

Tvorba bezpečné webové aplikace

Maroš Kopas

Bakalářská práce
2023



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Maroš Kopas**
Osobní číslo: **A20009**
Studijní program: **B0613A140020 Softwarové inženýrství**
Forma studia: **Prezenční**
Téma práce: **Tvorba bezpečné webové aplikace**
Téma práce anglicky: **Creation of a Secure Web Application**

Zásady pro vypracování

1. Nastudujte a popište problematiku spojenou s tvorbou bezpečných webových aplikací.
2. Zvolte vhodné vývojové technologie pro naplnění zadání práce.
3. Uveďte nejčastější bezpečnostní problémy při implementaci webových aplikací.
4. Demonstrujte vybrané problémy ve formě ukázkové aplikace.
5. Navrhněte a implementujte protiopatření.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Java Script documentation. Devdocs[online]. [cit. 2022-09-21]. Dostupné z: <https://devdocs.io/javascript/>
2. Orange Juice Shop Project[online]. [cit. 2022-09-21]. Dostupné z: <https://owasp.org/www-project-juice-shop/>
3. Cloudflare. Web Security[online]. [cit. 2022-09-21]. Dostupné z: <https://www.cloudflare.com/learning/security/what-is-web-application-security/>
4. Owasp. Owasp Top 10[online]. [cit. 2022-09-21]. Dostupné z: <https://owasp.org/www-project-top-ten/>
5. Angular Docs[online]. [cit. 2022-10-18]. Dostupné z: <https://angular.io/docs>
6. BASTA, Alfred a Melissa ZGOLA.Database Security.

Vedoucí bakalářské práce: **Ing. Petr Žáček, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **2. prosince 2022**

Termín odevzdání bakalářské práce: **26. května 2023**



doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Maroš Kopas, v.r.

.....
podpis studenta

ABSTRACT

The rapid growth of web-based applications has led to an increased need for effective security measures to protect users' data and privacy. This bachelor's thesis aims to explore various web application security techniques, including authentication and access control, data encryption, database security and protection, and many others. The study will analyze the effectiveness of these techniques and identify potential vulnerabilities in web applications. Furthermore, the thesis will propose practical solutions to enhance the security of web applications.

Finally, this research seeks to educate programmers on the importance of secure web application making and their correct implementation.

Keywords: Web security, Educational application, Vulnerability prevention

ABSTRAKT

Rýchly rast webových aplikácií viedol k zvýšenej potrebe účinných bezpečnostných opatrení na ochranu údajov a súkromia používateľov. Cieľom tejto bakalárskej práce je preskúmať rôzne techniky zabezpečenia a bezpečnosti webových aplikácií vrátane autentifikácie a riadenia prístupu, šifrovania údajov, bezpečnosti a ochrany databáz a mnohých ďalších. Štúdia bude analyzovať účinnosť týchto techník a identifikovať potenciálne zraniteľnosti webových aplikácií. Okrem toho práca navrhuje praktické riešenia na zvýšenie bezpečnosti a ochrany webových aplikácií.

V konečnom dôsledku je cieľom tohto výskumu vzdelávať programátorov o dôležitosti tvorby bezpečných a zabezpečených webových aplikácií a ich správnej implementácie.

Kľúčové slová: Webová bezpečnosť, vzdelávacia aplikácia, prevencia zraniteľností

Poděkování, motto a čestné prohlášení, že odevzdaná verze bakalářské práce a verze elektronická, nahraná do IS/STAG jsou totožné ve znění:

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

CONTENT

INTRODUCTION	8
I. THEORETICAL PART	9
1 EXAMANATION OF SECURITY ISSUES IN WEB APPLICATIONS	10
1.1 OVERVIEW OF COMMON WEB APPLICATION ISSUES	10
1.2 COMMON SECURITY ISSUES.....	10
1.3 ANALYSIS OF SECURITY BREACH CONSEQUENCES	11
1.4 IMPORTANCE OF CORRECT SECURITY	11
1.5 TESTING APPLICATIONS SECURITY	11
1.6 SECURITY FROM THE POINT OF FRONT END.....	11
1.7 SECURITY FROM THE POINT OF BACK END.....	12
1.8 SECURITY FROM THE POINT OF NETWORKING	12
2 CHOSEN TOOLS FOR DEVELOPMENT.....	13
2.1 OVERVIEW OF AVAILABLE TECHNOLOGIES	13
2.2 RANKING THE TECHNOLOGIES.....	13
2.3 CHOOSING CORRECT TECHNOLOGIES.....	14
2.4 CRITERIA FOR CHOOSING THE CORRECT TECHNOLOGIES	14
3 COMMON SECURITY ISSUES	15
3.1 BROKEN ACCESS CONTROL	15
3.2 CRYPTOGRAPHIC FAILURES	16
3.3 INJECTION	17
3.4 INSECURE DESIGN	18
3.5 SECURITY MISCONFIGURATION.....	19
3.6 VULNERABLE AND OUTDATED COMPONENTS	20
3.7 IDENTIFICATION AND AUTHENTICATION FAILURES	21
3.8 SOFTWARE AND DATA INTEGRITY FAILURES	22
3.9 SECURITY LOGGING AND MONITORING FAILURES.....	22
3.10 SERVER-SIDE REQUEST FORGERY (SSRF)	23
4 DESIGN	24
4.1 UI DESIGN	24
4.2 FUNCTIONAL DESIGN.....	24
4.3 TECHNICAL REQUIREMENTS AND LIMITATIONS	24
II. PRACTICAL PART	26
5 DEMONSTRATION OF SECURITY ISSUES IN THE PROJECT.....	27
5.1 WEBSITE DESCRIPTION.....	27
5.2 DEMONSTRATION OF EACH ISSUE	27
5.2.1 <i>Broken Access Control</i>	28
5.2.2 <i>Injections</i>	29
5.2.3 <i>Cryptographic Failures</i>	30
5.2.4 <i>Identification and Authentication Failures</i>	30

5.2.5	<i>Vulnerable and Outdated Components</i>	31
6	COUNTERMEASURES IMPLEMENTATION	33
6.1	OVERVIEW OF AVAILABLE MEASURES	33
6.1.1	<i>Broken Access Control</i>	33
6.1.2	<i>SQL Injections</i>	33
6.1.3	<i>Cryptographic failures</i>	34
6.1.4	<i>Identification and Authentication Failures</i>	34
6.1.5	<i>Vulnerable and Outdated Components</i>	35
6.2	SUMMARY OF AVAILABLE MEASURES	35
6.3	IMPLEMENTATION OF SECURITY MEASURES	36
6.3.1	<i>Broken Access Prevention</i>	36
6.3.2	<i>SQL Injections</i>	39
6.3.3	<i>Cryptographic Failures</i>	40
6.3.4	<i>Identification and Authentication Failures</i>	42
6.3.5	<i>Vulnerable and Outdated Components</i>	43
6.4	EFFICIENCY OF USED COUNTERMEASURES	43
7	EXAMPLES OF IMPLEMENTATION	44
7.1	GENERAL OVERVIEW	44
7.2	IMPLEMENTING EXAMPLES	44
7.2.1	<i>Broken Access Control</i>	44
7.2.2	<i>SQL Injections</i>	45
7.2.3	<i>Cryptographic Failures</i>	47
7.2.4	<i>Identification and Authentication Failures</i>	48
7.2.5	<i>Vulnerable and Outdated Components</i>	50
8	CONCLUSION	51
9	REFERENCES	52
10	LIST OF PICTURES	58

INTRODUCTION

The motivation for this theme was self-improvement. This work is meant to broaden my general programming knowledge and better prepare me for the working environment. And at the same time, it serves as a tool for beginner programmers to make better and more secure web applications.

One of the points is to create guidelines for people who have never made web applications. It should serve as a learning tool to prevent the worst of mistakes that could have been easily avoided or to spread awareness of the consequences it could have. As for the scale, I decided to pick a handful of the most common and, in my opinion, most severe issues and how to solve them, so my work will not show every possible security issue of web app making but just the most important ones. That is also done because some OWASP categories combine multiple previous ones or are generally aimed at practices rather than code implementations.

On the world scale, cyber security is a well-studied and broad informatics discipline. The Internet provides many solutions to most of the known web security issues. Libraries and frameworks already include countermeasures to many, if not most, common vulnerabilities.

The theoretical part is structured to explain the most common security issues based on the OWASP organization. It explains the importance of securing against all kinds of security threats and the consequences if a programmer fails to do so. Each vulnerability is described in detail, along with a few examples.

The practical part of this thesis is a web application. This application serves as a learning tool for above mention security vulnerabilities. The application includes five of the ten OWASP categories. Each category is explained along with examples and countermeasures. Parts of the website have purposefully made vulnerabilities that the users can attack.

I. THEORETICAL PART

1 EXAMINATION OF SECURITY ISSUES IN WEB APPLICATIONS

This section describes general issues with application security, lists the most common security issues, and outlines their dangers and consequences, and testing to prevent mentioned issues.

1.1 Overview of common web application issues

Web-based applications can suffer from many issues, not only security ones. Some are only convenience and user experience issues, while others might affect the user's security and data. Even the smallest of issues and inconveniences may turn users away. That's why we need to be extra careful not to miss any. We could categorize these issues as security, UI, site readability, performance, dependence on third-party services, network, and connectivity issues. [1]

1.2 Common security issues

Here belong the most common security issues. These are well-known issues, with many software explicitly developed to exploit them. Some are harmless, while others can lead to vast leaks of a user, personal or other kinds of data. [2] [3]

Some of these problems are:

- Broken Access Control
- Cryptographic Failures
- Injections
- Insecure Design
- Security misconfiguration
- Vulnerable and outdated components
- Identification and Authentication Failures
- Software and Data Integrity Failures
- Security Logging and Monitoring Failures
- Server-Side request forgery (SSRF)

1.3 Analysis of security breach consequences

Protecting our applications is extremely important. We don't want to leave any back-doors that could be exploited. Obviously, this doesn't apply to all web applications, we will focus on the ones that gather or work with user, company, or any other kinds of data.

One consequence of a security breach that could, depending on the company's scale, be less or more harmful is Reputational damage. In this scenario where no data loss occurred, a company loses face over things like service obstruction or similar issues. However, when it comes to data theft, more pressing matters arise. The reputation will be damaged, and monetary loss and lawsuits may also occur. To attackers, data is valuable since it can be sold. [4]

1.4 Importance of correct security

As mentioned above, we want to avoid breaches at all times. Since nobody wants monetary and reputational damage, we can see the importance of correct web app security. Not to mention the danger of leaking personal user data. Which could lead to lawsuits and even company disbandment. When such leaks happen to big and well-renowned companies, it may lead to public distrust and decreased number of users on your platform. This thesis describes how to avoid those mistakes in simple and effective manner.

1.5 Testing applications security

It is recommended to test applications before deploying them. Applications are generally tested during and after development. It may prevent leaving unsecured parts of the application. There are many methods for testing, such as Static Application Security Testing (SAST), which explores the application's inner workings, such as source code, or Dynamic application security testing (DAST) performs attacks on already running applications to find any unnoticed weaknesses and many other approaches. To help us categorize and better identify new attacks, organizations like OWASP are created (Common security issues). [5] [6]

1.6 Security from the point of front end

Front-end security, or client-side security, can be explained by common comparisons like leaving the front door to your house open. These attacks try to find any hole in website security and exploit them. A few of those vulnerabilities are unauthorized access, Cross-site scripting (XSS), DDos, and CSS injection attacks, among many others. Some common

tactics to prevent these attacks are validation and sanitization of user input, authentication, and authorization mechanisms, communication encryption, and many more. [7] [8]

1.7 Security from the point of back end

If we were to use the example with the house, while the front-end is the front door of your house, leaving the back-end unsecured is like leaving your safe open. Back-end or server-side attacks are aimed at the company's databases and servers, with the primary aim usually being personal data. There is, again, a wide range of vulnerabilities, such as data injection, lack of authentication, software misconfigurations, sensitive data exposure, and more. And even more solutions for these issues like implementing access control, encryption, multi-factor authentication, regular security audits, monitoring, etc. [9] [10]

1.8 Security from the point of networking

Another essential category, depending on your application type, is networking. This category might not be such a big issue in application-making since most modern frameworks already have measures in place, but still, one to consider and be wary of. As a developer, it is important to implement secure communication protocols, such as HTTPS, to encrypt data transmitted between the web application and users' browsers. This helps protect sensitive information, such as login credentials and personal data, from being intercepted or manipulated. Additionally, implementing proper input validation and sanitization techniques on the server side helps prevent common attacks like SQL injection and cross-site scripting (XSS). Keeping the underlying server infrastructure and network devices up to date with security patches is also essential to prevent vulnerabilities that attackers can exploit. By prioritizing network security, web application developers can help create a safe and trustworthy environment for users, reducing the risk of data breaches and unauthorized access. [11] [12]

2 CHOSEN TOOLS FOR DEVELOPMENT

Tools we work with can be essential and aid us in making secure web applications. This section looks at the most commonly used developer tools and frameworks, their most suitable usage, and a few tips and criteria for choosing the correct set of tools.

2.1 Overview of available technologies

The development of web applications has seen a rapid increase in recent years, with a wide range of technologies available for creating high-quality and interactive web apps. These technologies include client-side scripting languages such as JavaScript and jQuery, server-side programming languages like PHP, ASP.NET, and Ruby on Rails, CSS frameworks such as Bootstrap and Foundation, and JavaScript frameworks such as Angular and React. [13] [14] Additionally, various content management systems (CMS) like WordPress and Drupal allow for the easy creation and maintenance of web applications. These technologies offer various tools for developers to choose from, making it easier to create web apps that meet specific requirements, from simple blogs to complex e-commerce websites. [15] As a result, it is essential for developers to have a good understanding of the available technologies and to choose the right set of tools for the job, to build high-quality, scalable, and maintainable web applications.

2.2 Ranking the technologies

Each developer would rank technologies differently based on their experience in the field. First, it depends on the website's purpose. An obvious choice would be to pick frameworks. We can choose client-sided frameworks if we need single-page applications that don't need to store their users' data. The preferred choice is more robust frameworks with bigger teams and continued support, such as Angular, React, and Vue.js. But if you only need a light application without additional features or conveniences, less-known and smaller frameworks will do just fine. If your aim is robust applications with extensive functionality and significant data flow, it's better to look at server-side frameworks. Frameworks with great online support and frequent updates are the preferred choice. Another role in picking should be framework security, functionality, flexibility, and design patterns such as MVC or MVT. Sizable frameworks should be picked such as Django, Laravel, ASP.NET, and Express.JS for those reasons. These are some of the recommended technologies for easier and safer web application development.

2.3 Choosing correct technologies

The choice of technology is critical in developing web applications, as it directly affects the final product's quality, performance, and scalability. Choosing the right technology can make the development process smoother, faster, and more efficient, while the wrong choice can lead to technical debt and difficulties in maintaining the application. Furthermore, certain technologies may have limitations in terms of security, compatibility, and integration with other systems, which can impact the success of the web application. Additionally, certain technologies may be more suited to specific types of web applications, such as e-commerce websites, social networks, or mobile applications. As a result, it is crucial for developers to carefully consider the requirements of the web application, the target audience, and the business goals before choosing the technologies to be used in its development. Proper technology selection can make all the difference in ensuring the web application's success and long-term viability. [16] [17] [18]

2.4 Criteria for choosing the correct technologies

When choosing the tools for web app development, there are several key criteria to consider:

1. The tools should meet the project's technical requirements, such as scalability, security, and performance.
2. The tools should be easy to use and offer high productivity for the development team to ensure efficient and timely project delivery.
3. Compatibility with existing systems and technologies is essential, as it can affect the ease of integration, security, and future application maintenance.
4. The tools should be supported by a large and active community to resolve any technical issues quickly and effectively.
5. Make sure the tools are within the budget of your application.
6. It is also a benefit if solutions are already made for the tools used.

[19]

3 COMMON SECURITY ISSUES

These are the most common web application security issues defined by OWASP. “The Open Web Application Security Project® (OWASP) is a nonprofit foundation that works to improve the security of software”. Through community-led open-source software projects, hundreds of local chapters worldwide, tens of thousands of members, and leading educational and training conferences, the OWASP Foundation is the source for developers and technologists to secure the web. Categories chosen for this project are selected from their OWASP Top 10 documentation made in 2021. The website splits all security vulnerabilities into ten categories. [20]

3.1 Broken Access Control

Description:

Broken access control is ranked fifth in OWASP, with an average incident rate of 3.81%. Broken access control talks about enforcing specific policies so users cannot act outside their intended permissions. Common access control vulnerabilities include:

- Violation of privileges. Access to creation, deletion, or modification of data with insufficient or no right. This access should be granted to only specific roles but is instead given to everybody.
- Access to privileges, views, and application states outside their authorization level by URL or API request modification. These changes give attackers free rein over the state of our website.
- Allowing the attacker to pretend to be other roles than is permitted, such as users or administrators. Hence having permission that should be inaccessible ("Elevation of privilege").
- Change other accounts' information with their unique identifier.

Prevention:

There are a few ways to prevent Broken Access Control. The idea is to monitor which users are accessing what part of the website and keep them out of the areas he has no authorization for. Few prevention examples:

- Deny by default - The application should deny access to areas with no authentication by default.

- Authentication and Authorization - Showing data based on users' roles or permission. Deny access with insufficient authorization. This system allows programmers to monitor access across the whole website easily.
- Validation of incoming users' input and modification of website output to prevent modification or exploitation.
- Role-Based Access Control (RBAC) - Control users by grouping them into roles and assigning specific permissions to each group.

Consequences:

Among the most common and potentially damaging risks are data breaches. Breaches lead to sensitive information disclosures, identity thefts, or fraud.

[3] [21] [22] [23]

3.2 Cryptographic Failures

Description:

This category is related to cryptography (or lack thereof). First, we must look at what kinds of data we are working with. Data concerning cryptography are, for example, Resting data such as personal user information and passwords, credit card numbers, health records, or Data en route: data currently being transferred which could be caught by men in the middle attack.

Here are some problems that could lead to Cryptographic Failures:

- Use of weak, unsupported, discontinued, or old cryptographic algorithms
- Use of old, deprecated hashing functions.
- Not using any or insufficient encryption across the website, database, or other data storages.
- Transmission or storage of data in plain text.
- Usage of default or weak keys in cryptographic algorithms.
- Improper handling and management of the cryptographic keys.

Prevention:

Some of the preventions are:

- Keep track of the data by storing it. Enforce proper security and handling of the stored data. Perform regular checkups on the data and its security. Get rid of all the sensitive data after it's no longer needed or usable.
- Disable caching of the website on all the sensitive responses.

- Use multiple layers of protection on sensitive data, such as encryption, hashing, salting, or other kinds of security.
- Use Initialization vectors to ensure true randomness on your keys.
- Use only up-to-date and established Cryptographic functions, algorithms, and general protection.

Consequences:

In case attackers get access to data, poor cryptography or non at all can lead to theft and modification of data to conduct fraud, as well as identity theft which can lead to severe consequences.

[3] [24] [25] [26]

3.3 Injection

Description:

With an average incident rate of 3%, injections fall to one of the common security issues occurrences. A vulnerability that allows an attacker to interfere or directly query the database. Allowing attacker access, modification or deletion of data.

Some common vulnerabilities are:

- Not validating or sanitizing incoming Input from the user.
- Incorrect or missing parametrization of data.
- Not properly separated queries.
- Allowing Error-based SQLIs - Attackers force a website to generate an error message, based on which the attacker can determine which database the website uses and more.
- Allowing Union-based SQLIs - Combining multiple instances of data by the UNION SQL operator to get a positive website response.
- Boolean-based SQLIs - Forcing websites to return a response based on the True or False outcome. Generated website response may then vary based on the result of the query.
- Time-based SQLIs - The attacker guesses the True or False response by the time it takes the website to respond.
- Out-of-band-based SQLIs - These are dependent on the server configuration used by the website.

Prevention:

- Input validation and sanitization - Before passing a user request, properly check the Input and clear of any unwanted context.
- Pre-prepared queries - Already prepared queries for particular tasks which programmers only supply with parameters.
- Parametrization - Separation of user input and the query itself. The Input is inserted into the query only as a parameter of the query.

Consequences:

Since the attacker can access all the data, consequences are similar to attacks described before, such as personal information and data theft. Along with theft, an attacker can modify data and compromise the underlying server or other back-end infrastructure, along with a denial-of-service attack.

[3] [27] [28] [29] [30]

3.4 Insecure design

Description:

A relatively new category made in 2021. It is a broad category representing multiple weaknesses, usually called "missing or ineffective control design." To be more specific insecure design may refer to things like ignoring design and best architectural practices, starting at the planning phase and even before implementation. It is essential to note insecure design differs from insecure implementation, near perfect implementation cannot avoid defects from insecure design. Unlike other threats, we can look at this more like a concept of correct/incorrect implementations, decided before the implementation phase.

Prevention:

- Make security one of your priorities. Always be up to the newest security risks and their preventions. Or have specified personal or company care for security for you.
- Always test your code, as it helps to prevent many issues further down the line. It helps to find vulnerabilities and security holes before the app is deployed.
- Ensure you and your team are always caring for security, and it is one of the every-day tasks of the implementation process.

Consequences:

There can be many consequences of insecure design. For the sake of an example, we will pick one. When designing applications, teams may ignore secret management and access control best practices, leading to vulnerabilities. These flaws lead to system compromises

due to harmful password management practices such as storing user passwords as plaintext in application properties, configuration files, or memory. Other examples may be Improper Isolation or Compartmentalization, which means failing to correctly separate entities with varying rights, privileges, and access permissions, resulting in broken access control.

[3] [31] [32] [33]

3.5 Security Misconfiguration

Description:

This category again looks at a broader range of mistakes and errors. Misconfiguration arises when essential security settings are either not implemented or implemented incorrectly. These errors can happen at any level of the application stack: servers, databases, platforms or frameworks, storage, cloud services, or the code itself. A few of those mistakes are:

- Missing security across one or more parts of the website.
- Using default credentials and accounts.
- Missing or insecure password policy allowing weak and easily crackable passwords.
- Leaving unnecessary libraries, pages, accounts, and privileges enabled
- Used software is discontinued, out of date, and insecure. Older versions of dependencies are allowed.
- Improper maintenance or configuration of security features.
- Poor coding practices.
- Failure to block URLs meant for staff personnel.

Prevention:

- Enforce and automate a repeatable process to always deploy new updates, security features, and general improvements without affecting the stable running of the application.
- Check and install any necessary patches and upgrades as soon as possible to avoid new threats.
- Ensure not to install any unnecessary features, libraries, or other dependencies your application doesn't need to prevent increasing the risk of exploiting one of them.
- Make sure you have a healthy and well-structured development cycle.

- Make sure you or your employees are aware of the significance of security and are always prepared to respond to them.
- Use automatization processes for the checks and updating of your systems.
- Exercise a minimalistic approach to keep your application compact and free from unnecessary dependencies.
-

Consequences:

This issue can lead to a plethora of consequences. In some instances, a data leak may occur and expose personal information. Improper configuration and forgotten vulnerabilities may lead to the exploitation of your website or complete services shut down.

[3] [34] [35] [36]

3.6 Vulnerable and Outdated Components

Description:

As the name implies, it refers to components of our application that are no longer supported or unsecure to use. It can be in the form of libraries or frameworks. These types of code are usually implemented with little to no regard for security to save time or make the work easier.

A few of those vulnerabilities are:

- Unawareness or ignorance of the version of dependencies, systems, libraries, or any other external tools you or your application uses. This includes both client and server-side.
- Used software is outdated or completely discontinued, unsupported by the creator of the software and the community, or vulnerable to any attack. This includes many technologies like servers, used Operation Systems, Database Management Systems, various APIs, libraries, etc.
- Unawareness of the newest security vulnerabilities or advancements in the security field.
- Forgetting or ignoring to scan for any vulnerabilities in your project
- Feigning ignorance and not fixing found vulnerabilities in due time

Prevention:

- Remove all unnecessary and unused dependencies in your project

- Keep tabs on the newest version of used dependencies manually or by an automated process. Check for any new and discovered vulnerabilities or ways to get past your security and prevent such issues as soon as possible.
- Download dependencies only from trusted sources and trusted authors. Always check for its credibility if you are unsure about a specific dependency.

Consequences:

As with most categories, this vulnerability can be exploitable and lead to many issues. Data theft and modifications are among them. Companies that do not care for their vulnerable and outdated components can be fined, sued, or lose their business license if they are found to be neglecting them.

[3] [37] [38]

3.7 Identification and Authentication Failures

Description:

These types of failures occur when an application fails to implement the functionality associated with users' identity, authenticity, and session management. A few of those weaknesses are:

- Automated attacks such as credential stuffing are allowed. The attacker attempts to force login credentials stolen from different websites.
- The website doesn't deny or punish brute-force attacks.
- The website allows weak passwords with no or insufficient password validation.
- The application incorrectly implements functions regarding users' identity, authenticity, and session management.
- The website doesn't hide session identifiers and is easily accessible in the URL.

Prevention:

- In the case of credential stuffing, implement multi-factor authentication.
- Don't use default authentication credentials with administrative accounts.
- Validate users' passwords correctly, such as password length, use of numbers, and at least one big letter. Ensure the password is not similar to the username or password doesn't belong to the library of the most common passwords.
- Follow NIST Special Publication 800-63B Digital Identity Guidelines [39]
- Limit request attempts to avoid Brute Force attacks.

Consequences:

Since this vulnerability issues around Brute force and similar attacks, it primarily targets users' data. However, if an attacker gets access to admin credentials, it could lead to exponentially increasing damage.

[3] [40] [41]

3.8 Software and data integrity failures

Description:

Another new category, made in 2021, focused on critical data and updates added to the pipeline without verifying their integrity. An example may be where the application relies on plugins, libraries, or modules from untrusted sources. Or where the application uses auto-updating with no implemented verification. This opens a backdoor to the attackers allowing them to upload their updates.

Prevention:

- Authentication - Ensures we know where from and to data is flowing.
- Using dependencies, libraries, or other external software from verified and trusted sources.
- Code must be adequately tested before being deployed to production. Every change and update in the code should be tested as well.
- Make sure data is secure by encryption.
- Continually update parts of the code, maintaining your application's security.

Consequences:

An example of consequences could be an attacker invoking client-side code to modify cookies that supply malicious input, perform injection attacks, or bypass authentication, on cookies without proper Validation and Integrity checking.

[3] [42] [43]

3.9 Security Logging and Monitoring Failures

Description:

Security Logging and Monitoring failures are not an attack. It is a failure on the developers' side to prevent an attack. Prevention could be done by logging, monitoring, or sending reports about suspicious behavior on a website. These attacks may not lead to any loss; however, not implementing prevention against this category significantly increases

the chance of one succeeding. It might be in the interest of the developers to log various authentications from users and other related information such as the date and time of the login, the location where the user logged from, ip address, high-value transactions, warnings, and errors. Storing logs locally and not backing them up should also be avoided.

[3] [44] [45] [46]

Prevention:

- Make sure that the issues mentioned above are logged-in properly
- Make sure that logs are easily understandable and readable
- Make sure logs are securely hidden and secure from attacks or other forms of tampering

Consequences:

Consequences are the same as with all other attack types, as failing to log and correctly report suspicious activity raises the chance of an attack being successful. However, it highly depends on what kind of website is used and other preventive measures' quality.

3.10 Server-Side Request Forgery (SSRF)

Description:

Similarly to Broken Access Control, SSRF uses URLs for its attacks. It is a fetch request made to the server without validating the users' URL. These applications use URLs to get, modify, or post data to the server. Attackers hijack this functionality to get access to data or bypass security. [3] [47] [48]

Prevention:

- Scan and validate all user input
- Prevent sending raw responses to users
- Whitelist IP addresses your application requires access to.

Consequences:

Attacks may lead to the loss or manipulation of sensitive data or hijack of the whole system. [47] [49]

4 DESIGN

UI is one of the most significant parts of web application making as it's the first thing a user sees. This section describes websites approach to the UI, functional design, and the difficulties encountered along the way.

4.1 UI Design

With UI, a minimalistic approach with no unnecessary pages and views was aimed for, as this is an education-oriented website. Next, websites main color theme was decided to be purple, with complementary colors like green and orange. For other UI elements such as containers, semi-transparent white was chosen. The font was also made white. Buttons and containers were rounded to eliminate rough edges. As the website was a bit boring, abstract pictures and animated text were added to the main page. Pictures were also added to the Problems page for a little bit of decoration. Throughout the whole page, a navbar was visible for easy navigation on the website. The Problems Tab was also divided into Explanation, Showcase, and Countermeasures for an easier viewing experience and overall clarity.

4.2 Functional Design

For the functional design, a lot of work was cut out by Django itself. Starting with things like input data and who can enter it. As this website is for showcase purposes big or difficult data structures and sizable forms for input weren't needed. For the websites purposes default Django forms provided all the necessities. As for input from users, they can register, which unlocks some functionality for a better understanding of the explanations the website provides. For planning what each action would do, there weren't any specific ideas at the planning stage, only rough concepts which were being added on top of as the implementation process continued. Overall, not enough time was spent at the planning phase, at times resulting in chaotic and not exactly guided implementation, which had to be fixed later.

4.3 Technical requirements and limitations

As mentioned above the planning phase was neglected, so I wasn't sure of the requirements I would have, mostly regarding software. That resulted in the start of implementation in the Angular framework which had to be scrapped later. As of now, I know one of the requirements was a server-side framework which made my work much easier. The other requirements were the security of the chosen framework, support of the framework, and

compatibility with other packages such as Tailwind, good documentation is always a nice thing to have but not necessarily a requirement. As for limitations, there weren't any.

II. PRACTICAL PART

5 DEMONSTRATION OF SECURITY ISSUES IN THE PROJECT

The first part of the application-making process was the research and structuring of the data provided to the user. It is important to present the user with concise and objective data on each picked subject so the user doesn't get lost in the information. Nobody likes to look at walls of text; thus, it's good practice to provide other visual clues to help the user understand the topic. This section describes my approach to this issue, with a detailed explanation of my steps.

5.1 Website description

This web application is built to be a solid foundation for starting developers, trying their hands in this field. It is meant as a tutorial of sorts, to show and explain difficulties and problems regarding secure applications. The website takes categories from Owasp (Common security issues) and explains them to them while implementing the security for the issue itself. Each issue has its category, shown with examples and best practices to use while securing their website.

5.2 Demonstration of each issue

As this is an education-oriented website, it needs to explain the security issues it describes. For that reason, each security issue has its category. Each of those categories is then split into 3 subcategories. The first, one being Explanation, here website explains what the risk is we are working with, how it works, and common mistakes and consequences of the risk.

The second one is Showcase, where the website presents the user with more thorough explanations along with code examples, and structures as well as shows how the actual attack looks and works.

And the last one is Examples, here we show the users how to prevent the actual attacks in general terms, rather than the code itself as the website is meant to be used with any language and frameworks. It explains best practices and shows an implementation using examples with either the project itself or some other type.

5.2.1 Broken Access Control

Having started with Broken Access Control, a general explanation of the security issue was provided, along with some common broken access vulnerabilities. In the showcase tab, a simple `http://127.0.0.1:8000/admin` URL was shown. This URL took us to the admin view log-in page. Initially, the intention was to hide this view and make it inaccessible, but since it did not pose a significant security issue, it was left in for showcase purposes. It was required for the admin to have a very strong password with the admin view public. Then, examples were shown on how to bypass or access places that users should not see without proper authentication, and a view with data was created as a showcase example of such URLs.

In the countermeasures tab, users were advised on the usage of libraries such as `UserCreationForm` and `AuthenticationForm` for Django, `PHP OAuth 2.0 Server` for PHP, and `Microsoft Authentication Library (MSAL)` for .NET. The process of hiding data from unauthenticated users was also explained, with pictures provided for further clarification. To explain how to prevent such attacks, the section on Preparations was started, where the general idea of authentication and the necessary requirements were explained, including the need for a database with a specific structure, using the provided example. In the database, an important column called `"is_superuser"` could be seen, which was used for authenticating admin users. Another column with higher authority than a user was `"is_staff"`. Code from a website was used to showcase how access can be conditioned on the front-end side of the application.

The second step, "Working with data," began with an explanation of simple SQL statements to provide some understanding for complete beginners. This was followed by a code snippet that was called every time a user attempted to log in. If the code was called, it first authenticated the user, and if the user existed, it logged them in and directed them to the main page. This aimed to give users a general idea of how the authentication process should appear and the steps they should follow.

The last step was "Access proofing our App," where the focus was on conditioning the URLs in the front and back-end of the application. This category aimed to explain Broken Access Control in simple and understandable terms, providing the most efficient solutions, along with good practices and advice to follow.

5.2.2 Injections

The website once again begins with a brief explanation of the security issue, outlining its consequences and what the attack leads to. Taking complete beginners into account, the Showcase Tab is started with a general explanation of how the SQL language works. First, a picture is provided to display the database structure, followed by the presentation of the SELECT SQL query and a picture depicting the outcome of the query. To enhance understanding, another SELECT query is demonstrated. The next topic covered is the UPDATE query, where before and after screenshots are shown to visually demonstrate the changes. After explaining the SQL language, the focus shifts to showcasing Structured Query Language Injection (SQLI), featuring screenshots performed using the website. The most common SQLI strings are presented in the username field, along with an explanation of these strings. Both SQLIs belong to the SELECT variation, which aims to reveal the website's database data. Text explanations are provided to aid comprehension of the security issue and the SQLIs. The Tab also includes a button that directs users to the playground Tab, enabling them to try the Injections on their own. Finally, a video from PortSwiggerTV is provided to enhance users' understanding of the issue.

The countermeasure Tab commences with advice on utilizing frameworks or libraries that are safeguarded against this security issue, as it is preferable to rely on expert-developed solutions rather than those created by beginner programmers. Since Django has a rather concealed and intricate implementation against this security issue, I had to devise my own approach. For this purpose, I opted for parametrization, which is the most commonly used method to prevent SQLIs. The implementation begins with a brief explanation of how parametrization functions, accompanied by some advice. A code block showcasing the implementation follows, featuring a python example using the mysql.connector library. The code snippet provides a concise demonstration of how to implement parametrization. After the implementation, the code is dissected and explained in greater detail. Additionally, an extra layer of security can be added with input validation, so I present another python example illustrating this technique. It operates by detecting and comparing keywords from a dictionary of unwanted terms, replacing them to eliminate most SQLIs prior to reaching the parametrization stage. An example of the output is also showcased.

5.2.3 Cryptographic Failures

As it's now a theme starting with a brief explanation of the Cryptographic Failures. The explanation includes an all-around overview of the issue, examples, and general knowledge. Then some security risks that the users should avoid are listed. The Showcase Tab is started with a discussion like questions targeting users' implementations and projects, aiming to evoke a sense of urgency to pay attention to this security risk. Then a showcase of how encrypted password should look, for beginners who may have never seen encryption. The example was made using python and an AES library. Next a warning is provided to urge readers to check for the security of the algorithms they are using as new encryption algorithms can become outdated very quickly. Continued with list of algorithms that are currently considered secure and best to use, as well as algorithms users should avoid.

To better understand how easy it is to break old algorithms a MD5 cracker script is made, using python and the hashlib library. Next, providing the users with an option to download both the code and text file with passwords, so they can try and play around with it if they want to. Since this website doesn't have any server-to-server communication it doesn't use its PBKDF2 encryption. That meant a different implementation of some encryption algorithm had to be made. The algorithm had to be one that is easy to understand and still secure for usage, RSA was picked.

In the beginning, the list of used libraries is provided. Then the code itself is sectioned off by its functions and each part is explained. At the end of the code, a warning is shown for the reader, that websites implementation is just an exemplary version and is not suited to be used professionally.

Moving on to an example of hashing using python and the hashlib library. The showcase uses SHA256 since it's the same algorithm my website utilizes for its password encryption. In the end a download for the RSA script is provided so users can try it out themselves, and a video from ArtOfTheProblem which explains the RSA algorithm in easy to understand way and in great detail.

5.2.4 Identification and Authentication Failures

Starting with the explanation of Identification and Authentication failures, the issue and consequences of the attack are explained. The attacks are then categorized into three parts. Lastly, reasons for the attacks are given to show readers what to prevent. Then, further

explanation of each issue is provided. As not much can be done by users with credential stuffing, there was nothing that could be showcased regarding that attack. Some examples of how a stealable session ID in the URL would look and how the URL should actually look like are made.

For brute force attacks, a simple Python code that attacks one of the website's views with the CSRF token disabled was made. The code is then explained, and download links for the script and necessary files are provided.

In the last tab, preventing credential stuffing is explained first, with the two-step verification being the top choice. A link to a website where users can check their email addresses for potential leaks is also provided. For Brute Force prevention, the implementation had to be coded again since Django implementation would not be suitable for explanation purposes. The code is done in Python with the Regular Expression (re) library to make it more readable and easier to understand. In this case, readers were not provided with a download link since the whole code is pasted in the code block. Following the code are a few pieces of advice on how to strengthen security. The last part is exposed session IDs. As explained in the paragraph, this security risk isn't that much of an issue nowadays as most frameworks have hiding session IDs in the cookies by default or as a simple yes or no option. Short guidelines for how to check session IDs on most websites are provided, backed by screenshots of the author's own "session cookie," in the case of Django csrf token.

5.2.5 Vulnerable and Outdated Components

The last category starts with the usual explanation of the security problem. The explanation includes examples as well as some general advice and consequences of ignoring this category. In the showcase category, the aim is to explain what dependencies are and how to get to them. A simple example is made using the Python pip freeze command, as well as the .NET list command, or checking manually in the project explorer. Once users know what dependencies are and where to look for them, the issue can be explained.

Starting with the simplest fix of deleting unnecessary dependencies. Then websites for checking library and framework vulnerabilities, such as NIST and CVE, as well as websites for NPM packages like SNYK, are provided. Then examples of how readers could check versions of their dependencies are given by again providing some links for websites or libraries. An example of how the whole process of checking their dependencies should look like is provided using the check-outdated library, as well as using the tools provided earlier

to check package vulnerability. The whole process is followed by examples and short explanations. Toward the end, some general advice and practices to follow are given.

6 COUNTERMEASURES IMPLEMENTATION

Implementing countermeasures is one of the ways programmers can protect their applications against security risks. In this section, the explanation will be provided on how the countermeasures were implemented.

6.1 Overview of available measures

As is customary, the internet is filled with ways to implement the desired functionality. Since the decision was made to use a robust framework like Django, a significant portion of the implementation was accomplished using built-in libraries. However, we will explore alternative measures that could have been taken. For the examples, bigger and more well-known frameworks will be used, along with the most common countermeasures.

6.1.1 Broken Access Control

Starting with Broken Access Control people using frameworks will have their workload significantly smaller than people trying to do it from scratch. Most frameworks have authentication systems already built in and the programmer only needs to implement them. But in general, a measure for this security issue is authentication. However, there are other ways such as denying access by default, limiting CORS usage, or mandatory access control. Frameworks like .NET solve broken access control using Identity and identity roles. PHP-based frameworks like Laravel also provide a wide selection of countermeasures such as authentication, policies, and libraries like Laravel Passport and Laravel Sanctum. And last but not least websites Broken Access prevention was done using the built-in libraries `UserCreationForm` and `AuthenticationForm` for user authentication, with Django generated SQL table. From the research done, a conclusion was made that authentication was the most reliable, and commonly used prevention of Broken Access Control.

6.1.2 SQL Injections

The Second category SQL injections is again a common issue amongst websites, but usually a quick fix. For some reason, this type of attack is ignored a lot even in bigger frameworks, leaving the user with more work and responsibility. Thankfully SQL injections are not that hard to prevent. Some of the prevention techniques are Validation and Sanitation of the input, Pre-prepared queries, Properly Constructed Stored Procedures, and Statements, parametrization of the queries, fetching results, and more. From the research done .NET

framework has no built-in protection against SQLIs, but the most common seems to be parametrization. Laravel has no prevention either however, there is good documentation explaining prevention, such as the validation of user inputs and parametrized queries. Client-side frameworks such as Angular and React also do not implement any security measures by default. One exception found is Django, where they have its robust system in place, taking care of SQLIs by parametrization. Their documentation discourages programmers from using raw queries, yet still, provides the option for the desired customization. The most common security prevention for Structured Query Language Injections (SQLI) seems to be parametrization, user input validation, and sanitation.

6.1.3 Cryptographic failures

The Third category is cryptographic failures, again a very important security failure to prevent. This category can be split into smaller pieces. Using the newest and most secure algorithms is important to keep users' data unbreachable. Some secure algorithms are Blowfish, Twofish, RSA, PBKDF2, and Triple DES. Thankfully there are many libraries across all frameworks that users can use without the need to program them themselves. .NET Framework advises its users to hash with their PasswordHasher. Laravel provides users with libraries for Bcrypt and Argon2, react and angular also provide a Bcrypt library. Django by default uses SHA256 for hashing however, supports both Bcrypt and Argon2. As we can see modern frameworks do provide a wide choice of cryptographic algorithms users can use. Nevertheless, programmers need to be aware of such technologies and never leave this security issue unchecked and always be sure to use up-to-date algorithms. They should always enforce encryption where necessary and not store any data in plaintext. For internet traffic use well-known protocols like HTTP and SMTP.

6.1.4 Identification and Authentication Failures

Identification and Authentication Failures can be split into a few subcategories. Starting with credential stuffing, this subcategory is depending heavily on the type of website we are programming. If we are making some utility-like web application with no sensitive users' data, this type of attack is not of big concern. However, once personal data is saved, proper security needs to be in place. The most efficient way to prevent credential stuffing would be 2-factor authentication. For this, frameworks usually do provide some sort of implementation in the case of .NET we have ASP.NET Core Identity, Laravel has Fortify, in Angular and React, we can use third-party libraries like Twilio or Google Authenticator and in

Django, we have libraries like `django-two-factor-auth` or `django-trench`. So as we can see this security issue has great coverage across all the most used frameworks. The next category is brute-force attacks. This category revolves around user authentication. Depending if your own or the framework's implementation of the authentication is used weak passwords become an issue. But looking through modern frameworks, they seem to have some sort of implementation in place for it. Most commonly it's password validation, for example, they won't allow the password unless it's `x` characters long and has an uppercase letter or number in it. While implementing this security point myself, I found that Django uses the so-called csrf token. This token disallows attackers to even connect with their script without the csrf header or makes it significantly harder. Then we have saving passwords into plaintext, which is a huge risk if data were to be leaked, it's important to always hash passwords. The last big category would be session identifier theft. This category is also not such a big issue nowadays. Frameworks by default store session ids in the website cookies, not in the URLs. That practice is also not bulletproof, but more secure.

6.1.5 Vulnerable and Outdated Components

For the last category on the website, Vulnerable and outdated components were chosen. This security issue can be easily overlooked and is not something frameworks will fix for you. It is very broad and more dependent on the programmer. However, there are always tools to make your job easier. For any frameworks compatible with npm, programmers can use tools like SNYK. For PHP frameworks, you can use the composer. For bigger websites, you may want to consider making an automated system for checking and updating your packages for you. Or use already done systems like Github Action and Renovate. It is also important to download packages either using managers like pip or from well-known and trusted sources. Another important thing to look out for is package support from the developer and the community for ease of use. Packages and dependencies aren't the only issues, other vulnerabilities may include the operating system used, the server the app is running on, the database management system used, runtime environments, or APIs.

6.2 Summary of available measures

As we can see, there is a plethora of different solutions for each of our problems. Helping us along the implementations are countless libraries and already-done scripts, saving us time and work. However, these are only some of the tools we can use as there is

another kind of third-party software for our aide. One example is WAF (Web Application Firewall). This type of protection can help with attacks like cross-site scripting, SQL Injections, file inclusion, and many others. It is added layer of protection that can stop attacks even before reaching our website by monitoring HTTP traffic. Another added measure can be an IPS (Intrusion Prevention System), helping us to filter malicious activity. IPS solutions are also very effective at detecting and preventing vulnerability exploits. "When a vulnerability is discovered, there is typically a window of opportunity for exploitation before a security patch can be applied. An intrusion prevention system is used here to quickly block these types of attacks." [50] If that is still insufficient for you, add NGFW (Next Generation Firewall) for traffic monitoring. Still, need more? Throw in some logging using tools like Sematext Logs, SolarWinds Loggly, Splunk, or many more. This should be enough of a showcase to see how many available technologies we have for securing and enforcing our web applications.

6.3 Implementation of security measures

There were a few steps and measures that needed to be done to make my web application secure. As with most things nowadays the internet is full of solutions and it's on the programmer to choose one. My decision to pick the Django framework turned out to be correct, as the framework has built-in systems made exactly for the purposes needed. However, that was only regarding the security of the application. As the purpose of the website is to educate it needs to find and display the best measures and solutions in more general terms and not the ones applied to my project. In most cases, starting the categories with strong emphasis for readers to use already-made libraries or framework-implemented solutions, the website took the same approach.

6.3.1 Broken Access Prevention

The implementation started with enforcing measures against Broken Access Control. After conducting some research, authentication was chosen as the solution. Since user authentication would be useful in later categories as well, it was an obvious choice. Once the countermeasure was decided upon, it was time to implement it. Fortunately, Django provided extensive and well-maintained documentation, along with an active community, making implementation straightforward. After reviewing available implementations, the built-in `UserCreationForm` and `AuthenticationForm` libraries were chosen. The implementation began

with setting up the database and data structure. It was pleasantly surprising to discover that Django had its own structure in place for authentication, which was easy to implement.

id	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active	date_joined
1	pbkdf2_sha256\$390000\$P1SwKHSdHFCR0bdNt...	2023-02-17 10:54:58.914962	0	asdasasd				0	1	2023-02-17 10:54:58.481384
3	pbkdf2_sha256\$390000\$JL8y6Ea5wJNELwq9r...	2023-03-20 10:11:24.924624	1	admin				1	1	2023-02-17 15:47:29.193573
4	pbkdf2_sha256\$390000\$naE9WqCGyO0TRgfi...	2023-02-17 16:41:17.396357	0	newUser				0	1	2023-02-17 16:41:16.966782
5	pbkdf2_sha256\$390000\$JTIGUJMqOqAlbGZVIG...	2023-02-20 10:03:08.662650	0	alteredUser1				0	1	2023-02-17 16:50:47.000000
6	pbkdf2_sha256\$390000\$3vuot7VcWVKTA6#Hd...	2023-03-21 08:44:50.931983	0	user2				0	1	2023-02-17 16:57:04.834599
7	pbkdf2_sha256\$390000\$KdeiI2wVSMid8KglMu...	2023-02-17 17:16:33.827106	0	user3				0	1	2023-02-17 16:57:44.586913
8	pbkdf2_sha256\$390000\$FEdqja74G6XmXVUzA...	2023-02-17 17:00:50.747935	0	user4				0	1	2023-02-17 17:00:50.317359
9	pbkdf2_sha256\$390000\$8damr0H1ysfSphvVIH...	2023-02-17 17:50:36.671080	0	user5				0	1	2023-02-17 17:50:36.239524
10	pbkdf2_sha256\$390000\$18Kzs8p3MakKTX1IuT...	2023-02-17 17:54:31.773469	0	user6				0	1	2023-02-17 17:53:49.422209
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 1: Database structure for authentication

The picture above shows the default database structure used by Django Authentication, which was suitable for use without modifications for this website. The "is_superuser" column in this table plays an important role in determining which user has administrative rights and can make changes to the database. This information can be utilized for checks in the future. With the data structure in place, the focus shifted to implementing the forms. Two views were required: "Sign Up" and "Log In." The "Sign Up" view included three fields: username, password, and password confirmation. The "Log In" view used the same fields for username and password. On the front-end side (Template) of the app, the <form> tag was utilized with the post method, and the fields named "username" and "password" were bound to the corresponding view methods. Now, let's take a look at the appearance of these view functions.

```
def signup(request):
    if request.user.is_authenticated:
        return redirect('/')
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password1')
            user = authenticate(username=username, password=password)
            login(request, user)
            return redirect('/')
        else:
            return render(request, 'core/signup.html', {'form': form})
    else:
        form = UserCreationForm()
        return render(request, 'core/signup.html', {'form': form})
```

Figure 2: View function for sign up

The figure above could be described following with the first if condition, we check if the user trying to sign in is already not authenticated, if he is, he gets redirected to the main page. The second condition checks if the incoming request is of type post, which the form being sent is. After the condition, the form is created from the given username and password. Another check controls if the form is valid, such as if the passwords match or if the username is already in use. A valid form is saved into the database and the code moves to authenticate the user. After authentication user is logged into the website. There is also an "else" statement that checks the situation where the form isn't valid, in such case the request is returned with the same view, and a form containing an error message which is then displayed.

```
<p>
    {% for field in form %}
        {% for error in field.errors %}
            <p style="...">{{ error }}</p>
        {% endfor %}
    {% endfor %}
</p>
```

Figure 3: Paragraph for displaying error messages in case of invalid form request

The Sign In view function works the same way as Sign Up, skipping the form creation steps.

```
def signin(request):
    if request.user.is_authenticated:
        return render(request, 'home.html')
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('/') # profile
        else:
            msg = 'Wrong Credentials'
            form = AuthenticationForm(request.POST)
            return render(request, 'core/signin.html', {'form': form, 'msg': msg})
    else:
        form = AuthenticationForm()
        return render(request, 'core/signin.html', {'form': form})
```

Figure 4: View function for sign in

Once the back end of the authentication was done, all that was left to do was secure each path by a simple "if" condition. For the showcase of this behavior a Profile page was made, it displays the users' username and hashed password.

```
{% if user.is_authenticated %}
  <li>
    <a href="/profile"
      class="block py-2 pl-3 pr-4 text-white rounded hover:bg-gray-100 md: hover: bg-transparent"
    >
  </li>
{% endif %}

{% if user.is_superuser %}
  <li>
    <a href="/admin"
      class="block py-2 pl-3 pr-4 text-white rounded hover:bg-gray-100 md: hover: bg-transparent"
    >
  </li>
{% endif %}
```

Figure 5: Conditioning the access based on the authentication level

Here we can see two different kinds of conditions. One was made for the user to make the before mentioned Profile path visible on the navigation bar. The other is for admin, it is also a simple link on the navigation bar for the admin view. As an addition log out functionality is also implemented.



Figure 6: Navigation bar without authentication

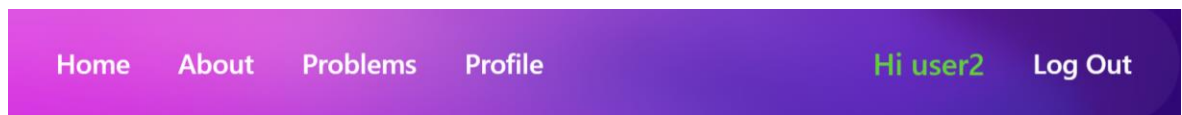


Figure 7: Navigation bar with authentication

6.3.2 SQL Injections

When implementing SQL Injection (SQLI) protection, first a research on how these attacks look and are performed was done. Seeing the different kinds and practices helped to better understand the countermeasures later on. After that, research on all the possible countermeasures. The most prominent ones that were found were parametrization and input

validation and sanitation. Once there was an idea of what to do implementation was started. Starting with looking for ways one can implement these countermeasures in Django, A discovery was made, that the website was already secure from SQL Injections. Implementation done for Broken Access Control already had protection built in.

```
user = authenticate(request, username=username, password=password)
if user is not None:
    login(request, user)
    return redirect('/') # profile
```

Figure 8: Code holding SQLI protection

To my surprise, the first line of code has already made SQLI protection. Unfortunately, Django doesn't allow to dig deeper into some of its functions so a search through documentation was done. From that newfound knowledge Django uses QuerySets and parametrization for its security. Each query set holds its own specific parameters and data sets. "A QuerySet is a collection of data from a database. A QuerySet is built up as a list of objects. QuerySets makes it easier to get the data you actually need, by allowing you to filter and order the data at an early stage." [51]

Once a QuerySet is performed all the data belonging to the QuerySet are fetched. It can have none or many filters. Filters determine how much data is fetched based on each filter's parameters. "In SQL terms, a QuerySet equates to a SELECT statement, and a filter is a limiting clause such as WHERE or LIMIT." [51]

This means that the input we get from the user never reaches the actual database or the data. It is only used as data inside the parameter supplied to one of the filters of QuerySets. This whole process is packaged in the "authenticate()" function. The function returns a user based on entry parameters if such a user exists. Afterward, the user is logged in. Django also allows us to make raw queries directly to the database, skipping the QuerySet process, as is shown in examples.

6.3.3 Cryptographic Failures

There wasn't much to fear with cryptographic failures as Django is a big and well-supported framework with regular updates. Python also has many cryptographic libraries such as hashlib, bcrypt, and pyca/cryptography. From my previous work with the database, there was already knowledge that Django uses some sort of algorithm to hide passwords.

After some reading through their documentation, usage of PBKDF2 algorithm with SHA256 hashing was discovered. Components used for storing a User's password - <algorithm>\${iterations}\${salt}\${hash}.

```
if request.method == 'POST':  
    form = UserCreationForm(request.POST)
```

Figure 9: Implementation of prevention

This is the same as with Injections. Prevention was already enforced along with the authentication implementation. Function "UserCreationForm(request.POST)" not only creates a user Form but also validates the password passed in. If credentials passed in are incorrect error message is returned.

```
AUTH_PASSWORD_VALIDATORS = [  
    {  
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',  
    },  
]
```

Figure 10: Default Django validators

The next step Django implements to prevent Cryptographic Failures is password validation. For that Django already has prepared validators. In the code, the default validators are used as they are sufficient. The first validator is "UserAttributeSimilarityValidator". It makes sure the password user wants to use is not similar to his username, email, and first or last name. We can also change the level of similarity allowed, which is by default 0.7. These are all the parameters we can set up in this validator. The second validator "MinimumLengthValidator" is kind of self-explanatory. It checks the length of password allowed, which is by default 8. We can change the length required to our needs. The third validator "CommonPasswordValidator" makes sure that the password is not among the list of the most popular passwords, the

list contains 20 000 entries. And the last "NumericPasswordValidator" make sure that the password is not composed entirely of numbers.

```
password  
pbkdf2_sha256$390000$fPtSwKHsDHFCR0bcNt9Pkg$3mJDSYxAekIk0/4AFAhR0FORNpbaJHDTG11UYpRx1SQ=
```

Figure 11: Password encrypted by Django

6.3.4 Identification and Authentication Failures

The fourth category is Identification and Authentication Failures. As with the previous ones, starting with reading what the security issue is and how we prevent it. Beginning with credential stuffing, where the solution is two-factor authentication. Decision to ignore this issue was made, for multiple reasons. First, this is a small-scale website that doesn't hold any personal data, from the perspective of users it would seem unnecessary and perhaps annoying. Second, adding unnecessary features may lead to more vulnerabilities and possibilities for exploitation. The third being the code couldn't be showcased since it's mostly done through libraries where we don't see the whole code. Fourth, this website is not meant for commercial use or data collection, users in the database are dummy users made for testing and showcase purposes. For these reasons the choice was to just mention this can be an issue, and not implement countermeasures. The second subcategory is Brute Force Attacks. For this category, the most common countermeasure is strong passwords. As that was already implemented by Django password validators, that wasn't an issue. Another layer of security is made by "csrf". This token makes sure that when a request is made, it requires the csrf header to be included. That already makes it impenetrable. The only tool found for getting through this measure is a software testing tool called "Burp Suite".

```
@ratelimit(key='ip', rate='10/m', block=True)  
def signin(request):
```

Figure 12: Limiting Sign In request rate

Once the attacker breaks through the csrf, another surprise will be waiting after his script makes 10 requests, an exception will be raised. After that he will need to restart the view, making it exponentially more time-consuming. And the last major risk issue is session identity theft.

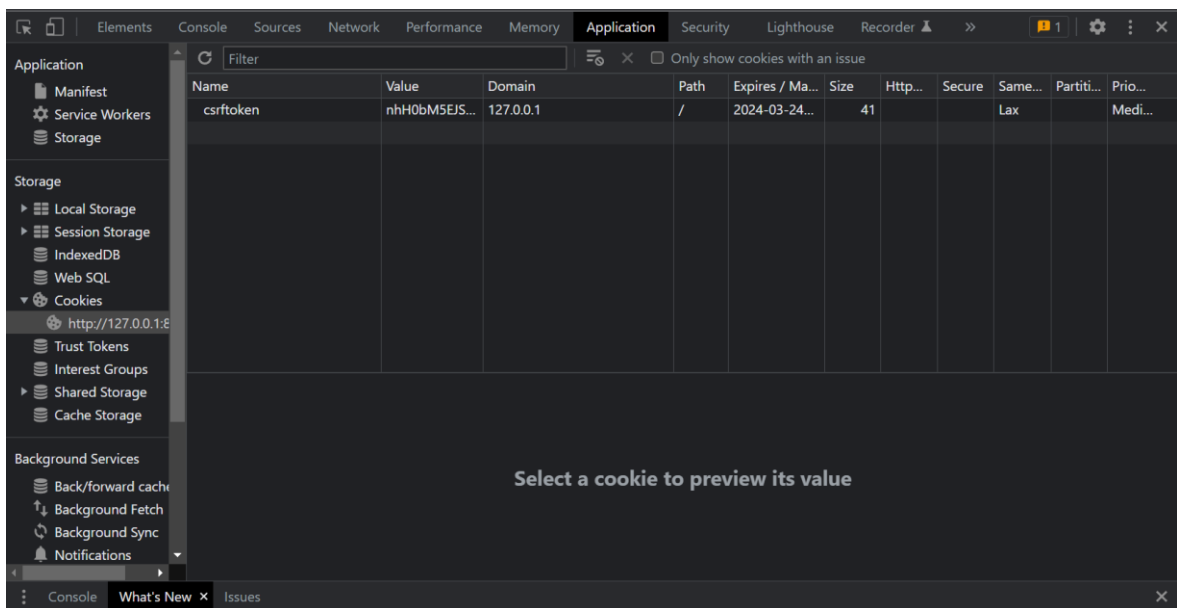


Figure 13: Django session token in cookies

There was no need for countermeasure implementation as this is protected by default. Django like most other frameworks hides identifiers in the cookies, the difference being "csrftoken" instead of id (identifier).

6.3.5 Vulnerable and Outdated Components

The last category Vulnerable and Outdated components wasn't about implementing countermeasures, rather it was about making sure my dependencies are up to date and supported. For that, a variety of tools and websites was used. A caution was taken not to download any unnecessary libraries and use only ones that receive frequent updates.

6.4 Efficiency of used countermeasures

Implementation of countermeasures used by the website are efficient and up to date with the newest threats. Since most of the countermeasures were implementations of Django libraries. After some research, a conclusion that each of the Django libraries and algorithms uses the most efficient and modern methods for solving corresponding security issues was reached. Built-In libraries have easy-to-understand and well-maintained documentation, supplemented by examples and guides. And as for implementations shown as examples on my website, they are also up-to-date and fully applicable to be used as solutions except one (RSA algorithm).

7 EXAMPLES OF IMPLEMENTATION

Learning is tedious if all we do is read the text. Hence, website provides users with a couple of examples for each category, some being interactive.

7.1 General overview

Since the objective of this website is education, it is not sufficient to merely secure the website itself. The website needs to be taught to our readers. The explanation should be straightforward and easy to understand. Therefore, the best implementations, practices, libraries, and tools to address each issue were researched. The problems, attacks, and countermeasures were explained in simple terms to ensure comprehension even for complete beginners. In cases where a deeper understanding of the subject was necessary, the process of reaching the final solution was demonstrated from the beginning. Teaching solely through text poses challenges, so the inclusion of pictures and code examples was maximized.

Furthermore, fully operational scripts were provided for readers to download and experiment with independently. However, what sets this website apart from others? The website is constructed in a way that encourages readers to explore the aforementioned attacks. For each implemented category, there is a vulnerability intentionally left open for readers to attempt to exploit and manipulate. Some security gaps serve as decoys, while others are fully functional parts of the website that remain accessible. This approach aims to deliver an engaging experience that goes beyond lengthy blocks of text.

7.2 Implementing examples

Each example is implemented either in the Showcase or Countermeasures tab. Implementations are done mostly using python code or, in a few cases, code from my website. Each example is inserted in a code block with syntax highlighting for better visibility. For more extended codes, download links are provided. New tabs were created distinct from the operational security measures for cases involving website usage. These tabs are always accessible by visible buttons on the website. These tabs are consistently accessible through visible buttons on the website. Detailed explanations were given for each provided code.

7.2.1 Broken Access Control

Commencing with Broken Access Control, a more straightforward example was presented, which is implemented in the Showcase Tab of the category. Due to the nature of

Django authentication and Session Identifiers, it was not feasible to utilize internal workings for the example. Hence, it was necessary to simulate the output of the attack. To accomplish this, I opted for an inexistent URL path and created a corresponding view.

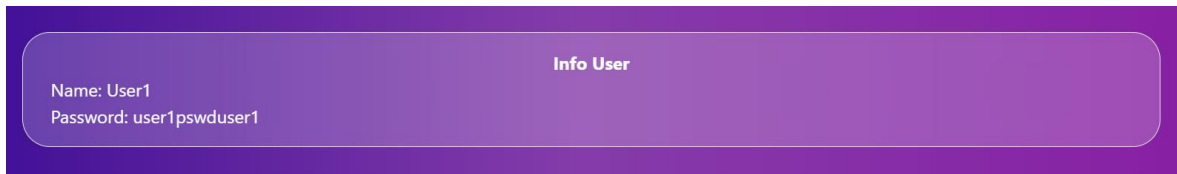


Figure 14: Mock view simulating Broken Access

The view itself is just a simple tab with two text fields. The first field holds the username of a dummy account, and the second contains its password. This login information is used as login credentials to simulate the attack better. Unfortunately, they are not pulled from the database as it would return a hashed password which would be unusable for the user. The reader can see the same view when logging in, in the "Profile" Tab, where the information is taken from the database, as is shown in the Countermeasures Tab.

In order to highlight the issue further, users are allowed to see the admin view. I find this practice to be unsecure. However, this approach is used on multiple websites. This behavior is a default one implemented by the Django framework. It is better than it may seem since the admin views require a login to access the actual admin tools. And when creating the "super_user" (admin) account, the framework doesn't allow weak passwords unless the programmer accepts such passwords himself despite the warnings. A path for changing the user by URL is also provided but leads to the same admin login page.

With this example, aim was to deepen the readers' understanding of this security issue and raise awareness of how easy it may be to miss it.

```
http://127.0.0.1:8000/admin  
http://127.0.0.1:8000/admin/auth/user/5/change/  
http://127.0.0.1:8000/problems/broken_access/user1
```

Figure 15: Exemplary URLs used for showcase

7.2.2 SQL Injections

The second category, SQL Injections, proved to be more challenging. Default Django forms, along with their authentication libraries, effectively handle Injections. Initially, the feasibility of allowing SQL Injections through functioning forms seemed impossible. The

initial approach involved creating fake forms with input fields and checking for specific user-input keywords. However, this solution required an extensive dictionary of potential outcomes, which appeared impractical overall. The idea was scrapped search for better solution began. After a short while, a discovery was made, that programmers can make "raw" queries, connecting them to the database. This was great news, as there was now a way to bypass the default Django parametrized queries. With the provided piece of information, the implementation was initiated. Interestingly enough, it was soon discovered that achieving Django's insecurity required significant effort, as the attempted queries consistently failed. Consequently, a decision was made to once again seek solutions, eventually stumbling upon one within Django's official documentation. This solution was presented as a warning against a particular action, which, ironically, was precisely what was done. After experimentation and thorough testing, the desired outcome was successfully attained.

```
cursor.execute(
    "SELECT * FROM auth_user WHERE username = '%s' AND password= '%s'" % (pass_username, pass_password))
row = cursor.fetchone()
```

Figure 16: Injectable SELECT query

For the example, a SELECT statement was employed, and quotes were included to render the parameters susceptible to exploitation. Following this, a consideration was made regarding whether to authenticate and log in the user. However, as it appeared illogical, the decision was made to present the query's output to the user instead. Since the query returned a tuple, it was appended to the return statement within the view, accompanied by the corresponding message.

```
if row is not None:
    msg = 'Log in Successful'
else:
    msg = 'Wrong Credentials'
```

Figure 17: Message display after a query

The message is "Log in Successful" or "Wrong Credentials" based on the outcome of the attack.

Along with those is also a pop-up window which, based on the query result, displays a fail message or window with the query's outcome.

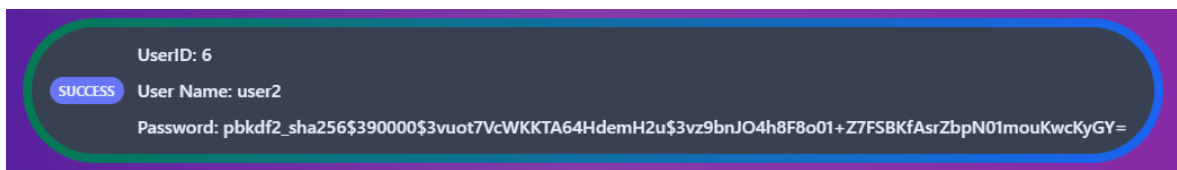


Figure 18: Pop up with query result

On the window, the decision was made to display the queried user ID, username, and encrypted password. The password was intentionally left encrypted to provide a more accurate simulation of the actual query result. As for the front-end implementation, a straightforward "if" condition was employed, dependent on the "sent" variable, to ensure that the text is only displayed if a post request is made. Furthermore, a check was implemented to determine what information should be shown based on the query outcome, whether it is empty or not. This view can be accessed through a button located in the showcase tab.

7.2.3 Cryptographic Failures

For the Cryptographic Failures category, the example required a bit of thinking. Not the implementation, but rather what the example should be. Given the limitations of not being able to crack Django's password encryption for the showcase example, an alternative approach had to be devised. The goal was to retain the concept of algorithm cracking while ensuring that the chosen algorithm was simple enough for readers to comprehend the code and weak enough to avoid excessive computational requirements. This was essential for the purpose of the showcase. After conducting research, an outdated hashing algorithm called MD5 was discovered, which proved to be relatively easy to crack. The example revolves around the scenario where the attacker already possesses the hashed password, obtained through an injection attack, for instance. To demonstrate this, a Python function was created as a password cracker. The function accepts two arguments: a list of passwords to iterate through and a password hashed with MD5. Within the function, a loop is implemented to iterate through each password and hash it using the "hashlib" library. After hashing, it is compared to the password supplied by the user. The outcome of each comparison is printed out on the console. The script stops once a match is found. Readers can find this script in the Showcase tab, along with an explanation. With the script come two download links for the script itself and the passwords used in the example. This example is meant to show how easy it can be to crack an outdated algorithm unsupported by any other encryption method, such as salting.

7.2.4 Identification and Authentication Failures

As Brute Force attacks are a significant aspect of identification and authentication failures, they were chosen as the focus for the example. However, since there was no prior experience with Brute Force attacks, it became necessary to learn how to execute one. Fortunately, simple Brute Force attacks are relatively straightforward to carry out, which facilitated the implementation process.

```
import requests

url = "http://127.0.0.1:8000/signin/"
req = requests.post(url)

print(req.text)
```

Figure 19: Post request to the website

Starting of by making a "post" request to the login URL (<http://127.0.0.1:8000/signin/>). That's where the issue started. Since the website uses Django's forms for authentication, there is a complication called the "csrf" token.

```
<div id="summary">
  <h1>Forbidden <span>(403)</span></h1>
  <p>CSRF verification failed. Request aborted.</p>

  <p>You are seeing this message because this site requires a CSRF cookie when
  submitting forms. This cookie is required for security reasons, to ensure that your
  browser is not being hijacked by third parties.</p>
  <p>If you have configured your browser to disable cookies, please re-enable them, at
  least for this site, or for "same-origin" requests.</p>
</div>

<div id="info">
  <h2>Help</h2>

  <p>Reason given for failure:</p>
  <pre>
  CSRF cookie not set.
  </pre>
```

Figure 20: Response after the post request

To enhance website security, a token system was implemented, requiring a header with a Cross-Site Request Forgery (CSRF) authenticator for every request made to the website. However, this token-based protection posed a more significant challenge than initially anticipated. After extensive research, it was discovered that penetrating this level of protection

proved to be nearly impossible. Only a few tools, such as "Burp Suite," were capable of bypassing it, but using such tools would not effectively demonstrate how the attack is performed in a general sense, specifically targeting Django. This realization led to a point of discouragement, almost causing the abandonment of the Brute Force attack idea. Fortunately, one last attempt was made to explore if there were any potential workarounds available to developers rather than attackers.

```
path('problems/identification_authentication_failures/', Ident_Auth_Failures, name='IAF'),
path('problems/identification_authentication_failures/explanation', ia_explanation, name='explanation'),
path('problems/identification_authentication_failures/example', csrf_exempt(ia_example), name='example'),
path('problems/identification_authentication_failures/showcase', ia_showcase, name='showcase'),
path('problems/identification_authentication_failures/countermeasures', ia_countermeasures, name='countermeasures'),
path('problems/identification_authentication_failures/usernames', download_usernames, name='download_usernames'),
path('problems/identification_authentication_failures/passwords', download_passwords, name='download_passwords'),
path('problems/identification_authentication_failures/script', download_script, name='download_script'),
```

Figure 21: Turning off csrf token for an URL

Upon further exploration, it was discovered that Django allows the option to flag specific URLs with "csrf_exempt()" to completely disable CSRF protection on those URLs. With this newfound knowledge, a decision was made to create a duplicate of the login page, leaving the actual login page secure while allowing the Brute Force attack to be performed on the replicated version. After implementing this change, the post requests started to come through successfully. Subsequently, research was conducted on how to recreate the attack using Python.

The initial step involved creating two text files and populating them with commonly used usernames and passwords, including one set of credentials specific to the website being targeted. A Python function was then developed, taking the parameters of usernames, passwords, and a URL. The usernames and passwords were stored in a dictionary and sent via a post request. If the request returned the response "Wrong Credentials," the script continued executing.

```
25 Login Failed
26 Login Failed
27 Login Failed
28 Login Successful
29
30 Credentials:
31 Username: user2
32 Password: user2pswd2
33 [Finished in 7.4s]
```

Figure 22: Successful breach

However, once it receives "Login Successful," the script gets the correct combinations and returns the credentials used, which are then printed. The whole script can be read on the website with a brief explanation, along with download links for all three files. The aim here was to show how easily their web applications can be the subject of such attacks without the proper protection and to give them a playground where they could try the attack themselves.

7.2.5 Vulnerable and Outdated Components

Creating an example within the Vulnerable and Outdated Components category posed a challenge due to the absence of exploitable vulnerabilities and the limitation of not being able to download insecure dependencies for the project. As an alternative approach, different tools were explored to include in the Countermeasures Tab. Fortunately, one of these tools identified an outdated library. To confirm its security status, the Snyk website was used, and it was determined that the library was free from vulnerabilities. This presented an opportunity to use it as an example.

To illustrate the process, the initial step involved using an "npx" library to check all the project's dependencies. Subsequently, various checks were performed on the Snyk website, capturing screenshots at each stage. These screenshots are included in the example to simulate the process a programmer should follow when encountering a suspicious or outdated dependency.

8 CONCLUSION

This thesis point was to create an educational application in the field of security. My goals were to study the security vulnerabilities of web-based applications and forward my findings as a web application. For that purpose, the five most common ones were picked and described in the application. Problems are demonstrated using explanations, examples, and countermeasures. The last goal was to make the application secure, which was also accomplished.

There are a few similar projects made for the same purpose. Such as OWASP Juice Shop or Try Hack Me. However, these are not beginner-friendly web applications and are more specialized. My work is meant to introduce people to security, and the abovementioned applications could work as extensions to my work for those who want to go beyond the basics. And as for other types of tutorials, I found they need improvement as they only explained it on the surface without good examples or a more comprehensive range of solutions. Many were purely text-based without examples, which might be overwhelming for beginners. As such, I aimed to combine the necessary textual explanations with easy-to-understand examples with pictures, screenshots, and others. Alongside that, I offer general advice and practices to make the programmers' life easier.

Now for the limitations and future of the work. It might seem as limitation, but there are some Prerequisites needed to try the application.. The user will need to have a Linux server with MariaDB installed, go through the configuration process, and then populate the database with his own data. That is a step this work would solve in the future. Deploy the website on a server accessible to everyone. Or make a package aimed more towards staff education and training in corporations or similar facilities. Another important thing is the website needs to be updated regularly, as the world of security is ever-expanding.

In summary, this thesis has demonstrated the importance of correctly implementing security in web applications. I hope this tool can help new programmers interested in the field of security and secure application making. Let's program a safer future together.

9 REFERENCES

- [1] "Atatus," 2023. [Online]. Available: <https://www.atatus.com/ask/what-are-the-most-common-web-application-issues>. [Accessed 2023].
- [2] "Relevant," 26 January 2023. [Online]. Available: <https://relevant.software/blog/web-application-security-vulnerabilities/>.
- [3] Owasp, "OWASP Top Ten," 2021. [Online]. Available: <https://owasp.org/www-project-top-ten/>. [Accessed 2023].
- [4] S. AS, "Sungardas," 11 October 2021. [Online]. Available: <https://www.sungardas.com/en-us/blog/the-consequences-of-a-cyber-security-breach/>. [Accessed 2023].
- [5] imperva, "Application Security Testing," 2023. [Online]. Available: <https://www.imperva.com/learn/application-security/application-security-testing/>. [Accessed 2023].
- [6] O. Moradov, "Bright," 8 December 2021. [Online]. Available: <https://brightsec.com/blog/application-security-testing/>. [Accessed 2023].
- [7] "Feroot," 2023. [Online]. Available: <https://www.feroot.com/education-center/what-is-front-end-security/>. [Accessed 2023].
- [8] C. ODOGWU, "makeuseof," 23 September 2021. [Online]. Available: <https://www.makeuseof.com/prevent-frontend-security-risks/>. [Accessed 2023].
- [9] C. ODOGWU, "makeuseof," 9 September 2021. [Online]. Available: <https://www.makeuseof.com/backend-security-risks-and-preventions/>. [Accessed 2023].
- [10] E. BORGES, "security trails," 11 March 2023. [Online]. Available: <https://securitytrails.com/blog/backend-security-risks>. [Accessed 2023].
- [11] essentialtech, "Network Security Threats And Solutions You Need To Know," 2023. [Online]. Available: <https://www.essentialtech.com.au/blog/5-most-common-network-security-risks>. [Accessed 2023].
- [12] checkpoint, "What is Network Security?," 2023. [Online]. Available: <https://www.checkpoint.com/cyber-hub/network-security/what-is-network-security/>. [Accessed 2023].
- [13] orient, "WHAT WEB DEVELOPMENT TECHNOLOGIES SHOULD YOU USE?," 2023. [Online]. Available: <https://www.orientsoftware.com/technologies/web-technologies/>. [Accessed 2023].
- [14] B. Kohan, "Guide to Web Application Development," 2023. [Online]. Available: <https://www.comentum.com/guide-to-web-application-development.html>. [Accessed 2023].
- [15] C. Newcomer, "Best CMS Software to Build a Website," 11 April 2022. [Online]. Available: <https://kinsta.com/blog/cms-software/>. [Accessed 2023].
- [16] M. Semenov, "10 Tips on How to Choose a Technology Stack for Web Application Development in 2023," 2023. [Online]. Available: <https://seclgroup.com/tips-to-choose-tech-stack-for-web-app-development/>. [Accessed 2023].

- [17] Artelogic, "How to choose the right technology stack for web applications," 18 January 2018. [Online]. Available: <https://artelogic.net/blog/how-to-choose-the-right-technology-stack-for-web-applications/>. [Accessed 2023].
- [18] D. A. Tripathi, "Right Technology Stack for Web Application Development?," 23 June 2022. [Online]. Available: <https://www.mtractionenterprise.com/blog/right-technology-stack-for-web-application-development/>. [Accessed 2023].
- [19] M. Semenov, "10 Tips on How to Choose a Technology Stack for Web Application Development in 2023," 2023. [Online]. Available: <https://seclgroup.com/tips-to-choose-tech-stack-for-web-app-development/>. [Accessed 2023].
- [20] OWASP, "Who is the OWASP Foundation?," 2023. [Online]. Available: <https://owasp.org/>. [Accessed 2023].
- [21] S. Sengupta, "Broken Access Control and How to Prevent It," 20 September 2021. [Online]. Available: <https://crashtest-security.com/broken-access-control-prevention/>. [Accessed 2023].
- [22] R. Clancy, "What Is Broken Access Control Vulnerability, and How Can I Prevent It?," 12 October 2022. [Online]. Available: <https://www.eccouncil.org/cybersecurity-exchange/web-application-hacking/broken-access-control-vulnerability/>. [Accessed 2023].
- [23] s. sharma, "How to Prevent Broken Access Control?," 27 February 2023. [Online]. Available: <https://www.tutorialspoint.com/how-to-prevent-broken-access-control/>. [Accessed 2023].
- [24] O. Hiremath, "Introduction to Cryptographic Failures," 25 July 2022. [Online]. Available: <https://www.softwaresecured.com/introduction-to-cryptographic-failures/>. [Accessed 2023].
- [25] S. Sengupta, "OWASP Top 10 Cryptographic Failures A02 – Explained," 7 Jun 2022. [Online]. Available: <https://crashtest-security.com/owasp-cryptographic-failures/>. [Accessed 2023].
- [26] M. Beshkov, "A02:2021 – Cryptographic Failures Owasp: Know This Cyber Trouble Better," 20 February 2023. [Online]. Available: <https://www.wallarm.com/what/a02-2021-cryptographic-failures>. [Accessed 2023].
- [27] PortSwigger, "SQL injection," 2023. [Online]. Available: <https://portswigger.net/web-security/sql-injection>. [Accessed 2023].
- [28] Imperva, "SQL (Structured query language) Injection," 2023. [Online]. Available: <https://www.imperva.com/learn/application-security/sql-injection-sqli/>. [Accessed 2023].
- [29] S. M., "What is SQL Injection & How to Prevent SQL Injection," 14 February 2023. [Online]. Available: <https://www.simplilearn.com/tutorials/cyber-security-tutorial/what-is-sql-injection>. [Accessed 2023].
- [30] Acunetix, "What is SQL Injection (SQLi) and How to Prevent It," 2023. [Online]. Available: <https://www.acunetix.com/websitesecurity/sql-injection/>. [Accessed 2023].
- [31] S. Sengupta, "Insecure Design," 23 May 2022. [Online]. Available: <https://crashtest-security.com/insecure-design-vulnerability/>. [Accessed 2023].
- [32] Foresite Security, "Foresite Security," 2023. [Online]. Available: <https://foresite.com/blog/owasp-top-10-insecure-design/>. [Accessed 2023].

- [33] Snyk, "Insecure design," 2023. [Online]. Available: <https://learn.snyk.io/lessons/insecure-design/javascript/>. [Accessed 2023].
- [34] Reciprocity, "Security Misconfigurations: Definition, Causes, and Avoidance Strategies," July 28 2022. [Online]. Available: <https://reciprocity.com/blog/security-misconfigurations-how-to-avoid-them/>. [Accessed 2023].
- [35] Balbix, "Security Misconfiguration: Impact, Examples and Prevention," 2023. [Online]. Available: <https://www.balbix.com/insights/security-misconfiguration-impact-examples-and-prevention/>. [Accessed 2023].
- [36] A. Dizdar, "Security Misconfiguration: Impact, Examples, and Prevention," 29 May 2022. [Online]. Available: <https://brightsec.com/blog/security-misconfiguration/>. [Accessed 2023].
- [37] S. Brathwaite, "The Risks in Vulnerable and Outdated Components," 26 September 2022. [Online]. Available: <https://www.softwaresecured.com/the-risks-in-vulnerable-and-outdated-components/>. [Accessed 2023].
- [38] Foresite Cybersecurity, "OWASP Top 10: #6 Vulnerable and Outdated Components," 2023. [Online]. Available: <https://foresite.com/blog/owasp-top-10-vulnerable-and-outdated-components/>. [Accessed 2023].
- [39] NIST, "Digital Identity Guidelines," 2 March 2020. [Online]. Available: <https://pages.nist.gov/800-63-3/sp800-63b.html>. [Accessed 2023].
- [40] Crashtest Security, "Crashtest Security," 2023. [Online]. Available: <https://crashtest-security.com/identification-failure-guide/>. [Accessed 2023].
- [41] Cyolo Team, "Identification And Authentication Failures And How To Prevent Them," 11 May 2022. [Online]. Available: <https://cyolo.io/blog/identification-and-authentication-failures-and-how-to-prevent-them/>. [Accessed 2023].
- [42] S. Sengupta, "A08:2021 – Software and Data Integrity Failures- Explained," 02 June 2022. [Online]. Available: <https://crashtest-security.com/owasp-software-data-integrity-failures/>. [Accessed 2023].
- [43] Educative, "What are software and data integrity failures?," 2023. [Online]. Available: <https://www.educative.io/answers/what-are-software-and-data-integrity-failures>. [Accessed 2023].
- [44] educative, "What are security logging and monitoring failures?," 2023. [Online]. Available: <https://www.educative.io/answers/what-are-security-logging-and-monitoring-failures>. [Accessed 2023].
- [45] Foresite, "OWASP Top Ten: #9 Security Logging and Monitoring Failures," 2023. [Online]. Available: <https://foresite.com/blog/owasp-top-ten-9-security-logging-and-monitoring-failures/>. [Accessed 2023].
- [46] MyF5, "K94068935: Security logging and monitoring failures (A9) | Secure against the OWASP Top 10 for 2021," 15 February 2023. [Online]. Available: <https://my.f5.com/manage/s/article/K94068935>. [Accessed 2023].
- [47] MyF5, "K36263043: Server-side request forgery (SSRF) (A10) | Secure against the OWASP Top 10 for 2021," 15 February 2023. [Online]. Available: <https://my.f5.com/manage/s/article/K36263043>. [Accessed 2023].
- [48] PortSwigger, "Server-side request forgery (SSRF)," 2023. [Online]. Available: <https://portswigger.net/web-security/ssrf>. [Accessed 2023].

- [49] imperva, "Server-Side Request Forgery (SSRF)," 2023. [Online]. Available: <https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/>. [Accessed 2023].
- [50] paloalto networks, "What is an Intrusion Prevention System?," 2023. [Online]. Available: <https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-prevention-system-ips/>. [Accessed 2023].
- [51] Django, "Django documentation," 2023. [Online]. Available: <https://docs.djangoproject.com/en/4.2/>. [Accessed 2023].
- [52] Sanjay, "Broken Access Control In ASP.NET Core – OWASP Top 10," 29 August 2022. [Online]. Available: <https://procodeguide.com/programming/broken-access-control-in-aspnet-core/>. [Accessed 2023].
- [53] Stackhawk, "Laravel Broken Access Control Guide," 9 November 2021. [Online]. Available: <https://www.stackhawk.com/blog/laravel-broken-access-control-guide-examples-and-prevention/>. [Accessed 2023].
- [54] StackHawk, "Laravel SQL Injection Guide," 2 March 2021. [Online]. Available: <https://www.stackhawk.com/blog/sql-injection-prevention-laravel/>. [Accessed 2023].
- [55] SimliLearn, "What Is Data Encryption: Types, Algorithms, Techniques and Methods," 11 November 2022. [Online]. Available: <https://www.simplilearn.com/data-encryption-methods-article>. [Accessed 2023].
- [56] Arcserve, "5 Common Encryption Algorithms and the Unbreakables of the Future," 14 February 2023. [Online]. Available: <https://www.arcserve.com/blog/5-common-encryption-algorithms-and-unbreakables-future>. [Accessed 2023].
- [57] H. Saini, "8 Strongest Data Encryption Algorithms in Cryptography," 11 March 2022. [Online]. Available: <https://www.analyticssteps.com/blogs/8-strongest-data-encryption-algorithms-cryptography>. [Accessed 2023].
- [58] J. Hadzima, "Two Factor Authentication with Email in ASP.NET Core," 21 June 2021. [Online]. Available: <https://www.marathonus.com/about/blog/two-factor-authentication-with-email-in-aspnet-core/>. [Accessed 2023].
- [59] codevoweb, "How to Implement Two-factor Authentication (2FA) in React.js," 14 December 2022. [Online]. Available: <https://codevoweb.com/two-factor-authentication-2fa-in-reactjs/>. [Accessed 2023].
- [60] P. Redmond, "Password Validation Rule Object in Laravel 8," 27 April 2021. [Online]. Available: <https://laravel-news.com/password-validation-rule-object-in-laravel-8>. [Accessed 2023].
- [61] R. Morcillo, "How to Update Dependencies Safely and Automatically with GitHub Actions and Renovate," 5 November 2020. [Online]. Available: <https://www.freecodecamp.org/news/update-dependencies-automatically-with-github-actions-and-renovate/>. [Accessed 2023].
- [62] A. Seher, "Default and Custom password validators in Django," 2 November 2022. [Online]. Available: <https://www.letscode.com/blog/how-to-add-custom-password-validators-in-django/>. [Accessed 2023].
- [63] R. KUĆ, "10+ Best Log Analysis Tools & Log Analyzers of 2023 (Paid, Free & Open-source)," 4 January 2023. [Online]. Available: <https://sematext.com/blog/log-analysis-tools/>. [Accessed 2023].

- [64] orient, "Web Development Technologies," 2023. [Online]. Available: <https://www.orientsoftware.com/technologies/web-technologies/>. [Accessed 2023].
- [65] O. Hiremath, "Introduction to Cryptographic Failures," 15 July 2022. [Online]. Available: <https://www.softwaresecured.com/introduction-to-cryptographic-failures/>. [Accessed 2023].
- [66] Django, "Password management in Django," 2023. [Online]. Available: <https://docs.djangoproject.com/en/4.1/topics/auth/passwords/>. [Accessed 2023].
- [67] w3schools, "Django QuerySet," 2023. [Online]. Available: https://www.w3schools.com/django/django_queryset.php. [Accessed 2023].
- [68] Django, "Making queries," 2023. [Online]. Available: <https://docs.djangoproject.com/en/4.1/topics/db/queries/>. [Accessed 2023].
- [69] Django, "Password management in Django," 2023. [Online]. Available: <https://docs.djangoproject.com/en/4.1/topics/auth/passwords/>. [Accessed 2023].
- [70] Django, "Security in Django," 2023. [Online]. Available: <https://docs.djangoproject.com/en/4.1/topics/security/>. [Accessed 2023].
- [71] Django packages, "Two-Factor Authentication (2FA)," 2023. [Online]. Available: <https://djangopackages.org/grids/g/two-factor-authentication/>. [Accessed 2023].
- [72] Twilio, "Authy: 2FA and Passwordless Login," 2023. [Online]. Available: <https://www.twilio.com/docs/authy>. [Accessed 2023].
- [73] PostSrc, "Get List of Outdated Packages in PHP Composer," 2023. [Online]. Available: <https://postsrc.com/code-snippets/get-list-of-outdated-packages-in-php-composer>. [Accessed 2023].
- [74] Laravel, "Laravel," 2023. [Online]. Available: <https://laravel.com/docs/8.x/sanctum>. [Accessed 2023].
- [75] Laravel, "Laravel Fortify," 2023. [Online]. Available: <https://laravel.com/docs/10.x/fortify>. [Accessed 2023].
- [76] Wikipedia, "Next-generation firewall," 6 March 2023. [Online]. Available: https://en.wikipedia.org/wiki/Next-generation_firewall. [Accessed 2023].
- [77] Laravel, "Laravel," 2023. [Online]. Available: <https://laravel.com/docs/8.x/passport>. [Accessed 2023].
- [78] diyaroy22, "How to Prevent Broken Access Control?," 14 January 2022. [Online]. Available: <https://www.geeksforgeeks.org/how-to-prevent-broken-access-control/>. [Accessed 2023].
- [79] F5, "What is a Web Application Firewall (WAF)?," 2023. [Online]. Available: <https://www.f5.com/glossary/web-application-firewall-waf>. [Accessed 2023].
- [80] Cloudflare, "What is a WAF? | Web Application Firewall explained," 2023. [Online]. Available: <https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/>. [Accessed 2023].
- [81] Cisco, "What Is a Next-Generation Firewall?," 2023. [Online]. Available: <https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-next-generation-firewall.html#~ngfw-firewall>. [Accessed 2023].
- [82] Snyk, "Vulnerable and outdated components," 2023. [Online]. Available: <https://learn.snyk.io/lessons/vulnerable-and-outdated-components/javascript/>. [Accessed 2023].
- [83] back4app, "Top 10 Server Side Frameworks," 2023. [Online]. Available: <https://blog.back4app.com/top-10-server-side-frameworks/>. [Accessed 2023].

- [84] AppMaster, "Top 10 Best Web Backend Frameworks in 2022 for Web Development," 6 September 2022. [Online]. Available: <https://appmaster.io/blog/10-best-web-backend-frameworks>. [Accessed 2023].
- [85] Technostacks, "Technostacks," 12 January 2023. [Online]. Available: <https://technostacks.com/blog/best-javascript-frameworks/>. [Accessed 2023].
- [86] CheatSheets Series Team, "SQL Injection Prevention Cheat Sheet," 2023. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. [Accessed 2023].
- [87] Dotnet Tutorials, "Password policy in ASP.NET Identity," 2023. [Online]. Available: <http://dotnet-tutorials.net/identity/password-policy>. [Accessed 2023].

10 LIST OF PICTURES

<i>Figure 1: Database structure for authentication</i>	37
<i>Figure 2: View function for sign up</i>	37
<i>Figure 3: Paragraph for displaying error messages in case of invalid form request</i>	38
<i>Figure 4: View function for sign in</i>	38
<i>Figure 5: Conditioning the access based on the authentication level</i>	39
<i>Figure 6: Navigation bar without authentication</i>	39
<i>Figure 7: Navigation bar with authentication</i>	39
<i>Figure 8: Code holding SQLI protection</i>	40
<i>Figure 9: Implementation of prevention</i>	41
<i>Figure 10: Default Django validators</i>	41
<i>Figure 11: Password encrypted by Django</i>	42
<i>Figure 12: Limiting Sign In request rate</i>	42
<i>Figure 13: Django session token in cookies</i>	43
<i>Figure 14: Mock view simulating Broken Access</i>	45
<i>Figure 15: Exemplary URLs used for showcase</i>	45
<i>Figure 16: Injectable SELECT query</i>	46
<i>Figure 17: Message display after a query</i>	46
<i>Figure 18: Pop up with query result</i>	47
<i>Figure 19: Post request to the website</i>	48
<i>Figure 20: Response after the post request</i>	48
<i>Figure 21: Turning off csrf token for an URL</i>	49
<i>Figure 22: Successful breach</i>	50