OWASP Report

S3 Individual Project

“Northern Eagle Deliveries”

Date	:	14.12.2020
Version	:	1.0
State	:	Proposal
Author	:	Kaloyan Aleksiev

Version history

Version	Date	Author(s)	Changes	State
1.0	14.12.2020	Kaloyan Aleksiev	-	Proposal

Table of Contents

1. Introduction

The aim of this document is to discuss and describe the 10 most often occurring security risks found in web applications (Top 10 Web Application Security Risks, 2020), address how they are tackled in this specific project, or if they are not, explain the reasoning of the software decision behind that.

The Open Web Application Security Project (OWASP, 2020) is a “nonprofit foundation that works to improve the security of software.” They produce publicly available articles and tools that aim to help developers build secure applications. The OWASP top 10 is important because it assists organizations and companies by giving them a priority of potential risks to focus on and helps them eliminate the vulnerabilities in their systems. According to information found on the organization’s website, each identified risk is prioritized according to prevalence, detectability, impact and exploitability. Therefore, following the security principles of OWASP is crucial when building a modern web application.

Having briefly explained the basis of this report, it is time to go over the top 10 security risks and how they are mitigated in my individual project.

2. Injection

The injection (Injection, 2020) is an attempt by an attacker to send information to an application in a way which would change the meaning of commands being sent to an interpreter. In other words, injection means manipulating the operations that the system runs through specifically written input which interferes with the communication between the different modules of the application. The most common type of injections are the SQL-injections, in which the attacker uses open input to get access to information that should not be available to a regular user.

According to OWASP, preventing injection requires keeping data separate from commands and queries, namely by using a safe API or Object Relational Mapping (ORM, 2020). Input validation and Output encoding are also methods of preventing injections.

The way I have tackled this potential risk in my own project is exactly by using an ORM-based API which uses objects representing the storage repository and their methods. When executing queries, the system runs the input through multiple 'layers' of code, and thus mitigates the possibility of an SQL-injection.

3. Broken authentication

This problem usually occurs when authentication- and session-related functions are not implemented correctly. In that case, attackers are able to compromise passwords or session tokens, and steal other users' identities.

Preventing broken authentication (Broken Authentication, 2020) is usually done by implementing multi-factor authentication, avoiding default credentials, adding password requirements.

In my project, I use JSON Web Token (JWT, 2020) as a means of preventing broken authentication. It functions as a session hijacking defense, as it encodes the user's authentication data in a web token used to authorize all requests being made by them. Furthermore, in order to provide further security, passwords are encrypted on registration. That way, even if someone gets access to user passwords, they will not be able to use them.

4. Sensitive Data Exposure

If a web application uses sensitive data such as financial or healthcare information, it is of significant importance that this data gets properly protected. Otherwise, users might become a victim to credit card fraud, identity theft, etc. This is why additional protection needs to be implemented in the case of sensitive data transmission. The data is at risk in case it is transmitted in clear text or there are no security directives or headers present.

According to OWASP (Sensitive Data Exposure, 2020), the most important part is identifying the sensitive data. Afterwards, possible precautions that can be taken are encryptions, using standard algorithms, protocols and keys, etc.

My application does not ask the user for sensitive data such as financial or health information, this is therefore not a threat for the scope of my project in its current state. As mentioned previously, password encryption is present.

5. XML External Entities

XML Processing (XML External Entities, 2020) can be exploited in case the attacker can upload XML files or include dangerous content in a document. This is the case with many older XML processors which allow specification of an external XML entity. This can be used by an attacker to steal information, execute unauthorized requests or scan internal systems.

These flaws can be mitigated by using updated XML processors or preventing the user from being able to directly import external XML documents or insert untrusted data directly into existing ones. Using some kind of validation for incoming data is also a step in the right direction.

As for this current project, it does not use any type of XML parser or processor, therefore there have been no steps taken in order to prevent this risk.

6. Broken Access Control

Broken access control (Broken Access Control, 2020) is related to the restrictions on what unauthorized users can or cannot do in a website, more specifically when these are not properly implemented.

This is in case the user can bypass the access control check by modifying the URL or the page itself, manipulate metadata such as the JSON Web Token or abuse its invalidation. Furthermore, this risk is present if CORS is not properly configured and allows access to unauthorized requests.

In my own project, an HTTP Security Config is present, which specifically takes care of the actions a user with a specific role has access to. Moreover, while it is still in development, the web application's CORS is configured in such a way that it only accepts requests from one specific client.

7. Security Misconfiguration

According to OWASP, this is the most commonly seen issue with web applications (Security Misconfiguration, 2020). It includes factors such as improperly configured permissions, unnecessary features being installed, default accounts still being in use, error handling which reveals the stack trace, etc.

Minimizing the amount of features and frameworks of the application, regularly updating all configurations and including an automated process which verifies their effectiveness are some of the precautions which can be taken in order to mitigate this risk.

Personally, I have not included such automated process in my application, but try to use the latest version of the features to which I have access.

8. Cross-Site Scripting (XSS)

Cross-Site Scripting (Cross-Site Scripting, 2020) means that an attacker has the ability to execute scripts in the victim’s browser, hijacking their session or redirecting them to a malicious website.

There are three forms of XSS – Reflected (unvalidated user input as part of HTML output), Stored (stored user input which is viewed later by another user) and DOM (unsafe JavaScript frameworks/APIs) XSS. Through an XSS attack, a session can be stolen, an account can be hijacked, security can be bypassed, even a client-sided keylogger can be activated for the user.

Preventing XSS can be done by using a framework which automatically escapes it by design. According to OWASP, ReactJS is one of these frameworks. Other forms of mitigation are using context-sensitive encoding and enabling Content Security Policy.

9. Insecure Deserialization

This is one of the less common types of web attacks (Insecure Deserialization, 2020). However, it is still a worrying risk due to the fact that a potential breach by deserialization could allow the attacker to execute remote code.

Applications can be vulnerable to such attacks if they deserialize objects supplied by the user. This can lead to an attacker changing the logic of the application or the content of existing data structures, executing a remote piece of code, etc.

This risk can be mitigated by implementing integrity checks on serialized objects, deserializing in low privilege environments, monitoring network connectivity or alerting if a user is deserializing constantly.

This is a potential risk with my project, as with the current state of the application, I have not taken any precautions against insecure deserialization. This means that, potentially, information about a delivery could be stolen during deserialization of a delivery object through JSON in the browser.

10. Using components with known vulnerabilities

This is a risk mainly related to older versions of frameworks/libraries, where specific downsides have been identified and later patched. An application can be subject to an attack if maintenance/updates are done way too rarely, if the system does not get scanned for underlying vulnerabilities, if updated components do not get properly tested and approved, etc. (Using Components with Known Vulnerabilities, 2020)

The versions of every component in the application, be it client- or server-sided, should be monitored and updated regularly. Similarly to previously mentioned risks, limiting the amount of features and removing unused/unnecessary ones is also a step in the right direction.

I have used the latest available version of every component in my project, and am constantly educating myself on the modern methods of development for each separate module. The amount of features in my application is also fairly limited, as the scope of the project is pretty small, so this specific risk should not be an issue in that case.

11. Insufficient logging and monitoring

According to OWASP (Insufficient Logging & Monitoring, 2020), among the obvious ones like failed logins or transactions, things that should also be monitored and logged are warnings and errors with unclear messages, logs of APIs and applications, etc. Apart from logging, alerting should also be properly set up in order to identify a potential threat on time, before any damage can be done.

With the current version of my application, there is no monitoring being in place, as this is one of the secondary risks about the scope of the project. However, it can be implemented at a later stage, when other more significant risks are mitigated.

12. Conclusion

After learning about all these different types of security risks, I can see that there is much more to be careful with when it comes to web applications than I expected. Of course, this is part of the learning process I inevitably have to go through, and even if the current project I am working on does not fully prevent each and every one of these top 10 security risks, it will help me learn how to develop and maintain proper web applications in the future.

13. References

Broken Access Control. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control

Broken Authentication. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

Cross-Site Scripting. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS)

Injection. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A1_2017-Injection

Insecure Deserialization. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization

Insufficient Logging & Monitoring. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring

JWT. (2020, December). Opgehaald van JWT: https://jwt.io/

ORM. (2020, December). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping

OWASP. (2020, December). Opgehaald van OWASP: https://owasp.org/

Security Misconfiguration. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration

Sensitive Data Exposure. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure

Top 10 Web Application Security Risks. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/

Using Components with Known Vulnerabilities. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities

XML External Entities. (2020, December). Opgehaald van OWASP: https://owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_(XXE)