Alexander WOLFBAUER, BSc

Security Analysis of Student Developed Learning Applications

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Priv.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner

Institute of Interactive Systems and Data Science
Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Frank Kappe

Co-Supervisor

Dipl.-Ing. Dr.techn. Josef Wachtler BSc

Graz, September 2023

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

Date	Signature

Acknowledgments

I would like to extend my heartfelt gratitude to my supervisors, Priv.-Doz. Dipl.-Ing. Dr. techn. Martin Ebner, for enabling me to undertake this master's thesis. I am also deeply appreciative of the guidance and support provided by Dipl.-Ing. Dr.techn. Josef Wachtler BSc, who played a crucial role, particularly in the practical aspects of this work.

I want to express my sincerest appreciation to my mother, whose unwavering support has been instrumental throughout my academic journey. Her encouragement and belief in me have been invaluable, and I am immensely grateful for her contribution to my completion of this master's thesis.

Finally, I extend my thanks to all those who have contributed to my growth and learning during this endeavor.

Abstract

This thesis provides a comprehensive security analysis of the "TU Graz Learning Lab" project, a suite of learning applications developed by students of "Graz University of Technology" for primary pupils and classes. In light of the fact that the project processes children's data, it is governed by the stringent regulations of the "General Data Protection Regulation". This study applies a static code analysis using "PhpMetrics" and a dynamic vulnerability scan using "Open Web Application Security Project Zed Attack Proxy", which exposed high code complexity, outdated dependencies, and numerous security vulnerabilities. These issues underline a significant deficiency in the implemented software development processes, leading to critical maintainability issues and a lack of coding standards.

Furthermore, an application of the "Open Web Application Security Project Application Security Verification Standard" highlighted a failure in over half of the "Level 1" requirements, indicating an absence of secure software development cycles and security requirements. An attempted implementation of "OAuth 2.0", conducted as a case study, further elucidated the impact of these issues on development. The endeavor was halted due to problems stemming from dependency management, data storage, and complexity in managing legacy issues, reinforcing the necessity for a secure software development cycle.

This research emphasizes the indispensability of such a cycle, particularly in student-led projects marked by high personnel turnover, affirming the integral role of secure software development practices in ensuring project maintainability and data security.

Abstrakt

Diese Arbeit bietet eine umfassende Sicherheitsanalyse des Projekts "TU Graz Learning Lab", einer Sammlung von Lernanwendungen, die von Studierenden der "Technischen Universität Graz" für Grundschulkinder und -klassen entwickelt wurden. Da das Projekt Daten von Kindern verarbeitet, unterliegt es den strengen Vorschriften der "Datenschutz-Grundverordnung".

In dieser Arbeit wird eine statische Codeanalyse mit "PhpMetrics" und ein dynamischer Sicherheitsscan mit "Open Web Application Security Project Zed Attack Proxy" durchgeführt, die eine hohe Codekomplexität, veraltete Abhängigkeiten und zahlreiche Sicherheitslücken aufdeckten. Diese Probleme zeigen ein erhebliches Defizit in den implementierten Softwareentwicklungsprozessen, was zu kritischen Wartungsproblemen und einem Fehlen von Programmierstandards führt.

Darüber hinaus zeigte die Anwendung des "Open Web Application Security Project Application Security Verification Standard" ein Versäumnis in über der Hälfte der "Level 1" Anforderungen, was auf das Fehlen sicherer Softwareentwicklungszyklen und Sicherheitsanforderungen hinweist. Eine versuchte Implementierung von "OAuth 2.0", die als Fallstudie durchgeführt wurde, verdeutlichte weiterhin die Auswirkungen dieser Probleme auf die Entwicklung. Das Unterfangen wurde aufgrund von Problemen, die sich aus "Depndency Managment", sicherer Datenverarbeitung und der Komplexität bei der Bewältigung von Altlasten im Code ergaben, gestoppt, was die Notwendigkeit eines sicheren Softwareentwicklungszyklus unterstreicht.

Diese Forschung zeigt die Unverzichtbarkeit eines sicheren Softwareentwicklungszyklus, insbesondere bei von Studierenden geleiteten Projekten mit einer hohen Personalrotation, und bestätigt die zentrale Rolle sicherer Softwareentwicklungspraktiken bei der Sicherstellung der Projektwartbarkeit und Datensicherheit.

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial intelligence
API Application Programming Interface
CI/CD Continuous Integration/Continuous Delivery
CSP Content Security Policy
CSRF Cross-Site Request Forgery
GDPR General Data Protection Regulation
GUI Graphical User Interface
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
IaC Infrastructure as Code
IETF Internet Engineering Task Force
JS Java Script
JWT JSON Web Token
LDAP Lightweight Directory Access Protocol
LTI Learning Tools Interoperability
MIME Multipurpose Internet Mail Extensions
ORM Object-Relational Mapping
OWASP Open Web Application Security Project
OWASP ASVS Open Web Application Security Project Application Security Verification Standard
OWASP ZAP Open Web Application Security Project Zed Attack Proxy
REST Representational State Transfer
RPC Remote Procedure Call
SQL Structured Query Language
TLS Transport Layer Security
TU Graz Graz University of Technology
URL Uniform Resource Locator
XML Extensible Markup Language
XSLT Extensible Stylesheet Language Transformations

XSS	Cross Site Scripting
ZID	Zentraler Informatikdienst

1 Introduction

The "TU Graz Learning Lab" is an educational platform developed by Graz University of Technology (TU Graz) students aimed at primary school pupils, providing various web-based applications to aid in mathematics and literacy learning, while also enabling progress tracking. The platform not only facilitates educational progress but also collects data for research, ensuring data is aggregated and anonymized for the enhancement of teaching in Austrian institutions. The platform's infrastructure relies on the "Laminas" framework, with individual applications developed as part of academic theses, all emphasizing user management, data security, and role-based access controls.

The "TU Graz Learning Lab" handles sensitive data from children, primarily under 14, which is stringently regulated under European GDPR law, especially when it concerns health or mental progress. Given the platform's lengthy development phase and transitions like moving from the "Zend Framework" to "Laminas," there are concerns about system clarity and potential vulnerabilities. The current "SOAP" Application Programming Interface (API) protocol for logins is considered outdated and not efficient, as it requires separate logins for each application, suggesting a need for more modern solutions.

This master's thesis seeks to evaluate the security of web learning applications developed by TU Graz students, identify the origins of potential vulnerabilities, and suggest future preventive measures. Given the project's extended development timeline and the diverse security proficiency of the student developers, it's hypothesized that there are likely security gaps present. The predominant vulnerabilities are believed to stem from a lack of security knowledge and inadequate secure development practices, with a recommendation to transition from the "SOAP" interface to the "OAuth 2.0" protocol for improvements.

The methodology for this research involves practical and experimental analysis to deeply understand the project and its context. A security analysis will be conducted, followed by a practical implementation of "OAuth 2.0" to identify design flaws and vulnerabilities. Based on this comprehensive assessment, recommendations will be formulated to enhance the security of student-developed projects.

1.1 TU Graz Learning Lab

"TU Graz Learning Lab" is a collection of different learning web applications, which were developed by TU Graz students (Technische Universität Graz, 2023f). The target group consists of primary schools. The idea is to provide an educational platform for schools and school classes, which not only allows pupils to make progress in mathematics and literacy, but also allows them and their teachers to track their development in basic school skills.

In addition, the information collected from the applications is used for research purposes. These research studies do not aim to present personal data. Once the results are obtained, the data is aggregated and anonymized. The scientific findings aim to significantly contribute to the improvement of teaching in Austrian educational institutions and the dissemination of knowledge among children (Technische Universität Graz, 2023i).

1.1.1 Applications

As stated before, the "TU Graz Learning Lab" consists of multiple different learning applications. The main site is responsible for the user management and for combining information from the individual applications for teachers and administrators, which are separate roles in the ranking system, Section 1.1.2 will explain. It also links to the separate web applications, which are categorized as "Math Apps"[1] and "Speach [sic!] Apps"[2]. At the

[1] https://schule.learninglab.tugraz.at/math visited on 05/30/2023
[2] https://schule.learninglab.tugraz.at/speach visited on 05/30/2023

time of creating this master's thesis, "TU Graz Learning Lab" provided the following applications:

"Math Apps":

1. "1×1 Trainer" (Technische Universität Graz, 2023a) : A ten times table multiplication trainer, including account-based progress tracking and a time-limited speed game.
2. "Mathe-Multi-Trainer" (Technische Universität Graz, 2023g): A multi-digit multiplication trainer with adaptive difficulty and account-based statistics to track the progress.
3. "Divisiontrainer" (Technische Universität Graz, 2023c): An educational tool for learning and practicing long divisions with adaptive difficulty and personalized feedback.
4. "Plus-Minus-Trainer" (Technische Universität Graz, 2023h): A trainer for pupils to practice addition and subtraction with self-adjustable difficulty parameters.
5. "Junior Plus-Minus-Trainer" (Technische Universität Graz, 2023e): Also a trainer for learning addition and subtraction, but in a simpler and more playful form than the Plus-Minus-Trainer. Additionally, it is possible to compare the progress with other pupils in the "Highscore" section.

"Speach [sic!] Apps":

1. "Buchstabenpost" (Technische Universität Graz, 2023b): An account-based learning application that tracks progress and allows teachers to create fill-in-the-blank tasks where a word is missing from a given sentence or question and the student must choose the correct word from a word list.
2. "IDeRBlog Exercises" (Technische Universität Graz, 2023d): A collection of different applications for learning spelling and grammar.
3. "Lesetrainer" (Technische Universität Graz, 2023b): A collection of three educational tools for assessing reading literacy, including speed reading tests, text comprehension tests and reading comprehension tests, the results of which are stored account-based for teachers.

The project also includes several other applications that are not addressed in this paper. This is either because these applications are no longer available on the live version, or they are not designed as web applications.

1.1.2 System Architecture

All previous applications were not developed at the same time, due to the fact that they are the result of scientific work. The first application was the "Mathe-Multi-Trainer", the theoretical work of which was published in 2012 (Steyrer, 2012). Therefore, all other projects were based on it and evolved over time. This led to an emerging project design (Proenca, Moura, and Hoek, 2010), which has been manually adapted several times. To understand this work, it is necessary to have an overview of the structure of the project as it is now.

Overall System Design

All applications are based on the "Laminas"[3] framework, which was formerly known as the "Zend Framework"[4]. Due to the fact that the "Zend Framework" migrated to "Laminas" in 2020, most of the applications were developed in the former framework and then migrated to "Laminas".

"Laminas" is an open source "PHP" framework that provides a collection of components and tools for building web applications. The main components of "Laminas" are as follows:

1. "MVC" provides a structured approach of developing web applications through the Model-View-Controller architectural pattern.
2. "Mezzio" is a middleware microframework that handles routing, error handling and dependency injection containers, through the "PSR-7"[5] and "PSR-15"[6] specifications.
3. "API Tools" is a framework for creating and documenting API.

[3]https://getlaminas.org/ visited on 05/30/2023
[4]https://framework.zend.com visited on 05/30/2023
[5]https://www.php-fig.org/psr/psr-7/ visited on 05/30/2023
[6]https://www.php-fig.org/psr/psr-15/ visited on 05/30/2023

All three components are used to varying degrees in the applications and form the basis of the project.

As noted in Section 1.1.1, the main page is not only responsible for linking to the other applications, but also for managing the general user data. Therefore, it is internally called "UserManager". Furthermore, the "UserManager" also provides an API and distributes all necessary information to the other applications. This is achieved over the "SOAP 1.2"[7] protocol.

Development and Documentation

Each of the web applications is the result of a Bachelor's or Master's thesis, as can be seen in Table 1.1. The applications themselves were primarily developed by the same students who wrote the corresponding academic papers. Revisions and maintenance were carried out either by other students as part of their academic work, or by academic staff.

Operationally, the development runs over a configured "Vagrant Box" currently running "Debian 10" and "PHP 7.3". "Vagrant"[8] is a utility for managing the lifecycle of virtual machines, isolating dependencies and providing a consistent development environment. The codebase is hosted on a private "GitLab" repository, the instance of which runs on the servers of TU Graz.

Students developed their own applications and adapted them to work with the existing framework. No modern software development process is used. Once they have completed their applications, their project will be merged under the supervision of research staff.

Documentation is provided within the "Git" repository, mostly through the bachelor's and master's thesis. Additionally, most of the applications also have an operating documentation and a user manual, mainly for the teaching personnel. The following table links the applications and the theoretical work of the students:

[7] `https://www.w3.org/TR/soap12/` visited on 05/30/2023
[8] `https://www.vagrantup.com/` visited on 05/30/2023

App	Author
1x1 Trainer	Kraja, 2016
Buchstabenpost	Schellander, 2022 (Revision)
Divisiontrainer	Geier, 2015
IDeRBlog Exercises	Burazer, 2019
Junior Plus-Minus-Trainer	Zöhrer, 2022 (Revision)
Lesetrainer	Koch, 2020 (Revision)
Mathe-Multi-Trainer	Steyrer, 2012
Plus-Minus-Trainer	Neuhold, 2013
UserManager	Wachtler, 2017

Table 1.1: Applications and Authors

Security and Data Management

Before looking at the authentication process, this thesis will describe the authorization handling. The following three roles are used to differentiate users:

1. "Student"
2. "Teacher"
3. "Admin"

"Students" have the ability to create their own accounts by registering directly from the main page. Additionally, "Teachers" or "Administrators" can create whole classes and distribute the credentials to their pupils or to the teacher. Authorization for all applications is handled by a role-based access control provided by "laminas-permissions-acl"[9].

As stated in the previous sections, the "UserManager" is responsible for the user data. It allows users to register, login and change their user information. All this information is stored in the "UserManager's" "MySQL" database. However, it does not directly handle the login for all other applications. Due to the system design, all other applications have their own "MySQL" database to store the necessary information for the application, such as tasks for the "Student". Furthermore, the applications store user specific

[9]https://docs.laminas.dev/laminas-permissions-acl/ visited on 05/30/2023

data like the username again to enable the tracking of account based information within their applications. All transfer of this data is handled over the "SOAP" API. So each application has its own login form and requests the authentication over the API via the username and the hashed password. At the first login of a new user, the application also requests the information needed to initialize the user from the "UserManager" API. Most applications track the session in an "HTTP-only" cookie and store it in a stateful way, some applications use the same session ID, others create a new one after a successful login.

1.2 Problem Statement

The "TU Graz Learning Lab" provides services that process data from pupils, most of whom are under the age of 14. The processing of children's data has a particularly sensitive position in Austrian law. Additionally, children's learning data can provide information about their mental progress and even about their mental health. Health data, including data which reveal information about one's (mental) health status, is especially protected by the European law. (Article 4(15) General Data Protection Regulation (GDPR))

As a result of the relatively long development period in the context of web applications, the migration from the "Zend Framework" to "Laminas" and the emerging system design, the project is in a state of obscurity. Therefore, the evaluation of the applications from a security perspective is critical to ensure secure data processing.

Moreover, the "SOAP" API protocol does not provide a good solution for the login process. It is necessary to log in separately for each application, although the login credentials are the same. Also, "SOAP1.2" is no longer the "Simple Object Access Protocol", like it was when it was established in 2004. Modern solutions may be better suited for the specific use case of the "TU Graz Leaning Lab" (Belqasmi et al., 2012).

1.3 Research Questions

This master's thesis will aim to answer the following research questions:

1. How secure are web learning applications, which were developed by TU Graz students within their bachelor or master thesis, from an information technical point of view?
2. Which security flaws happened, how can they be fixed, and why have students implemented their learning applications insecurely?
3. Which mitigation actions could be taken to avoid security vulnerabilities in student developed learning applications within their studies in the future?

1.4 Hypotheses

Considering the context, the evolution and the current state of the "TU Graz Learning Lab", following hypothesis can be formulated:

1. Due to long-term development, mostly conducted from students with different levels of security awareness and the fact that no active security audit was ever conducted, it is likely to find security vulnerabilities in the applications.
2. These vulnerabilities appear primarily due to insufficient security knowledge among student developers and the lack of a secure software development cycle, rather than intentional negligence.
3. The current architectural setup, with the "SOAP" based interface for the applications, can be improved through the introduction of the "OAuth 2.0" protocol.

1.5 Methodology

The methodology of this scientific work will certainly be dominated by practical and experimental analysis. Firstly, the project needs to be understood in its context, in order to be able to recognize factual connections later on. Secondly, a security analysis will be conducted to gain a deep insight into the project and its security flaws and vulnerabilities.

Then design flaws and security vulnerabilities will be analyzed using the practical implementation of "OAuth 2.0" in the project. This case study and the combination with the actual security analysis should then give a clear picture of the security state of the projects and their background in the context of the student-developed projects. Based on these findings, a recommendation can be made for student-developed projects to increase security.

2 Related Work

This chapter provides an overview of the work within which this thesis is situated. Whereas security analysis in web applications, student developed applications and learning applications are separate well researched topics, there is hardly any literature on the effects of these topics in combination. Therefore, this thesis is embedded in the following subjects.

2.1 Security Standards in Web and Learning Applications

Web applications are under a constant threat from a variety of security risks, due to their continuous exposure to the Internet and vast amount of data being used online (Yadav et al., 2018). As a result, security standards play an indispensable role in modern web development. These guidelines provide methods for developing and maintaining secure applications (Scarfone, Benigni, and Grance, 2009).

Further, the implementation of security protocols is a critical aspect and is often addressed by security standards at all levels. Protocols provide a standardized way to manage authorized access to resources and create interfaces to enable secure data exchange. When implemented correctly, they can not only help prevent security vulnerabilities, but also improve the functionality and modularity of applications (Gerodimos et al., 2023).

Additionally, learning applications often deal with sensitive data, in the context of minors they have an increased responsibility. This responsibility must be addressed by implementing security standards and secure protocols. Ensuring secure data exchange, authorization and privacy in combination with standards can significantly increase the security of learning applications and thereby promote trust and safety (Aldabbagh et al., 2021).

2.1.1 OWASP Foundation

Open Web Application Security Project (OWASP) is a non-profit foundation, that works to improve the security of software. Its primary goal is to enable organizations to conceive, develop, acquire, operate, and maintain applications securely. All OWASP Foundation projects are open source and community driven. With over 250 local chapters worldwide and tens of thousands of members, it provides industry-leading education and training conferences (OWASP Foundation, 2023).

One of the most widely used OWASP tools is the "OWASP Top Ten", a regularly updated standard awareness document. It describes the ten most critical security risks to web applications. It is globally recognized by developers as the first step towards more secure coding and starts the process of ensuring that web applications minimize their risks, when implemented (OWASP Foundation, 2021b).

Further, OWASP provides the Open Web Application Security Project Application Security Verification Standard (OWASP ASVS), which is a framework for security-related activities that are involved during the software development cycle and is used to establish a level of confidence in the security of web applications. It has several levels, and even the lowest level includes not only the requirements of the OWASP Top Ten, but also includes additional security checks. The requirements have been developed as a metric to determine the level of trust that can be placed in web applications, as a guide for security control developers, and as a procurement to provide a contractual basis (OWASP Foundation, 2021a).

2.1.2 OAuth 2.0

"OAuth 2.0" is an industry-standard protocol for authorization, created and maintained by the Internet Engineering Task Force (IETF). It is used as an open source standard for access delegation or allowing applications to access their information from third parties without providing their credentials. The simplicity while providing standardized authorization flows makes it to a widely used and adapted protocol (Internet Engineering Task Force (IETF), 2023).

2.1.3 1EdTech

"1Edtech", formerly known as the "IMS Global Learning Consortium", is the world's leading member-based non-profit community partnership of educational technical organizations. Therefore, it provides open standards and innovation for learning environments. The "1EdTech" Learning Tools Interoperability (LTI) 1.3 standard is used to streamline and standardize the exchange of data between learning tools, digital resources, and student information systems (1EdTech, 2021).

The LTI "Core Specification", specifies a framework for interoperability between learning applications. This is based on "OAuth 2.0" for secure API calls, providing a standardized way to authorize requests and ensuring secure data transfer between integrated systems. This enhances the security and functionality of a project that adapts to this standard (IMS Global Learning Consortium, 2019).

2.2 Security Analysis of Software Applications

Software systems have moved from an artistic phase based on highly skilled craftsmen to an industrial phase where quality is controlled by introducing structural workflows. Security analysis is a process of evaluating the security features and workflows of an application against predefined security standards. These features include testing, evaluating, and mitigating risks associated with vulnerabilities (Ricca and Tonella, 2001).

Further, the utilization of open source frameworks in web applications provides pre-written code for common functionality. This reduces project time and complexity. However, choosing a well-maintained and secure framework is critical, as vulnerabilities in the framework can affect the security of the entire project (Walden, Stuckman, and Scandariato, 2014).

2.2.1 OWASP Web Security Testing

Open Web Application Security Project Zed Attack Proxy (OWASP ZAP) is a security tool, which is actively managed by the OWASP Foundation and therefore maintained by hundreds of international volunteers. It not only provides security testing capabilities, but can also be used to automate security checks in the development cycle. Although OWASP ZAP is superior to other tools, it is still not perfect and manual evaluation is required (Makino and Klyuev, 2015).

2.2.2 Open Source Frameworks

In the realm of open source frameworks, "Laminas", previously known as the "Zend Framework", is a renowned open source PHP framework for web applications. Since this project is community-driven, its development process is transparent, which ensures prompt resolution of security issues. The Linux Foundation maintains it (Linux Foundation, 2023).

3 Security Analysis

Security analysis in the context of web applications is a process to evaluate and improve the security of their services. As written in Section 1.2, the "TU Graz Learning Lab" processes data of pupils, mostly under 14 years old, which is particularly worthy of protection. Furthermore, security breaches can cause irreparable damage to the reputation of the organization, in this case, TU Graz (M., Haddad, and A., 2009).

Additionally, security is not a final product and should not be dealt as such. It is a process that must be present throughout the entire project and software development life cycle (Teodoro and Serrao, 2011).

3.1 Security Analysis Tools and Techniques

To get a general overview of the project, at first a code analysis with "PhpMetrics v2.8.2" is conducted. "PhpMetrics"[1] is an open source static code analysis tool for "PHP", providing software quality metrics about the applications. The tool shows insights into the maintainability, complexity of the project and the degree of coupling between the objects. Additionally, it provides information about the dependencies. Whereas all these parameters are not directly related to a security analysis, they provide the foundation for secure software development and therefore have a huge indirect impact on software security (Mcgraw, 2004).

With this information in mind, OWASP ZAP version 2.12.0[2] is used to start the actual security analysis. OWASP ZAP is an open source web application

[1] `https://PhpMetrics.org/` visited on 07/04/2023
[2] `https://www.zaproxy.org/` visited on 07/04/2023

security scanner, which provides several tools for static and dynamic security analysis. Further, it allows testing the application in different contexts.

At this stage, each web application is tested independently, aligning with the system design of the "TU Graz Learning Lab" as described in Section 1.1.1, where each application operates autonomously. Therefore, the results have to be combined in order to be able to make statements about the whole project. This is done using the OWASP ASVS version 4.0.3.

OWASP ASVS[3] is an open source framework that helps to build secure web applications. It is basically a collection of security requirements and tests that can be used to define what level of security an application has and what level it needs. OWASP (2021) defines the following three levels of security:

Level 1 ("First steps, automated, or whole of portfolio view") is meant as a basic level that every application should strive for.

Level 2 ("Most applications") is meant for applications that use sensitive data.

Level 3 ("High value, high assurance, or high safety") is the highest level of security and reserved for applications in the military, critical infrastructure or safety.

This thesis will concentrate on the "Level 1" requirements. "Level 1" is the foundation for all other higher levels and although the "TU Graz Learning Lab" deals with sensitive data, it makes sense to concentrate on the lowest requirements and if they are fulfilled the next higher level can be analyzed.

3.2 Application Scanning and Testing

This section will describe methodology and the main results of the analysis performed. This is achieved by first defining the scope of the security analysis and then dividing the analysis into the static part, the dynamic part and the manual part. As described in the previous section, the main tools used for the analysis were "PhpMetrics" and OWASP ZAP.

[3]https://owasp.org/www-project-application-security-verification-standard/ visited on 07/04/2023

3.2.1 Scope

When testing the application, it is set up locally on a "Vagrant" box. Therefore, the scope of the application does not include the actual infrastructure of the live version, with one exception, which will be discussed in the following paragraph. The testing approach can be seen as a gray box testing analysis, which is well suited for web applications. Gray box testing for web applications merges methodologies from white box testing, in which a tester has access to all resources of an application, and black box testing, where a tester accesses the system no differently than a regular user. This approach allows testing the project effectively by combining insights from the project's application's structure and codebase with a user's perspective (Acharya, Pandya, and Department, 2012).

The live version of "TU Graz Learning Lab"[4] is tested for Transport Layer Security (TLS) configurations via the tool "SSL Labs v2.1.10"[5] from "Qualys", as this information is required to apply "Level 1" OWASP ASVS. Therefore, the scope of the analysis stretches from the configuration of the provisioned infrastructure, to the TLS configuration of the live version to the actual codebase.

3.2.2 Static Analysis

As described in the previous section, "PhpMetrics" is used for the first part of the static analysis. In addition, OWASP ZAP also provides static analysis tools. For simplicity, and due to the fact that OWASP ZAP uses static tools during dynamic scan, the entire analysis of OWASP ZAP is included in the dynamic part of the security analysis.

General Code Quality and Testability

Each application is individually tested with "PhpMetrics", after which all the information is consolidated. The results offer a comprehensive overview

[4]`https://schule.learninglab.tugraz.at/` visited on 07/04/2023
[5]`https://geocerts.ssllabs.com/` visited on 07/04/2023

of the project, enabling an evaluation of its overall state.

Table 3.1 shows the lines of code and the number of classes each application has. While the individual applications have relatively small codebases, all applications together have a moderate size with 796 classes and 70863 lines of code in the context of web applications. This metrics alone does not have any significance, but the information should be kept in mind for the next points dealing with maintainability.

Application	**Classes**	**Lines of Code**
Buchstabenpost	35	2679
Divisionstrainer	72	4473
EinmalEins	79	10855
IderBlogExercises	188	12825
JuniorPlusMinus	82	4761
Lesetrainer	65	7089
Multiplikationstrainer	63	8346
PlusMinus	111	10447
UserManager	101	9388
TOTAL	**796**	**70863**

Table 3.1: PhpMetrics Number of Classes and Lines of Code

Further, "PhpMetrics" also provides the cyclomatic complexity of the project. The cyclomatic complexity can be used to evaluate the testability and the maintainability of the project. This is done by estimating the potential number of independent paths through the code (Gill and Kemerer, 1991). This number can then be used to evaluate maintainability and testability with the following threshold value (Martin, 2009):

1-10: Low Complexity, is code which is easy to understand, it has a typically linear control flow.

11-20: Moderate complexity, is code that has some branching, but is still manageable.

21-50: High complexity, indicates code that has a more complex structure. It may require additional attention through good understanding and testing.

>50: Untestable complexity, indicates code that is very challenging to understand and nearly impossible to test.

Especially automated tests play an extremely important role in developing web applications (Zandstra, 2016). Some applications like the "UserManager" and the "Buchstabenpost" contain "PHP" unit tests, but these are basically scattered. Additionally, they are not automated and the coverage is so low, that they play no role in the actual development and can therefore be neglected. The only exception is the "SOAP" API, which would have sufficient tests for a secure software development in an automated environment. The consequences of this are visible in the actual security analysis later.

Table 3.2 shows the number of classes categorized by the previously described cyclomatic complexity thresholds of each individual application. As described in Section 1.1.1, the functionality of the applications is not so complex that such a high complexity is justified, mostly for the most important classes of the project. This is one reason why the project lacks automated testing, which in turn makes it challenging to implement tests retrospectively (Belonozkin and Rybanov, 2016).

Application	#Moderate	#High	#Untestable
Buchstabenpost	4	5	0
Divisionstrainer	8	6	0
EinmalEins	9	6	6
IderBlogExercises	25	11	0
JuniorPlusMinus	6	1	1
Lesetrainer	7	7	1
Multiplikationstrainer	0	2	4
PlusMinus	20	10	0
UserManager	16	8	2
TOTAL	**95**	**56**	**14**

Table 3.2: PhpMetrics Cyclomatic Complexity

Furthermore, "PhpMetrics" tests the applications for violations. Violations are categorized as follows:

Information are non-critical findings, that include unused variables, code deduplication, inefficient constructs and general improvement of the code.

Warnings are potential issues, that are triggered by complexity thresholds, code style violations or security vulnerabilities.

Errors are critical findings, that represent issues that violate coding standard or best practice. They require immediate attention.

Table 3.3 shows the results for the tested applications of "TU Graz Learning Lab". The total number of 294 Violations, consisting of 34 Information, 149 Warnings and 111 Errors, indicates poor software quality and confirms the project's state of obscurity, which was described in Section 1.2.

Application	Information	Warnings	Errors	Σ
Buchstabenpost	1	6	7	14
Divisionstrainer	0	8	5	13
EinmalEins	10	25	19	54
IderBlogExercises	1	32	12	45
JuniorPlusMinus	2	5	17	24
Lesetrainer	5	15	12	32
Multiplikationstrainer	6	17	10	33
PlusMinus	2	21	8	31
UserManager	7	20	21	48
TOTAL	**34**	**149**	**111**	**294**

Table 3.3: PhpMetrics Violations

Dependencies

Keeping the dependencies of the project up to date is very important for security. Outdated modules not only impact the maintainability of the project, but can also reveal information about the general security of the web application. Additionally, dependencies between security requirements may cause additional vulnerabilities (Wang et al., 2022).

Table 3.4 presents the number of each application's direct dependencies, excluding transitive ones, and indicates how many require an upgrade. All applications collectively use 42 different modules. Thus, the total number in the table includes dependencies that are counted more than once. This approach was deliberately chosen as the process of updating might be similar across applications, yet the extent of a dependency's integration into a project can vary. Additionally, not all dependencies across applications are outdated from the same version. In general, the analysis reveals that there is no system in place to ensure comprehensive project updating or to guarantee that all application dependencies are of the same version.

Application	**#Modules**	**#Need Upgrade**
1x1 Trainer	9	9
Buchstabenpost	27	24
Divisiontrainer	9	9
IDeRBlog Exercises	13	13
Junior Plus-Minus-Trainer	18	17
Lesetrainer	25	22
Mathe-Multi-Trainer	12	12
Plus-Minus-Trainer	12	12
UserManager	22	20
TOTAL	**147**	**138**

Table 3.4: Applications and Dependencies

3.2.3 Dynamic Analysis

The dynamic analysis is done with OWASP ZAP. The following workflow is performed for each application:

1. Define scope and context: Given that each application has its own authentication process, they must be tested individually. For this purpose, the scope must be defined to ensure that the dynamic test focuses solely on the respective application, without involving others. Furthermore, the testing process needs to be designed in such a way that it understands how to perform login procedures and ascertain authentication status.

2. Spider: The traditional spider process creates a map of Uniform Resource Locator (URL) routes within the application. It achieves this by starting at an initial point and recursively following the links present in the application.
3. Ajax Spider: This extends the URL map from the previous step, but not merely through link following. The Ajax Spider generates links dynamically via a real browser, ensuring broader coverage. This is particularly important because modern applications often rely on "Asynchronous JavaScript" and Extensible Markup Language (XML), elements that are difficult to detect with a traditional spider.
4. Active Scan: Once all the routes have been identified, OWASP ZAP then can use them to perform an active scan, attempting various attacks to find potential vulnerabilities. The active scan is performed within different contexts, implying that it is run multiple times to ensure all routes associated with different roles are thoroughly tested.

Then OWASP ZAP documents the vulnerabilities found along with the request and associated parameters. It also categorizes them and assigns a risk level. The following risk level exists:

1. Informational
2. Low Risk
3. Medium Risk
4. High Risk

OWASP defines the risk level as a combination of the likelihood of a vulnerability being exploited and the impact of what could happen. That is, the higher the likelihood and impact, the higher the risk level, and vice versa.[6]

OWASP ZAP is able to find 22 different categories of potential vulnerabilities or security flaws in all applications together. Table 3.5 shows how many categories with the corresponding risks were found per application. All applications revealed potential vulnerabilities at the low and medium risk levels. However, high-level alerts were only found in "IderBlogExercises", "JuniorPlusMinus", and "PlusMinus".

[6] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology visited on 07/10/2023

Application	Informational	Low	Medium	High	Σ
Buchstabenpost	3	5	2	0	10
Divisionstrainer	3	6	5	0	14
EinmalEins	3	5	4	0	12
IderBlogExercises	3	7	5	1	16
JuniorPlusMinus	3	5	6	2	16
Lesetrainer	3	6	3	0	12
Multiplikationstrainer	4	6	4	0	14
PlusMinus	4	7	8	1	20
UserManager	4	5	4	0	13

Table 3.5: Application Risk Level Count

Table 3.6 presents all categories of potential vulnerabilities found, as well as the number of corresponding requests that triggered them across all applications. Four instances of "Cross Site Scripting (Reflected)" were identified, wherein Java Script (JS) can be executed through a URL, along with one instance of a potential "SQL Injection". Both are classified as high-risk vulnerabilities. Additionally, numerous medium-risk alerts were detected, such as "Content Security Policy (CSP) Header Not Set" and "Absence of Anti-CSRF Tokens". The low-risk alerts mostly include information that a potential attacker could utilize.

Alerts	Risk Level	Total
Cross Site Scripting (Reflected)	High	4
SQL Injection	High	1
Absence of Anti-CSRF Tokens	Medium	142
Application Error Disclosure	Medium	2
Buffer Overflow	Medium	15
Content Security Policy (CSP) Header Not Set	Medium	2094
Missing Anti-clickjacking Header	Medium	84
Vulnerable JS Library	Medium	11
Parameter Tampering	Medium	7
XSLT Injection	Medium	3
.htaccess Information Leak	Medium	4
Application Error Disclosure	Low	10
Cookie No HttpOnly Flag	Low	52
Cookie without SameSite Attribute	Low	77
Server Leaks Version Information via “Server” HTTP Response Header Field	Low	2342
Timestamp Disclosure - Unix	Low	8430
X-Content-Type-Options Header Missing	Low	251
Big Redirect Detected (Potential Sensitive Information Leak)	Low	93
Information Disclosure - Suspicious Comments	Informational	186
Modern Web Application	Informational	101
User Agent Fuzzer	Informational	1574
User Controllable HTML Element Attribute (Potential XSS)	Informational	162

Table 3.6: OWASP ZAP Scan Results

The presence of numerous potential vulnerabilities indicates that the application does not adhere to best security practices in most aspects. Additionally, the findings of the dynamic analysis correspond with previously discovered flaws in the static analysis, which highlighted a general lack of software quality. Indeed, this convergence is expected, given that sound security principles are based on the foundation of good software development processes. The absence of such processes further exacerbates the issues.

Remarkably, the "SOAP" API did not trigger any OWASP ZAP errors, indicating its security from a dynamic testing standpoint. This outcome may be attributable to thorough testing. The "SOAP" API represents the most crucial component of the project, serving as the interface for all applications and managing the authentication process.

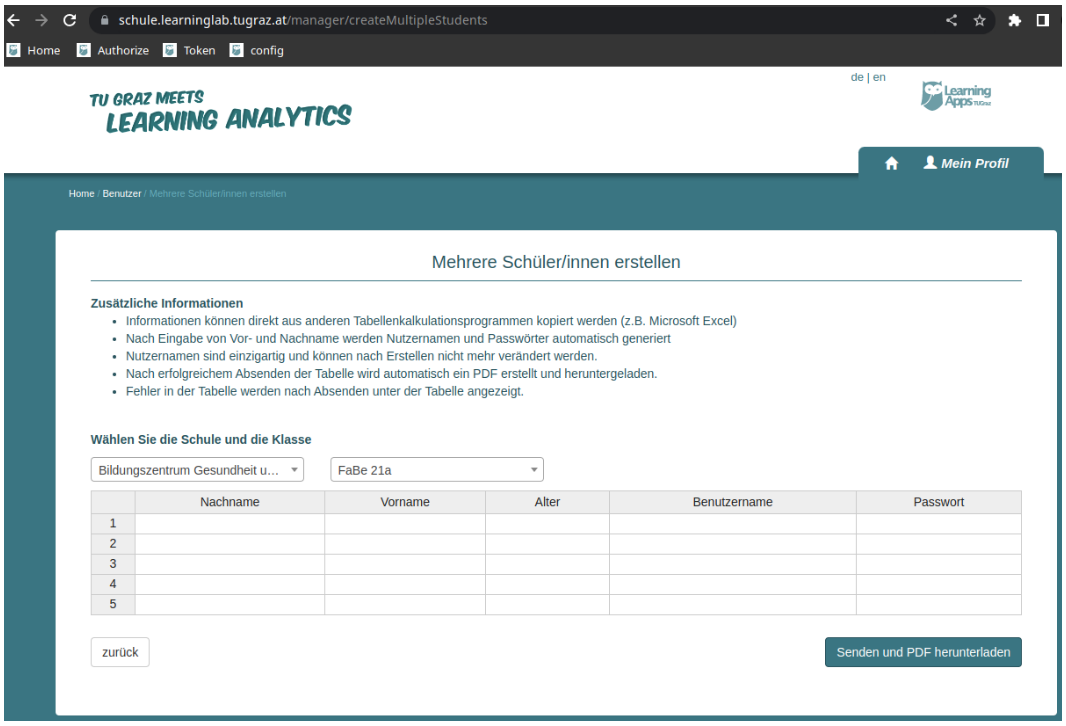
Additionally, the quality of the TLS connection is also dynamically tested. This is why the live page is tested via "Qualys SSL Labs". The test did not find any serious vulnerabilities, moreover the configuration follows best practices and therefore can be considered secure.

3.2.4 Manual Review

The manual review is carried out in conjunction with the static and the dynamic review. Firstly, OWASP ZAP provides a so-called "HUD" which allows intersecting requests directly in the browser and to apply various attacks or fuzzing while browsing. In general, it is very easy to put the applications into "Internal Server Errors", which means that the server is in an undefined state or throws an exception.

This can usually be done by either deleting or modifying parameters or cookies directly in the requests. Further, it shows that the authorization process is severely broken, some applications do not log out when the session ID is deleted, or it is at least possible at the front end to view admin or teacher content.

Additionally, certain routes such as "baseurl/manager" check only for user login status and not for authorization. This implies that a logged-in user, regardless of their role, can add arbitrary users to arbitrary school classes, as evidenced in Screenshot 3.1. Whereas teachers should only be able to add students to their class and only admins should have full access to all classes. However, the primary focus of the manual review lies in the vulnerabilities already discovered, most of which could be confirmed. The outcome is a security report for each application, the details of which are primarily described in the dynamic analysis.


schule.learninglab.tugraz.at/manager/createMultipleStudents
Home
Authorize
Token
config
de | en
TU GRAZ MEETS
LEARNING ANALYTICS
Learning Apps
Mein Profil
Home / Benutzer / Mehrere Schüler/innen erstellen

Mehrere Schüler/innen erstellen

Zusätzliche Informationen

- Informationen können direkt aus anderen Tabellenkalkulationsprogrammen kopiert werden (z.B. Microsoft Excel)
- Nach Eingabe von Vor- und Nachname werden Nutzernamen und Passwörter automatisch generiert
- Nutzernamen sind einzigartig und können nach Erstellen nicht mehr verändert werden.
- Nach erfolgreichem Absenden der Tabelle wird automatisch ein PDF erstellt und heruntergeladen.
- Fehler in der Tabelle werden nach Absenden unter der Tabelle angezeigt.

Wählen Sie die Schule und die Klasse

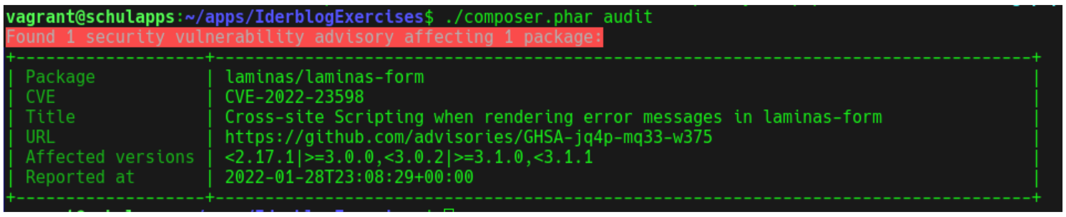
Bildungszentrum Gesundheit u... | FaBe 21a

	Nachname	Vorname	Alter	Benutzername	Passwort
1					
2					
3					
4					
5					

Figure 3.1: Failed Authorization on baseurl/manager

Furthermore, OWASP ASVS is used to check if the "TU GRAZ Learning Lab" reaches its security "Level 1". It has many requirements that cannot be checked automatically. Therefore, it is necessary to look at each constraint manually and check if it is fulfilled or if it is possible to find a negative example in the reports or manually in the project.

In the context of out-of-date dependencies, the tool "Composer" offers the audit functionality, where it is possible to discover modules with potential security vulnerabilities. Especially in the "Cross Site Scripting (Reflected)" vulnerabilities, it was also possible to find the cause in an outdated module, as the Screenshot 3.2 shows. This shows that the vulnerability could have been mitigated, when the application would have been updated.



```
vagrant@schulapps:~/apps/IderblogExercises$ ./composer.phar audit
Found 1 security vulnerability advisory affecting 1 package:
+-------------------+----------------------------------------------------------------------------------+
| Package           | laminas/laminas-form                                                             |
| CVE               | CVE-2022-23598                                                                   |
| Title             | Cross-site Scripting when rendering error messages in laminas-form               |
| URL               | https://github.com/advisories/GHSA-jq4p-mq33-w375                                |
| Affected versions | <2.17.1|>=3.0.0,<3.0.2|>=3.1.0,<3.1.1                                            |
| Reported at       | 2022-01-28T23:08:29+00:00                                                        |
+-------------------+----------------------------------------------------------------------------------+
```

Figure 3.2: Security Advisory Composer Audit

3.3 Vulnerability Analysis

This section presents the identified vulnerabilities and analyzes their implications. Firstly, the issues related to code quality and maintainability are discussed. Then, the problems associated with dependency management are examined. This is followed by a discussion of the dynamic analysis results. Lastly, this section synthesizes all these issues in an attempt to uncover the root cause of the identified vulnerabilities.

3.3.1 Maintainability

As described in Section 3.2.2 the project has 95 classes of moderate complexity, 56 classes of high complexity and 14 classes of untestable complexity. High cyclomatic complexity indicates the tendency for escalated testing and general maintenance issues. Therefore, a refactoring of the existing codebase is required to improve at least the 14 classes with untestable complexity. Furthermore, a refactoring of the high complexity classes will also provide an improvement for the maintainability of the project.

After that, it makes sense to write tests for the improved classes and introduce an automated workflow for the software development process. This can be achieved by introducing a test-driven development. Test-driven development involves writing a test case for a new feature before the actual implementation of the feature. Once the feature is implemented, it's immediately tested using the predefined test case (Jureczko and Mlynarski, 2010).

Further tests for the whole project must be written to a sufficient extent to have a significant impact on the development cycle. This is an integral factor for the preservation of the code quality, therefor for the maintainability of the whole project and consequently for the security of the "TU Graz Learning Lab".

Equally important is the introduction of an automated static analysis tool, such as a linter (Acar et al., 2017). "PhpMetrics" found 294 violations, which indicates serious concerns within the code, that has again consequences on maintainability and security. This grievance must be remedied in order to enable a proper workflow in the project.

Refactoring of the project can be used to introduce a proper software development cycle. This includes code reviews and a maintained documentation, in addition to the previously mentioned automated tests and static analysis tool.

This process will require considerable effort, hence the necessity to prioritize the problems. The following list offers a prioritized approach to resolving these issues:

1. Introduction of a secure software development cycle with code reviews. This can improve the quality of the subsequent work.
2. Refactoring of untestable classes.
3. Massive extension of the existing "PHP" unit tests to achieve sufficient code coverage.
4. Introduction of an automated testing process, to ensure proper testing within the previously established software development cycle.
5. Correction of "PhpMetrics" violations and an integration of an automated linter.
6. Continuous refactoring of the remaining high-risk classes to align the structure of the various applications.

This process must be documented to ensure that the project can proceed independently of the current development personnel.

3.3.2 Dependencies

As seen in Section 3.2.2 the analysis reveals a conspicuous absence of a system that guarantees a consistent and updated state of the project's dependencies. Outdated modules not only impact maintainability problems, but are also a direct indicator of the overall security state of web applications (Manjunath, 2018).

While the "TU Graz Learning Lab" uses "Composer" as a dependency manager, the system consists of nine separate applications, making management potentially confusing. Each application requires individual maintenance and updates, which can be time-consuming due to the manual work needed to gain an overview of the entire project. Consequently, the applications are not consistently updated.

Given that the applications are developed with the assistance of "GitLab", one option is to use "GitLab Dependency Scanning"[7] to automate the process. This tool can scan the dependencies of the applications for known vulnerabilities and facilitate their fixing as soon as they become publicly known.

3.3.3 OWASP ZAP

This section describes the vulnerability categories found, analyzes their causes, and provides a mitigation strategy. As described in Section 3.2.3, 22 categories of different risk levels were found in numerous instances. Due to the enormous number of over 15,000 alerts, the order of remediation must be prioritized.

Furthermore, it makes sense to start fixing the problems only after the maintainability and dependency problems described in the previous sections have been solved. On the one hand, the introduced software development cycle can improve the code of the fixed vulnerabilities, mitigate new vulnerabilities during implementation, and on the other hand, the upgrade process will fix many of the existing vulnerabilities.

[7] https://docs.gitlab.com/ee/user/application_security/dependency_scanning/ visited on 08/02/2023

Despite the vast amount of individual vulnerabilities, this thesis discusses the common themes and evident trends, which are visible through the actual OWASP ZAP reports. The following description of the categories can be seen as a prioritization by the risk level of the vulnerabilities found.

Cross Site Scripting (Reflected)

Risk Level: High

Total Instances: 4

"Reflected Cross Site Scripting (XSS)" is a vulnerability where malicious scripts are injected into a trusted website. In the reflected context, the script is embedded in an URL, which is then sent to the victim. When the user clicks on the link, the script is executed in their browser, reflecting the exploit from the web server. The script can then access sensitive information on the browser. For example, in our case, the hashed password can be accessed as it is stored in local memory (KirstenS et al., 2023).

The prevention of XSS involves strategies like input validation, output encoding, and the implementation of a Content Security Policy (CSP) through headers to mitigate the impact of such attacks. Additionally, it should be noted that some outdated "Laminas" modules carry known security vulnerabilities related to XSS.

SQL Injection

Risk Level: High

Total Instances: 1

"Structured Query Language (SQL) injection" is a technique that manipulates SQL queries through user input, potentially leading to unauthorized data access and manipulation. Although Laminas uses Object-Relational Mapping (ORM), this does not entirely eliminate the risk of SQL injections. Hence, there's a need to ensure that user input is appropriately validated.[8]

[8] https://www.zaproxy.org/docs/alerts/40018/ visited on 07/15/2023

Absence of Anti-CSRF Tokens

Risk Level: Medium

Total Instances: 142

Implementation of anti-Cross-Site Request Forgery (CSRF) Tokens mitigates CSRF attacks, which trick victims into sending a malicious request using their authenticated identity. This is achieved by providing each session with unique and unpredictable tokens that are included in each state-changing request.[9]

In this context, the "Laminas-Form" module provides an element for exactly this problem, and it must be used to mitigate this kind of attack. XSS attacks can bypass CSRF protection.[10]

Application Error Disclosure

Risk Level: 2 Medium, 10 Low

Total Instances: 12

"Application Error Disclosure" pertains to the unintentional disclosure of sensitive information through error messages. Such disclosure can give an attacker a valuable insight into the structure of the project or provide general information, that can be used for an exploit. [11]

In the case of "TU Graz Learning Lab", the two medium-risk alerts disclose paths within the project that can be used to gain information about its structure. The low-risk alerts do not provide direct path information, but information can also be gathered via error codes. These errors mainly occur due to unhandled exceptions or the absence of custom error pages, with the details typically stored in server-side logs.

[9]https://www.zaproxy.org/docs/alerts/10202/ visited on 07/15/2023
[10]https://docs.laminas.dev/laminas-form/ visited on 07/15/2023
[11]https://www.zaproxy.org/docs/alerts/90022/ visited on 07/15/2023

Buffer Overflow

Risk Level: Medium

Total Instances: 15

"Buffer Overflow" errors occur when a program writes more data to a buffer than it can accommodate. In the worst case, it may allow an attacker to execute arbitrary code on the server side by overwriting the memory spaces of the background processes. While "PHP" is generally not vulnerable to buffer overflows due to its high-level nature, such errors can still trigger consequential damage. For example, they may cause service unavailability through denial-of-service attacks. [12]

To mitigate this type of error, a proper validation and sanitation of the parameters is necessary. Additionally, it is necessary to be aware how the parameters are used in the whole context. Implementing a secure coding standard can help to avoid such vulnerabilities.

Content Security Policy (CSP) Header Not Set

Risk Level: Medium

Total Instances: 2094

Content Security Policy (CSP) is a security feature that helps detect and mitigate certain types of attacks, including XSS and data injection attacks. This is achieved by adding a layer directly into the Hypertext Transfer Protocol (HTTP) headers in a standardized manner. Consequently, it becomes possible to declare whitelisted sources of content that the browser is permitted to load and execute on the page.[13]

To mitigate the absence of CSP headers a policy must be defined and then this can be implemented in a secure coding standard. Further, automated tests can be implemented to spot the absence during development without manual review. This policy must be regularly updated.

[12] `https://www.zaproxy.org/docs/alerts/30001/` visited on 07/15/2023
[13] `https://www.zaproxy.org/docs/alerts/10038-1/` visited on 07/15/2023

Missing Anti-clickjacking Header

Risk Level: Medium

Total Instances: 84

The "Anti-clickjacking Header" is a similar security feature to the CSP policy header. However, it protects against clickjacking, also known as a UI redress attack. In this attack, a malicious site tricks the user into clicking on a hidden element by overlaying it. The user believes they are performing actions on the malicious site when, in fact, the interactions are being executed on the legitimate site below.[14]

This can be mitigated by implementing an "X-Frame-Option" header in the web applications and setting it to either "SAMEORIGIN" or "DENY". In Laminas this can be configured via the "laminas-http" module.[15] It should also be included in the security policy.

Vulnerable JS Library

Risk Level: Medium

Total Instances: 11

Known vulnerable JS libraries, potentially allow an attacker to exploit the vulnerability and harm the web application. In the context of "TU Graz Learning Lab" this happens because of outdated dependencies. Therefore, this warning will be fixed after the update process of the project. As written in Section 3.3.2 an automated dependency system would have mitigated it or at least it would be known.[16]

[14]https://www.zaproxy.org/docs/alerts/10020-1/ visited on 07/15/2023
[15]https://docs.laminas.dev/laminas-http/headers/ visited on 07/15/2023
[16]https://www.zaproxy.org/docs/alerts/10003/ visited on 07/15/2023

XSLT Injection

Risk Level: Medium

Total Instances: 3

While Extensible Stylesheet Language Transformations (XSLT) is not used directly in the project, the warning indicates that there may be other chain trigger configuration issues that cause this error. In general, XSLT injection is an attack that allows an attacker to manipulate data processed by a XSLT engine. In the worst case, an attacker could execute arbitrary code through specially crafted input.[17]

To mitigate this type of error, proper validation and sanitization of the parameters is required. Since this is not the first time this problem has occurred, it must be included in the security policy so that all user input is properly validated.

.htaccess Information Leak

Risk Level: Medium

Total Instances: 4

The ".htaccess" file is typically used in the configuration of the "Apache" web server. This file can contain valuable information for an attacker. So this error has nothing to do with the "PHP" framework itself, but with the permissions of the file on the web server. This file should only be accessible from the server side. To mitigate this vulnerability, the file's permissions must be checked, especially on the production site.[18]

[17]https://www.zaproxy.org/docs/alerts/90017/ visited on 07/15/2023
[18]https://www.zaproxy.org/docs/alerts/40032/ visited on 07/15/2023

Cookie No HttpOnly Flag

Risk Level: Low

Total Instances: 52

The "HTTP-only" flag in a cookie ensures that the cookie cannot be accessed by a client-side script. This reduces the damage from XSS attacks. If the flag is not present, it is possible for JavaScript to access the cookie, and the information can be transferred or even the session hijacked. [19]

In the context of the "TU Graz Learning Lab" even some of the session ID cookies are not set to "HTTP-only" in all cases. Therefore, it must be ensured that the relevant cookies are always set to "HTTP-only". This fact must also be included in the security policy documentation.

Cookie without SameSite Attribute

Risk Level: Low

Total Instances: 77

The "SameSite" attribute in a cookie ensures that the cookie cannot be set as a result of a cross-site request. This reduces the damage from CSRF attacks. Without it, cookies can be accessed and sent by third-party sites. This can potentially lead to information leakage or make the application more vulnerable to CSRF or timing attacks. [20]

Again, to mitigate this vulnerability, it is necessary to ensure that every relevant cookie has the "SameSite" attribute. To ensure this, it is necessary to include this fact in the security policy documentation and to think about it during code reviews.

[19] https://www.zaproxy.org/docs/alerts/10010/ visited on 07/15/2023
[20] https://www.zaproxy.org/docs/alerts/10054/ visited on 07/15/2023

Server Leaks Version Information via “Server” HTTP Response Header Field

Risk Level: Low

Total Instances: 2342

The HTTP response header "Server" contains information about the server and the program being used. This can reveal information, which can be abused by an attacker and is a form of information leakage. [21]

In the context of "TU Graz Learning Lab" all found instances provide information about the specific version of the web server "Apache" and the name of the operating system "Debian". This can be mitigated by configuration of the web server.

Timestamp Disclosure - Unix

Risk Level: Low

Total Instances: 8430

Timestamp disclosure is also a form of information leakage. Unix timestamps in web applications can be aggregated to reveal exploitable patterns. Therefore, the exposure of system-specific data must be minimized. This is another point that needs to be addressed in the security policy documentation.[22]

X-Content-Type-Options Header Missing

Risk Level: Low

Total Instances: 251

The "X-Content-Type-Options: nosniff" HTTP response header protects the browser from Multipurpose Internet Mail Extensions (MIME) sniffing. While

[21]https://www.zaproxy.org/docs/alerts/10036-2/ visited on 07/15/2023
[22]https://www.zaproxy.org/docs/alerts/10096// visited on 07/15/2023

MIME was originally designed for email, it is used on the Web to indicate the type of files being transferred. If an "X-Content-Type-Options" header is set to "nosniff", the browser does not interpret the response and therefore cannot be interpreted as a different type, leading to vulnerabilities.[23]

To mitigate this vulnerability, it is necessary to include this header in server responses. This must also be documented in the software security policy and tested regularly.

Big Redirect Detected (Potential Sensitive Information Leak)

Risk Level: Low

Total Instances: 93

This error is thrown because of non-empty bodies in redirect responses. In the context of the "TU Graz Learning" lab, no information leakage is found, but for best practice, redirect responses should have almost no content. This bug needs to be included in the project's software security policy.[24]

3.3.4 Interim Summary

As seen in this section, the maintainability problems, the dependency issues and the actual security flaws are strongly interconnected. Maintainability issues not only slow down the development, they also provide a direct risk to the security of the project. The problem is amplified by the dependency issues, as outdated dependencies are not only affecting maintainability, but also bring in known vulnerabilities that attackers can exploit.

The introduction of a secure software development cycle, including code reviews, automated testing and frequent dependency updates, can mitigate the problems. In addition, an automated dependency management system and a well documented environment, including a security policy, can address the challenges. Further, a continuous review and improvement process,

[23] `https://www.zaproxy.org/docs/alerts/10021/` visited on 07/15/2023
[24] `https://www.zaproxy.org/docs/alerts/10044/` visited on 07/15/2023

following best practices and coding standards, including static and dynamic analysis through Continuous Integration/Continuous Delivery (CI/CD), will ensure a more secure and maintainable application.

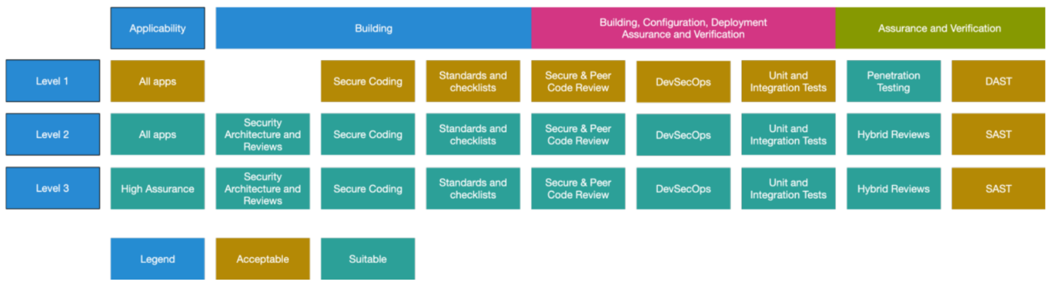
3.4 Comparison with Security Standards

With all the information from the security analysis and the vulnerability analysis, it is possible to benchmark the security status of the "TU Graz Learning Lab" with the OWASP ASVS "Level 1" requirements as stated in Section 3.1. The Table 3.7 shows the result of the comparison. The OWASP ASVS "Level 1" has a total of 128 requirements. Of these, the project satisfies 45, fails to meet 68, and 15 are requirements for which it is irrelevant to comment, as they were not implemented in the project, such as, for instance, an Lightweight Directory Access Protocol (LDAP) authentication.

Status	**#ASVS Level 1 Requirements**
Fulfilled	45
Not Fulfilled	68
Not Implemented	15
TOTAL	**128**

Table 3.7: ASVS Level 1 Requirements Status

The fact that over half of the "Level 1" requirements aren't met indicates significant security gaps that need to be addressed. "Level 1" is the minimum standard that all web applications should fulfill, according to OWASP ASVS. This outcome is not surprising, given the lack of structure at every level observed so far, which exacerbates the issues. In OWASP, 2021, Figure 3.3 presents the various levels of application security verification and provides an explanation as to why numerous requirements often remain unmet. Almost none of the "Level 1" requirements shown in the picture are fulfilled.



	Applicability	Building			Building, Configuration, Deployment Assurance and Verification			Assurance and Verification	
Level 1	🟨 All apps		🟨 Secure Coding	🟨 Standards and checklists	🟨 Secure & Peer Code Review	🟨 DevSecOps	🟨 Unit and Integration Tests	🟩 Penetration Testing	🟨 DAST
Level 2	🟩 All apps	🟩 Security Architecture and Reviews	🟩 Secure Coding	🟩 Standards and checklists	🟩 Secure & Peer Code Review	🟩 DevSecOps	🟩 Unit and Integration Tests	🟩 Hybrid Reviews	🟨 SAST
Level 3	🟩 High Assurance	🟩 Security Architecture and Reviews	🟩 Secure Coding	🟩 Standards and checklists	🟩 Secure & Peer Code Review	🟩 DevSecOps	🟩 Unit and Integration Tests	🟩 Hybrid Reviews	🟨 SAST

Legend: 🟨 Acceptable, 🟩 Suitable

Figure 3.3: OWASP Application Security Verification Standard 4.0 Levels

Moreover, the "Level 1" requirements do not encompass automated testing or a secure software development cycle, both of which are suggested solutions for the project's structural problems. Considering the sensitive data handled by the "TU Graz Learning Lab", it should aim to meet the OWASP ASVS Level 2 standards in the long run. Given that these requirements are essential, it is particularly crucial in a student development context to implement them as soon as possible, as will be discussed in the following section.

3.5 Interpretation of Results in the Student Development Context

The security analysis gives a significant insight into the project "TU Graz Learning Lab". The analysis reveals serious challenges and characteristics that have an enormous impact on the development and security of the applications. As stated before, the project has grown continuously over a decade, without any serious project management nor any software development cycle.

The actual implementation can be seen as "cowboy coding" with a "hands-off" development process, which refers to a development process where once a part is finished, it is passed on to the next developer and the original author has no further involvement with it. Unfortunately, this is difficult to prevent in a student environment (Włodarski, Poniszewska-Marańda, and Falleri, 2022).

Undergraduate and graduate students typically participate in such a project for a short period of time, usually a semester or an academic year. Furthermore, there is a transient nature in student involvement, the experience in web development can vary enormously due to the fact that especially in bachelor classes there are few practical web application development tasks and so it depends on the student's specific background how experienced they are.

All of this exacerbates existing problems with code maintainability and documentation, which are critical parts of a secure software development cycle. The findings from the analysis, with high cyclomatic complexity in the codebase and a lack of comprehensive automated testing, are further indications that confirm these problems.

Therefore, maintainability is extremely important in a student-developed context. With high staff turnover, the code must be sufficiently understandable to new developers. While many students primarily develop their standalone applications and use the "SOAP" interface for integration, code is often borrowed from other applications, which can introduce the same problems into the new applications.

In addition, the scientific staff, who are usually not involved in the project for very long, must be able to learn in a reasonable amount of time and maintain the project without too much effort. High complexity, insufficient testing, and lack of proper documentation create significant barriers for new developers. This also leads to the inconsistent update pattern for dependencies and all the vulnerabilities discovered by security analysis.

To mitigate these risks, a secure software development lifecycle must be implemented. This includes comprehensive documentation, including a secure development policy and coding standard for maintainable coding practices, automated testing, and dependency management. Therefore, a planned development process can also improve the usability of the project and make it less likely to fail (Germain and Robillard, 2005).

Further, iterative methodologies like agile practices can also improve the development in a student context enormously (Wang et al., 2022). Future work could explore how these practices can be implemented in the student context.

4 Security Implications in Protocol Implementation: A Case Study of OAuth 2.0 and TU Graz Learning Lab

After the security analysis, this part of the thesis shows how the found issues influence an actual development. Therefore, none of the mentioned mitigation strategies are applied to see how the issues affect development in a practical environment.

This chapter describes an attempted implementation of "OAuth 2.0" to replace the "SOAP" API of the "TU Graz Learning Lab". The first subsections explain what is important when implementing a protocol in web applications and give a general overview of "OAuth 2.0" and how it can be used for authorization and authentication in web applications.

Afterward, this thesis documents the implementation step by step and analyzes it in the context of system design, dependency management and student developed applications. Finally, a connection to the previous chapter will be drawn.

4.1 Introduction to Protocol Implementation and Security

Introducing a new protocol implementation into an existing application entails numerous key considerations that should be addressed. Particularly in web applications, gaining a complete understanding of the system and the implications of new components within the overall context can be challenging. Furthermore, in software projects, there is often a discrepancy in the level of attention given to security and functionality (Rowe, Baskerville, and Wolff, 2012).

Implementing a protocol into an existing project in web development need a clear structure and a plan how to achieve the goal. Therefore, following steps can be considered:

1. Understanding the Protocol: The protocol must be understood in the context of the entire system.
2. Security: The implementation of the protocol must improve the security and not introduce new vulnerabilities.
3. Interoperability: The protocol must work seamlessly with the existing components, and it must be easy to implement new components with it.
4. Error Handling: Implementation of a robust error handling is essential for future arising issues.
5. Testing: Rigorous testing is crucial to assure the protocol's functionality and to secure maintainability following future adaptations and modifications.
6. Maintenance and Documentation: The protocol implementation must be readily maintainable and comprehensively documented for future developers.
7. Compliance: The protocol must be implemented in a way that assures following best practices and up-to-date standards.
8. Updates and Evolution: Plans for future updates or extensions must be kept in mind.

4.2 OAuth2: An Overview

> "The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service,

> or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849." Hardt (2012)

Therefor, "OAuth 2.0" is an authorization protocol that allows applications to obtain delegated access to HTTP services by issuing access tokens. It depends on a secure channel in the context of web application based on Hypertext Transfer Protocol Secure (HTTPS). The provided confidentiality by HTTPS enables the non-disclosure of unauthorized individuals through encryption and the integrity of HTTPS ensures, that no data is tampered with or altered through source legitimation.

"OAuth 2.0" lacks built-in mechanisms for authenticating users on client applications. "OpenIDConnect" fills this gap by adding an authentication layer on top of it. This structure can be seen in the Figure 4.1 where "OAuth 2.0" handles the authorization and "OpenIDConnect" is responsible for the authentication.

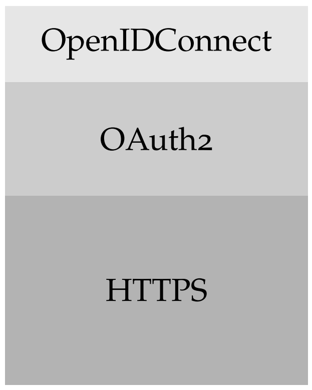


Figure 4.1: Authentication over OpenIDConnect based on OAuth 2.0 and HTTPS

To understand the protocol and its benefits in the context of "TU Graz Learning Lab" the next subsections will describe the "Authorization Code Flow" of "OAuth 2.0" and the "OpenIDConnect" layer above it.

4.2.1 Authorization Code Flow

The "Authorization Code Flow" is a "grant type", respectively a way to obtain an access token, in "OAuth 2.0". This is achieved by redirecting the user from the requested application to the authentication server. Due to this structure, this flow is the best choice for "TU Graz Learning Lab". As described in the Section 1.1.2 each application has its own login form and this can be unified through one login at the authentication server via the "Authentication Code Flow".

The following Figure 4.2 explains the process, whereby the "User" is often denoted as "Resource Owner", the "Application" as "Client", the "OAuth2Server" as "Authorization Server" and the "API" as "Resource Server" (Bucher and Christensen, 2019):

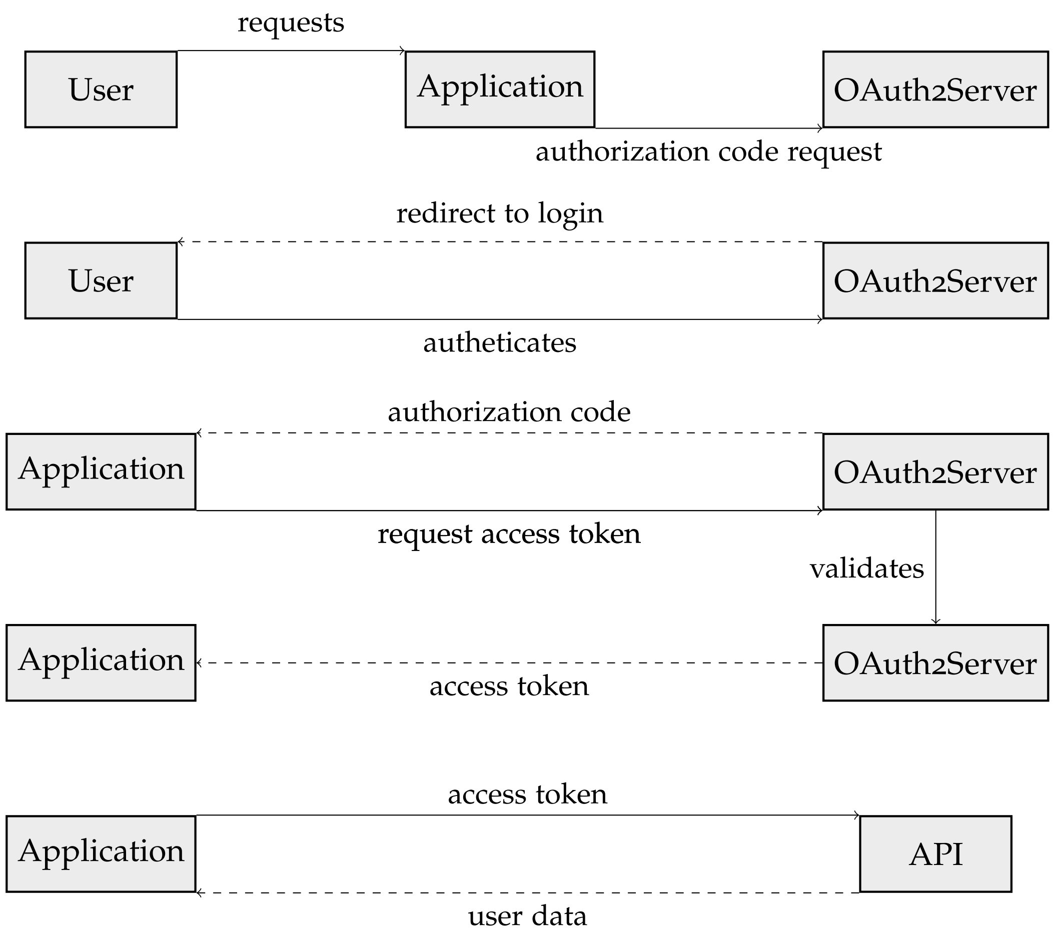


Figure 4.2: Authorization Code Flow

1. The user requests the application and gets redirected by the application to the "OAuth 2.0" server. The application must be registered at the "OAuth 2.0" server initially with an ID and a secret, which gets sent and checked. Additionally, a redirect URL is also stored or sent from the application to the "OAuth 2.0" server and used to redirect the user back to the application, once the user is authenticated.
2. The user is redirected to the login form from the "OAuth 2.0" server, in our case the "UserManagers" login page, to authenticate. If the user is already authenticated, this step is skipped, due to the session based login, that already exists.

3. Then the "OAuth 2.0" server sends the authorization code directly to the application using the stored redirect URL.
4. The application can use this code to request an access token. Therefore, the server validates the code and sends the token to the application, which stores it.
5. The application can use this token to access protected resources on the API.

4.2.2 OpenIDConnect

After the client has requested the access token, the "OpenIDConnect" extension comes into play. Since "Oauth 2.0" has not a standardized way for authentication, through "OpenIDConnect" sends an additional ID token in the request. This ID token is a JSON Web Token (JWT), that contains user information in a standardized way. This ensures compatibility and is especially important, when the user visits an application the first time, so that a new entry with his information can be created in the respective database of the application.

4.3 Attempted Implementation of OAuth 2.0

As stated in the previous section and in combination with the introduction, "OAuth 2.0" and its mechanism for authentication with "OpenIDConnect" can be a good alternative to the current authentication over the "SOAP" protocol. The following subsection describes the attempted implementation and its problems. This is achieved by a journal like structure, so that all issues can be addressed and interpreted in their context. Given that not only the applications were developed by students, but this document is also authored by a student, this approach enables an insightful examination of workflow through self-reflection.

Additionally, it should be noted that the sequence of the following sections does not align with the actual progression during implementation. Given the inadequate documentation, insufficient maintenance, and suboptimal system design, many insights were only gained when progress was obstructed at a certain point, necessitating a return to previous steps to rectify issues. Rather than adhering rigidly to the chronological order of issue discovery, this thesis organizes them in a manner that provides a more coherent understanding of the implementation process.

4.3.1 Provision

Due to infrastructural reasons from Zentraler Informatikdienst (ZID) , which is responsible for hosting the applications on the servers of TU Graz, it is necessary to update the server operating system of the applications from "Debian 10" to "Debian 11" and "PHP 7.3" to "7.4". During the development process, the decision was made to upgrade to Debian 12 and PHP 8.1, as detailed in the subsequent section. In addition, when introducing new components in web applications, it is best practice to update the project beforehand.

This is important not only for security reasons, such as mitigating outdated components with known vulnerabilities, but also for maintainability. The setup of the project is based on a script-driven development environment. Therefore, the following files play a role in the setup of the Vagrant Box and must be adapted when upgrading to new components. Figure 4.3 shows the structure.

```
/webapps
├── /vagrant
│   └── /config
│       └── /shell
│           ├── build__einmaleins.sh
│           ├── custom_php.sh
│           ├── final.sh
│           ├── setup__buchstabenpost21.sh
│           ├── setup__divisionstrainer.sh
│           ├── setup__einmaleins.sh
│           ├── setup__iderblogexercises.sh
│           ├── setup__junior_plus_minus.sh
│           ├── setup__lesetrainer.sh
│           ├── setup__multiplikationstrainer.sh
│           ├── setup__plus_minus_trainer.sh
│           ├── setup__portal_usermanager.sh
│           └── setup__schoolappsconfig.sh
├── install.sh
└── Vagrantfile
```

Figure 4.3: Configuration of provisioned Infrastructure

This structure relies on the Infrastructure as Code (IaC) methodology. Specifically, these files manage the provisioning and configuration of the provisioned infrastructure. IaC in combination with "Vagrant" allows the project to be set up relatively quickly and additional projects with different environmental prerequisites can be integrated simply. Unfortunately, no additional tools are used to develop and maintain the structure. Therefore, every dependency had to be identified manually, and the prerequisites needed for the applications to run had to be researched individually (Rahman, Farhana, and Williams, 2020).

Additionally, some repositories need to be manually added to secure the correct version of the required dependency. During development, numerous problems were encountered with modules no longer supporting "PHP 7.4". Consequently, the decision was made to upgrade to "PHP 8.1".

4.3.2 Dependency Management

"Composer" is a dependency management tool for "PHP". Due to the fact, that all applications have their own dependencies, each application has its own "composer.json" file, which must be updated. The following file is necessary for updating the respective application, as Table 4.4 shows.

```
/webapps
└─/<application>
   └─/src
      └─/<application>
         └─composer.json
```

Figure 4.4: Path of the composer configuration files

Most "Composer" modules are built using the Dependency Injection design pattern and are automatically loaded into applications on demand. For this to happen, these plugins must be configured in the following files, as Table 4.5 shows:

```
/webapps
└─/<application>
   └─/src
      └─/<application>
         ├─/config
         │  ├─/autoload
         │  │  └─/{,*.}{global,local}.php
         │  └─<application>.config.php
         └─/module
            └─/config
               └─/module.config.php
```

Figure 4.5: Configuration files of the imported modules

4.3.3 Artifacts from Zend Framework to Laminas Migration

As mentioned in the Introduction 1.1.2 "Zend Technologies" decided to discontinue the "Zend Framework" in 2019 and the open source community decided to continue the project under the new name "Laminas". During the rebranding, not only the name was changed, but also the structure and the modules were adapted, so that a migration was necessary. This migration process stretched over a longer period of time. Unfortunately, the "TU Graz Learning Lab" applications did not complete this process completely, so the project is still in a state between the two frameworks.

So to be able to install a new module, it was necessary to migrate the main project from "Zend Framework" to "Laminas". This was done under the instruction of the official migration guide.[1] Additionally, multiple modules like "laminas-api-tools" also needed specific migrations. Noticing the need to migrate a module is a matter of checking the documentation, and it is not clear until errors occur while using an application. "Composer's" information about plugins is limited to deprecate only. This means that "Composer" cannot indicate missing migration in "Laminas".

4.3.4 laminas-api-tools

As commonly known in information security and shown by many counterexamples, it is not a good idea to implement new security relevant libraries on your own (Tobias, 2020). Therefore, the best practice is to find good maintained libraries and modules which, not only simplify the implementation but also make it possible to work with a secure and maintained module in the application.

Fortunately, "Laminas", in paritucalar "laminas-api-tools" provides many modules for this use case. After integrating and configuring "laminas-api-tools" the following modules made sense to integrate into the "TU Graz Learning Lab" on the "Usermanager" application to enable authorization and authentication via "OAuth 2.0":

[1] `https://docs.laminas.dev/migration/` visited on 06/13/2023

1. "api-tools-mvc-auth"[2] handles the authentication and authorization part in an API. In the context of "OAuth 2.0" this module handles the authentication request from the client and returns the access token.
2. "api-tools-admin"[3] provides an administration platform for creating, managing and versioning Representational State Transfer (REST) and Remote Procedure Call (RPC) services through a web based Graphical User Interface (GUI), so-called "Laminas API Tools Admin UI".
3. "api-tools-oauth2"[4] provides an "OAuth 2.0" server, based on "oauth2-server-php" [5] library by Brent Shaffer, including all grant types, scopes, "OpenIDConnect" and JWT bearer assertions. It also includes storage adapters to handle tokens, clients and users. In addition, it handles validation, scope checking, and token lifecycle.
4. "api-tools-documentation"[6] is used for creating and managing the API documentation. This is crucial for client implementation, as it helps understand how to interact with the API.
5. "api-tools-documentation-swagger"[7] extends the "api-tools-documentation", generates human and machine-readable documentation and allows the automatic generation of documentation.

These five modules create a foundation for "OAuth 2.0" authorization and authentication in the "UserManager" application. While "api-tools-mvc-auth" and "api-tools-oauth2" are responsible for the technical implementation, the other three allow for easy implementation and maintenance of the "OAuth 2.0" secured API. This gives the project the maintainability and modularity it needs.

[2]https://api-tools.getlaminas.org/documentation/modules/api-tools-mvc-auth visited on 06/14/2023

[3]https://api-tools.getlaminas.org/documentation/modules/api-tools-admin visited on 06/14/2023

[4]https://api-tools.getlaminas.org/documentation/modules/api-tools-oauth2 visited on 06/14/2023

[5]https://github.com/bshaffer/oauth2-server-php visited on 06/14/2023

[6]https://api-tools.getlaminas.org/documentation/modules/api-tools-documentation visited on 06/14/2023

[7]https://api-tools.getlaminas.org/documentation/modules/api-tools-documentation-swagger visited on 06/14/2023

4.3.5 OAuth 2.0 Storage on Server-Side

As stated previously, "OAuth 2.0" requires some information, like keys, user IDs and tokens. These need to be stored securely. The "api-tools-oauth2" module comes with storage adapters that provide an interface for this.

As for now, most of the database structure in the "UserManager" gets build using two SQL files, which are executed during the provisioning process. They also populate the required data. Nowadays, creating tables for the "api-tools-oauth2" module using basic SQL commands has many disadvantages:

1. Data Handling and Manipulation: When using raw SQL object conversations have to be done manually.
2. Complexity: Handling data management through SQL files, which in the case of the "UserManager" have more than 10000 lines, complicates schema migrations and data validation.
3. Security: When using raw SQL it is mandatory to protect against SQL injection and other security vulnerabilities, by using prepared statements or parameterized queries.

Fortunately, the "doctrine" module is also integrated in the "UserManager", but with the deprecated "YAML" notation. "Doctrine" is a ORM tool for "PHP", that provides an interface for mapping and interacting with a database. Therefore, as it is now, all database manipulation, except the initial setup, is done through doctrine.

However, as mentioned above, it would be best practice to use doctrine for initial setup and data insertion as well. This can be done using the "apiskeletons-oauth2-doctrine" module[8], which comes with a mapping for the "api-tools-oauth2" module. Inserting the data into the database will be discussed in the next session.

[8]`https://apiskeletons-oauth2-doctrine.readthedocs.io` visited on 06/14/2023

4.3.6 Setup of laminas-cli for Secure Database Initialization

As stated in the previous section, a mechanism for secure storage of the client data, especially for the keys, is still missing. For that, the module "laminas-cli"[9] fulfills the requirements. This plugin allows executing predefined commands in the "UserManager" from the command line.

Therefore, it is possible to encrypt the client secret using the "bcrypt" hashing algorithm provided by "api-tools-oauth2", and store it in the database using "doctrine". "Bcrypt" uses a salt and performs a series of iterations of a key derivation function, which makes it computationally expensive, thereby offering greater resistance to bruteforce and dictionary attacks, compared to alternatives such as "SHA256". This process ensures secure key storage in an object-oriented manner, which enhances maintainability. Furthermore, this approach can be used to generate dummy entries for future testing, thereby eliminating the need for the "SQL" files.

4.3.7 Transformation of SOAP API to RPC/REST via laminas-api-tools Admin UI

As written in the Problem Statement, Section 1.2, "SOAP" no longer meets the needs of the project. Additionally, it is not part of the "Laminas API Tools" collection, but of the "Laminas MVC" modules, which makes protecting it via the "api-tools-oauth2" complicated and harder to maintain.

Due to the integration of "api-tools-admin", it is now possible to create "PHP" classes for the API through the "admin UI", eliminating the need to alter any configuration files for routing and protecting the API. The structure of the "SOAP" API aligns more with RPC, meaning the function signature can be easily adapted. For REST, however, it is more sensible to configure the entities and the corresponding database structure so that data manipulation can be done directly through REST. Due to the complexity and the fact that all databases of the client applications would need to be adapted, it was decided to initially transfer the "SOAP" structure to RPC.

[9]`https://docs.laminas.dev/laminas-cli/` visited on 06/14/2023

After successful implementation, all necessary adaptations can then be made to integrate REST.

4.3.8 Problems encountered with Secure Token storage on Client-side

As highlighted in the introduction in Section 1.1.2, every application has its own database and is relatively independent. The data management ensures that each application has its own database and gets the required user information from the API at the first login to the application and then stores it in its own database.

Therefore, it's not only the server that needs secure storage for the token, but the client must also store the token securely. Given that "OAuth 2.0" in the context of "TU Graz Learning Lab" is solely used for server-to-server communication, it's unnecessary to send the token to the user, or to store it on the user's side, like it would be in storing it in local storage or in a cookie. This can mitigate attacks such as "Cross-Site Request Forgery", "Session Sidejacking", and "Cookie Theft".

To reuse the existing Session based Login for every application, the idea is to store the tokens in the session storage securely. Thus, the already implemented module "laminas-authentication"[10] can be reused. Firstly, this approach is implemented into the "Divisiontrainer", where the following problem arises.

Whenever a user is authenticated and attempts to access the super global `$_SESSION` variable, the application times out after executing an "exec" command, leaving no error messages or traceable notes in the logs. It is emphasized, that the exec command is used in the "laminas-http"[11] module, which is responsible for the HTTP request. In the background, "laminas-http" build the request to a "curl" command and the executes it via "exec".

To debug this issue, the same form of authentication and token storage in the session was implemented in the "PlusMinus", where the issue manifested

[10] `https://docs.laminas.dev/laminas-cli/` visited on 07/23/2023
[11] `https://docs.laminas.dev/laminas-http/request/` visited on 07/23/2023

itself in the same way. After intercepting the traffic between the application and the "OAuth 2.0" server with a proxy, it is seen that no requests are sent in this context. So this indicates that the problem indeed lies in the codebase.

Further research showed, that although the `$_SESSION` variable is used multiple times in all applications, there can exist problems with the "laminas-session" module.[12]. This module is not directly used by the "Division-strainer", but has a transitive dependency that is not mentioned in the documentation. This happens through the transitive dependency to the "laminas-cache"[13] module, which is indeed a requirement in development environments.

Moreover, the "laminas-cache" module needs a migration to version 3.0, which was not performed till now, due to the fact, that as noted in Section 3.3.2, it is not possible to know that a migration is required without looking at the module documentation.[14] The migration requires the implementation of new satellite packages.

After this realization, it was decided to consider the implementation of "OAuth 2.0" under the circumstances of the not adapted software development process as a failure. Not only because the time resources of this work were more than exhausted, but also for the reason that it is not possible to make statements about the success in the future. Since this is not the first problem of its kind, but the main task of the actual implementation was to find and fix such bugs, this example shows the importance of a well-maintained and up-to-date software development cycle.

4.3.9 Actual Implementation Flow

When beginning the implementation phase of the project, it was decided to update the infrastructure as described in the Section 4.3.1. Due to the lack of testing in the "TU Graz Learning Lab" and the fact that introducing tests for

[12]`https://docs.laminas.dev/laminas-session/storage/` visited on 07/23/2023
[13]`https://docs.laminas.dev/laminas-cache/` visited on 07/23/2023
[14]`https://docs.laminas.dev/laminas-cache/v3/migration/to-version-3/` visited on 07/23/2023

all applications would have exceeded the scope of the work, it was decided to update and install modules as needed and to correct errors when they occur in manual testing after implementation of the components needed for the "OAuth 2.0" authentication flow. Furthermore, this represents the actual coding practice as it has been carried out in the history of development.

This led to a so-called "cowboy coding" implementation flow, which is an approach where there is little to no planning, automated testing or formal procedures. Whereas "cowboy coding" can have advantages such as faster, more flexible and easier development, it only makes sense in an experimental development environment. Maintainability and quality issues make it unsuitable for actual software projects that are also intended for end users (Kropp and Meier, 2015).

Thus, the previous subsections do not reflect the actual order of implementation, but rather can be seen as running back and forth to the already processed sections, as already described in Section 4.3. This workflow functions moderately well when updating the "UserManager", but as soon as more than one application comes into play, the process becomes confusing and the effects of the changes become difficult to comprehend, as seen in the last subsection.

4.4 Linking Implementation Flaws to Security Analysis

When examining the security flaws discovered during the implementation, of the "OAuth 2.0" protocol, several correlations emerge. Therefore, this section will link the security vulnerabilities and design flaws from the security analysis to the issues encountered. Some vulnerabilities from the test reports and the manual review were also noticed during the implementation, but as long as they did not hinder implementation, they will not be mentioned, as this does not provide any additional information.

4.4.1 Manual Dependency Management

The first major challenge was to update the operating system and "PHP". After that, the deprecated and incompatible modules of the "UserManager" had to be updated and additional migrations from "Zend Framework" to "Laminas" were necessary. The incomplete migration revealed a lack of maintenance and proper documentation, issues highlighted in the security analysis. As mentioned above, this only becomes apparent when, for example, the integration of a module does not work, during manual testing and by researching the module's documentation.

4.4.2 Complex Codebase

The high cyclomatic complexity of some classes had direct impacts on understanding the interplay of different components, including dependencies. Therefore, this can make manual dependency management error-prone, potentially leading to issues, as seen during the attempted implementation of "OAuth 2.0". Additionally, this increases the time required for such processes. Furthermore, the relatively complex configuration of the deployed infrastructure complicates the process with ten different .sh files, as seen during the implementation phase.

4.4.3 Lack of Automated Testing

Further, automated testing is critical during dependency updates and, more importantly, during framework migrations. Without testing, it is virtually impossible for a new developer to verify the correct migration of the project, or even the implementation and configuration of existing or new modules. On the long run, this not only introduces functional bugs, but can also cause unnoticed security vulnerabilities.

4.4.4 Insufficient Secure Storage Mechanisms

Although not directly mentioned in the security analysis, the lack of sufficient storage mechanisms is also a major problem in the "TU Graz Learning Lab". Not only are security-relevant parameters such as the session ID stored without any visible security patterns, but also the complexity and workflow of the database initialization could have been reduced by implementing a secure development policy. Besides the complexity of the database initialization, the resulting two-track system of SQL files and "doctrine" is also not best practice and should be simplified, as mentioned in the security analysis.

4.4.5 Lack of Security Policy

Thus, the authentication of the "UserManager" is not following best security practices either, due to the fact that it uses "Sha256", with a salt for password hashing. "Sha256" is not the ideal algorithm for this use case, because of its fast and efficient nature, whereas in password hashing a computationally expensive hash function should be used, to mitigate dictionary attacks (Naiakshina et al., 2017). Although an attempt was made to address this issue during the implementation of "bycrypt", the migration of the old database would be a major concern. This could also have been mitigated by implementing a secure software development policy and is also seen in the security analysis.

4.5 Lessons Learned and Recommendations

The case study of the implementation of "OAuth 2.0" in the "TU Graz Learning Lab" showed by its failure major challenges in the project and gives insights how not to develop software. In particular, the lack of maintenance and modern development practices make a successful development of secure web applications unattainable. The following lessons have been learned from this experience:

4.5.1 Implementation of a Software Development Process

As noted earlier, the challenges encountered during implementation underscored the indispensability of a software development process. As described in the security analysis, an iterative approach is best suited to the student development context. This ensures that the work can be broken down into smaller, more manageable iterations or sprints. It also ensures that smaller parts of the implementation can be used for further development even if a whole part of the project fails, rather than losing all the progress in the event of a failure.

Besides that, security can be taken into account at ever stage. The goal is to catch and resolve security issues as early as possible, when they are typically easier to fix. This requires an additional expense in resources, but the potential cost of not taking security seriously can be much higher. This would lead to an iterative, secure software development cycle, slightly adapted to the small team structure of the "TU Graz Learning Lab", which often consists of only one student and the project owner, or academic staff (Daud, 2010):

1. Planning: The developer, in consultation with the staff, plans the work to be done.
2. Design and Implementation: The developer designs and implements the portion of the application. The work must comply with the established security policy and should be checked with static analysis tools.
3. Testing: Each iteration should include extensive testing, including security testing. This includes automated tests that pass before the next step.
4. Review: Before merging into the main project, the project owner or scientific staff reviews the implementation.
5. Continuous monitoring and maintenance: Even in an iterative model, continuous monitoring and maintenance is required. Manual effort can be minimized by using automated processes such as dependency scanners and regular automated testing.

4.5.2 Data Management and Monitoring

Secure, maintainable, and efficient data management is critical, especially when storing sensitive data. While all the applications process children's data, requiring strict data protection and privacy measures, the project itself suffers from significant weaknesses in key and data management. Combined with outdated modules and missing migrations, the secure implementation of new secure data storage is not guaranteed. Furthermore, it is necessary to ensure that actual data breaches are detected in order to comply with the legal requirements of the GDPR in the event of a data leak. Therefore, the following steps are necessary to address these issues:

1. Data Classification: First, existing data must be classified based on sensitivity. Data that allows conclusions to be drawn about mental health and learning progress must be classified as highly sensitive and should be subject to strict access controls.
2. Data storage and handling: The project must ensure that sensitive data is processed and stored according to the latest best practices. This includes using the latest modules and storing sensitive data using the latest encryption methods.
3. Access controls: Strict and effective access control mechanisms must be implemented to ensure that only authorized individuals have access to sensitive data. A consistent process must be documented for all applications.
4. Error Tracking Tools: Ensure that potential data breaches are detected and responded to in accordance with regulatory requirements.
5. Security Policy: A comprehensive security policy must be documented to ensure that all important security aspects are considered at every stage of development.

4.5.3 Dependency Management

The failure of the attempted "OAuth 2.0" implementation revealed the importance of a robust dependency management system. Without up-to date dependencies, it is extremely challenging to develop secure and reliable software. As a result, the following lessons were learned:

1. Automated updates: The use of automated dependency management tools can facilitate regular checking of dependencies and ensure that the project is using the latest versions of modules.
2. Adequate testing: Testing is an essential dependency for updating modules. Without testing, it is impossible to guarantee that an update will not introduce functional and security problems. Therefore, test-driven development must be implemented in the secure software development cycle.
3. Maintainability: All dependencies must be reviewed regularly to replace unmaintained modules and remove unused ones, thereby increasing maintainability and simplifying workflow.

5 Discussion

The last two sections provided a technical view of the security of the "TU Graz Learning Lab". First, a security analysis was performed, and then a case study of a practical implementation of a security-related protocol was attempted. This researched information can not only provide a valuable insight into the "TU Graz Learning Lab", but also information for other small university projects with a similar structure can be discussed.

5.1 Interpretation of Findings

The hypothesis and research questions are used and answered to interpret the findings of this thesis.

5.1.1 Hypotheses

1. Due to long-term development, mostly conducted from students with different levels of security awareness and the fact that no active security audit was ever conducted, it is likely to find security vulnerabilities in the applications.

As can be seen from the analysis and the case study, the number of vulnerabilities found definitely exceeded expectations. The lack of security-related analysis in the project has not only allowed many vulnerabilities to persist unnoticed, but also meant that security-related processes were never implemented in the project. The evolving project design over such a long period of development in a production environment is not an appropriate process for developing secure software. This allows vulnerabilities to accumulate, whether introduced by experienced or inexperienced student developers.

2. These vulnerabilities are primarily due to insufficient security knowledge among student developers and the lack of a secure software development cycle, rather than intentional negligence.

This hypothesis is partially correct. While the lack of a secure development cycle has certainly led to security vulnerabilities, the students' lack of security awareness contributed at most to their failure to implement security processes on their own initiative. The relatively short time that the student developers spent on this project makes it practically impossible to think of all the areas that play a role in software development.

This leads to functional dominated programming where security is no longer consciously integrated. Mitigating these problems can only be done externally, with specific policies and documentation that force the developer to consider security at every stage. This would ensure a secure software development cycle.

3. The current architectural setup, with the "SOAP" based interface for the applications, can be improved through the introduction of the "OAuth 2.0" protocol.

This hypothesis cannot be conclusively resolved in this thesis, but there are numerous indications suggesting potential improvement. As observed in the case study, the implementation of "OAuth 2.0" prompts developers to scrutinize processes within the project. In order to implement "OAuth 2.0", secure key distribution is essential and issues like insecure session storage become evident. Moreover, a secure implementation of this protocol is unattainable without addressing existing problems, as illustrated by the case study. Thus, at least from a security perspective, it can be indirectly posited that a correct implementation of "OAuth 2.0" would improve the architecture simply by virtue of its introduction.

5.1.2 Research Questions

1. How secure are web learning applications, which were developed by TU Graz students within their bachelor or master thesis, from an information technical point of view?

The security of the "TU Graz Learning Lab", which was developed by TU Graz students, is at high risk on all levels, as shown in the security analysis and in the case study. From the numerous errors starting with the static analysis, which shows that the codebase of the project has serious structural issues, to the large amount of vulnerabilities found in OWASP ZAP, which mostly indicate missing configuration and not performed out updates, to the OWASP ASVS requirements, where more than half were not fulfilled, to the manual security issues discovered and during the attempted implementation of the "OAuth 2.0" protocol. All these areas show that this learning applications developed by TU Graz students are not secure.

2. Which security flaws happened, how can they be fixed, and why have students implemented their learning applications insecurely?

The following security flaws were found and build the base for all other found vulnerabilities:

- Lack of Secure Development Life Cycle: Firstly the complex codebase, with no coding standard and with high alerts in the static analysis indicates the lack of a general software development cycle. The absence of security policies lead to misconfigurations in security relevant parts of the project. These are the reasons why vulnerabilities like "Content Security Policy (CSP) Header Not Set" and "Cross-Site Request Forgery" were discovered, and in fact this point can be associated with all flaws.
- Insufficient Data Management: The structure of the database, the initialization via SQL dumps in combination with outdated "doctrine" notations, show that no active plan for data management was conducted. Furthermore, the security flaws in authorization, session management and secure storage mechanisms further confirm this point. Again, this is just a consequence of the lack of a security policy or a secure software development cycle.
- Outdated Modules: Dependencies not only cause problems in the maintainability of the project, but also have a direct impact on security vulnerabilities such as the XSS vulnerabilities. Again, this is related to the secure software development cycle, which includes a strategy for keeping modules up-to-date.

- Inadequate Testing and Monitoring: Test-driven development not only ensures functionality and its verifiability when reviewing the newly developed code, but is also essential for further modifications and extensions of the code base. Without it, it is virtually impossible to estimate the impact of changes and updates because they are based only on manual testing. Furthermore, the lack of continuous monitoring, including regular static and dynamic testing of the codebase, allows vulnerabilities to persist undetected, and even allows attackers to grab data undetected.

The reason for the security flaws can certainly be attributed to the lack of essential software development processes. In combination with the emerging system design, students did not see the real problems and only focused on functionality. Since the effort to maintain such projects is constantly growing, even the scientific staff cannot guarantee the security of the implementation without errors during the reviews.

Fixing the security flaws at this stage will now cost an immense amount of effort. Basically, the whole project needs to be refactored and, more importantly, secure software development processes need to be implemented so that such flaws are less likely to occur in the future. Therefore, it is essential to implement a secure software development cycle and start refactoring and fixing the vulnerabilities, guided by the prioritized results of the security analysis.

3. Which mitigation actions could be taken to avoid security vulnerabilities in student developed learning applications within their studies in the future?

In the future, it will be essential for students to introduce software development processes from the very beginning of a project. Security must be an issue that is considered at every stage of the development process. The earlier security and design flaws are discovered, the easier they are to fix.

Furthermore, for projects like the "TU Graz Learning Lab", which will continue over a long period of time, it is essential to implement a secure product lifecycle. All requirements must be clear at every stage of the project, and project owners must ensure that all necessary tasks are performed on an ongoing basis. This is especially important for web applications, which are constantly available over the Internet and require regular maintenance. As noted in Chapter 3, security is not a final product, but must be present throughout the product life cycle.

5.2 Comparison with Existing Literature

As stated in Chapter 2, there exists hardly any scientific literature, which brings student development into the context of a security analysis.

The scientific paper by Licorish et al. (2022), entitled "Understanding students' software development projects: Effort, performance, satisfaction, skills and their relation to the adequacy of outcomes developed" examines practical software development courses, often team-based. While this paper does not research actual real-world software development courses, it is possible to compare the general findings of the paper with this thesis. The biggest problem found in the paper is a lack of technical skills. This indicates that not only TU Graz students, have potential for improvement on technical skills, but also students from other universities. This is not surprising, since computer science-related studies offer a very broad spectrum and it is not possible to master technical skills from all areas. This makes it all the more important to give developers guidance on what is important, especially in areas like security, where a bad implementation not only jeopardizes a grade in a subject, but also has consequences for the whole project and, in our case, for the privacy of minors. Furthermore, this work also finds a need to catch up in project management. This also confirms the importance of development processes in student-developed applications.

Furthermore, Missiroli, Russo, and Ciancarini (2017) published the scientific paper entitled "Agile for millennials: a comparative study", which compares the two software methodologies "Scrum" and "Waterfall" for millennial high school students. The results of the paper show no significant preference of these two, but the recommendation is to use at least one of them in a software development life cycle. The lack of a software development life cycle can also be seen in this thesis.

The paper titled "Implementation of a Security Strengthening System to Mitigate Vulnerabilities in Academic Web Applications" from Márquez et al. (2023) uses virtually the same OWASP ZAP tools as used in this thesis to find vulnerabilities in academic web applications. The results show that a workflow using OWASP ZAP can reduce vulnerabilities in academic web applications.

5.3 Impact of Student-Developed Context

This thesis shows that, especially in the context of student development, it is not possible to create secure web applications without sufficient security assurance processes. Students, like all humans, will never be error free. Especially in the student context, where lack of technical skills, high staff turnover and insufficient independent process management can lead to insecure applications, it is all the more important to implement a secure software development cycle from the very beginning.

Moreover, this shows that other student-developed web applications than the "TU Graz Learning Lab", which also do not use development cycles, are very likely to have security vulnerabilities. Especially if the projects exist for a similar length of time. This could be a topic for further research.

6 Outlook

The analysis of the "TU Graz Learning Lab" is performed on a real-world student-developed project, and it has shown that student-developed web applications have a significant potential for security flaws and vulnerabilities. While this thesis provides a deep insight into the project and the development practices, it cannot provide all the answers to the questions arising from the security analysis and from the case study. Therefore, this chapter addresses the areas where further research is needed to improve the security of the "TU Graz Learning Lab".

6.1 Student Context: Key Findings

The codebase of the "TU Graz Learning Lab" has serious maintainability problems due to high complexity in relation to the actual functionality. This is due to a lack of development structure and processes that ensure the implementation of secure code. The hands-off development, which has been practiced for years because students usually spend only a short time on the project, has led to the lack of these processes. In addition, these processes were not as widespread at the start of the project as they are today, and are best implemented, when introduced at the beginning of a software development project.

In addition, the lack of test-driven development resulted in insufficient test cases needed to ensure the maintainability of the project with acceptable resources. This, in turn, resulted in poorly updated module dependencies, which led to maintainability issues and security vulnerabilities. Consequently, the codebase became harder to manage and troubleshoot, exacerbating security concerns and creating an environment conducive to the emergence and persistence of unnoticed vulnerabilities.

The absence of a robust security policy became a significant problem, leading to the existence of multiple potential attack vectors within the system. This neglect is further evidenced by the flawed authorization mechanisms and insecure session management that is present in the applications. Additionally, the disregard for secure data management is evident in the weak data storage security practices implemented.

6.2 Mitigating Security Vulnerabilities

The key findings identified in the "TU Graz Learning Lab" provide numerous areas ripe for further research. Addressing these issues requires a holistic approach that encompasses various facets of secure software development. These aspects can be explored during implementation in a student development environment, providing valuable insights for projects with severely limited human resources.

Future studies could explore the impact of implementing a secure software development lifecycle in a student development context. Therefore, the proposed workflow described in Section 4.5.1 can be used as a starting point. These studies can then observe how peer review and code audit practices affect code quality and provide insights into their effectiveness and ways to optimize them for the student development context.

Another fertile ground for investigation is the role of test-driven development in student-developed applications. The impact on overall code quality, dependency update cycle, functional bugs and security vulnerabilities can be observed and analyzed. This can potentially lead to the development of new test-driven development processes specifically adapted to student projects. Furthermore, the latest research on Artificial intelligence (AI) generated unit tests can also be considered (Fontes et al., 2023).

The development and implementation of security policies in student development environments is also an area worthy of investigation. A variety of security policies can be explored that affect the development of authorization mechanisms, session management, and secure data storage. Understanding the common pitfalls can help to create new security policies and best practices in the student development context.

Each of these areas is not only necessary to achieve an acceptable security state of the "TU Graz Learning Lab", but can also contribute to how software development is taught and practiced in educational settings. Ongoing efforts and regular re-evaluation of these issues are necessary to evolve security practices in the context of the applications developed by students.

6.3 OAuth 2.0 in TU Graz Learning Lab

The case study of this thesis shows that the implementation of application level security protocols in student-developed learning applications, which have complex legacy issues and outdated dependencies, leads to significant functional and security flaws. It also outlines existing problems during development. Therefore, the attempted implementation provides a rich source of insights and lessons learned for future studies in the area of secure authorization and authentication protocol implementation in the "TU Graz Learning Lab" and other similar projects.

Future studies could look at how implementing "OAuth 2.0" in a project with resolved legacy issues can affect overall security, and in particular how this affects authorization mechanisms. Furthermore, the implementation of "OpenIDConnect" for authentication is also a big topic of potential research, as this thesis could only address it due to the failed implementation. In addition, the implementation of the "LTI Core Specification" can also be used to gain knowledge about the authentication of applications in an educational setting, as this topic could also only be mentioned by this thesis.

In conclusion, this potential research could solve significant problems in the applications of "TU Graz Learning Lab". Furthermore, it can help to introduce the latest best practices in learning applications, such as standardized authentication methods, which provide a more secure workflow and allow the application to be more modular and connect to other standardized educational applications. In general, this research can provide benefits to a broader field of secure software development in an educational setting.

6.4 Security Education & Practice

The state of the "TU Graz Learning Lab" shows that current education and practice need significant improvement, especially in the areas of software engineering techniques. Further research can explore effective ways to teach secure coding practices among students. Therefore, the current pedagogical methods need to be reviewed in the context of TU Graz. Then the effectiveness of the different methods can be evaluated.

As described in Section 5.3, the responsibility for security in student software development projects does not lie solely with the developers. The case of the "TU Graz Learning Lab" illustrates the potential security risks associated with the loosely structured, hands-off development approaches commonly used in student projects. Examining how different software development methodologies and practices can help identify effective strategies for mitigating security risks in student projects. The human factor is often the weakest link in the security chain, and further study can help identify methods to foster a security-conscious culture among students.

By addressing these areas, future research can help shape security education and practice. Continuous efforts and regular re-evaluation of this topic are necessary to keep up with the ever-changing security risks in web applications, especially in the context of student-developed applications. This can not only ensure a more secure "TU Graz Learning Lab" project, but also influence student-developed applications in general.

6.5 Future Security Planning

Security planning, especially in the context of student-led projects, needs to be an integral part of the software development life cycle from the beginning. Therefore, security requirements must be defined in a comprehensive and integrated manner before the first line of code is written. This must include secure coding practices and automated security checks via CI/CD. The focus is on instilling an appreciation for security and providing practical tools and strategies in an educational context. This we will certainly lead to a more secure project than the current "TU Graz Learning Lab" and thus improve software security in general.

6.6 Future of the TU Graz Learning Lab

In light of the extensive security issues, structural challenges, and dependency management concerns identified in the "TU Graz Learning Lab," determining an optimal course of action to address and further develop the project is essential. The two main strategies under consideration are: Refactoring the current code or undertaking a comprehensive rewrite (Fairbanks, 2019).

6.6.1 Refactor

Refactoring primarily emphasizes incremental improvements without changing the system's fundamental behavior. Its chief objective is to enhance the code's structure and maintainability. As discussed in this thesis, the "TU Graz Learning Lab" exhibits numerous challenges related to its structure and maintainability, primarily stemming from inadequate software development processes. Consequently, a refactoring effort should encompass not just the codebase's restructuring but also the establishment of a rigorous software development cycle, incorporating all the sub-areas highlighted in the security analysis and case study. Given the project's vast structural issues, lack of adequate testing, and dependency management problems, estimating the required effort accurately is challenging, with the expectation of it being considerable.

6.6.2 Rewrite

Conversely, a rewrite entails a total system overhaul from scratch. This approach is typically favored when the current structure presents significant constraints, or when the foundational domain model no longer resonates with contemporary needs. The "TU Graz Learning Lab" is encumbered by severe legacy issues. Addressing these concerns would be unnecessary in a rewrite, allowing for the planning and integration of a secure software development lifecycle from inception. This could mitigate the majority of identified challenges. However, given the project's moderate size, exceeding 77,000 lines of code, a rewrite may prove more time-intensive than a refactor. This is especially true if a streamlined implementation method accommodating the project's moderate functional complexity isn't identified.

7 Conclusion

The "TU Graz Learning Lab" project, comprised of various learning applications developed by TU Graz students, caters to primary pupils and classes. Given that the project involves data processing of children, it falls under the stringent protection of the GDPR. This thesis conducted an in-depth security analysis of the project, exposing various critical security issues and areas for improvement.

The static code analysis using "PhpMetrics" provided a comprehensive overview of the project's security landscape, revealing a high complexity of the code and outdated dependencies. The lack of an established system for maintaining up-to-date dependencies, or even ensuring consistent versions, was detected. This issue, stemming from non-implemented software development processes, led to serious maintainability issues, as evidenced by the large number of errors in "PhpMetrics". Additionally, the analysis highlighted a notable absence of coding standards.

The dynamic analysis conducted with OWASP ZAP uncovered numerous categories of vulnerabilities. These issues were linked to the outdated modules, lack of security policy, and insufficient testing. As the root cause of these problems lies in the absence of a secure software development cycle, implementing such a cycle, alongside refactoring the project's complex structure and rectifying the discovered flaws, is required for mitigation.

Further, the application of the OWASP ASVS indicated that over half of the "Level 1" requirements failed in "TU Graz Learning Lab". This result underlines the findings from the security analysis, emphasizing the lack of processes for secure software development cycles and security requirements.

The latter part of this thesis encompassed an attempt to implement "OAuth 2.0" in the existing setup without addressing the issues found in the security analysis, aiming to examine the influence of these issues on actual development. The endeavor was ultimately halted due to problems with the update process of the provisioned infrastructure, issues with data management in databases and session storage, all of which were caused by the dependency problems. The hurdles encountered in this implementation case study, notably with dependency management, data storage, and dealing with complexity and legacy issues, echoed the issues identified in the security analysis.

In sum, this work emphasizes that the implementation of a secure software development cycle is indispensable for a project like the "TU Graz Learning Lab", especially in a student-led context with high personnel rotation. It calls for the integration of secure software development practices into the education system, emphasizing their vital role in project maintainability and data security.

Bibliography

1EdTech (2021). *1EdTech Security Framework*. URL: https://www.imsglobal.org/1edtech-security-framework (visited on 05/24/2023) (cit. on p. 12).

Acar, Yasemin et al. (Sept. 2017). "Developers Need Support, Too: A Survey of Security Advice for Software Developers." In: *2017 IEEE Cybersecurity Development (SecDev)*. 2017 IEEE Cybersecurity Development (SecDev). Cambridge, MA, USA: IEEE, pp. 22–26. ISBN: 978-1-5386-3467-7. DOI: 10.1109/SecDev.2017.17. URL: http://ieeexplore.ieee.org/document/8077802/ (visited on 08/01/2023) (cit. on p. 27).

Acharya, Shivani, Vidhi Pandya, and I T Department (2012). "Bridge between Black Box and White Box – Gray Box Testing Technique." In: *International Journal of Electronics and Computer Science Engineering* 2.1, pp. 175–185. ISSN: 2277-1956 (cit. on p. 16).

Aldabbagh, Ghadah et al. (June 8, 2021). "Secure Data Exchange in M-Learning Platform using Adaptive Tunicate Slime-Mold-Based Hybrid Optimal Elliptic Curve Cryptography." In: *Applied Sciences* 11.12, p. 5316. ISSN: 2076-3417. DOI: 10.3390/app11125316. URL: https://www.mdpi.com/2076-3417/11/12/5316 (visited on 07/25/2023) (cit. on p. 11).

Belonozkin, Anton Valerevic and Aleksandr Aleksandrovic Rybanov (2016). "Proverka Kacestva Programmnogo Koda S Pomosju PHPMetrics [Quality Control of Software Code with PHPMetrics]." In: *NovaInfo. Ru* 4.47, pp. 8–14 (cit. on p. 18).

Belqasmi, Fatna et al. (2012). "SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing." In: *IEEE Internet Computing* 16.4, pp. 54–63. DOI: 10.1109/MIC.2012.62 (cit. on p. 7).

Bucher, Patrick P and Christopher J Christensen (2019). "Computer Science Hot Topics: OAuth 2." Hochschule Luzern. URL: https://raw.githubusercontent.com/patrickbucher/inf-stud-hslu/master/infkol/thesis/paper.pdf (visited on 08/01/2023) (cit. on p. 43).

Burazer, Marko (Sept. 2019). "Implementation of Interactive Learning Objects for German Language Acquisition in Primary School based on Learning Analytics Measurements." Co-Supervisor: Dipl.-Ing. Ebner Markus. Master's Thesis. Graz, Austria: Graz University of Technology (cit. on p. 6).

Daud, Malik Imran (July 2010). "Secure Software Development Model: A Guide for Secure Software Life Cycle." In: *Lecture Notes in Engineering and Computer Science* 2180 (cit. on p. 58).

Fairbanks, George (Mar. 2019). "Ignore, Refactor, or Rewrite." In: *IEEE Software* 36.2, pp. 133–136. ISSN: 0740-7459, 1937-4194. DOI: 10.1109/MS.2018.2880662. URL: https://ieeexplore.ieee.org/document/8648269/ (visited on 08/24/2023) (cit. on p. 71).

Fontes, Afonso et al. (July 2023). "Automated Support for Unit Test Generation." In: *Optimising the Software Development Process with Artificial Intelligence*. London: Springer, pp. 179–219. ISBN: 978-981-19-9947-5. DOI: 10.1007/978-981-19-9948-2_7 (cit. on p. 68).

Geier, Gerald (June 2015). "Adaptives Informationssystem zur Erlernung mehrstelliger Division." Master's Thesis. Graz, Austria: Technische Universität Graz (cit. on p. 6).

Germain, Éric and Pierre N. Robillard (Feb. 2005). "Engineering-based processes and agile methodologies for software development: a comparative case study." In: *Journal of Systems and Software* 75.1, pp. 17–27. ISSN: 01641212. DOI: 10.1016/j.jss.2004.02.022. URL: https://linkinghub.elsevier.com/retrieve/pii/S016412120400038X (visited on 07/18/2023) (cit. on p. 39).

Gerodimos, Apostolos et al. (2023). "IoT: Communication protocols and security threats." In: *Internet of Things and Cyber-Physical Systems* 3, pp. 1–13. ISSN: 26673452. DOI: 10.1016/j.iotcps.2022.12.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S2667345222000293 (visited on 07/25/2023) (cit. on p. 10).

Gill, G.K. and C.F. Kemerer (Dec. 1991). "Cyclomatic complexity density and software maintenance productivity." In: *IEEE Transactions on Software Engineering* 17.12. Conference Name: IEEE Transactions on Software Engineering, pp. 1284–1288. ISSN: 1939-3520. DOI: 10.1109/32.106988 (cit. on p. 17).

Hardt, Dick (Oct. 2012). *The OAuth 2.0 Authorization Framework*. RFC 6749. DOI: 10.17487/RFC6749. URL: https://www.rfc-editor.org/info/rfc6749 (cit. on p. 42).

IMS Global Learning Consortium (2019). *Learning Tools Interoperability Core Specification 1.3*. URL: https://www.imsglobal.org/spec/lti/v1p3/ (visited on 07/26/2023) (cit. on p. 12).

Internet Engineering Task Force (IETF) (2023). *OAuth 2.0 — OAuth*. URL: https://oauth.net/2/ (visited on 07/25/2023) (cit. on p. 12).

Jureczko, Marian and Michal Mlynarski (2010). "Automated acceptance testing tools for web applications using Test-Driven Development." In: *PRZEGLĄD ELEKTROTECHNICZNY (Electrical Review)* 86.9, pp. 198–202. ISSN: 0033-2097 (cit. on p. 26).

KirstenS et al. (2023). *Cross Site Scripting (XSS)*. URL: https://owasp.org/www-community/attacks/xss/ (visited on 07/15/2023) (cit. on p. 29).

Koch, Christoph (Mar. 2020). "Overhaul of the Lesetrainer's Website and API." Institute of Interactive Systems and Data Science. Head: Univ.-Prof. Dipl-Inf. Dr. Stefanie Lindstaedt. Bachelor's Thesis. Graz, Austria: Graz University of Technology (cit. on p. 6).

Kraja, Eltion (Nov. 2016). "Die Multiplikationstabelle als innovative Learning-Analytics-Applikation." Master's Thesis. Graz, Austria: Technische Universität Graz (cit. on p. 6).

Kropp, Martin and Andreas Meier (2015). "Swiss Agile Study 2014." In: Publisher: FHNW and ZHAW. ISSN: 2296-2476. DOI: 10.13140/RG.2.1.3372.3045. URL: http://rgdoi.net/10.13140/RG.2.1.3372.3045 (visited on 06/16/2023) (cit. on p. 55).

Licorish, Sherlock A. et al. (Apr. 2022). "Understanding students' software development projects: Effort, performance, satisfaction, skills and their relation to the adequacy of outcomes developed." In: *Journal of Systems and Software* 186, p. 111156. ISSN: 01641212. DOI: 10.1016/j.jss.2021.111156. URL: https://linkinghub.elsevier.com/retrieve/pii/S0164121221002466 (visited on 07/24/2023) (cit. on p. 65).

Linux Foundation (2023). *Laminas Project*. URL: https://getlaminas.org/ (visited on 07/26/2023) (cit. on p. 13).

M., Brunil D. Romero, Hisham M. Haddad, and Jorge E. Molero A. (Sept. 2009). "A Methodological Tool for Asset Identification in Web Applications: Security Risk Assessment." In: *2009 Fourth International Conference on Software Engineering Advances*. 2009 Fourth International Conference on

Software Engineering Advances, pp. 413–418. DOI: 10.1109/ICSEA.2009.66 (cit. on p. 14).

Makino, Yuma and Vitaly Klyuev (Sept. 2015). "Evaluation of web vulnerability scanners." In: *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*. Warsaw, Poland: IEEE, pp. 399–402. ISBN: 978-1-4673-8359-2. DOI: 10.1109/IDAACS.2015.7340766. URL: http://ieeexplore.ieee.org/document/7340766/ (visited on 07/26/2023) (cit. on p. 13).

Manjunath, M (Aug. 22, 2018). *How Secure Are Your JavaScript Open-Source Dependencies?* Code Envato Tuts+. URL: https://code.tutsplus.com/articles/how-secure-are-your-javascript-open-source-dependencies--cms-31685 (visited on 05/24/2023) (cit. on p. 28).

Márquez, Ing Jimmy Fernando Ramírez et al. (Apr. 30, 2023). "Implementation of a Security Strengthening System to Mitigate Vulnerabilities in Academic Web Applications." In: *Journal of Survey in Fisheries Sciences* 10.3. Number: 3S, pp. 5447–5456. ISSN: 2368-7487. URL: http://sifisheriessciences.com/journal/index.php/journal/article/view/1837 (visited on 05/24/2023) (cit. on p. 66).

Martin, Robert C. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Pearson Education Inc. ISBN: 978-0-13-235088-4 (cit. on p. 17).

Mcgraw, G. (Mar. 2004). "Software security." In: *IEEE Security & Privacy Magazine* 2.2, pp. 80–83. ISSN: 1540-7993. DOI: 10.1109/MSECP.2004.1281254. URL: http://ieeexplore.ieee.org/document/1281254/ (visited on 07/31/2023) (cit. on p. 14).

Missiroli, Marcello, Daniel Russo, and Paolo Ciancarini (May 2017). "Agile for Millennials: A Comparative Study." In: *2017 IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM)*. 2017 IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM). Buenos Aires, Argentina: IEEE, pp. 47–53. ISBN: 978-1-5386-2795-2. DOI: 10.1109/SECM.2017.7. URL: http://ieeexplore.ieee.org/document/7964622/ (visited on 07/18/2023) (cit. on p. 65).

Naiakshina, Alena et al. (Oct. 30, 2017). "Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study." In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 311–328. DOI: 10.1145/3133956.3134082. arXiv: 1708.08759[cs].

URL: http://arxiv.org/abs/1708.08759 (visited on 07/20/2023) (cit. on p. 57).

Neuhold, Benedikt (July 2013). "Adaptives Informationssystem zur Erlernung mehrstelliger Addition und Subtraktion." Institutsleiter: Univ.-Prof. Dipl.-Ing. Dr.techn. Frank Kappe. Diplomarbeit. Graz, Austria: Technische Universität Graz (cit. on p. 6).

OWASP (2021). *Application Security Verification Standard 4.3*. Version 4.3. URL: https://raw.githubusercontent.com/OWASP/ASVS/v4.0.3/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-en.pdf (visited on 07/04/2023) (cit. on pp. 15, 37).

OWASP Foundation (2021a). *OWASP Application Security Verification Standard*. URL: https://owasp.org/www-project-application-security-verification-standard/ (visited on 07/25/2023) (cit. on p. 11).

OWASP Foundation (2021b). *OWASP Top Ten*. URL: https://owasp.org/www-project-top-ten/ (visited on 07/25/2023) (cit. on p. 11).

OWASP Foundation (2023). *About the OWASP Foundation | OWASP Foundation*. URL: https://owasp.org/about/ (visited on 07/25/2023) (cit. on p. 11).

Proenca, Tiago, Nilmax Teones Moura, and André van der Hoek (2010). "On the Use of Emerging Design as a Basis for Knowledge Collaboration." In: *New Frontiers in Artificial Intelligence*. Ed. by Kumiyo Nakakoji, Yohei Murakami, and Elin McCready. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 124–134. ISBN: 978-3-642-14888-0 (cit. on p. 4).

Rahman, Akond, Effat Farhana, and Laurie Williams (Sept. 2020). "The 'as Code' Activities: Development Anti-patterns for Infrastructure as Code." In: *Empirical Software Engineering* 25.5, pp. 3430–3467. ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-020-09841-8. arXiv: 2006.00177[cs]. URL: http://arxiv.org/abs/2006.00177 (visited on 06/06/2023) (cit. on p. 47).

Ricca, F. and P. Tonella (May 2001). "Analysis and testing of Web applications." In: *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*. Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001. ISSN: 0270-5257, pp. 25–34. DOI: 10.1109/ICSE.2001.919078 (cit. on p. 12).

Rowe, Frantz, Richard Baskerville, and Francois-Charles Wolff (2012). "Functionality vs. Security in IS: Tradeoff or Equilibrium?" In: *Proceedings of* *the Thirty-Third International Conference on Information Systems*. Orlando, Florida, USA (cit. on p. 41).

Scarfone, Karen, Daniel Benigni, and Timothy Grance (June 15, 2009). "Cyber Security Standards." en. In: *Wiley Handbook of Science and Technology for Homeland Security*. Hoboken, NJ: John Wiley & Sons, Inc. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=152153 (cit. on p. 10).

Schellander, Stefan (Mar. 2022). "Neuentwicklung der Webapp „Buchstabenpost"." Bachelor's Thesis. Graz, Austria: Technische Universität Graz (cit. on p. 6).

Steyrer, Michael (Aug. 2012). "Adaptives Informationssystem für Mathematische Lernanwendungen." German. Master's thesis. Technische Universität Graz. URL: https://diglib.tugraz.at/download.php?id=576a76e493e80&location=browse (cit. on pp. 4, 6).

Technische Universität Graz (2023a). *1×1 Trainer*. URL: https://schule.learninglab.tugraz.at/einmaleins/ (visited on 05/30/2023) (cit. on p. 3).

Technische Universität Graz (2023b). *Buchstabenpost*. URL: https://schule.learninglab.tugraz.at/buchstabenpost/ (visited on 05/30/2023) (cit. on p. 3).

Technische Universität Graz (2023c). *Divisiontrainer*. URL: https://schule.learninglab.tugraz.at/divisionstrainer/ (visited on 05/30/2023) (cit. on p. 3).

Technische Universität Graz (2023d). *IDeRBlog Exercises*. URL: https://schule.learninglab.tugraz.at/iderblogexercises/ (visited on 05/30/2023) (cit. on p. 3).

Technische Universität Graz (2023e). *Junior Plus-Minus-Trainer*. URL: https://schule.learninglab.tugraz.at/juniorplusminus/ (visited on 05/30/2023) (cit. on p. 3).

Technische Universität Graz (2023f). *Learning Lab*. URL: http://schule.learninglab.tugraz.at/ (visited on 05/30/2023) (cit. on p. 2).

Technische Universität Graz (2023g). *Mathe-Multi-Trainer*. URL: https://schule.learninglab.tugraz.at/multitrainer/ (visited on 05/30/2023) (cit. on p. 3).

Technische Universität Graz (2023h). *Plus-Minus-Trainer*. URL: https://schule.learninglab.tugraz.at/plusminus/ (visited on 05/30/2023) (cit. on p. 3).

Technische Universität Graz (2023i). *Privacy Declaration*. URL: https://schule.learninglab.tugraz.at/privacy (visited on 05/30/2023) (cit. on p. 2).

Teodoro, Nuno and Carlos Serrao (June 2011). "Web application security: Improving critical web-based applications quality through in-depth security analysis." In: *2011 International Conference on Information Society*. London: IEEE, pp. 457–462. ISBN: 978-1-61284-148-9. DOI: 10.1109/i-Society18435.2011.5978496. URL: https://ieeexplore.ieee.org/document/5978496/ (visited on 06/20/2023) (cit. on p. 14).

Tobias, Eric (Nov. 3, 2020). *Zerologon: Why You Should Never "Roll Your Own" Cryptography*. Ubiq. URL: https://www.ubiqsecurity.com/never-roll-your-own-cryptography/ (visited on 06/14/2023) (cit. on p. 49).

Walden, James, Jeff Stuckman, and Riccardo Scandariato (Nov. 2014). "Predicting Vulnerable Components: Software Metrics vs Text Mining." In: *2014 IEEE 25th International Symposium on Software Reliability Engineering*. Naples, Italy: IEEE, pp. 23–33. ISBN: 978-1-4799-6033-0. DOI: 10.1109/ISSRE.2014.32. URL: http://ieeexplore.ieee.org/document/6982351/ (visited on 07/26/2023) (cit. on p. 13).

Wang, Wentao et al. (May 2022). "Detecting Software Security Vulnerabilities Via Requirements Dependency Analysis." In: *IEEE Transactions on Software Engineering* 48.5. Conference Name: IEEE Transactions on Software Engineering, pp. 1665–1675. ISSN: 1939-3520. DOI: 10.1109/TSE.2020.3030745 (cit. on pp. 19, 39).

Włodarski, Rafal, Aneta Poniszewska-Marańda, and Jean-Remy Falleri (Apr. 2022). "Impact of software development processes on the outcomes of student computing projects: A tale of two universities." In: *Information and Software Technology* 144, p. 106787. ISSN: 09505849. DOI: 10.1016/j.infsof.2021.106787. URL: https://linkinghub.elsevier.com/retrieve/pii/S0950584921002287 (visited on 08/01/2023) (cit. on p. 38).

Yadav, Divyani et al. (Dec. 2018). "Vulnerabilities and Security of Web Applications." In: *2018 4th International Conference on Computing Communication and Automation (ICCCA)*. 2018 4th International Conference on Computing Communication and Automation (ICCCA). Greater Noida, India: IEEE, pp. 1–5. ISBN: 978-1-5386-6947-1. DOI: 10.1109/CCAA.2018.8777558. URL: https://ieeexplore.ieee.org/document/8777558/ (visited on 07/25/2023) (cit. on p. 10).

Zandstra, Matt (2016). “Testing with PHPUnit.” In: *PHP Objects, Patterns, and Practice*. Berkeley, CA: Apress, pp. 435–464. ISBN: 978-1-4842-1996-6. DOI: 10.1007/978-1-4842-1996-6_18. URL: https://doi.org/10.1007/978-1-4842-1996-6_18 (cit. on p. 18).