Roland Urbano, BSc

Design and Implementation of Advanced Honeynet Use Cases in an Enterprise Environment

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institute for Technical Informatics

Advisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger
Raphael Otto (Infineon Technologies AG)

Graz, September 2017

Affidavit

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 indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Graz, _____ _____

Date Signature

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 entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Graz, am _____ _____

Datum Unterschrift

Kurzfassung

Über die Jahre hinweg haben sich die unterschiedlichsten Gefahren im Internet verbreitet, um sowohl Firmennetzwerke als auch private Nutzer anzugreifen. Diese Gefahren können in Form von Schadsoftware, Denial-of-Service Angriffen und sogar in persistenter Form auftreten. Während Angreifer neue Wege suchen um in Computer Netzwerke einzudringen, entwickeln Sicherheitsexperten Werkzeuge um deren Angriffe zu erkennen und zu verhindern. Ein solches Werkzeug ist ein Honeynet, ein Netzwerk speziell ausgerichtet um Angreifer anzulocken und mehr über deren Taktiken, Werkzeuge und Vorgehensweisen zu lernen. In dieser Arbeit wird der aktuelle Stand in Bezug auf Honeynets evaluiert und fortgeschrittene Fallstudien, basierend auf existierenden Lösungen, werden beschrieben. In diesem Zusammhang werden Vorlagen erstellt um ein solches Honeynet zu bauen. Mit der praktischen Realisierung zweier dieser Vorlagen wird deren verbesserte Anwendbarkeit und deren Vorteil gegenüber traditionellen Honeypots aufgezeigt.

Abstract

Over the years the Internet has seen a broad variety of attacks targeting both, commercial networks and customers using it for day-to-day business. These threats can occur as malware, denial-of-service attacks, and even in a persistent form. As attackers find new ways to compromise computer systems, the security community develops tools to detect and mitigate these attacks. One such a method is a honeynet, a network dedicated to attract hackers and to learn about their tactics, tools and procedures by monitoring every interaction. In this work the current state-of-the-art in honeynet systems is evaluated and advanced honeynet deployment blueprints are designed, based on use cases for enterprise environments. By implementing two of these blueprints, their applicability, and advantages over traditional honeypots are outlined.

Danksagung

Diese Diplomarbeit wurde 2017 in Zusammenarbeit mit Infineon Technologies Austria AG in Villach durchgeführt.

Ich danke Raphael Otto, meinem technischen und fachlichen Betreuer bei Infineon für die Unterstützung bei der Arbeit und den ständigen Input an Informationen und Ideen, sowie die stundenlangen Diskussionen zu diversen Design- und Umsetzungsentscheidungen. Außerdem danke ich Ass.Prof. Dipl.-Ing Dr.techn. Christian Steger für die organisatorische und persönliche Betreuung auf universitärer Seite.

Ich danke meiner Lebensgefährtin Heike und meiner Tochter Lea für die Geduld und Unterstützung während der intensiven Schreibphase und den wöchentlichen Aufenthalten in Villach.

Graz, August 2017 Roland Urbano

Contents

1 Introduction **8**

2 Background **11**
2.1 Definitions . . . 12
2.1.1 Honeypot . . . 12
2.1.2 Honeynet . . . 14
2.1.3 Honeytoken . . . 14
2.2 Considerations . . . 14
2.3 History . . . 16
2.4 Related Work . . . 17
2.4.1 Analysis studies of honeypot software . . . 17
2.4.2 Honeynet implementations . . . 20

3 Evaluation of Existing Honeynet Implementations **25**
3.1 CyberTrap . . . 26
3.2 Modern Honey Network . . . 28
3.3 T-Pot . . . 30
3.4 SURFcert IDS . . . 31
3.5 Honeywall . . . 33
3.6 Honeybrid . . . 34

4 Design **36**
4.1 IT Use cases . . . 37
4.1.1 Web Application Honeynet . . . 37
4.1.2 Darkspace . . . 39
4.1.3 Ransomware Detection . . . 40
4.1.4 APT Detection . . . 42
4.1.5 APT Monitoring . . . 44
4.2 OT Use cases . . . 45
4.2.1 SCADA/ICS Honeynet . . . 46
4.2.2 Passive Field-Bus Monitoring . . . 47
4.3 Honeynets & Use cases . . . 49
4.3.1 IT Use cases . . . 49
4.3.2 OT Use cases . . . 50
4.3.3 Design Conclusion . . . 51

5 Implementation **52**
5.1 Modern Honey Network . . . 52
5.1.1 Advantages - Disadvantages . . . 54
5.2 Development Environment . . . 55
5.3 Contribution . . . 57
5.3.1 Ransomware Detection . . . 57
5.3.2 Advanced Persistent Threat (APT) Detection . . . 61
5.4 Experimental Results . . . 63
5.4.1 Ransomware Detection Honeypot . . . 63
5.4.2 APT Detection Honeypot . . . 64
5.5 Experiences gained . . . 65
5.5.1 Use Case . . . 66
5.5.2 Design . . . 66
5.5.3 Implementation . . . 67
5.5.4 Analysis . . . 69
5.6 Identified Threats . . . 69

6 Conclusion and Future Work **70**
6.1 Future Work . . . 70

A Source Code **72**
A.1 Ransomware Detection Honeypot . . . 72
A.2 APT Detection Honeypot . . . 76

Literaturverzeichnis **81**

List of Figures

2.1 Attacks per destination port as observed by Moore et al [MA15] 18
2.2 Schematic network layout used by Sochor et al [SZ15] 19
2.3 HoneyLabs's architecture consisting of the HLC and the sensor nodes [Chi+09] 21
2.4 HoneyBrid is combined with Netflow analysis and a threat data aggregation unit [Ber09] 22
2.5 HoneyBrid's internal architecture consisting of a gateway, a set of low interaction honeypots and a set of high interaction honeypots [Ber09] 23
2.6 An example setup of a honeywall with Honeywall CDROM and a local honeynet consisting of three honeypots [Pro05] 24

3.1 MHN modular overview showing the different parts [Anob] 28
3.2 T-Pot by Deutsche Telekom's Security Team 30
3.3 SURFcert IDS consists of a internal network and the distributed clients . . 32
3.4 Honeybrid's architecture overview showing the interconnected modules . . . 34

4.1 Network topology scheme for the web application honeynet use case 38
4.2 Network topology scheme for the ransomware and apt detection usecases . . 43
4.3 Industrial Control System (ICS) network toplology scheme with an active SCADA honeypot 46
4.4 Network topology scheme for a passive ICS monitor sensor 48

5.1 Modern Honey Network (MHN)'s dashboard with the individual tabs on top. 54
5.2 Network view of the honeynet setup 56
5.3 MHN's attack report page showing events of unauthorized access to the honeypot fileshare 64
5.4 MHN's attack report page listing captured pass-the-hash authentication requests with a *honeyhash* from user *user*1 65

List of Tables

3.1 Top- and sub-level categories 26
3.2 Discrete selection criteria listed for each top- and sub-level category 27

4.1 Blueprint section description 37
4.2 Recommended Honeypots to cover the three core principles of Information Technologies (IT) security 45
4.3 Mapping of proposed use cases to honeynet implementations with the degree of support categorized as: **supported** ($\surd$), **extendable** (+) and **not possible** ($\times$) 50
4.4 Mapping of supported T use cases to honeynet solutions. 51

5.1 Data mappings for mnemosyne 60
5.2 Data mappings for mnemosyne 63

Chapter 1

Introduction

Cyber–attacks on Information Technologies systems have always been a problem to companies around the world. With the rise of the Internet and the fact that more and more, IT systems are getting connected, cyber–criminals start to reach out for new targets. Today almost everybody carries a smartphone or a laptop with personal data which represents a valuable target for these cyber–criminals. The way people communicate and organize their work has changed, everything seems to be connected to the Internet now. Before these connected systems became ubiquitous, targeted attacks, and intrusions were reserved to skilled hackers. Viruses, breaking or rendering the system unusable, were the only threat for ordinary users on the Internet. Besides of viruses and worms, which spread around the Internet quite quickly and affected many users, the first encryption malware variants appeared in the late 80s. These encryption malware variants demanded a small ransom for the decryption key, but were rather easy to break since cryptography implementations were not sophisticated enough yet, and used simple symmetric cipher.

Titled the "first ever ransomware" the PC Cyborg Trojan or AIDS Trojan spread in 1986 via floppy disks handed out to attendees of the World Health Organization's AIDS conference [Kno]. Fast forward to the 21st century, with the rise of the Bitcoin crypto–currency and it's popularity beyond cyber–criminals, ransomware reached new heights and dozens of variants appeared. In addition to that, companies and governments are not only facing criminals, but also sponsored hacker groups and blackhats, hackers with criminal intentions, who have the knowlege and the skills to break into any system and exfiltrate confidential data.

Companies and individuals need to be aware of the risks and the implications following their ubiquitous global interconnections more then ever before. Regularly published statistics by Kaspersky Security Lab[1] show that there are dozens of globally acting criminal groups targeting individuals as well as companies. Groups like Lazarus are especially dangerous, because they stay undetected on the network for a long time. Lazarus, which was

[1] https://securelist.com

responsible for the first digital banking heist on the SWIFT network [Kha], has also been responsible for several other major incidents such as the attack on Sony Pictures [Bau14].

In the last few years a new kind of attack emerged. In late 2015 an attack on an Ukrainian energy provider cut the electricity in large parts of west Ukraine [EUK16]. This was possible, because of the "fourth industrial revolution" (Industry 4.0 − since everything now needs a number). As a result, more and more Operational Technologiess (OTs), such as facility networks and ICSs, were connected to traditional IT networks. Unfortunately these OT systems were never designed to be secure and connected to the Internet. They are designed to be robust and the software is designed and programmed to run for a long time without restart. Therefore many of these systems still rely on old technologies without all of the latest security patches.

With increasing complexity of both local networks and the Internet, there will be even more motivation for blackhats to break into these systems. They have recourse to a huge digital black market for stolen user data, information and account credentials. Moreover there are many black markets available to get undisclosed software vulnerabilities and exploit–kits for different systems. Because most of these vulnerabilities are traded secretly it is very unlikely that they are disclosed and patched by the vendors. System administrators are not only responsible for the latest security patches, they are additionally in charge of keeping an eye on potential malicious activities within their systems. This advantage will always enable sponsored hackers to be one step ahead of system administrators. To overcome their digital advance special systems have been developed to learn the insight of attacks against the network. It's a constant arms–race between cyber–criminals exploiting software vulnerabilities and security experts finding these vulnerabilities and keeping the production systems secure. Therefore finding vulnerabilities and disclosing them to the developers has to have top priority for system administrators and security experts.

In addition to firewalls and Intrusion Detection Systems (IDSs), honeypots are tools to detect undisclosed vulnerabilities and malware being distributed. Among all the available tools used by security experts, honeypots play an important role in observing malicious activities with "fake" production systems. They can be used to detect everything from full Internet Protocol (IP) range scanning to APT techniques and provide a steady stream of threat intelligence to the security community and dedicated experts. By analyzing this threat data, techniques of criminals and hackers can be analyzed, which immediately results in patches for the exploited vulnerabilities. Today, countless honeypots are deployed by companies and individuals to collect valuable information on how their systems might be attacked. Most of them contribute their individual log collection to a shared threat data set which can be subscribed by software vendors and other security experts.

Today, more and more companies are becoming aware of their need for dedicated threat intelligence. IDS mostly do not suffice the need for learning how the company might be attacked. They rely on known signatures and will not detect unknown attack patterns. Furthermore the signal to noise ratio is far higher with IDSs, because they tend to collect lots of false positive events. With the right design, honeypots can fill this gap for companies by detecting if the company is attacked and providing information about these attacks.

Honeypots significantly reduce the amount of false positives by design.

With this thesis the state-of-the-art of honeynet management frameworks is analyzed and enterprise honeypot use cases are designed to build a reliable honeynet inside the company's network. It will also provide a guide on implementing and integrating new custom honeypot sensors with a central honeynet management. The structure of the thesis will be as follows: Chapter 2 will provide baisc background on honeypots and honeynets. Chapter 3 will discuss the evaluation of multiple honeynet management frameworks. In chapter 4 the use cases will be designed and implemented in chapter 5. Chapter 6 will be a explantaion on how to integrate these use cases with the central management. Lastly chapter 7 will conclude what can be learned from designing honeynet use cases and their implementations.

Chapter 2

Background

The concept of using honeypots in enterprise environments is not really new in the information security sector, but it has changed radically over the last ten years. From simple protocol emulating scripts to stealth root-kits and kernel modules, monitoring the operating system itself, a vast amount of different solutions were developed and tested in research and production environments. In enterprise information systems, honeypots are commonly used as an additional layer to the company's information security filling the gap of conventional firewalls and Intrusion Detection Systems. This additional layer is used to capture information about "real" attacker actions and to analyze them with further forensic methods. This information is primarily used to improve detection and defense methods of the company's security team. Furthermore new signatures of monitored attacks can be generated and added to the IDS, which automatically will lower the number of false positive alerts [Pro05]. Because every connection to a honeypot is assumed to be unauthorized, all interactions provide information about ongoing attacks to the honeynet operator and this gathered information is used to identify tactics, techniques, and procedures of adversaries on the network.

To integrate a honeynet in an enterprise network it is necessary to understand the principles behind honeypots and honeynets and learn to select the right products for individual enterprise needs and environments. Therefore, a definition for *honeypot* and *honeynet* is given and strategic considerations are discussed. Moreover, a brief history of honeypots and related work will be given at the end of the chapter.

2.1 Definitions

2.1.1 Honeypot

The first use of the term honeypot dates back to Lance Spitzner, who defined honeypots as follows: "a honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource" [Spi]. They have no legit production use [Spi03b] and are designed to be used in an unauthorized way by attackers. Therefore any interaction with such a resource is deemed to be malicious [MA15]. This leads to much less, but high valued traffic since only traffic directly targeting the honeypot is monitored. A honeypot can either be any arbitrary service or also a full operating system exposed to the attacker [Gor+11]. Its efficiency is defined as the the amount of incoming connections observed. To increase efficiency, honeypots may be advertised either locally or publicly on the Internet. This advertisement should lure possible attackers onto the honeypot to observe as much about their motivations and background as possible. This leads to a common goal of gathering as much information about suspicious interactions as possible and gain knowledge of the attacker's tactics, techniques, and procedures (TTP).

Honeypots are divided into two main categories: passive sensors, which only passively observe and monitor incoming traffic; and active sensors, which offer interaction in the form of network protocols, to the attacker [Ber09]. Passive sensor honeypots could be so called network telescopes [F+08], darknets or darkspaces. In this work the term darkspace will be used, since darknet is a common synonym for the Tor[1] network and network telescope refers to one dedicated product. With active sensors the degree of interaction can vary from zero interaction to full system interaction. Additionally, honeypots can be further sectioned into *low interaction* and *high interaction* honeypots depending on this degree of interaction and the system layer which is monitored.

Low interaction

A *low interaction* honeypot offers only very basic interaction to an attacker. These honeypots mostly emulate only initial steps of different protocols, but do not provide further interaction beyond this initial communication. Emulation in this context means that the honeypot is bound to specific ports simulating some standard protocol like HTTP[2], FTP[3] or SSH[4]. The degree of accuracy significantly affects the deception efficiency, because insufficient emulation accuracy might cause an attacker to terminate the connection and leave the honeypot. Since they are lightweight and can be deployed quickly, many different *low interaction* honeypots can be used to deceive attackers.

[1] https://www.torproject.org/
[2] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[3] https://en.wikipedia.org/wiki/File_Transfer_Protocol
[4] https://en.wikipedia.org/wiki/Secure_Shell

High interaction

High interaction honeypots are conventional network resources [Ber09], which do not limit the degree of interaction by emulating services, but use the full operating system with its user-applications as a valuable target. Specially crafted surveillance tools are used to monitor every system interaction. This leads to more information than with *low interaction* honeypots. With *high interaction* honeypots it is possible to monitor interactions beyond the initial communication and observe multi-stage attacks. Furthermore *high interaction* honeypots deceive an attacker much longer than *low interaction* honeypots and as long as an attacker is attacking the honeypot, the attacks are not directed at the production system.

Types of Honeypots

Apart from the classification based on the degree of interaction, honeypots can also be classified with respect to their purpose. Thereby the purpose does not define the class of attacks, such as web attacks or malware, but the type of deployment and goal the honeypot serves.

Research Honeypots are designed to collect information about all kinds of attacks hitting the honeypot regardless of the severity and origin. They collect statistics of blackhat activity and their attacks, but do not add direct value to the organization [SSY05]. As research resources these honeypots learn general information about threats computer networks may face and provide this information to help organizations securing their networks against such threats. Their goal is to study attacks and the blackhat community's motivations to learn their objectives, behavior and methods [SPP]. Research honeypots are complex systems which need more effort in maintenance since they collect huge amounts of data about many different attacks. In many cases research honeypots are operated by universities or other security research facilities.

Production Honeypots on the other hand, have a lower the risk of compromise and collect more specific attack data by simulating a true production system using structures and data that are parallel the real system [Swe03]. Their goal is not to observe automated attacks and scans, but to detect targeted attacks on the company's "crown jewels". Most production honeypots are deployed on local networks to detect intruders already on the inside of other security mechanisms and who are looking for sensitive information. To reduce maintenance costs and effort, low interaction honeypots are commonly used as production honeypots. Operators of production honeypots expect to be provided with information about relevant attacks only and can neglect everything without directly endangering the company's values.

2.1.2 Honeynet

In advance to a honeypot, a honeynet is not only a single decoy resource, but a network of such resources. Furthermore, this network can consist of any arbitrary type of system such as honeypots or networking hardware and might provide a threat intelligence interface to the system administrator. Most honeynets nowadays indeed contain such a management interface to visualize reported events and to deploy new honeypots. Practical examples for such an interface could be a custom web interface or a integration with data visualization tools based on the elastic stack [5]. In fact the plausibility of such a network can be greatly increased if it consists not only of honeypots but also of standard production systems [KK]. Honeynets scale much better than honeypots with respect to large company networks and if virtual, literally dozens of honeypots can be deployed on a single physical machine.

2.1.3 Honeytoken

A honeytoken is a very unspecific resource which can be seen as the complemented part of a honeypot. They share the same concepts and similar to a honeypot, every interaction with a honeytoken is of malicious purpose [STI03]. While a honeypot can be seen as a computer, server or physical resource, a honeytoken is everything digital which fulfills the same purpose like digital data or breadcrumbs leading to a honeypot. Defined as a digital entity, a honeytoken can be a credit card number [Spi03a], authentication credentials, emails, documents and even database entries. Because of this flexibility to adapt honeytokens to every environment, they can be used in many different contexts where honeypots would fail. They solely serve as a resource to mislead the attacker and lure him to the honeypot. This can also be seen as honeypot advertising, which aims to increase the range of the honeypot and attract hackers more than the real production system. Although their value seems to be rather small, honeytokens can be used to detect targeted attacks and advanced persistent threats.

2.2 Considerations

Before starting to deploy a honeynet, some strategic considerations should be made. The following considerations can help to determine the right honeypot types, the right hardware and the location of deployment: It's crucial for the effectiveness of a honeypot to look like a real system and not to reveal it's purpose to an attacker. An insufficient deception or incomplete emulation may reveal the honeypot to a skilled attacker who will immediately leave the network or, worse, compromise the host system and network. With the goal to learn about the attacker's tools, tactics and procedures, the main focus must be keeping the intruder connected to the honeypot for as long as possible. Every interaction with the honeypot helps to learn how to defend the real production system from being compromised.

[5] https://www.elastic.co/

To select the best fitting honeynet product it is necessary to define the core requirements of the honeynet, like how statistics are provided to the honeypot operator. The data might be accessable using a dedicated web interface with different kind of visualizations. Or a commandline interface can be provided to forward the collected data to the company's Security Information and Event Management (SIEM) system. In case of requiring a dedicated web interface, functionality to deploy honeypots directly and monitor their status might be useful. Also different kinds of attack statistics can be displayed on the web interface to show activity on the honeynet. On the other hand, a dedicated web interface also introduces additional management overhead by requiring additional server infrastructure and maintenance. Without the web interface, a command line interface can provide basic management functionality such as deployment and event data forwarding, but reduces the usability and increases the amount of required knowledge for the honeynet operator.

Besides the central management server, the modularity and flexibility of the honeynet should be considered. Some products are very flexible and modular, so customizations and module extensions are possible. Other ones which are difficult to customize, do not need to be built and installed from source. Most of these products come with a pre-built system image which can be installed either virtually or on a dedicated server directly, but any sort of customization would require rebuilding of the system image of the product and reinstalling it to the server. Furthermore, a good modular design of the honeynet enables the operator to replace specific modules with company specific ones; e.g. replacing the the communication module that collects data from the distributed honeypots with a publishsubscribe protocol like hpfeeds [6].

A flexible solution can also be integrated more easily in existing SIEM systems within the company's information security department. Having threat data from deployed honeypots available in the SIEM system enables the company's CERT team to learn about targeted attacks against the company and prepare necessary incident recovery tools. Thereby, the value of honeypots in detecting new, unknown exploits plays an important role to extend the abilities of an existing IDS. It is not enough to have either a honeypot or an IDS, but the combination is the best option since both complement each other.

It is also worth considering the blocking of outgoing traffic from the honeynet to prevent attackers to misuse the honeypot for further attacks on remote computers on the Internet or the local network. Blocking this potentially malicious traffic does not directly increase the security, but lowers the risk that attackers can spread their exploit from the honeypot on. Unfortunately this also stops malware from connecting back to a command and control center and therefore the amount of data collected about the malware is significantly decreased.

When deploying a honeypot inside the honeynet, the limited emulation possibilities of low interaction honeypots have to be kept in mind. They are more common and can be deployed easier, but do not collect as much information as high interaction honeypots. In cases where a high interaction honeypot is required it is necessary to have the possibility

[6] https://redmine.honeynet.org/projects/hpfeeds/wiki

to revert the system to its original state immediately to prevent further spreading of the attack. If the company is more interested in malware and automated attacks low interaction honeypots should be enough. For targeted and multi-stage attacks high interaction honeypots are required to observe the attacker's actions on the system for some time period. Additional honeytokens can be used to make up an efficient honeynet. Another consideration should be made regarding the scalability of the honeynet. Low interaction honeypots usually scale a lot better since they mostly require zero maintenance. With enough resources, distributed deployment of honeypots can help to increase the range of attacks observed on the honeynet, but requires a centralized management interface to collect all the data reported from the honeypots.

2.3 History

The first documented story of tracking an intruder is about Clifford Stoll [Sto89], an astrophysicist who worked as a system administrator at Lawrence Berkeley National Laboratory. In 1986, Clifford observed discrepancy in the computer time billing system. Only 75-cent were enough for him to carefully review system log files and set up network monitoring to detect suspicious traffic. Thereby, he found a hacker accessing the system remotely to search for files containing confidential information like military documents. To decoy the hacker and increase the time to monitor his activity, Clifford created fake, but legit looking documents for him to find. In collaboration with the FBI and after months of observation they identified the intruder as a German citizen, selling the information to foreign intelligence agencies.

Another use of a honeypot as a system to attract and lure attackers to learn about their techniques and to examine their tools by researchers was published by Bill Cheswick in, An Evening with Berferd [Che]. Cheswick not only tracked the intruder and logged his activity, but also emulated an operating system, together with a variety of services, manually to deceive him. Cheswick concluded that his chroot jail wasn't worth the effort since it was hard to get it right, and never quite secure [Che].

Following these first pioneers, honeypots gained more attention with the rise of the Honeynet Project[7] in 1999. The Honeynet Project was founded by Lance Spitzner with the goal to improve the security of the Internet in three ways: increasing awareness of security threats, providing details about securing and defending system resources and data, and providing the necessary tools for further research and development [Pro04]. Since then, various different honeypots and tools have been developed by the Honeynet Project. Honeywall [8], Glastopf [VKM10], Conpot [9], Cuckoo [10] and Dionaea [11] are just a short excerpt from a long list of tools.

[7] http://honeynet.org
[8] https://projects.honeynet.org/honeywall/
[9] http://conpot.org/
[10] https://cuckoosandbox.org/
[11] https://www.edgis- security.org/honeypot/dionaea/

Lately a new focus on the Internet of Things (IoT) has emerged with more and more systems being connected. With botnets like Mirai, Stuxnet [Che10] and others attacking vulnerabilities in embedded devices, the need for security mechanisms to detect new attacks has grown significantly.

2.4 Related Work

2.4.1 Analysis studies of honeypot software

There are many different studies evaluating honeypot software with different levels of detail. Moore et al. [MA15] performed an analysis of different honeypot systems to identify an appropriate product for further threat data analysis. The selection of honeypot systems they evaluated covered both Linux and Windows based systems such as:

- *BackOfficer Friendly*
- *HoneyBot*
- *Nepenthes*
- *Dionaea*
- *Kippo*

Through the evaluation, which loosely focused on the simplicity and the reported information, they decided to run an instance of Dionaea deployed with the MHN for a research period. The experimental system was set up on an Ubuntu 12.04 VMware workstation virtual appliance and connected to a network arm of the Demilitarized Zone (DMZ). To collect incoming traffic it was given a public IP address. The collected threat data and malware samples were exported from the MHN server via FTP to another workstation where they used the *R* statistical programming language [12] to analyze it. They used R-scripts to analyze the source IP of attacks, the destination port and the corresponding timestamp. Figure 2.1 shows the different attacked destination ports with the amount of attacks per port. They explained the high amount of attacks on port 5060 with an sustained attack on the Session Initiation Protocol (SIP) which handles Voice-over-IP. They concluded that any system with a public IP needs dedicated security mechanisms and honeypots can be a great source for identifying potential risks for the network operator.

Another study conducted by Mokube et al. [MA07] who examined different honeypot solutions and described their concepts, approaches to implementations and challenges. At first the paper gives a good introduction to the topic and explains the differences of low, medium and high-interaction honeypots, as well as honeytokens. They describe

[12] https://en.wikipedia.org/wiki/R_(programming_language)

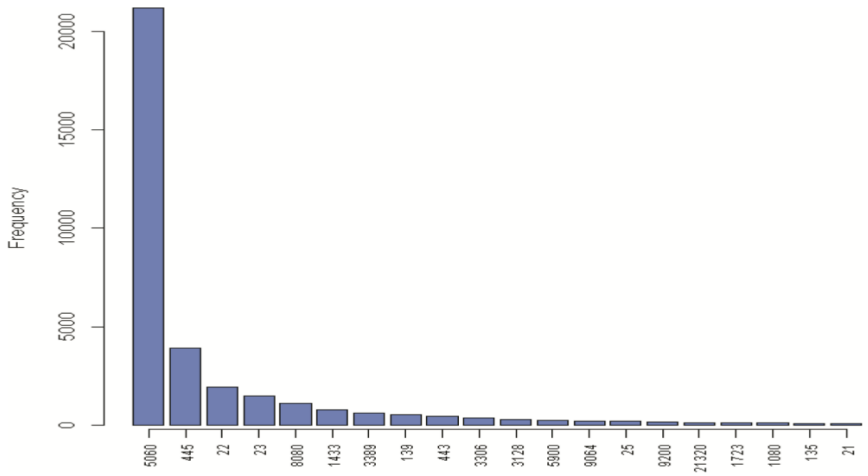

Figure 2.1: Attacks per destination port as observed by Moore et al [MA15]

honeytokens as a simple and flexible tool similar to a honeypot which ensures the detection of unauthorized access to a file or a database. Their main contribution lies in their discussion of honeypot concepts and approaches to honeypot implementation. According to their work the following questions should be answered to determine the concept and strategy for the implementation of a new honeypot:

- What kind of data should be made available through the honeypot?
- How do you prevent uplink liability?
- To build or not to build?
- Where is the best location for your honeypot?

In addition to the aforementioned discussion of concepts and approaches Mokube et al. comment on the legal pitfalls when deploying and using a honeypot. They count three major legal issues which must be addressed when deploying a honeypot. The first issue, **Entrapment**, is not of any concern to private honeypot operators as it requires an involvement of the government. The second issue has to be considered with more care as it is about **Privacy**. Mokube et al. argued that although a network operator has the responsibility to keep the network secure, there are some limitations to network monitor such as federal statutes, privacy policies or terms of service agreements. Last but not least, **Liability** is of concern for the operator, because a compromised system could be used by the attacker to attack others. Therefore they recommend to come up with a good strategy and goals before setting up the system.

A more recent and practical study on the attractiveness of honeypots was published by Sochor et al. [SZ15] in 2015. The study aimed to evaluate the attractiveness of honeypots

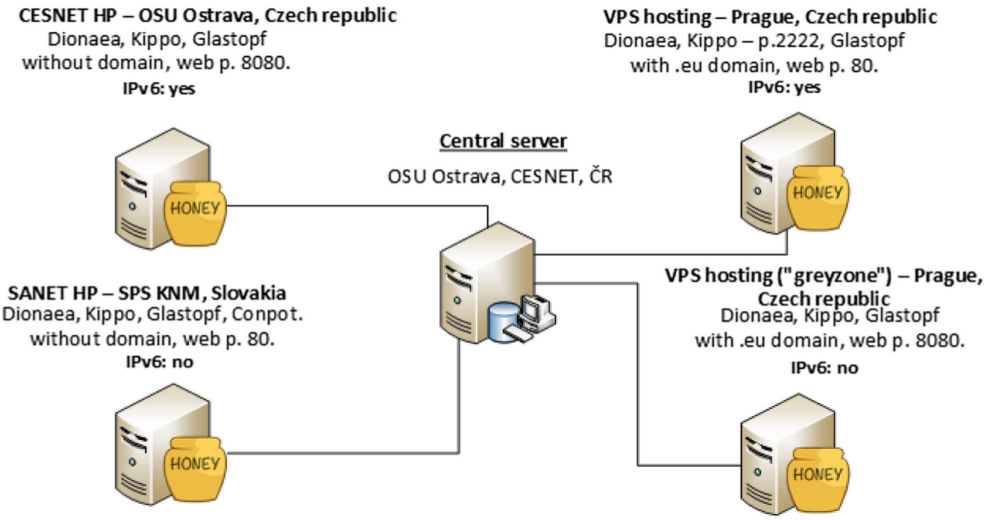

Figure 2.2: Schematic network layout used by Sochor et al [SZ15]

with respect to the underlying network. In the work they compared honeypots deployed on the Czech academic network CESNET and the Slovak academic network SANET versus honeypots deployed on commercial private hosting networks. They analyzed data obtained from a honeynet consisting of three low interaction honeypots: Dionaea, Kippo and Glastopf. The honeynet itself was composed of four servers with all low interaction honeypots implemented. Dionaea was configured to emulate SMB, MySQL MSSQL and FTP on well known ports. Kippo, a SSH honeypot, emulated the SSH protocol on standard port 22 and non-standard port 2222. Glastopf used ports 80 and 8080 to serve web content. The first server, connected into the Czech academic network CESNET, got no Internet domain assigned to its IP address. The second server was connected into the Slovak academic network SANET and got no Internet domain assigned neither. The third server used a commercial private hosting in the Czech Republic and its IP address belonged to the "grayzone" (eg. adult content) of the Internet. The fourth server was also connected to a commercial VPS hosting in the Czech Republic, but wasn't available to adult contents. In contrast to the first three servers, the last used port 2222. The overall schematic structure is also highlighted in figure 2.2.

Sochor et al. further performed a detailed analysis on the gathered information and statistics. They compared the different servers with respect to the attacked ports and classified attacks according to the emulated service. In the end they concluded, that attackers distinguish between sensors according to their IP. It's relevant to them weather an IP address belongs to a range of interest to them, an what services are associated with this address range. Further, the security mechanisms used have an influence on the attractiveness for attackers. As future work the author propose to focus on SCADA honeynets and their attractiveness.

Mairh et al. [Mai+11] reviewed advances in honeypot strategies, applications and de-

ployment. With their study they evaluated proposed aspects and tactics with honeypot applications and deployments in different environments. Apart from the classical *educational* and *reseach* applications, they discuss combinations with IDS and *hybrid* solutions. Both are not new, but Mairh et al. provide some good detail on the purpose and the strategy used with both deployments.

2.4.2 Honeynet implementations

A key constraint in honeypot-based attack detection is the need for large-scale deployment, as timely detection is strongly dependent on a large number of honeypot sensors spread over different location on the Internet [Chi+09]. Even though honeypots are a valuable security resource not every company can afford to devote resources for deploying a large scale honeynet. Chin et al [Chi+09]. proposed a novel approach to share infrastructure for deploying and monitoring honeypots on a distributed base. With their HoneyLab project the authors present an architecture and implementation of this approach which is designed to increase coverage and accelerate innovation among security researchers and security industry experts. Figure 2.3 highlights the overall architecture with the individual parts. The system was designed with focus on scalability, flexibility and ease of use. It should scale easily up to a large number of deployments and it should be flexible enough to run any arbitrary honeypot configuration. It should also be easy to use for researchers without any configuration overhead and deployment steps. Additionally, a compromised honeypot should not be able to affect any other honeypot in an malicious way.

The proposed system is built around a central HoneyLab Central (HLC) server and multiple distributed sensor nodes. The HLC manages the users and available resources, as well as the web interface where resources can be purchased. The sensor nodes are deployed on Xen servers and can either host only the sensor software, or both sensor software and honeypot software. Each honeypot can be deployed dynamically and is executed in a virtual machine (VM). To collect traffic from sensor-only nodes, the client is provided with a VPN server address and the traffic is redirected from the distributed sensor to the client-side honeypot. Due to obvious performance and scalability issues, Chin et al. recommend to host the honeypot on the same node as the sensor software. To ensure isolation, a XenoServer platform is used which allows to manage and deploy Virtual Machines across a large server infrastructure. The XenoServers enable users to manage and organize their deployments and keep track of used resources which are relevant for billing. To ensure that client honeypots cannot be used as part in a botnet or to attack external resources, HoneyLab imposes traffic restrictions to outgoing traffic of these honeypots. By default all outgoing connections, except connections back to the probing host, are denied. The main advantage of HoneyLab over other honeynet implementations is it's collaborative resource sharing and simplicity in requesting resources for new honeypots. Clients with specific interest in only a small set of ports can share their IPs with other clients. Moreover, clients can share their complete traces captured with the honeypot with other parties which analyse these traces.

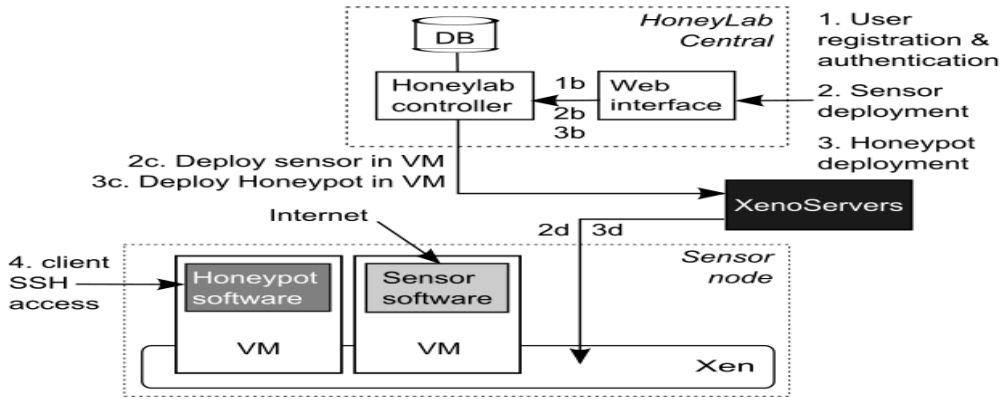

Figure 2.3: HoneyLabs's architecture consisting of the HLC and the sensor nodes [Chi+09]

Unlike others, HoneyLab allows security firms and security researchers to deploy their own honeypot services, instrumentation code and detection algorithms, dispensing the need for setting up a separate honeypot infrastructure whenever a new attack detection method needs to be deployed or tested. At a high-level the authors envision their service oriented architecture to be much like PlanetLab[13], an open plattform for planetary-scale services.

According to Berthier [Ber09] malicious network activity can be categorized by two distinct approaches: 1) monitoring production network which contain live hosts, actually used by clients, and 2) monitoring an unset address space. The second approach profits from non-existing production traffic and therefore no traffic filter needs to be applied. Tools used in these approaches are of the active and passive kind. Active tools, which are honeypots, are defined as network devices "that provide a mechanism for completing network connections not normally provided on a system and logging those connection attempts". Passive tools such as IDS are used to monitor unused address ranges and are also known as *zero interaction* honeypots. Berthier organized honeypots according to three main categories: Fidelity, Scalability and Security. Whereas the fidelity of a honeypot determines its level of interaction itself, scalability and security describe implementation details and the attributes of honeypots. The overall goal of this work was to build an advanced honeypot system which integrates with existing security tools and to deploy it at the University of Maryland to improve the quantification of malicious activity on the network.

The advanced honeypot system provided with this work, called *HoneyBrid*, is a hybrid honeynet which can be configured dynamically by combining network flows with autmated honeypot management. Figure 2.4 shows how Honeybrid integrates with the network and network flow analysis. The final system combines not only the honeynet Honeybrid, but also netflow analysis and a data classification engine. Nevertheless, the most interesting part is the design of Honeybrid itself, as this system combines low and high interaction honeypots and is able to redirect traffic from the low to the high interaction honeypots.

[13]https://www.planet-lab.org/

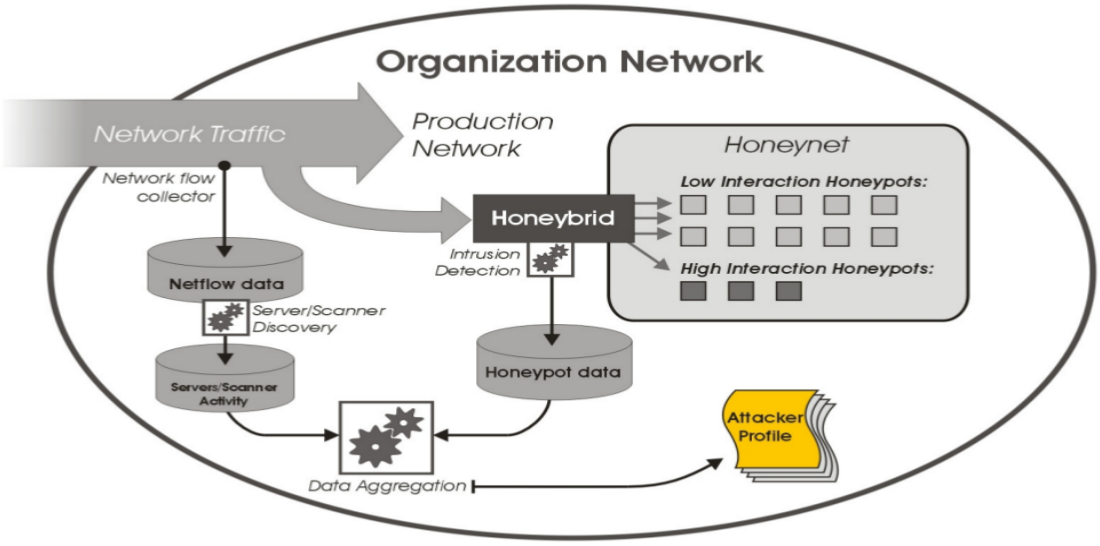

Figure 2.4: HoneyBrid is combined with Netflow analysis and a threat data aggregation unit [Ber09]

Inside HoneyBrid two distinct engines handle incoming traffic. First every packet is viewed by the *decision engine* which performs active filtering. Second the *redirection engine* handles the redirection of filtered packets to the high interaction honeypots. The idea is to combine the scalability of low interaction honeypots with the detection capabilities of high interaction honeypots. A large set of IPs is monitored with low interaction honeypots, such as honeyd or dionaea. These honeypots are used to cover as much attack surface as possible and if an interesting attack is detected the redirection engine is in charge of transparently redirecting the traffic to the high interaction honeypots. This architecture enables the operator to specify multiple parameters which define an "interesting" attack for further investigation. The role of the high-interaction honeypots is to offer full interaction to the attacker, in order to record detailed information about the attack process that was flagged as interesting. In figure 2.5 HoneyBrid's internal engines are visualized on a high level.

Another approach to build a honeynet was published as part of the Honeynet Projects Know Your Enemy [Pro05] whitepaper series and describes a centralized approach to deploy honeypots behind a firewall. This firewall, also called honeywall, is equipped with network level monitoring and firewall rules to monitor incoming and block outgoing traffic. It is also the key element of the honeynet, because it separates the honeynet from the rest of the Internet and ensures that no honeypot is used to attack other victims. A feature–rich implementation of such a firewall is provided by the Honeynet Project under the name *Honeywall CDROM*. It implements all the tools and techniques required by such a firewall. Since all traffic incoming and outgoing has to pass the honeywall, it's the critical point for threat data aggregation and logging. Figure 2.6 provides an example setup for a honeynet featuring a honeywall as command and control center. The honeywall acts

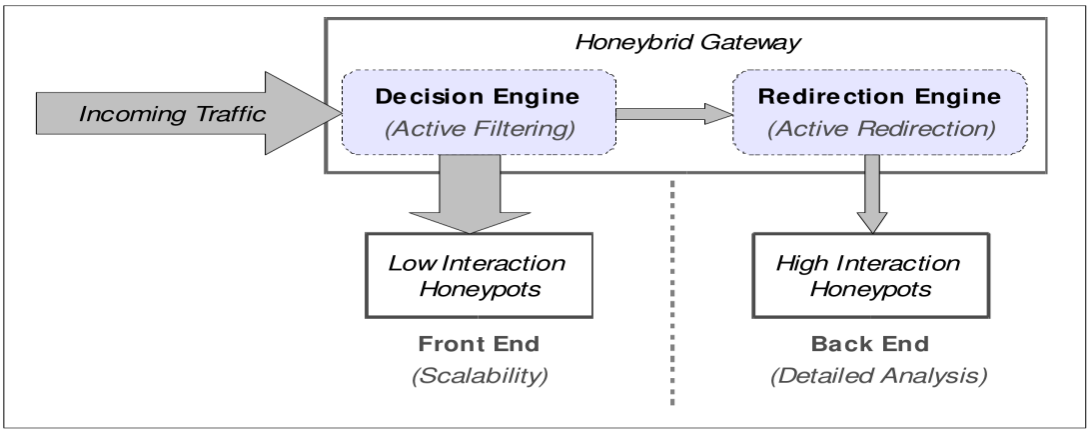

Figure 2.5: HoneyBrid's internal architecture consisting of a gateway, a set of low interaction honeypots and a set of high interaction honeypots [Ber09]

as a layer two bridge with at least three interfaces. The first (eth0) is connected to the production network, the second (eth1) is connected to the honeynet's system network. A third interface is also required as management interface to setup the honeywall and remote administration. Any honeynet is required to implement *Data Control* and *Data Capture*. Data Control describes the ability to control incoming and outgoing traffic in a way that no honeypot can be used in an adversary way. It does not eliminate, but reduces the risk since one of the questions the operator has to answer is how much outbound activity is controllable? The more activity allowed to the attacker, the more can be learned, but also the more harm they can potentially cause. Honeywall CDROM implements multiple features, such as IDS and connection limiting to ensure Data Control. Having Data Control implemented, *Data Capture* can be added to collect information about incoming attacks. The purpose of Data Capture is to log all of the attacker's activity. This is the whole purpose of the honeynet, to collect information. The key is to collect data on different levels. Source IP, destination IP and related ports on the TCP/IP protocol level are a good starting point, but for advanced learning, application data needs to be collected too. Differing on the amount and the type of honeypots deployed the honeynet might be able to log firewall, network and system level data. The last one is highly important to learn, because if provides information about attackers interaction with the honeypots and possible unknown attacks. With the rise of encrypted protocol channels such as SSH it isn't enough to capture the data on the honeywall only. Each honeypot on the honeynet has to contribute logs to the overall data collection. The last feature recommended by the Honeynet Project is *Alerting*. The authors recommend to either use a round-the-clock monitoring staff or, if not affordable, an automated monitoring tool. Such a tool should be able to report via email or similar to the operator with a overview of successful attacks. But overall, constant supervision is required by every honeynet. With this work the Honeynet Project provided the security community with guidance on how to setup an advanced honeynet and the authors commented on the key elements on such systems which need to be taken into account.

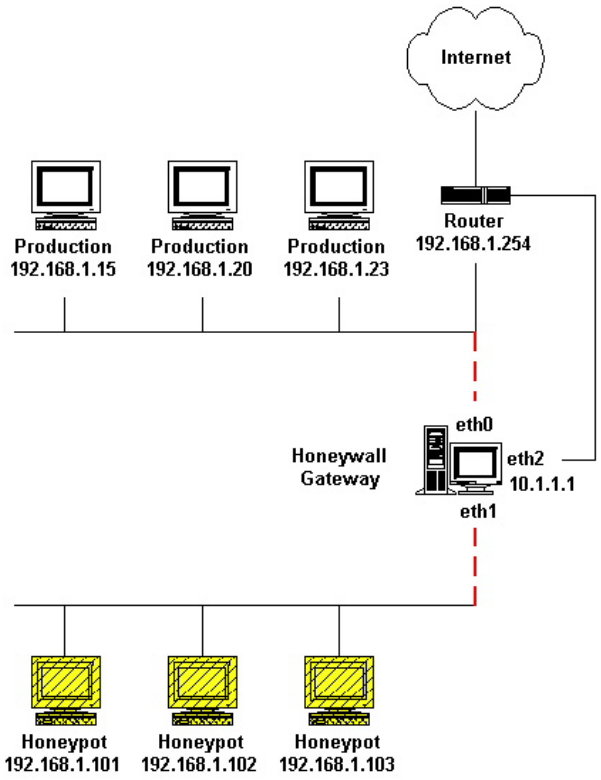

Figure 2.6: An example setup of a honeywall with Honeywall CDROM and a local honeynet consisting of three honeypots [Pro05]

Chapter 3

Evaluation of Existing Honeynet Implementations

Prior to the evaluation some minimum requirements, which need to be satisfied for further consideration, were defined. These minimum requirements contain the possibility to integrate the honeynet with an existing SIEM system, the distribution property and the support for modern state-of-the-art honeypots. To analyze state-of-the-art honeynet products, a market research was conducted where multiple honeynet products were evaluated and compared on a uniform basis. The evaluation period lasted from November 2016 to January 2017 and only previous releases were included in the results. Furthermore, the existence of many "research-only" solutions, which are either not feature complete or do not meet the minimum requirements to be used in an production system, draw the focus to just a couple of production-ready systems. Out of these products a selection of six possible candidates which represented the broadest variety of different solutions were chosen, because a fully detailed evaluation of all would have exceeded the limits of this thesis.

As a frame for the evaluation three top-level categories were chosen: *1)* Architecture; *2)* Integration; *3)* Information collection. Each top-level category was further split in two to three sub-categories. in Table 3.1 all sub-level categories are listed according to their top-level category. To get a consistent grading several predefined criteria determined the points for each sub-level category. The criteria are designed to be very specific on different aspects and a 0/2/4 points grading scale was used to assign points to each criterion. 0 points indicated that the criterion could not be satisfied, 2 points were given if the criterion is partially satisfied, but additional effort is needed or minor parts were missing. To a fully satisfied criterion 4 points were assigned. Table 3.2 lists the criteria for each sub-level category.

The evaluation of six selected candidates is now described in detail. Focusing on the top level categories each candidate will be sectioned in three parts: *Architecture*, *Integration*,

Architecture	Central Management
	Design
	Distribution
Integration	Infrastructure
	Deployment & Maintenance
	Project Support
Information Collection	Honeypots
	Threat Intelligence

Table 3.1: Top- and sub-level categories

Information Collection; each part will give a brief summary of the criteria.

3.1 CyberTrap

CyberTrap is a commercial product from *SEC Consult*[1] security company, which they offer either as a product to be used and maintained by the customer himself or as a service maintained by SEC Consult. Either way, the system is integrated in the customer's network infrastructure.

Architecture

The architecture of CyberTrap is straightforward. It is a high interaction honeypot featuring a variety of deception techniques. It can convert a client computer or a server to a honeypot. It also features a web interface accessible for the operator where logged events can be analyzed. If the operator wants to deploy multiple instances of CyberTrap through the network, then it needs to be done manually because CyberTrap does not offer a remote deployment option on a central server.

Integration

Since CyberTrap is only available for Microsoft Windows, it requires a Windows standard client or server on a dedicated hardware or in a virtual machine to be setup. The advantage CyberTrap has, with respect to maintenance is that it does not need to be maintained by the operator, but can be maintained and updated by SEC Consult as a service. This reduces the maintenance work load for the operator and ensures that CyberTrap is always up-to-date with the latest features and updates. Through evaluation of CyberTrap it was found to be quite difficult to lure attackers to the honeypot if it is facing the internet on a DMZ, because a lot of advertisement of the honeypot's IP address is needed.

[1] https://www.sec-consult.com/index.html

Central Management	web interface	is a webinterface provided
	commandline interface	is a commandline interface provided
	central honeypot deployment	can honeypots be deployed from the central management
	deployment	what is the effort to deploy the system
	online IDS rulset	are IDS rules online and can they be selectively activated
Design	modular	is the software moduar designed
	exchange moduls	can moduls be exchanged
	add moduls	can we add new moduls like honeypots
	extend moduls	can we modify or extend single moduls
	extendability	can we extend the system itself to support more moduls
Distribution	distributed deployment	is the honeynet designed with distributed honeypots
	distribution paradigm	how is the software distributed
	automated distributed deployment	is the deployment automated
	distributed honeypots	are the honeypots distributed
	distributed proxies	are proxies ditributed, but honeypots local
Infrastructure	virtualization supported	does the system work in a virtualized environment
	addition Hardware required	is hardware required to setup the system
	gateway	is the system working as a gateway
	firewall required	is a firewall required for the honeynet
	host OS requirments	are there OS requirements
Deployment & Maintenance	deploy using virtualization	honeypot deployable in VMs
	deploy using container	honeypots deployable in docker-container
	scripted deployment	are scripts provided for deployment
	provided by package manager	can honeypots be installed from package manager
	build from source	need to build honeypots from source
	easy to maintain	does it cost a lot of maintenance effort
	easy to revert honeypot	can the honeypots be reverted easily
	easy to redeploy	can we revert and redeploy the system easily
Project Support	active development	is the product under active development
	active maintenance	is the product actively maintained by the developers
	recent contributions	was there contribution to the product during the last 6 months
	quality of documentation	what's the size and quality of the documentation
	license	under which license is the product developed
Honeypots	modern honeypots supported	are state-of-the-art honeypots supported
	high interaction honeypots	are high interaction honeypots supported
	medium interaction honeypots	are medium interaction honeypots supported
	low interaction honeypots	are low interaction honeypots supported
	web application honeypots	are web application honeypots supported
	ICS honeypots	are Industrial Control System honeypots supported
Threat Intelligence	automatic threat data acquisition	is the threat data collected automatically
	application layer data	is application layer threat data provided
	network layer data	is network layer threat data provided
	OS layer data	is operating system layer threat data provided
	IDS supported	are *IDS*s integrated or supported
	SIEM integration	can the threat data be forwarded to a Security Information and Event Management system

Table 3.2: Discrete selection criteria listed for each top- and sub-level category

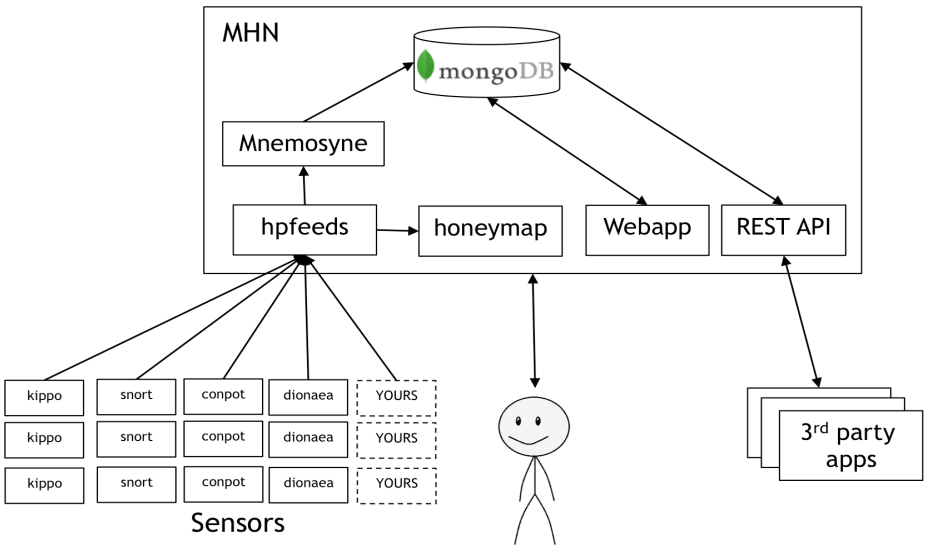

Figure 3.1: MHN modular overview showing the different parts [Anob]

Information Collection

Even though CyberTrap can be seen as a traditional honeynet when deployed distributed or locally as a honeyfarm, it does not support any state-of-the-art honeypots. To deploy, e.g. a web application honeypot with CyberTrap, the operator has to configure the host operating system to act like a web server hosting a web application. On the other hand, the rootkit used to convert the client to a honeypot gives highly detailed logs on an operating system level where attackers can be observed quite well. With support for syslog [2] forwarding, SIEM integration is not a problem.

3.2 Modern Honey Network

Developed by *Anomali* [3] for their threat intelligence platform, the MHN is a state-of-the-art honeynet solution. Anomali defines it an enterprise-grade [Anoa] honeypot management system with fast honeypot deployment and a central management unit.

Architecture

The MHN consists of two independent parts: the MHN server and remote honeypot sensor nodes. The server is responsible for data collection, control and presentation.

[2] https://en.wikipedia.org/wiki/Syslog
[3] https://www.anomali.com

The honeypots can be located locally on the same server, on the same network or even distributed over the Internet. The only requirement is that they are able to send the data to the central server. The threat data is collected from incoming attacks and send to the central management server via the hpfeeds[4] protocol. Since the deployment of honeypots has to be done manually, SSH [5] can be used to connect to each sensor node for deployment of new honeypots or update existing ones. To deploy a new honeypot bash scripts are available on the MHN management server for most of the typical honeypots. New custom deployment scripts can be added to support new honeypot products. These scripts must be added before the server is installed. Additionally, the provided REST-API can be used to attach specialized analysis and data processing modules to the honeynet or to extract the raw data from the database. Only the ability to automatically deploy honeypots remotely over the network is missing at the central server. A normalization of the collected data is listed on the web interface together with some statistics about past and current attacks. It also features a *threat map* where source IPs are marked according to their geoIP resolution.

Integration

The official recommendation for the host operating system from Anomali is a Debian based *Ubuntu 14.04* or *CentOS 6.7*. Both can also be used inside a virtual environment without any additional hardware. 2 network interfaces are further recommended on the honeypots, because one should be used for external connections, and one for internal maintenance by the operator. To ensure data control, which is the concern of protecting non-honeynet systems that an attacker might target from a compromised honeypot [GKO05], a good firewall setup and network separation are required anyway. The MHN can be used in many different setups and is quite flexible with respect to the underlying network, but nevertheless a DMZ is probably the simplest setup to ensure security and deception together. The general installation process of the honeynet is rather easy since everything is automated with bash scripts. It also adds support for *ELK stack* and *SPLUNK*[6], as well as email notifications for the operator. The advantage of a virtual setup arise when reverting the honeypots back to a clean state after a compromise. To simplify the process of installing the honeynet, the community has created extension to deploy it using docker and or ansible playbooks. In general, the support from the community is good and issues are addressed quite fast, but without a commercial product the maintenance is in the responsibility of each operator.

Information Collection

Due to the active community and the interest in the honeynet many state-of-the-art honeypots, including Glastopf [7], Kippo [8], Dionaea or Conpot, are already supported by the

[4] https://github.com/rep/hpfeeds
[5] https://en.wikipedia.org/wiki/Secure_Shell
[6] https://www.splunk.com/
[7] https://github.com/mushorg/glastopf
[8] https://github.com/disaster/kippo

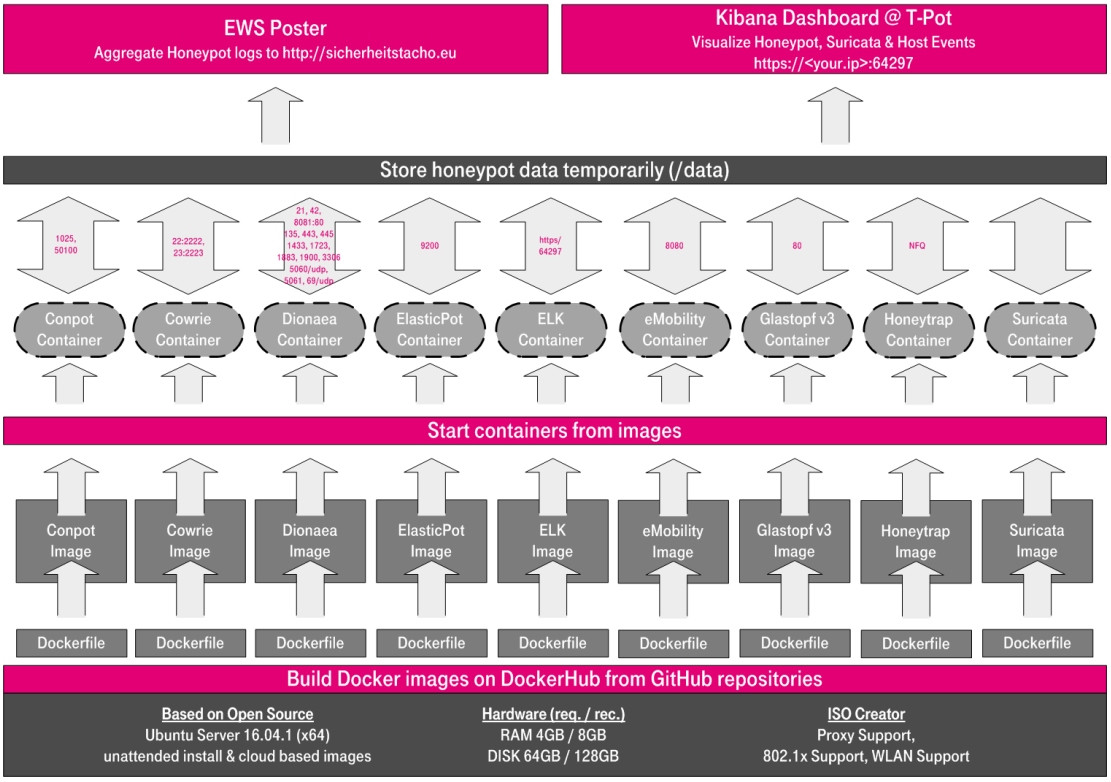

Figure 3.2: T-Pot by Deutsche Telekom's Security Team

MHN. However, the MHN does not support high interaction honeypots such as Argos [9] yet, and therefore no operating system level attack data is collected. Aside from operating system level data, network and application level threat information is collected and stored in a database on the central server. Lastly, the SIEM integration, which is one of the minimal requirements on the evaluation, can be achieved with the MHN by either, using already provided scripts for ArcSight and Splunk, or extending the existing hpfeeds-logger module to support further SIEM systems.

3.3 T-Pot

Developed by the security team of *"Deutsche Telekom"*, *T-Pot* is an enhanced multi-honeypot platform. The intention behind T-Pot was to build a platform easy to deploy and simple to maintain, which can be used in home networks as well as in large corporal networks and where people are motivated to contribute to security research [Deu15].

Architecture

[9] https://github.com/rvermeulen/argos

T-Pot comes as a fully loaded operating system image bundled with many different well-established honeypots and a web application using ELK [10] - Elasticsearch, Logstash, Kibana - for data visualization. Since T-Pot is designed as a multi-honeypot platform including management and honeypots, out of the box it cannot distribute honeypots across the network with a centralized management server. Luckily CyberPoint, a north American security company, adapted T-Pot to be able to distribute honeypot nodes and gather all the data on one server. The utilization of open source tools for different parts of the honeynet simplifies the customization and extendability of the overall system. The honeypots are further deployed using Docker [11] container and the data is visualized on the Kibana web interface.

Integration

To integrate T-Pot in an existing network only a dedicated or virtual server is required to deploy it. The only maintenance required is the redeployment of honeypot container in case of a compromise, but since there are no high interaction honeypots it's very unlikely to happen often. Also the redeployment is made very easy through a custom-developed container management system on the web interface. Similar to the Modern Honey Network T-Pot is developed by a company for their own use and therefore gets regular updates and new features. Additional contributions from the community regarding new tools or honeypots to integrate are also merged by this core developer team.

Information Collection

Currently the most important open source honeypots are already integrated with the honeynet. Additional dockerized honeypots can easily be added, but additional configuration of the ELK stack is required to visualize the logged data. The standard setup of T-Pot monitors both application level and network level threat information from honeypots and IDS. By default the collected data is submitted to the "Sicherheitstacho" [12]. Unfortunately there is no direct option to integrate T-Pot with any existing SIEM system, but with support for syslog this can be implemented straight forward.

3.4 SURFcert IDS

SURFcert IDS, the successor of SURFids, is an open source Distributed Intrusion Detection System based on passive sensors [13]. It's a centralized approach to gather data from distributed sensors which forward incoming traffic to the central server.

Architecture

[10] ELK
[11] https://www.docker.com/
[12] http://www.sicherheitstacho.eu/
[13] ids.surfnet.nl

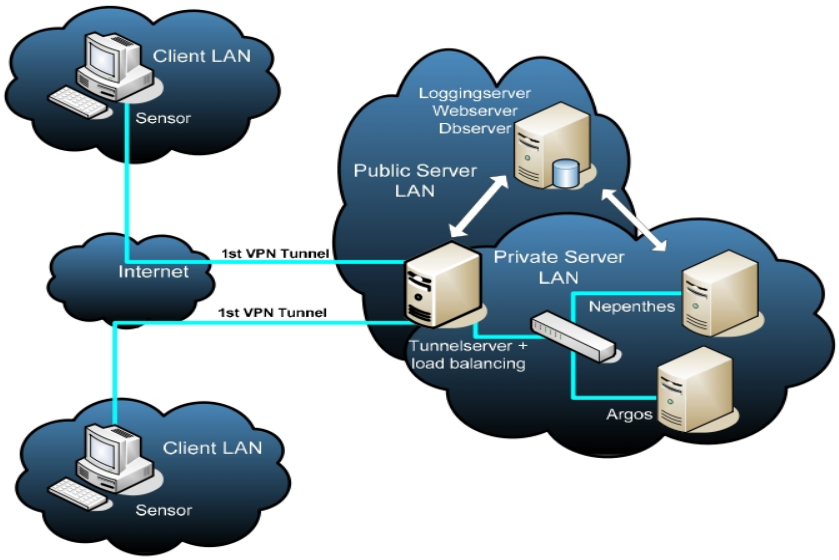

Figure 3.3: SURFcert IDS consists of a internal network and the distributed clients

Compared to the MHN SURFcert IDS does not really distribute the honeypots, but sensor nodes redirecting traffic through a layer-2 VPN tunnel to a centralized tunnel server and further to the local honeypot farm (honeyfarm). For the VPN tunnel OpenVPN [14] is used on the tunnel server and it can also run a loadbalancer for higher amount of traffic to one specific honeypot type. Locally connected to the tunnel server and the honeyfarm a logging server with a web interface and a database are installed which collect the threat data and visualize it for the operator.

Integration

SURFcert IDS also ships with a pre-build operating system images to be easy to deploy and maintain. During deployment the logging server, the tunnel server with the honeyfarm and sensor nodes are setup. Both servers can be configured with the official system images and the sensors can be converted using tools from a custom package repository. Although it's not maintained anymore by the developer, the official website features a considerably large and detailed documentation and installation guide.

Information Collection

Supported honeypots are Nepenthes and Dionaea, Kippo and Argos, a high interaction honeypot. It does not support cutting-edge honeypots like SNARE [15] or conpot, but with Dionaea and Kippo two high-value low interaction honeypots can provide enough information about manual and automated attacks. The data collected on the honeypots

[14] https://openvpn.net/
[15] https://github.com/mushorg/snare

is further stored in a PostgreSQL database on the logging server which hosts the web interface. Furthermore SURFcert IDS data can also be manually integrated with a SIEM system via syslog forwarding.

3.5 Honeywall

The *Honeywall CDROM* is a bootable CDROM that installs all of the tools and functionality necessary to quickly create, easily maintain, and effectively analyze a third generation honeynet [Pro]. Developed by the Honeynet Project as a gateway for honeynets it found wide adoption in research and production honeynet solutions. Although it was released back in 2005 it is still used in many implementations nowadays.

Architecture

Honeywall acts as a gateway for a network of low and high interaction honeypots. Every incoming connection is routed through the Honeywall to one of the honeypots depending on IP and port. The honeypots are deployed behind the Honeywall gateway inside an isolated subnetwork where outgoing traffic is blocked by the firewall to stop attackers from using compromised honeypot to attack other computers on the Internet. Since the gateway is configured as a layer 2 bridging device it requires at least 2 Network Interface Cards (NICs) to bridge the traffic from the Internet to the local network. In between these NICs the traffic is inspected and monitored to be visualized on the *Walleye* web interface. This web interface also serves as a administration platform to remotely configure the honeynet and the gateway.

Integration

The purpose of the Honeywall CDROM is to make setup and deployment of a honeynet with a central management interface easy and user-friendly. The honeynet project therefore provides an installation image which can either be used with a CDROM or with virtualization techniques to setup a fully monitored and secured honeynet. The image configures the gateway with snort monitoring and the web interface during installation process. The downside of the pre-build image is that honeypots are not directly included and must be managed separately. Also the system is quite old compared to T-Pot or MHN, which reflects in the lack of support and the outdated technology used. In some cases where this is neglectable Honeywall might be a valuable option considering the good documentation and the guided installation.

Information Collection

Since Honeywall was developed many years back and dropped support a few years later, recent honeypot developments are not supported. In general Honeywall is designed to

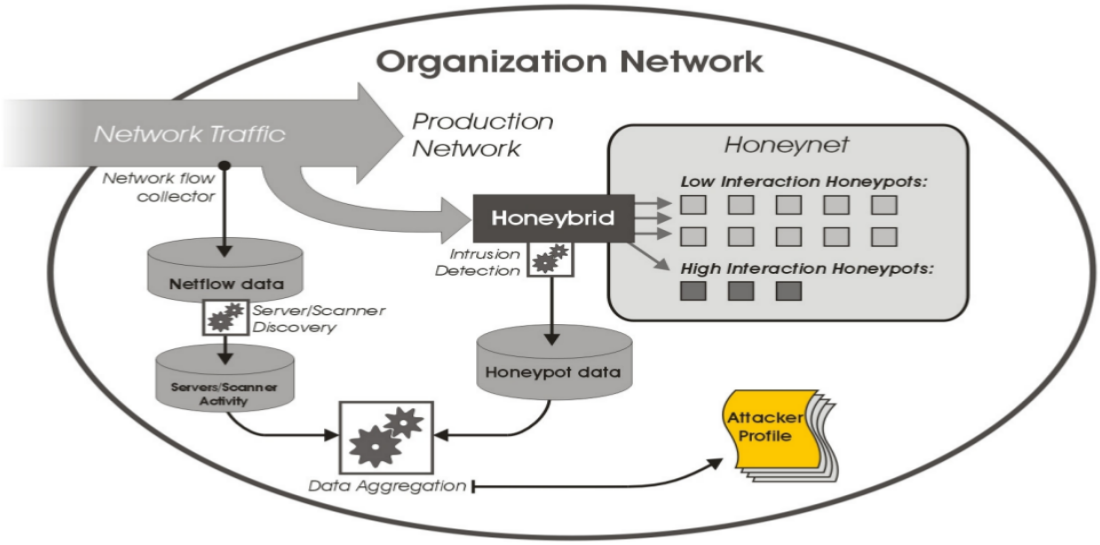

Figure 3.4: Honeybrid's architecture overview showing the interconnected modules

support all kind of honeypots behind the gateway, but honeywall cannot automatically collect application based event logs. The only automated log collection supported is for data collected with sebek [16]. Requiring sebek installed on each honeypot on the subnetwork reduces performance for the honeypots and can lead to fingerprinting possibilities for attackers to reveal the honeypots. Last but not least, no SIEM system is supported where collected data can be forwarded for real time analysis.

3.6 Honeybrid

Honeybrid is a hybrid honeynet which combines both, low and high interaction honeypots in a smart way. It can independently decide weather incoming traffic is routed to low interaction honeypots or high interaction ones. The main work was done by Robin G. Berthier as part of his dissertation [Ber09] at the University of Maryland. The idea behind Honeybrid was, first, to develop a highly scalable and highly interactive honeynet, and second, to adopt a more active approach on the honeypots when receiving illegitimate connections.

Architecture

As a hybrid honeypot, Honeybrid follows a quite different approach then the other honeypots and honeynets evaluated. It is described as a innovative technique to combine low and high-interaction honeypots in a smart way to reduce the load for high-interaction

[16] https://projects.honeynet.org/sebek/

honeypots. This is achieved by selecting only specific attacks hitting the low-interaction honeypots which will be redirected to high-interaction ones for further investigation. The overall architecture contains three parts: a gateway, a set of high interaction honeypots and a set of low-interaction honeypots. No central management interface is provided to analyze the collected threat data. Due to the specific design there are several limitations with respect to distributed deployment and custom modifications.

Integration

With support for virtualization, Honeybrid can be deployed either on a virtual instance or native on a gateway server. It requires a Debian host operating system but can be ported to other UNIX distributions with medium effort. The installation procedure is scripted, but Honeybrid needs to be build from source during the installation. Also the maintenance is an issue, because the project is not officially maintained and developed anymore and the documentation only consists of Berthier's PhD publication. No source code documentation or user manual was available at the time of evaluation.

Information Collection

More recent honeypot projects are not supported with the Honeybrid gateway. Projects like Glastopf or Cowrie, which are state-of-the-art honeypot projects might give more insight in special attack vectors, but Honeybrid only supports Honeyd[17] as low interaction honeypots and Nepenthes as high-interaction honeypots. On the other hand Honeybrid benefits from its decision engine and the ability to switch from low to high interaction honeypots, which enables high monitoring capabilities. Using syslog forwarding, it is also possible to integrate Honeybrid with a SIEM system.

[17] http://www.honeyd.org/

Chapter 4

Design

Following the evaluation of different honeynet managment systems the Modern Honey Network proved to be the most flexible and maintainable solution to be used in an enterprise environment. Therefore it will be used as the base framework for the following honeynet use case designs and implementations. In this chapter possible use cases for an enterprise environment will be discussed and detailed blueprints will be provided. Each use case was seleceted carefully with with respect to the most important threats to enterprise environments. Due to the interconnection between IT and OT in the last years and the ensuing risks for companies running critical infrastructure as well as production sites, both IT and OT use cases will be considered.

Various honeypot and honeynet use case designs have already been published, but most of them approach the design on an operational level [Anoc], [Sma]. They discuss high level threats such as external and internal threats or combinations with IDSs, but do not provide any low level configuration advice. These use case designs rely on existing honeypot software and already tested setups, without considering new strategies to build and deploy honeypots in IT environments. Especially corporate IT networks require specific designs with respect to the individual network setup and potential threats. Therefore new ways to design and build honeynets based on existing, as well as custom developed, honeypots have to be found. Even though the community behind honeypot development is quite active, only a few designs manage to leave development and are being published with a corresponding release. As a contribution this work propose a broad collection of use case designs with high and low level configuration advice for building and integrating these use cases in an enterprise network. Each of the following use cases is structured into four parts representing the blueprint: *Topology*, *Configuration*, *Detection* and *Presentation*. Table 4.1 explains the different parts and their purpose for the blueprints.

Topology	Hardware setup and network design
Configuration	Software requirements and configuration
Detection	Threat data being collected by the honeynet
Presentation	Presentation design for central management interface

Table 4.1: Blueprint section description

4.1 IT Use cases

With IT being an essential part of almost every company regardless of their economic field it becomes more and more important to secure the IT infrastructure and prevent criminals from taking control. Therefore five enterprise honeynet use cases, ranging from web application honeynets to APT monitoring, were designed to provide security mechanisms to the company's IT department.

4.1.1 Web Application Honeynet

Today every large company has at least one presentation on the internet. Either a self hosted static website or a dynamic web application with all the relevant information. These public domains are already facing a vast amount of attacks every day. Because they are easily accessible, they require a good isolation from the rest of the network to prevent possible attackers from compromising the whole network. Nevertheless, logging these attacks gives insights on how attackers try to compromise such internet facing machines and what are the potential weaknesses. With the first use case, an internet facing web application honeynet blueprint is given and all the configurations are explained.

Topology

Best isolation for a web application honeynet can be achieved by placing it inside a DMZ. There, strict firewall rules avoid any connection from the DMZ to the internal network. It must be ensured that the firewalls are set up correctly and if needed, applications only connect via VPN tunnel to the internal network. The honeynet itself is built up as an multi-tier architecture consisting of a web application honeypot (presentation layer), a MySQL database server (data layer) and the honeynet management server (business layer). Both the application honeypot and the database honeypot are situated in the DMZ. The management server should be setup behind the firewall on the internal network to avoid compromise. Inside the DMZ only the web application honeypot can communicate with the database, but not the other way round. Furthermore, both honeypots send the collected attack data via a secure VPN tunnel to the management server on the internal network.

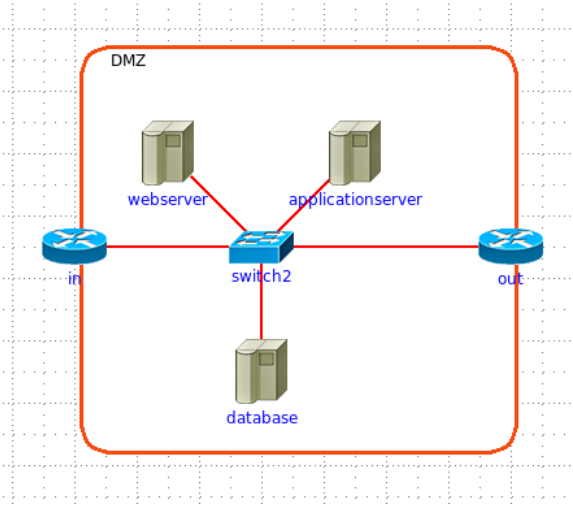

Figure 4.1: Network topology scheme for the web application honeynet use case

Configuration

For this blueprint it is recommended to use snare[1] whih tanner[2] as the main web application honeypot and OpenCanary[3] for the database. Snare is a web application honeypot with a separated vulnerability emulation backend (Tanner) and the ability to clone existing websites. OpenCanary is a low interaction honeypot and provides many options to emulate file sharing or database protocols. Operating system requirements for both honeypots recommend an Ubuntu Trusty Tahr (14.04 LTS) host system. To increase the range of the honeypot and to attract more attackers, the honeynet should be advertised. Possible advertisement could be done by adding DNS entries for the honeypots, embed links to the honeypots in the source code of the production site or add the honeypots to the *robots.txt*. Additional coverage can be achieved by a higher density of honeypots or IP addresses per honeypot. Translating multiple IP addresses to a single honeypot increases the possibility that attackers find their way to the honeypot.

Detection

Data collected with this honeynet configuration includes not only classic web application exploits like SQL injection, XSS or local/remote file inclusion, but also multi-stage attacks which origin from the web application honeypot and target the database. This is important since multi-stage attacks, which gain persistence on the database, are much more harmful than non persistent attacks against the web server. Moreover, the database honeypot collects data regarding all kind of exploits against the database which might not

[1] https://github.com/mushorg/snare
[2] https://github.com/mushorg/tanner
[3] https://github.com/thinkst/OpenCanary

be captured by the web application honeypot.

Presentation

The most important part of any honeynet is the possibility to visualize and present the collected data to the operator. With this web application honeypot not all the information for each attack should be shown initially. Therefore the raw data is stored in a database and the operator is presented with the most important fields:

- *Who*: source IP, user-agent
- *What*: honeypot IP, destination port, protocol
- *How*: payload (http packet body)

4.1.2 Darkspace

A darkspace is a fraction of the available IP range of a company which is eaten up by a single zero-interaction network resource. This fraction of the IP address space can be a whole 16 or 24 bit masked subnetwork routed to a passive monitoring and logging resource. Its main advantage over other low interaction honeypots is the collection of raw connection attempts on any port within the IP range. Without the active interaction it also significantly reduces the risk of getting compromised.

Topology

Recommended location for such a passive monitoring resource is similar to the web application honeypot inside a DMZ to reduce the risks if it gets compromised. If deployed internally it will only be able to detect connection attempts from within the network. It should be avoided to mix external and internal deployments by forwarding traffic through the DMZ to an internal darkspace. If deployed in the DMZ, collected data is submitted to the central management via a secure VPN tunnel.

Configuration

When configuring a darkspace honeypot, it is important to select the right size for the subnetwork. The more IP addresses from the overall IP range of the company go into the darkspace, the less addresses are available for production systems and self-hosted servers.

Therefore it is recommended to use a 16 or 24 bit mask [Tea] to separate the subnetwork. To track incoming connections the honeypot uses argus [4] and logs the packets to log files.

Detection

The goal for this honeynet is to capture unauthorized connections to IP addresses which do not belong to the production site of a company and automated scanning of the whole IP range of the company. If someone is scanning the IP range, this will apear on the data collected by the darkspace. Inspecting packets collected with the darkspace gives an overview of currently attacked ports and the related payloads. Another advantage of its unversial collection and logging is the abilty to capture unexpected unknown vulnerability exploits and self propagating malware.

Presentation

Detailed analysis of specific attack data has to be done manually, but data to trace the origin of the attack should be presented on the visualization interface. Important data to be presented on the web interface to the operator is:

- *Who*: source IP
- *What*: honeypot IP, destination port, protocol
- *How*: payload

4.1.3 Ransomware Detection

Ransomware exposes companies to an entirely new kind of threat. It is not the IT infrastructure and the software which gets exploited, but the human itself. Large phishing campaigns aim at distributing their malware, which, if executed, encrypts all the files on the victims computer and demands a ransom in form of bitcoins. Because for most ransomware variants no software bugs are used, there is no network level prevention like IDS rules. Without any real prevention against ransomware it is highly important to add security mechanisms in place to detect infections as soon as possible.

[4] http://www.qosient.com/ argus/

Topology

One way to address this issue is by setting up a file share on your network which does not contain any real data, but dummy files, and is hidden from client discovery. It is further mapped to all client machines to increase the possibility of detecting ransomware variants as soon as they try to encrypt this new share.

Configuration

This "fake" fileshare (honeyshare) can either be configured using Samba on a linux host, or as a standard Windows network share. Both offer the ability to log file system changes like read/write operations. On Samba the *full-audit* extention must be enabled and configured within `samba.conf` 5.1 and on Windows the `Windows Local Security Policy: File System Audit Policy`[5] can be used to monitor file system changes. The *Windows Local Securtiy Policy* can be set using the *Advanced Audit Policy Configuration* with Windows Server 2008 and later. To limit the amount of events logged to a resonable level, the honeyshare should be configured to only care about write operations, because those are the detection features if ransomware starts to decrypt dozens of files simultaneously. Furthermore each event is logged and can be send to the central management interface for visualization.

Detection

Yet, there is no working prevention for ransomware in general. It is only possible to detect ransomware activities as soon as it starts encrypting files on the compromised system. With detecting ransomware it is crucial to get the hostname of the compromised machine on the network and the user which is authenticated with that machine.

Presentation

As part of the visualization the hostname, the IP and the username should be instantly extractable. Optional the modified files can be enumerated and presented with each event. The following information is presented on the interface:

- *Who*: source IP, hostname
- *What*: fileshare name, files
- *How*: readwriteopen

[5] https://blogs.technet.microsoft.com/mspfe/2013/08/26/auditing-file-access-on-file-servers/

4.1.4 APT Detection

Probably the most dangerous threat for companies and their valuable assets are Advanced Persistent Threats. Besides their persistent presence on the network, they collect detailed information about the structure and topology of it for further lateral movement. They can stay undetected for a very long time before they extract the information they are looking for. This use case aims to detect *APT*s when they start moving laterally from one machine to another one by spreading user credentials on client machines which do not map to any real user at the domain. On windows clients *APT*s commonly dump such stored credentials from the LSASS[6] process and pass them on for authentication. This attack, commonly refered as "pass the hash"[7] and it's the most used method to move laterally on a compromised network. If the APT gets his hands on these *canary user* credentials and tries to use them for authentication with the domain controller, the domain controller logs the failed authentication request. For this use case it is assumed that the adversary has already compromised one computer and established a persistent backdoor.

Topology

For this blueprint an active directory based network including at least one domain controller is required. The domain controller authenticates users with the domain using *Kerberos* protocol[8]. All client machines on the network can connect to the domain controller and are granted access to fileshares and servers.

Configuration

First the canary user credentials must be generated and distributed to all client machines on the network. One method to do this on a Windows domain is by using the `runas /user:<Domain>\<User>` command[Bag15]. The `runas` command pushes a username:NTLM hash pair into the memory without communication with the domain controller. Logging on the domain controller for usage of this canary user can be configured through enabling *Audit Account Logon Events* for the domain policy and filtering events for *Failure Audit* on logon events (#533)[Mic] To collect the event logs from the domain controller on the central interface a logfile parser publishes the logged *Failure Audit* events via hpfeeds. The corresponding channel should be subscribed by the central management server.

[6] https://en.wikipedia.org/wiki/Local_Security_Authority_Subsystem_Service
[7] https://en.wikipedia.org/wiki/Pass_the_hash
[8] https://en.wikipedia.org/wiki/Kerberos_(protocol)

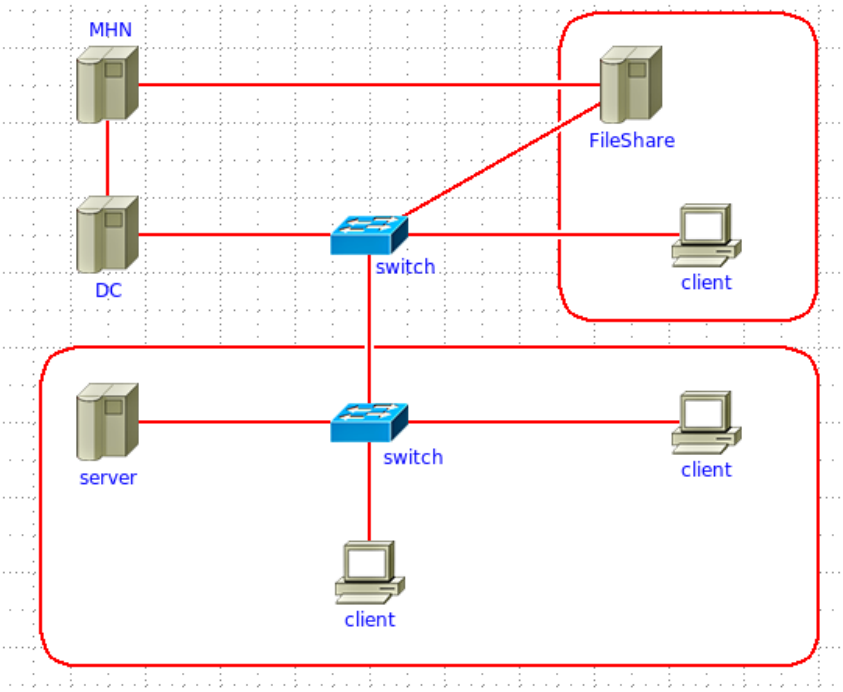

Figure 4.2: Network topology scheme for the ransomware and apt detection usecases

Detection

*APT*s on the network are dangerous and it is very important to detect their presence as soon as possible. Besides other security mechanisms to detect the intrusion and exploitation phase, detecting the lateral movement can be achieved with this canary user honeynet use case blueprint. If the attacker uses a hash-dumping tool like *mimikatz*[9] the canary user credentials are dumped and might get used. In any case when they are used to authenticat with the domain controller an failed authentication event automatically logged.

Presentation

For this use case the presentation on the central web interface should yield the IP or the hostname of the compromised machine and the IP for the lateral movement destination. Further the used NTLM hash might be visualized together with the username. The honeynet operator should get required information to instruct the emergency response team to react and investigate how the APT got in the network.

- *Who*: source IP, username

[9] https://github.com/gentilkiwi/mimikatz

- *What*: honeypot, destination IP
- *How*: canary user name

4.1.5 APT Monitoring

Following the use case design to detect *APT*s on the network, this advanced blueprint aims to gather further information about the threat and its potential targets on the network. What targets are of special interest for the adversary? Which tools are used and what vulnerabilities exploited during exploitation phase? To answer these questions the setup is based on the idea of spreading canary users to all client machines. These canary users get valid access to specific machines isolated from the rest of the network.

Topology

The honeynet is designed as a subnetwork of the company's network consisting of multiple honeypots. This subnetwork can be made as complex as needed to build a legit looking subtree of the network. Within this isolated subnetwork the attacker is allowed to authenticate with the machines as a regular user. Only the central management interface is located outside the isolated network and data is pushed via VPN-tunnel from the clients to the server.

Configuration

To deceive the attacker and observe his movements beyond the nodes on the subnetwork, each client and each server is configured as a honeypot. The deployed honeypots should consist of different types to cover the three core principles: *confidentiality, integrity* and *availability*. To build a believable honeynet inside the subnetwork it might help to add standard client machines as high interaction honeypots so that the overall setup cannot be differentiated from other subnetworks of the global network. Recommended honeypots to be deployed are listed in table 4.2. The isolation of the subnetwork requires a firewall set up to block outgoing traffic and a static IP for routing. To report the observed events to the central management server, the honeypots are configured to publish it via hpfeeds on a secure tunnel.

Detection

Added to the information gathered with the canary user use case, an additional honeynet as an isolated subnetwork of the company will also detect exploits used on these honey-

Confidentiality	Open Canary	SMB share
	Honeyd	SMB share
	Argos	high interaction
Integrity	Open Canary	MSSQL, MySQL
	ESPot	Elasticsearch
	Honeyd	MSSQL, MySQL
Availability	snare/tanner	web application
	Honeyd	HTTP
	DCEPT	domain controller
	glutton[10]	proxy mode honeypot

Table 4.2: Recommended Honeypots to cover the three core principles of IT security

pots. Therefore not only the presence can be detected, but also the tools, techniques and procedures. The broader the variety of deployed honeypots, the more distinct information can be learned about the threat and each interaction with one of the honeypots hold the attacker off attacking the production systems.

Presentation

As most of the recommended honeypots are traditional types, the visualization on the management interface should extend their collected threat data by combining it with the canary user information. A continuous observation of attacker activities should be possible without any blind spots.

- *Who*: source IP, username
- *What*: honeypot, protocol
- *How*: canary user name, payload

4.2 OT Use cases

In addition to the five IT use cases, two OT use cases have been designed. These use case are more important for companies with actual OT infrastructure like ICS or Supervisory Control And Data Acquisition (SCADA) systems and for research groups which are interested in attacks on these infrastructures. Another type of OT is critical infrastructure. Many people depend on the reliability of this critical infrastructure, such as power plants or water supply. If parts of such systems are caused to fail, this would have tremendous consequences. As already mentioned in chapter 1 attacks on critical infrastructure are not only a serious theoretical threat, but can happen at any time. The following use cases are

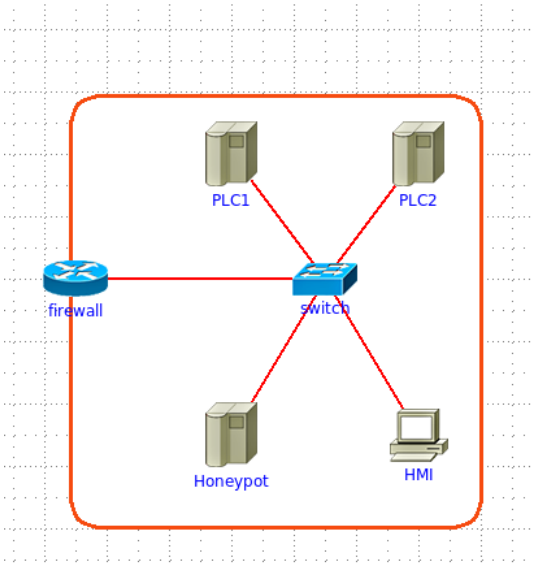

Figure 4.3: ICS network toplology scheme with an active SCADA honeypot

designed to capture active attacks against various systems as well as monitor rouge nodes within such networks. The blueprints are structured similar to the IT use cases.

4.2.1 SCADA/ICS Honeynet

A SCADA/ICS network should never be directly connected to the IT network. Large companies air-gap their SCADA system from conventional office IT systems. Another possibility is to connect the SCADA/ICS network to the DMZ with strict firewall rules. This isolation is crucial for every OT network and also holds for ICS honeynets. These SCADA honeynets are build to simulate common OT protocols and nodes, such as Programmable Logic Controllers (*PLC*s), which talk to a Human Machine Interface (HMI). This HMI is the operator's interface to control the sates of each node.

Topology

Therefore such a honeynet should consist of multiple PLC honeypots and at least one HMI. It is separated from the IT network by the DMZ. The DMZ ensures the critical firewall rules and the routing. The HMI which is logically connected to the *PLC*s can also be accessed from the DMZ.

Configuration

The *PLC*s are conpot deployments with custom configurations for different types of sensors. It is important not to use the default configuration as it has a known fingerprint and can be detected. Modbus[11], a common OT protocol can be emulated with conpot. Conpot can also be used as a proxy for a real PLC if available. In proxy mode it is possible to generate more reliable and legit data as with emulating a protocol. Furthermore, the HMI can be implemented with a simple web server serving static data on a web frontend. Both the PLC and the HMI transmit the collected logs to a central MHN server via a secure tunnel.

Detection

Conpot is a modern OT honeypot with the ability to emulate some of the most common OT protocols. Together with an HMI this honeynet aims at catching and deceiving attackers who manage to bridge over from the IT network. Any interaction with the *PLC*s or the HMI is recorded as malicious and can be evaluated live. In this way it is possible to capture malicious interactions with *PLC*s and the HMI. Included in the capture are also the payloads and potential 0-day exploits. Overall, this honeynet design detects intruders to the simulated OT network as soon as they interact with any of the honeypot nodes and collects all possible information.

Presentation

By presenting the hostname of the compromised machine and the destination of the malicious interaction, the operator is instantly aware of the threat. Additional payloads and protocols can be visualized as well and depending on the implementation some data may be add important information.

- *Who*: source IP, src PLC ID
- *What*: destination PLC ID, destination port
- *How*: command sequence, payload

4.2.2 Passive Field-Bus Monitoring

Another option to learn about what's going on within the OT network is to add passive monitoring nodes. It's different from the active honeynet approach but such passive mon-

[11] https://de.wikipedia.org/wiki/Modbus

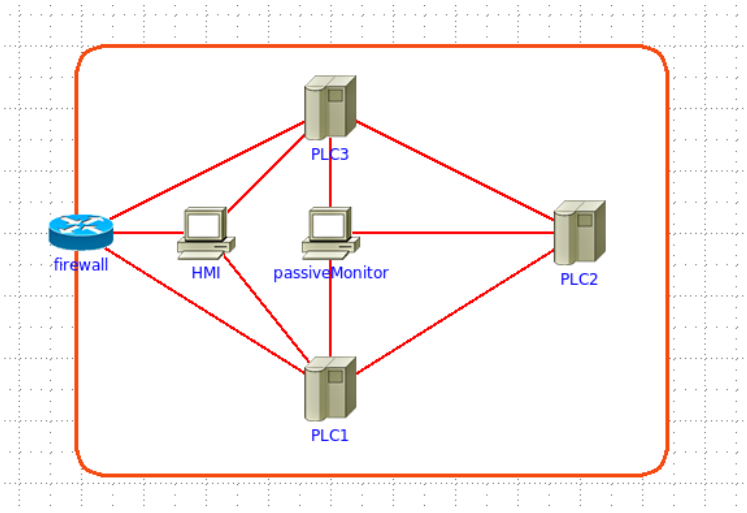

Figure 4.4: Network topology scheme for a passive ICS monitor sensor

itoring nodes do not require to set up a dedicated network. They can be plugged into any OT network and configured to listen on traffic broadcasted to specific ports. The advantage is the ability to detect already compromised nodes on the existing network.

Topology

Without the need to build a dedicated network, any already existing OT network can be extended with a passive monitoring node. This node is placed strategically on the network topology to be able to observe as much traffic as possible.

Configuration

The configuration for a passive monitor on the OT network is similar to the *darkspace* use case on the IT network. It is a listening only node, configured to log any command it observes on a specified port. The chosen port has to be related to the used OT protocol. For example TCP port 502 is reserved for Modbus. Since it is only passively listening on these ports it does not provide any form of interaction and only traffic which is broadcasted on the network or is directly targeting the passive monitoring node can be logged.

Detection

As already mentioned this use case should reliably detect malicious nodes on an exiting OT network. If one node is compromised and misbehaves or tries to manipulate other

nodes it will broadcast exploits to all nodes on the corresponding port. Additionally, the monitoring node can provide further information about unwanted interactions between *PLC*s.

Presentation

This blueprint recommends to collect informations such as src IP and src port together with the transmitted payload. Since it only passively monitors specific ports it won't collect much more information.

- *Who*: source IP, src PLC ID
- *What*: destination port
- *How*: payload

4.3 Honeynets & Use cases

Above, several novel use cases were discussed and blueprint designs provided. These designs will now be theoretically tested against different honeynet solutions from the evaluation. Thereby the goal is to identify the best fitting solution to use for implemented two selected use cases with a practical test environment. The comparison will partition the use case in *IT* and *OT*, similar to the design section. In the end a comparsion table shows a mapping of use cases and honeynets.

4.3.1 IT Use cases

The web application use case is probably the most common one and therefore all honeynet solutions are able to form the designed honeynet. Building a darkspace honeypot would work out of the box with CyberTrap and with some additional work with the MHN and Honeywall. To realise it with MHN or Honeywall the missing darkspace honeypot would need to be built and integrated with the management server. This is possible due to MHN's extensible design and Honeywall's design per se. Honeywall does not directly specify the honeypots behind the firewall, but inspects any incoming traffic regardless of the targeted honeypot. All other honeynets are not capable to realize a darkspace without overwhelming adaptions and extensions.

Ransomware detection via honeypot can be realised, based on the proposed design, by extending the MHN and Honeywall again. Similar to the darkspace, both require the honeypot to be built first, but can integrate it with the existing design easily. There are

	CyberTrap	MHN	T-Pot	SURFcert IDS	Honeywall	Honeybrid
Web Application	√	√	√	√	√	√
Darkspace	√	+	×	×	+	×
Ransomware det.	×	+	×	×	+	×
APT det.	+	+	×	×	×	×
APT mon.	+	√	√	√	√	√

Table 4.3: Mapping of proposed use cases to honeynet implementations with the degree of support categorized as: **supported** (√), **extendable** (+) and **not possible** (×)

only small adaptions with the web interface of the MHN to set up data presentation. With Honeywall the incoming file share accesses are automatically logged with the gateway if the fileshare is located behind the firewall.

CyberTrap and the MHN furthermore support APT detection with some additions. First, the honeytokens need to be distributed and logging on the corresponding domain controller (DC) enabled. Second, the honeytokens need to be integrated with the management server to provide the operator with the captured information. CyberTrap can be used directly on any Windows DC to log the authentication requrest of interrest. MHN requires to build a Samaba DC honeypot with hpfeeds integration to transmit data. Extending the APT usecase from detection to monitoring can be done with all evaluated honeynets since this is again based on the more common honeypots such as Kippo, Honeyd, Glastopf and so on. Depending on the honeypot selection CyberTrap and MHN will require additional extension to maximize the amount of information collected. The other honeynet solutions can be used in individual ways to perform similar to the aformentioned.

Figure 4.3 highlits the mapping of use case and honeynet visually by setting √ for supported use cases, + if additional work is neccessary and × if it is not possible to realize the use case design with the corresponding honeynet solution.

4.3.2 OT Use cases

In contrast to the five IT use cases, there are only two interesting OT use cases which need to be tested. The first, the SCADA/ICS honeynet ca be realised with the MHN and T-Pot out of the box, because both support conpot as a honeypot. They do not require any manual integration as with Honeywall. Honeywall can support an ICS, but needs to be connected with the honeypots. All the other solutions are not easily integrateable with any ICS capable honeypot.

The last OT use case, the passive field-bus monitoring, can be build using the MHN and CyberTrap. The others are not designed to run small passive monitoring nodes, or are too overloaded with firewalling or network setup. CyberTrap can convert any monitoring node into a honeypot and with MHN Honeyd nodes can be configured to run silently on the ICS network.

	CyberTrap	**MHN**	**T-Pot**	**SURFcert IDS**	**Honeywall**	**Honeybrid**
ICS Honeynet	×	√	√	×	+	×
Field-Bus Mon.	√	+	×	×	×	×

Table 4.4: Mapping of supported T use cases to honeynet solutions.

Figure 4.4 shows the mapping of OT use cases similar to figure 4.3 and uses the same legend. Supported use cases are marked √, improvements with + and unsupported use cases are marked with a ×.

4.3.3 Design Conclusion

By looking at the tabels 4.3 and 4.4 it can be seen, that the MHN is the only solution capable of realising all use case concept designs. Either via extension or out-of-the-box it supports both IT and OT deployments and the designed setups. T-Pot which is also very feature rich, misses out on 3 of 5 IT designs at one OT design. Therefore it cannot be considered further for implementing some of the designs. On the other hand, with CyperTrap only one IT use case, the ransomware detection design, and one OT cannot be covered. Due to CyberTrap's system design it is not very easy to manage and or extend, so it is also not useful for the procatical implementation. The others, Honeywall, SURFcert IDS and Honeybrid, are either outdated with respect to supported honeypot implementations or out of scope for implementing the proposed designs. Honeybrid's network for example is too complicated for a simple test setup and would not fit in the existing security landscape. Honeywall misses the automation to install honeypots and connect them with the management server and SURFcert IDS uses distributed sensors, but local honeypots and forwards the traffic through VPN tunnels. Because each of these violates at least one of the requirements for a useable honeynet solution, the MHN is the solution to work with.

Chapter 5

Implementation

To show the feasibility and the real value of the discussed use cases, a Proof of Concept (POC) implementation for two use cases was done. The selected use cases were a) the ransomware detection; b) the APT detection. Both are novel in their design and the provided blueprints. The POC implementations feature the recommended architecture and software from the blueprints and provide the data to a central MHN server. More details on the network setup, the development environment and the implementaion will be discussed in the contribution section 5.3.

This chapter will start with the introduction of the MHN as the used management framework with its advantages and disadvantages. The important parts to implement the use cases will be highlighted explicitly. Further the development environment and the used tools will be explained shortly. The main part of this chapter will be on the implementation of both POC implementations and how to integrate them with the MHN. In the end the improvements for data collection and the practical results of a testing phase will be discussed.

5.1 Modern Honey Network

As already mentioned, the MHN performed best with regards to the requirements defined in chapter 3. It was also the only system evaluated, which can integrate all use case design with the least work required. Therefore it was selected to be the honeynet managment framework for the blueprint implementations and test phase.

The Modern Honey Network is a distributed network of honeypots with a central management server which implements the honeynet logic. This management server consists of four essential parts: a hpfeeds broker, a mnemosyne data normalizer, a web app and a SQLite database. The hpfeeds broker subscribes to the honeypot channels and stores

all incoming events in the SQLite database. Mnemosyne also subscribes to the honeypot channels via hpfeeds, but normalizes the incoming events and pushes the resulting data to the web app's *attacks* interface. To normalize the events, each honeypot got its individual normalizer implemented within mnemosyne with python. The web application is a python web app which provides a high-level overview of attacks, as well as statistics for the individual honeypots. There are eight different views on the web interface:

- Dashboard
- Map
- Deploy
- Attacks
- Payloads
- Rules
- Sensors
- Charts

Each one provides a different feature to the operator.

The dashboard, seen in figure 5.1 highlights "TOP 5" statistics on events over the last 24 hours. This includes the number of attacks, the most attacked ports or sensors and event the top attacking IPs. The map provides geo-ip resolution for attacking IP addresses. This might be very interesting with respect to the countries where most attacks are coming from. The deployment view of the web interface provides honeypot deployment scripts. Via a dropdown selection the operator can select the honeypot to deploy and he will get a *bash* command which can be directly executed on the server where the honeypot will be installed. The command automatically pulls a script which registers with the management server and adds a new sensor to the sensors tab. The script further installs the honeypot with its prerequisites and starts it. If the operator want to adapt the deployment script, he can either edit the file on the server directly or edit the preview on the deployment view. Custom honeypots have to add their own install script to be selectable via the web interface. Next the attacks can be filtered on the attacks page of the web interface. There, filters can be applied for **sensors**, **honeypots**, **date**, and **IP/port**. This attack listing is actually only the listing of the normalized mnemosyne data and therefore it cannot be inspected in greater detail. Depending on the honeypot, more details can be seen on the next tab, the payloads tab. If the honeynet has a *snort* instance running as a sensor, the recorded payloads can be inspected on this tab. It lists the payloads with respect to their source IP, destination port and sensor together with a timestamp and a snort signature. Without a snort IDS, this view is unfortunatelly useless. Same holds for the next tab,

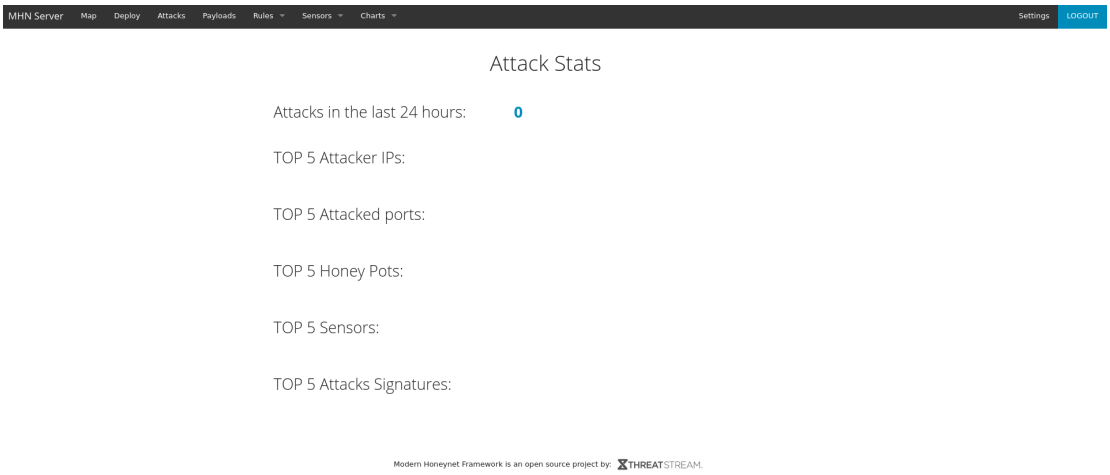

Figure 5.1: MHN's dashboard with the individual tabs on top.

because the rules managed on the rules view are snort IDS rules. They can be enabled, disabled and updated on this tab. Very usefull is the sensors tab, it lists the active sensors with their IP, the installed honeypots and the number of attacks per sensor. Also, sensors can be removed from the honeynet via the trash symbol in front of each sensor. There is also a subpage to add new sensor nodes manually, if they already existed or are migrated from another honeynet. The last tab on the web interface is the charts view. These charts are honeypot specific statistics about connections and interactions. MHN comes only with views for *Kippo* and *Cowrie* available. These charts might be the place to add further honeypot specific views for custom honeypots later on.

The following advantages and disadvantages will summarize the considerations taken into account for the selection of MHN over its competitors.

5.1.1 Advantages - Disadvantages

One of the key features of the Modern Honey Network is its central deployment system. The operator can use the web interface to generate a bash command which will automatically install the honeypot when executed on a remote machine. The only requirement is SSH access to the honeypot. All software dependencies will be installed automatically. The installed honeypot will further start and publish its data via hpfeeds to the central server. Also most of the state-of-the-art honeypots are supported already with deployment scripts. For new honeypots custom deployment scripts can be added and bash commands generated on the web interface.

Another good feature is the data visualization on the web interface. Attack trends of the last 24h are plotted on the dashboard. The collection of events can be viewed and filtered on the attacks page and for Dionaea and Kippo/Cowrie separate visualizations for their

data sets are available. The listing of all attacks presents only the normalized data stored by mnemosyne[1], which contains the source IP, the destination and the port. Since the web applications is build with a python Django web backend it can be extended easily. For example new visualization can be added or existing ones could be customized.

The only obvious disadvantage is the installation process. To start a fresh install the source code needs to be pulled and built on the target server. Dependencies need to be installed beforehand and there are quite a few steps during the process which can fail if software dependencies are missing or a timing issues occurs. Even though it automatically installs all the different parts, some customizations can be made during installation. First, it can be integrated with Splunk[2]. Second an ELK (Elasticsearch, Logstash, Kibana) stack deployment can replace the Django web app for enhanced data visualization.

One missing feature found with the evaluation is that there is no option to deploy honeypots remotely directly from the central server. A SSH connection to the honeypots has to be done manually and the bash command needs to be executed by hand. It would be better if this could be done automatically from the central server.

5.2 Development Environment

To build the POCs a set of development tools was used. Since the MHN is a Linux based server application, all in the honeynet participating nodes, were set up using Linux based operating systems. Especially Ubuntu Trusty Tahr (14.04 LTS) was used for the central server and the honeypots. The only exception was made for the Windows client machines which represented the company's standard roll out.

For the implementation of honeypots integrated with MHN, a local distributed setup was chosen, where the server and the honeypots where distributed over multiple virtual appliances. Figure 5.2 shows the network toplogical overview of the setup with all components. The clients, running *Windows 7* are connected to both, the domain controller honeypot for "authentication" and the file share honeypot. In addition the honeypots are connected to a MHN server instance which can be locally or remote. The honeypots only submit events via hpfeeds protocol to the MHN server.

[1] https://github.com/threatstream/mnemosyne
[2] https://www.splunk.com/en_us/products/premium-solutions/splunk-enterprise-security.html

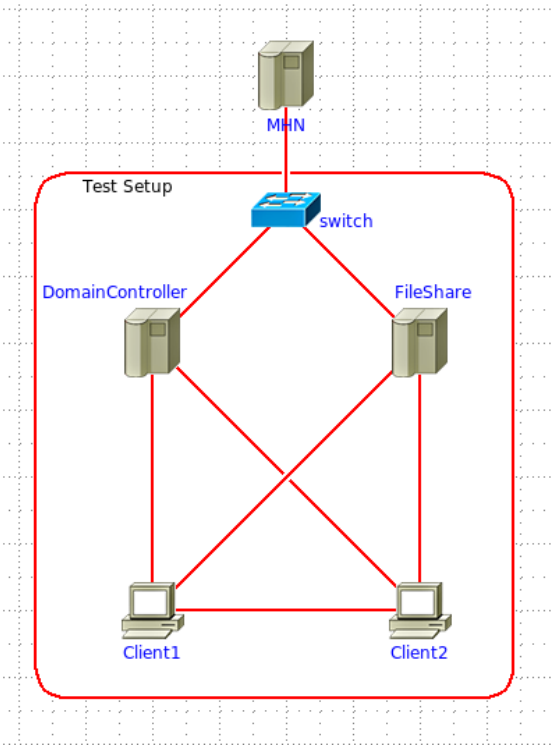

Figure 5.2: Network view of the honeynet setup

Following software tools were used to build the honeynet:

Linux Command Line

Bash was used as command line and is already included in Ubuntu Trusty Tahr.

python

Ubuntu Trusty Tahr includes python version 2.7.6 out of the box. This version is also the minimum requirement for the MHN. With olde versions of Ubuntu it is recommended to update python to version 2.7.6 before starting the installation, because for some parts it is required and will fail to install.

vim

Vi Improved (Vim) was used as the main editor to modify existing python source code files and to write new ones. Vim offers good python syntax highlighting and autocompletion.

No special plugins were used as part of the editor.

git

Like most other open source software the MHN is under version control with git. It's maintained on Github, a platform for open source projects with git version control. Modifications were made to a dedicated fork of Anomaly's own MHN repository. Additionally *mnemosyne* and *hpfeeds-collector* were forked for code changes and integration.

Samba

Samba is a software to use Windows protocols such as SMB/CIFS file, print, and login server for Unix. For both POC implementations a dedicated samba server was used to run the required protocols on an Ubuntu operating system.

5.3 Contribution

Both honeypots, the ransomware detection and the APT detection POCs, were build on top of a MHN deployment. Differences in what software was installed and used only depended on the blueprints. This work's main contribution is in the novel design of the presented blueprints and their first practical use to implement an enterprise honeynet. There is currently a lot of work going on in building honeypots with similar goals, but most of them are either extensions to old honeypots like Dionaea, or exerimental research projects with limited practicability. In contrast to these research projects, the implemented POCs are integrated into a full functional honeynet management and deliver all important data in real time. They are easy to integrate and require a minimum of setup to work with an existing MHN deployment. Furthermore, both are dedicated to a special use case and therefore they can simulate a more realistic system.

5.3.1 Ransomware Detection

Using honeypots to detect ransomware is a complicated task. As an automated malware most of its different variants are distributed via phishing emails and do not spread on common vulnerabilities. The only exception was *WannaCry*, which used an undisclosed SMB vulnerability. For most of the others it holds that the user has to open a malicious document which runs some sort of code to download the actual malware. This malware will then encrypt the personal files of the user and demand some ransom to be paid in

Bitcoins[3]. With this ransomware detection honeypot it is aimed to detect the phase where the malware starts encrypting files. More precisely, file modifications to a network file share are logged. This, indeed, can detect ransomware, because the malware will not only encrypt local files, but also files on network shares. Such a network share can be used to build a reliable ransomware honeypot by logging all file modifications consistently and parse the log files in realtime. The advantage of using a honeypot is the continuous logging of the source IP from where the ransomware is operating. This will help the Computer Emergency Response Team (CERT) to take immediate actions to recover the files and stop the malware from spreading further. To avoid false positive measurements, caused by users which accidently modify files on such a ransomware detection share, it is of importance to hide this share from regular users. No client in the network should be able to see the network share mounted as a remote drive using Windows explorer or Linux file manager. But still, under the surface the share is indeed mapped and can be accessed by all users using SMB. Since most ransomware variants attack Windows operating systems, they use the "`net use`" command to list mapped network shares. Due to simplicity reasons, the implemented POC ransomware honeypot is not hidden from client machines, because this would require a Windows Server based network share and modifiactions to the *Windows Group Policy*. Actually, the hiding is just a usability feature which has no effect on the monitoring capabilities and can be neglected for a working POC.

Implementation

A samba server was set up and configured, providing the necessary SMB protocol to share a folder of the server with all clients on the network. Listing 5.1 shows the relevant parts of the samba config file. There are 2 parts of the config file which are of importance for the setup. First, the `[global]` section contains the *log level* and *log file* specifier. The log level needs to be set to 6 or above to get the necessary information. The log file specifies the file path of samba's log file on the linux server. Additionally the *vfs_full_audit* options are added to this section. Second the `[fileshare]` section, which describes the location and the properties of the shared folder. This POC uses the */home/dev/Public* folder as root for the network share. It is set writable and public to be accessable and also browsable by all clients. *Valid users* are the *Administrator* and a domain user with name *client*. To enable file audit logging, which reports all changes to the files on the share, *vfs objects* is set to *full_audit*. The *full_audit* samba extension handles the file monitoring and uses syslog to collect the log events. These events are further stored by syslog in a `audit.log` logfile.

```
1   [global]
2     log level = 5 vfs:6
3     log file = /var/log/samba/samba.log
4
5     full_audit:prefix = %u|%I|%S
6     full_audit:failure = connect
7     full_audit:success =  write pwrite
```

[3]https://en.wikipedia.org/wiki/Bitcoin

```
8       full_audit:facility = local5
9       full_audit:priority = notice
10
11    [fileshare]
12      path = /home/dev/Public
13      writeable = yes
14      public = yes
15      browsable = yes
16      valid users = TEST\client Administrator
17      vfs objects = full_audit
18      create mask = 0777
```

Source Code 5.1: Samba config file

Within this shared folder some dummy files are placed which will be the honey in the honeypot. These files can be plain text files, images or documents such as `.pdf`, `.doc`, `.ppt`. For the honeypot to work properly it is important that the share is not left empty, because files are needed to monitor the read/write accesses with samba. Because of the concurrent encryption of files and network shares, it won't give any advantage if the amount of files is increased to slow down the encryption. The encryption for one share might take a little bit longer, but in paralell other shares and drives are encrypted. To monitor the file system changes samba's *vfs_full_audit* extension need to be installed and activated for the shared folder. This extension's configuration can be done on the `[global]` section in samba's config and contains the following attributes:

- `prefix`: Prepend audit messages with the specified string
- `failure`: List of VFS operations that should be recorded if they failed
- `success`: List of VFS operations that should be recorded if they succeed
- `facility`: Log messages to the named syslog facility
- `priority`: Log messages to the named syslog priority

To detect ransomware encryptions it is sufficient to enable logging for write and pwrite operations only. Other operations would only cause double occurences for each file that is modified. In addition to successful write operations, failed connection attempts are logged to see if some application tries to connect to the network share and fails. This may be due to an implementation mistake or a incompatible protocol such as SMBv3. Syslog's *local5* facility is used with prioity *notice* to copy the log entries to a dedicated logfile. With samba configured correctly, syslog requires to specify where to store the log entries collected from samba's *vfs_full_audit*. On Ubuntu **rsyslog**[4] is the preferred syslog implementation and was used for this setup. There are a few more configurations which

[4] http://www.rsyslog.com/

Database field	JSON data
timestamp	submission_timestamp
source_ip	source_ip
source_port	username——hostname
destination_ip	fileshare_name
destination_port	135
honeypot	samba
protocol	action

Table 5.1: Data mappings for mnemosyne

are required by rsyslog, but they are in no relation to the honeypot setup and can be found on Martin's blog[5].

To connect the honeypot with the central honeynet server, the *audit.log* logfile must be parsed and each event published on a hpfeeds channel reserved for this honeypot type. A new channel was created for this blueprint and it's ID is "samba.fileaudit". To parse the logfile the python script shown in A.1 was created. It uses `multitail2` to continuously collect events from the logfile and parses each line with a regex. If the line matches the given pattern, parts of it are published as json using client-side `hpfeeds`-side.

Server side, *mnemosyne* is used to collect the events published on all channels and to normalize the information. To enable mnemosyne to collect the events published on the "samba.fileaudit" channel, a new normalizer was added. The code shown in A.2 normalizes the events published by the ransomware honeypot and stores them to the database. Table 5.1 highlights the mappings between the json data published by the honeypot and the one stored. To store and also present the necessary information some fields are used with different data. Since the *source_port* is not relevant, this field is used to store the *username* and *hostname* of the compromised machine. The *destination_ip* store the *fileshare name* and the *protocol* field is used for the *action*. Other fields such as *destination_port* and *honeypot* are set constant since they do not provide any relevant information.

To deploy the honeypot from the central management server's web interface, a deployment script was added. The script A.3 installs the samba log file parser and its dependencies on a working samba server. This samba server was already working and the fileshare with the corresponding full_audit logging was set up in advance to the honeypot deployment. The script first registers the honeypot with the management server using the pulled `registration.sh`. Further it updates the system packages and installs `git, python-pip` and `supervisor`. Git is required to pull the code for the logfile parser from Github. Python-pip is required to install `virtualenv` and the requirements for the project specified in a `requirements.txt`. Last, supervisor is a system tool to automatically manage starting and stopping applications on the server. A config for the honeypot logfile parser is written to a config file in the working directory of the honeypot. The supervisor config

[5]http://linux-sys-adm.com/how-to-install-and-configuration-samba-server-on-ubuntu-server14.04-lts-step-by-step/

file for the logfile parser is added to `/etc/supervisor/conf.d/` as the last step. This enables the honeypot to start automatically after a reboot.

5.3.2 APT Detection

*APT*s are a serious threat to companies and governments. They require good skills and a lot of attention to be detected. Many companies simply don't have enough resources in IT personal to continuously monitor and analyze network traffic. A dedicated APT detection honeypot has the advantage that it captures only actions which are related to such a threat. In case of *APT*s, the honeypot captures malicious authentication requests with the DC. *APT*s commonly use a technique called "pass the hash" to move from one client machine to another one. Thereby credentials stored in memory are dumped and the NTLM hash is reused to login on another machine on the domain. Since most client computer not only store the current user credentials, but also credentials from remote logins, a compromised machine can leak priviledged account credentials. An APT can use this technique to find a way from an unpriviledged client to the domain administrator account with just a few hops on the network. The proposed honeypot aims to detect this *lateral movement*, by spreading fake user credentials on all client computers and watch authentication requests on the DC. If an APT finds and uses these fake user credentials, the honeypot will immediate capture the authentication request and store the necessary information including the IP of the compromised computer and the username. For this POC implemantation a Samba instance worked as the primary DC. The clients were standard Windows 7 Enterprise installations. The drawback of the POC implementation is that the information retrieval from Samba's log file noisier than with a DC running on Windows Server. With a Windows DC failed login attempts can be logged clean by enabling *Audit Account Logon Events* for the domain policy.

Implementation

The used Samba server acting as the DC was set up manually on a virtual machine running Ubuntu Trusty Tahr. The VM was configured with one core and 4 GB RAM. Samba itself was installed from the Ubuntu repository, which provides version 4.3. To provision Samba as a DC, the `samba-tool domain provision` command was used. Listing 5.2 highligts the full command used to provision the DC. Samba's internal dns backend is specified with the `--dns-backend` parameter. With `--realm` the full qualified domain name is set to *test.honey.lab* which implies that *test* is also the domain specification with the `--domain` parameter. Last the `--adminpass`, which defines the password for the domain administrator, is set to *Passw0rd*.

```
1    samba-tool domain provision --server-role=dc --use-rfc2307 \
2    --dns-backend=SAMBA_INTERNAL --realm=TEST.HONEY.LAB --domain=TEST \
3    --adminpass=Passw0rd
```

Source Code 5.2: Samba-tool provisioning command

By provisioning samba, a `smb.conf` as shown in A.4 is automatically created. There are some parts which are resbonsible for the DC and some which are general. The DC specific entries contain the `workgroup`, `realm`, `server role`, `dns forwarder`, `idmap_ldb:use rfc2307` and different `auth` methods. Most of them are related to the parameters provided to the provisioning, the others are specific to the Samba version. Important for the DC is the `server role`, which is set to "*active directory domain controller*" and the `realm`. The realm represents the full qualified domain name of the company's active directory domain. A `logon script` is set to run when a client authenticates with the DC. There are two more sections apart from the `[global]` section: The `[netlogon]` and the `[sysvol]`. Since both are without effect to the POC implementation, the default values were accepted.

The server was configured with a static IP address and its own IP for the primary dns-resolver. Furthermore it was added to the domain by specifying `search test.honey.lab` and `domain test.honey.lab` in Ubuntu's `/etc/resolv.conf`. With a working DC and DNS backend a "*client*" user was added via the `samba-tool`. This client was the primary user for the Windows clients. All Windows clients executed a logon script after booting which pushed the fake user credentials into the memory. This "canary user" was the objective of interest for the DC honeypot.

To convert the standard DC into a dedicated honeypot a python logfile parser was written to capture logon events to the DC. If a failed authentication attempt including the *canary user* credentials occurs in the samba log, the honeypot publishes it on a "samba.events". Listing A.5 shows the source code for the parser. It is very similar to the ransomware honeypot parser A.1. The `regex` is adjusted to the *samba.log* and it uses a different channel to publish, but the general structure is the same for both python scripts.

To have the threat data available on the honeynet management web interface, mnemosyne is extended similar to the ransomware honeypot implementation. Table 5.2 shows the mappings between published hpfeeds data and the database fields to store the information. For this POC implementation both destination fields are set to the same value since only the `destination_ip` is visualized and for remote logon to another network computer the port can be neglected. The honeypot is primarily interested in the *username*, stored as `action`, the *destination* and the already compromised machine (*source_ip*). The stored information is furthermore presented on the web interface.

To provide a possibility to deploy the honeypot to any Samba DC setup, a deployment script was added to MHN's deployment list. This script, shown in Listing A.6, is almost identical structured as the one for the ransomware honeypot. It registers the honeypot with the central server; updates the system packages and installs dependencies such as `git` or `python-pip`; it creates a virtualenv for the python logfile parser and creates a config file for it. Similar to most of the other honeypots provided by the MHN this one also utilizes *supervisor* to run in background. Supervisor therefore handles `autostart` and `autorestart` if the honeypot should crash.

Database field	JSON data
timestamp	submission_timestamp
source_ip	source_ip
source_port	source_port
destination_ip	destination
destination_port	destination
honeypot	DomainController
protocol	username

Table 5.2: Data mappings for mnemosyne

5.4 Experimental Results

To show not only the theoretical design and feasibility for such honeynet deployments some tests were executed to test both POC implementations. The tests were performed independendly and focused on the main targets of the honeypots. The goal was to check if the data presented on the honeynet web interface provides enough information to react and recover from the related incident. Both POC implementations were deployed on the same MHN honeynet. The test honeynet was setup on VMs based on Ubuntu Trusty Tahr and Windows 7 Enterprise clients.

5.4.1 Ransomware Detection Honeypot

For the ransomware honeypot one of the clients was empowered with a Powershell script to encrypt all files on connected network shares. The ransomware honeypot network share was mounted, because it was not explicitly hidden from the clients. The Powershell script connected to the network share and encrypted the files on the share to cause filesystem changes which should be captured by the honeypot. Checking the central management web interface the following data per event should be listed:

- **Date**: Submission timestamp
- **Sensor**: Name of the sensor
- **Country**: Country flag
- **Src IP**: $< Domain >\backslash < username >:< Sensor >$
- **Dst port**: 135, SMB port
- **Protocol**: file operation
- **Honeypot**: Honeypot name

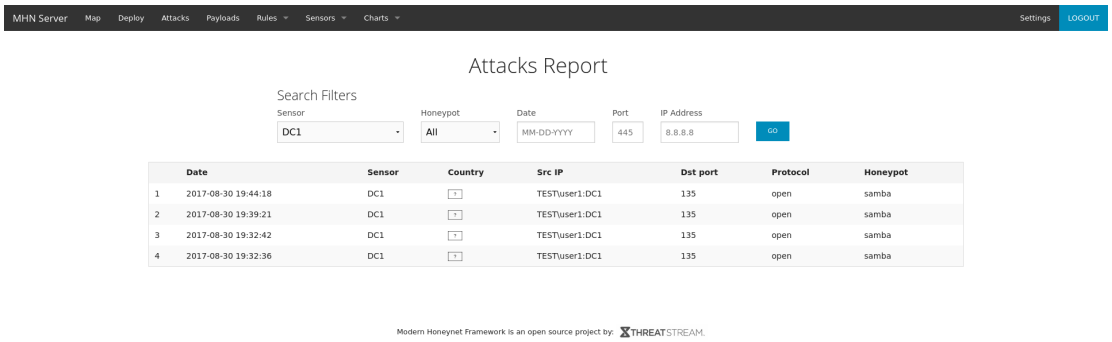

Figure 5.3: MHN's attack report page showing events of unauthorized access to the honeypot fileshare

Figure 5.3 shows a collection of event entries presented on the "Attack Report" page of the web interface. The POC implementation for this honeypot does not provide an extension to the existing table, but misuses some fields to print the required information.

5.4.2 APT Detection Honeypot

To test the APT honeypot POC implementation one of the Windows 7 clients was intentionally compromised with a Remote Access Trojan (RAT). This RAT was created with *Powershell Empire*[6], a powerfull post-exploitation framework which can also create RATs. After successful connecting back to the *Empire* server, an agent was downloaded and commanded to gain higher privileges. With these local admin privileges mimikatz was executed remotely to dump the stored credentials. Since the compromised client has authenticated with the domain, it had run the logon script which pushes the fake credentials into memory. Therefore mimikatz also dumped the fake user credentials. For this test only these fake "canary user" credentials were used with a *pass-the-hash* attack performed with the corresponding Empire module. Checking the honeynet web interface on the central management server the following data per event should be listed:

- **Date**: Submission timestamp
- **Sensor**: Name of the sensor
- **Country**: Country flag
- **Src IP**: IP address of compromised machine
- **Dst port**: Host and domain info

[6]https://www.powershellempire.com

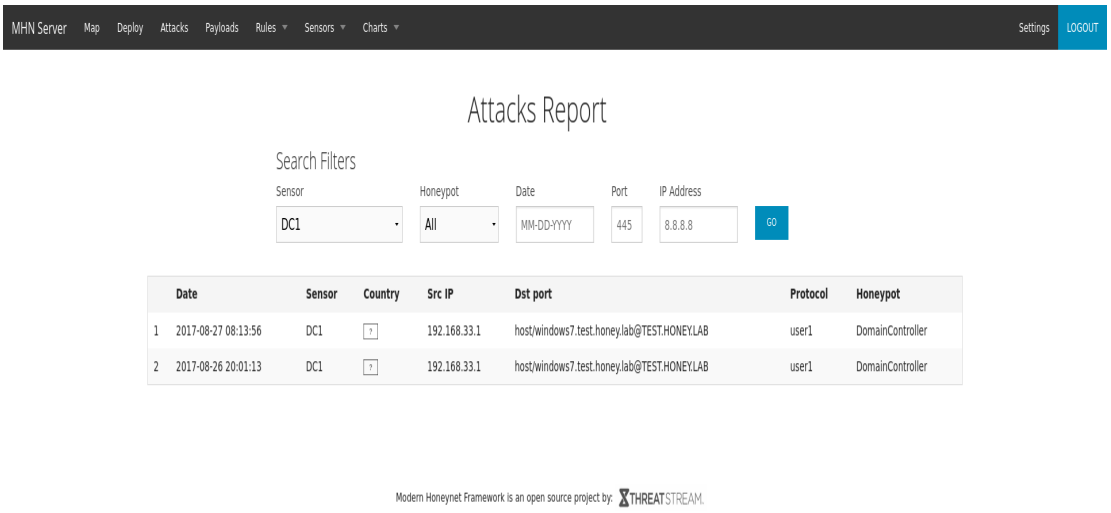

Figure 5.4: MHN's attack report page listing captured pass-the-hash authentication requests with a *honeyhash* from user *user*1

- **Protocol**: canary user's username
- **Honeypot**: Honeypot name

Similar to the presentation of the ransomware honeypot data, the APT honeypot implementation also misuses some table fields to show different data. The test results for unauthorized uses of the "canary user" credentials are shown in figure 5.4.

5.5 Experiences gained

By performing research and active work in the field of defensive security methods, such as honeypots and honeynets, a lot of first hand experience has been gained. From conceptual design to practical implementations everything was covered to some degree and increased the author's knowledge on these topics significantly. By performing research, designing use case blueprints and integrateable implementations, each step to build a custom self-made honeynet could be analysed and lessons learned. Within the following section, these experience will be shared with the reader to help develop their own honeynet and honeypots. The experiences will be bundled according to the steps of development:

- Use Case

- Design
- Implementation
- Analysis

It might be noted, that the following experiences were gathered by the author while working on this topic and reflect his personal opinion which might differ from that of others.

5.5.1 Use Case

Before an actual honeynet can be developed, the threat scenario and the contained systems have to be specified. This requires a good knowlege of the underlying network as well as the services offered. It is of importance that the final honeynet looks similar to the production environment and mirrors the same services. This provides better understanding of incoming attacks and exploits are more relevant to be captured with systems used also in production. There is no sense in providing services with the honeynet which does not occur on the production environment. Besides the similarity to the production network, the honeynet should be kept as simple as possible. This means that there is no gain in providing dozens of servers and clients, but only a few fully monitored honeypots on the honeynet network. The most important systems should be kept together with just a few dummy clients to ensure the legitimacy of the network. Furthermore the different attack vectors have an important influence on the use case of the honeynet. As shown with the different use cases in the *Design* chapter, the resulting networks differ a lot. With an external facing honeynet different services have to be provided than with an internal only honeynet. By working out a strategy for the corresponding honeynet, a goal will be defined. Each honeypot has to serve the purpose of collecting specific threat information helping to secure the overall network as well as individual applications. This information has to contain at least 1) *source IP* 2) *destination IP* 3) *port* 4) *protocol* 5) *payload*. Depending on the protocol and the attack additional information might be neccessary.

5.5.2 Design

To design the system which fits best, some essential steps must be taken. At first, the goal of the design is to come up with some sort of description of the final solution. This can either be a graphical design, a textual description or a component diagram. Either way, it must define the essential components and their connections. Furthermore, the software and hardware used must be defined explicitly. Having the design document should enable any developer to build the described system. By this design the actual honeynet, according to the defined use case and the selected threat of interest is becoming more clear. It will also specify strict requirements concerning *network*, *hardware* and *software* to use for the implementation. As with every honeynet, the design process should focus on

maximizing the threat information output. This means, that neglecting performance for better intelligence is tolerated to some degree. The implicit requirements for the different design levels are discussed in more detail in the following.

Network specific requirements define the topology of the honeynet as well as its location. Depending on these requirements different setups and network configurations can be made. A honeynet can be internal or external facing. It can be connected to a central switch, providing its own subnetwork or in a DMZ denying attacks from hitting the internal network. Additionally, firewalls might be considered with the design to ensure that compromised machines in the honeynet cannot attack machines outside the honeynet. One good example therefore would be the *Honeywall CDROM* honeynet gateway from the *Honeynet Project*. Honeypots as sensor nodes can be located locally on the same system or on the local network and also on any reachable network anywhere. With such network requirements a connection between all parts of the honeynet is ensured.

Hardware requirements such as dedicated servers or virtualization techniques should also be defined with the design of the system. Using virtualization is more resource friendly, but depending on the software and the use case dedicated hardware might also be required. An adversary on the honeynet should not be able to identify a sensor honeypot as a virutal machine when he is expecting a client computer or an ICS. Server honeypots can normally be virtualized to ensure isolation. Honeypots acting as end-user systems should be installed on dedicated hardware to make the system look legit for an adversary.

Software requirements on the other hand define the actual system components such as honeypots, protocols, analysis server and so on. Any software component from central management with web interface to honeypot sensor software and potential VPN tunneling should be defined with detail. Therefore, not only the existing software components are specified, but also missing parts, such as required sensors, servers or protocols must be designed. A good component design should cover the programming language, the interaction protocols and the structure for the threat information collected. With the software requirements the method of user-interaction is chosen. The user, so the operator of the honeynet, should be able to interact with web interfaces from the management server, or get the collected information into a SIEM system for analysis.

5.5.3 Implementation

Now, with the finished design it should be straight forward to implement the system. If any question regarding the tools, components or setup is unclear, a redefinition of the design is required. Depending on the use case and the resulting design most honeynets are built as distributed systems around a central management server. This management server does not neccessarily have to be located on the same subnetwork as the sensor nodes, but can be placed strongly guarded by firewalls, on the internal network. The sensor nodes are using honeypots such as **Kippo**, **Glastopf**, **Conpot**, **Dionaea**, and **Honeyd** to deceive attackers and capture information about their actions and malware.

Distributing these honeypot sensor nodes across any company network ensures the best coverage to catch attacks. Companies or institutions with tighter network structures might place the honeypots locally on a range of servers, or even all on the same server, and forward traffic from remote deployed sensor nodes via VPN tunneling. By tunneling the traffic, it is important to keep the source address, because otherwise this information is lost for analysis and no geo-ip resolution can be made. This second approach is also known as *honeyfarm*.

Besides using existing honeypots to capture different attacks it might be required to implement custom honeypots for special needs. These custom implementations can be build using any programming language supported on the host system. The only requirement for such custom implementations is to standardize the information stream directed to the analysis and management server. With the design the used protocol should have been specified, so this protocol should be used within the honeypot to transmit collected threat data to the central server. An example of such a protocol is **hpfeeds**, used by the MHN to publish data on different channels. The server on the other side subscribes to these channels and stores the data in a local database.

This central server is the key part with every honeynet as it should provide management for the sensor nodes and the honeypots and offer possibilities to inspect incomming attacks by analyzing the collected information. By designing a custom management server it is recommended to include a web interface where the data can be listed, filtered, and inspected with good detail. *T-Pot* includes a *elasticsearch, logstash, kibana* setup, which is a modern data visualization plattform offering many features to filter data and plot it. This is also the author's recommendation since it runs out-of-the box and there exist client implementations to push data to *elasticsearch* with almost any programming language. Apart from data visualization, the management server should also offer automated deployment for distributed sensors and honeypots. If the operator of the honeynet is required to deploy dozens of honeypots through the network, it might not be efficient to install every single one by hand. Therefore, with an automated deployment the operator can deploy new honeypot by providing the destination IP and the credentials. To automate the further installation process, there are different approaches recommended: 1) **Ansible**[7] or 2) SSH with deployment scripts. The simpler the automated deployment is, the more honeypots can be deployed and managed by the operator of the honeynet. By including the automation process in the design concept, it is ensured that the overall system can be kept simple and extendable as well. Most of the modern honeynet implementations use *Python* or *Go* as their primary programming language since both languages offer sufficient server side frameworks and come with the neccessary feature set to realise such distributed systems. In the author's oppinion the MHN is a great example for modular and structured code, even though it lacks the ability to automate the honeypot deployment.

[7] https://www.ansible.com/

5.5.4 Analysis

Analysis of the collected threat information is the first real performance indication for any new honeynet. The honeypot performance is well known from other implementations, but the performance of the interconnected sensors, honeypots and the management server can only be evaluated by capturing real attacks and deceiving attackers. The honeynet should run at least 30 days with the complete setup to gather enough threat data for analysis. Large companies might want to integrate the honeynet with existing SIEM systems to combine it with data from *IDS*s or other sources. Nevertheless, it should be kept in mind that more data also blurs the details of individual attacks. Without artificial intelligence to analyse the incomming data, it is still the operator or an analyst who has to look through all the information to find potential unknown attack vectors or vulnerability exploits. The data should therefore be precise and not overwhelming any interface with doublicated entries or false positives. Applying filters, based on honeypots and data of interest, can help to find the source of the attack and its course.

5.6 Identified Threats

Over the course of the work two major threats have been identified: *a)* ransomware and *b) APT*s. Ransomware is probably the most used type of malware at the moment and does not only cause problems to private users, but also to big companies with dozens of systems locked down after an infection. It cannot be preveted before starting to encrypt files, and even then it's difficult to distinguish ransomware from normal user activities such as copying or moving a lot of files. A honeypot, on the other hand, sees every connection as malicious and does not produce false positive events. By designing and implementing a honeypot to detect ransomware, it was shown that the idea of honeypot can also be adopted to a less interactive, but more automated kind of threat.

The second threat, *APT*s are far more interactive and difficult to detect. They can stay stealth for a long time before taking any action and if they do, they move slow and carefully. A traditional honeynet featuring state-of-the-art honeypots might catch some of these *APT*s, but the goal for this work to close the gap and focus on a more interesting part of their process. The *lateral movement*. In this work a domain controller honeypot was proposed which utilizes fake "canary user" credentials as honeytokens to lure an APT to use these credentials for authentication. Every potential use of these honeytokens is recorded by the DC and deemed to be malicious. The implemented POC showed that a DC can be extended to act like a honeypot for honeytokens by defining rules to identify malicious events.

Chapter 6

Conclusion and Future Work

The purpose of this thesis was to show that the idea of using honeypots and honeytokens is not only limited to interactive applications and protocols such as *SSH* or *HTTP*, but can also be adapted to a wider range of use cases. At the moment honeypots see a growth in development, but most of them provide limited usability and applicability. One severe limitation to honeypots is the missing integration to advanced honeynet frameworks such as the Modern Honey Network. In this work new use cases for honeypots and honeynets within an enterprise environment are discussed and implementation concepts are provided. The goal was to extract threats which are of danger for global companies and design honeynet extensions which can detect and inspect action performed by such threats.

To complement the honeynet research with practical applications for the implemented use cases, a real world enterprise scenario was assumed. By providing such a scenario, the cornerstone for future honeynet use cases and implementations was laid.

6.1 Future Work

Due to time and network limitations the implemented use case blueprints were only Proof of Concepts. They showed the feasibility and the basic concept, but for real production use, they should be tested on a real company network. The POCs diverged from the recommended setup in the blueprints by setting up a Samba instead of a Windows server. So migrating from Samba on Ubuntu to a Windows server should be the first step with both implementations. The ransomware honeypot could be deployed using Samba, but it is recommended to move it to a Windows server too. The DC for the APT honeypot should definitly be migrated to a Windows active directory DC, as it provides better logging capabilities than Samba. In addition, the deployment for both use cases should not rely on existing Samba installations, but should take care of installing the correct solution for either Windows or Linux DCs.

The next step could be further implementations of the other proposed use cases such as the *Darkspace*, the *Web application honeypot* or the two *OT use cases*. Especially the OT use cases should be implemented to enhance the security of OT networks and connected systems. These systems are relatively new and security hasn't bee their top focus, so experts expect more attacks on OT emerging within the next few years.

As honeynets can grow quite fast with the number of deployed honeypots it is essential to use a honeynet management system such as the Modern Honey Network. For this reason the proposed use cases should be integrated or integratable with it. To ensure good data visualization the general attack report page could be improved, or custom visualization pages added similar to the *Dionaea Malware* report page. These custom visualizations are not required for all of the proposed use cases, but for *Ransomware* and *APT* detection this could increase the amount of information presented to the operator.

In general Advanced Persistent Threats are the most serious threat to companies and their "crown jewels". Based on this research about Advanced Persistent Threat detection and observation using honeypot technology the author of this work recommends further investigation in advanced techniques to capture post-exploitation activities or even detect the initial exploitation.

APPENDIX

Appendix A

Source Code

A.1 Ransomware Detection Honeypot

```
#!/usr/bin/python

import multitail2
import hpfeeds

import sys
import datetime
import json
import hpfeeds
import logging
import re

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

ch = logging.StreamHandler(sys.stdout)
ch.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - \
  %(message)s')
ch.setFormatter(formatter)
logger.addHandler(ch)

def parse(line):
    regex =
    r'\^(?P<timestamp>\S{3}\s+\d+\s+\d{2}:\d{2}:\d{2})\s+ \
      (?P<hostname>\S+)\s(?P<flag>\w+):\s+ \
      (?P<username>\w+|\d+)\| \
```

```
28          (?P<source_ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\| \
29          (?P<fileshare_name>\w+|\d+)\|(?P<action>\w+)\| \
30          (?P<status>\w+)\|(?P<filename>\S*)'
31          match = re.match(regex, line)
32          if match:
33              res = match.groupdict()
34              for name in res.keys():
35                  if not res[name]:
36                      del res[name]
37              return res
38          return None
39
40      def hpfeeds_connect(host, port, ident, secret):
41          try:
42              logger.info('{0}, {1}'.format(ident, secret))
43              connection = hpfeeds.new(host, port, ident, secret)
44          except hpfeeds.FeedException as e:
45              logger.error('feed exception: %s'%e)
46              sys.exit(1)
47          logger.info('connected to %s (%s:%s)'%(connection.brokername, \
48            host, port))
49          return connection
50
51      def main():
52          cfg = {
53              'host' : '',
54              'port' : 10000,
55              'channel' : '',
56              'ident' : '',
57              'secret' : '',
58              'tail_file' : ''
59          }
60
61          if len(sys.argv) > 1:
62              logger.info("Parsing config file: %s"%sys.argv[1])
63              cfg.update(json.load(file(sys.argv[1])))
64
65              for name,value in cfg.items():
66                  if isinstance(value, basestring):
67                      cfg[name] = value.encode("utf-8")
68          else:
69              logger.warning("Warning: no config found, using default values for hpfeeds server")
70          publisher  = hpfeeds_connect(cfg['host'], cfg['port'], cfg['ident'], cfg['secret'])
71
72          tail = multitail2.MultiTail(cfg['tail_file'])
73          for filemeta, line in tail:
74              record = parse(line)
```

```
            if record:
                publisher.publish(cfg['channel'], json.dumps(record))
                logger.debug(json.dumps(record))
        publisher.stop()
        return 0

    if __name__ == '__main__':
        try:
            sys.exit(main())
        except KeyboardInterrupt:
            sys.exit(0)
```

Source Code A.1: Ransomware detection collector.py log file parser

```
import json

from normalizer.modules.basenormalizer import BaseNormalizer


class SambaFileaudit(BaseNormalizer):
    channels = ('samba.fileaudit',)

    def normalize(self, data, channel, submission_timestamp, ignore_rfc1918=True):
        o_data = self.parse_record_data(data)

        if ignore_rfc1918 and self.is_RFC1918_addr(o_data['source_ip']):
            return []

        session = {
            'timestamp': submission_timestamp,
            'source_ip': o_data['username']+':'+o_data['hostname'],
            'source_port': o_data['source_ip'],
            'destination_ip': o_data['fileshare_name'],
            'destination_port': 135,
            'honeypot': 'samba',
            'protocol': o_data['action']
        }
        relations = {'session': session}
        return [relations]
```

Source Code A.2: mnemosyne normalizer for samba's file_audit events

```
#!/bin/bash

if [ $# -ne 2 ]
```

```
4       then
5           echo "Wrong number of arguments supplied."
6           echo "Usage: $0 <server_url> <deploy_key>."
7           exit 1
8   fi
9
10  server_url=$1
11  deploy_key=$2
12
13  wget $server_url/static/registration.txt -O registration.sh
14  chmod 755 registration.sh
15
16  # Note: this will export the HPF_* variables
17  . ./registration.sh $server_url $deploy_key "samba"
18
19  echo "deb http://en.archive.ubuntu.com/ubuntu precise main \
20  multiverse" | sudo tee -a /etc/apt/sources.list
21
22  apt-get update
23  apt-get -y install git python-pip supervisor
24  pip install virtualenv
25
26  # Get the hpfeeds-collector source
27  cd /opt
28  git clone https://github.com/x4mp/hpfeeds-collector.git
29  cd hpfeeds-collector
30
31  virtualenv env
32  . env/bin/activate
33  pip install -r requirements.txt
34
35  cat >> hpfeeds-collector.conf <<EOF
36  {
37    "host": "$HPF_HOST",
38    "port" : $HPF_PORT,
39    "channel" : "samba.fileaudit",
40    "ident" : "$HPF_IDENT",
41    "secret" : "$HPF_SECRET",
42    "tail_file" : "/var/log/samba/audit.log"
43
44  }
45  EOF
46
47  # Set up supervisor
48  cat > /etc/supervisor/conf.d/samba-fileaudit-collector.conf <<EOF
49  [program:hpfeeds-collector]
50  command=/usr/bin/python /opt/hpfeeds-collector/collector.py \
```

```
/opt/hpfeeds-collector/hpfeeds-collector.conf
stdout_logfile=/var/log/audit-collector.log
stderr_logfile=/var/log/audit-collector.err
autostart=true
autorestart=true
redirect_stderr=true
stopsignal=QUIT
EOF

supervisorctl update
```

Source Code A.3: MHN ransomware honeypot deplyment script

A.2 APT Detection Honeypot

```
[global]
        workgroup = TEST
        realm = TEST.HONEY.LAB
        netbios name = DC1
        server role = active directory domain controller
        dns forwarder = 10.0.3.2
        acl allow execute always = yes
        idmap_ldb:use rfc2307 = yes
        ntlm auth = no
        lanman auth = no
        client ntlmv2 auth = yes

# Logon script
        logon script = logon.bat

# Logging
        log level = 5
        log file = /var/log/samba/samba.log
        max log size = 5000


[netlogon]
        path = /var/lib/samba/sysvol/test.honey.lab/scripts
        writeable = no
        browsable = no

[sysvol]
        path = /var/lib/samba/sysvol
```

```
        read only = no
        browsable = no
        writeable = no
```

Source Code A.4: Samba DC config file

```
#!/usr/bin/python

import multitail2
import hpfeeds

import sys
import datetime
import json
import hpfeeds
import logging
import re

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

ch = logging.StreamHandler(sys.stdout)
ch.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - \
  %(message)s')
ch.setFormatter(formatter)
logger.addHandler(ch)

def parse(line):
    regex = r"\s*Kerberos:\s(?P<flag>\w+)-REQ\s \
      (?P<username>\w+)@(\S*)\s+from\s+ipv4: \
      (?P<source_ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}): \
      (?P<source_port>\d+)\sfor\s(?P<destination>host\/\S+)"
    match = re.match(regex, line)
    if match:
        print(line)
        res = match.groupdict()
        for name in res.keys():
            if not res[name]:
                del res[name]
        return res
    return None

def hpfeeds_connect(host, port, ident, secret):
    try:
        logger.info('{0}, {1}'.format(ident, secret))
```

```
41          connection = hpfeeds.new(host, port, ident, secret)
42      except hpfeeds.FeedException as e:
43          logger.error('feed exception: %s'%e)
44          sys.exit(1)
45      logger.info('connected to %s (%s:%s)'%(connection.brokername, \
46        host, port))
47      return connection
48
49  def main():
50      cfg = {
51          'host' : '',
52          'port' : 10000,
53          'channel' : '',
54          'ident' : '',
55          'secret' : '',
56          'tail_file' : '/var/log/samba/samba.log'
57      }
58
59      if len(sys.argv) > 1:
60          logger.info("Parsing config file: %s"%sys.argv[1])
61          cfg.update(json.load(file(sys.argv[1])))
62
63          for name,value in cfg.items():
64              if isinstance(value, basestring):
65                  cfg[name] = value.encode("utf-8")
66      else:
67          logger.warning("Warning: no config found, using default values for hpfeeds server")
68      publisher  = hpfeeds_connect(cfg['host'], cfg['port'], cfg['ident'], cfg['secret'])
69
70      tail = multitail2.MultiTail(cfg['tail_file'])
71      for filemeta, line in tail:
72          record = parse(line)
73          if record:
74              publisher.publish(cfg['channel'], json.dumps(record))
75              logger.debug(json.dumps(record))
76      publisher.stop()
77      return 0
78
79  if __name__ == '__main__':
80      try:
81          sys.exit(main())
82      except KeyboardInterrupt:
83          sys.exit(0)
```

Source Code A.5: DC logfile parser script

```
#!/bin/bash

if [ $# -ne 2 ]
    then
        echo "Wrong number of arguments supplied."
        echo "Usage: $0 <server_url> <deploy_key>."
        exit 1
fi

server_url=$1
deploy_key=$2

wget $server_url/static/registration.txt -O registration.sh
chmod 755 registration.sh

# Note: this will export the HPF_* variables
. ./registration.sh $server_url $deploy_key "samba"

echo "deb http://en.archive.ubuntu.com/ubuntu precise main \
multiverse" | sudo tee -a /etc/apt/sources.list

apt-get update
apt-get -y install git python-pip supervisor
pip install virtualenv

# Get the DC logfile parser source
cd /opt
git clone https://github.com/x4mp/samba-dc-hpfeeds.git
cd samba-dc-hpfeeds

virtualenv env
. env/bin/activate
pip install -r requirements.txt

cat >> samba-dc-hpfeeds.conf <<EOF
{
  "host": "$HPF_HOST",
  "port" : $HPF_PORT,
  "channel" : "samba.events",
  "ident" : "$HPF_IDENT",
  "secret" : "$HPF_SECRET",
  "tail_file" : "/var/log/samba/samba.log"

}
EOF

```

```
47 # Set up supervisor
48 cat > /etc/supervisor/conf.d/collector-dc.conf <<EOF
49 [program:hpfeeds-collector-dc]
50 command=/usr/bin/python /opt/samba-dc-hpfeeds/collector.py \
51 /opt/samba-dc-hpfeeds/samba-dc-hpfeeds.conf
52 stdout_logfile=/var/log/collector.log
53 stderr_logfile=/var/log/collector.err
54 autostart=true
55 autorestart=true
56 redirect_stderr=true
57 stopsignal=QUIT
58 EOF
59
60 supervisorctl update
```

Source Code A.6: DC deployment script

Thank you

Bibliography

[Anoa] Anomali. *Modern Honey Net — Manage & Deploy Honeypot Sensors*. URL: https://www.anomali.com/platform/modern-honey-net (visited on 04/25/2017) (cit. on p. 28).

[Anob] Anomali. *Modern Honey Network*. URL: https://threatstream.github.io/mhn/ (visited on 08/13/2017) (cit. on p. 28).

[Anoc] Anomali. *Modern Honey Network & Honeypot Use Cases*. URL: https://www.anomali.com/platform/modern-honey-net/mhn-usage-cases (visited on 04/25/2017) (cit. on p. 36).

[Bag15] Mark Baggett. *Detecting Mimikatz Use On Your Network*. 2015. URL: https://isc.sans.edu/diary/Detecting+Mimikatz+Use+On+Your+Network/19311 (cit. on p. 42).

[Bau14] Kurt Baumgartner. *Sony/Destover: mystery North Korean actor's destructive and past network activity*. 2014. URL: https://securelist.com/blog/research/67985/destover/ (visited on 06/04/2017) (cit. on p. 9).

[Ber09] Robin G Berthier. *Advanced honeypot architecture for network threats quantification*. ProQuest, 2009 (cit. on pp. 12, 13, 21–23, 34).

[Che] Bill Cheswick. „An Evening with Berferd In Which a Cracker is Lured, Endured, and Studied". In: (). URL: http://web.cheswick.com/ches/papers/berferd.pdf (cit. on p. 16).

[Che10] Thomas Chen. „Stuxnet, the real start of cyber warfare?" In: *IEEE Network* 24.6 (2010). ISSN: 08908044. DOI: 10.1109/MNET.2010.5634434 (cit. on p. 17).

[Chi+09] W Y Chin et al. „HoneyLab: large-scale honeypot deployment and resource sharing". In: *Network and System Security, 2009. NSS'09. Third International Conference on*. 2009, pp. 381–388 (cit. on pp. 20, 21).

[Deu15] Deutsche Telekom AG Honeypot Project. *T-Pot: A Multi-Honeypot Platform*. 2015. URL: http://dtag-dev-sec.github.io/mediator/feature/2015/03/17/concept.html (visited on 05/05/2017) (cit. on p. 30).

[EUK16] David Emm, Roman Unuchek, and Kirill Kruglov. *Kaspersky Security Bulletin 2016/2017*. Tech. rep. 2016. URL: https://kasperskycontenthub.com/securelist-germany/files/2016/12/Kaspersky%7B%5C_%7DSecurity%7B%5C_%7DBulletin%7B%5C_%7D2016%7B%5C_%7DReview%7B%5C_%7DDE.pdf (cit. on p. 9).

[F+08] Jérôme Francois, Olivier Festor, et al. „Activity monitoring for large honeynets and network telescopes". In: *International Journal on Advances in Systems and Measurements* 1.1 (2008), pp. 1–13 (cit. on p. 12).

[GKO05] Julian B. Grizzard, Sven Krasser, and Henry L. Owen. „The Use of Honeynets to Increase Computer Network Security and User Awareness". In: *Journal of Security Education* 1 (2005). ISSN: 1550-7890. DOI: 10.1300/J460v01n02_03. URL: http://www.tandfonline.com/doi/abs/10.1300/J460v01n02%7B%5C_%7D03 (cit. on p. 29).

[Gor+11] Katarzyna Gorzelak et al. „Proactive detection of network security incidents". In: *ENISA report (A. Belasovs, Ed.)* (2011) (cit. on p. 12).

[Kha] Swati Khandelwal. *Here's How Hackers Stole $80 Million from Bangladesh Bank*. URL: https://thehackernews.com/2016/03/bank-hacking-malware.html (visited on 06/04/2017) (cit. on p. 9).

[KK] Rauno. Kuusisto and Erkki. Kurkinen. *Proceedings of the 12th European Conference on Information Warfare and Security : University of Jyvaskyla, Finland 11-12 July 2013*, p. 406. ISBN: 1909507342. URL: https://books.google.de/books?id=CrIVBAAAQBAJ%7B%5C&%7Ddq=On+the+use+of+Honeypots+for+Detecting+Cyber+Attacks+on+Industrial+Control+Networks%7B%5C&%7Dlr=%7B%5C&%7Dhl=de%7B%5C&%7Dsource=gbs%7B%5C_%7Dnavlinks%7B%5C_%7Ds (cit. on p. 14).

[Kno] KnowB4.com. *AIDS Trojan — PC Cyborg — Original Ransomware — KnowBe4*. URL: https://www.knowbe4.com/aids-trojan (visited on 05/09/2017) (cit. on p. 8).

[MA07] Iyatiti Mokube and Michele Adams. „Honeypots: concepts, approaches, and challenges". In: *Proceedings of the 45th annual southeast regional conference*. 2007, pp. 321–326 (cit. on p. 17).

[MA15] Chris Moore and Ameer Al-Nemrat. „An Analysis of Honeypot Programs and the Attack Data Collected". In: *International Conference on Global Security, Safety, and Sustainability*. 2015, pp. 228–238 (cit. on pp. 12, 17, 18).

[Mai+11] Abhishek Mairh et al. „Honeypot in network security". In: *Proceedings of the 2011 International Conference on Communication, Computing & Security - ICCCS '11*. ACM Press, 2011. ISBN: 9781450304641. DOI: 10.1145/1947940.1948065. URL: http://portal.acm.org/citation.cfm?doid=1947940.1948065 (cit. on p. 19).

[Mic] Microsoft. *Audit logon events: Security Configuration Editor; Security Services*. URL: https://technet.microsoft.com/en-us/library/cc787567.aspx (visited on 05/24/2017) (cit. on p. 42).

[Pro] The Honeynet Project. *Honeywall CDROM*. URL: https://projects.honeynet.org/honeywall/ (cit. on p. 33).

[Pro04] The Honeynet Project. *Know Your Enemy: Learning about Security Threats*. Second. Addison-Wesley, 2004. ISBN: 0321166469 (cit. on p. 16).

[Pro05] The Honeynet Project. *Know your enemy: GenII Honeynets*. 2005. URL: http://old.honeynet.org/papers/gen2/ (cit. on pp. 11, 22, 24).

[Sma] SmartHoneypot. *5 effective use-cases of a honeypot system for enterprises - Smart Honeypot Blog*. URL: https://blog.smarthoneypot.com/5-effective-use-cases-of-a-honeypot-system-for-enterprises/ (visited on 04/25/2017) (cit. on p. 36).

[Spi] L. Spitzner. „Honeypots: catching the insider threat". In: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE, pp. 170–179. ISBN: 0-7695-2041-3. DOI: 10.1109/CSAC.2003.1254322. URL: http://ieeexplore.ieee.org/document/1254322/ (cit. on p. 12).

[Spi03a] Lance Spitzner. *Honeytokens: The Other Honeypot*. 2003. URL: https://www.symantec.com/connect/articles/honeytokens-other-honeypot (visited on 05/04/2017) (cit. on p. 14).

[Spi03b] Lance Spitzner. „The honeynet project: Trapping the hackers". In: *IEEE Security & Privacy* 99.2 (2003), pp. 15–23 (cit. on p. 12).

[SPP] Hemraj Saini, H N Pratihari, and T C Panda. „EXTENDED HONEYPOT FRAMEWORK TO DETECT OLD / NEW CYBER ATTACKS". In: (). URL: https://www.researchgate.net/profile/Bimal%7B%5C_%7DMishra/publication/50984926%7B%5C_%7DEXTENDED%7B%5C_%7DHONEYPOT%7B%5C_%7DFRAMEWORK%7B%5C_%7DTO%7B%5C_%7DDETECT%7B%5C_%7DOLDNEW%7B%5C_%7DCYBER%7B%5C_%7DATTACKS/links/0fcfd509b86b54225e000000.pdf (cit. on p. 13).

[SSY05] K Sadasivam, B Samudrala, and TA Yang. „Design of network security projects using honeypots". In: *Journal of Computing Sciences* (2005). URL: http://dl.acm.org/citation.cfm?id=1047890 (cit. on p. 13).

[STI03] Lance Spitzner, Honeypot Technologies, and Inc. „Catching the Insider Threat". In: *Acsac* Acsac (2003), pp. 170–179. ISSN: 10639527. DOI: 10.1109/CSAC.2003.1254322. URL: http://dblp.uni-trier.de/db/conf/acsac/acsac2003.html%7B%5C#%7DSpitzner03 (cit. on p. 14).

[Sto89] C. Stoll. „The Cuckoo ' s Egg". In: (1989) (cit. on p. 16).

[Swe03] Craig Sweigart. „Honey Pots - Strategic considerations". In: *Global Information Assurance Certification Paper* (2003) (cit. on p. 13).

[SZ15] Tomas Sochor and Matej Zuzcak. „Attractiveness Study of Honeypots and Honeynets in Internet Threat Detection". In: *Computer Networks: 22nd International Conference, CN 2015, Brunów, Poland, June 16-19, 2015. Proceedings*. Ed. by Piotr Gaj, Andrzej Kwiecień, and Piotr Stera. Cham: Springer International Publishing, 2015, pp. 69–81. ISBN: 978-3-319-19419-6. DOI: 10.1007/978-3-319-19419-6_7. URL: http://dx.doi.org/10.1007/978-3-319-19419-6%7B%5C_%7D7 (cit. on pp. 18, 19).

[Tea] Team CYMRU. *The Darknet Project - Team Cymru*. URL: `http://www.team-cymru.org/darknet.html` (visited on 04/27/2017) (cit. on p. 40).

[VKM10] Co-authors Sven Vetsch, Marcel Koßin, and Michael Mauer. „Know Your Tools : Glastopf“. In: *The Honeynet Project* (2010). URL: `http://www.honeynet.org` (cit. on p. 16).