Jakob Heher, BSc

Security Considerations in Online Learning

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Maria Eichlseder, Ass.Prof. Dipl.-Ing. Dr.techn. BSc BSc

Institute of Applied Information Processing and Communications

Graz, March 2021

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.

SIGNATURINFORMATIONEN	**Unterzeichner**	Jakob Maria Heher
	Datum/Zeit-UTC	2021-03-15T00:41:11+01:00
	Prüfinformation	Informationen zur Prüfung der elektronischen Signatur finden Sie unter: https://www.signaturpruefung.gv.at
Hinweis	Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr. 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.	

Date, Signature

Special Thanks

This work would not have been possible without the assistance of a number of people, whom I would like to thank here.

My thanks to Michael Eberl and Egon Humer of FIRSTMEDIA network GmbH, for granting us access to their product.

Thanks also go to the developers at solocode GmbH, FIRSTMEDIA network GmbH, and Werbe-Medien-Internetagentur M. Hicke, for being very forthcoming throughout the disclosure process, and quick to remedy the reported issues.

A very special thanks to Lorenzo Angeli for his invaluable aid in proof-reading and structuring this work.

Finally, thanks to my supervisor Maria Eichlseder for her support throughout all stages of the process. In particular, thanks for helping me craft my loose thoughts into an actual topic, and for putting up with my intermittent bouts of getting distracted by other work.

Acknowledgements

Compiling a list of fellow beings that have been instrumental in getting me to this point in my journey offers an opportunity for reflection, but is also a cruel ask. Human memory is fallible, and I am bound to omit some who unquestionably would deserve a mention.

Alas, I will still try my best – but I will nonetheless undoubtedly fail to do the task justice.

So, without further ado, I would like to thank all of the following. Their contributions are as varied as human nature, but I would not be who I am today without all of their influences.

Anton Voit, for introducing me to actual mathematics, and his patience with me. Neville Smit, for giving me a chance I probably didn't deserve, and letting me experience the joys of teaching for the first time. Bettina Klinz and Maria Eichlseder, for having answers when I had questions. Edward Snowden, for letting us see what was in front of our eyes. Sarah Jamie Lewis and Vanessa Teague, for being an inspiration.

Emmy Radich, for opening my eyes to humans' humanity. Kim Cavill, Kate Lister, and Wagatwe Wanjuki, for making me realize I had my head on backwards. Natascha Strobl, Rami Ali, Hasnain Kazim, and many others, for letting me look beyond the tip of my own nose.

My good friend Lorenzo Angeli, for always being there to let me know if I was going overboard.

My parents, for always supporting me to the best of their ability. Being your child has been an immense privilege.

There are so many more faces whose trajectories intersected mine briefly over the years, yet who have left an indelible impression on the amalgamation that is me. To all of them, I am eternally grateful.

Abstract

In early 2020, the CoViD-19 pandemic spread like wildfire around the globe. Governments reacted by enacting sweeping measures, bringing entire countries to a grinding halt. In this environment, online learning has become a mainstay of education. Many small platforms, developed on meager resources, are faced with an unforeseen torrent of data.

In this work, we investigate the state of online learning platforms in the DACH countries. We compile a list of resources recommended by public institutions, and identify from it a number of services likely to handle sensitive data. We then analyze the platforms' functionality through a security and privacy lens, and assist developers in correcting flaws.

We present severe vulnerabilities, leading to full account compromise, in all reviewed platforms. On a portion, we demonstrate unrestricted access to all enrolled students' data. Disclosure of these flaws has allowed them to be remediated, contributing to a more secure learning environment for thousands of students. We then discuss systemic factors that led to the outlined issues, and offer potential mitigations going forward.

Keywords: Information Security · Privacy · Web Security · Distance Learning · Online Learning Platforms · SQL Injection · Session Authentication · Cross-Site Scripting

Kurzfassung

Im Frühjahr 2020 verbreitete sich die CoVid-19-Pandemie rasend schnell auf der gesamten Welt. Viele Nationalstaaten reagierten zunächst mit drastischen Einschränkungen. Das öffentliche Leben stand in weiten Teilen still. Auch die Schulbildung, zuvor zumeist als traditioneller Präsenzvortrag gestaltet, wurde im Eiltempo digitalisiert. Auch viele kleine Unternehmen, teils unerfahren und mit wenigen Ressourcen ausgestattet, wurden aus heiterem Himmel mit der Verwahrung und Verarbeitung von gewaltigen Datenmengen betraut.

Im Zuge dieser Arbeit untersuchen wir im DACH-Raum verbreitete Online-Lernangebote. Um einen Überblick zu gewinnen, konsolidieren wir zunächst diverse Ressourcenlisten öffentlicher Stellen. In Folge identifizieren wir mehrere Plattformen besonderen Interesses und führen an diesen Sicherheitsanalysen durch.

In allen untersuchten Plattformen entdecken wir Schwachstellen, die die vollständige Übernahme eines gewählten Lehrer-Accounts ermöglichen. Weiters erlangen wir bei mehreren Angeboten Vollzugriff auf die Daten sämtlicher registrierter Schüler*innen. In Kooperation mit den Entwickler*innen erreichen wir die Behebung dieser Sicherheitslücken. Weiters diskutieren wir Faktoren, die wir für die Entstehung der beschriebenen Probleme als verantwortlich betrachten. Wir stellen Überlegungen an, wie solche Schwachstellen in Zukunft systemisch vermieden werden könnten.

Schlagwörter: Informationssicherheit · Datenschutz · Web Security · Distanzunterricht · Online-Lernplattformen · SQL Injection · Session-Authentifizierung · Cross-Site Scripting

Contents

List of Figures

Chapter 1. Introduction

In early 2020, the CoViD-19 ("Coronavirus") pandemic spread like wildfire across the globe. As infections surged among immunologically naïve populations, activities that previously defaulted to physical proximity had to rapidly be reconsidered, with many of them shifting to an online-only model.

Many users were scrambling to choose the digital platforms to use – they were often helplessly out of their depth, and always under extreme time pressure. Services that had previously been relegated niche roles were seeing their user count increase by orders of magnitude. With this came commeasurate attention to their security and privacy guarantees.

In-person schooling was hit particularly hard. Practically overnight, curricula worldwide were upended. Hundreds of thousands of educators were suddenly tasked with constructing an online learning environment for many millions of students [UNE20]. In a study, Johnson, Veletsianos, and Seaman find that "more than half of faculty members reported that they were using new teaching methods following the transition to an online setting" [JVS20].

The results, perhaps, could be called predictable. Take the California-based startup Zoom, which established itself as a leading provider of seamless video conferencing during the early stages of the pandemic. Almost immediately, it was rattled by a rapid-fire sequence of high-impact vulnerabilities [Wag21], such as a lack of proper end-to-end encryption [GL20]. It is not alone in this – Cisco's WebEx, Google's Drive, and Microsoft's Teams have all faced similar trouble in 2020 [Jan+20; Has20; Veg20].

In this work, we set out to contribute to gradual improvement, in our own little way. While videoconferencing giants such as the above certainly had their share of problems going in, they were also placed under intense public scrutiny due to their widespread and global use. Thus, we instead chose to focus on a more specific category – online learning applications. Previously often consigned to being an "adornment" for otherwise in-person teaching activities, these services also saw a surge of adoption throughout 2020 [VB20]. However, they typically do not command the development budget, enticing bug bounty program, or army of engineers that an application targeting enterprise customers might.

To this end, we first survey resource listings provided by various public institutions in Austria and Germany [BdStmk; BMBWF; KMK] for kids aged 6–10. We categorize the referenced platforms, and identify four targets of particular interest, which we considered likely to hold sensitive information: ANTON [sola] and Schlaukopf.at [Hic+12a], two platforms offering access to pre-defined interactive content; Antonwelt [Hum+18c], a creative workshop that allows students to explore writing short stories in a colorful environment; and LearningApps.org [Hie12], a platform allowing teachers to easily create their own specialized content through use of generic templates.

Reviewing each of these platforms, we initially view the application server as a black box. We inspect its behavior through "normal" use, and develop an understanding of the exchanged information's structure. Using the information gained as a basis, we then make incremental changes to the requests transmitted and observe the results. In doing so, we aim to identify erroneous behavior in the application server's handling.

For every one of the four platforms we reviewed, we were able to discover and disclose serious vulnerabilities allowing account compromise. This includes a SQL injection vulnerability in Antonwelt, Cross-Site Scripting vulnerabilities in Schlaukopf.at and LearningApps, and Cross-Site Request Forgery vulnerabilities in Antonwelt, Schlaukopf.at, and LearningApps.org. Furthermore, we analyzed the custom API design of ANTON and were able to compromise its fundamental authentication design. This allowed us to demonstrate continuous and complete access to all enrolled students' data in two of the four reviewed platforms. All issues outlined have been responsibly disclosed to the respective developers, and have since been mitigated as a result of our reports.

Finally, we offer some systemic criticisms adjacent to our work. We opine on the negative impact that poorly chosen default values and disorganized documentation can have on software security, and connect this to some of the concrete issues we discovered. Next, we consider proper design of account reset procedures, and how poor decisions here can circumvent even the most secure systems. Lastly, we discuss the concept of "adversarial thinking" – and posit the necessity of ingraining it in the next generation of software engineers.

The remainder of this work is organized as follows: in Chapter 2, we introduce the technical concepts that will be necessary throughout; in Chapter 3, we summarize some general concepts, introduce the methodology used in our case studies, and make some additional considerations; in Chapters 4 to 7, we present the findings of our investigation of each of the targets identified; in Chapter 8, we explore the nature of the flaws uncovered, and theorize on what systemic changes could have helped avoid them; finally, in Chapter 9, we reflect on our results and suggest potential future work in the area.

Chapter 2.

Background

When developing an application for public use, ensuring its functionality across different platforms is more important than ever. According to data aggregator NetMarketShare, as of September 2020, the majority of web activity (58%) originates on mobile devices, up from just 25% in 2016 [Net20].

In this environment, developing a native application is often a losing proposition, as it involves duplicating much work for each supported platform. Thus, developers have taken to using web technologies – already designed to be platform-agnostic, and ever-growing in capability – as a means to deliver a consistent user experience across practically all modern end user devices, in spite of significant performance overheads [CSS12]. A number of frameworks have sprung up to enable this, such as Electron [Ope20] and Apache Cordova [Apa12].

In the remainder of this chapter, we introduce a selection of web technologies and related issues. As enumerating the ever-growing list of interfaces exposed to the modern internet could fill volumes all by itself [Mozd], we limit ourselves to concepts that will be relevant to our analysis. This includes common underpinnings such as JavaScript, as well as more specific security concerns such as password storage, and attack approaches such as XSS and XSRF.

2.1. A brief history of interactive web applications

The internet's foundational means of data exchange, the **HyperText Transfer Protocol (HTTP)**, was originally conceived to allow "a client to acquire a (hypertext) document from an HTTP server, given an HTTP document address", and any document thus retrieved was specified to be "in hypertext mark-up language" [Ber91]. While the *document* in the 1991 specification was likely envisioned to be a *static* document, akin to retrieving a book from a library, it did not take long for this definition to expand.

By 1993, the **Common Gateway Interface (CGI)** standard was being established [McC93] as a means to "interface external applications with information servers, such as HTTP or Web servers" [McC94]. CGI scripts allowed HTTP responses to be generated dynamically – however, the retrieval of information was still intricately linked with the act of page navigation. A number of other technologies and languages, such as PHP (1995), ASP (1996) and Java Servlets (1997) sprung up over the next years, but did not change this fundamental paradigm.

Yet, in time, web applications evolved to provide more and more access that was not purely *reading* information, but also *modifying* it. This began to result in significant user experience challenges. For one, any user interaction that requires server feedback would necessarily introduce significant wait times during which the application appeared unresponsive – or worse, during which the user could inadvertently abort the operation by clicking another button. Furthermore, browsers' native history navigation could easily lead to users duplicating actions by accident – for example, innocously clicking "Back" after placing a purchase order could result in that same order being placed yet again.

To address this, in the late 1990s, Microsoft developed the **XMLHTTP ActiveX control**, at the time primarily for use in Outlook Web Access [Hop06]. This control allowed web pages to make requests to the server in the background, allowing them to send or retrieve data without needing page navigation. It was implemented starting with Internet Explorer 5 [Mic]. By 2002, the XMLHTTP control had been adapted into the **XMLHttpRequest (XHR)** JavaScript object for the NetScape (later Mozilla), Safari and Opera browers [App05]. This technology did not see widespread use until the mid-2000s, when the Google search engine began using it to dynamically load search suggestions on the fly, based on user input. The company's Gmail web interface and Google Maps navigation software also made heavy use of XHR to provide a responsive and seamless interface to users, setting them apart from most other web applications of their era [Hof19].

In 2005, the term **Asynchronous Javascript And XML (Ajax)** was coined [Gar05] for this use of technology – and the changed viewpoint on web development that came with it – and stuck. In the intervening decade-and-a-half, the preferred encoding for information has moved on from XML, and even the XMLHttpRequest interface is reaching the end of its tenure. However, the fundamental shift of decoupling information exchange from document navigation – this fundamental departure from the original structure of the internet as a mere means of document transfer – is likely to stay with us for a very long time.

2.2. JavaScript

JavaScript is the predominant language for client-side scripting in the modern web. A document may directly embed JavaScript code, or may instead reference an outside script file to be requested separately. Regardless, the browser evaluates the referenced instructions, and executes them in a sandboxed environment – allowing them to dynamically modify the content displayed to the user while isolating them from the underlying system.

The server architectures and programming languages in use may vary wildly between different websites. However, a client-side scripting language, by its nature, must be widely supported by various different web browsers to be useful. Thus, too, JavaScript remained hobbled for some time after its inception in 1993 – as a scripting language that was syntactically similar to Java for use in Netscape Navigator [Eic08b] – by a lack of standardization. This was exacerbated by Microsoft packaging their own brand of

JavaScript – named JScript for trademark reasons – with Internet Explorer [Mic97]. As a result, designers needed to cope with many subtle differences between browsers for the early oughts of the burgeoning millennium [Lak07].

However, by the mid-2000s, the then-stagnant browser market was shaken up by the entry of Netscape's successor, Mozilla Firefox [Bak04]. This, alongside the developing AJAX paradigm shift, brought with it a renewed push for true cross-browser standardization of JavaScript, again under the ECMA standards process to which it had originally been submitted in the 1990s. By July 2008, most fundamental disagreements had been resolved [Eic08a], and the ECMAScript 5 standard was released in December of 2009 [Ecm09]. While adjustments to browsers to comply with the new specification took time, by the time ECMAScript 6 was released in 2015 [Ecm15], most major browsers had settled into a mostly-interoperable state of affairs. Thus, the unified JavaScript remained as the definitive scripting language of the modern web.

Only the most basic of web pages in the modern day make do without executing any JavaScript, with some estimates giving usage rates in excess of 95% [W3T]. Use cases vary widely, from basic functionality such as form validation or dynamic repositioning of UI elements, to advanced applications such as on-demand loading or submission of data, or even fully-featured site-specific proxy servers [Mozc].

2.2.1. Functionality

Once loaded by a HTML `<script src="...">` tag, the script is evaluated in the browser's JavaScript sandbox environment. Execution is inherently single-threaded, but cooperative multitasking is supported in most modern browsers via `async/await`. To defer functionality, callback functions can be specified, either as event handlers or on a timed interval.

While interaction with the underlying OS is strictly controlled and gated by user approval, within the context of the web page itself, JavaScript is practically omnipotent. The entire HTML document tree is exposed to it via the **Document Object Model**, short **DOM**, a tree-like structure of object representations. This allows JavaScript logic to freely rearrange the document, adding or removing nodes at leisure.

Additionally, a number of means for making arbitrary HTTP requests are available [Mozf; Moza], though programmatic access to information requested from third-party domains is restricted (see Section 2.9). Various mechanisms for persisting data between visits are also widely supported, such as WebStorage [Moze] or IndexedDB [Mozb].

A plethora of other APIs are also exposed by modern browsers [Mozd], allowing JavaScript to seamlessly perform an ever-growing number of tasks such as editing images in real time, securely encrypting data, accessing the device's camera and microphone, and many more.

2.3. JSON

JavaScript Object Notation, or **JSON**, is a language-agnostic interchange format for structured data. As the name indicates, its syntax is a subset of that of JavaScript. An example is provided as Listing 2.1.

JSON supports six types of data:

- **Numbers**, in decimal notation
- **Strings** delimited by double quotes (`"`)
- **Booleans**
- **Arrays**, which are lists of data
- **Objects**, which are key-value pairs
- `null` as its own type

This limited feature set makes implementing a JSON parser a fairly simple endeavour. In particular, it is quite achievable to audit such a parser fully; this is crucial, as it is a piece of logic that will frequently deal with potentially malicious user input. Most modern languages provide JSON encoders and decoders as part of their standard library.

```
{
  "id": 7,
  "name": "Bond, James",
  "active": true,
  "affiliations": [
    {
      "id": 0,
      "name": "M"
    },
    {
      "id": 6,
      "name": "Trevelyan, Alec"
    }
  ],
  "restrictions": []
}
```

Listing 2.1: An example of a JSON-encoded user profile

2.4. SQL

Structured Query Language (**SQL** for short) is a family of syntactically adjacent dialects used by both users and software to communicate with the majority of widely-used database servers. Such a communication is made up of a number of individual instructions, or **queries**. Data is stored in **tables**, which are structured as a set of named **columns** of data. Each such table contains zero or more **rows**, i.e., data items.

At the most basic, each query is transmitted as a simple human-readable string containing both raw *data*, and *instructions* on what to do with that data:

```
UPDATE users SET status='At home' WHERE id=42
```

In this example, the *instructions* given are “Update the ‘users’ table, changing the column ‘status’ to a value for all rows where the ‘id’ column matches a second value.” – meanwhile, the *data* provided are the string “At home” for the first value, and the integer “42” for the second value.

2.4.1. SQL Injection Attacks

However, mixing data and instructions can have vast repercussions if a malicious party controls the data being processed, and the designer does not carefully consider this. For example, if an application allows the user to specify a custom status message, it might simply create a query like above by inserting the user’s requested status message into the raw string. However, a crafty attacker can then insert instructions into the query:

```
UPDATE users SET status='', isAdmin=true, status='' WHERE id=42
```

As a result, this innocent-seeming query is manipulated into having entirely unexpected side effects, as confusion occurs between the entered *data* and the underlying *instructions*. This class of attack is called an **SQL injection attack**. It is possible to programmatically apply **escape characters** to user input to prevent this behavior. However, this still requires the programmer to consciously invoke an appropriate function each time user input is processed – and any single oversight can have far-reaching consequences. If an attacker attempts the attack above, but proper escaping techniques are applied, this results in the following query:

```
UPDATE users SET status='\', isAdmin=true, status=\'' WHERE id=42
```

2.4.2. Prepared statements

Many SQL dialects allow for strict separation between instructions and data by introducing a separate preparation step. Here, the instructions contained in the query are processed first, with placeholders in positions where data would ordinarily be specified. Afterwards, data can be bound to the placeholders in this **prepared statement**. For example, the instructions contained in the statement above, would be expressed using placeholders as:

```
UPDATE users SET status=? WHERE id=?
```

While the particular placeholders and syntax used in prepared statements differ across implementations, the underlying concept remains unchanged. As the (user-controlled) data is strictly separated from the underlying instructions, no confusion can occur.

2.5. PHP

PHP – a recursive acronym for **PHP: Hypertext Preprocessor** – is “a widely-used Open Source general-purpose scripting language that is especially suited for web development” [PPre]. Created in 1994 as a set of CGI binaries [PHis], it is today embedded by many popular web servers as a way to enrich static HTML content. Its ability to seamlessly integrate into existing static HTML content makes it especially attractive as an entry point to web programming. As illustrated in Listing 2.2, the PHP parser will output any text not wrapped in a `<?php/?>` pair as-is – this allows users to take their existing static files, and add tiny specks of PHP over time without ever needing to re-engineer to entire page.

```
<html>
  <head>
    <title>My First Dynamic Web Page</title>
  </head>
  <body>
    <?php
      $one = 'Hello';
      $two = 'World';
      echo $one.'␣'.$two;
    ?>
  </body>
</html>
```

Listing 2.2: An example PHP file

This prioritization of accessibility also extends to many other aspects of the language's design. Variables in PHP are not only weakly typed, but will be very permissively coerced without requiring explicit instruction. For example, this leads to the string `'0, also evil code'` being considered equal to the integer `0` by the default equality operator. Dahse and Holz [DH15, section 2] provide a number of similar situations where such attempts at accessibility instead lead to hard-to-detect unsafe behavior.

2.6. HTTP cookies

As described in Section 2.1 and Section 2.2, the majority of web applications still rely on HTTP for most of their communication with the back-end server. However, HTTP is inherently a stateless protocol – each individual HTTP request is disconnected from the previous one. This makes HTTP servers comparatively easy to implement – but it also places significant constraints on the features web applications can provide. This was already recognized in 1994 [Kri01]. In response, at the request of one of their customers, Netscape added a proprietary feature to their Navigator browser that would allow web servers to store a string of text. This string would not be processed, but simply sent back unaltered with any subsequent requests.

In Unix lingo, the term "magic cookie" was commonly used to describe "a result whose contents are not defined, but which can be passed back to the same or some other program later", such as "the result of `ftell` [which] may be a magic cookie rather than a byte offset; it can be passed to `fseek`, but not operated on in any meaningful way" [Ray03]. Thus, the term **HTTP cookie** was coined for this new technology [Sch01].

The widespread adoption of this seemingly innocuous technology, and the associated ability to link disparate web requests made by the same client, had massive impact on the web ecosystem. With cookies quickly becoming required to use practically any web application, users were left with little recourse as a vast multi-billion dollar industry evolved around weaponizing their devices' functionality against them, allowing their every motion to be profiled, dissected, and turned into profit. Combined with a server's ability to force most clients into making requests to arbitrary domains – sending their identifying cookies along for the ride – the HTTP cookie allows disparate user profiles to be associated across the entire internet, producing an ever-growing and nigh-inescapable record of users' every move [Wod20].

2.6.1. Cookie attributes

To store a cookie in the client browser, the server includes a `Set-Cookie: key=value` HTTP header in any response. Optionally, this header can also include **cookie attributes**, which modify the cookie's behavior. Each attribute is simply appended to the header, with a leading semicolon (`;`) as delimiter.

Many cookie attributes exist to work around security concerns in the base cookie specification, which may not have been considered during its inception in the 1990s. We list some examples.

Setting the `Secure` attribute prevents the cookie from being included in non-HTTPS requests. Many browsers still make an initial HTTP request when the user types a URL. This request will typically just result in a `301 Moved Permanently` status message directing the user to the HTTPS version of the site. However, the request itself still includes cookies in plain text. This allows them to be intercepted by an on-the-wire passive attacker. If `Secure` is set, this is no longer the case.

Setting the `HttpOnly` attribute makes the cookie "invisible" to JavaScript code. By default, all cookies for a host are available to JavaScript sent from that host. This is often unnecessary, and allows cookie content to be compromised by a successful XSS attack (see Section 2.9). Setting `HttpOnly` thus provides a useful defense-in-depth mechanism with little downside attached to it.

Setting the `SameSite` attribute causes the cookie to only be sent with first-party requests. First -party requests are requests initiated by the application itself, rather than initiated by a third-party website. We elaborate on this further in Section 2.10.

2.7. Hash functions

A **cryptographic hash function** is a deterministic mapping of arbitrary binary data to fixed-length binary data. Generic hash functions are used in many areas, such as as a fingerprint of data, or for indexing of data structures. In a cryptographic context, the primary properties desired of a hash function are **pre-image resistance** and **collision resistance**: given the hash value (or **digest**) of some secret, it should be infeasible for an attacker to compute the original secret; furthermore, it should be infeasible for them to produce two values with the same digest. These properties, though they may appear simple at first, allow hash functions to be used in a wide variety of applications.

Of particular interest to us is the use of hash functions in password-based authentication. Here, exposure of users' passwords to an attacker would compromise not only their account on the affected service, but very possibly also their account across a number of other services [Das+14]. Thus, it has become common practice to store a hash digest of the user's password instead of the plain-text password. When a user then attempts to authenticate, the hash digest of the password candidate can be calculated and compared against the stored value.

Note that the focus on these particular properties leads to a very different incentive system in the design of password hash functions. While a general hash function should typically have minimal runtime and memory use, this is actually counter-productive in the case of password hashing use. Common attacks will often involve a vast number of evaluations of the hash function, while legitimate use needs comparatively few. Thus, the goal should be to *maximize* resource use while still remaining performant *enough* in the environment of actual use. Many hash functions even provide parameters that allow for dynamic adjustment of the cost of evaluation.

Widespread use of password hashing has also led to a corresponding focus on techniques for breaking, in particular, weak passwords. For instance, by investing some significant amount of resources, an attacker could pre-compute the hash value of all alphanumeric passwords of length 7 or lower (of which there are roughly 2^{42}) for some well-known password hashing function. This advance investment would then permit them to immediately extract some – likely not negligible – portion of the compromised passwords.

To defeat such an approach, modern password hashing procedures typically involve a **salt**; a random – though not secret – string that serves as an additional input to the hash function. As the salt does not need to remain secret, it can be encoded alongside the hash digest itself, allowing storage of the entire result as a single string. When a user attempts to log in using a password candidate, the salt can then first be extracted from the encoded digest string, after which the hash value can be re-computed using that same salt and compared against the original digest.

By introducing this additional unpredictable factor, the advance work an attacker would need to perform for such a pre-computation increases significantly – not only do they need to account for all possible passwords, but also for all possible salt values for each password. For a sufficiently sized and appropriately random salt, this makes such approaches infeasible at no relevant cost to genuine users.

Listing 2.3 shows an example of an encoded result of applying the Argon2 password hashing function [BDK16] to the password `swordfish`. Note how the password digest (`ySstDtm0...`) is stored alongside the random salt (`em1rS...`), as well as all other parameters required to reproduce the original environment the digest was created in.

```
$argon2id$v=19$m=128,t=15,p=4
    $em1rSXdEY3k0emp6REhGSQ$ySstDtm005SHDdenQnwACGfND7259jt2/
    gkpdSPdW8ZWS9udjX8FPh0hD1EqEsKB
```

Listing 2.3: An encoded password hash using the Argon2 hash function

2.8. Session identifiers

As previously mentioned in Section 2.6, HTTP is inherently a stateless protocol. This presents additional complexities for applications that require users to authenticate themselves to gain additional access.

A first naive approach may be to have a user provide authentication with every request they make. However, in even a modest application that simply allows navigation between multiple authenticated resources, this quickly becomes intractable for multiple reasons. For one, requiring the user to constantly re-enter their password is undesirable from a usability standpoint, while keeping their password in memory may not be an option due to security concerns. Furthermore, constantly re-authenticating a user will, if credentials are properly stored, also consume a nontrivial amount of server resources. (See Section 2.7 for more on secure password storage.)

Instead, it has become common practice to store short-term authentication tokens called **session tokens** – meant to be used for the duration of a user's active session – on the client, either through HTTP cookies (see Section 2.6) or other means. The client presents this information whenever necessary, allowing the server to link the user's request to their ongoing authenticated session without requiring the client to store a password. As long as this token is short-lived, the risks of potential compromise are reduced. Additionally, it is common to require the user to explicitly "re-authenticate" select high-impact operations – such as a password change, or finalizing an online purchase – with their password. This limits the harm inflicted if, for instance, a user fails to properly terminate their session on a public machine, and another user stumbles across it.

We can further differentiate between *stateful* and *stateless* session management techniques.

A *stateful* session manager stores data about the user's ongoing session on the server, indexed with a cryptographically secure random value, which serves as the session token. This random value is only shared with the client establishing the session, and a user's knowledge of the value is thus considered sufficient proof that they are that same client. However, stateful session management can be a concern when scaling web applications, as it cannot be guaranteed that the same web server responds to each request made by any given user. Thus, each web server must access and modify some centralized repository of session data with each user request, producing a performance bottleneck.

Instead, *stateless* session management relies on cryptography to guarantee authenticity of data stored by the user. Instead of storing session data centrally, the user is provided with a token identifying them that is signed using a private signing key. When the user then presents that token to a web server with knowledge of the associated public key, that server can verify validity of the token and establish the user's identity as vouched for by the signing server. In particular, the web server validating the token does not need knowledge of the private signing key. This allows for large flexibility in design of such stateless systems, and is at the core of many of today's federated authentication systems.

2.9. Cross-Site Scripting

As outlined in Section 2.2, JavaScript is the foundation of seamless interactive web design, and is supported by practically all modern web browsers. Of course, this has made it a very attractive target for attackers. For this reason, many features have become strictly limited in their scope over time. In particular, most JavaScript logic cannot "pass across" a domain boundary – for example, JavaScript served from `https://attacker.evil/` is unable to access the content of an inline frame in which a page from `https://secure.banking/` is displayed, and unable to inspect the contents of a `fetch()` query targeted at `https://secure.banking/`.

For this reason, being able to execute JavaScript in a **first-party context** – having it appear as if it is JavaScript originating from the target site's servers – is an extremely desirable goal for any attacker. This is the foundation of a **Cross-Site Scripting** attack. To execute such an attack, the attacker must have some way to entice the server into serving attacker-controlled code to the victim. In most cases, this uses methods akin to the SQL injection attacks outlined in Section 2.4.1, inducing confusion between *instructions* – such as a HTML `<script>` tag – and *data*, which is typically expected to be plain text. Thus, it can again be viewed as a failure to sufficiently distrust and validate user-supplied information.

For example, imagine a basic chat platform that allows users to send text messages. If a user sends the message "No, Mr. Bond, I expect you to die.", and the recipient views that message, they may be presented with a dynamically-generated HTML page similar to that in Listing 2.4. However, a crafty attacker can embed HTML in their message body. If the server does not properly validate the input, this results in a malicious script running in a first-party context (see Listing 2.5).

```
<html>
  <head><title>Incoming Message</title></head>
  <body>
    <b>From: Goldfinger, Auric</b>
    <b>To: Bond, James</b>
    <p>No, Mr. Bond, I expect you to die.</p>
  </body>
</html>
```

Listing 2.4: Example HTML generated by a benign message

```
<html>
  <head><title>Incoming Message</title></head>
  <body>
    <b>From: Attacker, Evil</b>
    <b>To: Bond, James</b>
    <p><script type="text/javascript">
       fetch("https://attacker.evil/?cookie="+document.cookie)
    </script></p>
  </body>
</html>
```

Listing 2.5: The same HTML for a malicious message which injects JavaScript

2.9.1. Countermeasures

Typical countermeasures can be roughly divided into two groups, akin for those for SQL injection outlined in Section 2.4.2. The former uses **escaping techniques** to render the user input “harmless” before inserting it, while keeping instructions and data intermingled. An example of this can be seen in Listing 2.6. As mentioned in Section 2.4.2, this relies on developers consistently inserting the escaping logic in all relevant locations – oversights are hard to spot, but any single one can compromise the security of the entire platform.

The latter group delegates transfer of data to designated data transfer formats, such as JSON (see Section 2.3), and then inserts that data into the respective elements. For example, JavaScript can use the `.innerText` attribute to specifically indicate that the content being inserted it purely data, or use `.createTextNode()` to create a text-only node with no parsing. Listing 2.7 shows an example of such strict separation in action.

```
<html>
  <head><title>Incoming Message</title></head>
  <body>
    <b>From: Attacker, Evil</b>
    <b>To: Bond, James</b>
    <p>&lt;script type=&quot;text/javascript&quot;&gt;
      fetch(&quot;https://attacker.evil/?cookie=&quot;+document.cookie)
    &lt;/script&gt;</p>
  </body>
</html>
```

Listing 2.6: The same HTML after the malicious message has been rendered harmless

```
<html>
  <head><title>Incoming Message</title></head>
  <body data-msg-id="42">
    <b>From: <span id="from"></span></b>
    <b>To: <span id="to"></span></b>
    <p id="msg"></p>
  </body>
  <script type="text/javascript">
    document.addEventListener('DOMContentLoaded', async () =>
    {
      const msgId = document.body.dataset.msgId;
      const url = ('/api/getMsg?id='+msgId);
      const {from, to, msg} = await (await fetch(url)).json();

      const ELEMENT = document.getElementById;
      ELEMENT('from').innerText = from;
      ELEMENT('to').innerText = to;
      ELEMENT('msg').innerText = msg;
    });
  </script>
</html>
```

Listing 2.7: An alternate approach that strictly separates the data from the HTML document

2.10. Cross-Site Request Forgery

HTTP Cookies, as described in Section 2.6, are opaque snippets of plain text that are stored in a user's browser and sent back with any request to the same server. They are a popular means of associating temporary authentication tokens with a user's browsing session, as outlined in Section 2.8 – the request being associated with the appropriate session token cookie is treated as proof of authenticity.

However, upon closer inspection, it becomes apparent that it is only proof that the request was made by the same web browser associated with the session. It is *not* proof that the request originated from the genuine website, or indeed that the user is even aware that they are making that request.

This opens a common vector for attack. If all data that must be submitted with a given request are known in advance, a malicious third-party site may ask that the user's browser submit that request – and the browser will comply without notification to the user. This results in a legitimate request being made to the server, including the user's session cookie. Such an attack is referred to as a **Cross-Site Request Forgery** attack, commonly shortened to **XSRF** or **CSRF**.

An example of this is shown in Listing 2.8. A seemingly innocent page includes an invisible `<iframe>` context containing a form targeted at `https://secure.banking`, with pre-set input values. This form is submitted when the page loads, causing the browser to make a request, which includes the user's session cookie. As the form is submitted in an inline frame, only that (invisible) frame is redirected to the banking portal. The attack is thus not apparent to the user.

```
<html>
  <head><title>Innocent Page</title></head>
  <body>
    <h1>Hello, user!</h1>
    <b>License Agreement</b>
    <p>This is dummy content to avoid suspicion.</p>
    <p>The user reads this, closes the page and forgets about it.</p>
    <iframe style="display:␣none" src="...">
      <!-- external content inlined for readability -->
      <form target="https://secure.banking/" method="POST" id="f">
        <input type="hidden" name="operation" value="sendMoney" />
        <input type="hidden" name="recipient" value="office@attacker.evil" />
        <input type="hidden" name="amount" value="42000" />
      </form>
      <script type="text/javascript">
        document.getElementById('f').submit()
      </script>
    </iframe>
  </body>
</html>
```

Listing 2.8: A malicious third-party causes a request without the user's knowledge

2.10.1. Countermeasures

To avoid such attacks, browsers have recently begun supporting the `SameSite` cookie attribute. This attribute lets the server indicate that a HTTP cookie should only be included if the request was initiated by that same host. This would cause the request in Listing 2.8, which was initiated by `https://attacker.evil/`, to omit the `SameSite`-enabled session cookie of `https://banking.secure/`.

This attribute supports the values `None`, `Lax` and `Strict`. `None` is the "original" cookie behavior, including the cookie with any request. `Lax` only includes the cookie with first-party requests, or with user navigation originating on a third-party site, such as a link click. `Strict` prevents the cookie from being included in third-party requests altogether.

As of 2020, adoption of these attributes is low. Van Goethem, Demir, and Pollard find that only 13.7% of first-party cookies specify any `SameSite` attribute – and of those, another 48% opt to use the insecure `None` option [VDP20].

However, major browsers are moving to make `Lax` the default unless specified otherwise. This change, which can be viewed as addressing an unintended side effect of the original cookie specification, may well herald the decline of the simple XSRF attacks outlined in this section.

Chapter 3.
Analysis of Online Learning Platforms

3.1. User roles

In educational applications, permissions are often arranged hierarchically, to mirror the structure of a traditional classroom scenario. As the names vary across services, we will assign one arbitrarily chosen label to each archetypal "role". We give these below, alongside some context.

Students are the least privileged users in the ecosystem. The exact scope of their ability will depend on the application's functionality, but they can generally be characterized as *consumers* of content provided by other users or the platform. Many of the typical assumptions of account ownership cannot be applied to student accounts due to their holders' typical age group. For example, common back-up authentication methods such as email or text messaging cannot be presumed to be available.

Teachers are afforded moderation privileges over a certain subset of students. They monitor those students' activities and, depending on the capabilities of the application, may also provide additional content for students to consume. Often, teachers are also provided the ability to perform certain administrative tasks on their students' accounts – for example, they may be able to reset passwords or otherwise restore access to an account. While this is extremely desirable from a real life usage standpoint, care needs to be taken in designing such features. We discuss this further in Section 8.2.

Teachers and students are typically organized into classrooms. A classroom may have one or more teachers, and any number of students.

Classrooms can be further grouped into schools. A school can have one or more **administrators**. These users will often have access to specialized tools designed for management of a large number of students and teachers. Batch account creation or data export functions are often provided here.

3.2. Attacker goals

Many abstract attacker scenarios can also be translated to the educational context. We attempt to identify and categorize some potential goals below.

I. Impersonation. The attacker aims to be able to produce content that is attributed to another genuine user by the platform, and then represented as that user's content to other users. For example, they might aim to send a message to a student that appears to originate from a teacher – or might submit an assignment in another student's name. This scenario gains special importance in a remote learning scenario, where the platform may well serve as a primary means of contact between a teacher and their students.

Here, the attacker aims to *write* data in a way that is *indistinguishable* from genuine user input.

II. Surveillance. The attacker aims to surreptitiously monitor a set of other users' activities. In particular, they may aim to monitor physical movements of the target's device, such as a mobile phone. Another aim may be to read communications between students, or between a student and a teacher, or to track their performance.

Here, the attacker aims to *read* data that is generated by *the genuine user.*

III. Suppression. The attacker aims to prevent a genuine user from accessing the service for a certain period of time. For example, they might aim to prevent a student from submitting an assignment during the hours leading up to an assignment deadline. Ways to achieve this are manifold, from corruption of user data to intentionally causing an account to become locked for security reasons.

Here, the attacker's aim is *not* to gain any access – instead, they purely aim to *deny* availability of the intended access from the victim.

IV. Reconnaissance. Instead of specifically targeting a single student, the attacker aims to gain less detailed information about a large number of users. For example, they may aim to enumerate all students of a school, alongside their full names and the classrooms they are members of. As a result, the attacker might obtain information needed for more specific targeting, such as internal user or classroom identifiers, which may otherwise not be readily available to them.

Here, the attacker aims to gain information on *many different users*, instead of focusing on a single victim.

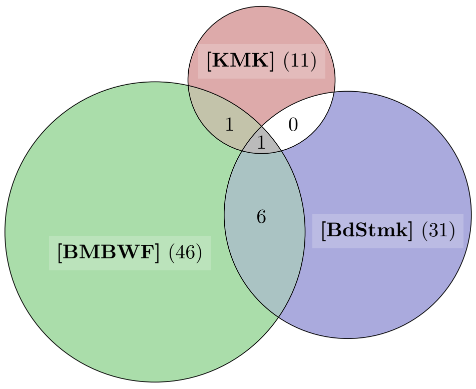
3.3. Case studies

In conducting our research, we reviewed a number of widely-used educational applications. In the following Chapters 4 to 7, we will be examining a select few in more detail. Note that all security flaws we discuss were communicated to the respective maintainers as part of a standard Responsible Disclosure process, and have since been addressed.

To determine which applications were actively being used in educational contexts in the DACH countries, we deferred to official information that was published early on during the COVID-19 pandemic. In particular, we used resource listings provided by the Austrian Federal Ministry of Education [BMBWF], the Styria Board of Education [BdStmk], and the Standing Conference of the Ministers of Education of Germany [KMK].

	[BMBWF]?	[BdStmk]?	[KMK]?
Ch.4 – ANTON	✓	✓	✓
Ch.5 – Antonwelt	✓		
Ch.6 – Schlaukopf.at	✓	✓	
Ch.7 – LearningApps.org	✓		

(a) Selected platforms



(b) Overlap between listings

Figure 3.1.: Public institutions' resource listings – aggregate data

From among these lists, we removed any applications that we deemed unlikely to hold sensitive user data, such as static task sheets, or other stateless services. An overview is provided as Figure 3.1, and a full list can be found as Appendix A.

We identified four targets of particular interest, which we considered likely to hold sensitive information: ANTON [sola] and Schlaukopf.at [Hic+12a], two platforms offering access to pre-defined interactive content; Antonwelt [Hum+18c], a creative workshop that allows students to explore writing short stories in a colorful environment; and LearningApps.org [Hie12], a platform allowing teachers to easily create their own specialized content through use of generic templates.

3.4. Review methodology

When reviewing a given platform, we aimed to start with the level of access typical to a student or teacher using the application. In some cases, such as that of Chapter 4, this did not require us to communicate with the administators, as registration was publicly available.

Other cases, such as that of Chapter 5, required accounts to be manually verified as belonging to a legitimate educational institution. In these cases, we used the listed support channels to disclose our intentions[1], and requested access to the application for research purposes. We would like to explicitly thank those responsible for any affirmative responses we received.

Once we had obtained the access described above, we began exploring the application's behavior in typical use cases. For example, given teacher-level access, we would observe the steps performed to create a student account, request their personal information, or add them to a group. Equipped with this knowledge of the program flow, we then began replaying the requests that had been made with minor modifications. Making small iterative changes, then observing whether they were still accepted as valid, allowed us to

[1]see Appendix B.1 for the form letter used

systematically identify the ways in which the application's server verified the received inputs.

We then compiled a list of these verification steps and compared them to the "intended" behavior of the application. In many cases, the two did not quite match up. In some, they did not match up in ways that allowed us to perform operations that exceeded the intended permissions. If we managed to elevate our access above the level we had been provided by the administrators, we then used that access to attempt to pivot further into the system.

Furthermore, we also considered common vulnerabilities while monitoring the application behavior. This includes SQL injection possibilities (see Section 2.4.1), as well as logic likely to be vulnerable to Cross-Site Scripting or Cross-Site Request Forgery attacks (see Sections 2.9 and 2.10).

3.5. Ethical considerations

We were typically conducting our reviews against a production environment that was processing the data of thousands of genuine users. Thus, we took a very cautious stance in evaluating potential attack vectors to test to minimize the risk of unintended consequences. From the outset, we determined that any attempt to compromise or otherwise target the underlying hardware would be out of scope for our review. Furthermore, we generally refrained from attempting data injection in any context that might potentially lead to loss of legitimate user data. For instance, we would typically avoid SQL injection inputs (see Section 2.4.1) that we could envision resulting in an unconstrained `UPDATE` statement being executed. As we were generally not able to observe the query that we were influencing, we erred on the side of caution when required.

Furthermore, we took great care not to come into possession of genuine users' private information. When probing and verifying targeted attack scenarios, we created dedicated "victim accounts" and performed legitimate activites to generate data where necessary. In a few cases involving exploration attacks, we demonstrated the ability to enumerate user data of "random" users without specific targeting being possible. Here, we ensured we accessed the minimum data necessary, and only stored it for the brief period of time required to verify that our attack had succeeded in retrieving what appeared to be genuine data, before disposing of it securely.

3.6. Responsible Disclosure

After analyzing the application, we compiled our findings for disclosure. To this effect, we summarized our findings in a PDF document. Typically, at least some of the flaws in question were indicative of a lack of fundamental security considerations during the platform's design and implementation stages.

Because of this, we included basic background information, akin to that of Chapter 2, outlining the systems in question and the general concepts that we used to compromise them. Following these explanatory paragraphs, we then described the particular attack vector used in full detail. In some cases where the attack required multiple interdependent steps, we also provided a proof-of-concept Python script or HTML file to further illustrate the process. Finally, we gave potential steps to address the specific vulnerability, as well as process-level recommendations for avoiding the same type of flaw in the future.

While this write-up document was being compiled, we attempted to establish a security contact for the affected platform. None of the platforms we evaluated had a `security.txt` file, or otherwise listed a specific disclosure contact. Thus, we instead directed our initial outreach to the most suitable address that was publicly listed. In this, we declared ourselves as researchers, provided that we had identified vulnerabilities, indicated that we would like to disclose them, and requested the appropriate channels through which to do so. We did not include any details of the vulnerabilities or information that would allow reproduction at this time. An example of such a form message can be found attached as Appendix B.2.

After we had received a reply directing us to a more suitable point of contact, we then supplied the necessary technical information, including our write-up document and accompanying files. We requested that the recipients confirm receipt of the message, asked to be informed as soon as an estimated schedule for remediation was available, and offered further assistance if requested. Platforms, generally, were very prompt in reacting to our disclosures and forthcoming in their communication.

3.7. About the following chapters

Each of the following chapters covers one platform that we reviewed at some point throughout 2020/21. Therein, we describe what we documented throughout that investigation, reflecting the subject platform's state during that brief window in time. We refer to each chapter's disclosure timeline for temporal context.

In the intervening time, each of the listed platforms has indicated to us that they consider none of the vulnerabilities we outline to still be applicable. As a result, many of the aspects we describe in this work will likely no longer be applicable at the time of publication.

Chapter 4.

ANTON

ANTON – Lernplattform für die Schule ("ANTON") is "a universal learning platform (web & mobile) for both independent learning and for students and schools who want to learn interactively in a classroom setting" [solb].

The app has garnered significant attention in DACH countries during the pandemic. It has the distinction of being the only app recommended by all three of the resource listings we used – by the Austrian Federal Ministry of Education [BMBWF], the Styria Board of Education [BdStmk], and the Standing Conference of the Ministers of Education of Germany [KMK]. It is developed by Berlin-based solocode GmbH, and its development is co-financed by the European Regional Development Fund [solb].

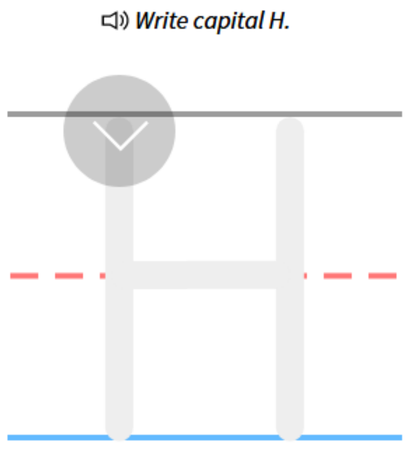
4.1. Features

ANTON's primary feature is a vast, comprehensive library of online self-study exercises for kids, covering a variety of topics at levels ranging from Kindergarten through to 4th grade (ages 4 to 10). Subjects are introduced and refined through a sequence of independent units, starting with interactive exercises before concluding with a short quiz.

The objectives are quite varied, and are not limited to mere multiple choice tests. For example, learners might be asked to write letters on their (touch-)screen, as seen in Figure 4.1a. Guidance is provided for this introductory exercise, which later units will successively reduce.

To further encourage learning, earning perfect scores rewards "coins", which can be spent to play simple, but engaging, arcade-like minigames, as seen in Figure 4.1b. Users can compete for high-score placement on leaderboards among their peer group and globally. This creates a feedback loop, as completing exercises is rewarded by allowing more attempts at a new record to be made.

To use the app, an account must be created through a very streamlined process. When doing so, the user must merely select a nickname and generate a randomized avatar, from one of the two palettes "human" and "monster". Afterwards, they can optionally associate their account with a school, which enables features such as intra-school rankings for minigames. After registration, a random "Login code" consisting of eight case-insensitive alphanumerical characters is provided, which serves as the sole identifier needed to log back on afterwards.



(a) An introductory writing exercise

(b) “Bat Cave”, a side-scrolling minigame

Figure 4.1.: Two example features of ANTON

In a classroom setting, additional features of the app become available. Teachers can allow students to join their classrooms by providing them an individualized eight-character code, which is also available in scannable QR form. Once joined, the teacher can assign individual assignments to their students for any given day, as well as monitor their progress and activities. The eight-character code the student used to join the classroom also functions as a login code for their account and is visible to the teacher, allowing seamless account retrieval if a student forgets their information.

In addition to the free base app, a subscription service called “ANTON Plus” is offered at an annual fee. While we did not purchase a subscription, its features are advertised as including off-line access to the exercise library, as well as additional cosmetic options for the user’s avatar [sola]. Multiple subscription tiers are available, such as a “Family” option that allows parents to customize their children’s experience in the app – for example by adding custom stretch goals with additional rewards. Additionally, school licenses are available, which enable Plus features for all students registered to the school. This also allows schools manage their students’ accounts in bulk, enabling tasks such as classroom assignments to be automated [sola].

4.2. Functionality

ANTON is a single-page web application. Its entire UX is handled in JavaScript with no top-level user navigation. The back-end server appears to be written in Node.js using the Express framework [Expr]. This is advertised in the `X-Powered-By` header with each response produced.

4.2.1. Basic API call structure

API requests to the server are encoded as JSON (see Section 2.3). All requests are made to `https://apis-db.solocode.com/pllsCallAuth`, which forwards the request to the backend. The request body is an object containing:

- `isDebug` *boolean*, which appears to always be `false`
  (We have not been able to determine what this value's impact is.)
- `src` *string*, a random four-character persistent identifier of the user device
- `logId` *string*, a token that authenticates the request for a specific user
  (This is omitted for unauthenticated requests. See Section 4.2.2 for details.)
- `path` *string*, a relative path that identifies the particular API call to make
  (All paths start with `/../server-apis-db2/apis`. We omit this going forward.)
- `params` *object*, the parameters to that specific API call

In response to such a request, the server returns a code 200 response consisting of an object that contains a `status` *string*. We have observed `"ok"` and `"error"` as values for this field.

`"status":"ok"` appears to indicate that the API handler script was executed without errors – not necessarily whether the requested operation was performed. It has the following additional fields:

- `url` *string*, the URL of the back-end server that processed this request
- `durationRequest` *number*, likely the time to execute the API handler script
  (We assume that this in milliseconds, and have not observed fractional values.)
- `postParams` *object* contains the original parameters passed with the request
- `jsonResult` *object* contains the actual response from the API handler

`"status":"error"` appears to indicate that the API handler script execution was aborted due to an error. The full stack trace of the error is provided in the `executionError` field.

4.2.2. User identifiers and authentication

When a user account is registered, that user account is assigned a random **user log ID**, which is the prefix `U-` followed by 32 random case-sensitive alphanumeric characters. This gives 62 possibilities for each character, or slightly over 2^{190} possible user log IDs.

To authenticate, the user sends an API request to `/login/step1/step1` with a *value* parameter containing the user's entered login code. This code is validated by the server. If it matches an existing account, the response contains the user's display name, avatar and user log ID. If the user indicates they wish to remain logged in, the user log ID is then stored in the browser's local WebStorage (see Section 2.2.1).

Login codes are eight characters long, and not case sensitive. This gives 36 possibilities for each character, or roughly 2^{41} possible login codes. The login API endpoint appears to be properly guarded against brute force attempts. After approximately 300 requests, the client IP address is prevented from making further requests for roughly a minute. If this limit is reached multiple times, successive blocks' duration increases rapidly.

With knowledge of the user ID, the client then requests that user's master log from `https://logger-lb-s2.anton.app/events2?filter[name]=subscribeUser&log=`. This log contains all information about the user as a JSON array of "log events" (see Listing 4.1). It is worth nothing that, despite being called a "log", it appears that outdated information is regularly pruned and is not included in future queries. Given this, we were not able to discern why this log format was chosen.

```
{
  "value": "max85021",
  "valueLowerCase": "max85021",
  "event": "setUniqName",
  "inserted": "2020-07-01T11:57:32.341Z",
  "src": "s-ddb-9",
  "created": "2020-07-01T11:57:32.341Z"
}
```

Listing 4.1: An example user log entry

Among the information included in this log are the user's "public ID", as well as their login code and any groups they are a member of. This public ID is an identifier of similar structure to a user log ID, but with prefix `P-` and apparently unrelated randomness. It is used to represent the user to other users.

4.2.3. Other identifiers

In addition to the user log ID described above, multiple other types of entities have similar identifiers associated with them.

Group identifiers

If a user is a member of any groups, their user log contains an `isGroupMember` event for each group they are a member of. This contains a "group code", which is the prefix `GROUP-` followed by two blocks of four case-insensitive alphanumerical characters. This group ID allows querying the group's log via `https://logger-lb-s2.anton.app/events2?filter[name]=subscribeGroup&log=`.

However, while this log contains group members' public IDs and similar metadata, it does *not* contain member portraits, login codes, or names. This information is loaded via a separate authenticated API request (see Section 4.2.1) to path `/group/members/getDescriptions/get` with the group code as a parameter.

School identifiers

Furthermore, if the group is associated with a school, that school's "school code" is also contained in the group log. A school code's structure is identical to that of a group code, except that the prefix used is `SCHOOL-`.

The school's log can be retrieved from `https://logger-lb-s2.anton.app/events2?filter[name]=subscribeSchool&log=` and contains various public information about the school. If the user is an administrator of a Plus-enabled school, an additional API call to path `/school/admin/membersReport/report` can retrieve information about all member groups and accounts of that school.

Device identifiers

Every individual device also has its own **device log ID**, with a similar structure to the user log ID. However, the prefix used is `D-`, and the fifth character of the suffix is always `-`, resulting in an ID of the form `D-ABCD-EFGHIJKLMNOPQRSTUVWXYZ12345`. The four-character block enclosed by the hyphens is included with every API request as the `source` parameter (see Section 4.2.1). If a user authenticates from a given device, that device's log ID is recorded in their user log, even if they do not choose "remember me".

The device log can be requested via `https://logger-lb-s2.anton.app/events2?filter[name]=subscribeDevice&log=`. It includes various information about the device, such as the device model, serial number or OS platform. Additionally, it also contains any user log IDs that are "remembered" on that device, and a periodically-updated physical location of that device. We speculate that this latitude/longitude pair is derived from the user's IP address, as the app does not appear to use the client's GPS functionality.

4.3. Analysis

In the following, we first analyze ANTON's fairly unique method of authentication. Then, we outline multiple issues we discovered throughout our investigation. We demonstrate faults in the fundamental design, granting us continuous and complete access to all enrolled students' data.

4.3.1. Identifier design

Given the information presented in the preceding sections, we can now analyze the sensitivity of each of these identifiers. We visualize how identifiers can be derived from each other in Figure 4.2.

In the figure, is easily apparent that login codes, user log IDs and device log IDs are essentially equivalent to each other. The default login flow translates a login code to a user log ID, while the user's user log includes both their login codes and their device log IDs. Meanwhile, at least their "main" device's device log is very likely to include their user log ID. As there is no concept of a server-stored "session", knowledge of a user's log

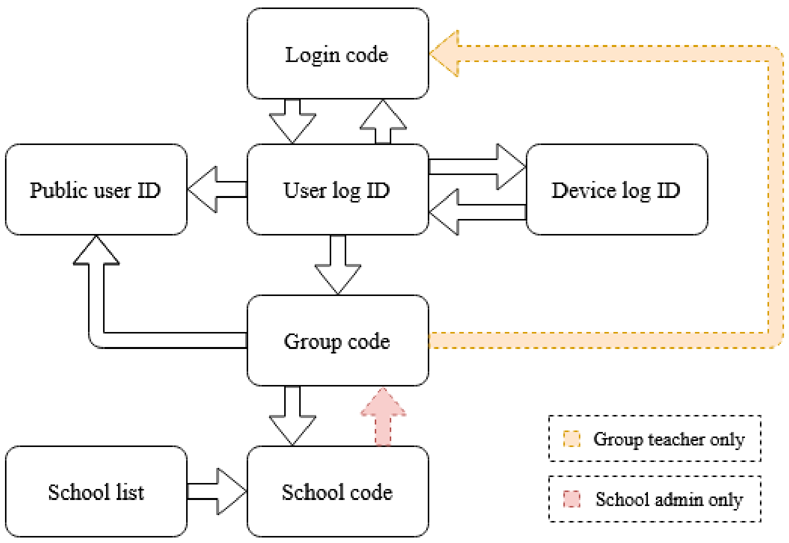


Figure 4.2.: Derivatory relationships between entity identifiers in ANTON

ID grants permanent irrevocable access to their account. This makes login codes, user log IDs, and device log IDs extremely sensitive information.

Meanwhile, group codes are visible at least to all members of the group, including those without any particular permissions. Users' public IDs are visible to any users they share a group with, and may also be globally visible via minigame leaderboards or similar. School codes are public by design, and are indexed by the school search system.

Authenticated APIs exist to allow school administrators to derive group codes from school codes (via path `/school/admin/membersReport/report`), as well as allow teachers to derive login codes of all student members from group codes (via path `/group/members/getDescriptions/get`).

4.3.2. Systemic vulnerability

As described in Section 4.2.1, the requesting user's log ID must be included with any authenticated API request. This log ID is then validated in the context of the call, such as specific group or school. The result of this validation is included in the API response's `jsonResult` as an `auth` field.

However, the result of this validation is only actually used by a small handful of paths. We theorize that these paths are ones that are intended to be invoked by users of different access levels. For instance, `/group/members/getDescriptions/get`, which is used both by students and teachers, refuses to return data for non-existent or non-member user log IDs. The vast majority of paths does not check the passed user log ID at all – indeed, specifying an empty string will change the `jsonResult.auth` field, but the API will still perform its function as requested, and otherwise return the same data it would for a legitimate user.

We speculate that an attacker sending API requests directly – rather than being constrained by the app's UX flow – was not considered when implementing the API[1]. This grants a malicious user unrestricted access to a wide variety of administrative tools.

While the "native" API path for retrieving students' log ID *does* only return that data if a valid teacher log ID is passed, this can be trivially circumvented. For example, the `/group/members/addExistingUserWithCode/step2/step2` path, which is used to add a student to a group, takes the new student's public ID as a parameter. However, it does *not* validate whether the call is being authenticated with that same student's log ID.

This allows an attacker to surreptiously obtain the user log ID of arbitrary accounts:

1. Create a new student code in an attacker-controlled group. As a result, the server returns this code, for example `ABCD-EFGH`.

2. Invoke `/group/members/addExistingUserWithCode/step2/step2` with `ABCD-EFG` and the target's public ID. The server will add the target to the group without authenticating that the request is from their user log ID.

3. Log into the target's account by using the `ABCD-EFGH` student code, which has become a login code for the target's account. This gains us their user log ID.

4. Remove the target from the attacker-controlled group, rendering the attack undetectable to the user.

As mentioned in Section 4.3.1, students' public ID is generally not considered to be privileged information. In particular, as the `/school/admin/membersReport/report` API also does not check the user's log ID, an attacker can list the public ID of all students in a given school. School codes can be freely obtained from the publicly-accessible school search API.

The same underlying flaw could likely allow an attacker to perform any number of unauthorized actions. However, as we could already demonstrate undetectable full account compromise, we did not pursue this further.

4.3.3. Device logs' existence

As previously outlined in Section 4.2.3, each device is assigned a unique **device identifier** when first using the app. This device identifier allows access to the **device log** for that device. This log contains not only information about the device, but also the device's physical location and the user IDs of any users logged into the device that have enabled "remember me".

This presents a significant risk factor. In many contexts, multiple students may use a single shared device over some time. Any one of these students could retrieve the device's device log ID and store it externally. This would then allow them to gain access to the accounts of any other students that uses that device at a later point in time. This is conditional on those students clicking "Yes, remember me" when logging in. However, we

[1]We discuss the importance of such "adversarial thinking" to software design further in Section 8.3.

posit that many will simply click the button labeled “Yes” without fully processing its meaning. Additionally, even a user that processes the question will not typically expect that agreement to grant access to their account to any *previous* user of that same device.

There is no legitimate reason for any device to ever query a device log that is not its own. It is thus unclear to us why these logs are stored centrally to begin with, instead of using various device-local storage methods available to web applications. In particular, these same storage methods are *already* in use – they are used to store the device log ID itself, after all. We find repurposing them to store the contents of the device log instead to be a fairly straightforward improvement.

4.3.4. Teachers can impersonate students

As described in Section 4.1, if a student joins a group, the code they used to do so becomes a login code for their account. Not only is this code known to the teacher that issued it, but is even displayed on the student’s profile for every teacher in the course.

This allows any teacher to authenticate as any student in any course they administrate. As a consequence, they can gain access to that student’s device logs, allowing them to monitor their physical location.

As we argue in Section 8.2, allowing a teacher to impersonate their students without their knowledge should never be permitted. Of course, allowing teachers some degree of administrative control over their students’ accounts is a useful feature, and provides for important convenience if students inevitably forget their login code.

However, any use of this ability should always impede students’ UX in a way that they are forced to notice. For example, a teacher could well be permitted to reset a student’s login code – however, doing so should then also invalidate all existing login codes, and log out all user devices. Thus, the user would immediately notice the access when they next attempt to use the app, making covert abuse impossible.

4.3.5. The API provides verbose error logs

If a request to an API server contains malformatted or otherwise invalid data, it is very likely that the API handler will fail uncontrolledly. If this happens, an error response including the full error message and call stack will be returned, as described in Section 4.2.1. This leaks an enormous amount of information about the server’s implementation to the user (see Listing 4.2), who should be considered a hostile entity in this context. This can then be used to guide further exploration of the system. In a production environment, the information revealed here should thus be minimized.

4.3.6. Potential for Arbitrary Code Execution

As briefly mentioned in Section 4.2.1, every API call includes a `path` parameter. As `/../` is included in genuine values of this parameter, we consider it unlikely that this path is constrained to subfolders of the web root. Additionally, the stack trace in Listing 4.2 suggests that the file at this path is `load`ed directly.

```
TypeError: Cannot read property 'replace' of undefined
 at load
   (/../server-apis-db/apis/pllsCallAuth/request.js:7:37)
 at eval
   (/../server-apis-db/apis/pllsCallAuth/request.js:62:12)
 at eval
   (/../server-apis-db/init/middlewareOverride/apiRequest/apiRequest.js:20:34)
 at routeRequest
   (/../server-apis-db/init/middlewareOverride/override.js:20:50)
 at initProject
   (/../server-apis-db/init/middlewareOverride/override.js:13:18)
 at global.middlewareOverride
   (/../server-apis-db/init/middlewareOverride/override.js:25:14)
```

Listing 4.2: An example stack trace, produced by an empty request body

We speculate that this would allow an attacker to execute any file that they manage to place on the server, such as via a file upload. However, due to the considerations outlined in Section 3.5, we did not pursue this attack vector further.

4.4. Disclosure

We initially reached out to the ANTON support contact (`support@anton.app`) on July 20th, 2020. We promptly received a response directing us at a disclosure address, to which we disclosed our findings on July 21st.

The ANTON developers swiftly reacted. By the 22nd, the primary APIs used to escalate privilege had been changed to properly use the result of user log ID validation. By the end of July, all API paths properly verified the user's permissions, and location information was no longer recorded in the device log. By late September of 2020, ANTON communicated to us that they considered all vulnerabilities we had disclosed to have been resolved.

Chapter 5.

Antonwelt

Antonwelt – Lernen mit Anton, dem kleinen Gespenst ("Antonwelt") is a German-language platform that "aims to make learning to read and write an enjoyable experience" [Hum+18a]. Despite the similarity in name, there is no connection to ANTON, the platform from the preceding Chapter 4.

It is developed by Vienna-based FIRSTMEDIA network GmbH [Hum+18b] in co-operation with a number of educators [Hum+18c]. Its development is supported by the Austrian Federal Ministry of Education [Hum+18d], and it is recommended by the Standing Conference of the Ministers of Education of Germany [KMK].

At the time of our evaluation, a total of 142 schools were registered with Antonwelt – 90 in Austria, 48 in Germany, 3 in Switzerland, and 1 in Italy. Across them, there were 7,774 user accounts, of which 6,418 were student accounts.

5.1. Features

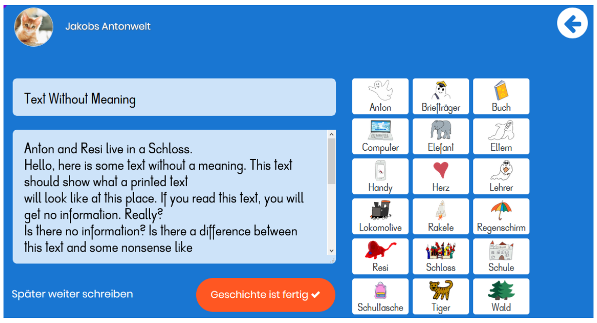
Antonwelt's primary feature is its "story workshop". Here, children are encouraged to read – and subsequently write – short stories featuring characters and settings from the "Anton" series of children's books.

Students first write their story. Certain keywords, such as objects, characters or settings from the books, are suggested. After they are finished writing, they submit the story for approval. This notifies their registered parent or teacher. After review and editing, it can then be approved for reading. Subject to parental consent, the story can also be read by other students. This process is shown in Figure 5.1.

To use the app, schools must purchase a license at an annual fee, which is tiered based on the number of supported classrooms. Once enrolled, a central administrative account can then be used to create individual teachers' accounts, who can then further create student accounts. For each student, the teacher is required to positively affirm the parents' approval for use of Antonwelt. Additionally, parents can optionally allow use of inter-classroom/inter-school chat functionality, and control public visibility of the student's approved stories.

After the account is created, the teacher then sets a nickname and password for the student. This information is then used by the child to sign in and use Antonwelt.

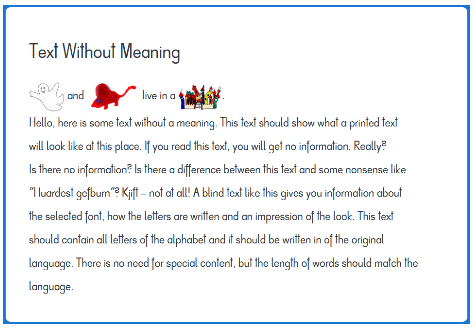
In addition to the story workshop, teachers can also manually enable two additional features during class hours, shown in Figure 5.2. A "digital postcard" feature lets students



Text Without Meaning

Anton and Resi live in a Schloss.
Hello, here is some text without a meaning. This text
should show what a printed text
will look like at this place. If you read this text, you will
get no information. Really?
Is there no information? Is there a difference between
this text and some nonsense like

(a) The story being written

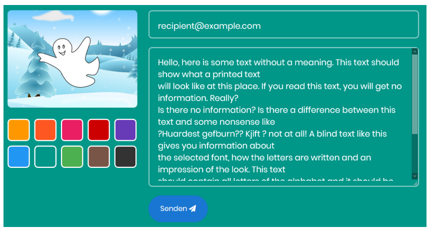


Text Without Meaning

and live in a .
Hello, here is some text without a meaning. This text should show what a printed text
will look like at this place. If you read this text, you will get no information. Really?
Is there no information? Is there a difference between this text and some nonsense like
"Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about
the selected font, how the letters are written and an impression of the look. This text
should contain all letters of the alphabet and it should be written in of the original
language. There is no need for special content, but the length of words should match the
language.

(b) The same story being read

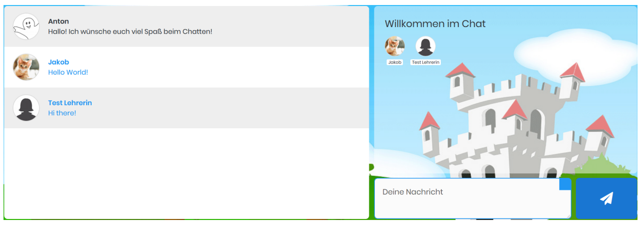
Figure 5.1.: A story's journey in Antonwelt



recipient@example.com

Hello, here is some text without a meaning. This text should
show what a printed text
will look like at this place. If you read this text, you will get no
information. Really?
Is there no information? Is there a difference between this
text and some nonsense like
?Huardest gefburn?? Kjift ? not at all! A blind text like this
gives you information about
the selected font, how the letters are written and an
impression of the look. This text

(a) Writing a digital postcard



Anton
Hallo! Ich wünsche euch viel Spaß beim Chatten!

Jakob
Hello World!

Test Lehrerin
Hi there!

Willkommen im Chat

(b) Intra-class chat room

Figure 5.2.: Additional features of Antonwelt during class hours

send colorful emails to friends and family. A chat room lets students exchange messages with other students in their class – or another class chosen by the teacher.

5.2. Functionality

Antonwelt is a traditional web application. It is written in PHP, as indicated by the `X-Powered-By: PHP/7.0.15` header sent with responses.

5.2.1. Basic environment

Antonwelt uses server-stored sessions, with what appear to be PHP's default settings. The session ID is stored in a `PHPSESSID` cookie with no security options specified. This causes it to adopt certain default behaviors, as described in Section 2.10.

API requests are made via AJAX, with each handler being a separate script in the `https://www.antonwelt.schule/plattform/api/ajax/` folder. Request data is transmitted in HTTP form encoding.

5.2.2. User authentication

Once a school purchases an Antonwelt license, a central school admin account for that school is created by Antonwelt staff, with a random password, which is provided to the school. School admin, teacher and parent accounts are uniquely identified by their associated email address.

To log in, a HTTP POST to `/api/ajax/login.php?login=` is made with three parameters - `email`, `password`, and `userType`. `userType` appears to be 1 for the parent login form, and 2 for the teacher login form. The latter form is also used for school admin accounts.

The API provides a response in plain text. For a successful login, this response is `ok#`, followed by a path to redirect the user to. For a failed attempt, the response is `error#`, followed by an error message to display. School admin accounts are redirected to `/useradmin/schule/`, teachers are redirected to `/useradmin/lehrer/`. A successful login response also sets a `aw` cookie, which contains the user's user ID. We were not able to determine what purposes this cookie serves – its presence, or lack thereof, does not appear to change the site's behavior.

5.2.3. School admin functionality

School admin accounts for Antonwelt are fairly limited in their functionality. They do not offer any direct control over enrolled teachers – beyond initial account creation – or any students in their classrooms.

To create a teacher account, a request to `/api/ajax/teacher.php?add_teacher=` is made with four parameters – `firstname`, `lastname`, `email` and `sex`[1]. After the request succeeds, an email is sent to the specified address containing a verification link. Clicking that link allows the teacher to finalize account creation.

Additionally, forms for modifying the school's information, and its admin account's password, are provided. To modify the password, a request is made to the multi-purpose endpoint `/api/ajax/user-data.php?save_new_password=` with a number of parameters. In addition to the functional parameters `password` and `password2`, which hold the dual values in the "new password" and "confirm new password" boxes, a total of six static values are also included:

- `password_required` with value `Neues Passwort`
- `password2_required` with value `Passwort wiederholen`
- `password_type` with value `password`
- `password2_type` with value `password`
- `schoolchild_userId` with value `0`
- `teacher_userId` with value `0`

We speculate that the two `userId` fields are used in other invocations of this API, and are included using zero as a "none" value, to indicate editing the current account. It is unclear to us what purpose the `_required` and `_type` fields' inclusion serves.

[1]Frustratingly enough, derived from a dropdown with "Herr" (i.e., "Mr.") and "Frau" (i.e., "Mrs.") as the only options

5.2.4. Teacher functionality

Once a school admin requisitions a new teacher account, the specified email address receives a message. This includes a confirmation link of form `https://www.antonwelt.schule/lehrer/confirm-registration-from-school/?sec=TOKEN&userId=USERID`. The token is a hex-encoded 128-bit value, which we assume to be random. Clicking the link allows the teacher to set a password.

Each teacher account is equivalent to a classroom. To add students, the teacher first makes a `POST` request to `https://www.antonwelt.schule/useradmin/lehrer/schueler/` with a numeric `addCount` parameter. Before each of the newly added accounts becomes usable, the teacher must first explicitly confirm parental consent with that student's use of Antonwelt. Doing so entails a `POST` form submission to `/api/ajax/teacher.php?agreement_schoolchild=` with a number of functional parameters:

- `firstname` *string*
- `lastname` *string*
- `sex` *1/2*
- `birthday_day`, `birthday_month`, `birthday_year` *integer*
- `agreeFlag`, `schoolchat_right`, `antongeschichten_right` *0/1*
- `chiffre` *string*
- `schoolchild_userId` *integer*

The `chiffre` appears to be an additional, presumably random, token. It is unclear what purpose this serves, as requests to this endpoint are authenticated against the user's session regardless. Additionally, a number of static fields are included, akin to those mentioned in the previous section.

After the user is thusly created, its data can then be adjusted by sending a `POST` to `/api/ajax/user-data.php?save_data_schoolchild=`. It is notable that this request does not redirect the user on success, and thus does not follow the usual `ok#/error#` return pattern. Instead, it appears to return the SQL query executed by the update operation as a plain text response body. An example of such a query is shown in Listing 5.1. We discuss the information that can be gleaned from this in Section 5.3.

```
UPDATE vm347hob_db.user SET FirstName='Test',LastName='Name',
UserData='$sex=1; $birthday_day=1; $birthday_month=1; $birthday_year=2008; 
    ↪ $pseudonym=\'\'; $teacher_userId=0; ' ,
AntongeschichtenRight=0,SchulchatRight=0 WHERE UserId=8875 LIMIT 1
```

Listing 5.1: An example response body

5.2.5. Student functionality

Once a teacher has setup the student's account, they can log in via a simple `POST` to `/api/ajax/login.php?login-schoolchildren=` with parameters `nickname` and `password`.

Submitting a story is then achieved via `POST` to `/api/ajax/schoolchildren.php?save_story` with a number of parameters:

- `titel` *string*
- `story` *string*

- `storyId` *integer*
- `topicId` *integer*, appears to always be `2`
- `language` *integer*, appears to always be `0`, presumably for German
- `currentStartTime` *integer*, a UNIX timestamp
- `finished` *0/1*, whether this is the story being submitted for approval
- `schoolFlag` *integer*, appears to always be `0`

For reasons that will become apparent in Section 5.3, we chose not to further explore students' functionality.

5.2.6. Antonwelt-Chat

After chat functionality is enabled by a teacher, navigating to the chat page causes a periodic `POST` to be sent to `/api/ajax/antonchat.php?update=`. The initial request will retrieve the channel's user list, while any subsequent requests will provide delta updates for user list and message history. It is likely that the last-requested time is stored on the server, as no dynamic request parameters appear to be included.

To send a message, a `POST` is sent to `/api/ajax/antonchat.php?send_text=` with a `chatText` parameter.

5.3. Analysis

In stark contrast to ANTON's API, as described in Section 4.2.1, Antonwelt's fundamentals are a lot more straightforward. We view this as a positive – reduced systems complexity, generally, also makes it easier to keep those systems audited and secure.

5.3.1. SQL injection vulnerability

As shown previously in Listing 5.1, one of Antonwelt's API endpoints appears to inadvertedly expose the SQL query that is run as a result of the API request. This allowed us to operate a lot more aggressively than we ordinarily would in light of the considerations in Section 3.5.

Indeed, it appears that the developers made a conscious effort towards preventing SQL injection by applying escaping techniques. However, as we outline in Section 2.4.1, all that stands between this approach and disaster is a single developer's due diligence, with no systemic safeguards in place. As is often the case, this also proves fatal here.

Antonwelt does not properly process the value submitted for the boolean parameter `schoolchat_right`, and simply trusts the user to provide a value of `0` and `1`. Specifying a malicious value for this parameter thus grants us the ability to read arbitrary data from the database into the user data fields of a student we control, as pictured in Listing 5.2.

```
UPDATE vm347hob_db.user SET (...), SchulchatRight=1,firstname=(
  SELECT UserData FROM vm347hob_db.user LIMIT 1
) WHERE UserId=8875 LIMIT 1
```

Listing 5.2: A malicious parameter leads to SQL injection

In particular, MySQL provides the `information_schema` meta-database, which provides database structure information in query-accessible form. This allows us to also use our SQL read primitive to gain structural information about additional tables, which we could then also read data from.

Among other things, this would have given us full access to all stories submitted by students, including not-yet-reviewed, private, or deleted stories. It would have also allowed us to retrieve all user data of users, as well as chat history, billing data, and more.

By changing the malicious parameter's structure, we believe we could have additionally achieved an arbitrary SQL write primitive. However, due to data integrity concerns, we did not pursue this further.

5.3.2. Passwords are not stored properly

Using the SQL read primitive we established in Section 5.3.1, we were able to inspect the way user passwords are stored on the server.

Passwords are stored as hash digests. However, the hash function used is identified by its prefix `$1$` to be MD5, as used by PHP's default one-parameter `crypt()` API. Collisions in MD5's compression function were first found by Dobbertin in 1996 [Dob96], with full collisions being demonstrated by Wang et al. in 2004 [Wan+04]. Additionally, calculating a MD5 digest is lightning fast and easily parallelized, making it utterly unsuitable for storing passwords (see Section 2.7).

We find it likely that this shortcoming is a direct consequence of flaws in PHP's documentation of `crypt()`, as described in Section 8.1.

5.3.3. Some data is stored as code

Referring back to the original exposed query in Listing 5.1, the `UserData` column stands out. A brief perusal suggests that this column contains multiple columns worth of data, stored as PHP code that sets multiple variables.

By modifying our original query slightly, we were able to test this theory.

```
UPDATE vm347hob_db.user SET (...), SchulchatRight=1,
    UserData='$sex=1; $birthday_day=1; $birthday_month=1; $birthday_year=2008; $pseudonym=getcwd();
$teacher_userId=0; '
WHERE UserId=8875 LIMIT 1
```

Listing 5.3: Attempting to execute code on the server

This indeed caused our student's pseudonym to be set to what appears to be a server path. We find it likely that this would allow an attacker to pivot further into the system. "Mere" write access to a single database table could allow them to transition into full and/or persistent system compromise.

Additionally, we note that even introducing a double quote (`"`) into a student's pseudonym via the genuine UX – without any modifications to the query – is sufficient to break the escaping mechanism used. Due to the resulting invalid PHP code that will cause page evaluation to fail, this renders the user account inaccessible. We consider it quite possible that this could cause an attacker to achieve code execution by itself via an appropriately crafted pseudonym, without requiring our original SQL injection flaw.

Due to the ethical considerations outlined in Section 3.5, we did not pursue these attack vectors further.

5.3.4. Session cookies are not secured

Antonwelt appears to use default settings for the PHP's session cookie. As mentioned in Section 2.5, this does not set any security attributes on the resulting cookie. As described in Section 2.10, this allows a third-party site to trivially issue singular requests with the user's session cookie.

In Antonwelt's case, the immediate target for this is the "change password" form, which we describe in Section 5.2.3. This API does not require any parameters that are unknown to the attacker beyond the user's session cookie. Thus, a third-party site can change the password of any visitor that is also logged into Antonwelt.

5.3.5. The chat member list is publicly visible

As described in Section 5.2.6, the initial `POST` request to the chat poll API at `/api/ajax/antonchat.php?update=` retrieves the channel's member list, while subsequent requests are used to monitor changes and messages.

If the current user does not have the necessary permissions to access the chat, the server still returns the member list, though subsequent requests will fail to retrieve updates or messages. This allows a malicious user to, for example, monitor which students are attending class on any given day.

5.4. Disclosure

As Antonwelt is a paid service, we initially reached out on September 11th, 2020, inquiring about research access. We promptly received a response, and were granted access akin to that of an enrolled school on September 13th.

By September 30th, we had concluded our review, and shared our findings with Antonwelt developers. Receipt was acknowledged by the October 8th, and on November 6th, we were notified that the SQL injection, PHP escaping, and chat monitoring flaws had been rectified. The remaining issues were projected to be corrected by Q1 2021.

Chapter 6.

Schlaukopf.at

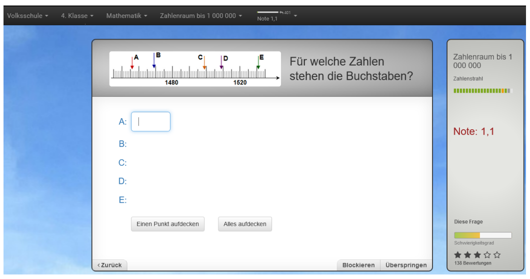
Schlaukopf.at is an interactive learning platform, providing tens of thousands of exercise questions for students aged six to eighteen [Hic+12a].

It is developed by Gomaringen-based Werbe-Medien-Internetagentur M. Hicke [Hic+12b]. It is recommended by the Austrian Federal Ministry of Education [BMBWF] and the Styria Board of Education [BdStmk].

6.1. Features

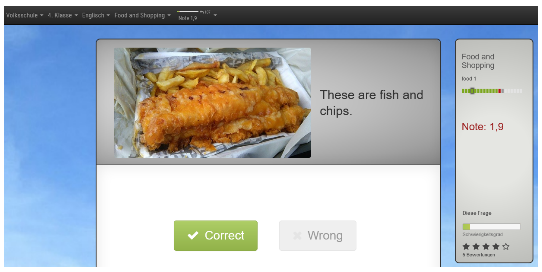
Schlaukopf.at provides multiple-choice and freetext questions for children. The content of the questions is tailored towards the specific curricula used in primary and secondary education in Austria, Germany and Switzerland.

After choosing the desired country, grade, and subject matter, children are presented with repeated prompts which dynamically adjust their difficulty to the user's performance. Some examples are shown in Figure 6.1.



Für welche Zahlen stehen die Buchstaben?

(a) Math for 10-year-olds

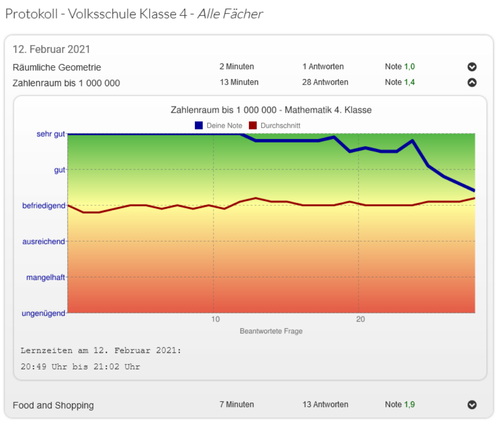


These are fish and chips.

(b) English for 10-year-olds

Figure 6.1.: Example problems on Schlaukopf.at

By default, users are assigned a guest account, which tracks their performance across sessions. If they wish, they can register with their email address and a password, which allows them to work on multiple devices. Parents, teachers, or other guardians can also associate the pupil's account as a "tutee" of their own account. This allows them to track the child's activity and performance statistics. They can also set goals, completion of which will be met with encouraging messages or other rewards. These features are shown in Figure 6.2.


Protokoll - Volksschule Klasse 4 - *Alle Fächer*

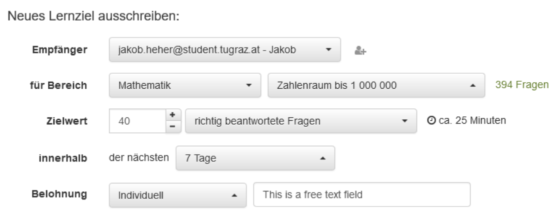
12. Februar 2021

Räumliche Geometrie	2 Minuten	1 Antworten	Note 1,0
Zahlenraum bis 1 000 000	13 Minuten	28 Antworten	Note 1,4

```
Lernzeiten am 12. Februar 2021:
20:49 Uhr bis 21:02 Uhr
```

Food and Shopping	7 Minuten	13 Antworten	Note 1,9

(a) Detailed monitoring of a tutee's activity


Neues Lernziel ausschreiben:

Empfänger	jakob.heher@student.tugraz.at - Jakob
für Bereich	Mathematik · Zahlenraum bis 1 000 000 · 394 Fragen
Zielwert	40 · richtig beantwortete Fragen · ca. 25 Minuten
innerhalb	der nächsten 7 Tage
Belohnung	Individuell · This is a free text field

(b) Setting a goal for a tutee

Figure 6.2.: Tutor features on Schlaukopf.at

6.2. Functionality

The Schlaukopf.at servers send an `X-Powered-By: PleskLin` header, which appears to merely indicate the hosting solution used. Some files use a `.php` extension, which may suggest that the entire page uses PHP (see Section 2.5).

After an initial dump of a concatenated master JavaScript file, which includes a number of minified libraries, any user interaction with the UI causes a `POST` request to be sent to the server.

Typically, asynchrous web applications will retrieve data via a structured API, then process it using already-loaded logic. Instead, for Schlaukopf.at, these requests are used to retrieve JavaScript code, which is then executed. This code appears to be dynamically generated, and has data embedded in it. An example is provided as Listing 6.1.

```
x_time = new Date().getTime();

$('#divModalDialog')
  .html("... <option value=\"1631529\">gast1631529@schlaukopf.at - <\/option> ... ").modal();
$('.selectpicker').selectpicker();

$('.active_popover').removeClass('active_popover').popover('hide');

$('#toggleicon_0').unbind('click').on('click', function() {
  if($('#toggleicon_0').hasClass('fa-chevron-circle-down'))
  {
    $('#toggleicon_0').removeClass('fa-chevron-circle-down').addClass('fa-chevron-circle-up');
    $('#toggleicon_0').attr('title','Details␣verbergen');
    xs('chart', 'protokollexpand', '0', '11590', '1610082868', 'Buchstaben␣-␣Deutsch␣1.␣Klasse');
  }
  else
  {
    $('#toggleicon_0').removeClass('fa-chevron-circle-up').addClass('fa-chevron-circle-down');
    $('#toggleicon_0').attr('title','Details␣anzeigen');
    $('#diagram_0').hide('clip');
  }
});

screen_optimization();
```

Listing 6.1: Clicking on "My Statistics"

To identify the user, a number of cookies are set. `elearning_userId` and `s_ids` both hold the current user's user ID, which appears to be a six-digit integer. Meanwhile, `elearning_encrypted` and `s_crypt` hold 128 base64-encoded bits. If a user is logged into multiple accounts, the `elearning_` cookies hold the current account's information, while the `s_` cookies hold a list of all available accounts.

To log a user in, a `POST` request to `?p1=user&p2=login` is made. It includes, among a number of other static parameters, the user's email address and the same value that will later be stored in the `_crypt` cookie pair. This suggests that the value is derived from the password. In response, the server sends a JavaScript response which sets fields on a global `user` JavaScript object before invoking its `.saveUserToCookie()` method, which appears to set the `elearning_` cookies previously observed. An example of such a response is provided as Listing 6.2.

```
x_time = new Date().getTime();

user.id = '834814';
user.encryptedPassword = '...';
user.email = '...';
user.firstLogon = '2021-01-05 05:16:06';
user.saveUserToCookie();
location.reload(true);

xs_return = "";
```

Listing 6.2: A successful login attempt

When the user reconnects to the site with an appropriate cookie pair, the JavaScript response to the initial request to `?p1=user&p2=initialize` also sets the same values, among a number of other things. We thus conclude that this password-derived "encrypted" value is used for both authentication and re-authentication, is equivalent to the user's password, and is stored in JavaScript memory.

If a user indicates they have forgotten their password, they are prompted to enter both their email address and a new password. The server then sends them a confirmation message containing a hyperlink. Clicking on the hyperlinks changes the account's password to the specified new password, and logs the user in. The message is shown in Figure 6.3.

6.3. Analysis

While we did not investigate Schlaukopf.at as thoroughly as we did ANTON in Chapter 4 or Antonwelt in Chapter 5, we nevertheless discovered multiple issues.

6.3.1. Password "encryption"

The password-equivalent "encrypted password" value stored in JavaScript memory by Listing 6.2 appeared suspicious due to its particular length of 128 bits. Indeed, this suspicion was swiftly validated – the value is the unsalted MD5 digest of the user's password. As outlined in Section 2.7 and Section 5.3.2, MD5 is utterly unsuitable for password storage.

Additionally, even if a suitable algorithm were used, storing a value that is equivalent to the user's password in JavaScript memory is not recommended, as it allows trivial extraction via a successful XSS attack, as described in Section 2.9.

6.3.2. XSS vulnerabilities

As illustrated in Listing 6.1, Schlaukopf.at widely uses jQuery's `.html()` method. The .html() method parses the specified string as HTML and inserts it into the document. This can easily allow malicious content to execute scripts in a first-party context. Especially if used with dynamically constructed strings, as is the case here, this makes it a very attractive vector for XSS attacks.

Schlaukopf.at provides very detailed user tracking. Yet, as it offers no inter-user communication features, it is surprisingly hard for an attacker to have their input impact the target's UX. However, we noticed that the "decline tutor request" email button hyperlinks to `https://www.schlaukopf.at/?showdialog=tutors&accept=0&newtutor=TUTOR_EMAIL`. When the page is loaded, a small modal dialog is then displayed, indicating that the tutor request from the specified email address has been declined.

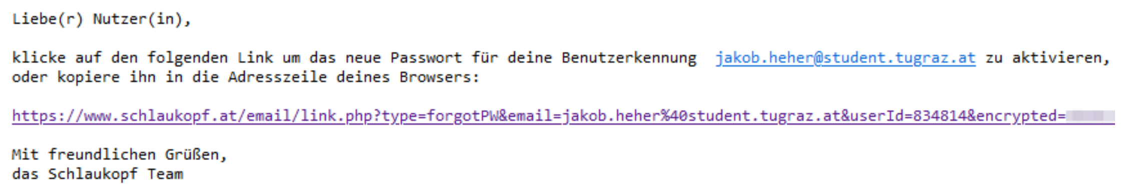
This modal dialog is dynamically constructed as HTML on the server, and sent back to the client as JavaScript, where the dialog is then parsed using `.html()`. The prospective tutor's "email address" is inserted as-is. As a result, an attacker can execute arbitrary JavaScript in a first-party context if a user clicks a link they control. The same attack vector may be used if the victim visits a third-party site controlled by the attacker, which then causes their browser to open the malicious link in an invisible `<iframe>`, similar to the scenario described in Section 2.10.

6.3.3. The "Reset Password" UX flow

If a user has forgotten their password, it is common to let them re-authenticate using an alternate authentication factor, such as access to their email address. Schlaukopf.at's implementation of this common process is unique in that it places control over the new password on the "unauthenticated" side of the flow, before the user has clicked the confirmation link.

As a result, an attacker can cause a fairly nondescript message – as pictured in Figure 6.3 – to be sent to the genuine user. If the contained link is clicked on, the attacker gains control of the victim's account, without the victim necessarily being aware of what just happened.

This parallels the concerns with administrative account resets we discuss in Section 8.2. However, here, the presumed-legitimate user is already required to take action as part of the process. Thus, it is trivial to rectify the issue by moving the new password entry to *after* the user has confirmed their identity, rather than before.



```
Liebe(r) Nutzer(in),

klicke auf den folgenden Link um das neue Passwort für deine Benutzerkennung  jakob.heher@student.tugraz.at zu aktivieren,
oder kopiere ihn in die Adresszeile deines Browsers:

https://www.schlaukopf.at/email/link.php?type=forgotPW&email=jakob.heher%40student.tugraz.at&userId=834814&encrypted=[redacted]

Mit freundlichen Grüßen,
das Schlaukopf Team
```

Figure 6.3.: A password reset message from Schlaukopf.at

6.3.4. Unauthorized tracking of users

The "Statistics" dialog includes a dropdown box of accounts that the current user can view statistics for, as seen in Listing 6.1. Selecting a target initiates a `POST` request to `?p1=dialog&p2=statistics` with, among other parameters, `p3[user]`[1] being set to the target user's ID. The response is JavaScript code which replaces the modal dialog's content with the target user's statistics data.

However, the server handler for this request does not verify that the specified user ID is one that the current user should actually have access to. Thus, any user's data can be requested given their user ID. Alternately, data for *all* users can presumably be collected by iterating over all 1,000,000 valid user IDs.

As shown in Figure 6.2a, this access would let an attacker monitor the victims' usage of the site in real time. It would also let them create profiles of victims' daily schedule, their behavior, or their performance over time.

6.4. Disclosure

We initially reached out to the Schlaukopf.at support contact (`email@schlaukopf.at`) on January 14th, 2021, and received a prompt response that same day. We then directed our findings to the indicated address on January 18th, and receipt was acknowledged on the 19th.

On February 6th, 2021, we were informed that Schlaukopf.at considered all vulnerabilities we had disclosed to have been resolved.

[1]This is a POST parameter, not part of the URL query string – Schlaukopf.at uses both in its requests

Chapter 7. LearningApps.org

LearningApps.org is “a Web 2.0 application, to support learning and teaching processes with small interactive modules” [Hie12]. It allows teachers to easily create their own specialized content through use of generic templates.

It is developed and maintained by Däniken-based Swiss non-profit Verein LearningApps interaktive Bausteine [Hie+15]. It is recommended by the Austrian Federal Ministry of Education [BMBWF].

7.1. Features

LearningApps.org provides a number of template building blocks for creating educational content. Users can then populate these templates with content, or combine templates, to construct learning “apps”. They can then share these apps with others, or list them in a publicly accessible directory. Some examples of such building blocks are pictured in Figure 7.1.

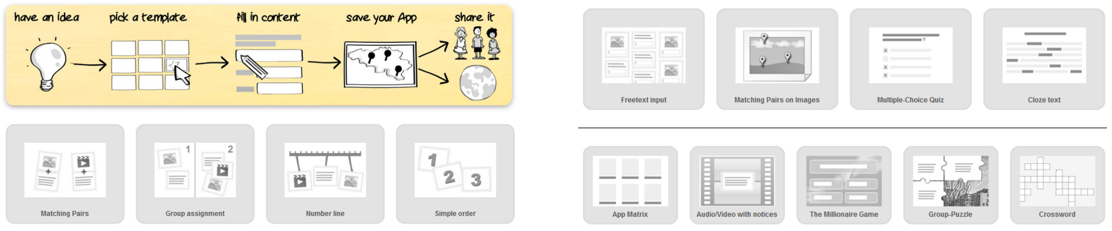
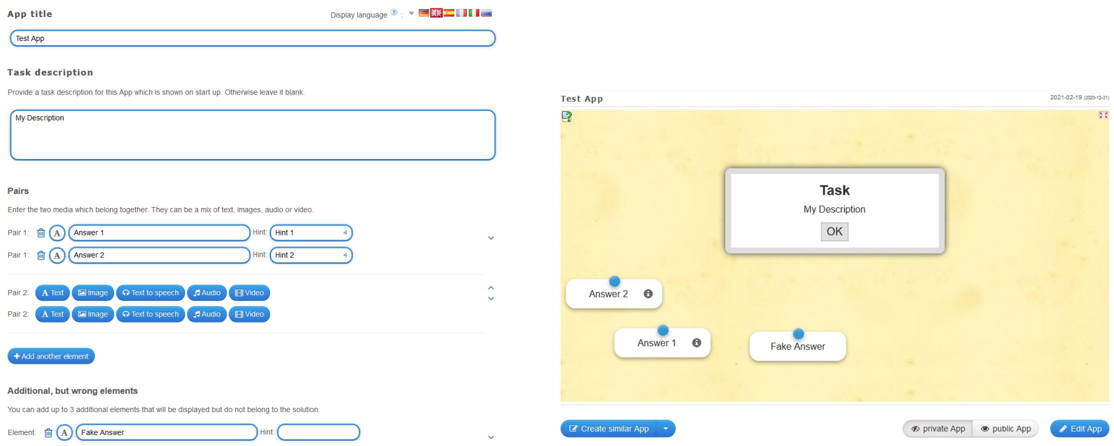


Figure 7.1.: Some of the available building blocks in LearningApps.org

When a user creates an app from a given template, they assign a name and description, which are presented to the viewer upon opening the app. Then, the author populates the template with content, in the form of text, audio, video, or others. The result can then be saved, shared via a direct link, or set to be publicly visible. This is pictured in Figure 7.2.

Users can create “classrooms”, becoming the teacher of that classroom. Other users can then join the classroom as students by clicking on a provided link. Teachers are provided administrative controls to their students’ accounts, and can monitor their activity, as well as unilaterally change their passwords.


App title

Test App

Task description

Provide a task description for this App which is shown on start up. Otherwise leave it blank.

My Description

Pairs

Enter the two media which belong together. They can be a mix of text, images, audio or video.

Pair 1:	Answer 1	Hint: Hint 1
Pair 1:	Answer 2	Hint: Hint 2

Additional, but wrong elements

You can add up to 3 additional elements that will be displayed but do not belong to the solution.

Element: Fake Answer Hint:

Test App

Task

My Description

Figure 7.2.: Creating an app in LearningApps.org

A direct message system is also provided, allowing users to send simple freetext messages to each other.

7.2. Functionality

A `PHPSESSID` cookie is set upon initial connection to `https://learningapps.org/`. This suggests that the platform uses PHP (see Section 2.5). The cookie has the `HttpOnly` and `Secure` attributes set, but does not specify a `SameSite` attribute.

For clarity's sake, we will henceforth refer to a type of app as a "template". For example, "crossword puzzle" is a "template". We will refer to a single instance of a template, populated with user data, as an "app". For example, a crossword puzzle populated with a list of animal names and associated photographs would be an "app".

In requests, templates are identified to by an internal numeric ID. Individual apps are identified to by a "GUID", which appears to be a random string of lowercase letters and numbers.

7.2.1. App creation and usage

Upon choosing the desired app template, the user is redirected to `/create?new=TEMPLATE_ID`. This page, pictured in Figure 7.2, loads `/AppCreator.js` as a JavaScript file. After the user has finished entering data, clicking the "Preview" button sends a `POST` request to that same JavaScript file with a number of parameters:

- `LearningApp_action`: `"preview"`
- `LearningApp_tool`: the chosen template's ID
- `LearningApp_version`: `""`, a blank string
- `LearningApp_language`: the chosen language identifier, a string (e.g., `"EN"`)
- `LearningApp_GUID`: the string `"null"`
  (if an existing app is being edited, this is set to that app's GUID)

- `LearningApp_title`: the app's title as a string
- `LearningApp_task`: the app's description text, shown when opening it
- `LearningApp_help`: the help text, accessible from a button in the UI
- `LearningApp_derived`: the string `"undefined"`
  (if the app is a modified copy of another app, this is set to that app's GUID)
- `save`: the string `"true"`
- `LearningApp_spamcheck`: a random numeric string
  (this appears to be dynamically embedded into `AppCreator.js` on page load)

This results in a plain text response containing the newly-created app's GUID. Then, a frame is loaded from `/show.php?id=APP_GUID`, showing the newly-created app. If the user confirms the preview, the `POST` request above is sent with `LearningApp_action` set to `"save"`, and `LearningApp_GUID` set to the app's newly-assigned GUID.

Viewing an app – be it in the preview screen or "for real" – loads a total of three JavaScript files. First, `/data?jsonp=1&id=APP_GUID&version=VERSION` populates a global variable with app-specific data. Then, `/AppClientServer.js?jquery` is loaded, which contains the general framework used by the individual templates. Finally, the template-specific `/tools/TEMPLATE_ID/VERSION/script.js` sets up the actual app using the data and framework previously loaded.

7.2.2. User profile and Direct Messaging

Clicking on "account settings", or the "you have new messages" notification, takes the user to `/my.php`. This page shows the user's own profile and also embeds recent direct messages. "Change password" option and "change email address" options are also offered.

To change the user's password, a `POST` request to `/change-password.php` is made with parameter `password` set to the current password, and `passwordc` and `passwordcheck` both set to the new password.

To change their email address, a `POST` request to `/update-email-address.php` is made with a lone `email` parameter set to the desired new address.

When viewing an app, clicking the creator's name takes the user to their profile at `/user/USERNAME`. Clicking the "message" icon pops up a frame loaded from `/message.php?sendto=USERID&app=&reply=`. To send a message, a `POST` request is made to the same URL with four parameters:

- `action`: the string `"send"`
- `sendto`: set to the recipient's user ID
- `messagetitle`: the message's subject line as a string
- `messagetext`: the message's content as a string

7.3. Analysis

Upon cursory inspection, it became clear that the main `AppClientServer.js` framework makes frequent use of `.html()` with dynamically generated strings. This is identical to the issues previously encountered with Schlaukopf.at (see Section 6.3). The `.html()` method parses the passed string as HTML before inserting it into the document. Thus, any attacker that can control the parameter can execute arbitrary JavaScript in a first-party context. This makes it a useful vector for an XSS attack (see Section 2.9).

7.3.1. XSS vulnerabilities

Initial basic tests immediately led to a vulnerability, as the "app description" field is passed to `.html()` via the insecure `createModalDialogFrame` method. This allows an attacker to execute arbitrary JavaScript in the user's browser when they view the attacker's app. Indeed, the app creation dialog performs no modification to the user's input. This even allows injection of malicious HTML elements to be performed via the UI, rather than needing to manually tweak HTTP requests.

Additionally, both the subject line and content of direct messages are inserted using `.html()`. This allows an attacker to run arbitrary JavaScript in the victim's browser by simply sending them a direct message. In particular, message contents are already displayed in the user's profile, rather than requiring explicit navigation to the message. Thus, merely clicking on "you have new messages" is enough to seal the victim's fate.

7.3.2. XSRF vulnerability

Altering a user's registered email address requires no parameters beyond the desired new email address. Additionally, a "reset password" flow is provided. Thus, control over the registered email address is equivalent to being able to access the account.

Even though changing the user's password requires knowledge of the current password, changing their registered email address does not. Additionally, the `PHPSESSID` cookie used by LearningApps.org does set the `SameSite` attribute, similar to that used by Antonwelt (see Section 5.3.4).

As a result, a malicious third-party site can execute an XSRF attack (see Section 2.10) against a visitor. If they are also logged into LearningApps.org, this allows the attacker to cause the victim's browser to send a properly-authenticated "change email address" request. This changes their registered email address to one controlled by the attacker. Afterwards, a password reset email can be requested to that new address, and the account can be taken over.

7.4. Disclosure

We initially reached out to the LearningApps.org support contact (`mail@learningapps.org`) on January 14th, 2021, received a response the same day, and disclosed our findings on January 19th.

On January 20th, we received a response. LearningApps.org acknowledged the XSRF vulnerabilities and addressed them, but declined to consider the XSS possibilities as a vulnerability. According to their response, "[they] do have a quite open plattform which allows users to use full HTML and JavaScript code in their work", so "[they] deliberately allow this and have discussed the potential harm this can cause". Additionally, they actively review "scripts and the use of HTML in user-generated content. [They] then manually decide whether the code is malicious and delete the content if necessary."

Chapter 8.

Discussion

We would like to emphasize that it is crucially important to not dismiss the flaws we outline in Chapters 4 to 7 as merely being the fault of individual designers. Instead, we feel strongly that one must try to understand how such flaws come to be, so that recreating them in the future can be avoided through systemic changes.

Thus, we will be using this section to discuss some systemic factors that contribute to security flaws being introduced into software. Section 8.1 outlines our thoughts on the importance of good, thought-out documentation, and factors we consider necessary to achieve such a thing.

8.1. On defaults and documentation

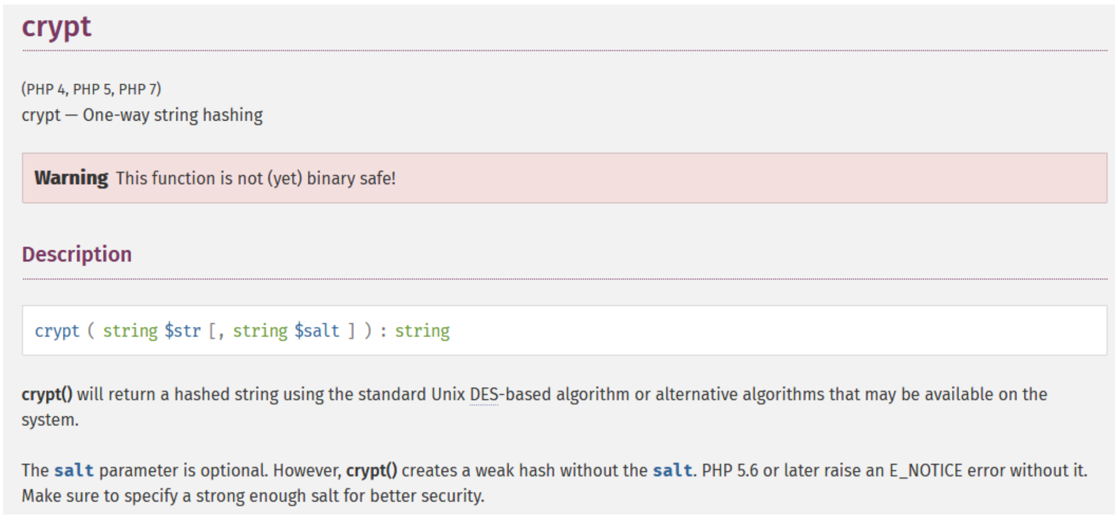
In the work of Xie, Lipford, and Chu [XLC11], they find that "most of [their] participants [had] a reasonable awareness and knowledge of software security issues". However, they also report that "the business logic of an application [is viewed as] the primary concern" – or, as one respondent they quote directly eloquently puts it: *"Now the only way you can fit a 3-month project into 3 weeks is to cut a whole lot of corners. Security was one of those corners that got cut."*

We therefore find it reasonable to assume that many developers are generally aware of common security concerns and make some conscious effort towards addressing them. However, they may not necessarily possess an in-depth understanding of the technologies involved, and would thus be heavily reliant on first-party documentation, provided examples and defaults. This would be further exacerbated by other outside factors, such as time pressure, that encourage valuing a *functional* application over a *secure* application; after all, once you have logic which *works*, it gets subsequently more difficult to justify any time spent investigating its security.

In Section 8.1.1, we offer some general thoughts on documentation usage, and associated concerns for authors. In Section 8.1.2, we then review some of the first-party documentation applicable to the issues raised throughout Chapters 4 to 7, and offer opinions on how its shortcomings may have contributed to their existence.

8.1.1. Some general considerations

When evaluating documentation covering a subject the reader is intimately familiar with, it is often easy to project one's existing knowledge onto the actual page being reviewed.


crypt

(PHP 4, PHP 5, PHP 7)
crypt — One-way string hashing

Warning This function is not (yet) binary safe!

Description

```
crypt ( string $str [, string $salt ] ) : string
```

crypt() will return a hashed string using the standard Unix DES-based algorithm or alternative algorithms that may be available on the system.

The `salt` parameter is optional. However, **crypt()** creates a weak hash without the `salt`. PHP 5.6 or later raise an E_NOTICE error without it. Make sure to specify a strong enough salt for better security.

Figure 8.1.: `crypt()` — PHP manual

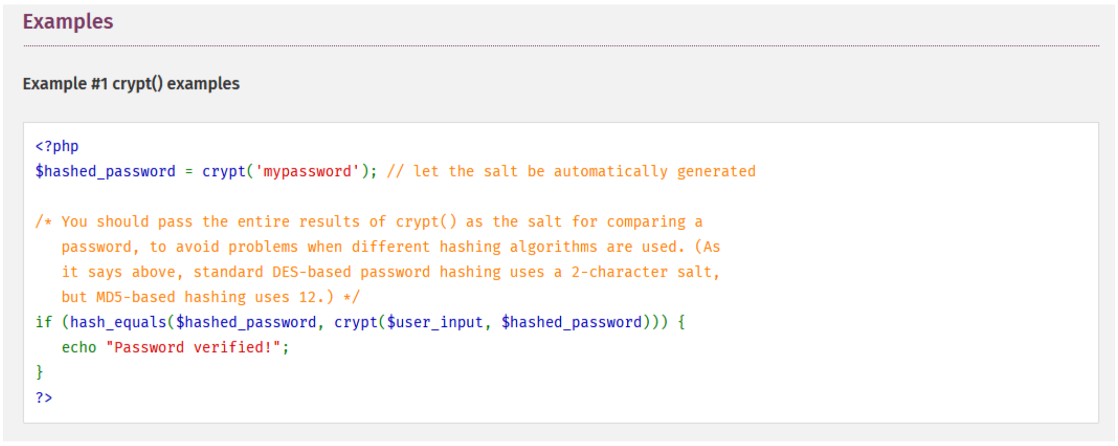
However, doing so fails to realize a fundamental truth, which may seem obvious when put into writing – the intended target audience is indeed *not* intimately familiar with the subject matter, or they would likely not be reading the document in question.

Thus, it does not suffice for a good manual page to just “check all the boxes”, include all individual bits of relevant information, and blame the reader for “not reading the ****** manual”. Instead, the party that has familiarity with the material – the writer – must actively prioritize certain pieces of information at the expense of others; they must, based on their knowledge, weigh which pieces are absolutely crucial for the party without that same familiarity – the user – to take away from their visit. For example, an introduction to heap memory in C would place `free`-ing the `malloc`-ed pointer front and center, not a discussion on allocation patterns and heap fragmentation.

Furthermore, consider that, as Abdalkareem, Shihab, and Rilling [ASR17] found, not only do developers commonly insert code snippets from online sourcing, but there is even correlation between these code snippets and bugs in the respective files. Additionally, Xie, Lipford, and Chu found that “participants seemed to trust reused code, even those with high security awareness” [XLC11]. These factors combine to make the security of any “usage examples” provided within documentation absolutely *crucial* to the quality of the overall product; if any one example demonstrates insecure usage, it undermines any security-minded paragraphs warning the user away from such usage.

8.1.2. PHP `crypt()`

In Section 5.3.2, we found that passwords were stored in the database as MD5 digests, as generated by PHP’s `crypt()` method. `crypt()` is deeply flawed in multiple ways. It not only defaults to MD5 if no further parameters are specified – use of a more secure algorithm also requires user code to independently generate a full, secure and



Examples

Example #1 crypt() examples

```
<?php
$hashed_password = crypt('mypassword'); // let the salt be automatically generated

/* You should pass the entire results of crypt() as the salt for comparing a
   password, to avoid problems when different hashing algorithms are used. (As
   it says above, standard DES-based password hashing uses a 2-character salt,
   but MD5-based hashing uses 12.) */
if (hash_equals($hashed_password, crypt($user_input, $hashed_password))) {
   echo "Password verified!";
}
?>
```

Figure 8.2.: `crypt()` examples — PHP manual

appropriately-formatted salt string. Since 2013, PHP has provided a far superior built-in alternative in `password_hash()`, which defaults to using an actual password hashing algorithm with a securely-generated salt. Additionally, `password_hash()`'s counterpart, `password_verify()`, is fully backwards compatible with hashes generated using `crypt()`. Thus, we struggle to envision any situation in which usage of `crypt()` in any newly-written application is appropriate.

Keeping this in mind, Figure 8.1 shows the PHP manual page [Pcryp] for the function. Indeed, there is a red warning box that immediately stands out on this page, warning us that `crypt()` "is not (yet) binary safe". There is no further elaboration on this, and this is the only occurrence of the word "binary" in the page. Being concerned about this, we search the web for the phrase "php crypt binary safe", and the results prominently feature an online discussion thread [red12] which not only explains the meaning of "binary safe", but also informs us that "[`crypt()` is] designed to crypt password strings [*sic*]".

While the contents of the featured result are not something that documentation authors have control over, we would generally consider it wise to elaborate on any prominently-placed warning banners to avoid sending readers on uncontrolled detours.

Disregarding the warning banner, we encounter the method signature, followed by a fairly long-winded explanation of its functionality. In the second and third paragraph of this explanation, we find the only references to `crypt()`'s deep flaws contained in the page, warning that "`crypt()` generates a weak hash without the salt", encouraging the user to "specify a strong enough salt for better security", and noting that "[u]se of `password_hash()` is encouraged". While this may seem sufficient to educated readers armed with knowledge of the flaws we outlined previously, we would raise doubts whether this language is strong enough to discourage use for the casual reader. In particular, we suspect that phrases like *better security* and *encouraged* do not sufficiently impress the severe weakness of the default behavior. Additionally, these notes are placed past the first paragraph break, in a fairly technical section on the behavior of `crypt()`.

A user might lose interest here and take this as guidance to move to the "Examples" section (see Figure 8.2). Here, they would find that the first, prominently-placed example, demonstrates password authentication logic that indeed uses the default MD5 implementation.

To summarize, given that most developers should *never* use `crypt()` in an application due to its deep flaws, we find its manual page to be woefully lacking in indicators of this. Indeed, we believe that it would be easy for a time-constrained developer to glance past the weakly-worded discouragements offered by the page, and to readily find a ready-to-use example of `crypt()` usage that is perfectly functional, yet utterly insecure.

8.2. On administrative account resets

Online services, generally speaking, identify a user using some subset of their known authentication factors. Typical factors are knowledge of a password, possession of a device generating a one-time passcode, or access to a specified email account. Even then, it is inevitable that genuine users will be left without any of their authentication factors being available to them at times.

Educational products aimed at young children, in particular, may not require some of the common authentication factors. For instance, it might simply not be assumed that users have their own email account. In such a situation, an account would typically be permanently inaccessible.

Here, it is common to allow an extraordinary procedure to restore access to an account, which we will here term an "administrative account reset". This allows an authenticated administrator user to waive the requirement for a pre-established authentication factor. In doing so, they substitute the platform's trust in a user's authentication factor with the platform's trust *in their person.*

This possibility for human intervention is generally considered desirable in a system. However, allowing such circumvention of the established authentication process carries quite clearly carries potential for abuse. We present some potential considerations and safeguards.

Be impossible to miss. When designing a notification regimen around an administrative account reset, it is important to keep in mind that use of this mechanism is extraordinary. Therefore, many of the usual considerations around not being overly intrusive can be disregarded. Instead, we would suggest designing the system to generate as many notifications as possible, via as many redundant contact mechanisms as are available. For instance, if an administrator forces the registered email address to change, this might generate a notification to the previous address, *and* an in-app intrusive dialog upon next login, *and* a revocation of all existing login sessions, *and* invalidation of the user's password. The main priority here is making sure that a genuine user will not remain unaware of the feature's use.

Allow for human intervention. Humans are, generally speaking, slower than computers. Thus, even in a situation where a genuine user is successfully made aware of an abusive attempt to gain control of their account, they may not be able to intervene in time to prevent unauthorized access. To combat this, we would suggest considering the introduction of *timed delays* into the access reset flow, after notifications have been made as per the above. This allows a genuine user to notice the attempted access, and take steps to assert its unapproved nature, before any real damage can be done.

Reduce the impact of compromise. Administrators, of course, are still users, and their accounts may also be compromised. In such a scenario, an attacker likely has access to the administrator's account for only a limited time period. Steps should be taken to minimize potential abuse of account reset features during this interval. To do so, we would recommend both rate limits and stringent logging on all administrative account resets. The former mitigates the harm an attacker can cause over a period of time, while the latter allows for post-incident investigation and remediation of potential knock-on compromises.

Closing thoughts. Of course, it is important to consider that each of these constraints also limits *legitimate* account resets. One needs to weigh any negative impact this is likely to have against the likelyhood that the feature will need to be used.

For example, requiring a two-day waiting period to re-gain access to an online message board will generally be a mere inconvenience. Meanwhile, requiring a two-day waiting period whenever a student forgets their ANTON login code (see Section 4.3.4) is likely to be infeasible.

8.3. On adversarial thinking

Throughout this work, we have encountered a plethora of flaws in real-life software, affecting thousands of students. Some were caused by poor documentation and dangerous choice of default values, as discussed in Section 8.1. However, we posit that a significant portion results from a lack of adversarial thinking at the design stage.

Adversarial thinking is a somewhat vague concept. It has been described as "an intuition for identifying assumptions that can be violated to achieve some goal" [Sch13], as "the ability to anticipate the strategic actions of hackers" [Ham+17], or as "analyz[ing] cybersecurity from the strategic perspective of [...] adversaries" [Kat19].

Regardless of the particular description chosen, this mindset represents a necessary condition for designing secure software. Indeed, we feel that it would be crucial that aspiring developers be taught this viewpoint consistently, and from an early point in their education – rather than having it consigned to specific courses, and otherwise only represented in "in production use, you would take care to ..." footnotes. Proper consideration of edge cases and potential malicious inputs should be a requirement of any assignment, rather than a "if we were to do this properly, we would ..." hand-wave. Only practice makes perfect.

Which values are under control of a potential attacker? What kind of values might they introduce? What assumptions is your code placing on the data it is working with? Could an adversary violate these assumptions? Are these assumptions guaranteed by code you can trust? Can you guarantee that our entry points are invoked in the order you expect? How do you deal with invalid requests? Can you determine if a request is invalid right away, and reject it before it has any impact on the state? Can you coerce any input into a form that is guaranteed to be benign and valid?

These questions, and many more, are integral to building a secure system – yet students are typically not exposed to them on a regular basis. While they might be made aware that they should "pay attention to security", they do not practice doing so consistently.

We posit that adversarial thinking, and associated questions, should be viewed as an integral part of software design. It should be taught early on, then refined and evaluated constantly – just like it is considered obvious that code submitted for an assignment must be *functional*, it should be just as obvious that the same code must be *secure* against basic classes of attacks.

Chapter 9. Conclusion

In the wake of the CoViD-19 pandemic reshaping the world as we know it, we set out to investigate the state of online learning platforms in the DACH countries.

We discussed general concepts that arise in an educational context. We categorized users as *students*, *teachers* and *administrators*, and group them into *classrooms* and *schools*. We identified some potential attacker goals – *impersonation* of a genuine user, *surveillance* of their activities, *suppression* of their access, or *reconnaissance* of many different users.

We identified three public institutions that provide resource listings for online learning – the Austrian Ministry of Education [BMBWF], the Styria Board of Education [BdStmk], and the Standing Conference of the Ministers of Education of Germany [KMK]. We compiled the respective lists into a single master list and selected four platforms of particular interest from it. We outlined our review methodology, which is based on a cycle of behavior analysis and iterative modification. We considered the ethical implications of conducting security analysis against a production system and discussed the limits we had placed on our activities ahead of time. We outlined the process we would use to disclose the discovered vulnerabilities.

Across four chapters, we dissected the four platforms of interest previously chosen – ANTON [sola] and Schlaukopf.at [Hic+12a], two platforms offering access to pre-defined interactive content; Antonwelt [Hum+18c], a creative workshop that allows students to explore writing short stories in a colorful environment; and LearningApps.org [Hie12], a platform allowing teachers to easily create their own specialized content through use of generic templates.

We demonstrated fundamental flaws in ANTON's API design, which would have allowed an unauthenticated attacker to gain persistent covert access to all enrolled students' data. We outlined an oversight in Schlaukopf.at's validation logic that granted access to activity logs of any user, and showed that passwords were stored as unsalted MD5 digests. We discovered a small mistake in Antonwelt's input escaping logic, which would have allowed us to inject malicious SQL instructions and gain full access to all enrolled students' data. We explored a number of ways in which LearningApps.org permits a malicious author to include JavaScript in their content, which will be surreptitiously executed for any user viewing the content.

We took to our proverbial soapbox to opine on various systemic factors, which we suspect to have played a part in the creation of the vulnerabilities we previously outlined. We reviewed the design and documentation of PHP's `crypt()` function, and found it to be confusing, disorganized, and encouraging of insecure code such as that found in Antonwelt. We discussed administrative account reset procedures, which are widely implemented by the platforms reviewed. We posit that any such procedure should be designed to be *impossible to miss* for the target user, should be *slow enough* to allow human intervention, and should be *stringently documented*. Furthermore, we described the need for heightened emphasis on adversarial thinking in educating the software engineers of tomorrow. We emphasize that this “attacker mindset” should be taught consistently and from an early point, and that consideration of edge cases and potential malicious inputs should be a requirement for any assignment.

9.1. Future work

We close with some musings on future directions.

Our work here, though limited in its sample size, suggests that there may be many yet-undiscovered vulnerabilities lurking in educational software. Further reviews of such products will have tangible real-world impact on the safety of students during the pandemic and beyond, yet their providers frequently lack the budget – and the incentives – to commission them.

We consider it likely that this extends beyond just the educational context, and into narrowly targeted applications in many other areas that suffer from a similar lack of funding. Exploration of further such areas, while perhaps not as scientifically titillating as theoretical research, may still yield insightful data on common pitfalls that should be addressed on a systemic level – and simultaneously allow the communities served to be more secure.

Finally, as mentioned throughout, we believe that many of the issues discovered are rooted in impractical or unintuitive cryptographic interfaces, which are unsuitable for the lay developer’s use. While we chose PHP’s `crypt()` as an example, PHP is far from the only language with such baggage [Dre20]. Much work remains to be done in this area, both in designing usable interfaces and in convincing language maintainers that this is an issue worth tackling. After all, the best cryptography is useless if developers commonly defeat it by making foreseeable mistakes.

Appendix A. Processed resource listings

Name	[BMBWF]?	[BdStmk]?	[KMK]?	Available?	Sign-up needed?	Native app exists?	Native app needed?	JavaScript needed?	Free-form comms?
AMIRA Leseprogramm		✓		✓	✗	✗	✗	✓	✗
`amira-lesen.de`	stories for children w/ german as second language								
Antolin Lesespiele	✓	✓		✓	✓	✗	✗	✓	✓
`antolin.westerm...`	quizzes to encourage reading of books and news								
ANTON	✓	✓	✓	✓	✓	✗	✗	✓	✗
`anton.app`	learning aid for various subjects								
Antonwelt	✓			✓	✓	✗	✗	✗	✓
`antonwelt.schule`	students write free-form stories								
BetterMarks			✓	✓	✓	✗	✗	✓	?
`de.bettermarks.com`	paid interactive math quizzes								
Blockly Games	✓			✓	✗	✗	✗	✓	✗
`blockly.games`	visual programming games								
Conni-Apps	✓			✗	✗	✓	✓	✗	✗
`conni.de`	3/4 apps lead to 404, only remaining one is a learning app for reading the time								
Corona-School			✓	✓	?	?	?	?	?
`corona-school.de`	tutor matchmaking								
Deutsche Verben	✓			✓	✗	✓	✓	?	?
`play.google.com`	german grammar assistant								
Deutscher Bildungsserver			✓	✓	✗	✗	✗	✓	✗
`bildungsserver.de`	official platform for documents on the german education system								
Deutschlerner-Blog		✓		✓	✗	✗	✗	✗	✗
`deutschlernerbl...`	tips for learning german								
Die Politikstunde			✓	✓	✗	✗	✗	✗	✗
`bpb.de`	weekly video lecture for students								

Name	[BMBWF]?	[BdStmk]?	[KMK]?	Available?	Sign-up needed?	Native app exists?	Native app needed?	JavaScript needed?	Free-form comms?
digi.komp4	✓			✓	✗	✗	✗	✗	✗
`digikomp.at`	list of learning resources								
Digitale Flaschenpost	✓			✓	✗	✗	✗	✗	✗
`edugroup.at`	digital literacy resources								
Duolingo		✓		✓	✓	✓	✓	✗	✗
`de.duolingo.com`	professional language trainer								
Edmodo	✓			✓	✓	✓	✗	✓	✓
`new.edmodo.com`	startuppy social media for schools								
EDU-Puzzle	✓			✓	✗	✗	✗	✓	✗
`baa.at`	quiz listing								
Edugroup		✓		✓	✗	✗	✗	✗	✗
`edugroup.at`	resource listing								
Eduthek	✓	✓		✓	✗	✗	✗	✓	✗
`eduthek.at`	resource listing								
einmaleins.at	✓			✓	✗	✗	✗	✓	✗
`einmaleins.at`	multiplication trainer								
GEOlino		✓		✓	✗	✗	✗	✗	✗
`geo.de`	magazine for kids								
Grundschule Arbeitsblä...		✓		✓	✗	✗	✗	✗	✗
`grundschule-arb...`	resource listing								
Grundschulkönig		✓		✓	✗	✗	✗	✗	✗
`grundschulkoeni...`	exercise sheet resource								
Helbling Verlag		✓		✓	✗	✓	✗	✓	✗
`helbling.at`	free access to books								
Illustratoren gegen Co...		✓		✓	✗	✗	✗	✗	✗
`illustratoren-g...`	resource listing								
InNote	✓			✓	✗	✓	✓	✗	✗
`play.google.com`	note-taking app								
Internet-ABC	✓			✓	✗	✗	✗	✗	✗
`internet-abc.de`	digital literacy resources								
IXL	✓	✓		✓	✓	✗	✗	?	✗
`de.ixl.com`	paid exercises								
Jugend & Volk Mathe	✓			✓	✗	✗	✗	✓	✗
`foxgames.at`	multiplication trainer								
Junge Klassik	✓			✓	✗	✗	✗	✗	✗
`junge-klassik.de`	music exercises								

Name	[BMBWF]?	[BdStmk]?	[KMK]?	Available?	Sign-up needed?	Native app exists?	Native app needed?	JavaScript needed?	Free-form comms?
KidsNet		✓		✓	✗	✗	✗	✗	✗
`kidsnet.at`	resource listing								
Kidsweb		✓		✓	✗	✗	✗	✓	✗
`kidsweb.wien`	resource listing								
KiGaPortal		✓		✓	✓	✗	✗	✓	✗
`kigaportal.com`	resource listing								
Klassenpinnwand	✓			✓	✓	✓	✗	✓	✓
`klassenpinnwand.at`	all-in-one educational portal								
Klassik4Kids	✓			✓	✗	✗	✓	✗	✗
`klassik4kids.at`	music education								
Learnattack			✓	✓	✓	?	?	✓	✓
`learnattack.de`	paid tutoring								
Learning Apps	✓			✓	✗	✗	✗	?	✓
`learningapps.org`	create & share apps								
LegaKids	✓			✓	✗	✗	✗	✗	✗
`legakids.net`	exercises for dyslexic kids								
Lern Deutsch	✓			✓	✓	✗	✗	✓	✓
`lernen.goethe.de`	cooperative german practice								
Lernburg	✓			✓	✓	?	?	✓	✗
`lernburg.at`	exercises for dyslexic kids								
Lernerfolg Grundschule	✓			✓	?	✓	?	?	✗$^?$
`apps.apple.com`	paid exercises								
Lernserver		✓		✓	✓	✓	✗	?	✗$^?$
`lernserver.de`	paid exercises								
Lesen mit Elbot	✓			✓	✗	✗	✗	✓	✗
`elbot.de`	CGI chatbot								
LL-Web		✓		✓	✗	✗	✗	✗	✗
`vs-material.weg...`	resource listing								
LMS.at	✓			✓	✓	✓	✗	?	?
`lms.at`	official austrian learning management system								
Mandala-Bilder		✓		✓	✗	✗	✗	✗	✗
`mandala-bilder.de`	resource listing								
Math Duel: 2 Player Ma...	✓			✓	✗	✓	✓	✗	✗
`play.google.com`	math problem competition								
Mathefuchs	✓			✓	✗	✗	✗	✗	✗
`mathe.aufgabenf...`	resource listing								

Name	[BMBWF]?	[BdStmk]?	[KMK]?	Available?	Sign-up needed?	Native app exists?	Native app needed?	JavaScript needed?	Free-form comms?
Mathway		✓		✓	✗	✗	✗	✓	✗
`mathway.com`	online algebra system								
Memrise		✓		✓	✓	?	✗	✓	?
`memrise.com`	professional language trainer								
Mindomo	✓			✓	✗	✓	✓	✗	✗
`play.google.com`	mindmap tool								
MINTMagie			✓	✓	✗	✗	✗	✗	✗
`mintmagie.de`	resource listing								
Multidingsda	✓			✓	✗	✓	✓	✗	✗
`apps.apple.com`	language trainer app								
Music Maker	✓			✓	✗	✗	✗	✓	✗
`musiclab.chrome...`	interactive music composition tool								
ÖSZ		✓		✓	✗	✗	✗	✗	✗
`oesz.at`	resource listing								
pcvs.info	✓			✓	✗	✗	✗	✗	✗
`pcvs.info`	resource listing								
PicCollage	✓			✓	✗	✓	✓	✗	✗
`play.google.com`	image cropping/editing tool								
Pixi-Lesestart	✓			?					
`apps.apple.com`	requires apple devices								
Planet Schule		✓	✓	✓	✗	✗	✗	✓	✗
`planet-schule.de`	video listing								
Quizlet	✓	✓		✓	✓	✓	?	?	?
`quizlet.com`	startuppy learning app								
Rechenarena	✓			✓	✗	✗	✗	✓	✓$^?$
`sgs.at`	math competition								
RoboBee	✓			✓	✗	✓	✓	✗	?
`bee.baa.at`	programming trainer								
Schlaukopf	✓	✓		✓	✗	✗	✗	✓	✗
`schlaukopf.at`	multiple choice questions								
schule.at		✓		✓	✗	✗	✗	✗	✗
`schule.at`	resource listing								
scook		✓		✓	✓	✗	✗	✓	✓
`scook.at`	interactive resources for Veritas books								
Scoyo			✓	✓	✓	✗	✗	✓	✗
`www-de.scoyo.com`	paid interactive learning								

Name	[BMBWF]?	[BdStmk]?	[KMK]?	Available?	Sign-up needed?	Native app exists?	Native app needed?	JavaScript needed?	Free-form comms?
Scratch	✓			✓	✓	✓	✗	✓	✗
`scratch.mit.edu`	visual programming								
Seesaw	✓			✓	✓	✓	✗	✓	?
`web.seesaw.me`	startuppy “classroom platform for student engagement”								
Seitenstark			✓	✓	✗	✗	✗	✗	✗
`seitenstark.de`	resource listing								
Serlo		✓		✓	✗	✗	✗	✓	✓?
`de.serlo.org`	free, crowd-sourced education								
Simpleclub		✓		✓	✓	✓	✗	✓	✗
`simpleclub.com`	paid interactive learning								
Skooly	✓			✓	✓	✓	✗	✓	✓
`skooly.at`	online class management								
Sofatutor			✓	✓	✓	✗	✗	✓	✗
`sofatutor.com`	paid interactive learning								
Studio Code	✓			✓	✓	✓	✗	✓	✗
`studio.code.org`	visual programming								
Veritas Verlag		✓		✓	✓	✗	✗	✓	✗
`veritas.at`	paid resource listing								
WordArt	✓			✓	✗	✓	✓	✗	✗
`play.google.com`	word cloud app								
Wunderbare Enkel		✓		✓	✗	✗	✗	✗	✗
`wunderbare-enke...`	resource listing								
Zahlen-Zorro	✓			✓	✓	✗	✗	✓	✗
`zahlenzorro.wes...`	paid interactive learning								
Zebra Schreibtabelle	✓			✓	✗	✓	✓	✗	✗
`play.google.com`	writing trainer app								
Zehn kleine Fingerlein	✓			✓	✗	✗	✗	✗	✗
`10kleinefingerl...`	physical typing trainer								

Appendix B.

Form letters

B.1. Request for Access

Dear `application` team,

we are researchers at Graz University of Technology doing work related to online education during the COVID-19 pandemic.

As part of this effort, we are reviewing widely-used products, with a focus on information security and privacy.
This has already led to us discovering multiple significant flaws in other software, which we were able to assist the authors in correcting.

We would like to also review your `application` platform, as it is in use by schools across Austria.

Would it be possible for us to be granted access – akin to what an enrolled school would have – to your product so we may review its functionality and investigate it for potential flaws?

Best regards,
s/

Sehr geehrtes `Plattform`-Team,

wir sind eine Forschungsgruppe an der TU Graz, die sich derzeit mit Distance Learning während der COVID-19-Pandemie beschäftigt.

Im Rahmen dieser Schwerpunktsetzung nehmen wir derzeit Software, die bei der Umstellung des Schulbetriebs im Rahmen von COVID-19 angewendet wird, sicherheitstechnisch unter die Lupe.
Unser Interesse begründet sich insbesondere darin, dass für Schüler*innen die Verwendung ebendieser Software – und damit Preisgabe ihrer Daten an ebendiese – zumeist in keinster Weise freiwillig geschieht.

Hierbei liegt der Fokus auf Datensicherheit und Privacy, und insbesondere in der Analyse potentieller produktübergreifender systematischer Probleme.
Im Zuge unserer Untersuchungen haben wir bereits mehrere Probleme in Software anderer Firmen aufgedeckt, und konnten im Zuge von Responsible Disclosure die entsprechenden Entwickler*innen bei der Korrektur ebendieser unterstützen.

Nun würden wir auch gerne Ihre Plattform "`Plattform`", die ja in vielen österreichischen Schulen verwendet wird, einbeziehen.
Wäre es für uns möglich, zu Forschungszwecken Zugriff – in etwa entsprechend dem der Direktion einer neu registrierten Schule – zu erhalten?

Mit allerbesten Grüßen,
s/

B.2. Request for Disclosure

Dear `application` team,

We are security researchers at Graz University of Technology in Graz, Austria. As part of a research effort into CoViD-related use of apps in education, we conducted a security review of the `application` platform. During this process, we have discovered several potential security flaws, which we would like to assist you in correcting.

However, we were unable to locate a contact address for Responsible Disclosure through your platform.
Please let us know where we should direct confidential details and technical documentation regarding the issue.

Kind regards,
s/

Sehr geehrtes `Plattform`-Team,

wir sind ein Forscherteam für Informationssicherheit an der Technischen Universität Graz. Im Rahmen eines Forschungsprojekts zur Verwendung von Online-Learning während der CoViD-19-Pandemie haben wir unter anderem eine Sicherheitsanalyse von `Plattform` durchgeführt.
Im Zuge dieser Analyse haben wir mehrere potentielle Schwachstellen entdeckt. Wir würden Ihnen gerne im Rahmen eines industrieüblichen Responsible-Disclosure-Prozesses bei der Behebung ebendieser behilflich sein.

Leider ist es uns nicht gelungen, auf Ihrer Plattform eine Kontaktadresse für Sicherheitsfragen aufzufinden.
Wäre es Ihnen möglich, uns mitzuteilen, an welche Adresse wir sensible Daten und technische Dokumente übersenden sollen?

Mit besten Grüßen,
s/

Bibliography

[BdStmk] Bildungsdirektion Steiermark. *Distance Learning – Links & Tips for Students and Parents With German as a Second Language.* 2020. URL: `https://www.bildung-stmk.gv.at/service/news/aktuelles/2020/corona.html` (visited on 2020-08-30) (cit. on pp. 1, 18, 19, 23, 41, 59, 61–65).

[BMBWF] Bundesministerium für Bildung, Wissenschaft und Forschung. *eEducation: Materialien Primärstufe.* 2020. URL: `https://eeducation.at/index.php?id=665&L=0` (visited on 2020-08-30) (cit. on pp. 1, 18, 19, 23, 41, 47, 59, 61–65).

[Expr] *Express – Node.js web application framework.* URL: `https://expressjs.com/` (visited on 2020-09-01) (cit. on p. 24).

[KMK] Ständige Konferenz der Kultusminister der Länder in der Bundesrepublik Deutschland. *Lernen von zu Hause – Digitale Lernangebote.* 2020. URL: `https://www.kmk.org/themen/bildung-in-der-digitalen-welt/lernen-von-zu-hause-digitale-lernangebote.html` (visited on 2020-08-30) (cit. on pp. 1, 18, 19, 23, 33, 59, 61–65).

[Pcryp] The PHP Group. *PHP: crypt – Manual.* URL: `https://www.php.net/manual/en/function.crypt.php` (visited on 2020-11-17) (cit. on p. 55).

[PHis] The PHP Group. *PHP: History of PHP – Manual.* URL: `https://www.php.net/manual/en/history.php.php` (visited on 2020-11-23) (cit. on p. 7).

[PPre] The PHP Group. *PHP: Preface – Manual.* URL: `https://www.php.net/manual/en/preface.php` (visited on 2020-11-19) (cit. on p. 7).

[red12] /u/1880. *Is this a bug? Shouldn't crypt() be binary safe?* 2012. URL: `https://www.reddit.com/r/PHP/comments/t0qzl/is_this_a_bug_shouldnt_crypt_be_binary_safe/` (visited on 2020-11-17) (cit. on p. 55).

[Apa12] Apache Software Foundation. *Apache Cordova.* 2012. URL: `https://cordova.apache.org/` (visited on 2020-09-15) (cit. on p. 3).

[App05] Apple Developer Connection. *Dynamic HTML and XML: The XMLHttpRequest Object.* 2005. URL: `https://web.archive.org/web/20080509103519/http://developer.apple.com/internet/webcontent/xmlhttpreq.html` (visited on 2008-05-09) (cit. on p. 4).

[ASR17] Rabe Abdalkareem, Emad Shihab, and Juergen Rilling. “On code reuse from StackOverflow: An exploratory study on Android apps”. In: *Information and Software Technology* 88 (2017), pp. 148–158. ISSN: 0950-5849. DOI: `10.1016/j.infsof.2017.04.005` (cit. on p. 54).

[Bak04] Loren Baker. *Mozilla Firefox Internet Browser Market Share Gains to 7.4%.* 2004. URL: https://www.searchenginejournal.com/mozilla-firefox-internet-browser-market-share-gains-to-74/1082/ (visited on 2020-09-24) (cit. on p. 5).

[BDK16] A. Biryukov, D. Dinu, and D. Khovratovich. "Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications". In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P).* 2016, pp. 292–302. DOI: 10.1109/EuroSP.2016.31 (cit. on p. 10).

[Ber91] Tim Berners-Lee. *The Original HTTP as defined in 1991.* 1991. URL: https://www.w3.org/Protocols/HTTP/AsImplemented.html (visited on 2020-09-15) (cit. on p. 3).

[CSS12] Luis Corral, Alberto Sillitti, and Giancarlo Succi. "Mobile Multiplatform Development: An Experiment for Performance Analysis". In: *Procedia Computer Science* 10 (2012). ANT 2012 and MobiWIS 2012, pp. 736–743. ISSN: 1877-0509. DOI: 10.1016/j.procs.2012.06.094 (cit. on p. 3).

[Das+14] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. "The Tangled Web of Password Reuse". In: *21st Annual Network and Distributed System Security Symposium, NDSS 2014.* The Internet Society, 2014. URL: https://www.ndss-symposium.org/ndss2014/tangled-web-password-reuse (cit. on p. 10).

[DH15] Johannes Dahse and Thorsten Holz. "Experience Report: An Empirical Study of PHP Security Mechanism Usage". In: *Proceedings of the 2015 International Symposium on Software Testing and Analysis.* ISSTA 2015. Baltimore, MD, USA: Association for Computing Machinery, 2015, pp. 60–70. ISBN: 9781450336208. DOI: 10.1145/2771783.2771787 (cit. on p. 8).

[Dob96] Hans Dobbertin. "Cryptanalysis of MD5 compress". In: *Rump session of Eurocrypt* 96 (1996). URL: http://web.mit.edu/afs.new/net.mit.edu/dev/user/tytso/papers/md5-collision.ps (cit. on p. 38).

[Dre20] Soatok Dreamseeker. *Cryptography Interface Design is a Security Concern.* 2020. URL: https://soatok.blog/2021/02/24/cryptography-interface-design-is-a-security-concern/ (visited on 2021-03-03) (cit. on p. 60).

[Ecm09] Ecma International. *Ecma International approves major revision of ECMAScript.* 2009. URL: https://www.ecma-international.org/news/PressReleases/PR_Ecma%5C%20approves%5C%20major%5C%20revision%5C%20of%5C%20ECMAScript.htm (visited on 2020-09-24) (cit. on p. 5).

[Ecm15] Ecma International. *ECMAScript 2015 Language Specification.* 2015. URL: https://www.ecma-international.org/ecma-262/6.0/index.html (visited on 2020-09-24) (cit. on p. 5).

[Eic08a] Brendan Eich. *ECMAScript Harmony.* 2008. URL: `https://mail.mozilla.org/pipermail/es-discuss/2008-August/003400.html` (visited on 2020-09-25) (cit. on p. 5).

[Eic08b] Brendan Eich. *Populatory.* 2008. URL: `https://brendaneich.com/2008/04/popularity/` (visited on 2020-09-24) (cit. on p. 4).

[Gar05] Jesse James Garrett. *Ajax: A New Approach to Web Applications.* 2005. URL: `https://web.archive.org/web/20150910072359/http://adaptivepath.org/ideas/ajax-new-approach-web-applications/` (visited on 2015-09-10) (cit. on p. 4).

[GL20] Yael Grauer and Micah Lee. *Zoom Meetings Do Not Support End-to-End Encryption.* 2020. URL: `https://theintercept.com/2020/03/31/zoom-meeting-encryption/` (visited on 2021-03-07) (cit. on p. 1).

[Ham+17] S. T. Hamman, K. M. Hopkinson, R. L. Markham, A. M. Chaplik, and G. E. Metzler. "Teaching Game Theory to Improve Adversarial Thinking in Cybersecurity Students". In: *IEEE Transactions on Education* 60.3 (2017), pp. 205–211. DOI: `10.1109/TE.2016.2636125` (cit. on p. 57).

[Has20] Abeerah Hashim. *Google Drive Vulnerability Allows Spearphishing Attacks.* 2020. URL: `https://latesthackingnews.com/2020/08/25/google-drive-vulnerability-allows-spearphishing-attacks/` (visited on 2021-02-24) (cit. on p. 1).

[Hic+12a] Michael Hicke, Agathe Hicke, Horst Hicke, and Bernd Dietrich. *Schlaukopf.at – Das Projekt.* 2012. URL: `https://www.schlaukopf.at/seiten/projekt.php` (visited on 2021-02-12) (cit. on pp. 1, 19, 41, 59).

[Hic+12b] Michael Hicke, Agathe Hicke, Horst Hicke, and Bernd Dietrich. *Schlaukopf.at – Impressum und Kontakt.* 2012. URL: `https://www.schlaukopf.at/seiten/impressum.php` (visited on 2021-02-12) (cit. on p. 41).

[Hie+15] Michael Hielscher, Werner Hartmann, Manuela Filzer, Nico Steinbach, Christian Wagenknecht, and Franz Rothlauf. *LearningApps – Imprint.* 2015. URL: `https://learningapps.org/impressum.php` (visited on 2021-02-19) (cit. on p. 47).

[Hie12] Michael Hielscher. *What is LearningApps.org.* 2012. URL: `https://learningapps.org/about.php` (visited on 2021-02-19) (cit. on pp. 2, 19, 47, 59).

[Hof19] Jay Hoffmann. *What Does AJAX Even Stand For?* 2019. URL: `https://thehistoryoftheweb.com/what-does-ajax-even-stand-for/` (visited on 2020-09-16) (cit. on p. 4).

[Hop06] Alex Hopmann. *The story of XMLHTTP.* 2006. URL: `https://web.archive.org/web/20160630074121/http://www.alexhopmann.com/xmlhttp.htm` (visited on 2016-06-30) (cit. on p. 4).

[Hum+18a] Rita Humer, Gabriele Saulich, Egon Humer, and Michael Eberl. *Antonwelt – Für Eltern*. 2018. URL: https://www.antonwelt.schule/eltern/ (visited on 2021-02-04) (cit. on p. 33).

[Hum+18b] Rita Humer, Gabriele Saulich, Egon Humer, and Michael Eberl. *Antonwelt – Impressum*. 2018. URL: https://www.antonwelt.schule/impressum/ (visited on 2021-02-04) (cit. on p. 33).

[Hum+18c] Rita Humer, Gabriele Saulich, Egon Humer, and Michael Eberl. *Antonwelt – Über Uns*. 2018. URL: https://www.antonwelt.schule/ueber-uns/ (visited on 2021-02-04) (cit. on pp. 1, 19, 33, 59).

[Hum+18d] Rita Humer, Gabriele Saulich, Egon Humer, and Michael Eberl. *Antonwelt – Unterstützer*. 2018. URL: https://www.antonwelt.schule/unterstuetzer/ (visited on 2021-02-04) (cit. on p. 33).

[Jan+20] Jiyong Jang, Dhilung Kirat, Ian Molloy, and J.R. Rao. *IBM Works With Cisco to Exorcise Ghosts From Webex Meetings*. 2020. URL: https://securityintelligence.com/posts/ibm-works-with-cisco-exorcise-ghosts-webex-meetings/ (visited on 2021-02-24) (cit. on p. 1).

[JVS20] Nicole Johnson, George Veletsianos, and Jeff Seaman. "US Faculty and Administrators' Experiences and Approaches in the Early Weeks of the COVID-19 Pandemic." In: *Online Learning* 24.2 (2020), pp. 6–21 (cit. on p. 1).

[Kat19] Frank Katz. "Adversarial thinking: teaching students to think like a hacker". In: *KSU Proceedings on Cybersecurity Education,Research and Practice*. 2019. URL: https://digitalcommons.kennesaw.edu/ccerp/2019/education/1/ (cit. on p. 57).

[Kri01] David M. Kristol. "HTTP Cookies: Standards, Privacy, and Politics". In: *ACM Trans. Internet Technol.* 1.2 (2001-11), pp. 151–198. ISSN: 1533-5399. DOI: 10.1145/502152.502153 (cit. on p. 8).

[Lak07] Pratap Lakshman. *JScript Deviations from ES3*. 2007. URL: https://regmedia.co.uk/2007/10/31/jscriptdeviationsfromes3.pdf (visited on 2020-09-24) (cit. on p. 5).

[McC93] Rob McCool. *Server Scripts*. 1993. URL: http://1997.webhistory.org/www.lists/www-talk.1993q4/0485.html (visited on 2020-09-15) (cit. on p. 3).

[McC94] Rob McCool. *Common Gateway Interface*. 1994. URL: https://web.archive.org/web/20100213224155/http://hoohoo.ncsa.illinois.edu/cgi/intro.html (visited on 2010-02-13) (cit. on p. 3).

[Mic] Microsoft. *IXMLHTTPRequest*. URL: https://web.archive.org/web/20160526164820/https://msdn.microsoft.com/en-us/library/ms759148(VS.85).aspx (visited on 2016-05-26) (cit. on p. 4).

[Mic97] Microsoft. *Microsoft Delivers ECMA-Complaint JScript 3.0 In Key Microsoft Products.* 1997. URL: `https://web.archive.org/web/20090112221530/http://www.microsoft.com/presspass/press/1997/Jun97/jecmapr.mspx` (visited on 2009-01-12) (cit. on p. 5).

[Moza] Mozilla MDN Web Docs. *Fetch API.* URL: `https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API` (visited on 2021-01-31) (cit. on p. 5).

[Mozb] Mozilla MDN Web Docs. *IndexedDB API.* URL: `https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API` (visited on 2021-01-31) (cit. on p. 5).

[Mozc] Mozilla MDN Web Docs. *Service Worker API.* URL: `https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API` (visited on 2020-09-25) (cit. on p. 5).

[Mozd] Mozilla MDN Web Docs. *Web APIs.* URL: `https://developer.mozilla.org/en-US/docs/Web/API` (visited on 2021-01-31) (cit. on pp. 3, 5).

[Moze] Mozilla MDN Web Docs. *WebStorage API.* URL: `https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API` (visited on 2021-01-31) (cit. on p. 5).

[Mozf] Mozilla MDN Web Docs. *XMLHTTPRequest.* URL: `https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest` (visited on 2021-01-31) (cit. on p. 5).

[Net20] NetMarketShare. *Browser Market Share.* 2020. URL: `https://netmarketshare.com/` (visited on 2020-09-15) (cit. on p. 3).

[Ope20] OpenJS Foundation. *Electron: Build cross-platform desktop apps with JavaScript, HTML and CSS.* 2020. URL: `https://www.electronjs.org/` (visited on 2020-09-15) (cit. on p. 3).

[Ray03] Eric S. Raymond. *The Jargon File: magic cookie.* 2003. URL: `https://web.archive.org/web/20030609151443/http://catb.org/jargon/html/M/magic-cookie.html` (visited on 2003-06-09) (cit. on p. 8).

[Sch01] John Schwartz. *Giving Web a Memory Cost Its Users Privacy.* 2001. URL: `https://www.nytimes.com/2001/09/04/business/giving-web-a-memory-cost-its-users-privacy.html` (visited on 2020-12-29) (cit. on p. 8).

[Sch13] F. B. Schneider. “Cybersecurity Education in Universities”. In: *IEEE Security Privacy* 11.4 (2013), pp. 3–4. DOI: `10.1109/MSP.2013.84` (cit. on p. 57).

[sola] solocode GmbH. *ANTON – FAQ.* URL: `https://anton.app/en_us/faq/` (visited on 2021-01-25) (cit. on pp. 1, 19, 24, 59).

[solb] solocode GmbH. *ANTON – Imprint.* URL: `https://anton.app/en_us/imprint/` (visited on 2020-09-01) (cit. on p. 23).

[UNE20] UNESCO. *School closures caused by Coronavirus (Covid-19).* 2020. URL: `https://en.unesco.org/covid19/educationresponse` (visited on 2021-03-08) (cit. on p. 1).

[VB20] Gerrit De Vynck and Mark Bergen. *Google Classroom Users Doubled as Quarantines Spread.* 2020. URL: `https://www.bloomberg.com/news/articles/2020-04-09/google-widens-lead-in-education-market-as-students-rush-online` (visited on 2021-03-08) (cit. on p. 1).

[VDP20] Tom Van Goethem, Nurullah Demir, and Barry Pollard. *Security – The 2020 Web Almanac.* 2020. URL: `https://almanac.httparchive.org/en/2020/security` (visited on 2021-01-15) (cit. on p. 15).

[Veg20] Oskars Vegeris. *"Important, Spoofing" – zero-click, wormable, cross-platform remote code execution in Microsoft Teams.* 2020. URL: `https://github.com/oskarsve/ms-teams-rce/blob/main/README.md` (visited on 2021-02-24) (cit. on p. 1).

[W3T] W3Techs Web Technology Surveys. *Usage statistics of JavaScript as a client-side programming language on websites.* URL: `https://w3techs.com/technologies/details/cp-javascript/` (visited on 2020-09-25) (cit. on p. 5).

[Wag21] Paul Wagenseil. *Zoom security issues: Here's everything that's gone wrong (so far).* 2021. URL: `https://www.tomsguide.com/news/zoom-security-privacy-woes` (visited on 2021-02-24) (cit. on p. 1).

[Wan+04] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD." In: *IACR Cryptol. ePrint Arch.* 2004 (2004), p. 199 (cit. on p. 38).

[Wod20] Shoshana Wodinsky. *The Butt Pajamas Will Follow You Forever.* 2020. URL: `https://gizmodo.com/the-butt-pajamas-will-follow-you-forever-1845929307` (visited on 2020-12-29) (cit. on p. 9).

[XLC11] J. Xie, H. R. Lipford, and B. Chu. "Why do programmers make security errors?" In: *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC).* 2011, pp. 161–164. DOI: `10.1109/VLHCC.2011.6070393` (cit. on pp. 53, 54).