

Project Luna

Web application pentest

Results report

MindBytes GmbH | Welfenstr. 88i | 70599 Stuttgart | Germany

+49 711 20709567 | hallo@mindbytes.de

Represented by Christian Stehle, Nina Wagner, Simon Holl
HRB 790784 | Local Court: Stuttgart

Version 1.0

Confidential

Contact: hallo@mindbytes.de

Example company

Contents

1 Management summary	3	5 Appendix	26
2 Technical summary	5	5.1 Explanations of rating scales	26
3 Findings	8	6 List of changes	26
3.1 FIN-01: Manipulating database queries – SQL Injection	8	7 Disclaimer	27
3.2 FIN-02: User impersonation	11	8 Legal information	27
3.3 FIN-03: Injecting malicious content into website cache	14		
3.4 FIN-04: Spring Boot Actuator enabled	18		
3.5 FIN-05: Execution of sensitive actions without identity verification	20		
3.6 FIN-06: Missing HTTP security headers	22		
4 Project scope	24		
4.1 Persons involved	24		
4.2 Test period	24		
4.3 Test subject	24		
4.4 Access method	24		
4.5 Provided accounts	25		
4.6 Provided information	25		

1 Management summary

Subject of the test: Example web application **Need for action:** Urgent

Overall risk

- Multiple vulnerabilities allow access to other customers' data. This includes personal data, which could lead to privacy violations and reputational damage.
- The security of user accounts is compromised due to various issues. Attackers can gain full control of accounts and view, modify, or delete data.
- The monitoring endpoints of the web server expose sensitive technical information that is not relevant for regular users of the application. For example, database credentials or session cookies of other users are disclosed. This information is useful for malicious actors to continue their attacks.
- Additional configuration can further improve the application's security and make it harder to exploit vulnerabilities.

Overall risk compared to other companies¹: Worse than average

MindBytes-Score

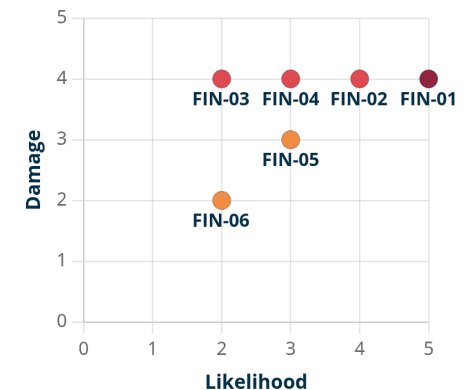


Figure 2 - Distribution according to damage and likelihood

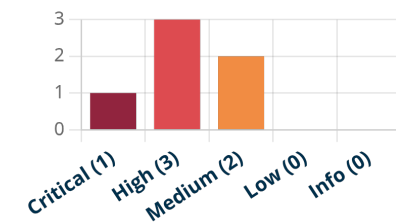


Figure 3 - Distribution according to risk

¹This is a rating in comparison to other companies and does not allow any conclusions to be drawn about the existing risk in general.

1.1 Recommended actions

The estimation for the remediation is based on our experience and should be validated internally. In general, successful attacks result from a combination of several vulnerabilities, which is why we recommend that all findings are rectified. When implementing measures, it is important not to view vulnerabilities as individual cases, but to work on the cause in order to prevent similar vulnerabilities in the future.

Action	Remediation	Notes on remediation	Findings
Deactivate actuator ⚡	🕒 Urgent 🕒 Hours 💰 No	▪ Quick win: Security can be significantly improved with minimal effort	3.4 FIN-04: Spring Boot Actuator enabled
Validate user input	🕒 Urgent 🕒 Days 💰 No	▪ User input should generally be considered malicious and validated before processing	3.1 FIN-01: Manipulating database queries – SQL Injection 3.3 FIN-03: Injecting malicious content into website cache
Prevent account takeover	🕒 Urgent 🕒 Days 💰 No	▪ A combination of the findings allows for the complete takeover of other user accounts	3.2 FIN-02: User impersonation 3.5 FIN-05: Execution of sensitive actions without identity verification
Harden web application	🕒 Medium-term 🕒 Weeks 💰 No	▪ Further improve the security of the application	3.6 FIN-06: Missing HTTP security headers

🕒 Priority: Urgent / Medium-term / Long-term | 🕒 Estimated remediation time per finding: Hours / Days / Weeks | 💰 Cost: No / Probably (not) / Yes | ⚡ Quick win

2 Technical summary

2.1 Table of findings

Finding	CVSS-Score (v3.1)	MindBytes score Damage	MindBytes score Likelihood
3.1 FIN-01: Manipulating database queries – SQL Injection 💡 Use parameterized queries for database queries	<u>9.3 (Critical)</u>	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.2 FIN-02: User impersonation 💡 Check signatures of JWTs	<u>8.1 (High)</u>	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.3 FIN-03: Injecting malicious content into website cache 💡 Generate website content, particularly the cache, without user input	<u>7.5 (High)</u>	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.4 FIN-04: Spring Boot Actuator enabled 💡 Disable the Spring Boot Actuator or restrict its functionality	<u>7.5 (High)</u>	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.5 FIN-05: Execution of sensitive actions without identity verification 💡 Require password verification before performing sensitive actions	<u>5.6 (Medium)</u>	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲
3.6 FIN-06: Missing HTTP security headers 💡 Configure HTTP security headers	<u>4.2 (Medium)</u>	🔥🔥🔥🔥🔥	🎲🎲🎲🎲🎲

Details for each of the findings are described in section 3 Findings. The following files are attached to this report:

📄 Graphical analysis, tabular overview of findings and list of assets with associated findings each asset is affected by:
Project-Luna-Overview.xlsx

🔍 Technical information referenced at relevant points in the findings: Project-Luna-Technical-Details.xlsx

2.2 Next steps

1. Post-processing (see section 2.5 Postprocessing)
2. Viewing and reviewing the results of this report, clarifying questions in the wrap-up meeting
3. Planning and prioritizing remediation measures, e.g., with the prepared table in the "Findings overview" sheet of the attached file
4. Implementation and follow-up of remediation measures
5. Recommended next tests:
 - Retesting the results to check the effectiveness of the implemented remediation measures
 - Physical Red Teaming to check how easily unauthorized persons can enter company buildings and production halls
 - Pentest of the internal infrastructure
 - Periodic repetition of this pentest to check changes made and test for any new attack techniques

2.3 Starting point in the project

Information provided ²	Test scope	Approach	Starting point ³
no (Black-Box)	complete	hidden (Red Teaming)	from outside
some (Grey-Box)	limited	obvious (Pentest)	from inside
comprehensive (White-Box)	focused		

2.4 Project limitations

There were no factors that impaired the implementation of the project.

²Details see section 4.6 Provided information

³Details see section 4.4 Access method and 4.5 Provided accounts

2.5 Postprocessing

1. Delete created exceptions in existing protection systems if no retest or follow-up test is planned
2. Disable provided accounts (see section [4.5 Provided accounts](#)) if you plan a retest or follow-up, otherwise delete those accounts
3. Delete created company MINDBYTES

3 Findings

3.1 FIN-01: Manipulating database queries – SQL Injection

Affected:

CVSS v3.1: [9.3 \(Critical\)](#)

- <https://example.com>

3.1.1 Summary

The application uses user input to create SQL queries. This allows manipulated queries to be sent to the database, enabling access to additional stored information.

Possible consequences of successful exploitation 🔥🔥🔥🔥

- Executing custom SQL queries
- Can generally be utilized for various actions:
 - Reading additional stored sensitive data
 - Triggering valid database queries without valid input values to correctly execute application flows such as logins
 - Modifying or deleting data
 - Overloading the database with resource-intensive queries
 - Abusing database functions, for instance to access files, execute operating system commands, or make network requests to internal systems

Examples of prerequisites for exploitation 🎲🎲🎲🎲

- We were able to exploit the vulnerability without authentication
- In-depth knowledge of SQL and the vulnerability is useful
- Tools are available that assist in exploiting the vulnerability

3.1.2 Recommendation

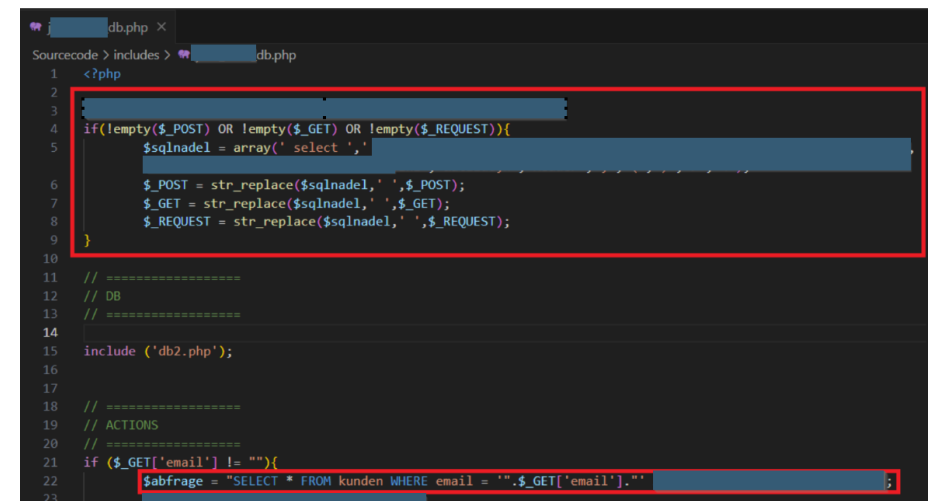
- Use [Prepared Statements with parameterized queries](#):
 - This allows the query and the required user input to be sent to the database separately
 - The database correctly handles user input, preventing query manipulation
 - An overview of how this can be implemented in different programming languages is provided in this [blog post](#)
- Look for additional locations within the application that might also be affected

3.1.3 Technical Details

In multiple locations the web application uses user input to construct database queries. We scanned the source code for this vulnerability using the tool [semgrep](#). The results of this tool, including further potentially vulnerable locations, can be found in the attached *semgrep-sql.txt*. Due to the large number of potentially vulnerable locations, we did not manually verify all of them.

Below, a specific location is described where, after bypassing the implemented filter, manipulated queries can be constructed to extract user information. The adjacent image shows the vulnerable location, including the implemented filter, in the file */includes/example_db.php*.

When accessing the URL https://example.com/includes/example_db.php?email=1\&hash=%20OR%201=1%20LIMIT%2010%20--%20, the application uses both the *email* and *hash* parameters to create a database query.



```
1 <?php
2
3
4 if(empty($_POST) OR empty($_GET) OR empty($_REQUEST)){
5     $sqlnadel = array(' select ',
6
7     $_POST = str_replace($sqlnadel, ' ', $_POST);
8     $_GET = str_replace($sqlnadel, ' ', $_GET);
9     $_REQUEST = str_replace($sqlnadel, ' ', $_REQUEST);
10 }
11
12 // DB
13 //
14
15 include ('db2.php');
16
17
18 // ACTIONS
19 //
20
21 if ($_GET['email'] != ""){
22     $abfrage = "SELECT * FROM kunden WHERE email = '$_GET['email']'";
23 }
```

Figure 4 - Vulnerable source code

The inputs are initially filtered to prevent possible query manipulations. However, this filter can be bypassed. When the above URL is accessed, the following SQL query is constructed:

```
SELECT * FROM kunden WHERE email = '1\' AND hash = ' OR 1=1 LIMIT 10 -- '
```

The inserted backslash (\) from the *email* parameter neutralizes the following quotation mark. The *hash* parameter is then used to introduce an always-true condition (*1=1*) and limit the number of results to 10 rows (*LIMIT 10*). Also, the rest of the predefined query is commented out (--).

Furthermore, the filter can be bypassed by writing the SQL keywords in uppercase. During execution, the database does not distinguish between uppercase and lowercase, but the implemented filter does. Exploiting the vulnerability without knowledge of the implemented filter is significantly more challenging but not impossible. A very targeted approach is required to succeed.

The adjacent image shows the execution of the mentioned URL in the browser.

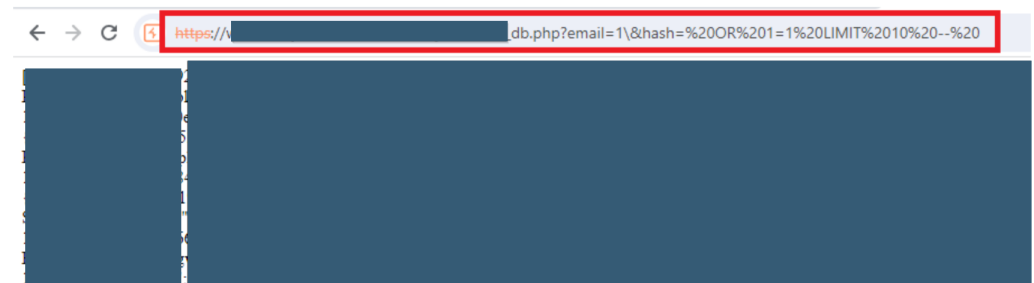


Figure 5 - Impact of the manipulated database query

Since some potentially vulnerable locations involved SQL queries with the *UPDATE* keyword, we assume that data could also be maliciously manipulated. To avoid impacting operations, we did not verify this.

3.2 FIN-02: User impersonation

Affected:

CVSS v3.1: [8.1 \(High\)](#)

- <https://example.com>

3.2.1 Summary

Due to a weakness in the validation of access tokens, users can take over any other user account and thereby access data from other organizations.

Possible consequences of successful exploitation 🔥🔥🔥🔥

- Takeover of other user accounts
- Access to data from other organizations, in particular

Examples of prerequisites for exploitation 🎲🎲🎲🎲🎲

- Knowledge of a valid JSON Web Token for any user, e.g., by accessing the publicly available demo environment
- Basic understanding of how JSON Web Tokens (JWT) work
- Knowledge of the user ID of the other user account:
 - These are used in URLs and exposed in various locations, such as browser histories and server logs; see also [RFC 3986](#)

2. Login with own user account, pentest01:

Log in normally to the application using the account *pentest01*. A JWT is then issued. The JWT is used for authentication in API calls via the HTTP `Authorization` header. The illustration for analyzing the JWT shows its content in a Base64-decoded format. The adjacent image shows an example of a legitimate HTTP call to an endpoint of Organization1 with a valid JWT for user *pentest01*.

3. JWT Manipulation:

Copy the middle part of the JWT (delimited by dots `.` as marked in the previous illustration), Base64-decode it, replace the ID in the `sub` parameter with the ID of user *pentest04* (`asd123-dead-beef-a12f-09asdoaijs22`), and Base64-encode it again. Replace the middle part of the JWT in the HTTP request with the manipulated version.

4. Impersonating user pentest04:

With the manipulated JWT, access to resources of MB-Organization2 is now possible, where user *pentest04* (and not *pentest01*) has privileges. An example is accessing the customer database of MB-Organization2. The successful call is shown in the adjacent image. The customer number required for accessing the endpoint must also be known and can likely be easily obtained through internet research.

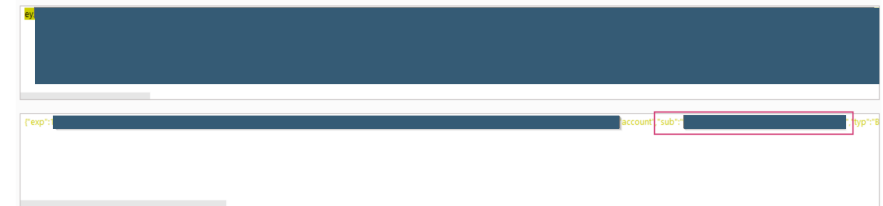


Figure 7 - Changing the `sub` parameter in the JWT to the user ID of *pentest04*

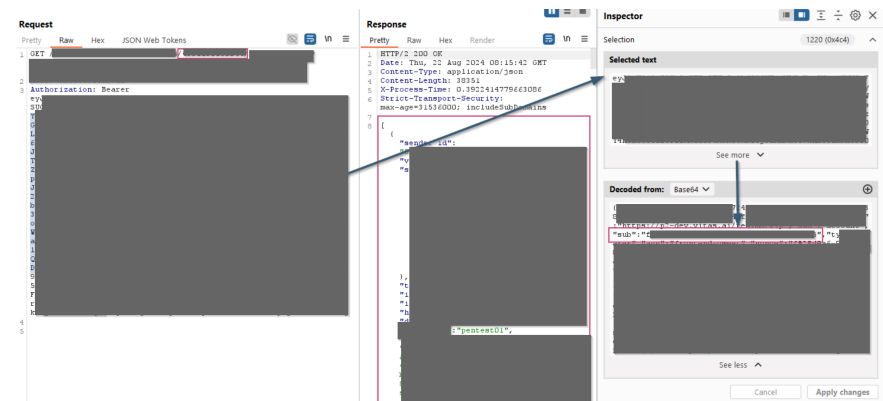


Figure 8 - Requesting data from MB-Organization2 with forged JWT

3.3 FIN-03: Injecting malicious content into website cache

Affected:

CVSS v3.1: [7.5 \(High\)](#)

- <https://example.com/>

3.3.1 Summary

Some website content is generated based on user input. Then the generated content is stored in the cache and delivered to other website visitors. Attackers could exploit this to inject malicious content and thereby attack other visitors.

Possible consequences of successful exploitation 🔥🔥🔥🔥

- Injecting malicious content, which is delivered to other website visitors, with the following impact:
 - Manipulating the appearance of the website
 - Executing malicious JavaScript code
 - Redirecting users to malicious websites

Examples of prerequisites for exploitation 🎲🎲🎲🎲

- Attackers must carefully analyze the caching behavior of the website
- Malicious content must be reinserted when the cache expires
- Users need to navigate to a cache entry manipulated by the attacker on the homepage
 - This could potentially be the entire website or it could happen by sharing of URLs to manipulated pages

3.3.2 Recommendation

- Evaluate whether the page content can be generated without relying on user input from the headers *X-Forwarded-Host* and *X-Forwarded-Prefix*
- If this is not possible, implement one of the following measures:
 - Overwrite any user input in headers from systems involved in communication with predefined values
 - Filter received values against a list of valid values
- Look for other headers, parameters, or locations that may be used to generate page content and can be influenced by users
- Possible temporary solution, until the vulnerability is resolved:
 - Disable the caching mechanism (this would also take away the benefits of caching, though)

3.3.3 Technical Details

The server cache content is generated based on the headers *X-Forwarded-Host* and *X-Forwarded-Prefix*.

Both headers can be set by users in HTTP requests and filled with malicious content. The content placed in the cache is then delivered to other visitors accessing the website. This type of attack is also known as [Web Cache Poisoning](#).

Based on our observation, the cache is valid for 200 seconds. After this time, the cache is renewed when the next website is accessed next. Therefore, after this time has elapsed, attackers must reinsert their malicious content before a legitimate user accesses the website.

To avoid disrupting regular website operations, we created a separate cache entry using the URL parameter *cachebuster*. This ensured that the cache entries required for regular website access remained untouched. However, attacks are also possible on the regular cache entries, which would affect every legitimate website visitor.

The following sections describe potential attacks resulting from the caching behavior.

Executing malicious JavaScript code

When calling the JavaScript resource [global.js](#), the application uses the HTTP header *X-Forwarded-Host* to generate parts of the code. This allows attackers to extend the functionality of the script and insert malicious code on every page that includes the script.

Execution of the injected code can be triggered by accessing the URL <https://example.com>. The manipulated script is embedded in a large number of pages, increasing the likelihood of execution.

To avoid disrupting production operations, the *cachebuster* parameter was used in this case as well. This parameter is automatically added by the proxy tool *Burp Suite* when loading the JavaScript resource. An attacker would need to manipulate the correct cache entry of the script to successfully execute the injected code.

Redirecting to malicious websites

When accessing the homepage and other existing directories, such as <https://www.example.com>, the application uses the header *X-Forwarded-Host* to generate the correct user redirection. The redirection is also stored in the website cache. Manipulating the HTTP header leads to a redirection to the specified domain, which could potentially be a malicious website.

A manipulation as shown in the figure causes other users accessing the URL <https://www.example.com/?cachebuster=mindbytes> to be redirected to the MindBytes website.

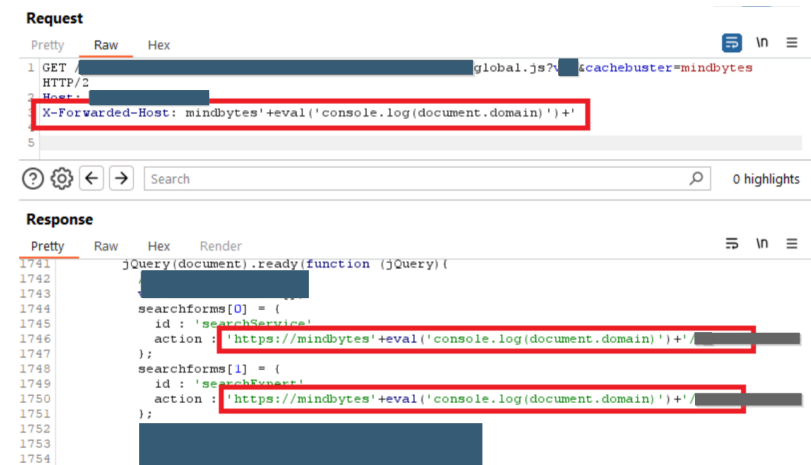


Figure 9 - Injecting JavaScript code into the cache



Figure 10 - Injecting a malicious redirect into the cache

Manipulating web content

For generating page content, both the *X-Forwarded-Host* and *X-Forwarded-Prefix* headers are used. Specifically, the provided content was embedded on the website as the HTML tag `<base href="https://<forwarded-host>/<forwarded-prefix>">`, among others. This meta tag instructs the browser to load all relative URLs from the specified domain or URL.

When accessing the manipulated website, this domain was blocked by the browser. This was due to the configured `Content-Security-Policy`, which allows loading resources from the site's own and selected domains only.

Request

```
1 GET /...html?cachebuster=mindbytes HTTP/2
2 Host: ...
3 X-Forwarded-Host: mind-bytes.de
4 X-Forwarded-Prefix: suburl
5
6
```

Response

```
21
22
23
24
25
26
27
28 <!doctype html>
29 <html lang="de">
30
31 <base href="https://mind-bytes.de/suburl/">
32
33 <title>
```

Figure 11 - Injecting manipulated content

3.4 FIN-04: Spring Boot Actuator enabled

Affected:

CVSS v3.1: [7.5 \(High\)](#)

- <https://example.com/>

3.4.1 Summary

The Actuator provides functions for monitoring the application and is part of the Java Spring Framework. The Spring Boot Actuator allows extracting detailed information valuable to attackers. In this specific case, it enables the takeover of sessions belonging to other users.

Possible consequences of successful exploitation 🔥🔥🔥🔥

- Extraction of sensitive information
- Access to session cookies of other users, enabling subsequent access to the application with their permissions
- Additional attacks made possible by utilizing the sensitive information, such as using disclosed credentials

Examples of prerequisites for exploitation 🎲🎲🎲🎲

- A valid user account is required for most endpoints
- The URL must be known – can be easily guessed
- The attacker may need to be in the correct network position to utilize the information
 - For example, the database must be accessible to use the credentials

3.4.2 Recommendation

- Evaluate whether the Actuator is necessary
- Disable the Actuator or restrict its functionality
- Change any potentially exposed sensitive information, such as database credentials

3.4.3 Technical Details

The Spring Boot Actuator is accessible at the URL <https://example.com/actuator>, listing all available functionalities.

In particular, the endpoint [/actuator/heapdump](#) is interesting for attackers. This endpoint allows a snapshot of the Java Heap to be created. The Heap is the memory area where the application stores its objects. For instance, it includes environment variables as well as objects related to current HTTP requests.

The illustration shows the heap, which was exported with the regular user `pentest1`, in a text editor. The snippet reveals that the heap contains details of current HTTP requests. Among other things, we were able to access the session cookies `JSESSIONID` of other active users.

The extracted session cookie is valid and can be used in a browser. This allows access to the application as another user.

The exported heap also contained other sensitive information. For example, it included the values of the environment variables `db.url`, `db.username`, and `db.password`. This would likely enable access to the database, provided it can be reached over the network.

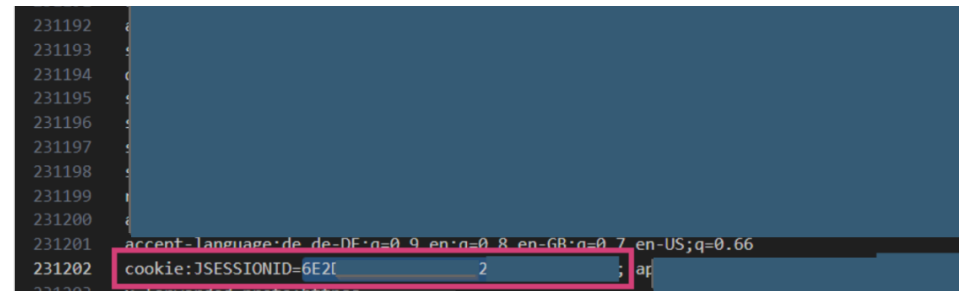


Figure 12 - Viewing a Session Cookie within the exported heap

3.5 FIN-05: Execution of sensitive actions without identity verification

Affected:

CVSS v3.1: [5.6 \(Medium\)](#)

- <https://example.com/>

3.5.1 Summary

Sensitive actions can be performed without first verifying the legitimacy of the user. Specifically, users can change their password without re-entering their current password. This increases the risk of account takeover in the case of unattended sessions.

Possible consequences of successful exploitation 🔥🔥🔥🔥

Attackers with access to an active user session can perform sensitive actions without restriction:

- Specifically, the password can be changed, granting the attacker full control over the account
- The legitimate user would (temporarily) lose access to the account
- The attacker would be able to perform any action permitted by the account's privileged
- Often enables access to sensitive information or deletion of the account

Examples of prerequisites for exploitation 🎲🎲🎲🎲

There are several attack scenarios where this lack of protection can be exploited:

- Attackers use a cross-site scripting vulnerability, for example, to perform the sensitive action remotely
- A user leaves an authenticated session unattended, e.g., on shared computers

3.5.2 Recommendation

- Verify the user's password or use a different authentication method before executing sensitive actions

3.5.3 Technical Details

The illustration shows that the user's password can be changed without requiring the old password to be entered.

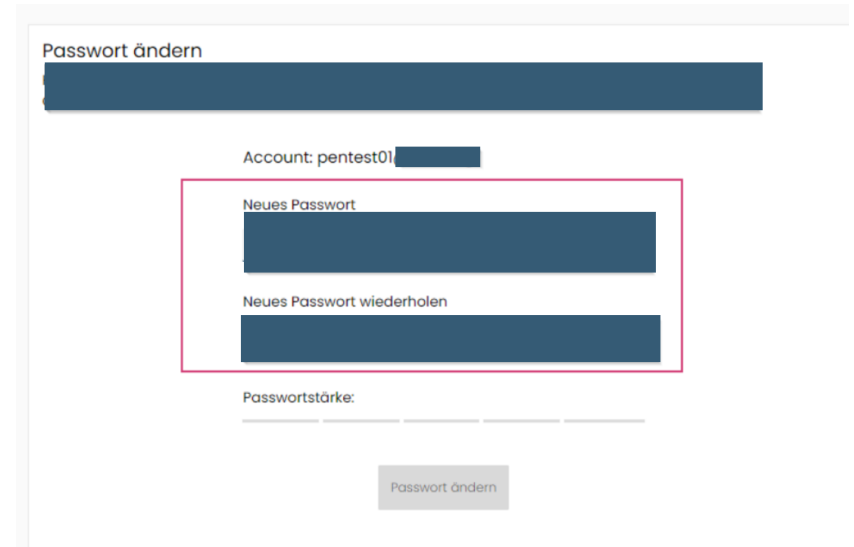


Figure 13 - Password change without knowledge of the old password

3.6 FIN-06: Missing HTTP security headers

Affected:

CVSS v3.1: [4.2 \(Medium\)](#)

- <https://example.com/>

3.6.1 Summary

Configuring HTTP security headers can increase the security of web applications. Typical attacks become more difficult or can be prevented altogether. The headers should be seen as an additional security measure and do not replace the secure programming of web applications.

Possible consequences of successful exploitation 🔥🔥🔥🔥

- Exploiting typical vulnerabilities, such as clickjacking or cross-site scripting, is easier when there are no security headers; configuring them can make exploits more difficult or prevent them altogether

Examples of prerequisites for exploitation 🎲🎲🎲🎲

- Scenarios in which the headers offer security-relevant advantages require vulnerabilities in the application and/or user interaction

3.6.2 Recommendation

- If possible, set the missing security headers on all listed websites (see "Technical details")
- Configuration recommendations can be found in the respective sections of the OWASP Cheat Sheets:
[Strict-Transport-Security](#) | [Content-Security-Policy](#) | [X-Frame-Options](#) | [X-Content-Type-Options](#) | [Referrer-Policy](#) | [Permissions-Policy](#)
- Improve header configuration for the following websites:
 - <https://example.com>
 - Don't use the *unsafe-eval* and *unsafe-inline* directives in the content security policy
- The configured content security policy can be evaluated with the [CSP Evaluator](#)

3.6.3 Technical Details

The following table provides an overview of the configured security headers on the websites.

- Headers marked with 👍 are set and classified as appropriate
- Headers marked with ⚠️ are set, but the configuration should be checked and improved
- Headers not marked are not set

Website	Strict-Transport-Security	Content-Security-Policy	X-Frame-Options	X-Content-Type-Options	Referrer-Policy	Permissions-Policy
https://example.com	👍	⚠️				

4 Project scope

4.1 Persons involved

Name	Role	Mail address
Simon Holl	Project lead & Pentester	hallo@mindbytes.de
Christian Stehle	Pentester	hallo@mindbytes.de
Nina Wagner	Pentester	hallo@mindbytes.de
Anja Neudert	Review	
Sarah Smith	Head of IT	sarah.smith@example.com

4.2 Test period

1/2/25–1/10/25

4.3 Test subject

Asset type	Value	Description
Web application	https://example.com/	Customer portal

4.4 Access method

Access took place over the Internet.

4.5 Provided accounts

Account	Role/Privileges
pentest01	Role: user
pentest02	Role: user
pentest03	Role: admin
pentest04	Role: admin

4.6 Provided information

To enable targeted and efficient assessments, the following data was provided:

- Source code of the web application

5 Appendix

5.1 Explanations of rating scales

	Common Vulnerability Scoring System (CVSS)	MindBytes score
Explanation	- Standardized rating system for the severity of security vulnerabilities in software and systems - Technical rating - De facto industry standard	- MindBytes' evaluation system with a risk-based approach and focus on (potential) damage and likelihood - In this context, likelihood means how easily a vulnerability can be exploited - The score is based on the CVSS rating but also takes into account the number and importance of the affected systems
Rating scales	Scale from 0 (Info) to 10 (critical) for classifying the severity of a vulnerability	Scale from 0-5 for classifying damage and likelihood

6 List of changes

Version	Date	Change	Who
1.0	17.01.25	Release	Christian Stehle

7 Disclaimer

This project was carried out in order to assess the security of the components in focus and to identify weaknesses.

1. This test is a snapshot and not a continuous security monitoring. The security situation may change over time, for example due to changes to the components, disclosed information, new attack techniques or vulnerabilities.
2. The project was carried out within a limited time frame. This may mean that not all potential vulnerabilities and disclosed information were identified.
3. Even though the project was carried out with great care, false positives cannot be completely ruled out.

8 Legal information

MindBytes GmbH | Welfenstr. 88i | 70599 Stuttgart | Germany

+49 711 20709567 | hallo@mindbytes.de | <https://mindbytes.de>

Local Court: Stuttgart, HRB 790784 | VAT number: DE363069855

Represented by **Christian Stehle, Nina Wagner, Simon Holl**