Niels Roefs

Abstract

Report containing the owasp top 10 and how I deal with them.

OWASP Top 10

Individual Project

Contents

[Introduction 2](#_Toc170042558)

[OWASP Top 10 Risks 2](#_Toc170042559)

[Broken Access Control 2](#_Toc170042560)

[Cryptographic Failures 2](#_Toc170042561)

[Injection 3](#_Toc170042562)

[Insecure Design 4](#_Toc170042563)

[Security Misconfiguration 4](#_Toc170042564)

[Vulnerable and Outdated Components 5](#_Toc170042565)

[Identification and Authentication Failures 5](#_Toc170042566)

[Software and Data Integrity Failures 6](#_Toc170042567)

[Security Logging and Monitoring Failures 6](#_Toc170042568)

[Server-Side Request Forgery (SSRF) 7](#_Toc170042569)

# Introduction

The **Open Web Application Security Project (OWASP)** is a nonprofit foundation dedicated to improving the security of software. OWASP's primary focus is on web application security, providing free and open resources to help organizations identify and mitigate security risks. One of OWASP's most well-known projects is the OWASP Top 10, which represents a broad consensus about the most critical security risks to web applications.

In this analysis, I will provide a detailed overview of each risk listed in the OWASP Top 10, along with the mitigation strategies that I employ to address these risks in my work. This proactive approach ensures that the applications I develop or manage are resilient against common and critical security threats.

# OWASP Top 10 Risks

## Broken Access Control

**Description:**Broken access control occurs when users can perform actions they are not authorized to. This can lead to unauthorized access to sensitive data or functionality.

**Risk:**The risk associated with broken access control is high because it directly affects the core security principles of confidentiality, integrity, and availability. Unauthorized users gaining access to sensitive data can lead to serious issues.

**Likelihood:**The likelihood of broken access control vulnerabilities is quite high. For most endpoints in my application, the user needs to be logged in and a few endpoints, they need to have the admin role.

**Mitigation:**

* Implement role-based access control (RBAC).
* Use the least privilege principle, granting only necessary permissions.
* Regularly audit and review access controls.
* Use secure coding practices to avoid bypassing authorization checks.

**Implementation:**I have implemented a role-based access control with JWT tokens. In the API gateway, it checks if the JWT token is a valid token and if the user who is trying to access the endpoint, has the right roles.

## Cryptographic Failures

**Description:**This category refers to failures in cryptographic operations, such as improper use of cryptographic algorithms, poor key management, or inadequate protection of data in transit and at rest.

**Risk:**  
The risk associated with cryptographic failures is critical because it directly impacts the confidentiality and integrity of sensitive data. When encryption is improperly implemented or managed, sensitive data such as personal information, financial data, and proprietary business information can be exposed to unauthorized parties.

**Likelihood:**The likelihood can be significant due to several factors. Misconfigurations and mistakes and easily occur without the right knowledge. But there is a low chance of it happening in my application.

**Mitigation:**

* Use strong, industry-standard cryptographic algorithms and protocols.
* Ensure proper key management and secure storage.
* Encrypt sensitive data both in transit and at rest.
* Regularly update and patch cryptographic libraries.

**Implementation:**I am making use of Azure Key Vault for Key/Secret management. Azure Key Vault provides secure storage for cryptographic keys, secrets, and certificates, ensuring they are managed and stored using industry-standard practices. And I am using Azure Cosmos DB for PostgreSQL. Whenever data is ingested into a node, Azure Cosmos DB for PostgreSQL secures your data by encrypting it in-transit with Transport Layer Security (TLS) 1.2. Encryption (SSL/TLS) is always enforced and cannot be disabled, ensuring data is protected from interception during transmission. The Azure Cosmos DB for PostgreSQL service uses the FIPS 140-2 validated cryptographic module for storage encryption of data at-rest. Data, including backups, is encrypted on disk using the AES 256-bit cipher included in Azure storage encryption.

## Injection

**Description:**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 unauthorized data. This can lead to data breaches, data loss, or unauthorized access to system functionality.

**Risk:**  
Injection vulnerabilities pose a severe risk because they can allow attackers to execute arbitrary code or queries, potentially leading to complete system compromise.

**Likelihood:**Because my application makes use of Entity Framework and ASP.NET for validation, the likelihood of injection vulnerabilities is lower compared to applications that do not use such protective measures.

**Mitigation:**

* Use parameterized queries and prepared statements.
* Validate and sanitize all user inputs.
* Use ORM frameworks to avoid direct database access.
* Employ input validation libraries.

**Implementation:**Entity Framework (EF) uses parameterized queries by default, which helps prevent SQL injection attacks. By using parameterized queries, EF ensures that user input is treated as data rather than executable code. ASP.NET provides built-in validation attributes that can be applied to models to enforce validation rules. I use data annotations to validate input data at the model level, ensuring that only valid data is accepted by the application.

## Insecure Design

**Description:**   
Insecure design refers to flaws that arise from missing or improper security controls during the design phase of software development.

**Risk:**The risk associated with insecure design is substantial because it can lead to systemic vulnerabilities that are deeply embedded in the application's architecture.

**Likelihood:**The likelihood of insecure design vulnerabilities is relatively high, especially in environments where security is not a primary consideration during the design phase. Considering I haven’t specifically looked at security when designing the application, so the likelihood remains relatively high.

**Mitigation:**

* Incorporate secure design principles early in the SDLC.
* Conduct threat modeling and risk assessments.
* Regularly review and update design documents with security in mind.
* Implement security training for developers.

**Implementation: N/A**

## Security Misconfiguration

**Description:**   
Security misconfiguration is the most seen issue. This can occur at any level of the application stack, including the web server, application server, database, and framework.

**Risk:**The risk associated with security misconfiguration is high because it can expose critical system components and sensitive data to attackers.

**Likelihood:**While the inherent likelihood of security misconfiguration vulnerabilities is relatively high due to factors such as human error, complexity of systems, lack of standardization, and frequent changes, the specific measures you have implemented help reduce this likelihood. By ensuring that only generic error messages are shown to users, disabling unused features, and not enabling testing accounts in production, the likelihood of security misconfiguration in your application is lowered compared to a typical environment where these precautions are not taken.

**Mitigation:**

* Implement a repeatable hardening process.
* Use secure defaults and disable unused features.
* Regularly patch and update all components.
* Automate configuration management and deploy static analysis tools.

**Implementation:**I have implemented proper error handling to ensure users only see generic error messages so that sensitive stack traces are not displayed. Unused features are not enabled which reduces the attack surface by limiting the number of potential entry points for attackers. Testing accounts are not enabled in the production environment. This prevents unauthorized access through default or testing credentials that might be overlooked during deployment.

## Vulnerable and Outdated Components

**Description:**   
The risk of vulnerable and outdated components arises when software dependencies, libraries, frameworks, or third-party components used in an application are not kept up-to-date or contain known security vulnerabilities.

**Risk:**The risk associated with vulnerable and outdated components is significant. Exploiting known vulnerabilities in these components can lead to unauthorized access, data breaches, and potential compromise of the entire application or system. Attackers actively target such weaknesses because they provide a relatively easy way to gain access and escalate their attacks.

**Likelihood:**The likelihood of vulnerabilities from outdated components is mitigated by the measures I have implemented:

* By regularly updating NuGet packages, I ensure that my dependencies are patched with the latest security fixes and updates, reducing the likelihood of known vulnerabilities.
* Scanning for security vulnerabilities using SonarQube helps identify and mitigate potential risks in third-party components before they can be exploited.
* Removing unused dependencies reduces the attack surface and minimizes the potential impact of vulnerabilities that may exist in those components.

## Identification and Authentication Failures

**Description:**   
These failures occur when functions related to user authentication and session management are incorrectly implemented, leading to compromised credentials and session hijacking.

**Risk:**The risk associated with identification and authentication failures is critical. Weak or compromised authentication mechanisms can lead to unauthorized access, account takeovers, and exposure of sensitive user information. Attackers may exploit these vulnerabilities to gain illegitimate access to systems or impersonate legitimate users.

**Likelihood:**The likelihood of identification and authentication failures is reduced by the security measures I have implemented:

* **Hashed and Salted Passwords:** Asp.net Identity ensures that passwords are securely hashed and salted, making it computationally impractical to reverse-engineer passwords even if the hashed values are compromised.
* **JWT Tokens:** I use JWT (JSON Web Tokens) for authentication provides a stateless and secure way to authenticate users and transmit claims between parties. JWT tokens, when implemented correctly with proper validation and expiration, reduce the likelihood of token-related vulnerabilities.

## Software and Data Integrity Failures

**Description:**   
Software and data integrity failures relate to code and infrastructure that do not protect against integrity violations. This can include untrusted code, data from untrusted sources, and unvalidated dependencies.

**Risk:**If I use untrusted packages, they could inject malicious code into my application, potentially leading to a data breach.

**Likelihood:**Since my application does not rely on any untrusted sources or code, and I have not implemented auto-updating of packages that could be exploited for injecting malicious code, the risk of such an incident is minimized.

**Mitigation:**

* Use digital signatures to verify the integrity of code and data.
* Implement code signing for software releases.
* Validate data from third-party sources.
* Employ version control and CI/CD tools with security checks.

**Implementation: N/A**

## Security Logging and Monitoring Failures

**Description:**   
Failure to log and monitor activities can lead to undetected breaches and delayed responses to incidents.

**Risk:**  
The risk associated with security logging and monitoring failures is significant. Without comprehensive logging and monitoring, organizations may fail to detect early indicators of security breaches, malicious activities, or policy violations. This can lead to delayed incident response, prolonged unauthorized access, and increased impact from security incidents.

**Likelihood:**  
Security logging and monitoring are more likely to fail if there aren't specific plans to deal with these risks:

* Logging warnings and errors is good, but not logging important security events like logins and failed logins means we might miss signs of potential security problems.

## Server-Side Request Forgery (SSRF)

**Description:**SSRF flaws occur when a web application is fetching a remote resource without validating the user-supplied URL. This can lead to the disclosure of internal systems and data.

**Risk:**The risk associated with SSRF is significant. Exploiting SSRF vulnerabilities can allow attackers to bypass firewalls and access internal systems, execute arbitrary code, or perform actions on behalf of the server, potentially leading to data breaches, service disruptions, or further exploitation of the network infrastructure.

**Likelihood:**The likelihood of SSRF vulnerabilities is mitigated by the security measures inherent in the ASP.NET framework:

* ASP.NET includes request validation features that help mitigate certain types of SSRF attacks by validating and sanitizing user input.
* The framework provides mechanisms to validate URIs and restrict requests to trusted domains or IP addresses.
* I have implemented input validation using ASP.net, which can help against ssrf attacks.