representation in the third model

STRIDE	Threat Description
<i>TA Read</i>	
✓	✓ Elevation Using Impersonation
✓	✓ Potential Lack of Input Validation for the Trusted Application
	✓ Cross Site Request Forgery
<i>TA Write</i>	
✓	✓ Elevation Using Impersonation
	✓ Potential Lack of Input Validation for the Trusted Application
	✓ Potential Process Crash or Stop for the ANDIX TEE Daemon
	✓ ANDIX TEE Daemon May be Subject to Elevation of Privilege Using Remote Code Execution
	✓ Elevation by Changing the Execution Flow in ANDIX TEE Daemon
	✓ Cross Site Request Forgery

threat list for the second model points out that

Potential Lack of Input Validation for Trusted Application

is a threat. The description of this threat continues to detail that

Data flowing across *TA Read* may be tampered with by an attacker. This may lead to a denial of service attack against Trusted Application or an elevation of privilege attack against Trusted Application or an information disclosure by Trusted Application. Failure to verify that input is as expected is a root cause of a very large number of exploitable issues. Consider all paths and the way they handle data.

Table C.2: Threats generated by the Microsoft Threat Modeling Tool 2016 for the third model (see Figure C.3) that have no direct representation in the second model

STRIDE	Threat Description
<i>Read</i>	
✓	✓ Datastore inaccessible
<i>Write</i>	
✓	✓ Datastore inaccessible

Verify that all input is verified for correctness using an approved list input validation approach.

Although the threat list for the third model contains two elevation of privilege and two Denial-of-Service attacks the fact that corrupting the Trusted Application can lead to an information disclosure is missing.

Conversely the threat list for the third model contains a threat

Data Store Inaccessible

for both the *Read* and *Write* data flow in Figure C.3 that is missing in the second model's threat list. However, the second threat list points out that the ANDIX TEE Daemon could crash or stop, which has a similar effect on availability from the Trusted Application point of view. We believe this to be the reason why the STRIDE-per-interaction heuristic does not generate the "Data Store Inaccessible" threat when a process reads from a Datastore in the same trust domain.

Finally, both threat lists contain entries where we cannot distinctly say that they identify the same threat. For example, the third threat list states that

The Datastore Data Store Could Be Corrupted.

The description of this threat reveals that

Data flowing across *TA Write* may be tampered with by an attacker. This may lead to corruption of Datastore. Ensure the integrity of the data flow to the Datastore.

Although the second threat list contains a threat pointing out a

Potential Lack of Input Validation for ANDIX TEE Daemon,

where the description states that

Data flowing across *TA Write* may be tampered with by an attacker. This may lead to a Denial-of-Service attack against ANDIX TEE Daemon or [...]

We are not sure if this points at the same threat. Even if it does we think that pointing out the need for input validation on an outgoing data flow to indicate potential data corruption is counterintuitive.

The net result of the threat by threat comparison of both models is that we chose to use an amalgam of both threat models. We base our threat model on the third model, but compound it with threats from the second model. Furthermore, we believe that the Microsoft Threat Modeling Tool family is beneficial for threat modeling. However, these tools use a heuristic for threat generation (STRIDE-per-interaction), and we think it is important to keep this in mind, when working with these tools. Finally, we suggest to try out different levels of abstractions when modeling with this tool family to help detect more potential threats.

C.3 Adversarial Model

The following two paragraphs are duplicated from Section 6.4.1. We include them here for convenience.

We analyse the security properties of the Secure Block Device (SBD) against an adversary that has full access to the cryptographically protected data at rest, but not to the Authenticated Encryption key and the Merkle Tree root hash. This adversary can read and modify all data while it is at rest and record operations on the data over the full lifetime of the SBD store, but cannot break the cryptography.

Specifically, when the SBD runs in a Trusted Application, the following assumptions about our adversary hold. Our adversary is restricted to software attacks. Our adversary has full privileges on the Normal World side, but can only access the Secure World and the Trusted Applications therein using the Remote Procedure Call interface provided by ANDIX OS. In the Normal World the adversary has full control over all processes, can access all memory, and can fully read and write all files, including the Datastore used by our Trusted Application. We do however deny our adversary the ability to perform software side-channel attacks to obtain the cryptographic keys used by the SBD, as we consider this to be out of scope for this work.

C.4 List of Threats

Following the approach of the Microsoft Threat Modeling Tool 2016, we list potential threats per data flow. Thus we have split the threat list into two tables. The first, Table C.3, details threats arising from reading from an untrusted Datastore, and the second, Table C.4, describes the threats pertaining to writing to an untrusted Datastore. Both tables were created by generating a threat analysis for the third model (cf. Figure C.3) using the Microsoft Threat Modeling Tool 2016. The list of threats and their descriptions was manually adapted to better reflect the use case they model and our adversarial model (see Section C.3). Finally, we compounded the lists of threats with threats inspired by a threat analysis of the second model.

Table C.3: List of potential threats, when a Trusted Application reads data from an untrusted Datastore.

S T R I D E	Threat Description
✓	<p>1 Spoofing of Datastore</p> <p>Our adversary can spoof the Datastore. Thus the adversary might serve arbitrary data to the Trusted Application. Given that the adversary has read access to the Datastore (DS) the adversary can serve a mash-up of valid data from the Datastore and forged data created by the adversary.</p>

Table C.3 – continued from previous page

S T R I D E	Threat Description
✓	<p>2 Weak access control for the Datastore</p> <p>Our adversary can read information not intended for disclosure, due to improper data protection of the Datastore. More specifically, our adversary can read any part of the Datastore.</p>
✓	<p>2a Data Flow Sniffing</p> <p>Our adversary can sniff any data sent from the Datastore to the Trusted Application.</p>
✓	<p>3 Spoofing the Trusted Application</p> <p>Our adversary may impersonate the Trusted Application versus the Datastore. As our adversary has full access to the Datastore and can furthermore eavesdrop, intercept, and modify all data read from, and written to, the Datastore our adversarial model subsumes this threat.</p>
✓	<p>4 Potential data repudiation by the Trusted Application</p> <p>The Trusted Application claims that it did not receive any data from the Datastore. As the Trusted Application is trusted by definition we disregard this threat.</p>
✓	<p>5 Potential process crash or stop for the Trusted Application</p> <p>The Trusted Application crashes, halts, stops or runs slowly; in all cases violating an availability metric. The Trusted Application reads data from the Datastore, presumably as part of providing some functionality or service at a higher level of interaction, for example directly to a user of the system. Thus, the effect of a potential crash or stop of the Trusted Application needs to be considered in the context of the purpose of the Trusted Application. Furthermore, the Trusted Application is considered trusted by definition, thus we disregard this threat.</p>

Table C.3 – continued from previous page

S T R I D E	Threat Description
✓	<p>6 Data flow <i>Read</i> from the Datastore to the Trusted Application is potentially interrupted</p> <p>Our adversary interrupts the data flow, when the Trusted Application is reading from the DS. Thus our adversary can potentially change the behavior of the Trusted Application by selectively interrupting specific data read operations up to and including a complete Denial-of-Service attack. This is similar, but not identical in effect to selectively destroying portions of data in the Datastore.</p>
✓	<p>7 Datastore inaccessible</p> <p>Our adversary prevents access to the Datastore. The Trusted Application will not receive requested data. The effect of this Denial-of-Service attack needs to be gauged in the context of the purpose of the Trusted Application. Here, we assume it to be a pertinent threat.</p>
✓	<p>8 The Trusted Application may be subject to elevation of privilege using remote code execution</p> <p>Our adversary might try injecting tampered data into the Trusted Application in a way that causes a remote code execution in Trusted Application. For example, the adversary can tamper with the data stored in the Datastore, and next time the Trusted Application reads the data it causes a remote code execution. We consider the Trusted Application trusted by definition, and as such free of implementation vulnerabilities that would allow remote code execution. Thus we disregard this threat.</p>

Table C.3 – continued from previous page

S T R I D E	Threat Description
✓	<p>9 Elevation by changing the execution flow in the Trusted Application</p> <p>Our adversary can inject tampered data into the Trusted Application in a way that changes the flow of the program execution within the Trusted Application. For example, the adversary could impersonate the Datastore and send tampered data that changes the behavior of the Datastore in favor of the adversary. We do consider the Trusted Application trusted and free of implementation vulnerabilities. However, an adversary tampering with the data cannot always be detected with logical checks, without measures that ensure overall data integrity. Therefore, we consider this threat pertinent.</p>

Table C.4: List of potential threats, when a Trusted Application writes data to an untrusted Datastore.

S T R I D E	Threat Description
✓	<p>10 Spoofing of the Datastore</p> <p>The Datastore may be spoofed by our adversary and this may lead to the Trusted Application writing sensitive data to the adversary's target instead of the Datastore.</p>
✓	<p>11 Potential excessive resource consumption for Trusted Application or Datastore</p> <p>The Trusted Application presumably provides services to an application, or directly to the user. For the Trusted Application excessive resource consumption needs to be considered in this context. Furthermore, the Trusted Application is considered trusted by definition, thus we disregard this sub-threat.</p> <p>The Datastore on the other hand might provide its storage service to a wide range of other applications, besides the Trusted Application. Here it might happen that the resource becomes unavailable due to high load, or run out of space to store data. The Trusted Application and also the Secure Block Device library need to take this into account when interacting with the Datastore to prevent deadlocks, and handle out of space exceptions in a controlled manner.</p>

Table C.4 – continued from previous page	
S T R I D E	Threat Description
✓	<p>12 Spoofing the Trusted Application</p> <p>Our adversary can impersonate the Trusted Application. The Datastore might provide its storage service to a wide range of other applications, besides the Trusted Application. Therefore, by spoofing the Trusted Application, our adversary can gain unauthorized access to the Datastore. However, our adversary already has access to the whole Datastore by definition. Thus she can gain access to the Trusted Application’s portion of the Datastore without even needing to impersonate the Trusted Application. Still, we consider a realization of this threat to be a problem, independent of how the threat is realized. Thus we let it stand.</p>
✓	<p>13 The Datastore could be corrupted</p> <p>When the Trusted Application writes to the Datastore, the adversary can tamper with this data flow. As a consequence, the Datastore might become corrupted. Again, our adversary can directly tamper with the data stored in Datastore directly. And again, we consider this threat pertinent, independent of its realization.</p>
✓	<p>14 The Datastore denies writing data, although the data was potentially written</p> <p>The Datastore claims not to have written data sent by the Trusted Application. For example, the Datastore could claim that to obfuscate a subsequent information disclosure.</p>
✓	<p>15 Data flow sniffing</p> <p>When the Trusted Application writes to the Datastore, our adversary can sniff the data flowing from the Trusted Application to the Datastore. Our adversary can by definition also read the data directly from the Datastore. Again, no matter the implementation, we consider a realization of this threat pertinent.</p>

Table C.4 – continued from previous page

S T R I D E	Threat Description
✓	16 The data flow from the Trusted Application to the Datastore is potentially interrupted An adversary interrupts the data flow from the Trusted Application to the Datastore. This threat needs to be considered in the context of the Trusted Application. For example, if the Trusted Application uses the Datastore to save security relevant control flow information, such as a Digital Rights Management access counter. In this example interrupting the update of the counter might violate a security goal. This threat is also of special note to the Secure Block Device, as the Secure Block Device stores security relevant control information in the Datastore (cf. Section 6.2 for the details). Thus the Secure Block Device library needs to prevent inconsistent internal states, when an adversary interrupts a write operation.
✓	17 Datastore inaccessible An adversary prevents the Trusted Application from accessing the Datastore, thus the Trusted Application will not be able to store information. Again, the impact of this threat needs to be evaluated in the context of the Trusted Application.

D

List of Publications And Cooperations

According to the Statutes of the Doctoral School of Computer Science at Graz University of Technology, a PhD thesis must contain a list of publications of the candidate, detailing the relationship between the thesis and the (relevant) publications. The reference date for the following list of publications is June 2016.

In science, many authors rightfully state that we build our work on the shoulders of giants, the people who worked the field before us, and who work on it next to us. Hence, next to the list of publications the Statutes of the Doctoral School of Computer Science at Graz University of Technology also require a PhD thesis to detail the collaborations pertaining to the work presented in it. We describe the collaborations in Section D.4.

D.1 Journal Publications

This section lists all journal publications in reverse chronological order.

- [TFF⁺16] Peter Teuff, Michaela Ferk, Andreas Fitzek, Daniel M. Hein, Stefan Kraxberger, and Clemens Orthacker. Malware detection by applying knowledge discovery processes to application metadata on the Android Market (Google Play). *Security and Communication Networks*, 9(5):389–419, 2016.
- [BGH⁺14] Zoltán Balogh, Emil Gatia, Ladislav Hluchý, Ronald Toegl, Martin Pirker, and Daniel M. Hein. Agent-based cloud resource management for secure cloud infrastructures. *Computing and Informatics*, 33(6):1333–1355, 2014.

- [WFK⁺12] Wojciech Wojciechowicz, Jacques Fournier, Mirosław Konecny, Stefan Vanya, John Stoodley, Phil Entwisle, Daniel M. Hein, Aurel Machalek, Apostolos P. Fournaris, Mikel Uriarte, Oscar Lopez, Shaun O’Neill, Hans Brandl, Zoltán Balogh, Emil Gatial, Ladislav Hluchý, Tomasz Mirosław, and Jan Zych. Seamless communication for crisis management. *Technical Sciences*, 15(1)/2012:65–79, 2012.
- [GBH⁺12] Emil Gatial, Zoltán Balogh, Daniel M. Hein, Ladislav Hluchý, Martin Pirker, and Ronald Toegl. Securing agents using secure docking module. *Technical Sciences*, 15(1)/2012:111–122, 2012.
- [FFHR12] Apostolos P. Fournaris, Jacques Fournier, Daniel Hein, and Guillaume Reymond. Secure docking station and its protection against hardware attacks. *Technical Sciences*, 15(1)/2012:123–138, 2012.
- [DH10] Peter Danner and Daniel M. Hein. A trusted computing identity collation protocol to simplify deployment of new disaster response devices. *J. UCS*, 16(9):1139–1151, 2010.
- [HTK10] Daniel M. Hein, Ronald Toegl, and Stefan Kraxberger. An autonomous attestation token to secure mobile agents in disaster response. *Security and Communication Networks*, 3(5):421–438, 2010.

D.2 Published Book Chapters

This section lists the one published book chapter.

- [FH11] Apostolos P. Fournaris and Daniel M. Hein. Trust management through hardware means: Design concerns and optimizations. In Nikolaos Voros, Amar Mukherjee, Nicolas Sklavos, Konstantinos Masselos, and Michael Huebner, editors, *VLSI 2010 Annual Symposium*, volume 105 of *Lecture Notes in Electrical Engineering*, pages 31–45. Springer Netherlands, 2011.

D.3 Publications in Conference and Workshop Proceedings

This section lists all publications in conference and workshop proceedings in reverse chronological order.

- [LHW15] Christian M. Lesjak, Daniel Hein, and Johannes Winter. Hardware-security technologies for industrial iot: Trustzone and security controller. In *41st Annual Conference of the IEEE Industrial Electronics Society, IECON 2015*, Yokohama, Japan, November 2015.

- [HWF15] Daniel M. Hein, Johannes Winter, and Andreas Fitzek. Secure block device – secure, flexible, and efficient data storage for ARM TrustZone systems. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20–22, 2015, Volume 1*, pages 222–229, 2015.
- [BHRS15] Roderick Bloem, Daniel M. Hein, Franz Röck, and Richard Schumi. Case study: Automatic test case generation for a secure cache implementation. In *Tests and Proofs – 9th International Conference, TAP 2015, Held as Part of STAF 2015, L’Aquila, Italy, July 22–24, 2015. Proceedings*, pages 58–75, 2015.
- [LHH⁺15] Christian M. Lesjak, Daniel Hein, Michael Hofmann, Martin Maritsch, Andreas Aldrian, Peter Priller, Thomas Ebner, Thomas Rupprechter, and Günther Pregartner. Securing smart maintenance services: Hardware-security and TLS for MQTT. In *13th IEEE International Conference on Industrial Informatics, INDIN 2015, Cambridge, United Kingdom, July 22–24, 2015*, pages 1243–1250, 2015.
- [FAWH15] Andreas Fitzek, Florian Achleitner, Johannes Winter, and Daniel Hein. The ANDIX research OS – ARM TrustZone meets industrial control systems security. In *13th IEEE International Conference on Industrial Informatics, INDIN 2015, Cambridge, United Kingdom, July 22–24, 2015*, pages 88–93, 2015.
- [PSE⁺14] Viktoria Pammer, Hermann Stern, Stefan Edler, Daniel M. Hein, Martin Pirker, Roderick Bloem, and Alfred Wertner. Security concepts for a distributed architecture for activity logging and analysis. In Stefanie N. Lindstaedt, editor, *14th International Conference on Knowledge Management and Data-driven Business, I-KNOW ’14, Graz, Austria, September 16–19, 2014*, pages 23:1–23:6. ACM, 2014.
- [OHT14] Alexander Oprisnik, Daniel Hein, and Peter Teufl. Identifying cryptographic functionality in Android applications. In Mohammad S. Obaidat, Andreas Holzinger, and Pierangela Samarati, editors, *SECURITY 2014 – Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria, August 28–30, 2014*, pages 151–162. SciTePress, 2014.
- [TFH⁺14] Peter Teufl, Andreas Fitzek, Daniel Hein, Alexander Marsalek, Alexander Oprisnik, and Thomas Zefferer. Android encryption systems. In *2014 International Conference on Privacy and Security in Mobile Systems, PRISMS 2014, Aalborg, Denmark, May 11–14, 2014*, pages 1–8. IEEE, 2014.
- [TZW⁺14] Peter Teufl, Thomas Zefferer, Christoph Wörgötter, Alexander Oprisnik, and Daniel Hein. Android – on-device detection of SMS catchers and sniffers. In *2014 International Conference on Privacy and Security in Mobile Systems, PRISMS 2014, Aalborg, Denmark, May 11–14, 2014*, pages 1–8. IEEE, 2014.

- [ZRS⁺14] Dominik Ziegler, Mattias Rauter, Christof Stromberger, Peter Teufl, and Daniel Hein. Do you think your passwords are secure? In *2014 International Conference on Privacy and Security in Mobile Systems, PRISMS 2014, Aalborg, Denmark, May 11–14, 2014*, pages 1–8. IEEE, 2014.
- [PWH13a] Klaus Potzmader, Johannes Winter, and Daniel Hein. STUNT: A simple, transparent, user-centered network of trust. In Sokratis K. Katsikas and Isaac Agudo, editors, *Public Key Infrastructures, Services and Applications – 10th European Workshop, EuroPKI 2013, Egham, UK, September 12–13, 2013, Revised Selected Papers*, volume 8341 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2013.
- [PWH⁺13b] Klaus Potzmader, Johannes Winter, Daniel Hein, Christian Hanser, Peter Teufl, and Liqun Chen. Group signatures on mobile devices: Practical experiences. In Michael Huth, N. Asokan, Srdjan Capkun, Ivan Flechais, and Lizzie Coles-Kemp, editors, *Trust and Trustworthy Computing – 6th International Conference, TRUST 2013, London, UK, June 17–19, 2013. Proceedings*, volume 7904 of *Lecture Notes in Computer Science*, pages 47–64. Springer, 2013.
- [FHF⁺13] Apostolos P. Fournaris, Daniel Hein, Jacques Fournier, Guillaume Reymond, and Hans Brandl. A secure docking module for providing trust in crisis management incidents. In *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*, pages 804–809, July 2013.
- [HTP⁺12] Daniel M. Hein, Ronald Toegl, Martin Pirker, Emil Gatial, Zoltán Balogh, Hans Brandl, and Ladislav Hluchý. Securing mobile agents for crisis management support. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing, STC '12*, pages 85–90, New York, NY, USA, 2012. ACM.
- [KDH10] Stefan Kraxberger, Peter Danner, and Daniel M. Hein. Secure multi-agent system for multi-hop environments. In Igor V. Kottenko and Victor A. Skormin, editors, *Computer Network Security, 5th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2010, St. Petersburg, Russia, September 8–10, 2010. Proceedings*, volume 6258 of *Lecture Notes in Computer Science*, pages 270–283. Springer, 2010.
- [DHK10] Peter Danner, Daniel M. Hein, and Stefan Kraxberger. Securing emergency response operations using distributed trust decisions. In Yang Xiang, Pierangela Samarati, Jiankun Hu, Wanlei Zhou, and Ahmad-Reza Sadeghi, editors, *Fourth International Conference on Network and System Security, NSS 2010, Melbourne, Victoria, Australia, September 1–3, 2010*, pages 62–69. IEEE Computer Society, 2010.
- [HMHW09] Michael Hutter, Marcel Medwed, Daniel M. Hein, and Johannes Wolkerstorfer. Attacking ECDSA-enabled RFID devices. In Michel

Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2–5, 2009. Proceedings*, volume 5536 of *Lecture Notes in Computer Science*, pages 519–534, 2009.

- [HT09] Daniel M. Hein and Ronald Toegl. An autonomous attestation token to secure mobile agents in disaster response. In Andreas U. Schmidt and Shiguo Lian, editors, *Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec 2009, Turin, Italy, June 3–5, 2009, Revised Selected Papers*, volume 17 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 46–57. Springer, 2009.
- [PTHD09] Martin Pirker, Ronald Toegl, Daniel M. Hein, and Peter Danner. A PrivacyCA for anonymity and trust. In Liqun Chen, Chris J. Mitchell, and Andrew Martin, editors, *Trusted Computing, Second International Conference, Trust 2009, Oxford, UK, April 6–8, 2009, Proceedings*, volume 5471 of *Lecture Notes in Computer Science*, pages 101–119. Springer, 2009.
- [HWF08] Daniel M. Hein, Johannes Wolkerstorfer, and Norbert Felber. ECC is ready for RFID – A proof in silicon. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14–15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 2008.

D.4 Relationship between Publications and Thesis

This thesis is based on the following publications (in chronological order):

- [HT09] Daniel M. Hein and Ronald Toegl. An autonomous attestation token to secure mobile agents in disaster response. In Andreas U. Schmidt and Shiguo Lian, editors, *Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec 2009, Turin, Italy, June 3–5, 2009, Revised Selected Papers*, volume 17 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 46–57. Springer, 2009.
- [HTK10] Daniel M. Hein, Ronald Toegl, and Stefan Kraxberger. An autonomous attestation token to secure mobile agents in disaster response. *Security and Communication Networks*, 3(5):421–438, 2010.
- [HTP⁺12] Daniel M. Hein, Ronald Toegl, Martin Pirker, Emil Gatia, Zoltán Balogh, Hans Brandl, and Ladislav Hluchý. Securing mobile agents

for crisis management support. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing, STC '12*, pages 85–90, New York, NY, USA, 2012. ACM.

- [HWF15] Daniel M. Hein, Johannes Winter, and Andreas Fitzek. Secure block device – secure, flexible, and efficient data storage for ARM Trust-Zone systems. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20–22, 2015, Volume 1*, pages 222–229, 2015.
- [BHRS15] Roderick Bloem, Daniel M. Hein, Franz Röck, and Richard Schumi. Case study: Automatic test case generation for a secure cache implementation. In *Tests and Proofs – 9th International Conference, TAP 2015, Held as Part of STAF 2015, L’Aquila, Italy, July 22–24, 2015. Proceedings*, pages 58–75, 2015.

Throughout this thesis, we have marked sections which reuse material from these publications with grey boxes entitled **Declaration of Sources**. Additionally, here we explain the most important relations between these publications and this thesis, and also detail with whom we have collaborated and how.

We introduced the concept of the Secure Docking Module (SDM) in [HT09]. We later elaborated the concept and presented first results based on a software emulation of the SDM in [HTK10]. We concluded are published work on the SDM with [HTP⁺12]. There we present our final results with the security controller based implementation of the SDM integrated with the acTvSM platform and the Secure Agent Infrastructure.

In [HT09, HTK10] Ronald Toegl contributed to the design of the SDM and helped us transition into the field of Trusted Computing with his vast knowledge of previous work in this field. Stefan Kraxberger contributed significantly to [HTK10], where he helped us to investigate the use of the SDM for authorizing network access to an ad-hoc peer-to-peer overlay network. We have completely elided this work in this thesis. The work presented in [HTP⁺12] is a collaboration of the people involved in the Secure Agent Infrastructure, the acTvSM platform, and us. Emil Gatjal, Zoltán Balogh, and Ladislav Hluchý are the minds behind the Secure Agent Infrastructure and they helped us integrate the Secure Agent Infrastructure with the SDM and the acTvSM platform. Martin Pirker, Ronald Tögl, and Michael Gissing helped us to setup and use the acTvSM platform, as well as, integrating it with the Secure Agent Infrastructure and the SDM. Nina Mocnik and Thomas Kastner tackled the Debian GNU Linux Live CD build system and were responsible for packaging the DSAP Outpost Application Virtual Machine. Last, but not least, Hans Brandl and Marco Scheibe were instrumental in providing us access to the Security Controller platform we used to implement the SDM.

In [HWF15] we present and evaluate the Secure Block Device. As such, Chapter 6 is largely based on [HWF15]. The Secure Block Device also features prominently in [BHRS15], where its cache component is used for a case study on automated test case generation. We have adapted the description of the Secure Block Device’s cache component in Chapter 6 from [BHRS15].

Johannes Winter and Andreas Fitzek helped us integrating the Secure Block Device into ANDIX OS and evaluating the Secure Block Device on the iMX53 Quick Start Board. More specifically, Andreas Fitzek helped with the ANDIX OS integration and Secure Block Device evaluation code. Johannes Winter was instrumental in evaluating the Secure Block Device on the iMX53 Quick Start Board. Johannes Winter also came up with the idea to evaluate the Secure Block Device by using the Secure Block Device Trusted Application as protection component for a Linux Network Block Device and helped significantly with evaluating it. Finally, in [BHRS15] we helped Franz Röck to integrate the Secure Block Device in his automated test case generation tool. Roderick Bloem provided valuable insights and ensured that a high quality publication arose from our work.

Bibliography

- [ABCS06] Ross Anderson, Mike Bond, Jolyon Clulow, and Sergei Skorobogatov. Cryptographic processors – a survey. *Proceedings of the IEEE*, 94(2):357–369, February 2006.
- [AES15] Gorka Irazoqui Apecechea, Thomas Eisenbarth, and Berk Sunar. S\$a: A shared cache attack that works across cores and defies VM sandboxing – and its application to AES. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17–21, 2015*, pages 591–604, 2015.
- [Ale03] Ian F. Alexander. Misuse cases: Use cases with hostile intent. *IEEE Software*, 20(1):58–66, 2003.
- [Apa16a] Apache Foundation. Apache river documentation. <https://river.apache.org/concepts.html>, 2016. [Online; accessed: January 12, 2016].
- [Apa16b] Apache Foundation. Apache river jini specification. <https://river.apache.org/doc/specs/html/jini-spec.html>, 2016. [Online; accessed: January 12, 2016].
- [aT14] SDL Team. Introducing Microsoft threat modeling tool 2014. Microsoft Cyber Trust Blog (<https://blogs.microsoft.com/cybertrust/>), April 2014.
- [BBF⁺04] Elisa Bertino, Danilo Bruschi, Stefano Franzoni, Igor Nai Fovino, and Stefano Valtolina. Threat modelling for SQL servers – designing a secure database in a web application. In *Communications and Multimedia Security – 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, September 15–18, 2004, Windermere, The Lake District, United Kingdom*, pages 159–171, 2004.
- [BC02] Elmarie Bierman and Elsabe Cloete. Classification of malicious host threats in mobile agent computing. In *Proceedings of the 2002 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology, SAICSIT '02*, pages 141–148, Republic of South Africa, 2002. South African Institute for Computer Scientists and Information Technologists.

- [BEG⁺94] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. *Algorithmica*, 12(2/3):225–244, 1994.
- [BG07] Shane Balfe and Eimear Gallery. Mobile agents and the deus ex machina. In *21st International Conference on Advanced Information Networking and Applications (AINA 2007), Workshops Proceedings, Volume 2, May 21–23, 2007, Niagara Falls, Canada*, pages 486–492. IEEE Computer Society, 2007.
- [BGH⁺11] Zoltán Balogh, Emil Gatial, Ladislav Hluchý, Vlasta Hudek, and Michal Konecný. Integration of secure agents with a secure communication infrastructure for crisis management. In *2011 International Conference on Collaboration Technologies and Systems, CTS 2011, Philadelphia, Pennsylvania, USA, May 23–27, 2011*, pages 367–372, 2011.
- [BGH⁺14] Zoltán Balogh, Emil Gatial, Ladislav Hluchý, Ronald Toegl, Martin Pirker, and Daniel M. Hein. Agent-based cloud resource management for secure cloud infrastructures. *Computing and Informatics*, 33(6):1333–1355, 2014.
- [BHR515] Roderick Bloem, Daniel M. Hein, Franz Röck, and Richard Schumi. Case study: Automatic test case generation for a secure cache implementation. In *Tests and Proofs – 9th International Conference, TAP 2015, Held as Part of STAF 2015, L’Aquila, Italy, July 22–24, 2015. Proceedings*, pages 58–75, 2015.
- [Bis02] Matthew A. Bishop. *The Art and Science of Computer Security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
- [BMMV06] Vidhya Balasubramanian, Daniel Massaguer, Sharad Mehrotra, and Nalini Venkatasubramanian. Drillsim: A simulation framework for emergency response drills. In *Intelligence and Security Informatics, IEEE International Conference on Intelligence and Security Informatics, ISI 2006, San Diego, CA, USA, May 23–24, 2006, Proceedings*, pages 237–248, 2006.
- [BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.
- [Bor02] N. Borselius. Mobile agent security. *Electronics Communication Engineering Journal*, 14(5):211–218, October 2002.
- [CGHH89] Paul R. Cohen, Michael L. Greenberg, David M. Hart, and Adele E. Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3):32–48, 1989.

- [CGL⁺11] George Coker, Joshua D. Guttman, Peter Loscocco, Amy L. Herzog, Jonathan K. Millen, Brian O’Hanlon, John D. Ramsdell, Ariel Segall, Justin Sheehy, and Brian T. Sniffen. Principles of remote attestation. *Int. J. Inf. Sec.*, 10(2):63–81, 2011.
- [Cha04] David Chadwick. Threat modelling for active directory. In *Communications and Multimedia Security – 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, September 15–18, 2004, Windermere, The Lake District, United Kingdom*, pages 173–182, 2004.
- [Cor05] Carlos Coronado. On the security and the efficiency of the merkle signature scheme. *IACR Cryptology ePrint Archive*, 2005:192, 2005.
- [CPV07] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Is code still moving around? looking back at a decade of code mobility. In *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20–26, 2007, Companion Volume*, pages 9–20, 2007.
- [CRP08] Victor Chan, Pradeep Ray, and Nandan Parameswaran. Mobile e-health monitoring: an agent-based approach. *IET Communications*, 2(2):223–230, 2008.
- [CWS⁺04] Danny De Cock, Karel Wouters, Dries Schellekens, Dave Singelée, and Bart Preneel. Threat modelling for security tokens in web applications. In *Communications and Multimedia Security – 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, September 15–18, 2004, Windermere, The Lake District, United Kingdom*, pages 183–193, 2004.
- [DH10] Peter Danner and Daniel M. Hein. A trusted computing identity collation protocol to simplify deployment of new disaster response devices. *J. UCS*, 16(9):1139–1151, 2010.
- [DHK10] Peter Danner, Daniel M. Hein, and Stefan Kraxberger. Securing emergency response operations using distributed trust decisions. In Yang Xiang, Pierangela Samarati, Jiankun Hu, Wanlei Zhou, and Ahmad-Reza Sadeghi, editors, *Fourth International Conference on Network and System Security, NSS 2010, Melbourne, Victoria, Australia, September 1–3, 2010*, pages 62–69. IEEE Computer Society, 2010.
- [DJPJ04a] Lieven Desmet, Bart Jacobs, Frank Piessens, and Wouter Joosen. A generic architecture for web applications to support threat analysis of infrastructural components. In *Communications and Multimedia Security – 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, September 15–18, 2004, Windermere, The Lake District, United Kingdom*, pages 125–130, 2004.

- [DJPJ04b] Lieven Desmet, Bart Jacobs, Frank Piessens, and Wouter Joosen. Threat modelling for web services based web applications. In *Communications and Multimedia Security – 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, September 15–18, 2004, Windermere, The Lake District, United Kingdom*, pages 131–144, 2004.
- [DK06] Guillaume Duc and Ronan Keryell. Cryptopage: An efficient secure architecture with memory encryption, integrity and information leakage protection. In *22nd Annual Computer Security Applications Conference (ACSAC 2006), December 11–15, 2006, Miami Beach, Florida, USA*, pages 483–492. IEEE Computer Society, 2006.
- [DL89] Edmund H. Durfee and Victor R. Lesser. Negotiating task decomposition and allocation using partial global planning. *Distributed artificial intelligence*, 2(1):229–244, 1989.
- [Dwo05] Morris J. Dworkin. SP 800-38B. recommendation for block cipher modes of operation: The CMAC mode for authentication. Technical report, Gaithersburg, MD, United States, 2005.
- [ECG⁺09] Reouven Elbaz, David Champagne, Catherine H. Gebotys, Ruby B. Lee, Nachiketh R. Potlapally, and Lionel Torres. Hardware mechanisms for memory authentication: A survey of existing techniques and engines. *Transactions on Computational Science*, 5430:1–22, 2009.
- [FAWH15] Andreas Fitzek, Florian Achleitner, Johannes Winter, and Daniel Hein. The ANDIX research OS – ARM TrustZone meets industrial control systems security. In *13th IEEE International Conference on Industrial Informatics, INDIN 2015, Cambridge, United Kingdom, July 22–24, 2015*, pages 88–93, 2015.
- [Fer02] Niels Ferguson. Collision attacks on OCB. www.cs.ucdavis.edu/~rogaway/ocb/fe02.pdf, 2002.
- [FFHR12] Apostolos P. Fournaris, Jacques Fournier, Daniel Hein, and Guillaume Reymond. Secure docking station and its protection against hardware attacks. *Technical Sciences*, 15(1)/2012:123–138, 2012.
- [FH11] Apostolos P. Fournaris and Daniel M. Hein. Trust management through hardware means: Design concerns and optimizations. In Nikolaos Voros, Amar Mukherjee, Nicolas Sklavos, Konstantinos Masselos, and Michael Huebner, editors, *VLSI 2010 Annual Symposium*, volume 105 of *Lecture Notes in Electrical Engineering*, pages 31–45. Springer Netherlands, 2011.
- [FHF⁺13] Apostolos P. Fournaris, Daniel Hein, Jacques Fournier, Guillaume Reymond, and Hans Brandl. A secure docking module for providing trust in crisis management incidents. In *Industrial Informatics*

- (INDIN), 2013 11th IEEE International Conference on, pages 804–809, July 2013.
- [Fit14] Andreas Fitzek. Development of an ARM TrustZone aware operating system ANDIX OS. Master’s thesis, Graz University of Technology, 2014.
- [Fou10] Apostolos P. Fournaris. Hardware module design for ensuring trust. In *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2010, July 5–7, 2010, Lixouri Kefalonia, Greece*, pages 155–160, 2010.
- [Gal88] Julia Rose Galliers. *A Theoretical Framework for Computer Models of Cooperative Dialogue, Acknowledging Multiagent Conflict*. PhD thesis, Open University (United Kingdom), 1988. AAIDX87541.
- [Gat10a] Emil Gatial. *Approach to Coordinating Distributed Execution of Agents for Trusted Information Collection*. PhD thesis, Institute of Informatics, Slovak Academy of Sciences, 2010.
- [Gat10b] Emil Gatial. Distributed secure agent platform source code. Subversion code repository, September 2010. Last commit: September 09, 2010 09:19:38.
- [GBH10] Emil Gatial, Zoltán Balogh, and Ladislav Hluchý. Platform for distributed execution of agents for trusted data collection. In *Proceedings of the International Conference on Computational Science, ICCS 2010, University of Amsterdam, The Netherlands, May 31–June 2, 2010*, number 1, pages 2023–2032, 2010.
- [GBH⁺12] Emil Gatial, Zoltán Balogh, Daniel M. Hein, Ladislav Hluchý, Martin Pirker, and Ronald Toegl. Securing agents using secure docking module. *Technical Sciences*, 15(1)/2012:111–122, 2012.
- [GBŠH11] Emil Gatial, Zoltán Balogh, Branislav Šimo, and Ladislav Hluchý. Distributed secure agent platform for crisis management. In *Applied Machine Intelligence and Informatics (SAMI), 2011 IEEE 9th International Symposium on*, pages 247–253, January 2011.
- [GE04] Rüdiger Grimm and Henrik Eichstädt. Threat modelling for ASP.NET – designing secure applications. In *Communications and Multimedia Security – 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, September 15–18, 2004, Windermere, The Lake District, United Kingdom*, pages 145–158, 2004.
- [Glo10] GlobalPlatform Inc. Globalplatform device technology TEE client API specification. <https://www.globalplatform.org/specificationform.asp?fid=7339>, July 2010.

- [Glo11] GlobalPlatform Inc. Globalplatform device technology TEE internal API specification. <https://www.globalplatform.org/specificationform.asp?fid=7762>, December 2011.
- [Glo13] GlobalPlatform Inc. Globalplatform device technology trusted user interface API. <https://www.globalplatform.org/specificationform.asp?fid=7779>, June 2013.
- [GNV09] Eimear Gallery, Aarthi Nagarajan, and Vijay Varadharajan. A property-dependent agent transfer protocol. In *Trusted Computing, Second International Conference, Trust 2009, Oxford, UK, April 6–8, 2009, Proceedings*, pages 240–263, 2009.
- [GR05] Berndt M. Gammel and Stefan J. Rüping. Smart cards inside. In *Solid-State Device Research Conference, 2005. ESSDERC 2005. Proceedings of 35th European*, pages 69–74, September 2005.
- [Gra09] David Grawrock. *Dynamics of a Trusted Platform: A Building Block Approach*. Intel Press, 1st edition, 2009.
- [GSM15] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In *USENIX Security Symposium*. USENIX Association, 2015.
- [GSMB03] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. Sirius: Securing remote untrusted storage. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA*. The Internet Society, 2003.
- [HBG10] Ladislav Hluchý, Zoltán Balogh, and Emil Gatjal. Distributed agent-based architecture for management of crisis situations using trusted code execution. In *Applied Machine Intelligence and Informatics (SAMII), 2010 IEEE 8th International Symposium on*, pages 25–30, January 2010.
- [HCD08] Jason Honda, Harry H. Cheng, and Donna Djordjevich. Mobile-first: a mobile agent based first responder system. In *Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering (SEKE'2008), San Francisco, CA, USA, July 1–3, 2008*, pages 565–568, 2008.
- [HDFL09] Daniel Hein, Peter Danner, Apostolos P. Fournaris, and Martin Liebl. SECRI COM deliverable 5.1, functional specification of the secure docking module. <http://www.secri.com.eu/public-deliverables>, May 2009.
- [HKA⁺14] Shafiq Hussain, Asif Kamal, Shabir Ahmad, Ghulam Rasool, and Sajid Iqbal. Threat modelling methodologies: A survey. *Sci. Int.(Lahore)*, 26(4):1607–1609, 2014.

- [HL02] Michael Howard and David E. Leblanc. *Writing Secure Code*. Microsoft Press, Redmond, WA, USA, 2nd edition, 2002.
- [HL06] Michael Howard and Steve Lipner. *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA, 2006.
- [HLMN08] Charles B. Haley, Robin C. Laney, Jonathan D. Moffett, and Bashar Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Trans. Software Eng.*, 34(1):133–153, 2008.
- [HMHW09] Michael Hutter, Marcel Medwed, Daniel M. Hein, and Johannes Wolkerstorfer. Attacking ECDSA-enabled RFID devices. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2–5, 2009. Proceedings*, volume 5536 of *Lecture Notes in Computer Science*, pages 519–534, 2009.
- [HT09] Daniel M. Hein and Ronald Toegl. An autonomous attestation token to secure mobile agents in disaster response. In Andreas U. Schmidt and Shiguo Lian, editors, *Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec 2009, Turin, Italy, June 3–5, 2009, Revised Selected Papers*, volume 17 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 46–57. Springer, 2009.
- [HTK10] Daniel M. Hein, Ronald Toegl, and Stefan Kraxberger. An autonomous attestation token to secure mobile agents in disaster response. *Security and Communication Networks*, 3(5):421–438, 2010.
- [HTP⁺12] Daniel M. Hein, Ronald Toegl, Martin Pirker, Emil Gatjal, Zoltán Balogh, Hans Brandl, and Ladislav Hluchý. Securing mobile agents for crisis management support. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing, STC '12*, pages 85–90, New York, NY, USA, 2012. ACM.
- [HWF08] Daniel M. Hein, Johannes Wolkerstorfer, and Norbert Felber. ECC is ready for RFID – A proof in silicon. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14–15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 2008.
- [HWF15] Daniel M. Hein, Johannes Winter, and Andreas Fitzek. Secure block device – secure, flexible, and efficient data storage for ARM

- TrustZone systems. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20–22, 2015, Volume 1*, pages 222–229, 2015.
- [IMIT13] Yang Ishigaki, Yoshinori Matsumoto, Ryo Ichimiya, and Kenji Tanaka. Development of mobile radiation monitoring system utilizing smartphone and its field tests in fukushima. *Sensors Journal, IEEE*, 13(10):3520–3526, October 2013.
- [Jan00] Wayne A. Jansen. Countermeasures for mobile agent security. *Computer Communications*, 23(17):1667–1676, 2000.
- [JK99] Wayne A. Jansen and Tom Karygiannis. NIST special publication 800-19 mobile agent security. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, 20899-8930, October 1999.
- [Joh10] Michael N. Johnstone. Threat modelling with STRIDE and UML. In *Proceedings of the 8th Australian Information Security Management Conference*. School of Computer and Information Science, Edith Cowan University, Perth, Western Australia, 2010.
- [JSW98] Nicholas R. Jennings, Katia P. Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.
- [KDH10] Stefan Kraxberger, Peter Danner, and Daniel M. Hein. Secure multi-agent system for multi-hop environments. In Igor V. Kottenko and Victor A. Skormin, editors, *Computer Network Security, 5th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2010, St. Petersburg, Russia, September 8–10, 2010. Proceedings*, volume 6258 of *Lecture Notes in Computer Science*, pages 270–283. Springer, 2010.
- [KEAR09] Kari Kostiainen, Jan-Erik Ekberg, N. Asokan, and Arne Rantala. On-board credentials with open provisioning. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009, Sydney, Australia, March 10–12, 2009*, pages 104–115. ACM, 2009.
- [KG99] Loren Kohnfelder and Praerit Garg. The threats to our products. Microsoft Interface, April 1999.
- [KLL⁺09] Ivan Kočíš, Oscar Lázaro, Oscar López, Mikel Uriarte, Daniel Hein, Peter Danner, Tomasz Miroslaw, Monika Świech, Wojciech Dymowski, Ladislav Hluchý, Branislav Šimo, Zoltán Balogh, Vladimir

- Hudek, and Aurel Machalek. SECRIком deliverable 2.1, analysis of external and internal system requirements, February 2009.
- [KR14] Ted Krovetz and Phillip Rogaway. The OCB authenticated-encryption algorithm. Internet Research Task Force (IRTF) Request for Comments: 7253, May 2014.
- [Lam09] Butler W. Lampson. Privacy and security – usable security: how to get it. *Commun. ACM*, 52(11):25–27, 2009.
- [LC14] Dongtao Liu and Landon P. Cox. Veriui: attested login for mobile devices. In *15th Workshop on Mobile Computing Systems and Applications, HotMobile '14, Santa Barbara, CA, USA, February 26–27, 2014*, pages 7:1–7:6. ACM, 2014.
- [LHB⁺14] Xiaolei Li, Hong Hu, Guangdong Bai, Yaoqi Jia, Zhenkai Liang, and Prateek Saxena. Droidvault: A trusted data vault for android devices. In *2014 19th International Conference on Engineering of Complex Computer Systems, Tianjin, China, August 4–7, 2014*, pages 29–38. IEEE Computer Society, 2014.
- [LHH⁺15] Christian M. Lesjak, Daniel Hein, Michael Hofmann, Martin Maritsch, Andreas Aldrian, Peter Priller, Thomas Ebner, Thomas Rupprechter, and Günther Pregartner. Securing smart maintenance services: Hardware-security and TLS for MQTT. In *13th IEEE International Conference on Industrial Informatics, INDIN 2015, Cambridge, United Kingdom, July 22–24, 2015*, pages 1243–1250, 2015.
- [LHW15] Christian M. Lesjak, Daniel Hein, and Johannes Winter. Hardware-security technologies for industrial iot: Trustzone and security controller. In *41st Annual Conference of the IEEE Industrial Electronics Society, IECON 2015, Yokohama, Japan, November 2015*.
- [LKMS04] Jinyuan Li, Maxwell N. Krohn, David Mazières, and Dennis Shasha. Secure untrusted data repository (SUNDR). In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6–8, 2004*, pages 121–136, 2004.
- [Low95] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.
- [LTM⁺00] David Lie, Chandramohan A. Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John C. Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In Larry Rudolph and Anoop Gupta, editors, *ASPLOS-IX Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, November 12–15, 2000.*, pages 168–177. ACM Press, 2000.

- [LYG⁺15] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17–21, 2015*, pages 605–622, 2015.
- [MA10] Caroline Moeckel and Ali E. Abdallah. Threat modeling approaches and tools for securing architectural designs of an e-banking application. In *Sixth International Conference on Information Assurance and Security, IAS 2010, Atlanta, GA, USA, August 23–25, 2010*, pages 149–154, 2010.
- [Mer80] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 14–16, 1980*, pages 122–134. IEEE Computer Society, 1980.
- [MF99] John P. McDermott and Chris Fox. Using abuse case models for security requirements analysis. In *15th Annual Computer Security Applications Conference (ACSAC 1999), December 6–10, 1999, Scottsdale, AZ, USA*, pages 55–64, 1999.
- [MLQ⁺10] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil D. Gligor, and Adrian Perrig. Trustvisor: Efficient TCB reduction and attestation. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16–19 May 2010, Berkeley/Oakland, California, USA*, pages 143–158, 2010.
- [MMRM09] Abraham Martín-Campillo, Ramon Martí, Sergi Robles, and Carles Martínez-García. Mobile agents for critical medical information retrieving from the emergency scene. In Yves Demazeau, Juan Pavón, Juan M. Corchado, and Javier Bajo, editors, *7th International Conference on Practical Applications of Agents and Multi-Agent Systems, PAAMS 2009, Salamanca, Spain, March 25–27, 2009*, volume 55 of *Advances in Intelligent and Soft Computing*, pages 30–39. Springer, 2009.
- [MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks – revealing the secrets of smart cards*. Springer, 2007.
- [MP12] Pramita Mitra and Christian Poellabauer. Emergency response in smartphone-based mobile ad-hoc networks. In *Proceedings of IEEE International Conference on Communications, ICC 2012, Ottawa, ON, Canada, June 10–15, 2012*, pages 6091–6095, 2012.
- [MPSvD07] Jonathan M. McCune, Adrian Perrig, Arvind Seshadri, and Leendert van Doorn. Turtles all the way down: Research challenges in user-based attestation. In *2nd USENIX Workshop on Hot Topics in Security, HotSec’07, Boston, MA, USA, August 7, 2007*, 2007.

- [MS01] David Mazières and Dennis Shasha. Don't trust your file server. In *Proceedings of HotOS-VIII: 8th Workshop on Hot Topics in Operating Systems, May 20–23, 2001, Elmau/Oberbayern, Germany*, pages 113–118. IEEE Computer Society, 2001.
- [MSW+04] John Marchesini, Sean W. Smith, Omen Wild, Joshua Stabiner, and Alex Barsamian. Open-source applications of TCPA hardware. In *20th Annual Computer Security Applications Conference (ACSAC 2004), December 6–10, 2004, Tucson, AZ, USA*, pages 294–303, 2004.
- [MTM15] Microsoft threat modeling tool 2016 user guide. Microsoft Download Center, <http://www.microsoft.com/en-us/download/details.aspx?id=49168>, October 2015.
- [NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.
- [OHT14] Alexander Oprisnik, Daniel Hein, and Peter Teuff. Identifying cryptographic functionality in Android applications. In Mohammad S. Obaidat, Andreas Holzinger, and Pierangela Samarati, editors, *SECRYPT 2014 – Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria, August 28–30, 2014*, pages 151–162. SciTePress, 2014.
- [OSZW12] Shaun O'Neill, Jim Strother, Jan Zych, and Wojciech Wojciechowicz. User requirements for mission-critical application – the SECRICOM case. *Technical Sciences*, 15(1)/2012:81–99, 2012.
- [Par08] Bryan Parno. Bootstrapping trust in a “trusted” platform. In Niels Provos, editor, *3rd USENIX Workshop on Hot Topics in Security, HotSec'08, San Jose, CA, USA, July 29, 2008, Proceedings*. USENIX Association, 2008.
- [PEM04] Marcus Peinado, Yuqun Chen, Paul England, and John Manferdelli. NGSCB: A trusted open system. In *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13–15, 2004. Proceedings*, pages 86–97, 2004.
- [Pee05] Johan Peeters. Agile security requirements engineering. In *Symposium on Requirements Engineering for Information Security*, 2005.
- [Pir15] Martin Pirker. *On Attestable and Trusted Services for the Cloud*. PhD thesis, Graz University of Technology, 2015.
- [PJ06] Adam Pridgen and Christine Julien. SMASH: modular security for mobile agents. In *Software Engineering for Multi-Agent Systems V, Research Issues and Practical Applications*, pages 99–116, 2006.

- [PSE⁺14] Viktoria Pammer, Hermann Stern, Stefan Edler, Daniel M. Hein, Martin Pirker, Roderick Bloem, and Alfred Wertner. Security concepts for a distributed architecture for activity logging and analysis. In Stefanie N. Lindstaedt, editor, *14th International Conference on Knowledge Management and Data-driven Business, I-KNOW '14, Graz, Austria, September 16–19, 2014*, pages 23:1–23:6. ACM, 2014.
- [PTHD09] Martin Pirker, Ronald Toegl, Daniel M. Hein, and Peter Danner. A PrivacyCA for anonymity and trust. In Liqun Chen, Chris J. Mitchell, and Andrew Martin, editors, *Trusted Computing, Second International Conference, Trust 2009, Oxford, UK, April 6–8, 2009, Proceedings*, volume 5471 of *Lecture Notes in Computer Science*, pages 101–119. Springer, 2009.
- [PWH13a] Klaus Potzmader, Johannes Winter, and Daniel Hein. STUNT: A simple, transparent, user-centered network of trust. In Sokratis K. Katsikas and Isaac Agudo, editors, *Public Key Infrastructures, Services and Applications – 10th European Workshop, EuroPKI 2013, Egham, UK, September 12–13, 2013, Revised Selected Papers*, volume 8341 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2013.
- [PWH⁺13b] Klaus Potzmader, Johannes Winter, Daniel Hein, Christian Hanser, Peter Teufl, and Liqun Chen. Group signatures on mobile devices: Practical experiences. In Michael Huth, N. Asokan, Srdjan Capkun, Ivan Flechais, and Lizzie Coles-Kemp, editors, *Trust and Trustworthy Computing – 6th International Conference, TRUST 2013, London, UK, June 17–19, 2013. Proceedings*, volume 7904 of *Lecture Notes in Computer Science*, pages 47–64. Springer, 2013.
- [RBB03] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.
- [RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6–8, 2001.*, pages 196–205, 2001.
- [RCPS07] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. Using address independent seed encryption and bonsai merkle trees to make secure processors OS- and performance-friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40 2007), December 1–5, 2007, Chicago, Illinois, USA*, pages 183–196. IEEE Computer Society, 2007.

- [RG85] Jeffrey S. Rosenschein and Michael R. Genesereth. Deals among rational agents. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence. Los Angeles, CA, August 1985*, pages 91–99, 1985.
- [Rog11] Phillip Rogaway. Evaluation of some blockcipher modes of operation. <http://www.cs.ucdavis.edu/~rogaway>, February 2011.
- [Rot04] Volker Roth. Obstacles to the adoption of mobile agents. In *5th IEEE International Conference on Mobile Data Management (MDM 2004), January 19–22, 2004, Berkeley, CA, USA*, pages 296–297. IEEE Computer Society, 2004.
- [RPS06] Brian Rogers, Milos Prvulovic, and Yan Solihin. Efficient data protection for distributed shared memory multiprocessors. In Erik R. Altman, Kevin Skadron, and Benjamin G. Zorn, editors, *15th International Conference on Parallel Architecture and Compilation Techniques (PACT 2006), Seattle, Washington, USA, September 16–20, 2006*, pages 84–94. ACM, 2006.
- [RRM⁺08] Sarvapali D. Ramchurn, Alex Rogers, Kathryn S. Macarthur, Alessandro Farinelli, Perukrishnen Vytelingum, Ioannis A. Vetsikas, and Nicholas R. Jennings. Agent-based coordination technologies in disaster management. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12–16, 2008, Demo Proceedings*, pages 1651–1652, 2008.
- [RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28–June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.
- [SB12] Andreas Schaad and Mike Borozdin. Tam²: automated threat analysis. In *Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26–30, 2012*, pages 1103–1108, 2012.
- [ŠBH⁺09] Branislav Šimo, Zoltán Balogh, Daniel Hein, Peter Danner, Oscar López, Mikel Uriarte, and Vladimír Hudek. SECRIком deliverable 4.1, security requirements and specification for docking station module. <http://www.secricom.eu/public-deliverables>, April 2009.
- [Sch99] Bruce Schneier. Attack trees. *Dr. Dobb’s journal*, 24(12):21–29, 1999.

- [Sea05] Robert Seacord. *Secure Coding in C and C++*. Addison-Wesley Professional, first edition, 2005.
- [SEC07] SECRIKOM Consortium. Part b: Description of work, SECRIKOM seamless communication for crisis management, collaborative project, integration project, sec-2007-4.2-04. Call FP7-SEC-2007-1, Theme 10: Security, 2007.
- [Sho08] Adam Shostack. Experiences threat modeling at Microsoft. In *Modeling Security Workshop. Dept. of Computing, Lancaster University, UK*, 2008.
- [Sho14] Adam Shostack. *Threat Modeling: Designing for Security*. John Wiley & Sons, Inc., April 2014.
- [SMS⁺09] Joshua Schiffman, Thomas Moyer, Christopher Shal, Trent Jaeger, and Patrick Drew McDaniel. Justifying integrity using a virtual machine verifier. In *Twenty-Fifth Annual Computer Security Applications Conference, ACSAC 2009, Honolulu, Hawaii, December 7–11, 2009*, pages 83–92, 2009.
- [SMT⁺05] Nathan Schurr, Janusz Marecki, Milind Tambe, Paul Scerri, Nikhil Kasinadhuni, and John P. Lewis. The future of disaster response: Humans working with multiagent teams using DEFACTO. In *AI Technologies for Homeland Security, Papers from the 2005 AAAI Spring Symposium, Technical Report SS-05-01, Stanford, California, USA, March 21–23, 2005*, pages 9–16. AAAI, 2005.
- [SO05] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, 2005.
- [SOD07] G. Edward Suh, Charles W. O'Donnell, and Srinivas Devadas. Aegis: A single-chip secure processor. *IEEE Design & Test of Computers*, 24(6):570–580, 2007.
- [SP13] Raphael Spreitzer and Thomas Plos. On the applicability of time-driven cache attacks on mobile devices (extended version). *IACR Cryptology ePrint Archive*, 2013:172, 2013.
- [SPA⁺06] S. Sadik, M. Pasha, A. Ali, H.F. Ahmad, and H. Suguri. Policy based migration of mobile agents in disaster management systems. In *Emerging Technologies, 2006. ICET '06. International Conference on*, pages 224–229, November 2006.
- [SPJ⁺03] Paul Scerri, David V. Pynadath, W. Lewis Johnson, Paul S. Rosenbloom, Mei Si, Nathan Schurr, and Milind Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14–18, 2003, Melbourne, Victoria, Australia, Proceedings*, pages 433–440. ACM, 2003.

- [SPLI06] JunHyuk Song, Radha Poovendran, Jicheol Lee, and Tetsu Iwata. The AES-CMAC algorithm. RFC 4493 (Informational), June 2006.
- [SRSW14] Nuno Santos, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Using ARM TrustZone to build a trusted language runtime for mobile applications. In Rajeev Balasubramonian, Al Davis, and Sarita V. Adve, editors, *Architectural Support for Programming Languages and Operating Systems, ASPLOS '14, Salt Lake City, UT, USA, March 1–5, 2014*, pages 67–80. ACM, 2014.
- [SS04] Frank Swiderski and Window Snyder. *Threat Modeling*. Microsoft Press, Redmond, WA, USA, 2004.
- [SS13] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4–8, 2013*, pages 247–258. ACM, 2013.
- [SSSW98] Chris Slater, O. Sami Saydjari, Bruce Schneier, and Jim Wallner. Toward a secure system engineering methodology. In *Proceedings of the 1998 Workshop on New Security Paradigms, Charlottesville, VA, USA, September 22–25, 1998*, pages 2–10, 1998.
- [SvDJO12] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: a scalable cloud file system with efficient integrity checks. In Robert H. Zakon, editor, *28th Annual Computer Security Applications Conference, ACSAC 2012, Orlando, FL, USA, December 3–7, 2012*, pages 229–238. ACM, 2012.
- [SW08] Zhi-Dong Shen and Xiaoping Wu. A trusted computing technology enabled mobile agent system. In *International Conference on Computer Science and Software Engineering, CSSE 2008, Volume 3: Grid Computing / Distributed and Parallel Computing / Information Security, December 12–14, 2008, Wuhan, China*, pages 567–570, 2008.
- [SWJ15] Riccardo Scandariato, Kim Wuyts, and Wouter Joosen. A descriptive study of Microsoft’s threat modeling technique. *Requir. Eng.*, 20(2):163–180, 2015.
- [SZJvD04] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th USENIX Security Symposium, August 9–13, 2004, San Diego, CA, USA*, pages 223–238, 2004.
- [TFF⁺16] Peter Teufl, Michaela Ferk, Andreas Fitzek, Daniel M. Hein, Stefan Kraxberger, and Clemens Orthacker. Malware detection by applying knowledge discovery processes to application metadata on the

- Android Market (Google Play). *Security and Communication Networks*, 9(5):389–419, 2016.
- [TFH⁺14] Peter Teufl, Andreas Fitzek, Daniel Hein, Alexander Marsalek, Alexander Oprisnik, and Thomas Zefferer. Android encryption systems. In *2014 International Conference on Privacy and Security in Mobile Systems, PRISMS 2014, Aalborg, Denmark, May 11–14, 2014*, pages 1–8. IEEE, 2014.
- [TJM08] Inger A. Tøndel, Martin G. Jaatun, and Per H. Meland. Security requirements for the rest of us: A survey. *IEEE Software*, 25(1):20–27, 2008.
- [Tor05] Peter Torr. Demystifying the threat-modeling process. *IEEE Security & Privacy*, 3(5):66–70, 2005.
- [Tru07a] Trusted Computing Group. TCG software stack (TSS) specification. http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification, 2007.
- [Tru07b] Trusted Computing Group. TCG TPM specification version 1.2 revision 103, 2007.
- [TWD⁺10] Chris Thompson, Jules White, Brian Dougherty, Adam Albright, and Douglas C. Schmidt. Using smartphones to detect car accidents and provide situational awareness to emergency responders. In *Mobile Wireless Middleware, Operating Systems, and Applications - Third International Conference, Mobilware 2010, Chicago, IL, USA, June 30–July 2, 2010. Revised Selected Papers*, pages 29–42, 2010.
- [TZW⁺14] Peter Teufl, Thomas Zefferer, Christoph Wörgötter, Alexander Oprisnik, and Daniel Hein. Android – on-device detection of SMS catchers and sniffers. In *2014 International Conference on Privacy and Security in Mobile Systems, PRISMS 2014, Aalborg, Denmark, May 11–14, 2014*, pages 1–8. IEEE, 2014.
- [UTG08] Marco Ughetti, Tiziana Trucco, and Danilo Gotta. Development of agent-based, peer-to-peer mobile applications on Android with JADE. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBIComm '08. The Second International Conference on*, pages 287–294, September 2008.
- [VPQ⁺12] Amit Vasudevan, Bryan Parno, Ning Qu, Virgil D. Gligor, and Adrian Perrig. Lockdown: Towards a safe and practical architecture for security applications on commodity platforms. In *Trust and Trustworthy Computing – 5th International Conference, TRUST 2012, Vienna, Austria, June 13–15, 2012. Proceedings*, pages 34–54, 2012.

- [WD13] Johannes Winter and Kurt Dietrich. A hijacker's guide to communication interfaces of the trusted platform module. *Computers & Mathematics with Applications*, 65(5):748–761, 2013.
- [WFK⁺12] Wojciech Wojciechowicz, Jacques Fournier, Mirosław Konecny, Stefan Vanya, John Stoodley, Phil Entwisle, Daniel M. Hein, Aurel Machalek, Apostolos P. Fournaris, Mikel Uriarte, Oscar Lopez, Shaun O'Neill, Hans Brandl, Zoltán Balogh, Emil Gatial, Ladislav Hluchý, Tomasz Mirosław, and Jan Zych. Seamless communication for crisis management. *Technical Sciences*, 15(1)/2012:65–79, 2012.
- [WFM⁺07] Peter Wilson, Alexandre Frey, Tom Mihm, Danny Kershaw, and Tiago Alves. Implementing embedded security on dual-virtual-cpu systems. *IEEE Design & Test of Computers*, 24(6):582–591, 2007.
- [WH08] Carsten Weinhold and Hermann Härtig. VPFS: building a virtual private file system with a small trusted computing base. In Joseph S. Sventek and Steven Hand, editors, *Proceedings of the 2008 EuroSys Conference, Glasgow, Scotland, UK, April 1–4, 2008*, pages 81–93. ACM, 2008.
- [Whi94] James E. White. Telescript technology: The foundation for the electronic marketplace. White paper, 1994. General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040.
- [Win14] Johannes Winter. Trusted computing and local hardware attacks. Master's thesis, Graz University of Technology, 2014.
- [WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *Knowledge Eng. Review*, 10(2):115–152, 1995.
- [WR09] Rafal Wojtczuk and Joanna Rutkowska. Attacking Intel trusted execution technology. Invisible Things Lab, 2009.
- [WR11] Rafal Wojtczuk and Joanna Rutkowska. Attacking Intel TXT via SINIT code execution hijacking. Invisible Things Lab, 2011.
- [WRT09] Rafal Wojtczuk, Joanna Rutkowska, and Alexander Tereshkin. Another way to circumvent Intel trusted execution technology. Invisible Things Lab, 2009.
- [WSB99] Uwe G. Wilhelm, Sebastian Staamann, and Levente Buttyán. Introducing trusted third parties to the mobile agent paradigm. In *Secure Internet Programming, Security Issues for Mobile and Distributed Objects*, pages 469–489, 1999.
- [WSZ06] Xiaoping Wu, Zhidong Shen, and Huanguo Zhang. The mobile agent security enhanced by trusted computing technology. *International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2006)*, pages 1–4, September 2006.

- [WWPT11] Johannes Winter, Paul Wiecele, Martin Pirker, and Ronald Toegl. A flexible software development and emulation framework for ARM TrustZone. In *Trusted Systems – Third International Conference, INTRUST 2011, Beijing, China, November 27–29, 2011, Revised Selected Papers*, pages 1–15, 2011.
- [WY15] Imano Williams and Xiaohong Yuan. Evaluating the effectiveness of Microsoft threat modeling tool. In *Proceedings of the 2015 Information Security Curriculum Development Conference, InfoSecCD 2015, Kennesaw, GA, USA, October 10, 2015*, page 9, 2015.
- [YCZD13] Hsin-Jung Yang, Victor Costan, Nikolai Zeldovich, and Srinivas Devadas. Authenticated storage using small trusted hardware. In Ari Juels and Bryan Parno, editors, *CCSW'13, Proceedings of the 2013 ACM Cloud Computing Security Workshop, Co-located with CCS 2013, Berlin, Germany, November 4, 2013*, pages 35–46. ACM, 2013.
- [YSK09] Aaram Yun, Chunhui Shi, and Yongdae Kim. On protecting integrity and confidentiality of cryptographic file system for out-sourced storage. In Radu Sion and Dawn Song, editors, *Proceedings of the first ACM Cloud Computing Security Workshop, CCSW 2009, Chicago, IL, USA, November 13, 2009*, pages 67–76. ACM, 2009.
- [ZRS⁺14] Dominik Ziegler, Mattias Rauter, Christof Stromberger, Peter Teuffl, and Daniel Hein. Do you think your passwords are secure? In *2014 International Conference on Privacy and Security in Mobile Systems, PRISMS 2014, Aalborg, Denmark, May 11–14, 2014*, pages 1–8. IEEE, 2014.

Index

- agent, 17
 - autonomy, 18
 - benevolence, 19
 - flexibility, 18
 - mobility, 18
 - multi-hop, 20
 - single-hop, 20
 - strong, 20
 - weak, 20
 - notion of agency, 18
 - platform, 19
 - pro-activeness, 18
 - rationality, 19
 - reactivity, 18
 - responsiveness, 18
 - situatedness, 18
 - social ability, 18
 - veracity, 18
- ANDIX OS, 74
- ARM TrustZone
 - see TrustZone, 72
- authenticated encryption, 53
- authentication, 40
- availability, 40
- CMAC, 53
- confidentiality, 39
- data confidentiality, 39
- Data Flow Diagrams, 43
- data integrity, 40
- denial of service, 42
- Distributed Secure Agent Platform
 - see DSAP, 31
- DSAP, 31, 35
 - Service, 132, 133
 - Virtual Appliance, 132
- elevation of privilege, 43
- hash function, 51
- hash tree, *see* Merkle Tree
- information disclosure, 42
- integrity, 40
- Jini, 26
 - lookup service, 27
 - service, 27
 - service provider, 27
- memory authenticity, 56
 - attacks
 - forking, 59
 - replay, 57
 - splicing, 57
 - spoofing, 57
- Merkle Tree, 62
- message authentication code, 53
- Microsoft threat modeling tool 2016, 44
- Mobile Agent System, 19
- Multi-Agent System, 19
- OCB, 54
- origin integrity, 40
- PCR, 65
- Platform Configuration Register
 - see PCR, 65
- Process Management System, 30
- repudiation, 42
- SBD, 153
- SDM, 137
 - Communication Library, 134
 - Emulator, 134
 - Manager, 133
 - Security Controller, 142

UI, 134
Secure Agent Infrastructure, 30
Secure Block Device
 see SBD, 153
Secure Docking Module
 see SDM, 137
security controller, 63
SHA-1, 52
SHA-256, 52
Single-User Block Datastore, 56
situational awareness, 2
SIV, 55
spoofing, 41
STRIDE, 45
STRIDE-per-element, 45
STRIDE-per-interaction, 46

tampering, 42
TDS, 135
TPM, *see* Trusted Platform Module
Trusted Docking Station
 see TDS, 135
Trusted Platform Module, 64
TrustZone, 72

Author Index

- Abdallah, Ali E. 79
Achleitner, Florian 74, 251
SDL Team 43, 45
Ahmad, H.F. 4, 84
Ahmad, Shabir 83
Albright, Adam 5, 154
Aldrian, Andreas 251
Alexander, Ian F. 83
Ali, A. 4, 84
Alves, Tiago 73
Anderson, Ross 64
Apache Foundation xvii, 27–30
Apecechea, Gorka Irazoqui 156
Asokan, N. 87, 88, 155
- Bai, Guangdong 87, 155
Balasubramanian, Vidhya 84
Balfe, Shane 85
Balogh, Zoltán 64, 80, 128, 249, 250,
252–254
Barsamian, Alex 86
Bellare, Mihir 53, 54, 163
Bertino, Elisa 79
Bierman, Elmarie 5, 20, 24, 93, 113,
127, 188, 193
Bishop, Matthew A. 39–42
Black, John 54, 163
Bloem, Roderick 153, 164, 251, 254,
255
Blum, Manuel 63
Bond, Mike 64
Boneh, Dan 87, 88
Borozdin, Mike 81, 92
Borselius, N. 5, 18, 20–22, 37, 93, 113,
127, 188, 193
- Brandl, Hans 64, 80, 128, 250,
252–254
Bruschi, Danilo 79
Buttyán, Levente 85
- Carzaniga, Antonio 20
Chadwick, David 79
Champagne, David 55–57, 87
Chan, Victor 5, 154
Chen, Liqun 252
Chen, Yuqun 73
Cheng, Harry H. 84
Chhabra, Siddhartha 87, 88, 180
Cloete, Elsabe 5, 20, 24, 93, 113, 127,
188, 193
Clulow, Jolyon 64
Cock, Danny De 79
Cohen, Paul R. 4, 84
Coker, George 86
Coronado, Carlos 62
Costan, Victor 87–89
Cox, Landon P. 74, 155
- Danner, Peter 5, 38, 136, 139, 154,
250, 252, 253
Datta, Anupam 86
Desmet, Lieven 79
Devadas, Srinivas 87–89
Dietrich, Kurt 71
Djordjevich, Donna 84
Dougherty, Brian 5, 154
Duc, Guillaume 87, 88
Durfee, Edmund H. 19
Dworkin, Morris J. 53
Dymowski, Wojciech 38

- Ebner, Thomas 251
 Edler, Stefan 251
 Eichstädt, Henrik 79
 Eisenbarth, Thomas 156
 Ekberg, Jan-Erik 87, 88, 155
 Elbaz, Reouven 55–57, 87
 England, Paul 73
 Entwisle, Phil 250
 Evans, William S. 63

 Farinelli, Alessandro 84
 Felber, Norbert 253
 Ferguson, Niels 55
 Ferk, Michaela 249
 Fitzek, Andreas 74, 80, 153, 249, 251, 254
 Fournaris, Apostolos P. 38, 144, 250, 252
 Fournier, Jacques 250, 252
 Fovino, Igor Nai 79
 Fox, Chris 83
 Franzoni, Stefano 79
 Frey, Alexandre 73

 Gallery, Eimear 85
 Galliers, Julia Rose 18, 19
 Gammel, Berndt M. 64
 Garg, Praerit 43, 45
 Gatial, Emil xvii, xviii, 1, 12, 30, 32, 33, 36–38, 64, 80, 128, 249, 250, 252–254
 Ge, Qian 156
 Gebotys, Catherine H. 55–57, 87
 Gemmell, Peter 63
 Genesereth, Michael R. 19
 Gligor, Virgil D. 86
 GlobalPlatform Inc. 73–75
 Goh, Eu-Jin 87, 88
 Gotta, Danilo 5, 154
 Grawrock, David 67–71
 Greenberg, Michael L. 4, 84
 Grimm, Rüdiger 79
 Gruss, Daniel 156
 Guttman, Joshua D. 86

 Haley, Charles B. 83, 110, 124

 Hanser, Christian 252
 Hart, David M. 4, 84
 Härtig, Hermann 87, 88
 Hein, Daniel 38, 63, 74, 250–252
 Hein, Daniel M. 5, 38, 64, 80, 128, 136, 139, 144, 153, 154, 164, 249–255
 Heiser, Gernot 156
 Herzog, Amy L. 86
 Hluchý, Ladislav xviii, 1, 12, 30, 32, 36, 38, 249, 250
 Hofmann, Michael 251
 Honda, Jason 84
 Horowitz, Mark 87, 88
 Howard, Michael 43, 82
 Howe, Adele E. 4, 84
 Hu, Hong 87, 155
 Hudek, Vladimir 38
 Hudek, Vlasta 12, 30, 32
 Hussain, Shafiq 83
 Hutter, Michael 252

 Ichimiya, Ryo 5, 154
 Iqbal, Sajid 83
 Ishigaki, Yang 5, 154
 Iwata, Tetsu 53

 Jaatun, Martin G. 83
 Jacobs, Bart 79
 Jaeger, Trent 86
 Jansen, Wayne A. xvii, 4, 5, 20–23, 85, 93, 113, 127, 188, 193
 Jennings, Nicholas R. 17–19, 22, 84
 Jia, Yaoqi 87, 155
 Johnson, W. Lewis 4, 84
 Johnstone, Michael N. 81
 Joosen, Wouter 46, 79, 82
 Juels, Ari 87, 88
 Julien, Christine 85

 Kamal, Asif 83
 Kannan, Sampath 63
 Karygiannis, Tom xvii, 4, 5, 20–23, 93, 113, 127, 188, 193
 Kasinadhuni, Nikhil 4, 84
 Kershaw, Danny 73

- Keryell, Ronan 87, 88
Kim, Yongdae 87, 88, 169
Kočiš, Ivan 38
Kohnfelder, Loren 43, 45
Konecný, Michal 12, 30, 32
Konecny, Mirosław 250
Kostiainen, Kari 87, 88, 155
Kraxberger, Stefan 5, 80, 128, 136,
139, 144, 154, 249, 250, 252–254
Krohn, Maxwell N. 55
Krovetz, Ted 54, 55
- Lampson, Butler W. 73
Laney, Robin C. 83, 110, 124
Lázaro, Oscar 38
Leblanc, David E. 43
Lee, Jicheol 53
Lee, Ruby B. 55–57, 87, 156
Lesjak, Christian M. 63, 250, 251
Lesser, Victor R. 19
Lewis, John P. 4, 84
Li, Jinyuan 55
Li, Xiaolei 87, 155
Li, Yanlin 86
Liang, Zhenkai 87, 155
Lie, David 87, 88
Liebl, Martin 38
Lincoln, Patrick 87, 88
Lipner, Steve 82
Liu, Dongtao 74, 155
Liu, Fangfei 156
Lopez, Oscar 250
Loscocco, Peter 86
Lowe, Gavin 139
- Macarthur, Kathryn S. 84
Machalek, Aurel 38, 250
Manferdelli, John 73
Mangard, Stefan 63, 156
Marchesini, John 86
Marecki, Janusz 4, 84
Maritsch, Martin 251
Marsalek, Alexander 251
Martí, Ramon 4
Martín-Campillo, Abraham 4
Martínez-García, Carles 4
- Massaguer, Daniel 84
Matsumoto, Yoshinori 5, 154
Mazières, David 55, 57, 87, 88
McCune, Jonathan M. 86
McDaniel, Patrick Drew 86
McDermott, John P. 83
Medwed, Marcel 252
Mehrotra, Sharad 84
Meland, Per H. 83
Merkle, Ralph C. 62
Mihm, Tom 73
Millen, Jonathan K. 86
Mirosław, Tomasz 250
Mitchell, John C. 87, 88
Mitchell, Mark 87, 88
Mitra, Pramita 5, 154
Modadugu, Nagendra 87, 88
Moeckel, Caroline 79
Moffett, Jonathan D. 83, 110, 124
Moyer, Thomas 86
- Nagarajan, Aarthi 85
Namprempre, Chanathip 53
Naor, Moni 63
Needham, Roger M. 139
Nuseibeh, Bashar 83, 110, 124
- O'Donnell, Charles W. 87, 88
O'Hanlon, Brian 86
O'Neill, Shaun xvii, 1–3, 6, 9, 91–93,
96, 99, 100, 104, 105, 107, 115, 119,
123, 154, 250
Opdahl, Andreas L. 83
Oprea, Alina 87, 88
Oprisnik, Alexander 251
Orthacker, Clemens 249
Oswald, Elisabeth 63
- Pammer, Viktoria 251
Parameswaran, Nandan 5, 154
Parno, Bryan 86
Pasha, M. 4, 84
Peeters, Johan 83
Peinado, Marcus 73
Perrig, Adrian 86
Picco, Gian Pietro 20

- Piessens, Frank 79
Pirker, Martin iv, 12, 64, 72, 74, 80, 86, 128, 134, 137, 151, 183, 249–254
Plos, Thomas 156
Poellabauer, Christian 5, 154
Poovendran, Radha 53
Popp, Thomas 63
Potlapally, Nachiketh R. 55–57, 87
Potzmader, Klaus 252
Pregartner, Günther 251
Preneel, Bart 79
Pridgen, Adam 85
Priller, Peter 251
Prvulovic, Milos 87, 88, 180
Pynadath, David V. 4, 84
- Qu, Ning 86
- Raj, Himanshu 88
Ramchurn, Sarvapali D. 84
Ramsdell, John D. 86
Rantala, Aarne 87, 88, 155
Rasool, Ghulam 83
Rauter, Mattias 252
Ray, Pradeep 5, 154
Reymond, Guillaume 250, 252
Robles, Sergi 4
Röck, Franz 153, 164, 251, 254, 255
Rogaway, Phillip 54, 55, 152, 163, 164
Rogers, Alex 84
Rogers, Brian 87, 88, 180
Rosenbloom, Paul S. 4, 84
Rosenschein, Jeffrey S. 19
Roth, Volker 1, 4, 20
Rüping, Stefan J. 64
Rupprechter, Thomas 251
Rutkowska, Joanna 71
- Sadik, S. 4, 84
Sailer, Reiner 86
Santos, Nuno 88
Saroiu, Stefan 88
Saxena, Prateek 87, 155
Saydjari, O. Sami 124
Scandariato, Riccardo 46, 82
Scerri, Paul 4, 84
- Schaad, Andreas 81, 92
Schellekens, Dries 79
Schiffman, Joshua 86
Schmidt, Douglas C. 5, 154
Schneier, Bruce 84, 124
Schroeder, Michael D. 139
Schumi, Richard 153, 164, 251, 254, 255
Schurr, Nathan 4, 84
Seacord, Robert 41
SECRICOM Consortium 38
Segall, Ariel 86
Seshadri, Arvind 86
Shacham, Hovav 87, 88
Shal, Christopher 86
Shasha, Dennis 55, 57, 87, 88
Sheehy, Justin 86
Shen, Zhi-Dong 85
Shen, Zhidong 85
Shi, Chunhui 87, 88, 169
Shi, Elaine 87, 88
Shostack, Adam xiii, 41, 43–48, 123
Shrimpton, Thomas 55, 164
Si, Mei 4, 84
Šimo, Branislav 12, 30, 38
Sindre, Guttorm 83
Singelée, Dave 79
Skorobogatov, Sergei 64
Slater, Chris 124
Smith, Sean W. 86
Sniffen, Brian T. 86
Snyder, Window 84
Solihin, Yan 87, 88, 180
Song, JunHyuk 53
Spreitzer, Raphael 156
Staamann, Sebastian 85
Stabiner, Joshua 86
Stefanov, Emil 87, 88
Stern, Hermann 251
Stoodley, John 250
Stromberger, Christof 252
Strother, Jim xvii, 1–3, 6, 9, 91–93, 96, 99, 100, 104, 105, 107, 115, 119, 123, 154
Suguri, H. 4, 84

- Suh, G. Edward 87, 88
Sunar, Berk 156
Swiderski, Frank 84
Świech, Monika 38
Sycara, Katia P. 17–19, 22
- Tambe, Milind 4, 84
Tanaka, Kenji 5, 154
Tereshkin, Alexander 71
Teufel, Peter 249, 251, 252
Thekkath, Chandramohan A. 87, 88
Thompson, Chris 5, 154
Toegl, Ronald 38, 64, 74, 80, 128, 144, 249, 250, 252–254
Tøndel, Inger A. 83
Torr, Peter 92, 110
Torres, Lionel 55–57, 87
Trucco, Tiziana 5, 154
Trusted Computing Group 64, 135
- Ughetti, Marco 5, 154
Uriarte, Mikel 38, 250
- Valtolina, Stefano 79
van Dijk, Marten 87, 88
van Doorn, Leendert 86
Vanya, Stefan 250
Varadharajan, Vijay 85
Vasudevan, Amit 86
Venkatasubramanian, Nalini 84
Vetsikas, Ioannis A. 84
Vigna, Giovanni 20
Vytelingum, Perukrishnen 84
- Wallner, Jim 124
Weinhold, Carsten 87, 88
- Wertner, Alfred 251
White, James E. 18, 20
White, Jules 5, 154
Wiegele, Paul 74
Wild, Omen 86
Wilhelm, Uwe G. 85
Williams, Imano 48, 82
Wilson, Peter 73
Winter, Johannes 63, 71, 74, 80, 153, 250–252, 254
Wojciechowicz, Wojciech xvii, 1–3, 6, 9, 91–93, 96, 99, 100, 104, 105, 107, 115, 119, 123, 154, 250
Wojtczuk, Rafal 71
Wolkerstorfer, Johannes 252, 253
Wolman, Alec 88
Wooldridge, Michael 17–19, 22
Wörgötter, Christoph 251
Wouters, Karel 79
Wu, Xiaoping 85
Wuyts, Kim 46, 82
- Yang, Hsin-Jung 87–89
Yarom, Yuval 156
Yuan, Xiaohong 48, 82
Yun, Aaram 87, 88, 169
- Zefferer, Thomas 251
Zeldovich, Nickolai 87–89
Zhang, Huanguo 85
Zhang, Xiaolan 86
Zhou, Zongwei 86
Ziegler, Dominik 252
Zych, Jan xvii, 1–3, 6, 9, 91–93, 96, 99, 100, 104, 105, 107, 115, 119, 123, 154, 250

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am

.....

(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

.....

date

.....

(signature)