Security Research



2022

Jelle Manders

Research report

S3-DB03

Contents

[Summary 3](#_Toc122435015)

[Introduction 3](#_Toc122435016)

[What is a security problem? 3](#_Toc122435017)

[What are the most common security problems? 3](#_Toc122435018)

[Owasp top 10 3](#_Toc122435019)

[1. Broken Access Control 4](#_Toc122435020)

[2. Cryptographic Failures 4](#_Toc122435021)

[3. Injection 4](#_Toc122435022)

[4. Insecure Design 4](#_Toc122435023)

[5. Security Misconfiguration 4](#_Toc122435024)

[6. Vulnerable and Outdated Components 5](#_Toc122435025)

[7. Identification and Authentication Failures 5](#_Toc122435026)

[8. Software and Data Integrity Failures 5](#_Toc122435027)

[9. Security Logging and Monitoring Failures 5](#_Toc122435028)

[10. Server-Side Request Forgery(SSRF) 5](#_Toc122435029)

[Bibliography 5](#_Toc122435030)

# Summary

In this document I will look to the security aspect of my application. First I will describe the owasp top 10. Once I did that I will go deeper into 1 of the owasp problems that is the most related to my project. I will describe what this problem is and what you can do to prevent it. I will do this research according to the DOT framework.

# Introduction

The main question of this report is:

* ‘What is sql injection and how can you prevent it?

Before I can answer the main question I’ll have to answer some sub questions so that they clarify the main questions. These sub-questions are:

* ‘What is injection?
* ‘Why is Sql injection dangerous?’
* ‘How can you prevent Sql injection?’

# What is a security problem?

Software security issues are vulnerabilities that exist within a software system, allowing malicious actors to gain unauthorized access to sensitive information, disrupt operations, or cause harm. Examples of software security issues include unpatched security flaws, inadequate authentication and authorization measures, improper data validation, lack of encryption, and insecure coding practices.

# What are the most common security problems?

## Owasp top 10

The Open Web Application Security Project (OWASP) is an international organization that focuses on developing secure applications. To help organizations improve their capability to produce secure code, the OWASP Foundation has compiled a list of the most common security errors that are still occurring and easy-to-prevent. Additionally, they have created various open-source toolsets to identify security issues within projects. These tools, documents, teaching materials, guidelines, checklists, and other resources are all available to help organizations make their applications as secure as possible.

(Owasp, 2022)

### 1. Broken Access Control

Access control is an important security measure to ensure users are only granted the permissions they are intended to have. Common access control vulnerabilities include violation of the principle of least privilege, bypassing access control checks, insecure direct object references, missing access controls for API methods, elevation of privilege, metadata manipulation, and CORS misconfiguration. All of these can lead to unauthorized information disclosure, modification, or destruction of data, or performing functions outside the user's limits.

### 2. Cryptographic Failures

Data protection is key to secure data in transit and at rest. To ensure data security, it is important to check for weak cryptographic algorithms, protocols, keys, and initialization vectors. In addition, proper key management, validation of server certificates, and securely seeded randomness are essential. Proper hash functions and padding methods should also be used to prevent attacks.

### 3. Injection

Application security is critical for protecting user data. Attackers can take advantage of vulnerabilities when user-supplied data is not properly validated, filtered, or sanitized. Common injection vulnerabilities include SQL, NoSQL, OS command, ORM, LDAP, and EL/OGNL injection. The best way to protect against these threats is to include static, dynamic, and interactive application security testing tools into the CI/CD pipeline and perform source code review.

### 4. Insecure Design

Secure design is a culture and methodology that involves threat modeling and constant evaluation of security threats to ensure robust design and testing. A secure development lifecycle must also be employed, which includes secure design patterns, a secured component library, tooling, and threat modeling. These components must be integrated from the beginning of the software project, through the build and testing stages, and into the maintenance phase. OWASP's Software Assurance Maturity Model (SAMM) can be used to help structure secure software development efforts.

### 5. Security Misconfiguration

Application security is the practice of protecting applications from external threats and malicious attacks. It involves securing code, configuration and data, as well as protecting the application from vulnerabilities that could be exploited by attackers. This is achieved through a combination of secure development processes, secure coding practices, secure configuration, and security testing. Application security is an essential component of any organization's overall security posture and should be taken seriously.

### 6. Vulnerable and Outdated Components

Software vulnerability can be a serious security risk if components used on both the client and server side, such as the operating system, applications, and libraries, are not kept up to date. Regular vulnerability scanning and subscribing to security bulletins is essential, as is testing the compatibility of updated components. Additionally, configurations must be secured to ensure safety.

### 7. Identification and Authentication Failures

The application must protect against authentication-related attacks by ensuring that user identities are properly confirmed, authenticated, and managed during a session. Weaknesses such as automated attacks, default passwords, weak credential recovery, plain text passwords, missing or ineffective multi-factor authentication, exposed session identifiers, reused session identifiers, and incorrect session ID invalidation must all be addressed in order to protect against authentication-related attacks.

### 8. Software and Data Integrity Failures

Software and data integrity failures occur when code or infrastructure does not protect against integrity violations. Examples include insecure CI/CD pipelines, auto-update functionality, and insecure deserialization. Without sufficient integrity verification, attackers could potentially upload their own updates or modify objects or data.

### 9. Security Logging and Monitoring Failures

The OWASP Top 10 2021 category of Insufficient Logging & Monitoring helps to detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. To be protected, organizations must ensure that auditable events are logged, warnings and errors generate clear log messages, logs are monitored for suspicious activity, logs are stored securely, appropriate alerting thresholds and response processes are in place, penetration tests and scans trigger alerts, and the application can detect and alert for active attacks in real-time.

### 10. Server-Side Request Forgery(SSRF)

SSRF flaws occur when web applications fetch a remote resource without validating the user-supplied URL. This can allow an attacker to send crafted requests to unexpected destinations, even when protected by firewalls, VPNs, and other network access control lists. The increasing use of cloud services and complex architectures has made SSRF more common and more severe.

(Owasp top 10, 2022)

## What is injection?

# Injection is a security vulnerability that occurs when an attacker is able to inject malicious code into a program or system. This code is then executed, giving the attacker full control of the system, allowing sensitive information or data stored on the system to be modified or deleted. There are various types of injection attacks such as SQL injection, cross-site scripting (XSS), command injection, OS command injection and LDAP injection. To protect against injection attacks it is important to take a number of security precautions when developing software. This means ensuring inputted user data is properly sanitized to prevent malicious code from entering, and that strict access control is applied to prevent unauthorized users from gaining access to sensitive data. Additionally, it is important to regularly install updates and patches for software and systems to prevent them from being vulnerable to known exploits.

(Code Injection, 2023)

## Forms of injection

### SQL injection

SQL injection is a type of security vulnerability that occurs in programs or systems that use an SQL database. It involves an attacker crafting a specially-formatted SQL query that contains malicious code, and then sending it through the vulnerable program or system to the database. When the database processes the query, the code is executed, giving the attacker access to the data in the database or allowing them to manipulate the data.

An attacker can carry out a SQL injection attack in several different ways. A common technique is to enter malicious code into a form field designed to accept user input. For example, a sign-up form may have fields for the user's username and password. An attacker can input their own SQL code into the username field, and when the program or system processes the query, the code is executed, potentially giving the attacker access to the system. Another widely used technique is exploiting vulnerabilities in the database itself. For example, an attacker can use a SQL injection attack to bypass authentication checks and gain access to the database without valid username and password. This allows the attacker to view, modify or delete data in the database.

Example of Sql injection:

// Attackers code:

username: ' OR 1=1;

// Original SQL query:

SELECT \* FROM users WHERE username = '$username' AND password = '$password';

// Modified SQL query:

SELECT \* FROM users WHERE username = '' OR 1=1; --' AND password = '$password';

### Cross Site Scripting (XSS)

Cross-site scripting (XSS) is a security flaw in a web application. The problem is caused when the input the web application receives (such as cookie, URL, request parameters) is not properly processed and this results in output being sent to the end user. Through this bug in the website, malicious code (JavaScript, VBScript, ActiveX, HTML, Flash, etc.) can be injected. This can be used to view session cookies, take over a user's session, enhance the functionality of a website, or carry out unintended actions for a user. At first, the acronym CSS was used to refer to cross-site scripting. To avoid confusion with Cascading Style Sheets and Content Scramble System, the acronym XSS was soon used, where the X stands for cross (English word for cross). Often, cross-site scripting is used in combination with Phishing[source?], where the end user is tricked into clicking on a link prepared with XSS in an email message. Once the person clicks on the link, the XSS attack is carried out.

Example of XSS:

Input attacker:

<script>

alert("XSS");

</script>

// Original HTML page:

<div id="comments">

<h2>Comments</h2>

<ul>

<li>Comment 1</li>

</ul>

</div>

// Edited HTML page:

<div id="comments">

<h2>Comments</h2>

<ul>

<li>Comment 1</li>

<li>

<script>

alert("XSS");

</script>

</li>

</ul>

</div>

(Cross Site Scripting, 2023)

## How to prevent injection

One way to prevent injection vulnerabilities is to use a combination of server-side and client-side validation techniques. Server-side validation checks user input on the server before it is processed, while client-side validation checks user input in the client's web browser before it is sent to the server. This can help prevent attackers from entering malicious code into the system through user input. Another way to prevent injection vulnerabilities is to use parameterized or prepared queries. With these techniques, user input is separated from the SQL code, preventing attackers from inserting malicious code into the SQL query. This can provide protection against SQL injection attacks, a common type of injection attack. Additionally, developers can use web application firewalls (WAFs) to protect themselves from injection attacks. WAFs check incoming traffic and block requests that may contain malicious code. This can help prevent attackers from entering malicious code into the system through user input.

(Sql injection, 2023)

## Is my application vulnerable to injection

Because I use SonarCloud I’m always aware of vulnerabilities of my code. In the picture below you can see how many vulnerabilities my application has

Graphical user interface, application

Description automatically generated

The following code uses an ORM to insert a user into a database:

`\_context.Add(Beer)`

This code will generate and execute a safe SQL query that will protect against injection vulnerabilities by automatically generating and executing a safe SQL query based on the objects used in the code. An Object Relational Mapping (ORM) can provide protection against injection vulnerabilities by automatically generating and executing safe SQL queries based on the objects used in the code. This can help prevent attackers from inserting malicious SQL code into the query, providing protection against SQL injection attacks.

# Bibliography

*Code Injection*. (2023, 1 11). Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Code\_injection

*Owasp*. (2022, 12 13). Opgehaald van FHICT: https://fhict.instructure.com/courses/12517/pages/secure-web-development?module\_item\_id=835979

*Owasp top 10*. (2022, 12 20). Opgehaald van Owasp.org: https://owasp.org/Top10/