CUSTOMER ACME CORPORATION

SUBJECT WEB APPLICATION

DOCUMENT SECURITY ASSESSMENT REPORT

1. EXECUTIVE SUMMARY

During the period between 2024-12-16 and 2024-12-20, Portal Cybersecurity AB conducted a security assessment of the ACME Corporation’s web application.

The objective was to evaluate the platform's resilience against potential exploitation and provide actionable insights to improve its overall security posture.

ACME handles user data, therefore the security assessment aimed to identify potential attacks that attackers could access user data or compromise the web application’s integrity.

This report presents the findings of the assessment, providing technical details about the identified vulnerabilities along with recommendations for their mitigation.

1.1 **Results**

During the security assessment of ACME Corporation's web application, there were several serious identified vulnerabilities that could expose the platform to unauthorized access, data breaches, and system misuse. These issues include weaknesses in the handling of user inputs and data uploads, making it possible for attackers to exploit the system without having login credentials.

Key risks include the potential for attackers to access sensitive user and administrative data.

1.2 **Recommendations**

High-severity vulnerabilities pose significant risks and require immediate attention to safeguard the organization’s security posture, but it should be noted that addressing all identified vulnerabilities including those of lower severity contributes to strengthening the application’s overall defences and preventing potential exploitation.

2. FINDINGS AND RECOMMENDATIONS

This section of the report groups vulnerabilities together at a high level and provides recommendations on improving the application's security posture.
More detailed vulnerability descriptions can be found in Section 3.

2.1 **Approach to Testing**

The security assessment was conducted as a web application assessment, focusing on identifying weaknesses, misconfigurations, technical flaws and vulnerabilities within the ACME Corporation web application.

The OWASP Top 10:2021 and OWASP Web Security Testing Guide are well known and comprehensive frameworks for testing web applications. These frameworks have served as a reference throughout the test, where it was applicable.

Security testing is not an exact science and it is not possible to list every single test-case that can be performed against an application, but some common areas are:

- Authentication and Authorization
- Injection attacks
- Error handling and information disclosure
- Cryptography
- Business logic

Certain vulnerabilities are better suited to be tested in an automatic fashion, others are better to be tested manually, and some are more easily identifiable using the available source code. Therefore, a combination of techniques has been employed throughout the test.

No test accounts or access to source code were provided by ACME, making this a black-box style assessment.

The primary tools used during the assessment have been Burp Suite Community Edition.

2.2 Findings and Recommendations

The security assessment of ACME Corporation's web application identified several areas susceptible to critical vulnerabilities, including: SQL Injection (SQLi) Cross-Site Scripting (XSS), and XML External Entity (XXE) exploitation.

The SQLi vulnerability in the product filter could allow attackers to run unauthorized database queries, exposing sensitive data such as login credentials.
XSS vulnerabilities in the blog search function and stock check page could allow attackers to run harmful scripts in a user's browser, which could result in session hijacking or misleading visitors and users with manipulated content.
Additionally, the XXE vulnerability in the comment avatar upload feature, could let attackers to upload files containing malicious scripts that could access server information.

To resolve these issues, database queries should use prepared statements to prevent SQLi, stricter rules for user input need to be applied, ensuring only expected data is processed. For example, input fields should restrict acceptable characters and formats based on their purpose. Outputs that include user data should also be encoded to prevent scripts or commands from being executed.
File uploads should have controls on allowable file types with measures to confirm they meet the expected format. For XML processing, disabling external entity features would prevent potential attackers from accessing server information.

By addressing the root cause of the applications way of handling user-supplied data, the customer web applications resilience against various types of injection attacks can be significantly enhanced. While specific, technical remediation steps for each identified vulnerability are detailed in Section 3, the recommendations provided here serve as a high-level overview for improving the overall security posture of the web application.

2.3 Delimitations and restrictions

The security assessment was conducted with several limitations to ensure minimal disruption. Testing was performed using black-box testing methods without access to the application's source code.

The assessment scope was limited to administrator and employee accounts; the ACME's visitor or user accounts were out of scope.

While we tested for potential access to server files, this was restricted to validating access without retrieving or altering any files beyond confirming the server hostname.

Script execution was also used, as method to identify vulnerabilities but no data was extracted from users or other parties interacting with the application.

3. RESULTS AND RECOMMENDATIONS

3.1 Severity ratings

Severity	Description
High	Security vulnerabilities that can give an attacker total or partial control over a system or allow access to or manipulation of sensitive data
Medium	Security vulnerabilities that can give an attacker access to sensitive data, but require special circumstances or social methods to fully succeed.
Low	Security vulnerabilities that can have a negative impact on some aspects of the security or credibility of the system or increase the severity of other vulnerabilities, but which do not by themselves directly compromise the integrity of the system.
Info.	Informational findings are observations that were made during the assessment that could have an impact on some aspects of security but in themselves do not classify as security vulnerabilities.

Table 1: Severity ratings.

3.2 Outline of identified vulnerabilities

Vulnerability	High	Medium	Low	Info.
3.3.1 SQL Injection (SQLi)	✓			
3.3.2 Stored Cross-Site Scripting (XSS)	✓			
3.3.3 XML External Entity injection (XXE)			✓	

Table 2: Identified vulnerabilities.

3.3 Technical description of findings

3.3.1 SQL Injection (SQLi)

Severity: High

Description

SQL injections (SQLi) allows an attacker to interfere with the queries that an application makes to its database. This can allow attackers to access data that they are not normally able to retrieve. This might include server stored data that belongs to other users, or other data that the application can access. In many cases, an attacker can modify or delete this data, causing changes to the application's content or behaviour.

During the assessment, it was discovered that the web application was vulnerable to SQLi within the "We Like to Shop" section. This vulnerability could allow attackers to **manipulate** the application's SQL queries by injecting input into **concatenated** strings, directly altering the query logic. By exploiting this vulnerability, it was possible to bypass authentication and authorization mechanisms.

Through a series of crafted GET requests it became possible to access database tables containing information including login credentials, which were successfully retrieved as illustrated in Image 1, using the payload in Example 1.

```
GET /filter?category=Corporate+gifts' UNION SELECT username_ffevah, password_liqxad FROM users_gtfvbh --
HTTP/2
Host: 0a1c00f503f209db804aadf50033005b.web-security-academy.net
Cookie: session= […]
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application
/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

Example 1: SQL injection payload

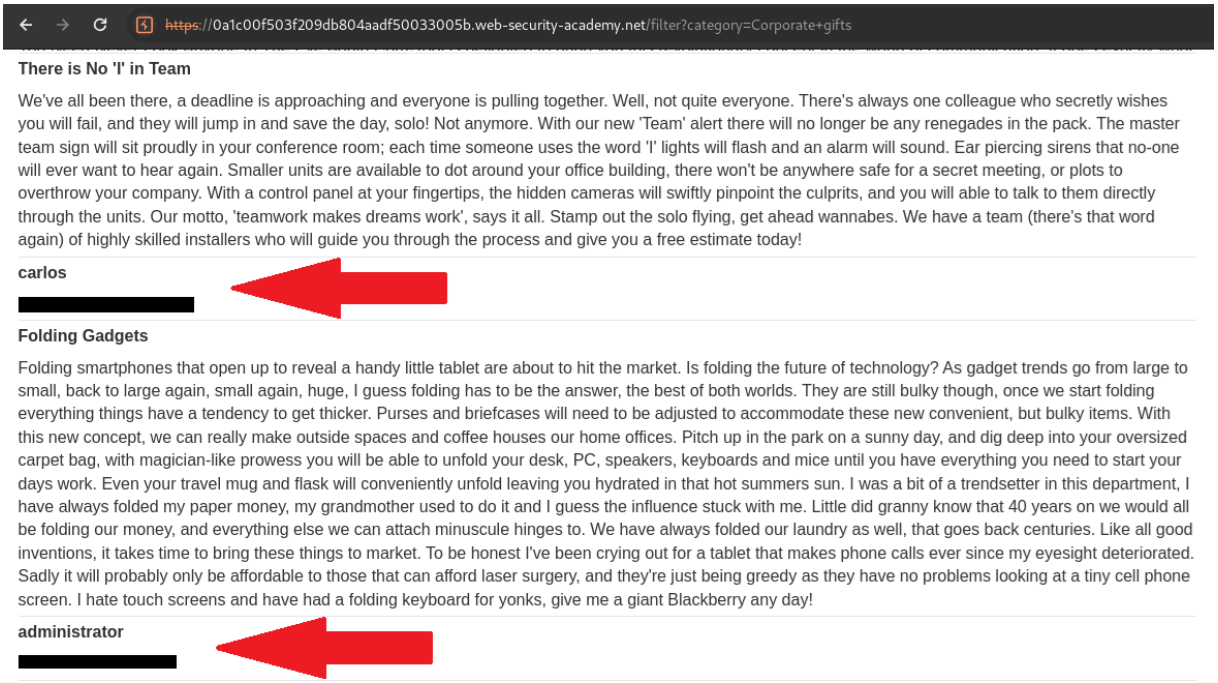

Image 1: SQLi output, employee and administrator login credentials

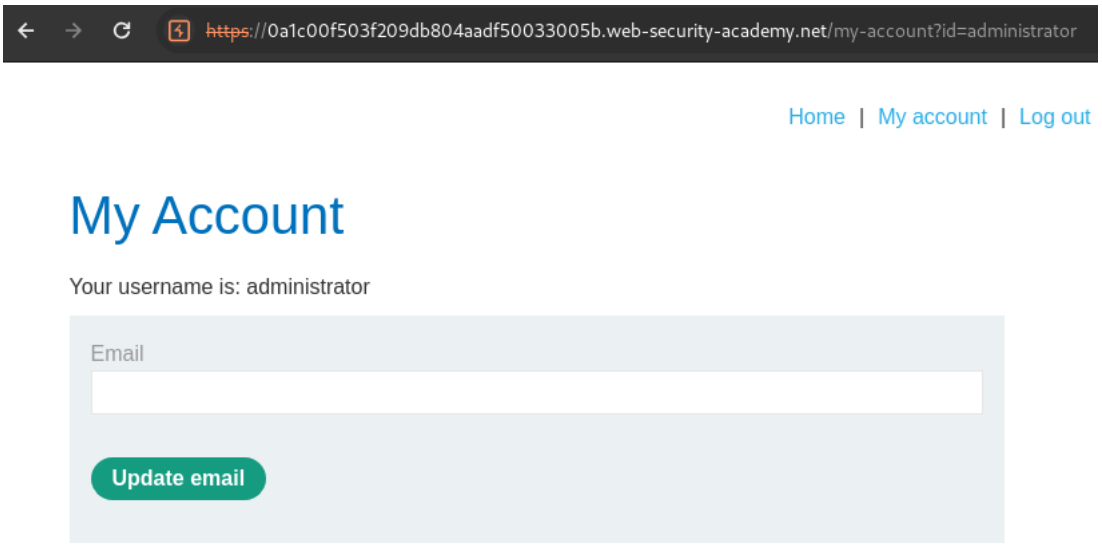

Image 2: Proof of concept, confirmed credentials

Recommendations

Ensure that queries are processed as expected formats by using:

- Parameterized queries

  or

- Prepared statements

Further; Ensure stored login credentials such as passwords are securely hashed, in case of a database breach.

For more information on Sequel injection mitigation and hashing, see:

Open Web Application Security Project – SQL Injection Prevention Cheat Sheet[1].

Open Web Application Security Project – Query Parameterization Cheat Sheet[2]

Open Web Application Security Project – Password Storage Cheat Sheet[3]

[1] https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
[2] https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html
[3] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

3.3.2 DOM based Cross-Site Scripting (XSS)

Severity: High

Description

Cross-Site Scripting (XSS) represents a type of attack that enables an attacker to deceive a user's web browser into running harmful scripts within the context of a vulnerable application. An attacker executing an XSS attack can hijack the user's session, acting within the application as the affected user.

A type of XSS is DOM-based XSS which occurs when a website's JavaScript processes user-input insecurely, allowing attackers to execute scripts. This typically involves an attacker placing harmful in a URL or input-field in the web application, which is then handled improperly from a security perspective. While DOM manipulation is essential for modern websites, poor handling of data can lead to serious security issues like account hijacking.

During the security assessment it was discovered that the ACME web application was vulnerable to two instances DOM-based XSS:

The first vulnerability showed that the web application was susceptible to XSS due to its user input handling in the search blog function, the JavaScript payload demonstrated in Example 2 was successfully run within this functionality.

The JavaScript payload demonstrated in Image 3 successfully executes in the search function, as seen in Image 4.

```
{{constructor.constructor('alert("XSS")')()}}
```

Example 2: Constructed XSS input

Image 3: XSS payload in search function

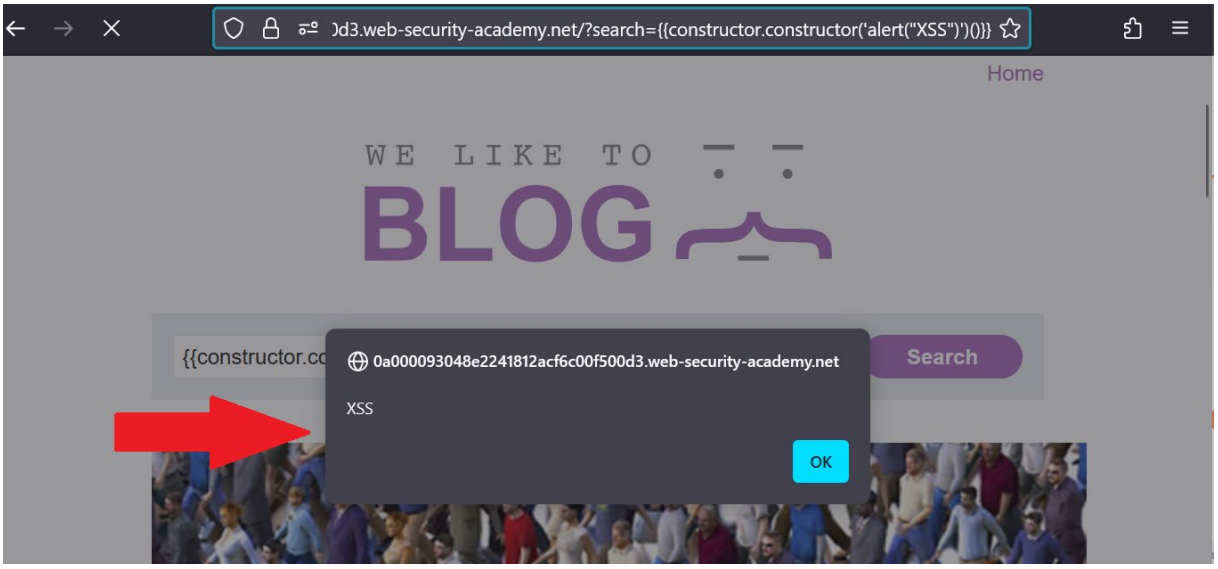

Image 4: XSS 1

The second vulnerability found in the product Check stock feature, this was due to the usage functions as *document.write()* based on the data from the URL processed by the *location.search* function, as seen in Image 5.

A URL query was constructed as seen in Example 3 and it was possible to run a script, which showed an alert message as seen in Image 6. This showed that it is possible for an attacker to manipulate the URL as the document.write() function makes the URL input act as HTML or JavaScript, as shown in Image 6 the URL processing enables XSS.

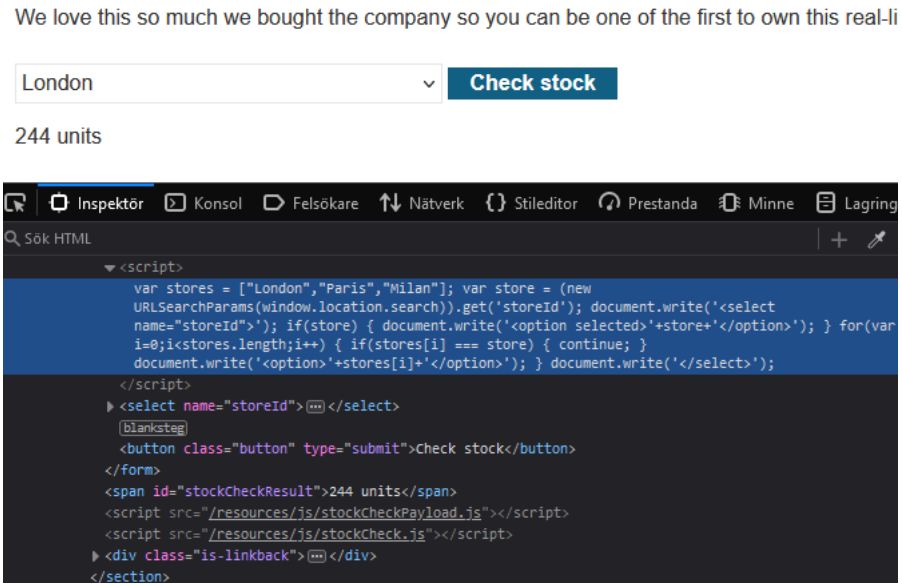

Image 5: Check stock inspected element

```
&storeID=</select><img%20src="123"%20onerror=alert("XSS")>
```

Example 3: Constructed XSS input for URL

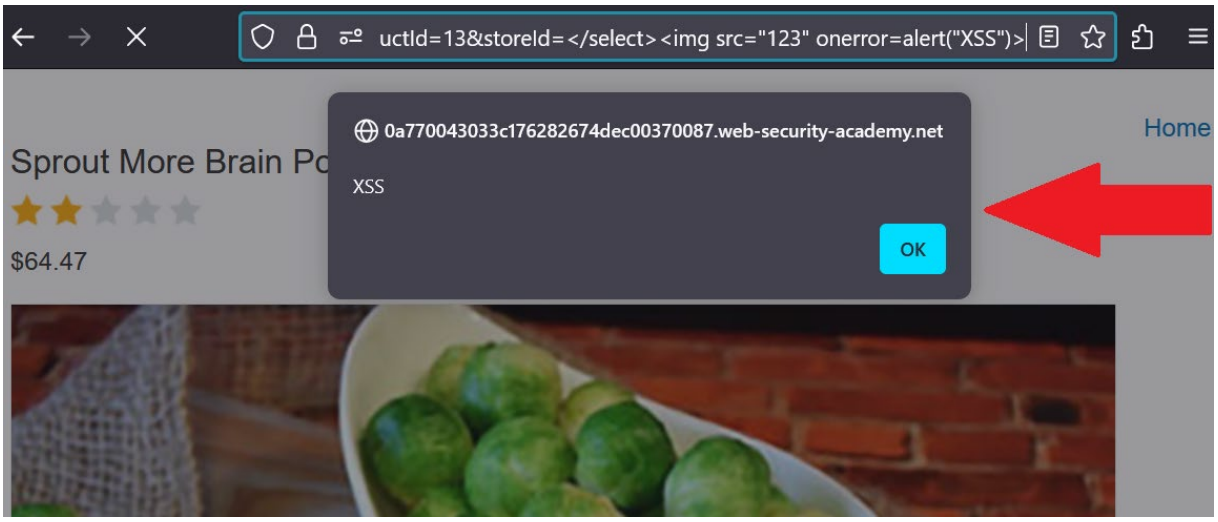

Image 6: XSS 2

Recommendations

To prevent DOM based Cross-Site Scripting, make sure always consider user supplied data as untrusted.

- Avoid user input being directly processed by the DOM by the using innerHTML or document.write()
- Make sure to treat the input as plain text to prevent scripts from being executed.

For more information on Cross-Site Scripting mitigation, see:

Open Web Application Security Project – XSS Prevention Cheat Sheet[4].

[4] https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

3.3.3 XML External Entity (XXE) injection

Severity: Low

Description

XML External Entity (XXE) injection is a web security vulnerability that allows an attacker to interfere with an application's processing of **XML data**. This type of attack allows an attacker to view files on the application server filesystem and to interact with any back-end or external systems that the application itself can access. The potential impact could be access to sensitive server-side information.

When testing the blog comment feature, which allows external users to upload avatars a vulnerability was discovered due to that the web application uses the Apache Batik library to process **SVG image files** but it failed to restrict unsafe XML features such as external entity injection.

An SVG file was uploaded containing an external entity reference, the uploaded file was then processed by the server, presenting internal server-side information as seen in Image 9.

The following crafted code was used to create an SVG file to exploit the XXE vulnerability:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg [
  <!ENTITY file SYSTEM "file:///etc/hostname">
]>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="500" height="500">
  <text x="10" y="50" font-size="50">&file;</text>

</svg>
```

Example 4: XXE injection payload in SVG-file

The image below shows the attached SVG file (picture.svg) being accepted as a valid file.

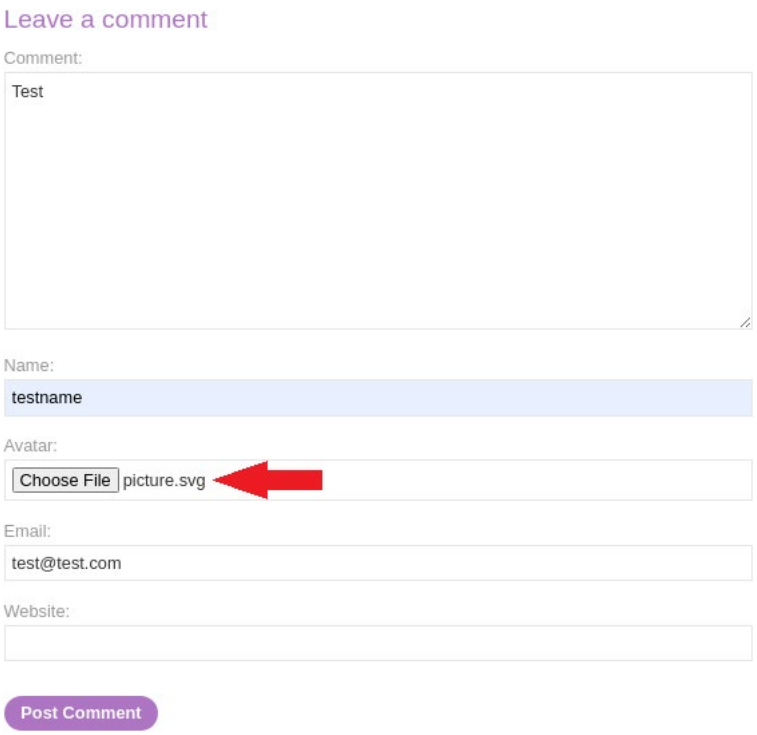

Image 7: Uploading SVG file containing XXE payload

In the image below, an example can be seen of the successfully uploaded comment with the SVG file text in the avatar section.

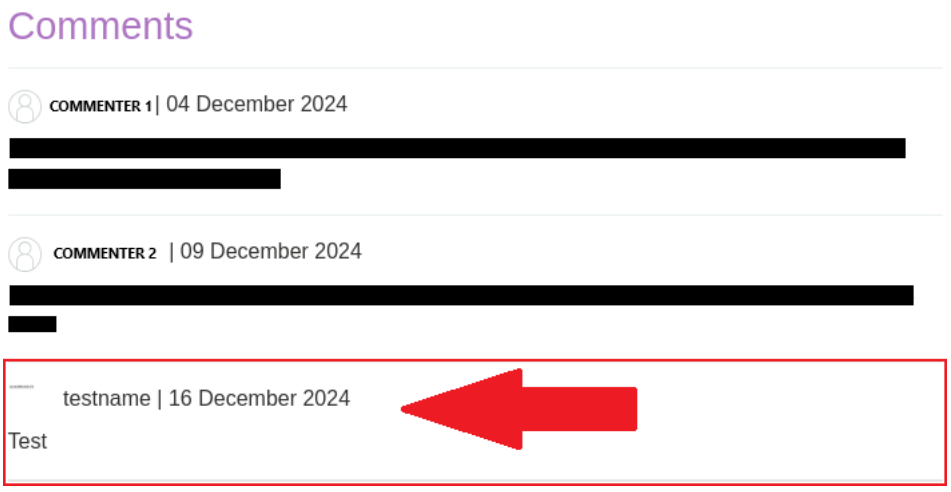

Image 8: Uploaded comment with SVG-file containing XXE payload

The image below shows the hostname of the server displayed in the avatar image.

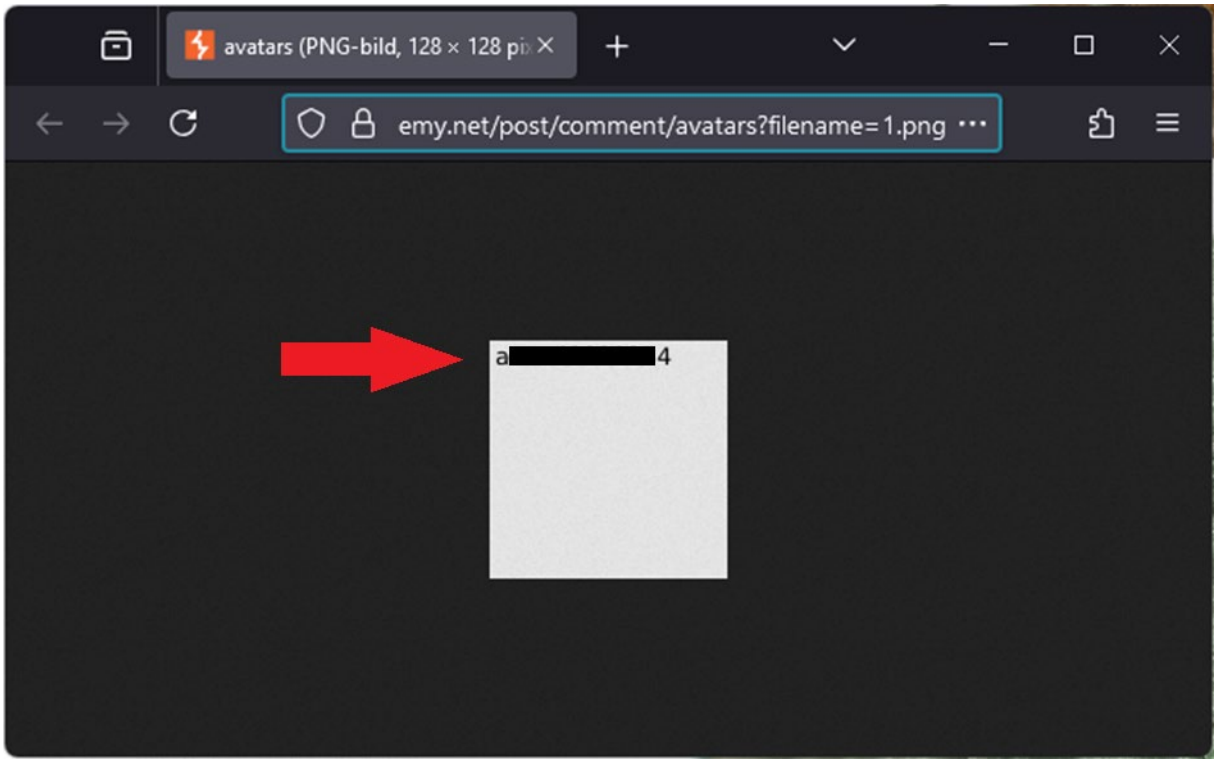

Image 9: Proof of concept, the server hostname extracted using the vulnerability.

Recommendations

It is recommended to adopt stricter controls:

- Disable External Entity Processing: Ensure the XML parser is configured to block external entities.
- Restrict Accepted File Types: Limit uploads to specific, safe file formats like PNG or JPEG, and reject XML-based files.
- Validate Uploaded File Content: Verify that uploaded files match the expected format and structure to avoid processing unauthorized content.

For more information on mitigating XML External Entity vulnerabilities, see:
Open Web Application Security Project – XXE Prevention Cheat Sheet[5].

[5] https //cheatsheetseries owasp org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet html