|  |
| --- |
| Petar Dakov  S3-CB04    2023/2024 |

Authentication & authorization

Crenit | The Netherlands

Security report

2023

# Version History

|  |  |
| --- | --- |
| When? | What? |
| 25/09/2023y. | Changed the main and sub-questions. |
| 03/11/2023y. | Refractured the references |
| 11/10/2023y. | Moved Content table to second page & added a second title. |

Contents

[Version History 2](#_Toc149999676)

[Problem description 3](#_Toc149999677)

[Main question 3](#_Toc149999678)

[Sub-questions 3](#_Toc149999679)

[Result 4](#_Toc149999680)

[Conclusion 7](#_Toc149999681)

[Bibliography 8](#_Toc149999682)

# Problem description

These days breaches of big companies and their websites are more common and happen more often. There are a lot of risks when creating a web application. Many people don’t know most of them and how to properly secure their website. In this document we will encounter most of the web application security risks and break them down to understand them.

# Main question

What are the best practices for securing user authentication and authorization on “QuizzTurn”?

# Sub-questions

1. What are the recommended authentication methods for user login?
2. Document analysis:
3. Community research:
4. Literature study:
5. How should password policies be enforced to enhance user password security?
6. Document analysis:
7. Expect interview:
8. Literature study:
9. How to protect against common security vulnerabilities like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF)?
10. Document analysis:
11. Community research:
12. Literature study:
13. What considerations should be made for securing APIs used for authentication and authorization?
14. Document analysis:
15. Expert interview:
16. Literature study:

# Result

**Sub-question 1**

**What are the recommended authentication methods for user login?**

When we think about authentication methods for users, we all can come up with an idea right now -Username and password. But with the rising of cyber-threats, companies need to ensure the security of their users. (Nyakundi, 2020)

One of the methods for authentication is **Biometric authentication.** It uses a unique fingerprint to verify the identity of the user. This method is mostly used in smart phones and recently there has been an increase in the use of this method in laptops, too.

**Two-factor authentication**, also known as 2FA, is an additional layer of security that can be used to protect your account. It provides the user with a second piece of information on his device (usually a code generated by an app on their phone). Enabling 2FA significantly enhances the security of your account, making it considerably harder for an unauthorized person to access it, even if they know your password. While 2FA is not entirely immune to breaches, it offers a higher level of security when compared to relying solely on a username and password. If you prioritize the security of your account, activating 2FA is a wise and effective measure.

**Passwordless login** is a way of accessing an account without the need for a username and password. There are several compelling reasons for transitioning to a passwordless login approach. Firstly, it offers enhanced convenience for users, sparing them from the burden of memorizing yet another set of login credentials. Secondly, it heightens security, eliminating the risk of weak passwords being targeted or guessed by potential attackers. Now, the process of establishing a passwordless login can be achieved through various methods, each with its advantages and disadvantages. (Nyakundi, 2020)

**Token-based authentication** provides a secure method for user verification by issuing unique tokens. These tokens grant access to specific resources and are typically generated by the system, sent to the user's device or email. Token-based authentication is favored for its enhanced security and scalability benefits. Unlike passwords or biometrics, tokens are only created upon request, offering heightened security. They can also be set to expire after a specified period, further improving security. This method, gaining popularity with the complexity of modern web applications, stores tokens on the client side, bolstering security. Additionally, it simplifies scaling, as there's no need to store tokens on the server. In summary, token-based authentication offers superior security and performance compared to other methods, making it a recommended choice for implementing authentication in web applications.

There are many more methods for authenticating the user, but there were the most common and used by most people.

**Sub-question 2**

**How should password policies be enforced to enhance user password security?**

Security issues in text-based password authentication are rarely caused by technical issues, but rather by the limitations of human memory, and human perceptions together with their consequential responses. (Yıldırım, 2019)

A lot of different authentication schemes which aim at aligning security have been proposed, but none of them can overcome the simplicity and affordability of typing a sequence of keyboard characters to allow authenticating users. A good password needs to be easy to remember and hard to guess at the same time. But if a password is short and easy to remember it becomes vulnerable and significantly creates a risk of being guessed by people close to the owner or someone who has information about them.

Research shows that users who must change their passwords regularly tend to choose more memorable phrases, which are easy for hackers to crack. This is why the use of third-party applications that ensure the management of your security data like passwords are used more and more. (Soare, 2021)

Some of the policies that should be enforced for every company to keep their users safe from security breaches:

- Enable the paste functionality on the password entry field to facilitate the utilization of password managers.

- A system should store a salted hash instead of passwords.

- Enable systems to permit users to display passwords when entering them, instead of the more secure dots or asterisks.

- Enabling multi-factor authentication

- Using authenticated protected channels and approved [encryption](https://cyberexperts.com/encyclopedia/encryption/) to request memorized secrets

- Create passwords with more than eight characters.

- Use a passphrase containing a combination of capitalized and small alphabets and punctuation marks.

- Avoid using common words and personal information to create passwords.

- Use unique passwords for different accounts

Also a good way to monitor your user accounts is to provide your specialist with the proper cybersecurity tools for them to monitor the users activity and identify compromised accounts in real-time.

**Sub-question 3**

**How to protect against common security vulnerabilities like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF)?**

**SQL Injection**

Hackers are always searching the internet and campus websites for SQL injection vulnerabilities. They use automated tools to find these weaknesses and typically aim to exploit them for financial gain, like stealing personal information for identity theft. (Berkeley, n.d.)

Because many modern web applications rely on data and are easily accessible online, SQL injection vulnerabilities are common and easy for attackers to target. Moreover, when multiple applications share the same database, a SQL injection flaw in one application can put others at risk.

SQL injection attacks can lead to:

* Theft, modification, or even destruction of sensitive data such as personal identifiable information.
* Elevation of privileges at the application, database, or even operating system level.

**How to protect form SQL injection attacks**

Developers can prevent SQL injection vulnerabilities in web applications by utilizing parameterized database queries, typed parameters etc.

Additionally, developers can take further steps to minimize attacks:

1. Keep all web application software components up to date with the latest security patches available.
2. Utilize the principle of least privilege when provisioning accounts used to connect to the SQL database.
3. Do not use shared database accounts between different web applications.

**Cross-site scripting**

Cross-site scripting works by manipulating a vulnerable web site so that it returns JavaScript to users. After executing inside the victim’s browser, the attacker can compromise their interaction with the application. (PortSwigger, n.d.)

There 3 main types of **XSS** attacks:

1. Reflected XSS – the script comes from the current HTTP request.
2. Stored XSS – the script comes from the database of the website.
3. DOM-based XSS – the vulnerability exists in client-side code rather than server-side code.

Preventing XSS is easy in some cases, but it can get harder with the complexity of the web application. Here are some of the measures to prevent cross-site scripting:

* Filter input on arrival (Filter and validate the user input)
* Encode data on output – Encode the output to prevent it from being interpreted as active content.
* Use appropriate response headers.
* Content Security Policy (CSP)

**Sub-question 4**

**What considerations should be made for securing APIs used for authentication and authorization?**

Most applications use a modern web framework and apply RESTful API. It’s a simple and flexible way of structuring a web API. An API created with these intentions is vulnerable to misuse by individuals with malicious or careless intentions. Your application should have a set of rules determining who has the privilege to access or alter data on your server. For example, only the author [Editor's note: the editors, as well] of a blog post should have the ability to make changes, while readers should have only the right to view it. If anyone could edit the post you're currently reading, it would lead to unwanted alterations and deletions by vandals, spammers, and others without control. (Neray, 2021)

**TLS**

Every web API should use TLS (Transport Layer Security). TLS protects the information your API sends (and the information that users send to your API) by encrypting your messages while they're in transit. You might know TLS by its predecessor's name, SSL. You'll know a website has TLS enabled when its URL starts with https:// instead of http://.

**SSO**

SSO lets the users verify themselves with a third-party application (like Google, Microsoft, or AWS) by way of token exchange to get access to a resource. They'll log in to their Google account, for instance, and be granted access to your app.

When creating an API for you website you should use a strong authentication and authorization solution. Poor or non-existent authentication and authorization are major risks. Since APIs provide an entry point to the database of the application, it’s critical that the owner strictly controls the access to them. Also some APIs reveal too much information. This typically occurs when an API leaves the task of filtering data to the user interface instead of the endpoint. The data passed through the API should aways be validated first. (Nyakundi, 2020)

# Conclusion

After my research I can conclude that protecting my Web application from vulnerabilities will be much easier than before. It helped me to understand how unsecure some sign in methods are, and why more and more password policies are required with registering to other applications. It is really useful to use a 2FA to prevent hackers from seeing your personal details. For the future I’ll be more careful with the requirements and the security of my applications.

# Bibliography

Berkeley, U. o. (n.d.). *UC Berkeley*. Retrieved from berkeley.edu.

Neray, S. S. (2021, October). *stackoverflow*. Retrieved from stackoverflow.blog.

Nyakundi, H. (2020). *Secure User Authentication Methods*. Retrieved from freecodecamp.org.

PortSwigger. (n.d.). *PortSwigger*. Retrieved from portswigger.net.

Soare, B. (2021, December). *Heimdal*. Retrieved from heimdalsecurity.com.

Yıldırım, M. (2019, April 11). *SpringerLink*. Retrieved from springer.com.