Security-Assessment

Security test of Example - Web Application

Recipient:

Example GmbH
Example Str. 12
1234 Example

Classification: confidential
Date: 21.08.2024
Version: **1.0**

Contact at A1 Digital International GmbH:

Alice Codex
ask.security@a1.digital
+431234567890
Department Security

Lassallestraße 9, A-1020 Wien

1 Change record

Date	Version	Author	Description
12.08.2024	0.1	Bob Binary	Initial Creation
16.08.2024	0.8	Alice Codex	First Completion
19.08.2024	0.9	Trent Trustworthy	Review
21.08.2024	1.0	Alice Codex	Published

Table 1: Change record

Contents

1 Change record 2

2 Management Summary 4
2.1 Results 4
2.2 Recommended next steps 4
2.3 Overview of weaknesses 6
2.4 Disclaimer 8

3 Scope 9
3.1 Systems tested 9
3.2 User accounts used 9

4 Procedure 10
4.1 Risk assessment according to CVSSv3 10

5 Identified weaknesses 11
5.1 Blind SQL Injection 11
5.2 Cross-Site Request Forgery (CSRF) 15
5.3 Subdomain Takeover 17
5.4 Reflected Cross-Site-Scripting (XSS) 19
5.5 Outdated Tomcat Installation 22
5.6 Missing Security Headers 24
5.7 Weak SSL/TLS Configuration 26

6 Appendix 28
6.1 Contact persons 28
6.2 CVSS v3.0 metrics 29
6.3 Text representation of CVSS v3.0 scores 31
6.4 List of Tables 32
6.5 List of Figures 32
6.6 OWASP Testing Guide Version 4.2 33

7 Imprint 38

2 Management Summary

The results of the security test are summarised briefly below. More detailed descriptions of the individual specific aspects with references to additional resources as well as recommended countermeasures can be found in chapter 5.

2.1 Results

A blind SQL injection vulnerability was detected in the login function of the webshop, which allowed unauthorized users to access, modify or delete user and product data stored in the database. This vulnerability could be used to compromise the data of over 1,000 shop users, including sensitive or personally identifiable information like addresses and password hashes.

The subdomain `takeover.example.com` had a `CNAME` set to `exampletrafficmanager.trafficmanager.net` at the time of the assessment, which was not allocated, and could therefore be registered via Microsoft Azure. This means that the subdomain `takeover.example.com` could fall under the control of attackers and be used for further attacks, like phishing campaigns.

A reflected cross-site scripting (XSS) vulnerability was identified where malicious JavaScript code could be injected into the application. Attackers would be able to steal session information by successfully exploiting this vulnerability and use it to take over other users' accounts.

The webshop system appeared to be using an outdated version of Tomcat that had at least one known vulnerability. This version contains known weaknesses which may allow access to users' data.

The functionality of the webshop that allows users to edit their profile did not have cross-site request forgery protection. As a result, attackers have the ability to cause logged-in victims to take actions on their account without their knowledge or consent. For example, attackers could change the victim's email address to subsequently change the password and take over the account.

It was determined that the affected systems used insecure SSL/TLS configurations. An attacker with access to the network traffic could potentially decrypt the transmitted packets, and thus get access to sensitive user usernames and passwords.

It was discovered that affected web applications had not consistently implemented common security headers. Setting security headers can increase the overall security of a web application and make it more difficult for attackers to carry out attacks such as cross-site scripting (XSS) or man-in-the-middle.

2.2 Recommended next steps

Recommendations for the next 3 months:

- The SQL injection vulnerability should be resolved by using prepared statements in all queries.
- It should be verified that no DNS CNAME records are pointing to unregistered domains.
- It should be evaluated whether recommended security headers can consistently be set for all web applications.

Recommendations for the next 6 months:

- It should be ensured that all software in use is up-to-date.
- All user input should be validated and properly encoded before being output back to a user (Input

Validation, Output Encoding).

- All authenticated user interactions should be equipped with CSRF protection.
- HTTPS and other encrypted protocols should be configured to only support secure and modern cipher, key exchange and MAC algorithms.

Recommendations for the next 12 months:

- An update process should be established for all software in use.
- The newly established security procedures should be tested for effectiveness.

2.3 Overview of weaknesses

The following table provides an overview of the identified weaknesses and an estimate by A1 Digital International GmbH of the effort required to implement countermeasures. Figure 1 shows a schematic representation of the identified weaknesses.

Weakness	Risk (accoring to CVSS3)	Countermeasures
Blind SQL Injection	Critical (10.0)	Medium
Cross-Site Request Forgery (CSRF)	High (7.1)	Complex
Subdomain Takeover	Medium (6.5)	Easy
Reflected Cross-Site-Scripting (XSS)	Medium (6.1)	Medium
Outdated Tomcat Installation	Medium (5.9)	Medium
Missing Security Headers	Low (3.7)	Medium
Weak SSL/TLS Configuration	Low (3.7)	Easy

Table 2: Overview of weaknesses

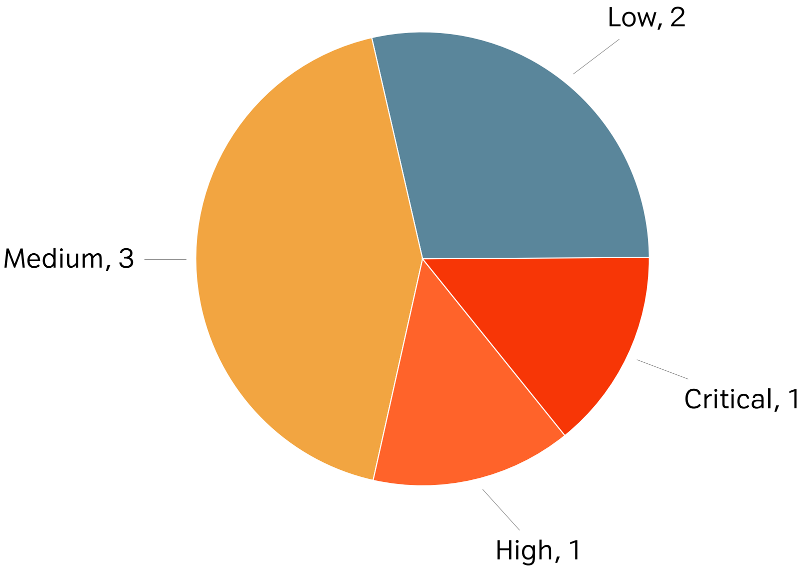


Figure 1: Overview of the identified weaknesses

2.3.1 Weakness categorisation

A coarse categorisation of the identified weaknesses was made to get an overview of the areas in which the most security-relevant findings were identified. The categories of weaknesses are as follows:

- **Configuration Issue**: Errors in the configuration of software or hardware components.
  - If repeated weaknesses have been identified within this category, training for system administrators on how to securely configure the components they support can help.
- **Outdated Software**: Outdated software components with known security-relevant problems.
  - If outdated software is a frequently identified problem, it is recommended to establish a continuous update and patch management process to install security-critical updates in a timely manner.
- **Input Validation/Output Encoding**: Missing validation of user inputs or missing correct encoding of outputs of the software.
  - Frequent errors in this category are likely related to a lack of secure coding training. Regular secure coding training for software developers could increase security and software quality.
- **Other**: Findings that do not fall into one of the three categories above.

The following table identifies the categorisation of weaknesses within the identified findings.

Weakness	Category
Blind SQL Injection	Input Validation/Output Encoding
Cross-Site Request Forgery (CSRF)	Other
Subdomain Takeover	Configuration Issue
Reflected Cross-Site-Scripting (XSS)	Input Validation/Output Encoding
Outdated Tomcat Installation	Outdated Software
Missing Security Headers	Configuration Issue
Weak SSL/TLS Configuration	Configuration Issue

Table 3: Weakness categorisation

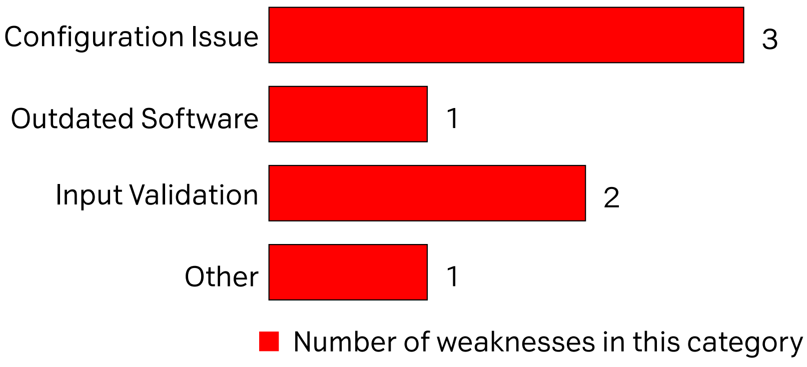


Figure 2: Weakness categorisation

2.4 Disclaimer

The effort for this test was estimated using a time box approach, i.e., only weaknesses within the agreed time window were identified. The aim was to identify and document as many security-relevant weaknesses as possible in the systems being tested. However, we do not assume any liability for completeness of the findings listed in the report.

The test provides a snapshot at the time of the security assessment, so future IT security risks cannot be derived from it.

3 Scope

Example GmbH commissioned A1 Digital International GmbH to perform a security test of the systems listed below.

The security test took place between 12.08.2024 and 16.08.2024.
A more detailed description regarding the procedure can be found in chapter 4.

3.1 Systems tested

The following systems were considered within the assessment.

IP	Hostname
203.0.13.64	www.example.com
203.0.13.65	shop.example.com

Table 4: Systems tested

3.2 User accounts used

No accounts for the web applications were provided.

To perform additional security checks, user accounts were created in the webshop (shop.example.com) during the assessment using usernames starting with **A1SecurityAssessment**.

The aforementioned **users accounts must be deleted / deactivated** after the security assessment.

4 Procedure

To cover the widest possible range of possible weakness categories, the test was conducted following the Open Web Application Security Project (OWASP) Testing Guide Version 4 (see chapter 6.6). The aim was to identify all security-relevant weaknesses that were present in the systems at the time of the test.

A number of criteria were defined in advance to enable classification of penetration tests that have been carried out. The following figure is based on the study ”implementation concept for penetration tests”[1] from the BSI and is intended to reflect the procedure within this test.

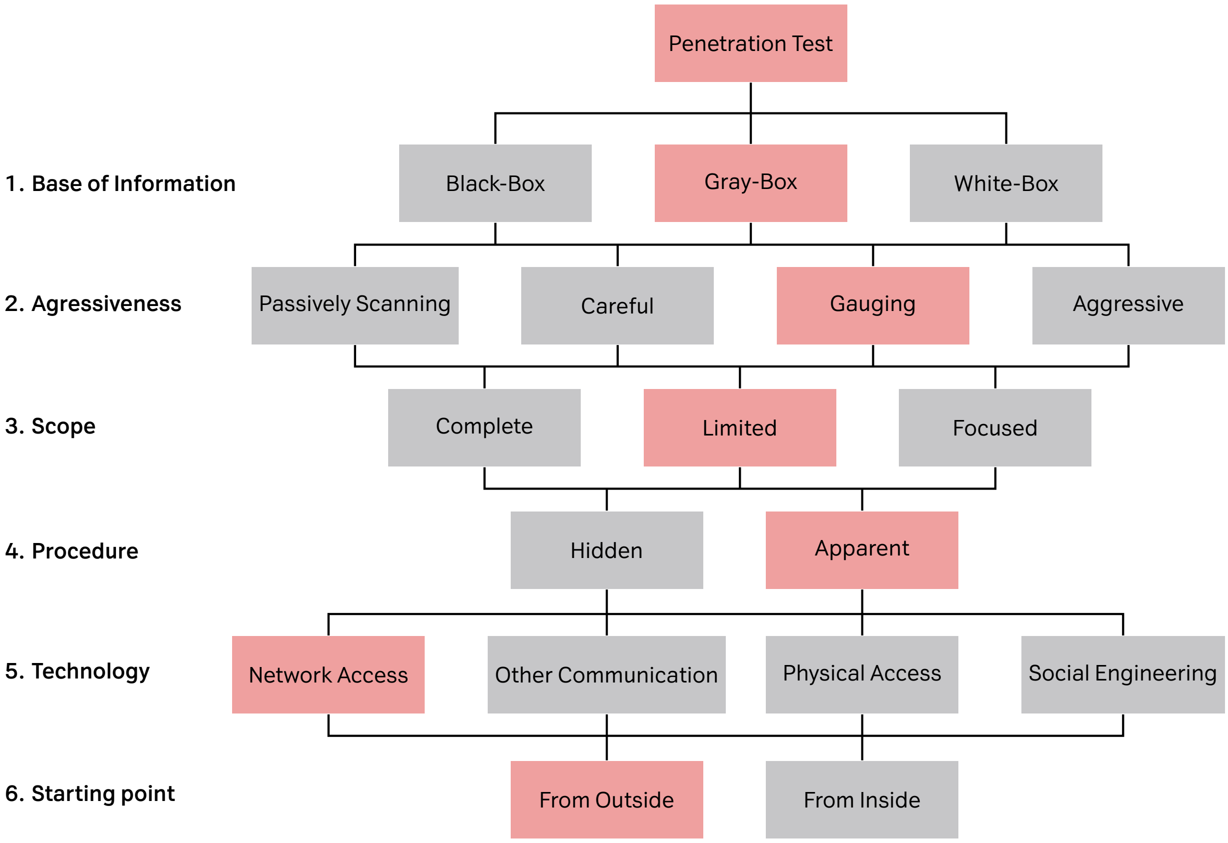


Figure 3: Implementation concept for penetration tests

4.1 Risk assessment according to CVSSv3

The Common Vulnerability Scoring System (CVSS) provides the ability to identify and score the underlying characteristics of a weakness. The result is a numerical value that can range between **0.0 and 10.0**, with **10.0** being the highest and thus most critical value. For a detailed description of the CVSS metrics, see 6.2. To be able to express the risk in words, five different value ranges are defined, which are described in the chapter 6.3. Accordingly, a risk can be classified as **”none”, ”low”, ”medium”, ”high”** and **”critical”**.

[1] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Penetrationstest/penetrationstest.pdf

5 Identified weaknesses

The weaknesses identified during the test are described below and assigned a risk rating. This risk assessment is carried out according to the CVSSv3 standard and was performed by the assessor to the best of his knowledge and belief. The risk assessment may therefore differ from the customer's assessments, as in most cases the assessor does not have sufficient background knowledge to perform a specific business risk assessment.
Each identified weakness described includes recommended countermeasures and references to external resources for further information.

5.1 Blind SQL Injection

CVSSv3 Score	**10.0 (Critical)**
CVSSv3 Vektor String	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H (show in first.org)

Affected Systems

- shop.example.com (203.0.13.65:443 (TCP))

Description

During the security assessment, a blind SQL injection vulnerability was detected in the login function of the webshop, which allowed unauthorized users to access, modify or delete user and product data stored in the database. This vulnerability could be used to compromise sensitive and personally identifiable information of over 1,000 shop users, including their addresses and password hashes.

Recommendations

- Most SQL injection vulnerabilities can be prevented by using parameterized queries (also known as prepared statements) instead of string concatenation within the query.
  - Prepared statements separate the SQL queries to the user supplied parameters they receive, making it impossible to escape the query and modify its purpose.
  - This should be done in all SQL queries used in all the applications of the company to ensure that no endpoint remains vulnerable.
- If the use of prepared statements is not possible in this application, ensure that all user input is properly sanitized before using it within an SQL query.
  - More information about SQL injection attacks and how to fix them can be found in the resources.

Technical Description

An **SQL injection** is a web application vulnerability that allows attackers to send queries directly to the database and therefore gain unauthorized access to it. The vulnerability occurs when the user's input data is not sufficiently validated on the server side and is passed directly to the database. The following illustration shows an example of how an **SQL injection** can be exploited.

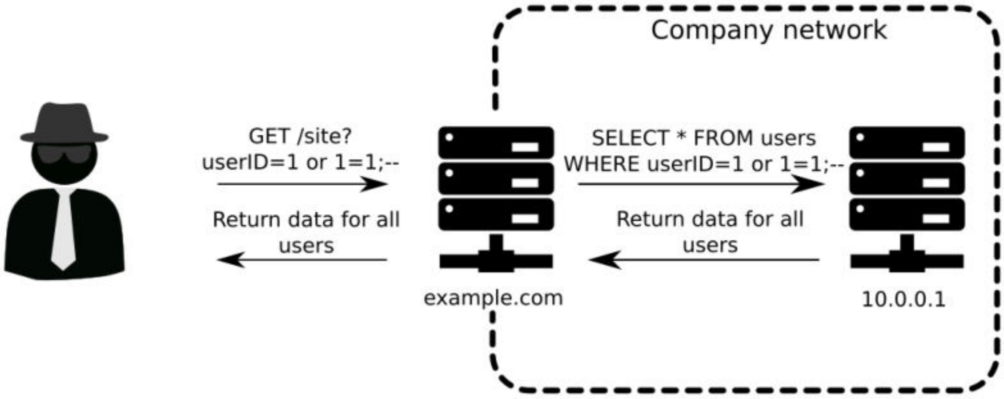
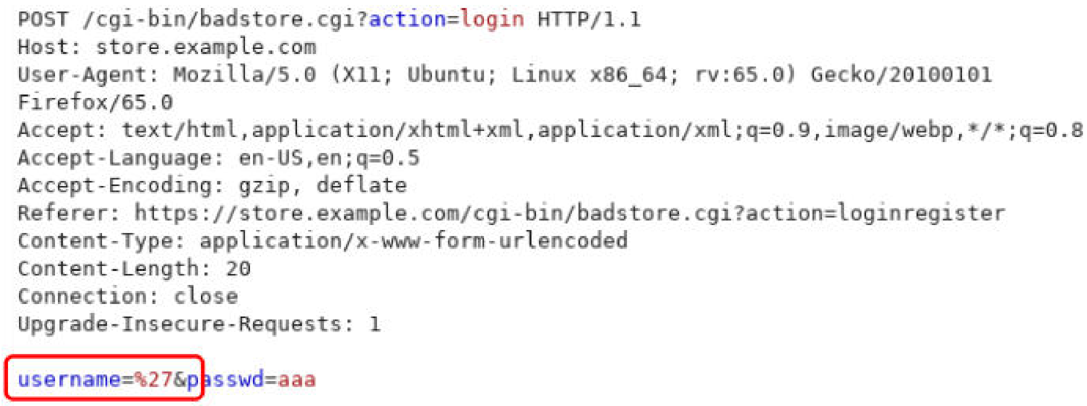


Figure 4: All user data is queried by exploiting an SQL injection vulnerability

During the assessment, it was detected that the login function of the webshop was affected by an SQL injection vulnerability. An attacker could send a crafted request with special characters on the **username** POST parameter that would modify the intended SQL query and allow **running arbitrary read-queries on the SQL server**. As there was no direct output from the query other than whether the user successfully logged in or an error was caused, this can be classified as error based SQL injection.

The following figure identifies the request to the application, which causes an SQL error:

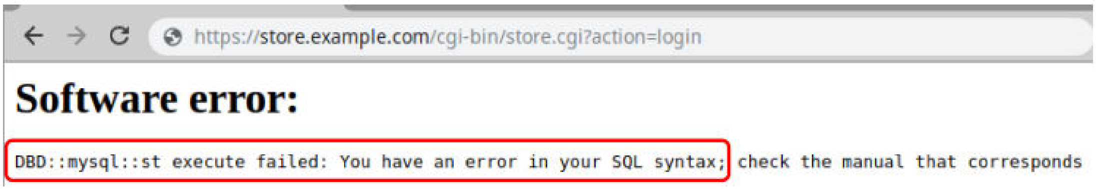


```
POST /cgi-bin/badstore.cgi?action=login HTTP/1.1
Host: store.example.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101
Firefox/65.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://store.example.com/cgi-bin/badstore.cgi?action=loginregister
Content-Type: application/x-www-form-urlencoded
Content-Length: 20
Connection: close
Upgrade-Insecure-Requests: 1

username=%27&passwd=aaa
```

Figure 5: Request to the application which causes an SQL error

The next figure demonstrates the SQL syntax error in the application caused by the above query, indicating that the request was not properly sanitized and broke the query process:



https://store.example.com/cgi-bin/store.cgi?action=login

Software error:

```
DBD::mysql::st execute failed: You have an error in your SQL syntax; check the manual that corresponds
```

Figure 6: Response showing the SQL error store.example.com

In the case of this vulnerability, no valid user account was necessary in order to exploit this vulnerability.

The following command allowed to log into the application without knowing the password of a user:

```
curl -X POST -F 'username=admin%27AND%271%27%3D%271%27---' -F 'passwd=aaa'
'https://shop.example.com/cgi-bin/badstore.cgi?action=login'
```

While attackers cannot retrieve the direct output of the query, they can enumerate their result (for example a dump of the whole database), character by character. This requires a large amount of queries, but can be automated with tools like **sqlmap**. With this, it was possible to extract the data of more than 1000 store users, and to access full names, address details and hashed passwords:

```
Database: storedb
Table: userdb
[6 columns]
+----------+--------------+
| Column   | Type         |
+----------+--------------+
| email    | varchar(40)  |
| fullname | varchar(50)  |
| address  | varchar(50)  |
| passwd   | varchar(32)  |
| pwdhint  | varchar(8)   |
| role     | char(1)      |
+----------+--------------+
```

As shown below, more than 1000 user data points are available in the database:

```
Database: storedb
+--------+---------+
| Table  | Entries |
+--------+---------+
| userdb | 1241    |
+--------+---------+
```

Furthermore, it was possible to chain multiple queries, including `INSERT`, `UPDATE` and `DELETE` queries, using a semicolon in the username parameter. Due to this, an attacker could arbitrarily modify or delete any data stored in the database.

Additional Information / References

- https://owasp.org/www-community/attacks/SQL_Injection
- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

5.2 Cross-Site Request Forgery (CSRF)

CVSSv3 Score	**7.1 (High)**
CVSSv3 Vektor String	CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:L/CR:H/IR:H (show in first.org)

Affected Systems

- shop.example.com (203.0.13.65:443 (TCP))

Description

At the time of testing, the functionality of the webshop that allows users to edit their profile did not have cross-site request forgery protection. As a result, attackers have the ability to cause logged-in victims to take actions on their account without their knowledge or consent. For example, attackers could change the victim's email address to subsequently change the password and take over the account.

Recommendations

- CSRF protection should be implemented.
  - This is usually solved using so-called CSRF tokens, which are assigned to a session and are sent with every write request as a body or header value and validated on the server side.
  - Since attackers do not know the value of the CSRF token, they no longer have the opportunity to carry out this attack.
- Another possibility would be to validate the 'origin' of the request.
  - The Referer or Origin HTTP headers can be used to identify where a request comes from.
  - Ensuring that the request comes from a trusted site can be used to prevent this kind of attack.
- It should be evaluated whether session-relevant cookies can be equipped with the `SameSite` attribute.
  - This cookie attribute prevents the browser from sending cookies if the requests come from external sites.
- Changing security-critical information, like the email address or the password, should only be possible after prior entry of the current password.
  - If this is not possible, after the change of the email address, an information mail should be sent to the old email address with the option to revert.

Technical Description

Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to trick users into performing actions they do not want to perform.

To do so, attackers must have control over a (possibly third-party) website that is used by the victim, or lure the victim onto said website. Once visited, this site contains malicious code that performs requests to the affected application in the background of the browser. If the victim is logged into the vulnerable service when this happens, these requests contain the session information of the victim and are accepted by the server as if done by the victim itself. Due to this, an attacker can make changes to the user's account or perform other actions in their name without the user's knowledge or consent. The following example will illustrate the exploitation of a CSRF vulnerability:

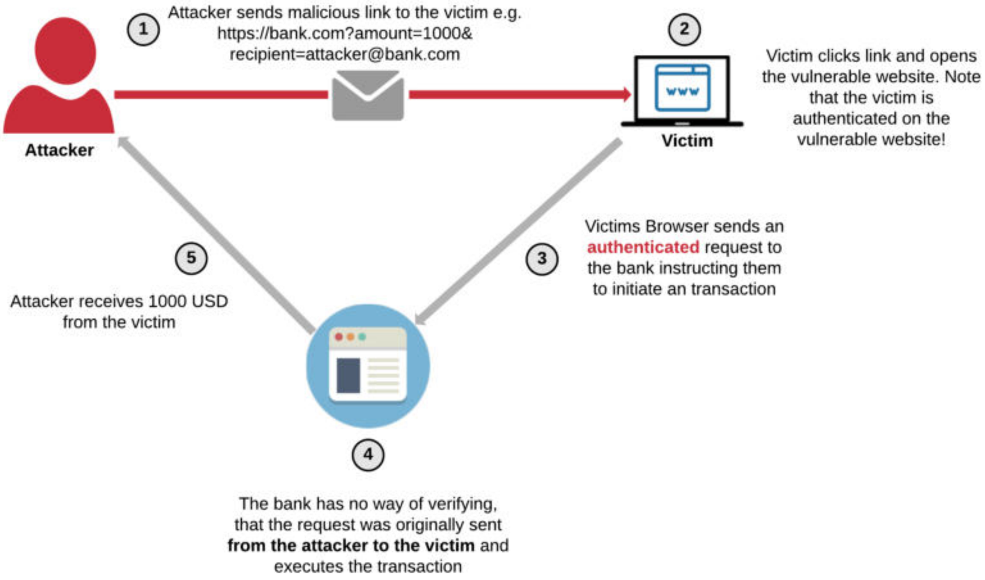

1
Attacker sends malicious link to the victim e.g. https://bank.com?amount=1000& recipient=attacker@bank.com
Attacker
2
Victim
Victim clicks link and opens the vulnerable website. Note that the victim is authenticated on the vulnerable website!
3
Victims Browser sends an authenticated request to the bank instructing them to initiate an transaction
4
The bank has no way of verifying, that the request was originally sent from the attacker to the victim and executes the transaction
5
Attacker receives 1000 USD from the victim

Figure 7: Illustration of Cross-Site Request Forgery

The email change function of the webshop application did not have CSRF protection implemented. This allows attackers to create a malicious website that, once visited, would send a request to the example website in the name of the user, and change the user's email to the email of the attacker. Then, the attacker could use the password reset functionality to reset the user's password and take over the account.

The following form demonstrates the attack:

```
<form action="https://shop.example.com/account/edit_profile" method="post" name="main">
<input type="hidden" name="email" value="attacker@evil.com">
<input type="hidden" name="btn_save" value="Save">
</form><script>document.main.submit();</script>
```

If a logged-in user visited a website that contained this code, their email on the webshop account would be changed to `attacker@evil.com`, which could then lead to a take-over of their account.

Additional Information / References

- https://owasp.org/www-community/attacks/csrf
- https://wiki.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005)
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie
- https://portswigger.net/web-security/csrf
- https://owasp.org/www-community/SameSite

5.3 Subdomain Takeover

CVSSv3 Score	**6.5 (Medium)**
CVSSv3 Vektor String	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N (show in first.org)

Affected Systems

- takeover.example.com

Description

The subdomain `takeover.example.com` had a `CNAME` set to `exampletrafficmanager.trafficmanager.net` at the time of the assessment, which was not allocated, and could therefore be registered via Microsoft Azure. This means that the subdomain `takeover.example.com` could fall under the control of attackers and be used for further attacks, like phishing campaigns.

Recommendations

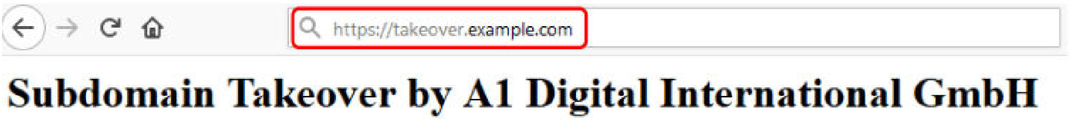
- All unused DNS entries should be removed.
- If this is not possible, the CNAME should be taken over again to prevent it from being taken over by external parties.
- Furthermore, a policy defining a lifecycle management process for domains should be created or adapted.
  - An inventory of all domains and subdomains in use should be kept and maintained.
  - Ensure that all changes made to the infrastructure don't leave domains pointing to IP addresses or domains that are no longer in control of the company.

Technical Description

It was detected that the domain `takeover.example.com` was configured with a `CNAME` record from the domain `exampletrafficmanager.trafficmanager.net`:

```
# nslookup takeover.example.com 8.8.8.8
Server: 8.8.8.8
Address: 8.8.8.8#53
Non-authoritative answer:
takeover.example.com canonical name = exampletrafficmanager.trafficmanager.net.
Name: exampletrafficmanager.trafficmanager.net
Address: 54.192.96.244
```

This subdomain was not registered by anyone, and it was possible to register it using Microsoft Azure and get full control of the `takeover.example.com` subdomain. To achieve this, Microsoft Azure was used to create a Web App and a Traffic Manager Profile with the name `exampletrafficmanager`, to which the Web App was added as an endpoint:



https://takeover.example.com

Subdomain Takeover by A1 Digital International GmbH

Figure 8: Subdomain Takeover of https://takeover.example.com

Attackers could use this vulnerability in order to create phishing campaigns that would use the trust on the domain of the company in order to make the attack more believable.

Additional Information / References

- https://blog.sweepatic.com/subdomain-takeover-principles/
- https://0xpatrik.com/subdomain-takeover-basics/

5.4 Reflected Cross-Site-Scripting (XSS)

CVSSv3 Score	**6.1 (Medium)**
CVSSv3 Vektor String	CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N (show in first.org)

Affected Systems

- www.example.com (203.0.13.64:443 (TCP))

Description

A reflected cross-site scripting (XSS) vulnerability was identified where malicious JavaScript code could be injected into the application. Attackers would be able to steal session information by successfully exploiting this vulnerability and use it to take over other users' accounts.

Recommendations

- It should be ensured that user input is validated and encoded in the source code when being output (Input Validation, Output Encoding).
- For output encoding, special attention should be paid to the following characters:

Character	Encoded Character
&	`&amp;`
<	`&lt;`
>	`&gt;`
"	`&quot;`
'	`&#x27;`
/	`&#x2F;`

Table 5: Output encoding of special characters

- Before using user input in the JavaScript code of the site, it should be escaped.
- More information about XSS vulnerabilities and more detailed information on how to properly fix them can be found on the Additional Information section.

Technical Description

Cross-site scripting (XSS) attacks can be used by attackers to execute malicious JavaScript code in the context of a web application. XSS attacks occur when an attacker introduces malicious JavaScript code in the vulnerable website that runs on the browser of a victim. In principle, a distinction can be made between 3 different types of XSS:

- Reflected Cross-Site Scripting
- Stored Cross-Site Scripting
- DOM-Based Cross-Site Scripting

In **Reflected XSS**, the malicious JavaScript code is usually passed to the web server by the attacker via GET or POST parameters. The web server processes the data and returns the content of the passed parameters unfiltered back to the end user. Thus, an attacker can send the victim a link, for example, that once visited will execute the malicious code on its browser in the context of the user of the victim.

The following illustration shows how a Reflected XSS attack can take place:

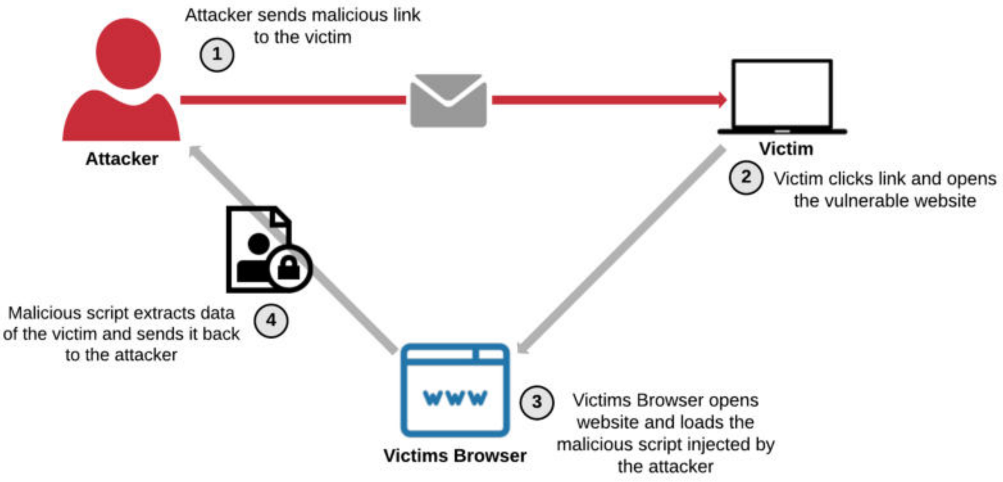

Attacker sends malicious link to the victim
1
Attacker
Victim
2
Victim clicks link and opens the vulnerable website
Malicious script extracts data of the victim and sends it back to the attacker
4
WWW
3
Victims Browser
Victims Browser opens website and loads the malicious script injected by the attacker

Figure 9: Illustration of Reflected XSS

In **Stored XSS**, the attacker stores the malicious JavaScript code in the vulnerable site. In contrast to Reflected XSS, an attacker does not need to insert a link to the victim - in most cases, a call to the vulnerable website is sufficient for the malicious JavaScript code to be executed.

With **DOM-Based XSS**, the malicious JavaScript code is only processed in the Document Object Model (DOM) of the browser - the malicious code usually never reaches the server. The attacker can, for example, pass the JavaScript code via a so-called anchor in the URL. An example would be `http://www.some.site/site.html#default=<script>alert(document.cookie)</script>`.

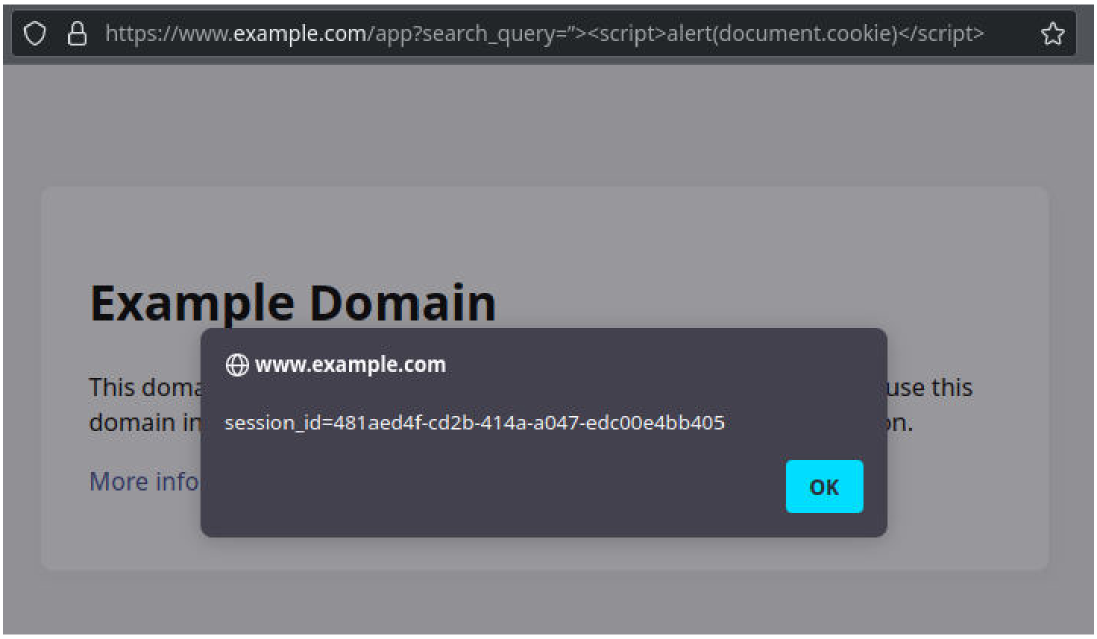
In the course of the assessment, a reflected cross-site scripting vulnerability was identified in the **Example Application** website. The following parameters were vulnerable at the time of the test:

- `search_query`

As this is a reflected cross-site scripting vulnerability, the injected JavaScript code is executed when the following URL is called:

- `https://www.example.com/app?search_query="><script>alert(document.cookie)</script>`

The following screenshot shows the execution of JavaScript code in the context of the affected website.



https://www.example.com/app?search_query=”><script>alert(document.cookie)</script>

www.example.com

session_id=481aed4f-cd2b-414a-a047-edc00e4bb405

Figure 10: Reflected XSS on example.com

Attackers could potentially use this in order to retrieve session information of the victim and take over the account. By gaining full access to an account, attackers have the ability to perform any action and access any data that the victim has access to.

Additional Information / References

- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- https://owasp.org/www-community/attacks/xss/

5.5 Outdated Tomcat Installation

CVSSv3 Score	**5.9 (Medium)**
CVSSv3 Vektor String	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L/E:U/RL:O/RC:U (show in first.org)

Affected Systems

- shop.example.com (203.0.13.65:443 (TCP))

Description

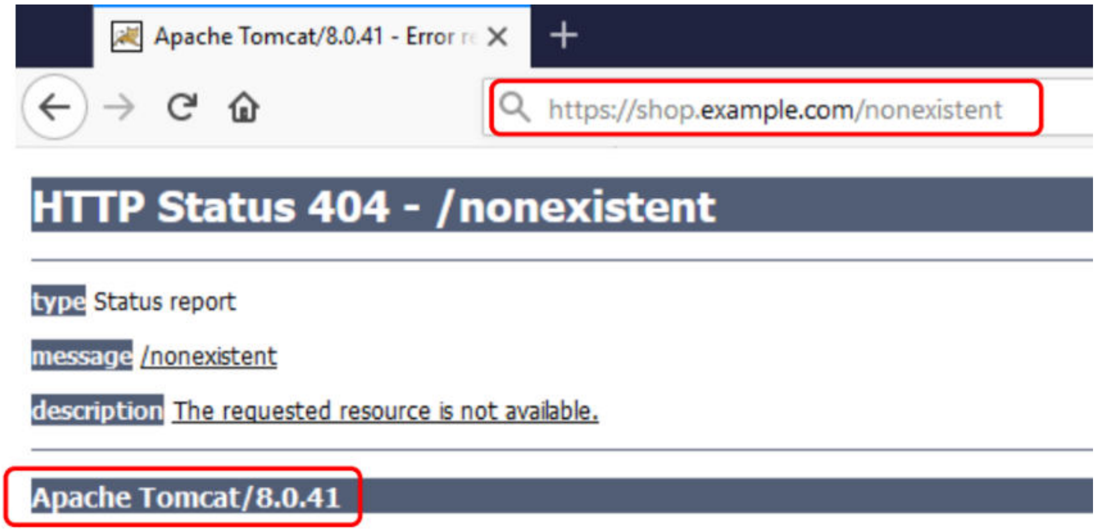
At the time of the assessment, the WebShop system appeared to be using an outdated version of Tomcat that had at least one known vulnerability. This version contains known weaknesses which may allow access to users' data.

Recommendations

- It is recommended to upgrade the Tomcat installation at least to the latest version of Tomcat 8.5.
- A policy detailing a continuous update process of all systems in the company should be established.
  - Available updates for products used in the company should be monitored periodically.
  - The patch status of all machines and services of the company should be centrally tracked.
  - All security-critical updates should be guaranteed to be installed in a timely manner.
- If updates are not possible, affected systems should be isolated:
  - Access should be restricted strictly to only the users that need it.
  - This restriction should happen on a network based level, so that all access to the affected systems is blocked as far as possible.
- Only generic error messages should be supplied to end users.
  - Avoid to give specific information regarding the software or hardware helps to prevent fingerprinting of the services in use.

Technical Description

During the test, it was possible to retrieve the Tomcat version running on `https://shop.example.com` by accessing a non-existent page, which returned an error message that included the Tomcat version used. This can be seen in the following figure:


Apache Tomcat/8.0.41 - Error re
https://shop.example.com/nonexistent

HTTP Status 404 - /nonexistent

type Status report

message /nonexistent

description The requested resource is not available.

Apache Tomcat/8.0.41

Figure 11: Tomcat version of the webshop server

Version 8.0.41 of Tomcat contains several known weaknesses, which could potentially be abused by attackers to carry out further attacks in order to gain access to private data of logged on users. Furthermore, Tomcat 8.0 is no longer supported, and will not receive further security patches. More information regarding the known weaknesses in this Tomcat version can be found in the references.

Additional Information / References

- https://tomcat.apache.org/security-8.html
- https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities
- https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration.html

5.6 Missing Security Headers

CVSSv3 Score	3.7 (Low)
CVSSv3 Vektor String	CVSS:3.0/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N/E:U/RL:O (show in first.org)

Affected Systems

- www.example.com (203.0.13.64:443 (TCP))
- shop.example.com (203.0.13.65:443 (TCP))

Description

In the course of the security assessment, it was discovered that affected web applications had not consistently implemented common security headers. Setting security headers can increase the overall security of a web application and make it more difficult for attackers to carry out attacks such as cross-site scripting (XSS) or man-in-the-middle.

Recommendations

- It should be evaluated whether the recommended security headers can consistently be set for all web applications.
- In particular, the content security policy (CSP) can provide good protection against cross-site scripting attacks. However, the configuration of the CSP can be complex and may cause errors in the web application. Therefore, it is recommended to initially test the CSP with “Report-Only”.

Technical Description

In the following, the security headers that were not (consistently) set at the time of the penetration test are described in detail.

The **Content-Security Policy** header is used to restrict, report and prevent e.g. cross-site scripting and framing attacks via access policies. The source restriction should only allow directly controlled addresses, the “unsafe” options are strongly discouraged. For now, start with “Report-Only” and “self” as origin to see which external requests are needed. After a more detailed specification of the origins it is recommended to enforce the CSP.

The **X-Frame-Options** header specifies whether the page may be included in another page as a “frame”, “iframe” or “embed”. This prevents so-called “clickjacking” attacks, in which users can be tricked into clicking on things that can be hidden behind other elements on foreign websites. It is recommended to at least prevent embedding of pages from external domains.

The **X-Content-Type-Options** header can disable automatic detection and correction of MIME types for JavaScript and CSS files in the browser with the “nosniff” option, thus blocking vulnerabilities where supposed JavaScript can be loaded from other files. It is advised to enable the header with “nosniff”.

The **Referrer-Policy** header specifies which “referers” (sic!) should be sent for which requests. This can be used to prevent exact information about the origin of users from being passed on to external pages or pages without encryption. This information could be problematic if it contains session tokens, names, IDs and other sensitive data. It is advised to send the header with “strict-origin-when-cross-origin” to minimize user tracking.

The **Permissions-Policy** header controls which JavaScript sources are allowed to use which browser features from the page. Among them are the use of the camera, microphone, geolocation and payment requests, which should be disabled by default. It is advised to disable as much as possible, as it makes it more difficult for attackers to collect data about users.

The **HTTP Strict-Transport-Security** header protects against encryption by specifying that a domain and optionally its subdomains may only be accessed in encrypted form for a certain period of time. For this purpose, the Strict-Transport-Security header must be sent with a time for which this restriction is to apply. It is recommended to set this header with a time of one year.

The following table provides a brief overview of which headers are set on the affected hosts.

Host	Content-Security Policy (CSP)	X-Frame-Options	X-Content-Type-Options	Referrer-Policy	Permissions-Policy	HTTP-Strict-Transport-Security HSTS)
https://www.example.com	🚫	⚠️	⚠️	🚫	🚫	✅
https://shop.example.com	🚫	✅	✅	🚫	🚫	✅

Table 6: Set Security Headers

Legend The graphical categorization of the upper table was done in two different levels:

- ✅ The header is correctly set and works as intended.
- 🚫 The header is not set.
- ⚠️ The header is misconfigured.

Additional Information / References

- https://securityheaders.com/
- https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet
- https://www.owasp.org/index.php/Security_Headers
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Feature-Policy
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security

5.7 Weak SSL/TLS Configuration

CVSSv3 Score	3.7 (Low)
CVSSv3 Vektor String	CVSS:3.0/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N/E:U/RL:O (show in first.org)

Affected Systems

- www.example.com (203.0.13.64:443 (TCP))
- shop.example.com (203.0.13.65:443 (TCP))

Description

During the assessment it was determined, that the affected systems used insecure SSL/TLS configurations. An attacker with access to the network traffic could potentially decrypt the transmitted packets, and thus get access to sensitive user usernames and passwords.

Recommendations

- Insecure cipher suites should be deactivated, as there are known vulnerabilities and attacks for these protocols.
- If possible, only the currently secure versions TLS 1.2 and TLS 1.3 should be used.
- Instructions for secure TLS configuration can be obtained from the **Additional Resources** section.

Technical Description

The Hypertext Transfer Protocol (short HTTP) is a protocol to transfer data between two systems. HTTP is a plain-text protocol, which means, it does not support encryption on its own. To protect data in transfer over the internet there are standards which work as an extension to HTTP and encrypt the transferred data, like Hypertext Transfer Protocol Secure (HTTPS). HTTPS uses SSL/TLS to implement the encryption during the transfer. An attacker could still access sensitive data by exploiting known vulnerabilities in the SSL/TLS protocol and thus breaking the encryption.

During the assessment some systems were identified using insecure SSL/TLS configuration.

The findings of the assessment were as follows:

Using SSL: At the time of testing, the affected system supported versions of the SSL protocol, which use obsolete encryption protocols that should no longer be used in production environments. Attackers with access to network traffic could attempt to break the weak encryption and thus gain access to sensitive data.

Use of weak encryption methods: At the time of testing, the affected system supported TLSv1.0 with the RC4 cipher. RC4 is considered insecure and should be disabled. TLSv1.0 also contains ciphers considered insecure and should be disabled if possible.

No use of TLSv1.2 or TLSv1.3: The affected system did not support TLSv1.2 or TLSv1.3 at the time of testing; these new modern protocols should be supported whenever possible to ensure a secure connection for users.

No support of so-called AEAD cipher suites: AEAD (Authenticated Encryption with Associated Data) ciphers are cipher suites that are considered secure. For example, TLSv1.3 now only relies on AEAD cipher suites.

A detailed overview of the affected systems and their TLS and SSL versions used at the time of testing can be found in the following table.

HOST	TLSv1.3	TLSv1.2	TLSv1.1	TLSv1.0	SSLv3	SSLv2	SSLv1
https://www.example.com	⚠️	⚠️	🚫	🚫	🚫	✅	✅
https://shop.example.com	⚠️	✅	🚫	🚫	✅	✅	✅

Table 7: SSL/TLS protocol versions used

Legend The graphical categorization of the upper table was done in three different levels:

- ✅ is correctly configured (active, when safe; inactive, when unsafe)
- ⚠️ is considered safe and is not used
- 🚫 is considered unsafe and is used

Additional Information / References

- https://english.ncsc.nl/publications/publications/2021/january/19/it-security-guidelines-for-transport-layer-security-2.1
- https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html
- https://www.ssllabs.com
- https://ssl-config.mozilla.org/

6 Appendix

6.1 Contact persons

A1 Digital International GmbH

Name	Role	Telephone	E-Mail
Alice Codex	Execution of Security Assessment	+431234567890	ask.security@a1.digital
Trent Trustworthy	Review	+431234567890	ask.security@a1.digital
Bob Binary	Review	+431234567890	ask.security@a1.digital

Table 8: Contact persons at A1 Digital International GmbH

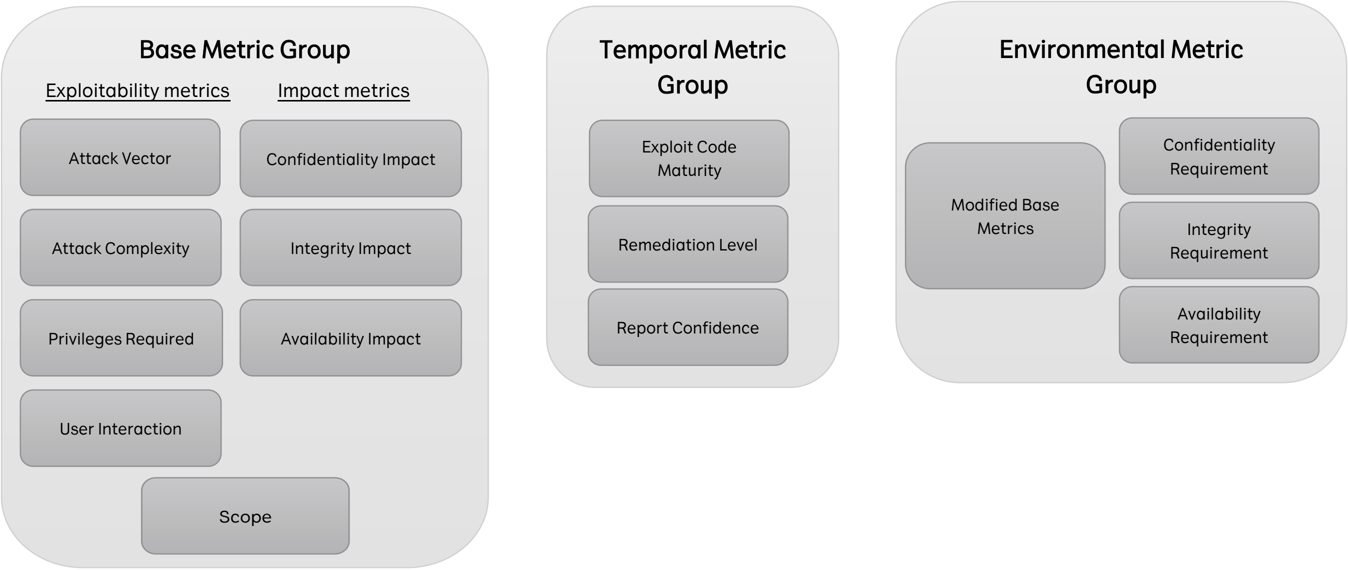
Example GmbH

Name	Role	Telephone	E-Mail
Jane Doe	Lead of Example Department	+4312345678901	jd@example.com
Maximilian Muster	Support	+4312345678902	mm@example.com

Table 9: Contact persons at Example GmbH

6.2 CVSS v3.0 metrics

CVSS comprises three metric groups: **Base, Temporal** and **Environmental** as shown in the figure below:



6.2.1 Base Metric Group

The **Base Metric Group** expresses the fundamental risk of a weakness and assesses the vulnerable component. No valid CVSS value can be formed without a Base Metric. In turn the Base Metric is divided into Exploitability Metrics and Impact Metrics.
The **Exploitability Metric** reflects the ease and required pre-requisites for successful utilisation of the weakness.
The **Impact Metric** on the other hand reflects the direct consequence of the successful utilisation of the weak point - is the confidentiality, integrity or availability of the affected data/ of the affected system endangered?

Metric	**Possible values**
Attack Vector (V) - attack vector	Network (N), Adjacent (A), Local (L), Physical (P)
Attack Complexity (AC) - attack complexity	Low (L), High (H)
Privileges Required (PR) - privileges required	None (N), Low (L), High (H)
User Interaction (UI) - required user interaction	None (N), Required (R)
Scope (S) - affected area	Changed (C), Unchanged (U)
Confidentiality Impact (C) - loss of confidentiality	None (N), Low (L), High (H)
Integrity Impact (I) - loss of integrity	None (N), Low (L), High (H)
Availability Impact (A) - loss of availability	None (N), Low (L), High (H)

Table 10: Overview of Base Metric Group

6.2.2 Temporal Metric Group

The **Temporal Metric Group** expresses the characteristics of a weak point which may change over time. For example after some time an official patch may be published, which would reduce the Temporal Score.

Metric	Possible values
Exploit Code Maturity (E) - degree of maturity of the exploit code present	Not Defined (X), High (H), Functional (F), Proof of Concept (P), Unproven (U)
Remediation Level (RL) - countermeasures present	Not Defined (X), Unavailable (U), Workaround (W), Temporal Fix (T), Official Fix (O)
Report Confidence (RC) - measures the reliability of the available information regarding the weakness	Not Defined (X), Confirmed (C), Reasonable (R), Unknown (U)

Table 11: Overview of Temporal Metric Group

6.2.3 Environmental Metric Group

The **Environmental Metric Group** is specially set for the user environment. This metric allows the adaptation of the scores with respect to the importance of an affected system for the user/customer. The adjustment is done based on the requirements for confidentiality, integrity and availability.

Metric	Possible values
Confidentiality Requirement (CR) - requirement for confidentiality	Network (N), Adjacent (A), Local (L), Physical (P)
Integrity Requirement (IR) - requirement for integrity	Low (L), High (H)
Availability Requirement (AR) - requirement for availability	None (N), Low (L), High (H)

Table 12: Overview of Environmental Metric Group

6.2.4 Modified Base Metric Group

In addition, the base metrics can be shown as a modified value (modified base metric).
This can be used to describe situations which increase the base score. For example a component could require multiple factors for authentication as standard (PR: High) in order to reach specific resources, whereas in the test environment no authentication was required (PR: None).

Metric	Possible values
Modified Attack Vector (MAV)	The same values as the associated base metrics + not defined (N).
Modified Attack Complexity (MAC)	
Modified Privileges Required (MPR)	
Modified User Interaction (MUI)	
Modified Scope (MS)	
Modified Confidentiality (MC)	
Modified Integrity (MI)	
Modified Availability (MA)	

Table 13: Overview of Modified Base Metric Group

Detailed information regarding the base, temporal and environmental metrics and their values are available on the *first.org* website[2]

6.3 Text representation of CVSS v3.0 scores

In most cases it is helpful to have a text representation of the numerical CVSS scores. Each individual metric (Base, Temporal and Environmental) can be brought into text form using the following table.[3][4]

Severity	CVSS v3 Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Table 14: Text representation of CVSS v3.0 scores

[2] https://www.first.org/cvss/specification-document
[3] https://nvd.nist.gov/vuln-metrics/cvss
[4] https://www.first.org/cvss/specification-document#Qualitative-Severity-Rating-Scale

6.4 List of Tables

Table 1 Change record . . . 2
Table 2 Overview of weaknesses . . . 6
Table 3 Weakness categorisation . . . 7
Table 4 Systems tested . . . 9
Table 5 Output encoding of special characters . . . 19
Table 6 Set Security Headers . . . 25
Table 7 SSL/TLS protocol versions used . . . 27
Table 8 Contact persons at A1 Digital International GmbH . . . 28
Table 9 Contact persons at Example GmbH . . . 28
Table 10 Overview of Base Metric Group . . . 29
Table 11 Overview of Temporal Metric Group . . . 30
Table 12 Overview of Environmental Metric Group . . . 30
Table 13 Overview of Modified Base Metric Group . . . 31
Table 14 Text representation of CVSS v3.0 scores . . . 31

6.5 List of Figures

Figure 1 Overview of the identified weaknesses . . . 6
Figure 2 Weakness categorisation . . . 7
Figure 3 Implementation concept for penetration tests . . . 10
Figure 4 All user data is queried by exploiting an SQL injection vulnerability . . . 12
Figure 5 Request to the application which causes an SQL error . . . 12
Figure 6 Response showing the SQL error store.example.com . . . 13
Figure 7 Illustration of Cross-Site Request Forgery . . . 16
Figure 8 Subdomain Takeover of https://takeover.example.com . . . 18
Figure 9 Illustration of Reflected XSS . . . 20
Figure 10 Reflected XSS on example.com . . . 21
Figure 11 Tomcat version of the webshop server . . . 22

6.6 OWASP Testing Guide Version 4.2

Information Gathering
Conduct Search Engine Discovery and Reconnaissance for Information Leakage (OTG-INFO-001)
Fingerprint Web Server (OTG-INFO-002)
Review Webserver Metafiles for Information Leakage (OTG-INFO-003)
Enumerate Applications on Webserver (OTG-INFO-004)
Review Webpage Comments and Metadata for Information Leakage (OTG-INFO-005)
Identify application entry points (OTG-INFO-006)
Map execution paths through application (OTG-INFO-007)
Fingerprint Web Application Framework (OTG-INFO-008)
Fingerprint Web Application (OTG-INFO-009)
Map Application Architecture (OTG-INFO-010)

Configuration and Deployment Management Testing
Test Network/Infrastructure Configuration (OTG-CONFIG-001)
Test Application Platform Configuration (OTG-CONFIG-002)
Test File Extensions Handling for Sensitive Information (OTG-CONFIG-003)
Review Old, Backup and Unreferenced Files for Sensitive Information (OTG-CONFIG-004)
Enumerate Infrastructure and Application Admin Interfaces (OTG-CONFIG-005)
Test HTTP Methods (OTG-CONFIG-006)
Test HTTP Strict Transport Security (OTG-CONFIG-007)
Test RIA cross domain policy (OTG-CONFIG-008)
Test File Permission (OTG-CONFIG-009)

Identity Management Testing
Test Role Definitions (OTG-IDENT-001)
Test User Registration Process (OTG-IDENT-002)
Test Account Provisioning Process (OTG-IDENT-003)
Testing for Account Enumeration and Guessable User Account (OTG-IDENT-004)
Testing for Weak or unenforced username policy (OTG-IDENT-005)

Authentication Testing
Testing for Credentials Transported over an Encrypted Channel (OTG-AUTHN-001)
Testing for default credentials (OTG-AUTHN-002)
Testing for Weak lock out mechanism (OTG-AUTHN-003)
Testing for bypassing authentication schema (OTG-AUTHN-004)
Test remember password functionality (OTG-AUTHN-005)
Testing for Browser cache weakness (OTG-AUTHN-006)
Testing for Weak password policy (OTG-AUTHN-007)
Testing for Weak security question/answer (OTG-AUTHN-008)
Testing for weak password change or reset functionalities (OTG-AUTHN-009)
Testing for Weaker authentication in alternative channel (OTG-AUTHN-010)

Authorization Testing
Testing Directory traversal/file include (OTG-AUTHZ-001)
Testing for bypassing authorization schema (OTG-AUTHZ-002)
Testing for Privilege Escalation (OTG-AUTHZ-003)
Testing for Insecure Direct Object References (OTG-AUTHZ-004)

Session Management Testing
Testing for Bypassing Session Management Schema (OTG-SESS-001)
Testing for Cookies attributes (OTG-SESS-002)
Testing for Session Fixation (OTG-SESS-003)
Testing for Exposed Session Variables (OTG-SESS-004)
Testing for Cross Site Request Forgery (CSRF) (OTG-SESS-005)
Testing for logout functionality (OTG-SESS-006)
Test Session Timeout (OTG-SESS-007)
Testing for Session puzzling (OTG-SESS-008)

Input Validation Testing
Testing for Reflected Cross Site Scripting (OTG-INPVAL-001)
Testing for Stored Cross Site Scripting (OTG-INPVAL-002)
Testing for HTTP Verb Tampering (OTG-INPVAL-003)
Testing for HTTP Parameter pollution (OTG-INPVAL-004)
Testing for SQL Injection (OTG-INPVAL-005)
Oracle Testing
MySQL Testing
SQL Server Testing
Testing PostgreSQL (from OWASP BSP)
MS Access Testing
Testing for NoSQL injection
Testing for LDAP Injection (OTG-INPVAL-006)
Testing for ORM Injection (OTG-INPVAL-007)
Testing for XML Injection (OTG-INPVAL-008)
Testing for SSI Injection (OTG-INPVAL-009)
Testing for XPath Injection (OTG-INPVAL-010)
IMAP/SMTP Injection (OTG-INPVAL-011)
Testing for Code Injection (OTG-INPVAL-012)
Testing for Local File Inclusion
Testing for Remote File Inclusion
Testing for Command Injection (OTG-INPVAL-013)
Testing for Buffer overflow (OTG-INPVAL-014)
Testing for Heap overflow
Testing for Stack overflow
Testing for Format string
Testing for incubated vulnerabilities (OTG-INPVAL-015)
Testing for HTTP Splitting/Smuggling (OTG-INPVAL-016)

Testing for Error Handling
Analysis of Error Codes (OTG-ERR-001)
Analysis of Stack Traces (OTG-ERR-002)

Testing for weak Cryptography
Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection (OTG-CRYPST-001)
Testing for Padding Oracle (OTG-CRYPST-002)
Testing for Sensitive information sent via unencrypted channels (OTG-CRYPST-003)

Business Logic Testing
Test Business Logic Data Validation (OTG-BUSLOGIC-001)
Test Ability to Forge Requests (OTG-BUSLOGIC-002)
Test Integrity Checks (OTG-BUSLOGIC-003)
Test for Process Timing (OTG-BUSLOGIC-004)
Test Number of Times a Function Can be Used Limits (OTG-BUSLOGIC-005)
Testing for the Circumvention of Work Flows (OTG-BUSLOGIC-006)
Test Defenses Against Application Mis-use (OTG-BUSLOGIC-007)
Test Upload of Unexpected File Types (OTG-BUSLOGIC-008)
Test Upload of Malicious Files (OTG-BUSLOGIC-009)

Client Side Testing
Testing for DOM based Cross Site Scripting (OTG-CLIENT-001)
Testing for JavaScript Execution (OTG-CLIENT-002)
Testing for HTML Injection (OTG-CLIENT-003)
Testing for Client Side URL Redirect (OTG-CLIENT-004)
Testing for CSS Injection (OTG-CLIENT-005)
Testing for Client Side Resource Manipulation (OTG-CLIENT-006)
Test Cross Origin Resource Sharing (OTG-CLIENT-007)
Testing for Cross Site Flashing (OTG-CLIENT-008)
Testing for Clickjacking (OTG-CLIENT-009)

Testing WebSockets (OTG-CLIENT-010)
Test Web Messaging (OTG-CLIENT-011)
Test Local Storage (OTG-CLIENT-012)

7 Imprint

A1 Digital International GmbH

Business area: Machine-to-machine communication services, IT solutions, devices and other associated products and services
UID number: ATU 66624566

Representative persons:
Dr. Elisabetta Castiglioni (CEO)
Martin Schiffmann (CFO)

FB number: 366000k
Company legal jurisdiction: HG Vienna
Company headquarters: Vienna
Address: Lassallestraße 9, A-1020 Vienna
Contact details: Telephone: (+43) 5 06640; E-Mail: info@a1.digital
Chamber membership: Wirtschaftskammer Wien
Applicable legal regulations: Telecommunication laws: www.ris.bka.gv.at
Regulatory authority/commercial authorities: Österreichische Regulierungsbehörde für Rundfunk und Telekommunikation (RTR GmbH)