**REPORT ON DAMN VULNERABLE WEB APPLICATION IN LOW AND MEDIUM SECURITY**



*REPORT COMPILED BY: ALUMASA STEPHEN*

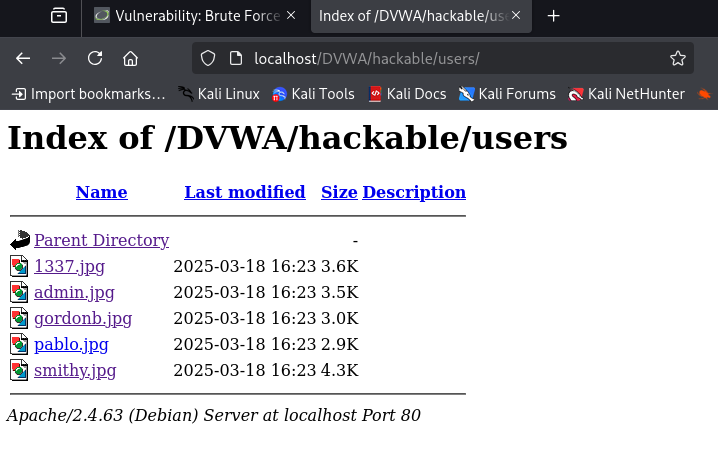
*YEAR: 2025*

# BRUTE FORCING

Before attempting anything first we have to find if the website has other users.

This can be done by altering the url to /DVWA/hackable/users/

This gives us a list of the users in the website:

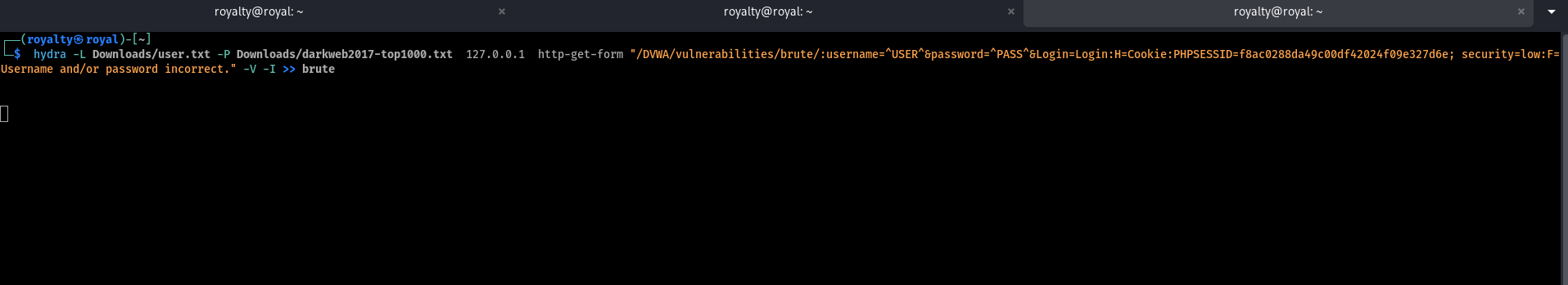


## using hydra command [LOW]

* Hydra is a tool used by attackers to bruteforce passwords using a list of passwords and/or usernames.
* It tends to be easier and faster to bruteforce login.
* It makes it possible for researchers and security consultants to show how easy it is to gain unauthorized access to a system remotely.

Concept proof:

To use hydra we have to call it out by simply saying hydra, followed by the command.

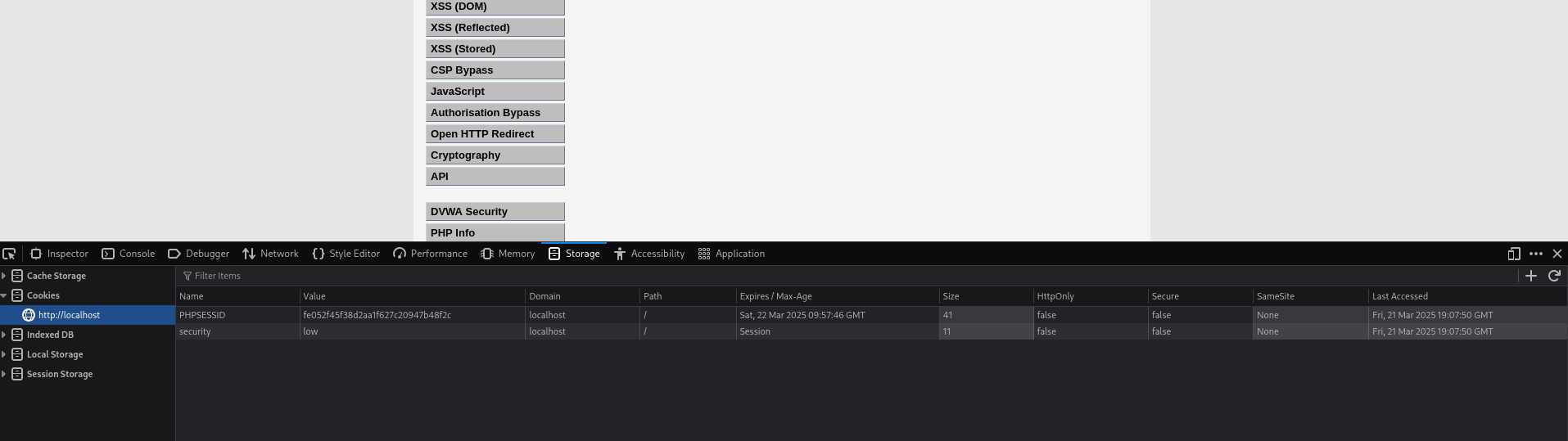
The general hydra command is : 

hydra -L<username-list> -P <password-list> <target> http-get-form "<login-url>:<login-field>^USER^&<password-field>^PASS^:F=<failed-login-string>"

We use -L -P if we have a list of usernames or passwords respectfully.

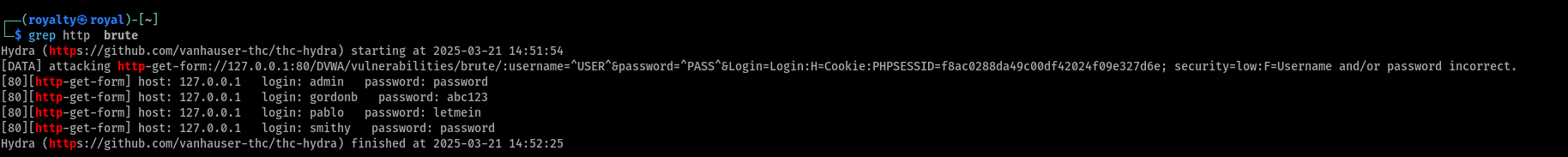
In my case I used the local host which is 127.0.0.1.

We have to use the cookies from the website

The cookies ensure that the command attempts to login in the current session.

After various login attempts formulate the hydra command to include this error message

Now run the hydra command.

I used the >> command so that the output from the hydra command would be stored in a file called brute in my computer

Inorder to get the correct password from the output, i used the grep command.

>The grep command is used to search for lines matching a specified text or pattern and print out the matching lines.

We see that the passwords and usernames are as follows:

| username | pass |
| --- | --- |
| admin | password |
| smithy | password |
| gordonb | abc123 |
| pablo | letmein |

To confirm that the brute force was a success we try logging in as one of the other users

We successfully logged in as “smithy” with the password as “password”

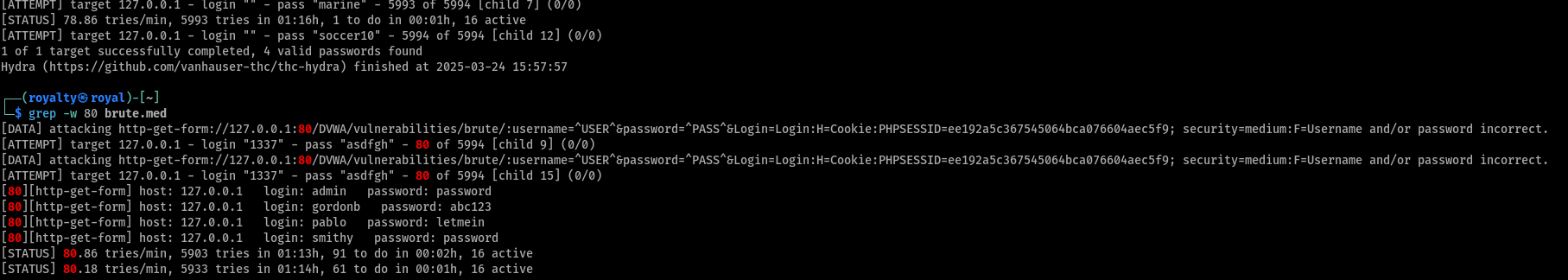
## Using Hydra (MED)

Inorder to use hydra, we have to call it out saying ‘hydra’ then the command

The general command to be used will be ‘ hydra -L “username list” -P “passwd list” local host http-get-form "/DVWA/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:H=Cookie:PHPSESSID=”SESSION ID”; security=medium:F=Username and/or password incorrect." -V -I >> brute.med ‘



I set the command to paste its output in a file called ‘ brute.med’ for easier access.

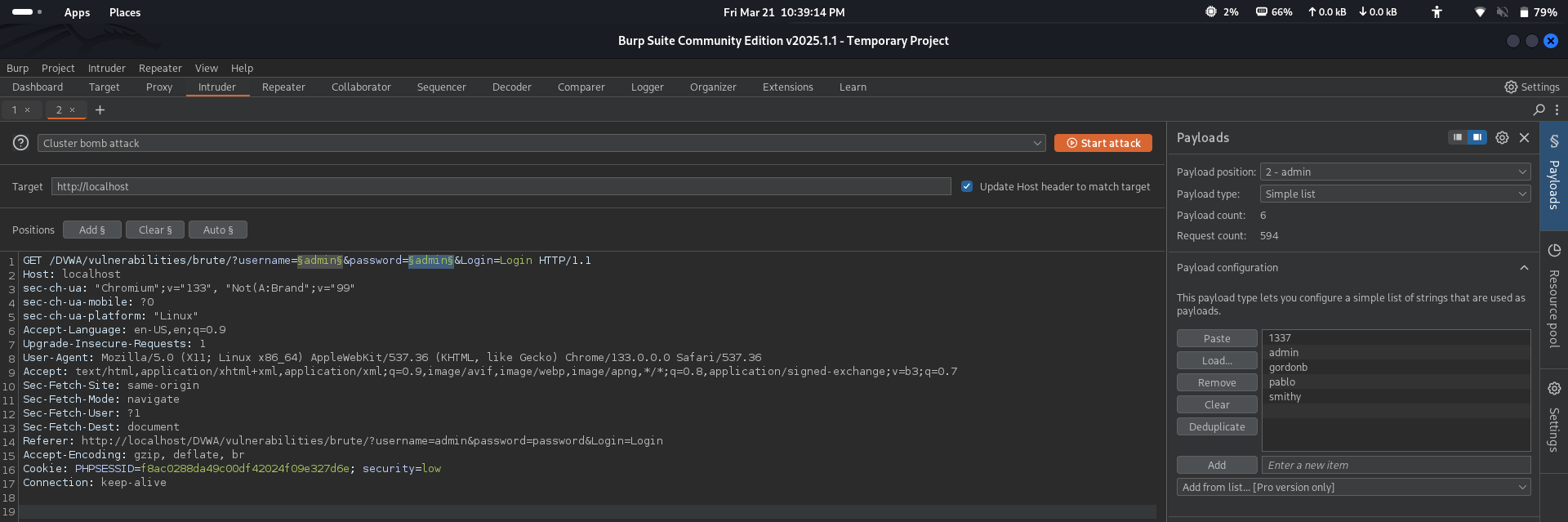


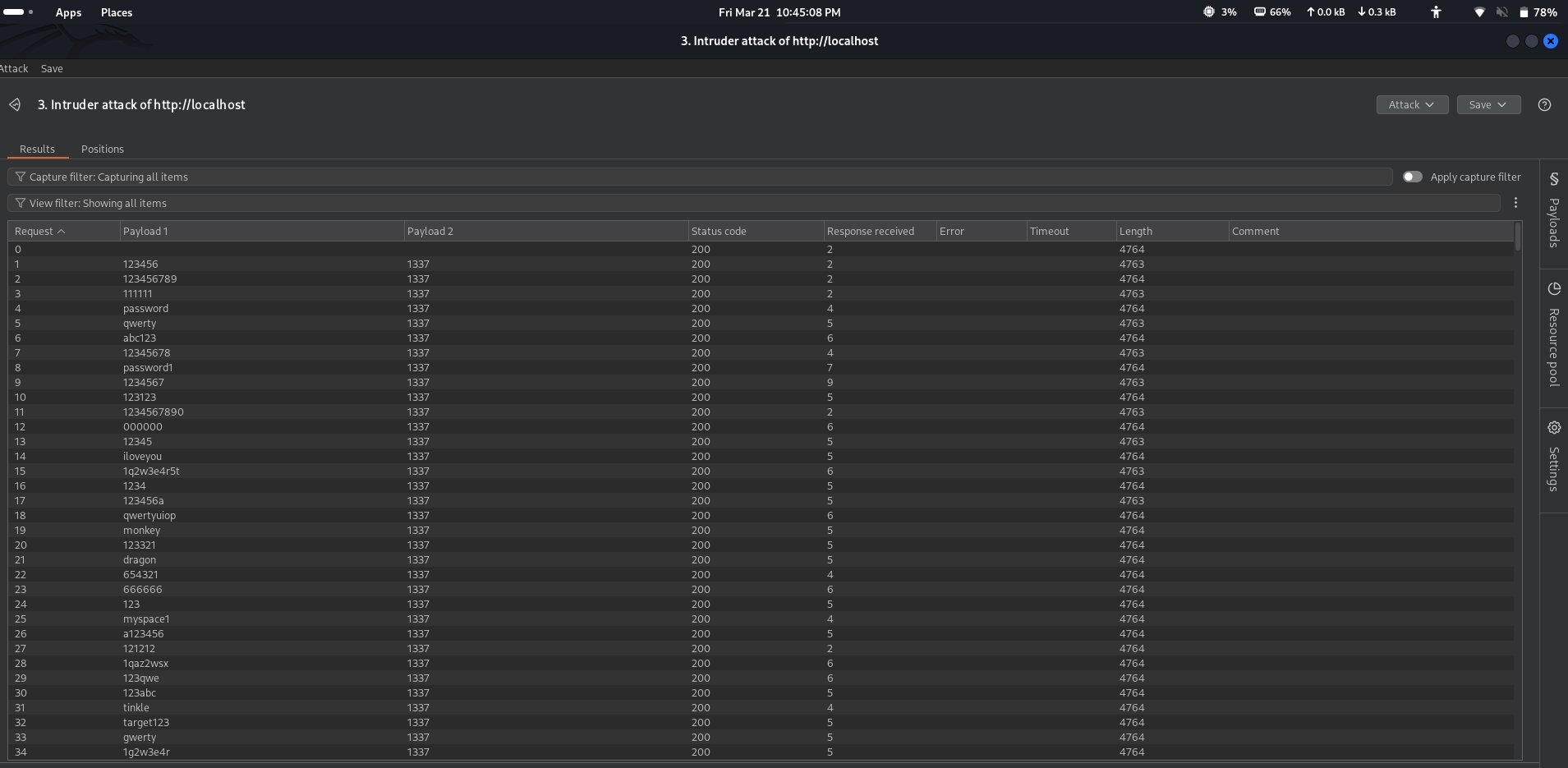
Inorder to get the usernames and their passwords i used the grep command: “ grep -w 80 brute.med “

## 

## using Burpsuite [LOW]

* Burp Suite is also a tool used in brute forcing if you send the GET request to the intruder and set the necessary payloads.
* I didn't use this tool as it is a time consuming method of attacking.
* You will have to set payloads for both the password and username which makes it scroll over 1000 passwords.

Although i didn't use it, one can use cluster comb to run the payloads simultaneously

This could reduce the workload although still time consuming:

### MITIGATION STRATEGIES

### 1. Use Strong Passwords.

Having a strong password policy is the simplest and most effective way of thwarting a brute-force attack. You would want to create a complex password for your web application or a public server that is impossible to guess but is relatively easy to remember. Follow these guidelines when creating a password:

* Don’t use your personal information for your passwords. Avoid using your birthday, name, or email address for your passwords.
* Never recycle passwords for your accounts. Use unique password combinations for each of your online accounts.
* wsCreate a password that’s longer than six characters. Ideally, passwords should be at least 15 characters long.
* Don’t use dictionary words from any language. It’s best to use random character strings rather than words.

### 2. Limit Login Attempts.

### 3. Monitor IP addresses.

### 4. Use Two-Factor Authentication (2FA).

### 5. Use CAPTCHAs.

### 6. Use Unique Login URLs

### 7. Disable Root SSH Logins

### 8. Use Web Application Firewalls (WAFs)

### 

# COMMAND INJECTION

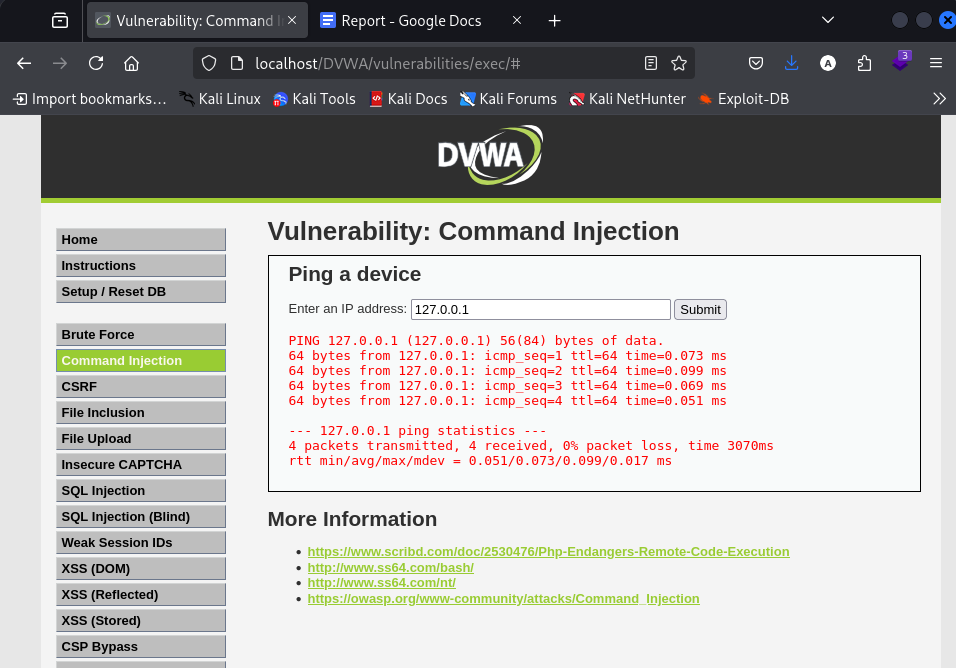
## Low sec

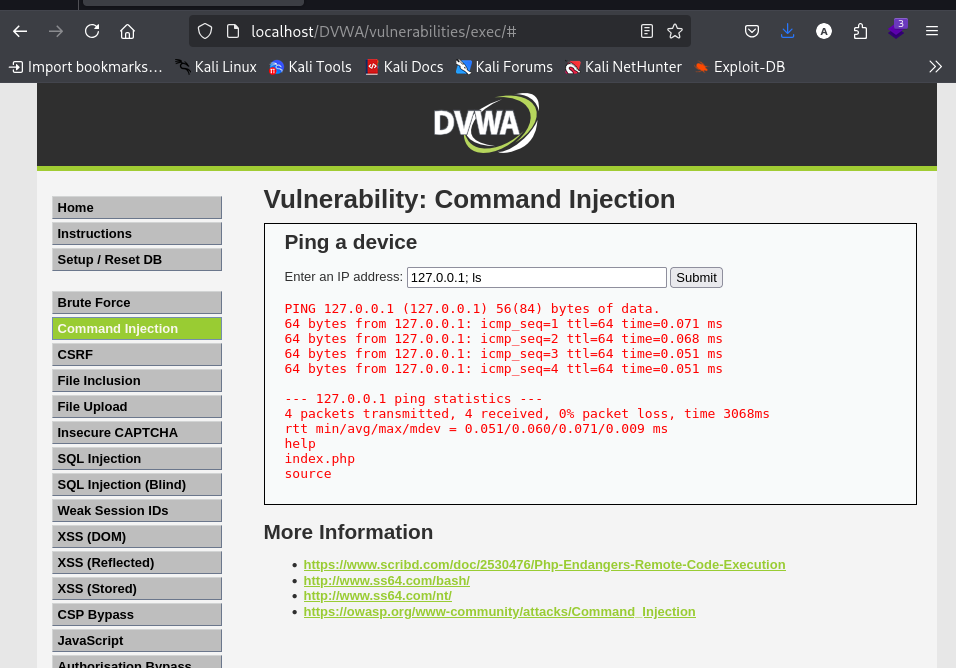
As we know , command injection is when an attacker runs commands remotely on a system.

When a server doesn't handler user input properly then an attacker gets a chance to inject and run commands in the system.

Concept proof:

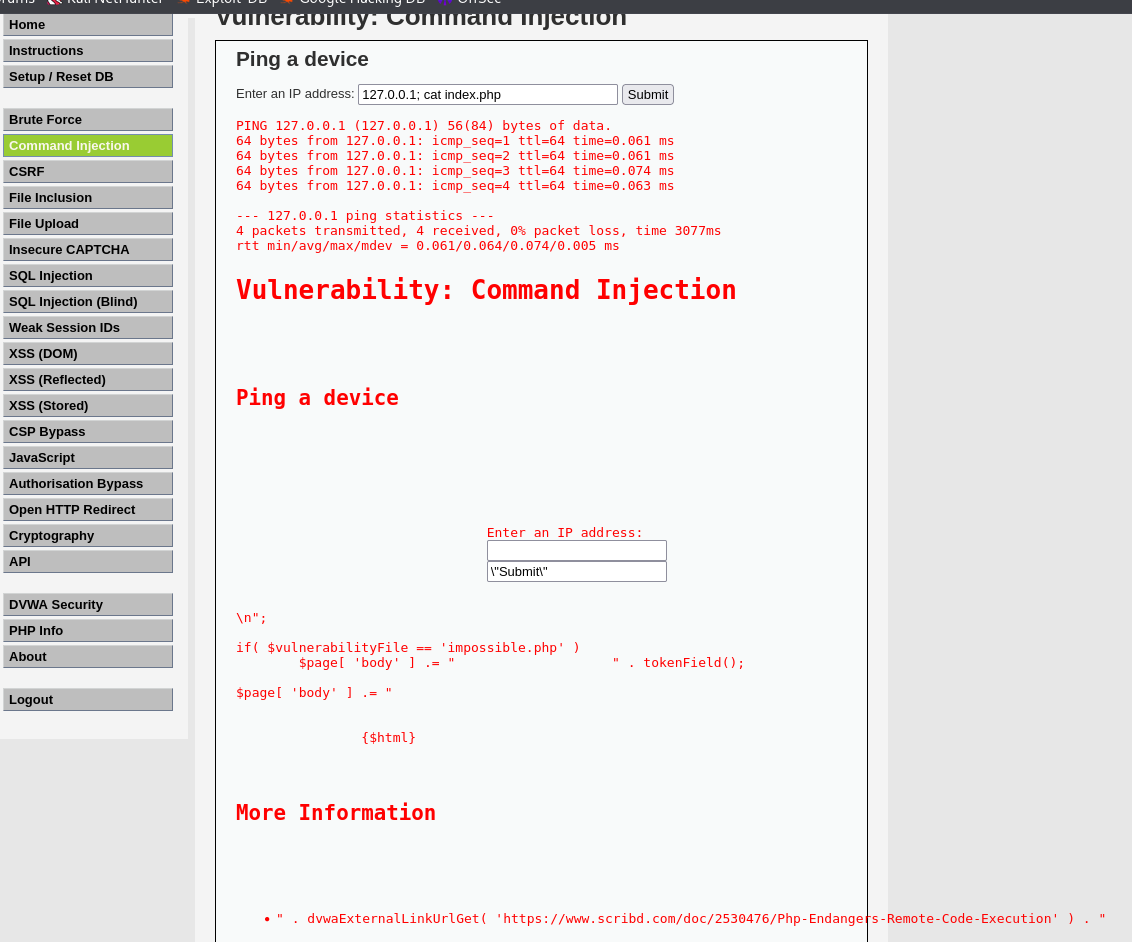
In this case, the site is asking to ping an ip address, we ping our local address and see the results:



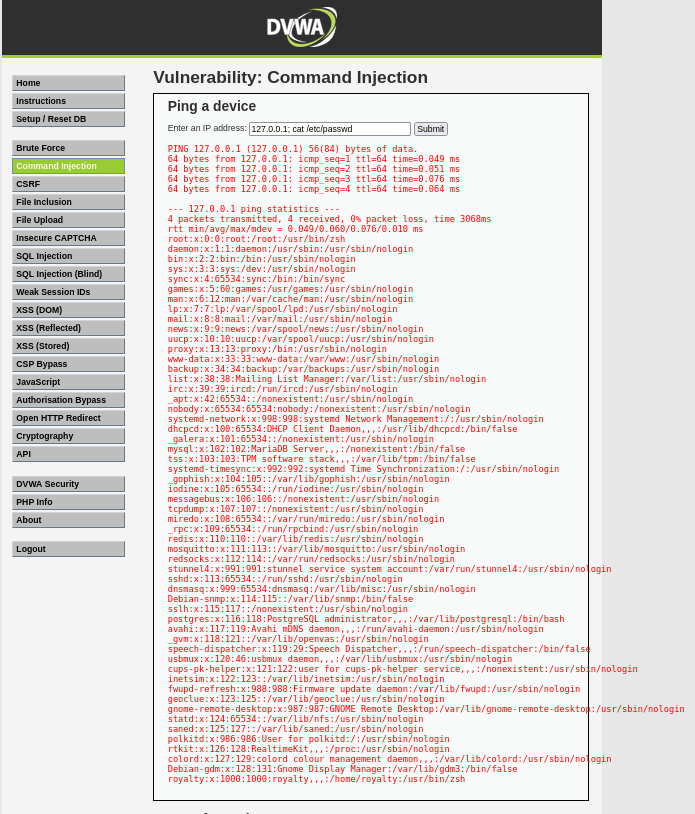
Now we can try stringing commands together with the ip and observe the execution process.

Stringing with the “ls” command lists files found.

Stringing with “cat” command reads out a file.

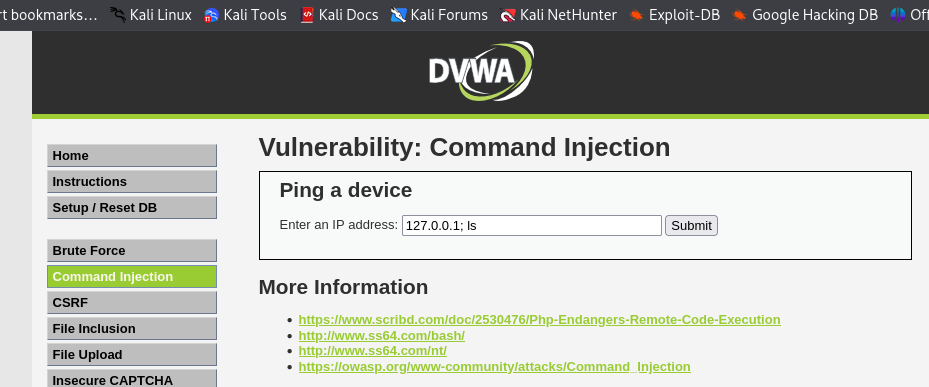


Proving it was possible running different commands after the IP address , I succeeded in reading a file /etc/passwd using 127.0.0.1: cat /etc/passwd payload.

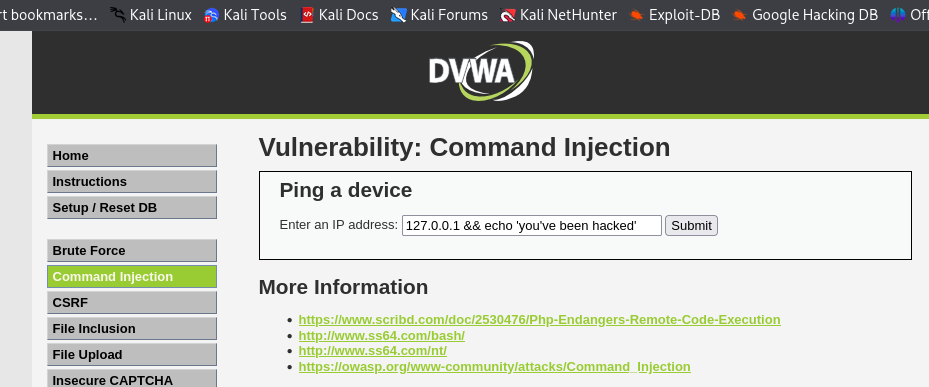


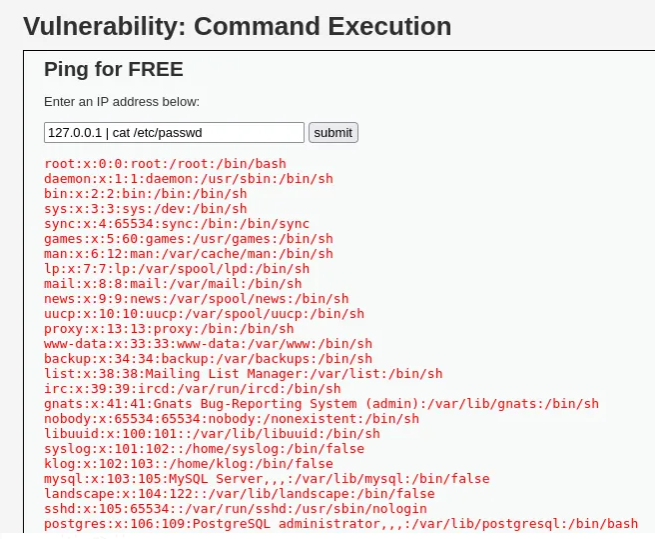
This file shows the application that are running on the server.

## Med sec

In this sec, i tried using the semicolon as low sec but it didnt work.

Other chaining command like “ &&” also didnt return any output

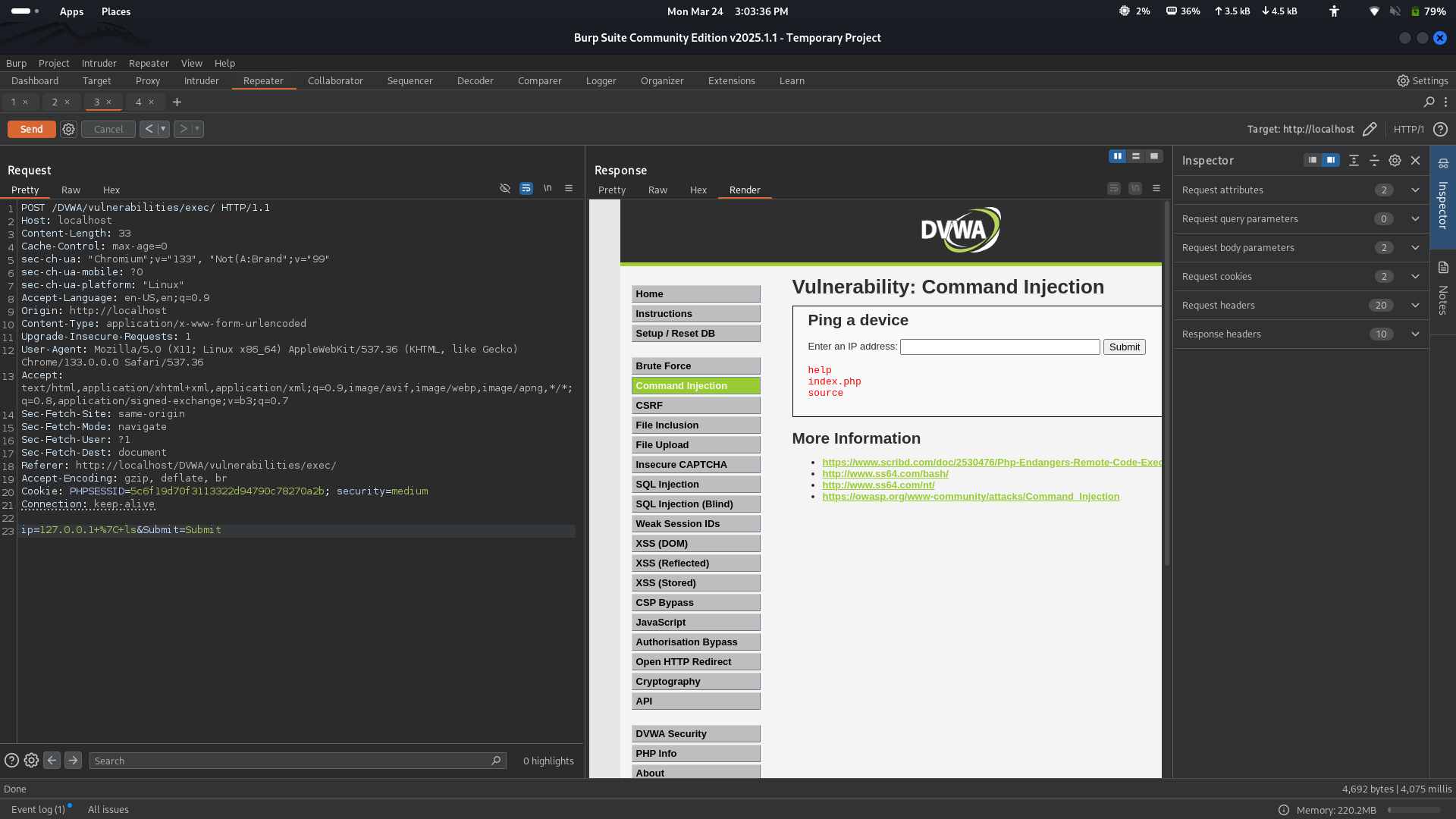


I decided to use the pipe to connect commands

### Using Burp suite

I sent the POST request to the intruder so as to intercept the request sent to the browser.

I tried injecting the ‘ls’ command and the output was as below:



#### MITIGATION STRATEGIES:

**Input Validation and Sanitization**

* Validate and sanitize all user-supplied input before using it in dynamic queries or commands.
* Utilize whitelisting only to allow expected characters or patterns in input fields, rejecting anything else.
* Escape or encode special characters that have a specific meaning in command interpreters (e.g., shell metacharacters).

**Parameterized Queries and Prepared Statements**

* Use parameterized queries or prepared statements when interacting with databases toprevent SQL injection, which can sometimes lead to command injection.
* Parameterized queries bind user input to query parameters, preventing the injection of additional commands.

**Safe APIs and Libraries**

* Employ secure APIs and libraries for executing system commands or interacting with the operating system.
* Avoid using functions like system() or exec() that execute commands with user-controlled input.

**Least Privilege Principle**

* Limit the privileges of the user or service executing system commands to the minimum required for the task.
* Employ dedicated service accounts with restricted permissions rather than executing commands with elevated privileges.

**Enable OS-level Protections**

**Content Security Policy (CSP)**

# SQL INJECTION

SQL Is a standard language for storing, manipulating and retrieving data in databases.

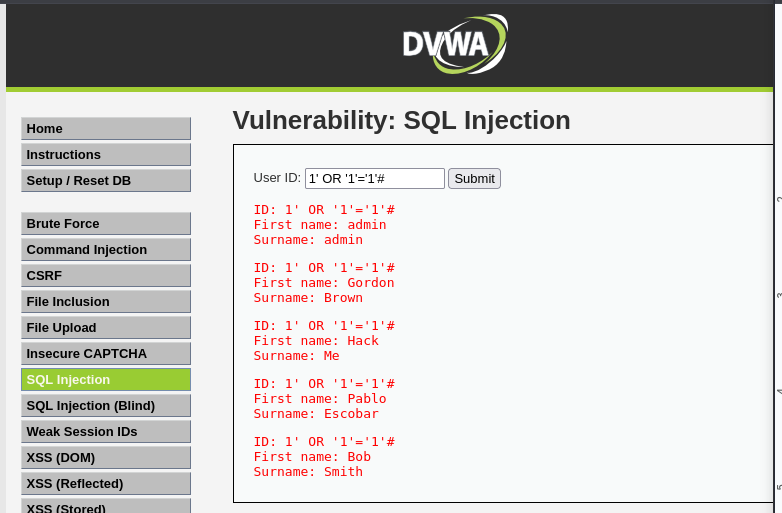
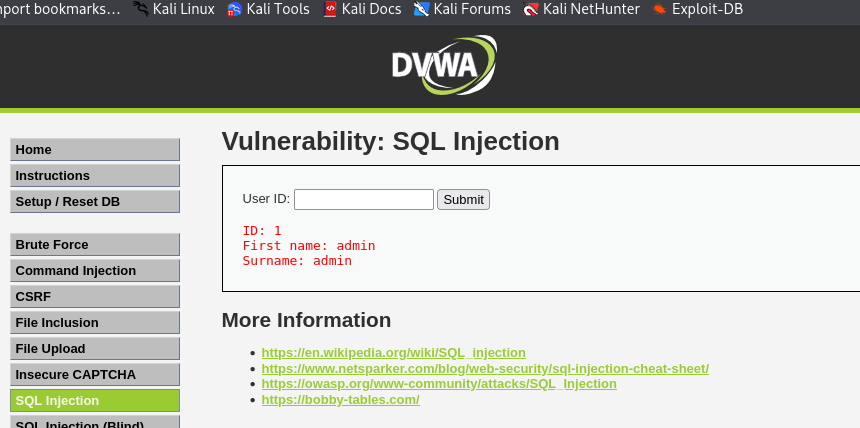
Attackers can also use SQL Injection to add, modify, and delete records in the database

## Low sec

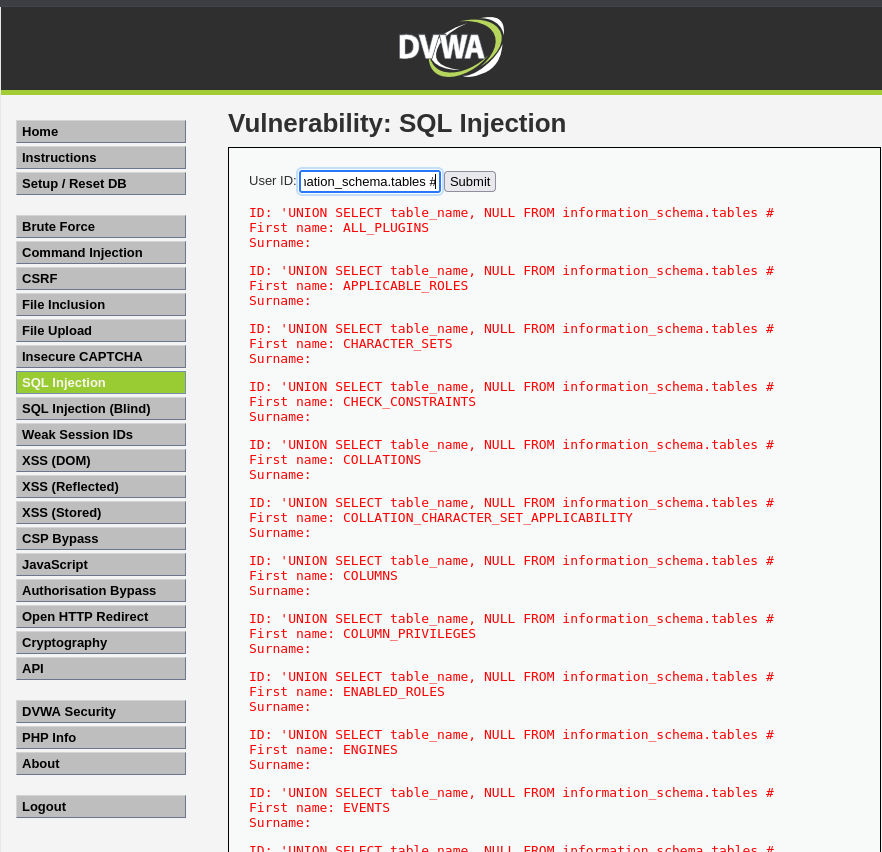
An SQL vulnerability was found in the user ID page where if a user inputs a user id on the page it outputs the user id 1st name and surname.{<http://localhost/DVWA/vulnerabilities/sqli/>}

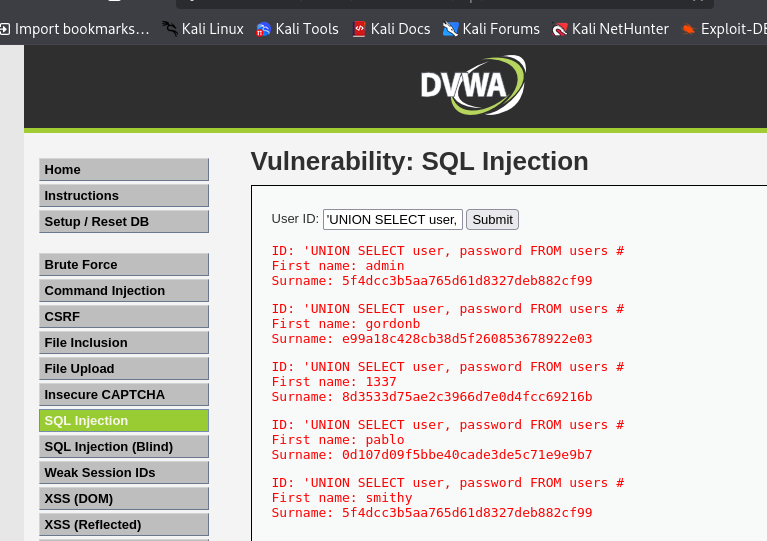
PROOF:

* When you input the user id “1” it returns the values:

I attempted the payload 1’ OR ‘1’=’1’# and the output gave all first and surnames:

This allowed us to skip the need for a specific user ID.

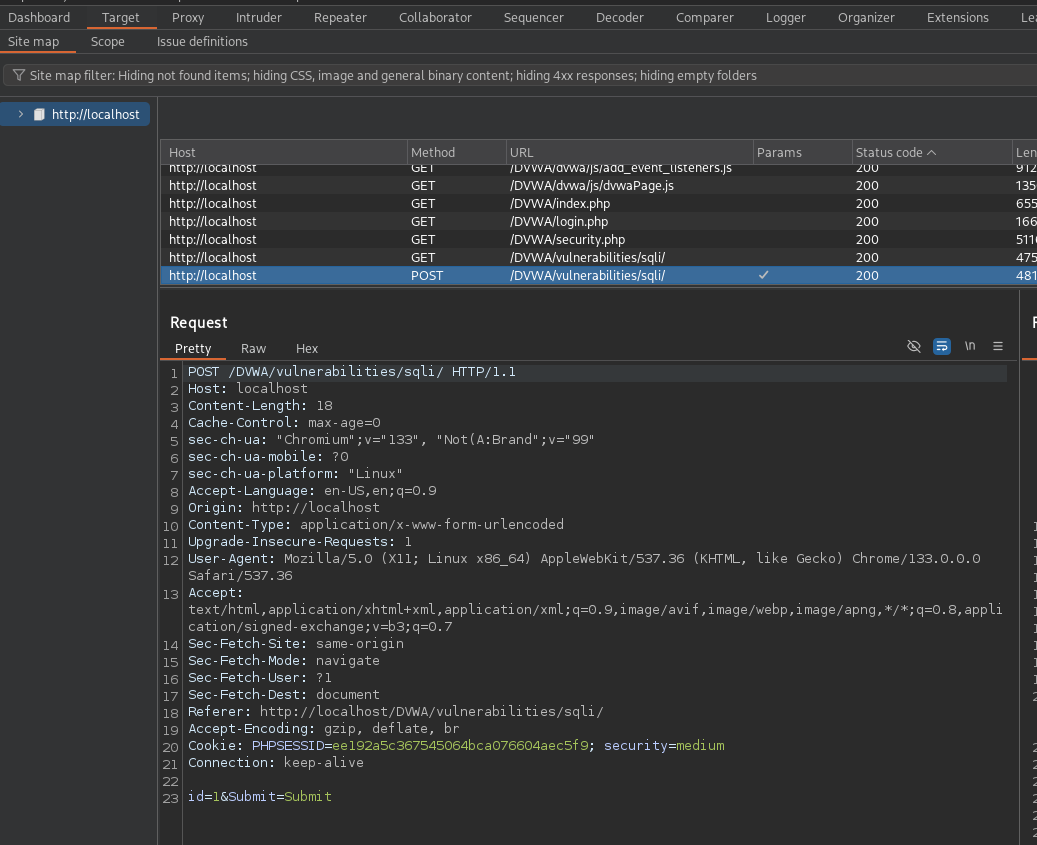
Next i attempted with the union payload:'(UNION SELECT table\_name, NULL FROM information\_schema.tables #)this outputs the tables in the database.

I also attempted to retrieve the usernames with their encrypted passwords:

## Medium sec

Here the ID input has been set to a drop box

We cannot directly input our malicious code hence i used burp suite.

Lets send the POST request to our repeater.

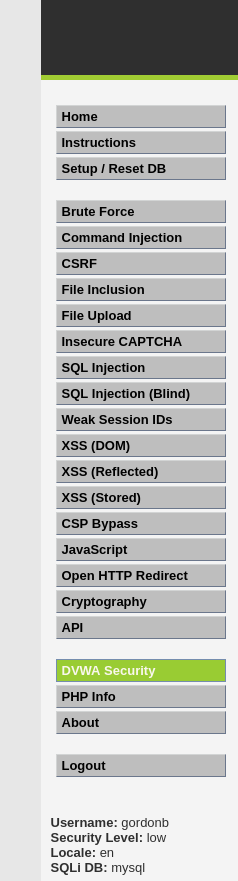
I injected the union payload and it output the users first name and encrypted surnames

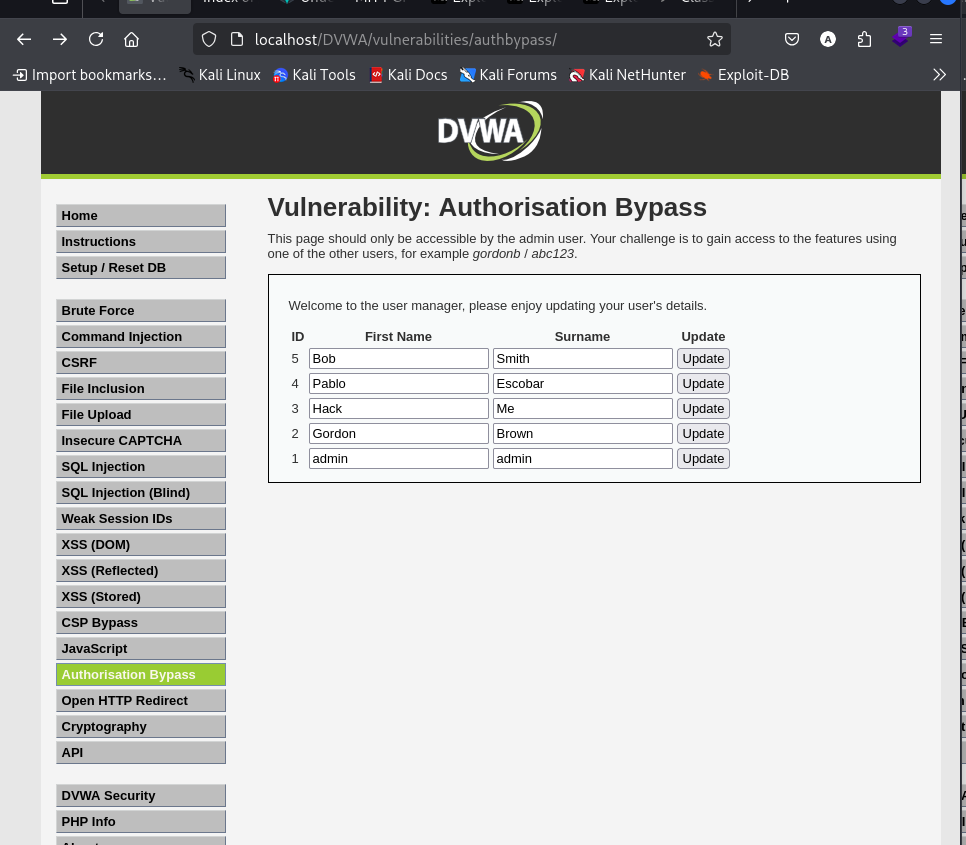
### MITIGATION STRATEGIES

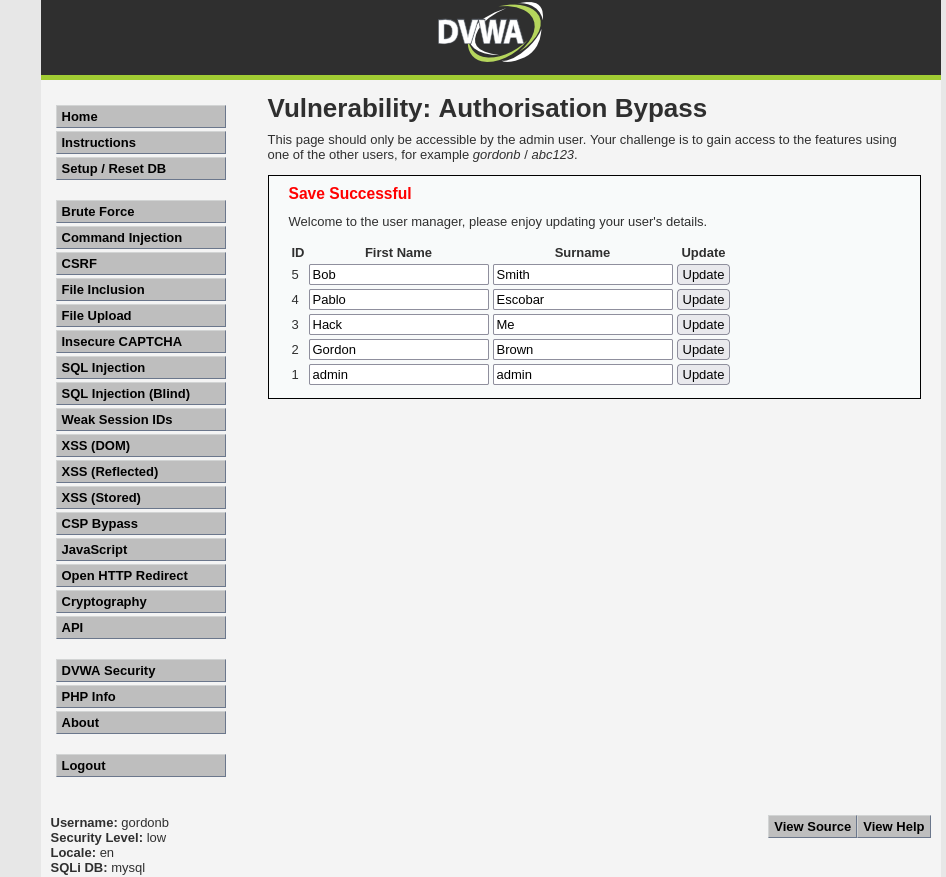
1. **Filter database inputs:** Detect and filter out malicious code from user inputs.
2. **Restrict database code:** Prevent unintended database queries and exploration by limiting database procedures and code.
3. **Restrict database access:** Prevent unauthorized data access, exfiltration, or deletion through access control restrictions.
4. **Maintain applications and databases:** Keep databases fully patched and updated. Upgrade when possible.
5. **Monitor application and database inputs and communications:** Monitor communication to detect and block malicious SQLi attempts.

# AUTHORIZATION BYPASS

## Low sec

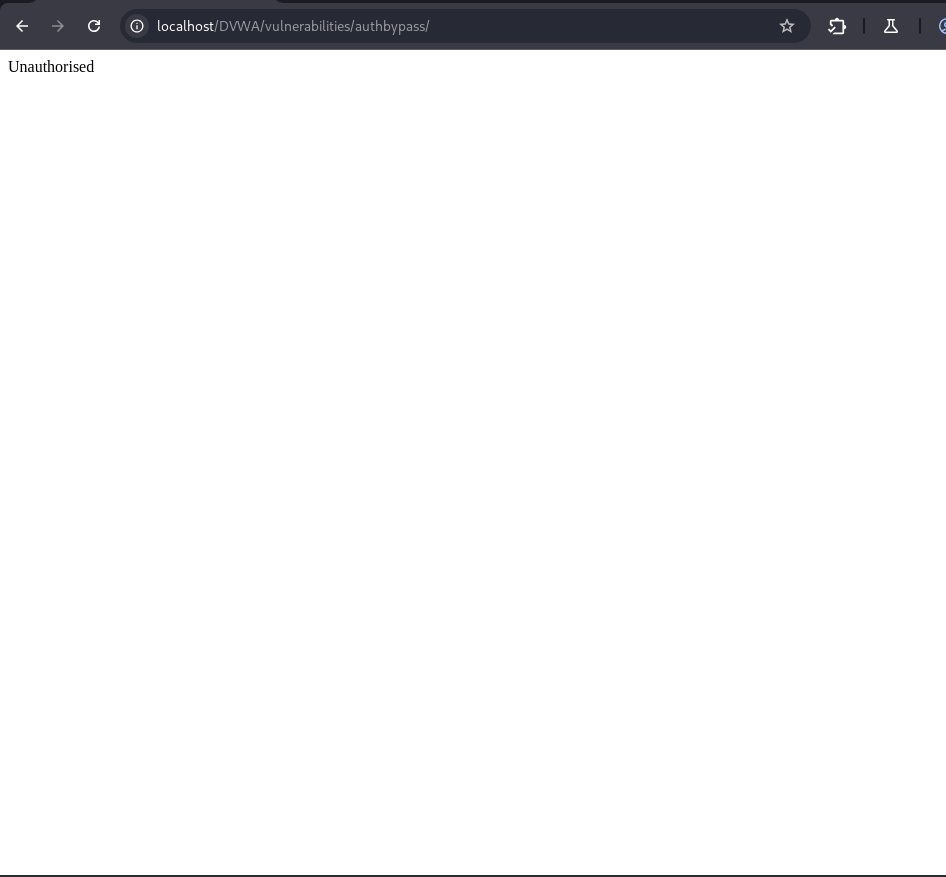
When logged in as gordonb, the authorisation bypass vulnerability module disappears

I logged back in as admin and copied the authorization bypasss url and paste it on the url as gordonb 

Now we are able to access the page as gordonb

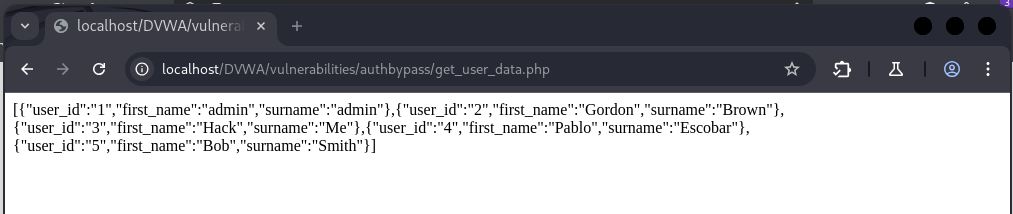
## medium sec

When i copied and pasted the url it gave the output as unauthorised, this is because the security is enhanced



I tried changing the request to a post request but it gave the same output ‘unauthorized.

When i went back to burp suite and copied the ‘get user data php’ the output was as follows:



#### MITIGATION STRATEGIES

**Strong Authentication:** Using MFA, strong passwords, and secure session management.

**Input Validation:** Sanitizing user inputs to prevent injection attacks.

**Access Control:** Implementing RBAC and the principle of least privilege.

**Secure Coding:** Following secure coding practices and using security frameworks.

**Defense-in-Depth:** Employing layered security and EDR tools.

**MFA Best Practices:** Avoiding SMS-based MFA and monitoring for suspicious activity.

**User Education:** Training users to recognize security threats.

**Regular Testing:** Conducting vulnerability assessments and penetration testing.

**Incident Response:** Developing a clear incident response plan.