OWASP Top10 Security

CodexHub

Version table

Version	Date	Author(s)	Changes
1.0	April 2022	Florin Deleanu	First draft

Contents

Introduction

For any application which contains user data, security is an important concern that the developers need to constantly keep in mind. This document will list the ten risks and then assess them based on the following criteria:

- Is the risk applicable for the application, and why (or why not)?
- How did the risk impact your implementation? How does this affect the risk?

A01:2021-Broken Access Control

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.

This security concern is solved by the back-end validating if the user has the necessary role to perform different tasks based on the JWT token

A02:2021-Cryptographic Failures

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS)

The password is hashed before being stores in the database and is not exposed anywhere else. The only other sensitive information collected could be credit card data used for payments, which is taken care by a 3rd party provider such as Stripe.

A03:2021-Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

My application uses two types of databases: MySql and MongoDB. All of the database interaction for MySql is managed by Entity Framework in .NET Core which by design is secure. I don’t use any direct sql in the application either.

For the NoSql part, the backend performs checks based on the data inputted by users and no direct server-side javascript can be executed.

A04:2021-Insecure Design

Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation

I use the latest versions of Asp.NET Core and React. All the packages and dependencies installed are at the latest version.

A05:2021-Security Misconfiguration

The application might be vulnerable if the application is:

- Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords are still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, the latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
- The server does not send security headers or directives, or they are not set to secure values.
- The software is out of date or vulnerable

All of the software is up to date and the credentials need to be passed by environment variables. Error handling for the production build does not show stack trace information but only generic error messages.

A06:2021-Vulnerable and Outdated Components

You are likely vulnerable:

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If the software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.

- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries

All of the technology I use is extremely popular and at the latest version.

A07:2021-Identification and Authentication Failures

Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be authentication weaknesses if the application:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe

The application used a JWT implementation according to the official documentation. Weak passwords are validated by the back-end to match certain criteria (long,contain number,special characters etc.)

There is also a rate limiter for API requests to restrict possible brute force attacks.

A08:2021-Software and Data Integrity Failures

Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise

My application uses only verified NuGet and Node packages that are very popular, well-maintained and up to date.

A09:2021-Security Logging and Monitoring Failures

This category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time:

- Auditable events, such as logins, failed logins, and high-value transactions, are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally

This is of course a natural concern for any application. At the moment I don't intend on implementing logging past the normal level for debugging purposes.

A10:2021-Server-Side Request Forgery

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

Request from the front-end are validated by the back-end not only for valid paths, but also for valid url parameters

Conclusion

To conclude, I have listed all the risks, gave a definition on what they are (OWASP) and how I address them in my application (if they apply).

Bibliography

OWASP. (n.d.). Retrieved from https://owasp.org/Top10/A01_2021-Broken_Access_Control/