

PENETRATION TEST REPORT

Example Application

Example Customer

Attn. Firstname Lastname

Hermannngasse 29

1070, Vienna

Vienna, November 18, 2024

Report Version: 1.0

Winston Wagner

Haitingergasse 12
3400 Klosterneuburg

Contact:

Phone: 0123456789

Mail: hello@winstonwww.com

Table of Contents

Team	3
List of Changes	3
Disclaimer	3
Executive Summary	4
Overview	4
Identified Vulnerabilities	5
Methodology and Scope	6
User Accounts and Permissions	6
Findings	7
C1: Blind SQL Injection in Password Reset Endpoint	7
C2: HTTP Host Header SSRF	9
H1: Session Fixation	11
M1: Missing or Incorrectly Configured HTTP Security Headers	13
M2: User Enumeration	15
M3: Web Cache Deception	17
L1: Verbose Error Messages	19

Team

Name	Contact	Role
Winston Wagner	hello@winstonwww.com	Lead Pentester

List of Changes

Version	Description	Date	Author
0.1	Initial Creation	Nov 12, 2024	Winston Wagner
0.9	Review	Nov 15, 2024	Winston Wagner
1.0	Published	Nov 18, 2024	Winston Wagner

Disclaimer

This test is has been planned using a time-box approach, whereby the goal was to identify as many vulnerabilities as possible in a reasonable timeframe. However, given the nature of cybersecurity, there are no guarantees for completeness for the findings listed in the report above. Futhermore, this test provides only a snapshot in time, future risks can not be derived from it. Finding a wide range of vulnerabilities has been prioritized over finding all occurances of one vulnerability in a category. It is therefore advised to check the source code for similar patterns after identifying the cause of a finding mentioned in this report.

Executive Summary

Overview

The penetration test conducted on the web application identified significant security risks, including two critical vulnerabilities, which demand immediate attention to protect sensitive data and system integrity.

A SQL injection vulnerability was identified in the password reset functionality, where attackers can manipulate the system by sending specially crafted requests. This vulnerability can allow an attacker to extract sensitive information such as usernames, passwords, and other data from the database, modify the data stored in the database, and perform reconnaissance to discover the structure of the database.

The application was vulnerable to a server-side request forgery (SSRF) attack due to improper handling of the Host header. Attackers can manipulate this header to send requests to internal resources, exposing sensitive data or causing further exploitation of internal services. This vulnerability can allow attackers to interact with services and systems that should be inaccessible from the public internet. A reachable admin service was found, allowing attackers to manipulate any user's data.

In course of the assessment, a security flaw was discovered in the application's session management system, allowing an attacker to force a victim to use a predetermined session ID. This could let attackers hijack the victim's session and access their account without needing their login credentials.

During the penetration test it was found that the web application either lacked essential HTTP security headers or had them configured insecurely. Without properly configured HTTP security headers, the web application's attack surface is expanded, increasing the likelihood of an attacker exploiting client-side vulnerabilities.

The web application was vulnerable to user enumeration. User enumeration is a common vulnerability in web applications that occurs when an attacker can use brute force techniques to determine valid user accounts in a system. Although user enumeration is a low risk in itself, it still provides an attacker with valuable information for follow-up attacks such as in brute force and credential stuffing attacks or in social engineering campaigns.

At the time of the security test, the application was vulnerable to web cache deception, where an attacker could trick the caching mechanism into storing sensitive user-specific pages such as the account dashboard as publicly accessible cached content. This could allow unauthorized users to access private information.

The application exposed detailed error messages when unexpected inputs are provided. These messages can reveal sensitive information about the system's internal workings, such as server configuration, database structure, or code logic. While this issue did not allow direct exploitation, it potentially provides attackers with useful information for future attacks.

Identified Vulnerabilities

#	CVSS	Description	Page
C1	9.8	Blind SQL Injection in Password Reset Endpoint	7
C2	9.8	HTTP Host Header SSRF	9
H1	8.1	Session Fixation	11
M1	5.4	Missing or Incorrectly Configured HTTP Security Headers	13
M2	5.3	User Enumeration	15
M3	5.3	Web Cache Deception	17
L1	3.7	Verbose Error Messages	19

Vulnerability Overview

In the course of this penetration test **2 Critical**, **1 High**, **3 Medium** and **1 Low** vulnerabilities were identified:

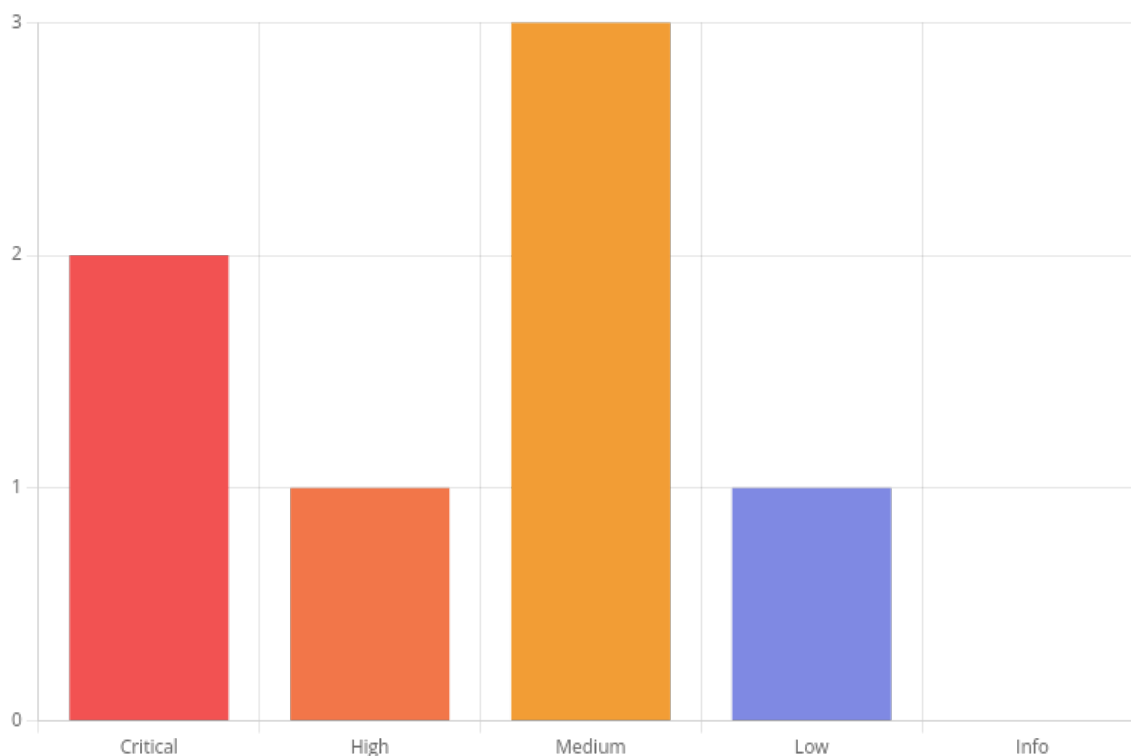


Figure 1 - Distribution of identified vulnerabilities

Methodology and Scope

The objective of this penetration test was to assess the security posture of the target web application, identify vulnerabilities, and provide actionable recommendations to mitigate risks.

The testing followed a gray-box approach, where testers had partial knowledge of the application internals.

All manual and automated tests have been performed to industry standard practices, including, but not limited to the OWASP Web Security Testing Guide version 4.

The following system was declared to be in the scope of the penetration test by Example Customer

System	Description
https://github.com/testsystem	Source code with Docker setup
https://testsystem.com	Test server

The test server has been set up by the customer as a testing instance solely for this assessment.

Exclusions (Out of Scope)

The following were explicitly excluded from this test:

- Systems, subdomains, or applications not explicitly mentioned as in-scope.
- Denial-of-Service (DoS) attacks or stress testing of the application or infrastructure.
- Social engineering attacks targeting employees or users of the application.

User Accounts and Permissions

Provided Users

- User1
- User2
- User3

Findings

C1: Blind SQL Injection in Password Reset Endpoint	
Score	9.8 (Critical)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Target	POST /password-reset
References	https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Overview

A SQL injection vulnerability was identified in the password reset functionality, where attackers can manipulate the system by sending specially crafted requests. This allows them to extract sensitive data by causing intentional delays in the server's responses, even though no visible errors or data are returned.

This vulnerability can allow an attacker to:

- Extract sensitive information such as usernames, passwords, and other data from the database.
- Perform reconnaissance to discover the structure of the database (e.g., tables, columns).
- Modify the data stored in the database.

This can lead to data breaches, financial losses, damaged reputation, and disruptions to business operations, making it a critical risk for organizations.

Details

A Timing-Based Blind SQL Injection vulnerability was discovered in the email parameter of the password-reset endpoint.

SQL Injection (SQLi) is a vulnerability that occurs when an application fails to properly validate or sanitize user-supplied input before including it in SQL queries. Attackers can inject malicious SQL code into input fields, URLs, or headers to manipulate the database. This can allow unauthorized access to sensitive data, modification or deletion of records, authentication bypass, or even execution of administrative commands on the database.

By injecting SQL payloads that deliberately induce delays in the server's response, it was possible to confirm the execution of SQL queries without any direct response output. This type of SQL injection leverages time delays to infer information about the database.

This vulnerability can be executed unauthorized, meaning no user account is necessary.

Proof of concept:

Two payloads were submitted to the email parameter to demonstrate the time-based blind SQL Injection vulnerability:

No Delay (False Condition):

```
email=test@example.com' AND 1=2--
```

The server responded immediately, indicating no delay.

Induced Delay (True Condition):

```
email=test@example.com' AND IF(1=1, SLEEP(5), 0)--
```

The server delayed for 5 seconds before responding, confirming the successful execution of the injected query.

Demonstration of Data Extraction:

By adjusting the SQL payload, an attacker could extract data one character at a time:

Payload to Extract the First Character of the Database Version:

```
email=test@example.com' AND IF(ASCII(SUBSTRING(@@version,1,1))=52, SLEEP(5), 0)--
```

If the server delays by 5 seconds, the first character of the database version is 4 (ASCII value 52).

Recommendation

- Use prepared statements throughout the application to effectively avoid SQL injection vulnerabilities. Prepared statements are parameterized statements and ensure that even if input values are manipulated, an attacker is unable to change the original intent of an SQL statement.
- Use existing stored procedures by default where possible. Typically, stored procedures are implemented as secure parameterized queries and thus protect against SQL injections.
- Always validate all user input. Ensure that only input that is expected and valid for the application is accepted. You should not sanitize potentially malicious input.
- To reduce the potential damage of a successful SQL Injection attack, you should minimize the assigned privileges of the database user used according to the principle of least privilege.

C2: HTTP Host Header SSRF

Score	9.8 (Critical)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Target	<ul style="list-style-type: none">• HTTP Request Handling• Internal Service Access
References	<ul style="list-style-type: none">• https://www.invicti.com/learn/host-header-attacks/• https://portswigger.net/web-security/host-header

Overview

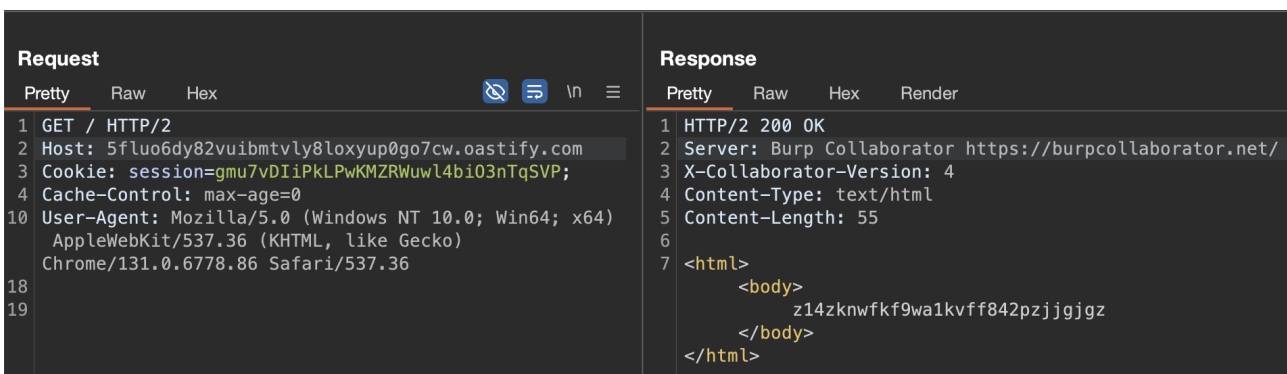
The application was vulnerable to a server-side request forgery (SSRF) attack due to improper handling of the Host header. Attackers can manipulate this header to send requests to internal resources, exposing sensitive data or causing further exploitation of internal services. This vulnerability can allow attackers to interact with services and systems that should be inaccessible from the public internet. A reachable admin service was found, allowing attackers to manipulate any user's data.

This can lead to data breaches, unauthorized access to sensitive user data, and potential system compromise.

Details

If the server did not validate the `Host` header properly, and forwarded requests to any address inside that header.

To confirm this the `Host` header of an intercepted request was edited to contain an attacker controlled domain. Sending the request resulted in a response containing content from the attacker controlled domain.



```
Request
Pretty Raw Hex
1 GET / HTTP/2
2 Host: 5fluo6dy82vuibmtvly8loxyup0go7cw.oastify.com
3 Cookie: session=gmu7vDIiPkLPwKMZRWuwL4bi03nTqSVP;
4 Cache-Control: max-age=0
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
    AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/131.0.6778.86 Safari/537.36
18
19

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Server: Burp Collaborator https://burpcollaborator.net/
3 X-Collaborator-Version: 4
4 Content-Type: text/html
5 Content-Length: 55
6
7 <html>
  <body>
    z14zknwfkf9wa1kvff842pzjjggz
  </body>
</html>
```

An attacker could try accessing various internal services by modifying the Host header to target internal IPs or services.

The header was set to a range of internal IP addresses from 192.168.0.0 to 192.168.0.255. The IP 192.168.0.170 resulted in a status code 302 as shown below. This confirmed the presence of a reachable internal service.

Intruder attack results filter: Showing all items			
Request	Payload	Status code ^	Response rec...
171	170	302	76
0		504	63
1	0	504	90
2	1	504	66
3	2	504	91
4	3	504	91
5	4	504	54
6	5	504	58

Request	Response
Pretty	Raw Hex
1 GET / HTTP/2	
2 Host: 192.168.0.170	

The response displayed the path /admin in the Location header revealing an internal admin page.

```
HTTP/2 302 Found
Location: /admin
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

By exploiting the Host header SSRF vulnerability, it was possible to access the admin page and admin endpoints such as /admin/users/delete/<id> without authentication.

Recommendation

- Ensure that the application validates and sanitizes the Host header before using it in any internal requests. Only allow trusted and predefined host values to be processed. Reject any requests with suspicious or unexpected host values, such as 127.0.0.1, localhost, or internal IP ranges that are not meant to be exposed.
- Restrict access to internal services (e.g., database servers, metadata services) by using firewall rules, network segmentation, or authentication mechanisms. Internal services should be isolated from external HTTP requests or should require strong authentication.
- To prevent routing-based attacks on internal infrastructure, you should configure your load balancer or any reverse proxies to forward requests only to a whitelist of permitted domains.

H1: Session Fixation

Score	8.1 (High)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N
Target	POST /login
References	https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Overview

A security flaw was discovered in the application's session management system, allowing an attacker to force a victim to use a predetermined session ID. This could let attackers hijack the victim's session and access their account without needing their login credentials. Using this technique, an attacker might gain control over administrative accounts.

This vulnerability can result in:

- Unauthorized access to user accounts and theft of sensitive data.
- Loss of user trust and potential regulatory violations.

Details

The application was found to be vulnerable to a Session Fixation attack. Upon initiating a session (e.g., visiting the login page), the session ID (`session_id`) remains unchanged after successful authentication. This allows an attacker to set a session ID for the victim and later hijack the session once the victim logs in.

The application does not generate a new session ID after authentication, making it possible for an attacker to predetermine the session ID and gain unauthorized access to the victim's account.

Proof of Concept:

1. Attacker Sets a Session ID:

The attacker initiates a session with the application and notes the session ID (e.g., `session_id=abc123`).

```
GET /login HTTP/1.1
Host: example.com
Cookie: session_id=abc123
```

2. Victim Uses the Predetermined Session ID:

The attacker tricks the victim into using the same session ID (e.g., by embedding a link in an email):

```
<a href="https://example.com/login?session_id=abc123">Click here to log in</a>
```

3. Victim Logs In:

The victim logs in with valid credentials, and the application does not change the session ID.

4. Attacker Hijacks the Session:

The attacker then uses the same session ID (abc123) to access the victim's authenticated session:

```
GET /dashboard HTTP/1.1  
Host: example.com  
Cookie: session_id=abc123
```

The attacker gains access to the victim's account.

Recommendation

- Ensure that the application generates a new, unique session ID upon successful login.
- Set session cookies with the HttpOnly and Secure flags to prevent client-side access and ensure cookies are transmitted over HTTPS.
- Use the SameSite attribute to restrict cookies from being sent in cross-site requests.
- Implement short session expiration times and enforce automatic logout after periods of inactivity.
- Educate users to avoid clicking on suspicious links that might contain predetermined session IDs.

M1: Missing or Incorrectly Configured HTTP Security Headers

Score	5.4 (Medium)
Vector string	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:L/I:L/A:N
Target	*
References	https://infosec.mozilla.org/guidelines/web_security#content-security-policy







Overview

During the penetration test it was found that the web application either lacked essential HTTP security headers or had them configured insecurely. HTTP security headers play a crucial role in enhancing a web application's security. They help mitigate risks associated with vulnerabilities like cross-site scripting, clickjacking, information disclosure, and others, either making them harder to exploit or preventing them entirely.




Without properly configured HTTP security headers, the web application's attack surface is expanded, increasing the likelihood of an attacker exploiting client-side vulnerabilities.

Details

The following table provides an overview over which headers were set correctly and which were not:

Content-Security Policy (CSP)	Referrer-Policy	HTTP-Strict-Transport-Security (HSTS)	X-Content-Type-Options	X-Frame-Options	Permissions-Policy
					

Legend:

-  Header was not set
-  Header was set but requires further configuration
-  Header was set correctly

Modern browsers offer robust support for various HTTP security headers that help enhance the security of web applications by mitigating client-side vulnerabilities like clickjacking, cross-site scripting (XSS), and other common attacks. These headers are part of the HTTP response, guiding browsers on which security measures to enable or disable during communication with a server. By defining these security-related details, HTTP headers strengthen the overall resilience of web applications against potential threats.

- **Content Security Policy:** The Content Security Policy header enables granular control over the sources from which a browser can load resources. This header is particularly effective at preventing cross-site scripting (XSS) attacks by restricting unauthorized content execution.

- **Referrer Policy.** The `Referrer-Policy` header governs how and when browsers share the Referer header with target pages. This header provides information about the source of an HTTP request, such as when users navigate via a link or load external resources, helping to protect privacy and control information flow.
- **HTTP Strict Transport Security (HSTS).** The HSTS header instructs browsers to interact with a web application exclusively over HTTPS, ensuring secure connections. It also prevents users from bypassing certificate-related errors by strictly enforcing transport layer security.
- **X-Content-Type-Options.** The `X-Content-Type-Options` header ensures that browsers only execute scripts and stylesheets with the correct MIME types specified by the server. Without this header, MIME sniffing may occur, where files are misinterpreted as scripts or stylesheets, potentially leading to XSS vulnerabilities.
- **X-Frame-Options** The `X-Frame-Options` header determines if and how a web page can be embedded within an iframe. This helps defend against clickjacking attacks, where malicious overlays trick users into performing unintended actions on legitimate pages. Although this header is marked as deprecated, it can still be beneficial for older browsers.
- **Permissions policy** The `Permissions-Policy` header allows developers to control the availability and behavior of specific browser features and APIs. Similar to CSP, it focuses on managing browser functionalities rather than overarching security behavior.

These HTTP security headers play an essential role in reducing attack surfaces and safeguarding user interactions on the web.

Recommendation

- The application contains a Content Security Policy (CSP) which is missing clickjacking protection. It is recommended to set the `frame-ancestors` directive to `none` if you do not want your site to be framed, or `self` if you want to allow it to frame itself.
- Restrict the `referrer policy` to prevent potentially sensitive information from being exposed to third party sites. You should define the header as follows: `Referrer-Policy: strict-origin-when-cross-origin`.
- Do not allow the web page to be included in a frame. Set `X-Frame-Options: DENY` for this. Alternatively you can restrict this setting to the same-origin with `X-Frame-Options: SAMEORIGIN`.
- Restrict the use of sensitive browser features such as the camera, microphone or speaker using `Permissions-Policy` headers. Set it and disable all the features that your site does not need or allow them only to the authorized domains: `Permissions-Policy: geolocation=(), camera=(), microphone=()`
This example is disabling geolocation, camera, and microphone for all domains.

M2: User Enumeration

Score	5.3 (Medium)
Vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target	POST /login
References	<ul style="list-style-type: none">• https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/03-Identity_Management_Testing/04-Testing_for_Account_Enumeration_and_Guessable_User_Account• https://blog.rapid7.com/2017/06/15/about-user-enumeration/

Overview

While assessing the web application, a user enumeration vulnerability was found. User enumeration is a common vulnerability in web applications that occurs when an attacker can use brute force techniques to determine valid user accounts in a system.

Although user enumeration is a low risk in itself, it still provides an attacker with valuable information for follow-up attacks such as in brute force and credential stuffing attacks or in social engineering campaigns.

Details

The login mechanism was found to be vulnerable to user enumeration due to differing timing in responses based on the validity of the username. This behavior allows an attacker to enumerate valid usernames by observing the application's responses. The issue arises because the application processes valid and invalid usernames differently, such as by checking the password for valid accounts or terminating the request earlier for nonexistent usernames.

Proof of Concept:

100 requests with valid and 100 requests with invalid usernames were sent to the login endpoint. Afterwards the timings of all requests were compared.

Login request with existing user:

```
POST /login HTTP/1.1
Host: example.com
Content-Type: application/json

{ "username": "existingUser", "password": "wrongPassword" }
```

Login request with nonexisting user:

```
POST /login HTTP/1.1
Host: example.com
Content-Type: application/json

{ "username": "nonexistingUser", "password": "anyPassword" }
```

Timing for login requests:

	Timing	Duration(s)
	valid max	1340
	valid min	754
	valid average	943.20
	invalid max	410
	invalid min	296
	invalid average	326.91

The response time for invalid usernames was consistently shorter (~327ms) compared to valid usernames (~943ms). This timing difference can be exploited using automated tools to identify valid usernames.

Recommendation

- Ensure that the web application always returns generic error messages when invalid usernames, passwords, or other credentials are entered.
- Login or Password-Reset operations should take the same amount of time, even if the user account for which they should be done does not exist.
 - This is best achieved by emulating all time-consuming operations that would be done in the case of a valid user account, such as hashing a dummy password instead of immediately returning.
 - If it is not possible to properly emulate an operation, it can be approximated by using a sleep function that has been tuned to take a similar amount of time than the operation in question.
 - If possible, run time consuming functions asynchronously.

M3: Web Cache Deception

Score	5.3 (Medium)
Vector string	CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:N/A:N
Target	GET /account
References	<ul style="list-style-type: none">• https://cwe.mitre.org/data/definitions/525.html• https://portswigger.net/daily-swig/path-confusion-web-cache-deception-threatens-user-information-online• https://www.invicti.com/web-vulnerability-scanner/vulnerabilities/web-cache-deception/

Overview

The application was vulnerable to web cache deception, where an attacker could trick the caching mechanism into storing sensitive user-specific pages such as the account dashboard as publicly accessible cached content. This could allow unauthorized users to access private information leading to data theft.

Details

The application did not properly validate or differentiate user-specific responses before storing them in the cache. During testing, it was observed that sensitive, user-specific pages could be cached and served to other users when specific URL patterns or query strings were manipulated.

This vulnerability occurs because the application relies solely on the URL or headers to determine cacheability and does not account for whether the response contains sensitive user data. An attacker could exploit this by crafting URLs that appear cacheable, causing private content to be cached by the server or intermediate proxies and accessible to other users.

Proof of Concept:

If an authenticated user navigated to the `https://example.com/account` page, the response contained sensitive information such as

```
<h2>Welcome, Firstname Lastname!</h2>
...
<p>Balance: €7,532.05</p>
```

Craft a malicious request:

Modifying the URL to append a non-standard extension (e.g., .css, .jpg, or .txt) tricked the caching mechanism into treating the response as a static resource:

```
https://example.com/account.css
```

The server processed the request and responded with the same sensitive content as the original /dashboard page. This response was treated as cacheable because of the .css extension. Additionally, the `Cache-Control` headers in the response allow caching:

```
Cache-Control: public, max-age=3600
```

Retrieving sensitive information:

If a victim was tricked into visiting the maliciously crafted URL and it was cached, an attacker could visit the same page to retrieve sensitive information displayed on that page.

To test this, the page was opened on a different browser. The server responded with the cached content from the original user's session.

```
<h2>Welcome, Firstname Lastname!</h2>  
...  
<p>Balance: €7,532.05</p>
```

Recommendation

- Ensure that sensitive, user-specific pages (e.g., /account) include proper `Cache-Control` headers, set with the directives `no-store` and `private`.
- Implement strict rules to determine which pages are cacheable. Avoid caching responses for URLs that include sensitive content or non-standard file extensions.
- If a CDN is used, configure it to respect cache-control headers and prevent unintended caching of dynamic content.
- Verify that there aren't any discrepancies between how the origin server and the cache interpret URL paths.

L1: Verbose Error Messages

Score	3.7 (Low)
Vector string	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N
Target	<ul style="list-style-type: none">• POST /search• POST /profile• Get /<invalid_path>
References	https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

Overview

The tested application exposed detailed error messages when unexpected inputs were provided. These messages can reveal sensitive information about the system's internal workings, such as server configuration, database structure, or code logic.

While this issue did not allow direct exploitation, it potentially:

- Assists attackers in identifying the database type, server software, and code structure.
- Increases the likelihood of future attacks such as SQL Injection, Remote Code Execution (RCE), or Local File Inclusion (LFI).
- Provides useful information for reconnaissance.

Details

During testing, it was found that certain application endpoints expose verbose error messages when unexpected inputs are provided. These messages include potentially sensitive details such as:

- Stack traces that reveal file paths, function names, or application code structure.
- Web server or framework version information.
- Operating system details

Such information disclosure can help an attacker gather reconnaissance on the application's backend, increasing the likelihood of successful exploitation in future attacks.

Proof of Concept:

Stack Trace Disclosure

Request:

```
GET /search?test= HTTP/1.1
Host: example.com
Content-Type: application/json

{ "username": "test", "password": {"malformed":} }
```

Response:

```
NullPointerException at com.example.auth.LoginHandler.handleRequest(LoginHandler.java:42)
```

This response reveals specific implementation details, including the file name and line number.

Exposing Web Server Configuration Details

Request:

```
GET /invalid-path HTTP/1.1
Host: example.com
```

Response:

```
HTTP/1.1 500 Internal Server Error
Server: Apache/2.4.46 (Ubuntu)
Date: Thu, 28 Nov 2024 15:30:00 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 345

Error: FileNotFoundException at /var/www/html/index.php in Line 20
```

Details Exposed:

Server Version: Apache/2.4.46 (Ubuntu)
Operating System: Ubuntu
File Paths: /var/www/html/index.php
Error Type: FileNotFoundException\

Recommendation

- Ensure that the application displays generic error messages such as "An error occurred. Please try again later."
- Configure the application to log detailed error messages internally but display only minimal information to end users.
- Implement a global exception handler to catch all unhandled exceptions and ensure a uniform, safe error message is returned.
- Log detailed error messages to a secure log file for internal debugging purposes only.