ESCUELA COLOMBIANA DE INGENIERIA
JULIO GARAVITO

IT SECURITY AND PRIVACY
GROUP 1L

LABORATORY 11

SUBMITTED BY:
JUAN PABLO FERNANDEZ GONZALES
MARIA VALENTINA TORRES MONSALVE

SUBMITTED TO:
Eng. DANIEL ESTEBAN VELA LOPEZ

BOGOTÁ D.C.
DATE:
07/04/2025

Introduction

The rapid advancement of information technologies has increased the dependence of organizations and users on web applications to perform everyday tasks. However, this growth has also brought with it an increase in associated security risks. In this context, it is essential that future professionals in areas such as systems engineering and cybersecurity understand and master the concepts related to vulnerabilities in web applications.

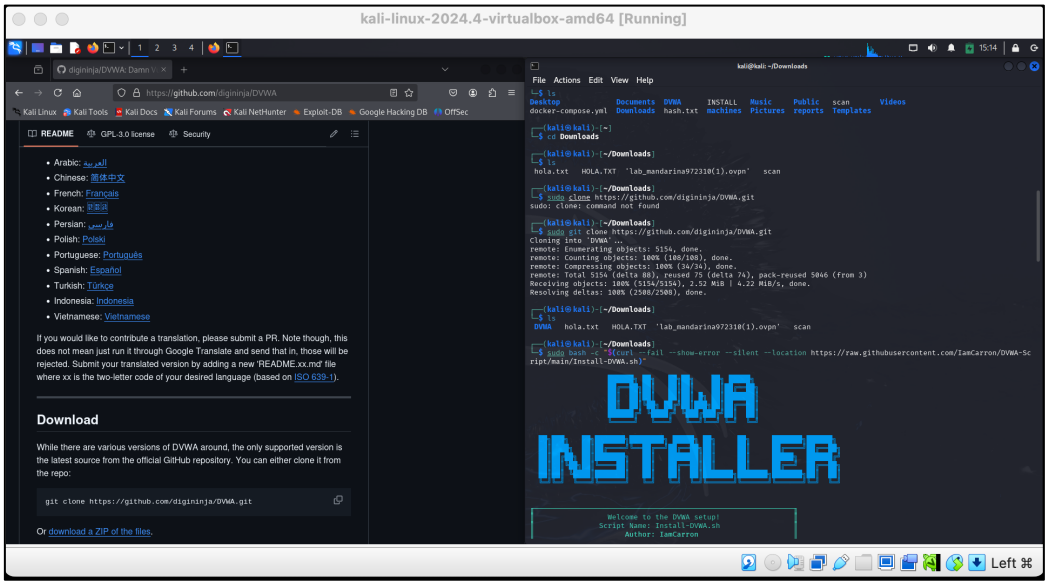
Damn Vulnerable Web Application (DVWA) is an education-based tool that allows you to explore, analyze, and experiment with various vulnerabilities in a safe and controlled manner. As an intentionally vulnerable application, DVWA provides an ideal environment for hands-on learning of security assessment techniques, such as SQL injections, cross-site scripting (XSS), cross-site request forgery (CSRF), and more.

The purpose of this paper is to present a practical study of the most common vulnerabilities affecting modern web applications, using DVWA as an analysis platform. The experience gained will contribute to the development of key competencies in the area of computer security and to the strengthening of a critical stance against the risks present in the development and deployment of web software.

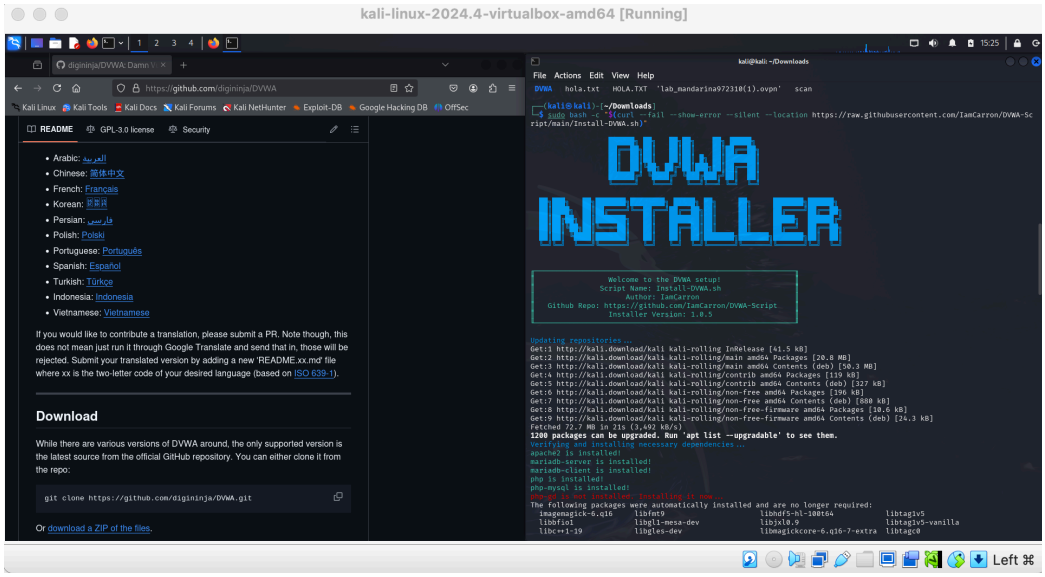
DVWA Installation

The first thing we will do is clone the github repository from the following link ***https://github.com/digininja/DVWA.git*** and follow the instructions found in the Re-adme.md

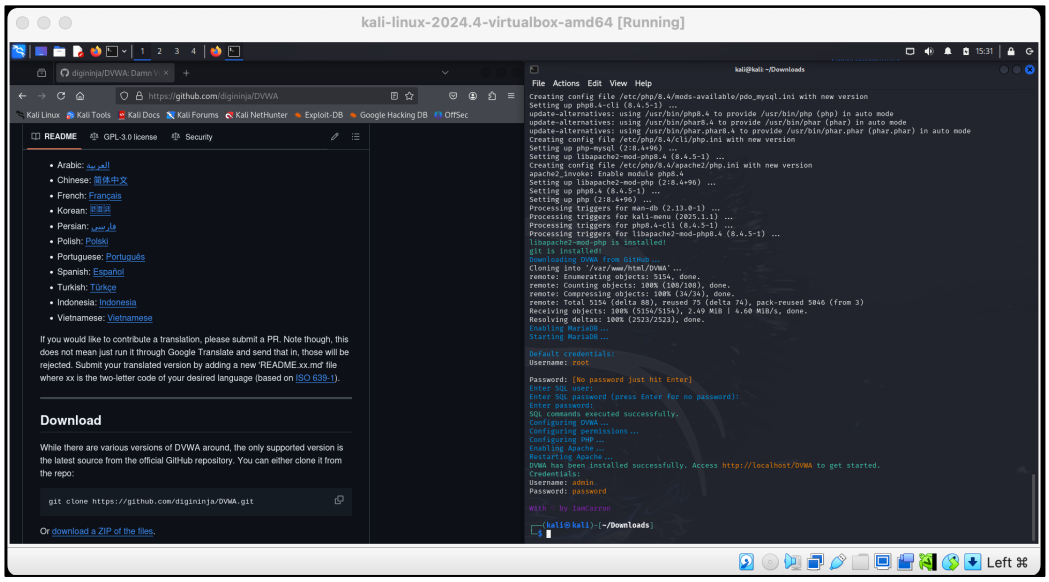
- Clone the repository with the ***git clone command*** ***https://github.com/digininja/DVWA.git***



- We will use the ***command sudo bash -c "$(curl --fail --show-error --silent --location https://raw.githubusercontent.com/IamCarron/DVWA-Script/main/Install-DVWA.sh)"*** , with this instruction *curl --fail --show-error --silent --location <URL>* we will be downloading the content of the script Install-DVWA.sh from GitHub in a silent way and with correct error handling. The other part of the *bash -c command "$(...)"* it runs the content being downloaded as a Bash.

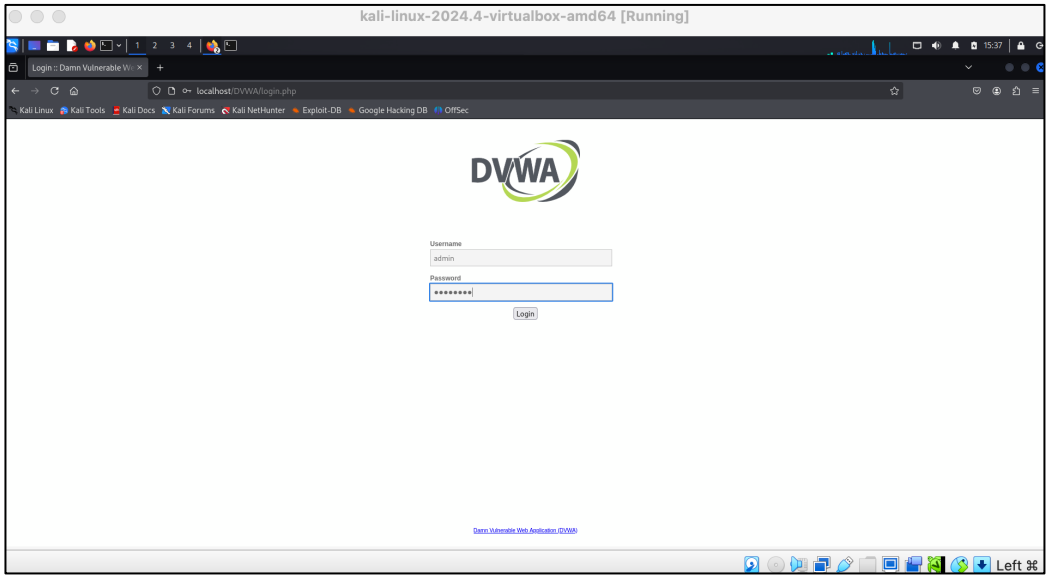


During the execution of the command we are asked to enter the connection credentials for MySQL, for this we will hit ***ENTER*** accepting the default values.



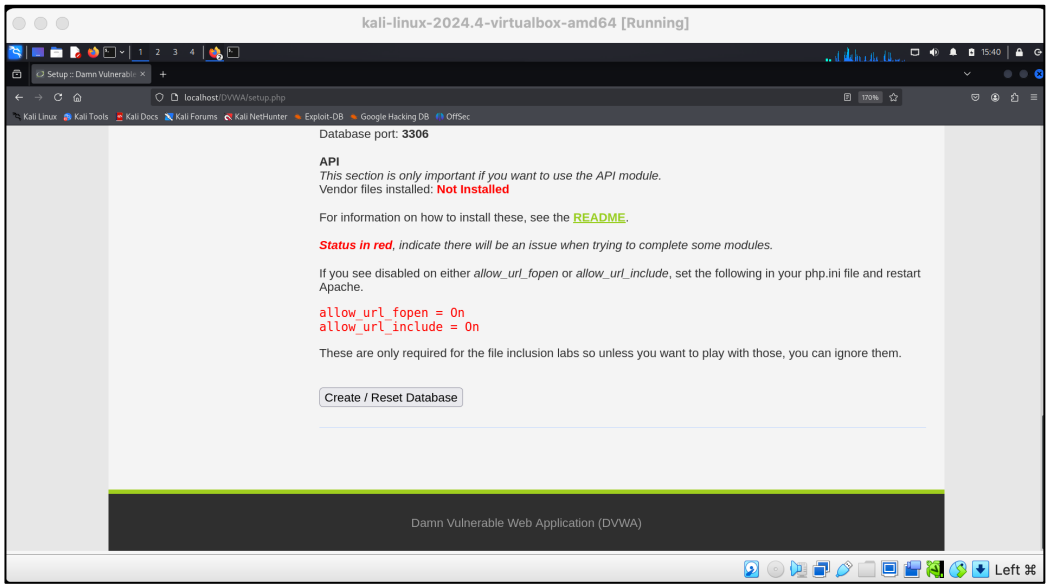
At the end they give us the connection credentials (user: admin and Password: password) to access the login that is in *http://localhost/DVWA*

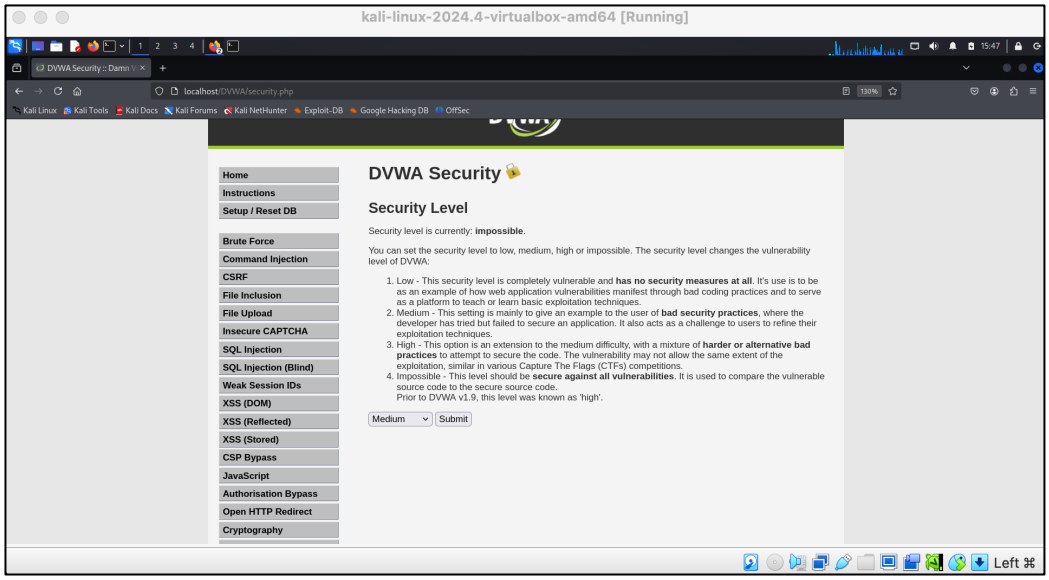
- We will access the application by running on the machine and entering the credentials.



We will press ENTER on the ***Create/Reset Database*** button to initialize the application's database, go to the application's security option and set the difficulty

level to ***Medium***.



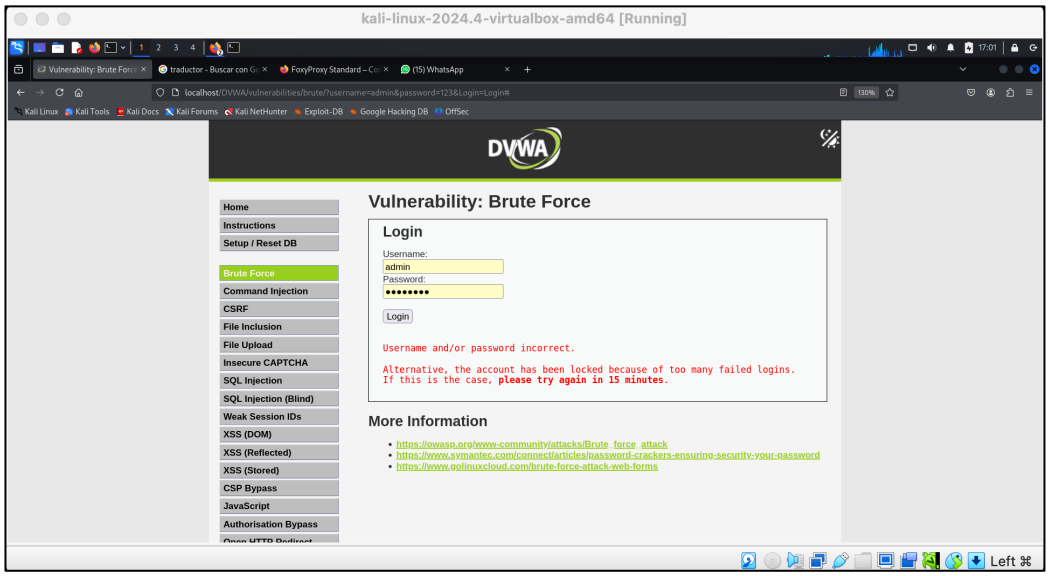


Solving the levels

- ***Brute Force***

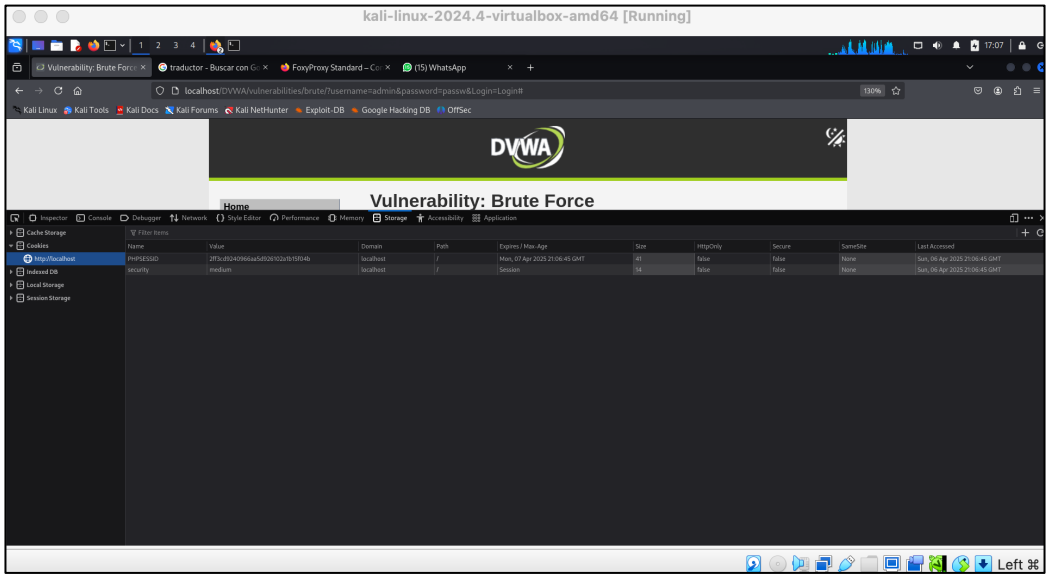
  A brute force attack systematically tests multiple password combinations until it finds the correct one. In DVWA, this vulnerability allows an attacker to force attacks on the login interface, as there are no protections against login attempts, allowing the attacker to make numerous attempts without any restrictions.

Initially we do not have the user's password and we will make any attempt to analyze the structure of the requests that are made when the login is made.

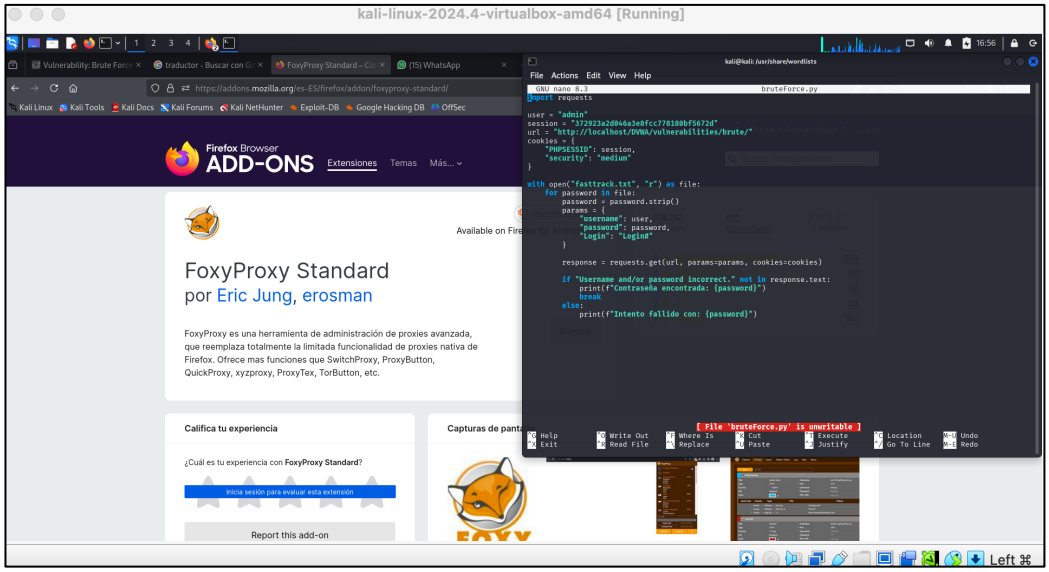


The URL structure will be ***http://localhost/DVWA/vulnerabilities/brute/?username=admin&password=123&Login=Login#*** access the DVWA (Damn Vulnerable Web Application) brute-force module that runs locally on the machine. It uses the GET method to send the parameters username=admin, password=123, and Login=Login, simulating an attempt to log in with those credentials. The final snippet # has no effect on the request, but can be used to redirect to a specific section within the page if necessary.

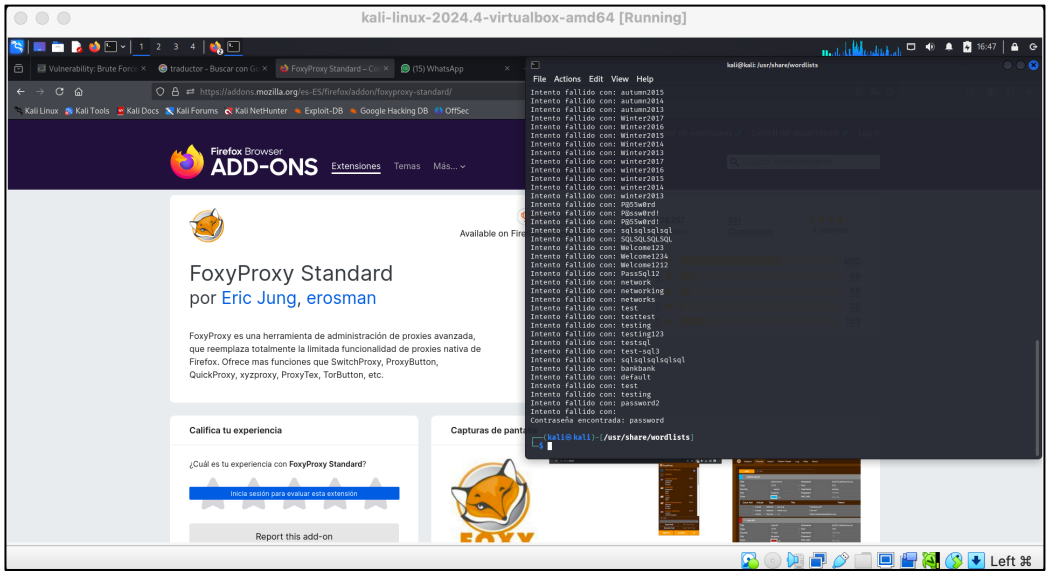
We will also review the cookies to see what information is being saved to make requests, two active cookies are observed for the http://localhost site: PHPSESSID, which stores the user's PHP session identifier and allows them to keep their session active, and security, which indicates the security level configured in DVWA, in this case "medium".

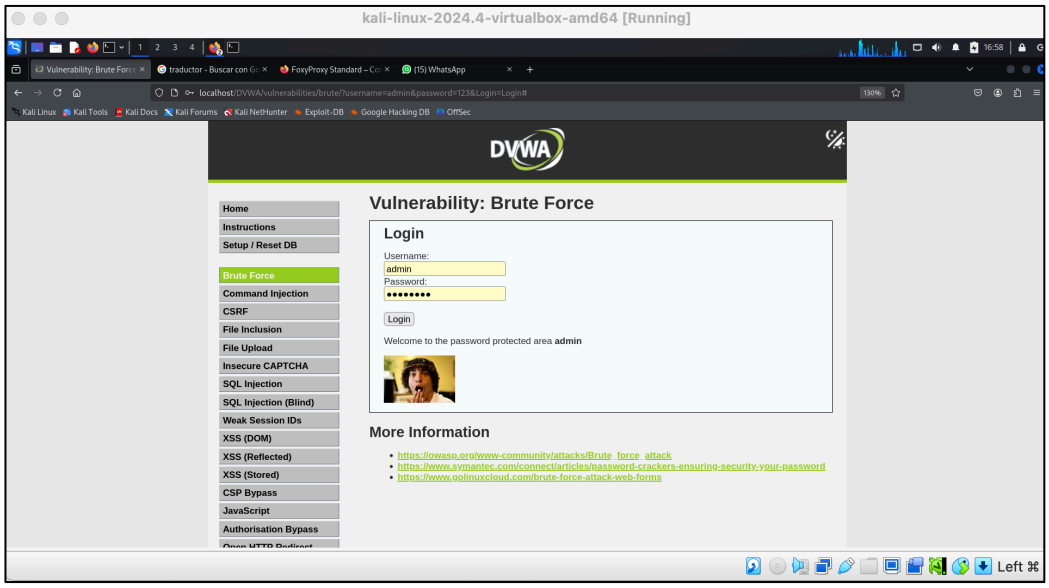


For this level we will be implementing a Python script that uses the ***requests*** library to perform a brute force attack against the vulnerable DVWA login module. We will define the target user (admin), a valid PHP session (PHPSESSID) and operates in low security mode (security=medium). Additionally, a list of possible passwords is loaded from the fasttrack.txt file and, for each one, it sends a GET request with login data and cookies to the server.

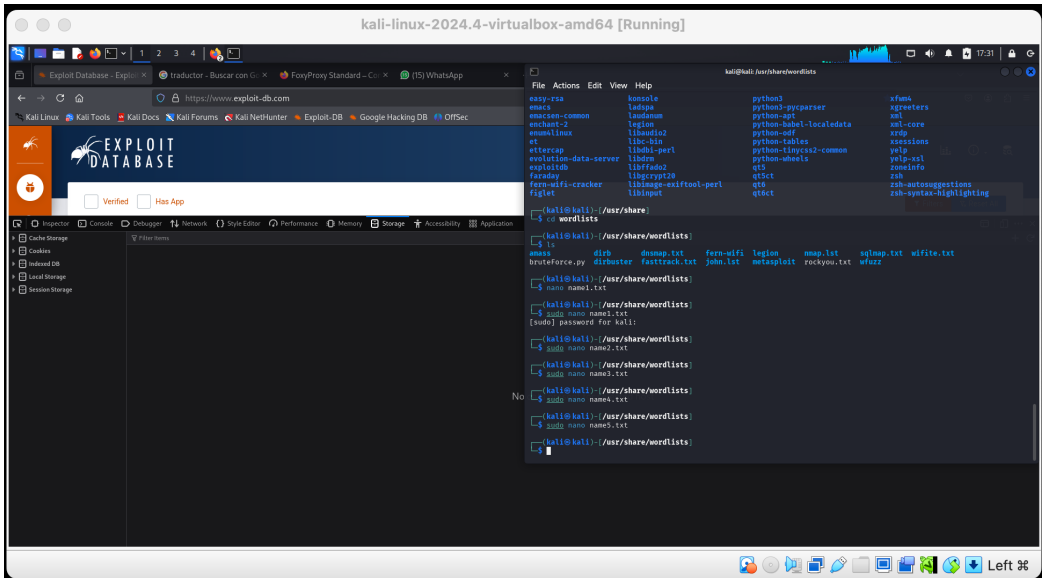


We run the Python file that we created earlier with the ***python3 command nombre_script.py,*** then after a while making several requests we will find that the password of the ***admin user*** and his password according to the script is ***password***

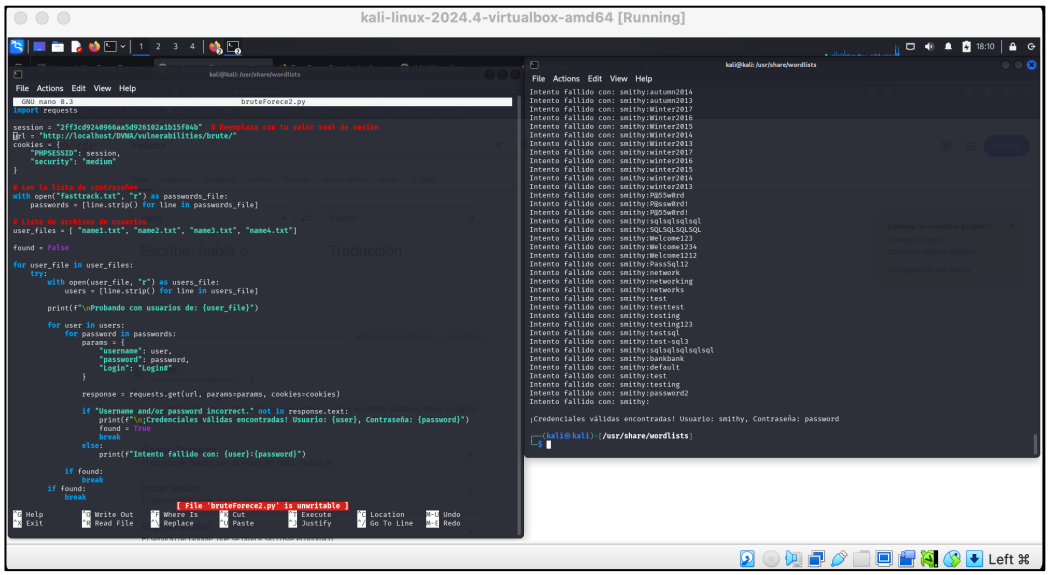




To find more usernames we will modify the code to analyze different names that we create with the most common names used and with a list of passwords.



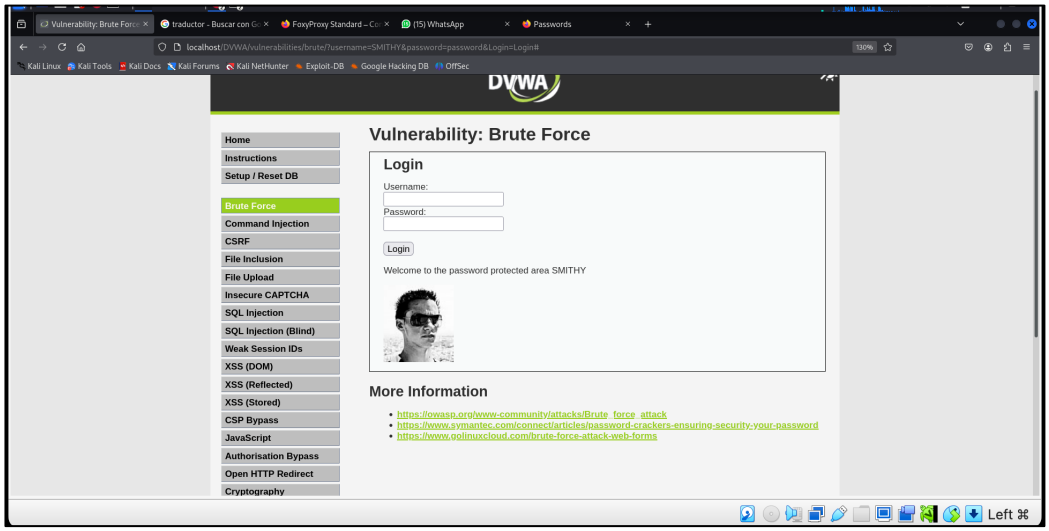
We will be running the script until we find the usernames that can be connected when we make the request and we remove it from the list of where that username was found, with this we will run it again and omit that name.



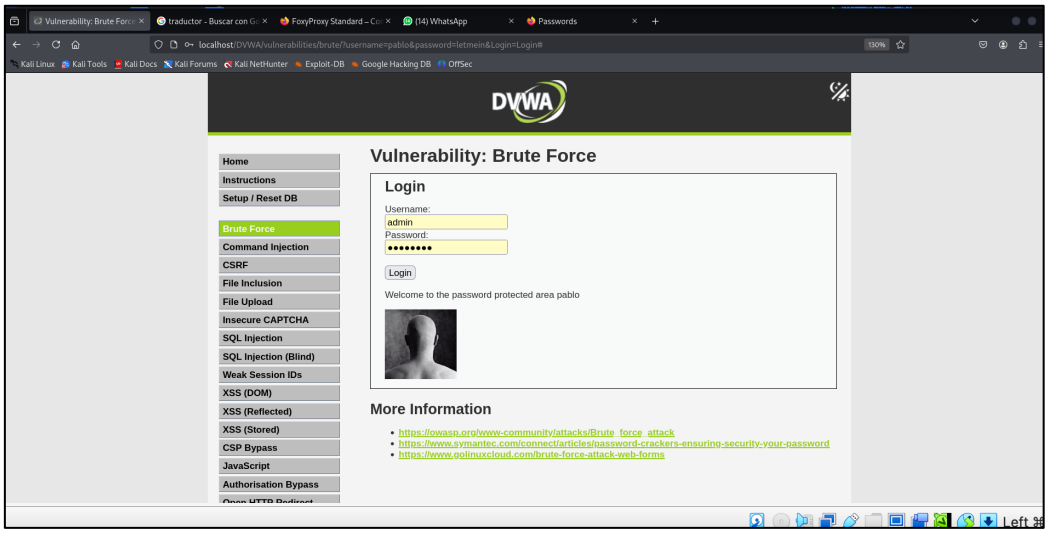
```python
import request

session = "mvalcbr6s9qavdsigu4ts8cr75"
url = "http://localhost/DVWA/vulnerabilities/brute/"
cookies = {
    "PHPSESSID": session,
    "security": "low"
}

# Lee la lista de contraseñas
with open("fasttrack.txt", "r") as passwords_file:
    passwords = [line.strip() for line in passwords_file]

# Lista de archivos de usuarios
user_files = ["users.txt", "name1.txt", "name2.txt", "name3.txt", "name4.txt"]

found = False

for user_file in user_files:
    try:
        with open(user_file, "r") as users_file:
            users = [line.strip() for line in users_file]

        print(f"\nProbando con usuarios de: {user_file}")

        for user in users:
            for password in passwords:
                params = {
                    "username": user,
                    "password": password,
                    "Login": "Login"
                }

                response = requests.get(url, params=params, cookies=cookies)

                if "Username and/or password incorrect." not in response.text:
                    print(f"\n¡Credenciales válidas encontradas! Usuario: {user}, Contraseña:
                        {password}")
                    found = True
                    break
                else:
                    print(f"Intento fallido con: {user}:{password}")

            if found:
                break
        if found:
            break
    except FileNotFoundError:
        print(f"Archivo no encontrado: {user_file}")

if not found:
    print("\nNo se encontraron credenciales válidas.")
```

The results obtained were:

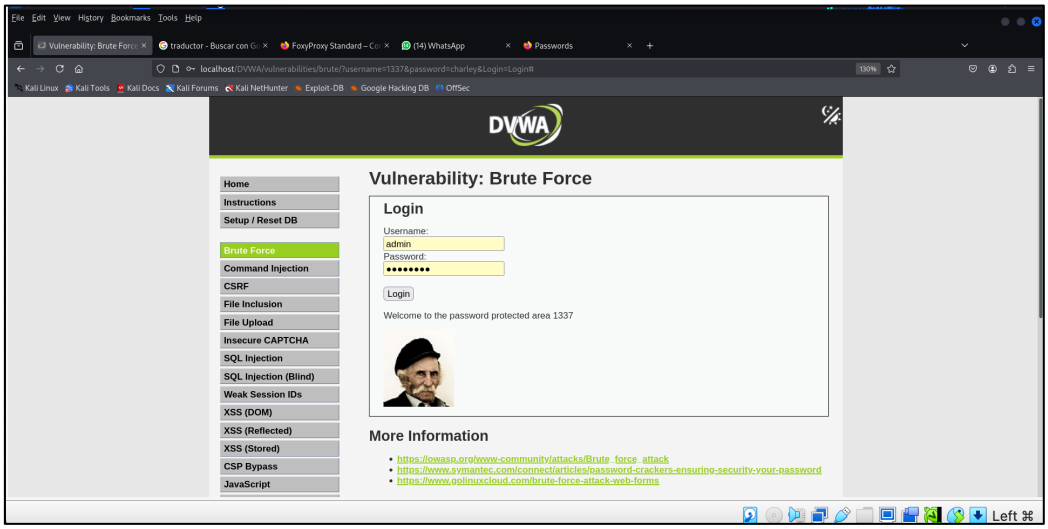
- User: smithy, Password: password



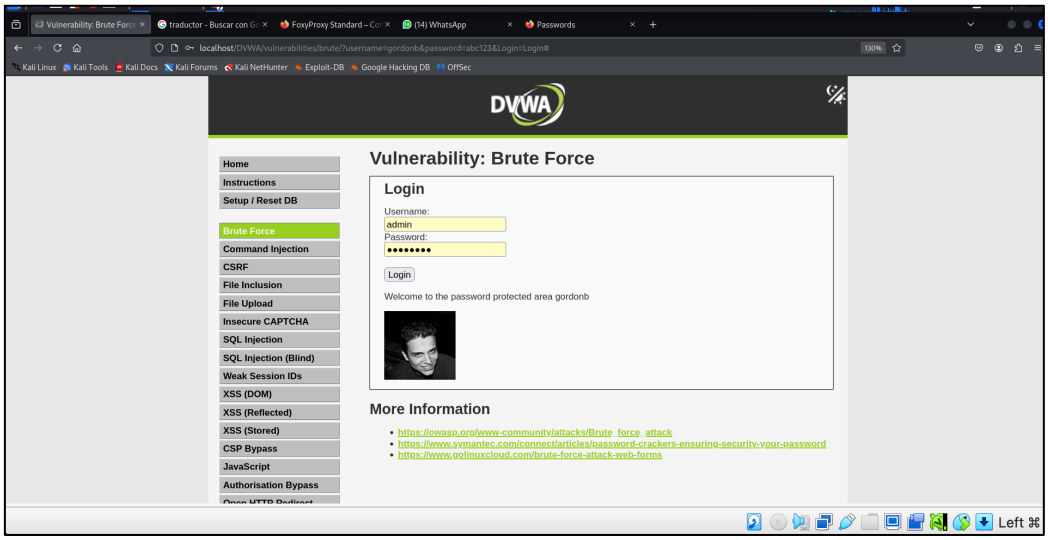
- User: pablo, Password: letmein



- User: 1337, Password: charley



- User: gordonb, Password: abc123

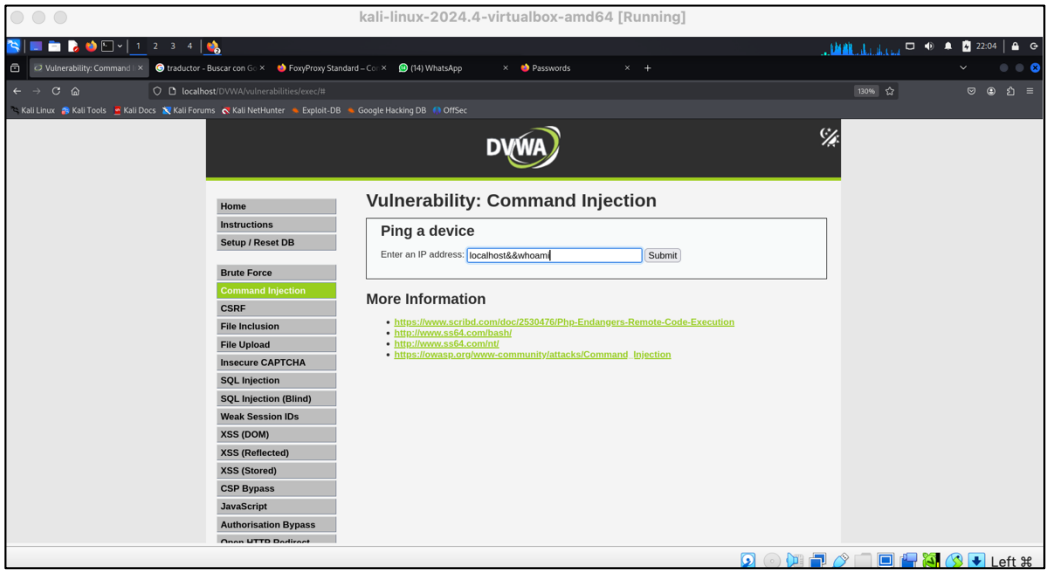


Command Injection

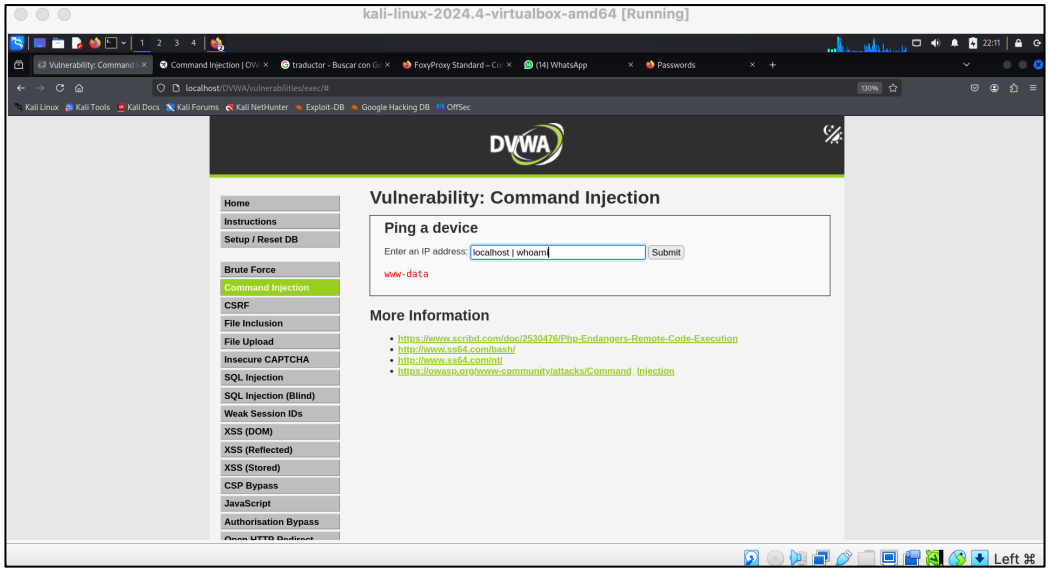
Command injection allows attackers to execute operating system commands from within the application due to improper input validation. This vulnerability allows an attacker to execute arbitrary commands on the system where the application resides.

Our goal will be to find sensitive information such as the name of the user who ran the application or the name of the machine itself.

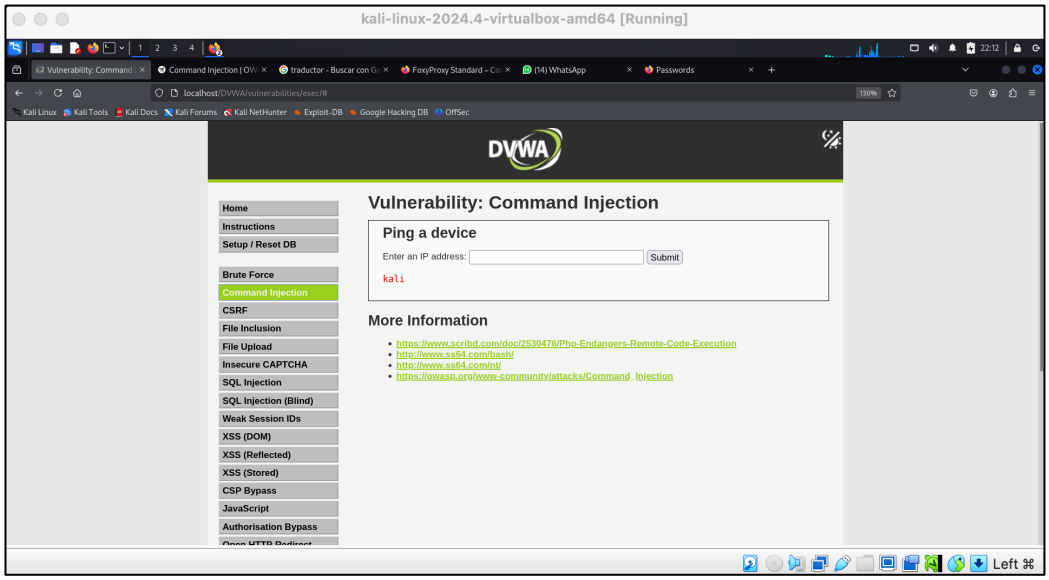
We'll try the usual Linux & command concatenation where localhost(127.0.0.1) and the whoami command (user I run)



We couldn't get any information since they apparently have the && concatenation blocked, we'll try the option to concatenate commands with |.



With this we already got the name of one of the users, for the hostname we will use the same approach in the commands but now ***hostname.***

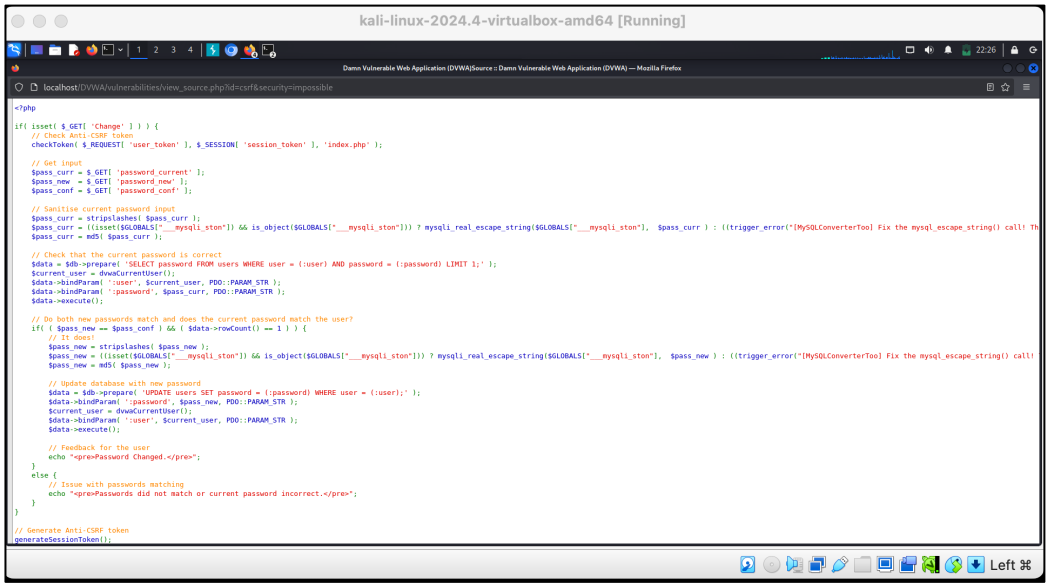


This vulnerability allows direct execution of commands on the underlying operating system, representing a significant risk if an attacker executes malicious commands that compromise server integrity.

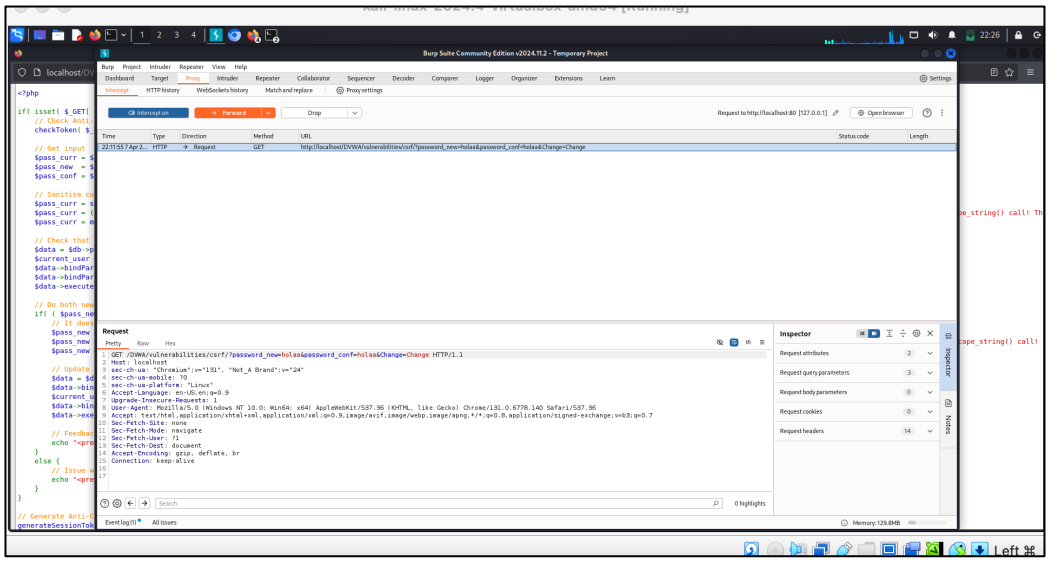
- ***CSRF***

CSRF forces an authenticated user to perform unwanted actions on a web application in which they are currently authenticated, exploiting the application's trust in the user's browser. With social engineering techniques, an attacker can redirect the user to a malicious link that executes actions on their behalf without their consent.

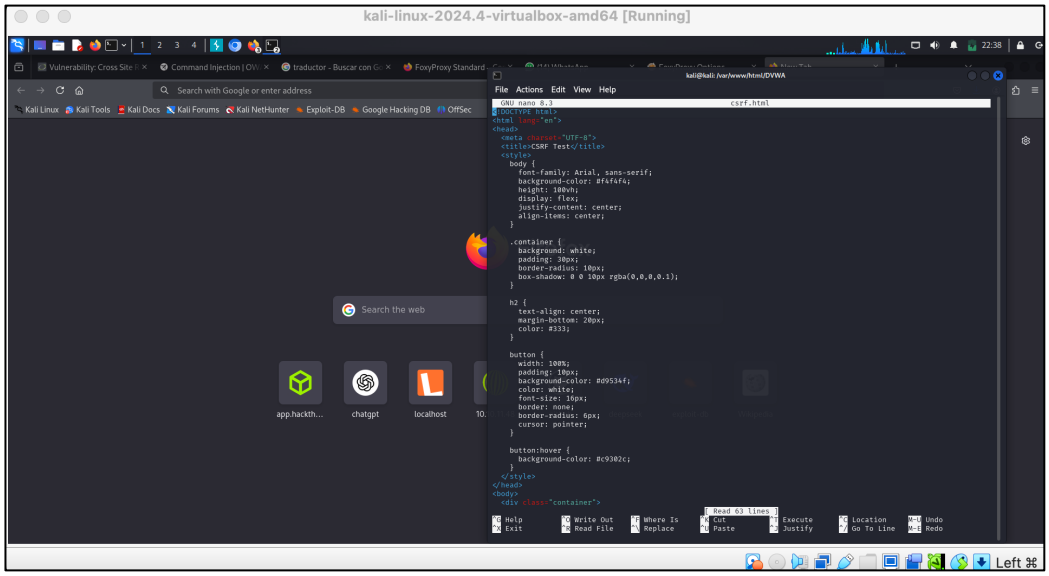
Analyzing the Source Code We can evidence that a verification of the existence of the server name exists in the URL is carried out. If you do, the password change process continues. This verification ensures that the request is coming from a page within the same domain, helping to prevent requests from external or malicious sites.

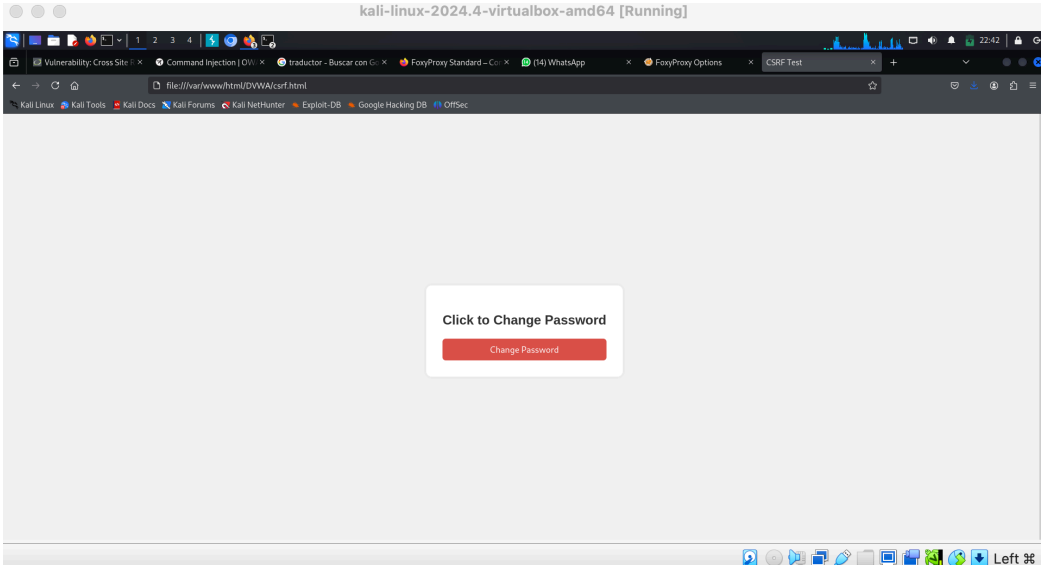


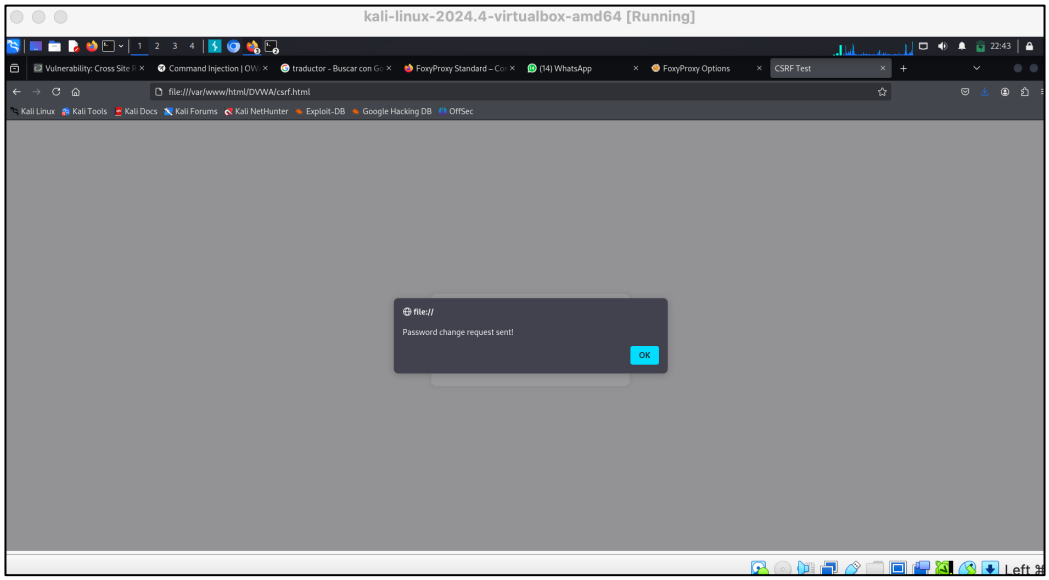
From burpSuite we can see the header of the request that is made, with this information we will have to create a page that seems to belong to a trusted place.

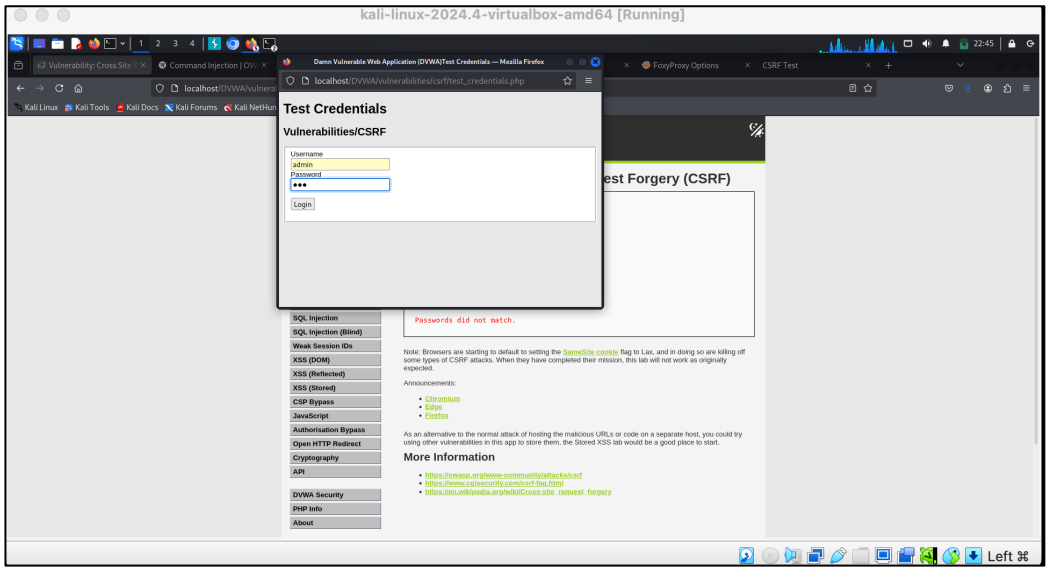


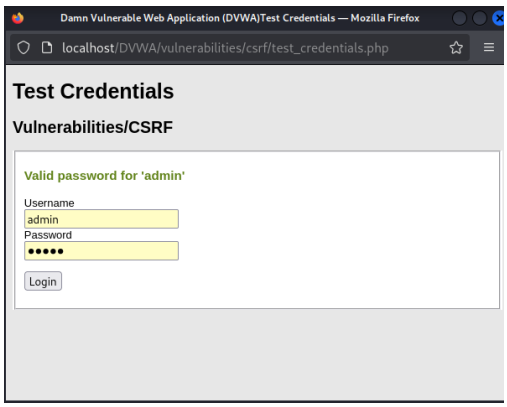
We create an html file, in which there is a button that when clicked will send the data that is burned in the code instead of capturing the data that the user entered but the data that I want.


kali-linux-2024.4-virtualbox-amd64 [Running]
File Actions Edit View Help
GNU nano 8.3
csrf.html
Search the web
app.hackth...
chatgpt
localhost
Left ⌘


kali-linux-2024.4-virtualbox-amd64 [Running]
CSRF Test
file:///var/www/html/DVWA/csrf.html
Click to Change Password
Change Password
Left ⌘


kali-linux-2024.4-virtualbox-amd64 [Running]
file:///var/www/html/DVWA/csrf.html
file://
Password change request sent!
OK

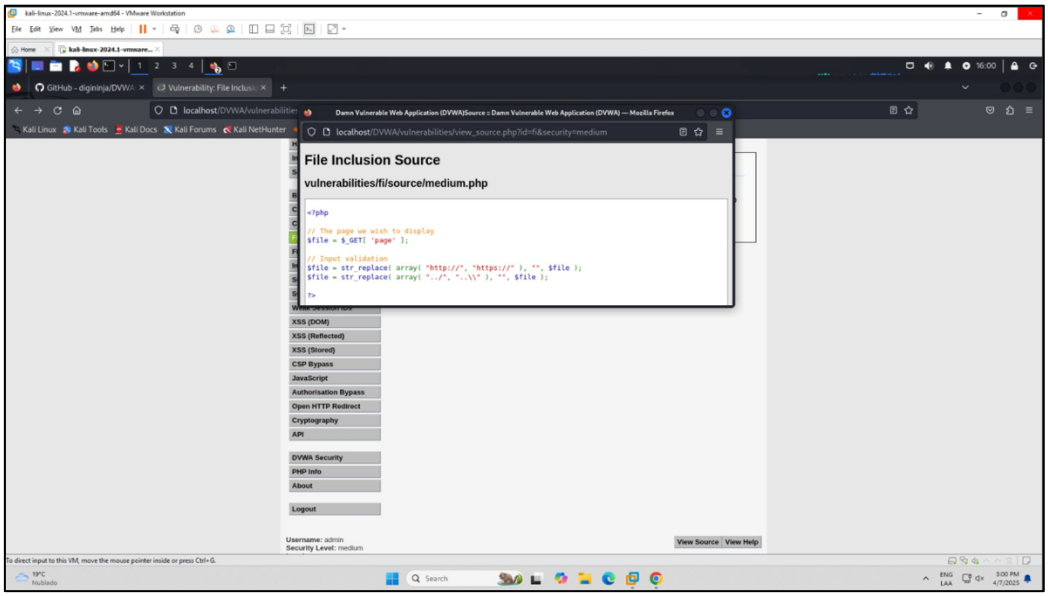

kali-linux-2024.4-virtualbox-amd64 [Running]
Damn Vulnerable Web Application (DVWA)Test Credentials — Mozilla Firefox
localhost/DVWA/vulnerabilities/csrf/test_credentials.php
Test Credentials
Vulnerabilities/CSRF
Username
admin
Password
Login
Passwords did not match.
More Information


Damn Vulnerable Web Application (DVWA)Test Credentials — Mozilla Firefox
localhost/DVWA/vulnerabilities/csrf/test_credentials.php
Test Credentials
Vulnerabilities/CSRF
Valid password for 'admin'
Username
admin
Password
Login

This test confirms that DVWA is vulnerable to CSRF attacks, allowing an attacker to change passwords or perform other sensitive actions if the target user clicks on the malicious link.

- ***FILE INCLUTION***

File inclusion is a vulnerability where an application allows users to choose which files to upload. If not managed securely, hackers can exploit it to upload sensitive files or even execute harmful code.

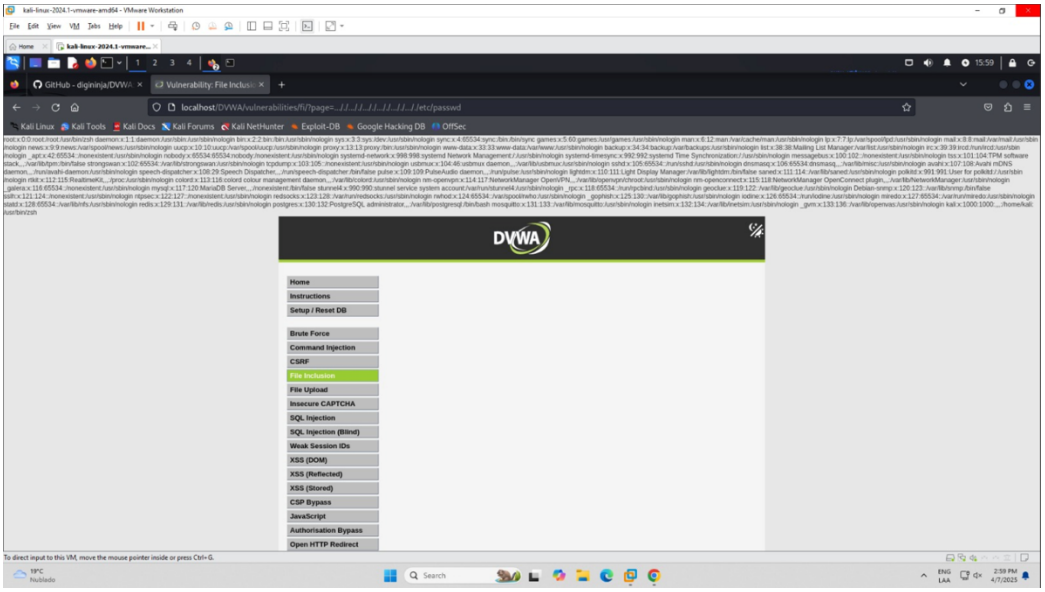


Analyzing the source code we can see how the function replaces http:// with empty strings. Also, characters such as "/" or "\\" are replaced with empty strings. However, if we use ..././, the .. / in the middle is replaced with a blank space and the remaining values are contracted to .. /, allowing us to navigate using this approach.

Therefore, we will use the following path to proceed with this attack:

```
..././..././..././..././..././..././..././etc/passwd
```

After searching for this url, the result we get is as follows:

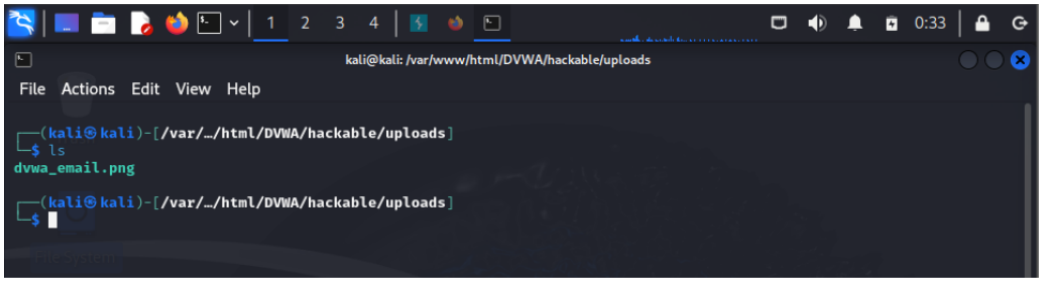


- ***File Upload (We can't do it at medium level and we do it at low level)***

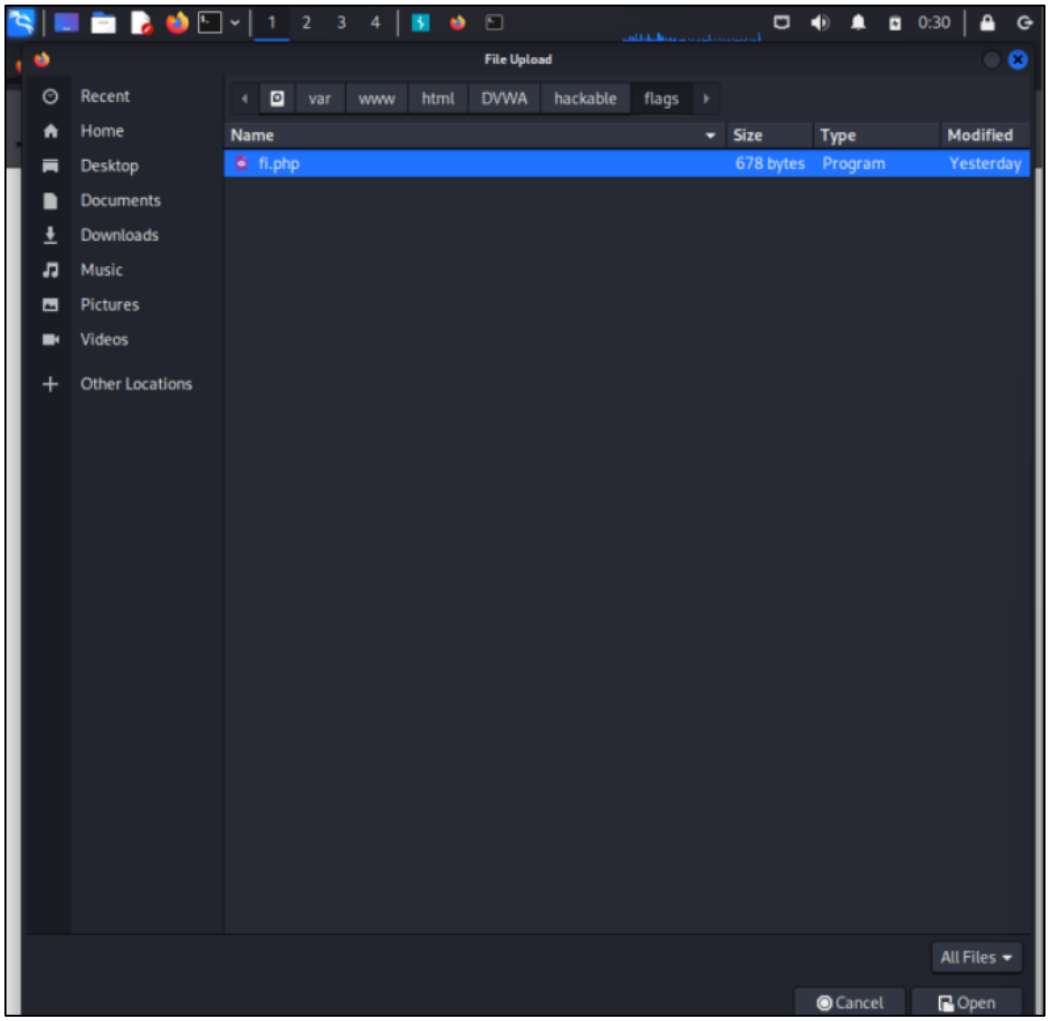
Unrestricted file upload allows an attacker to upload malicious files to the server. If the uploaded file is an executable script, the attacker can use it to compromise the system and execute arbitrary commands.

We can notice that the file that is loaded must be of type image/jpeg or image/png.

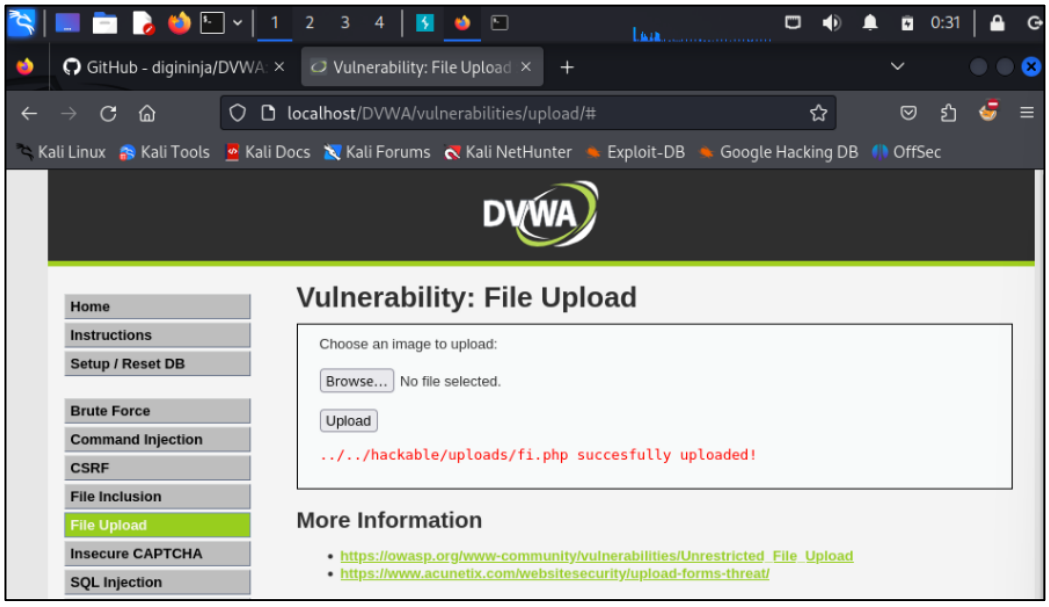
Initially, the /var/www/html/DVWA/hackable/uploads directory is checked to understand the contents and verify the location where files are being stored. This provides a baseline of what is present before any new files are uploaded.



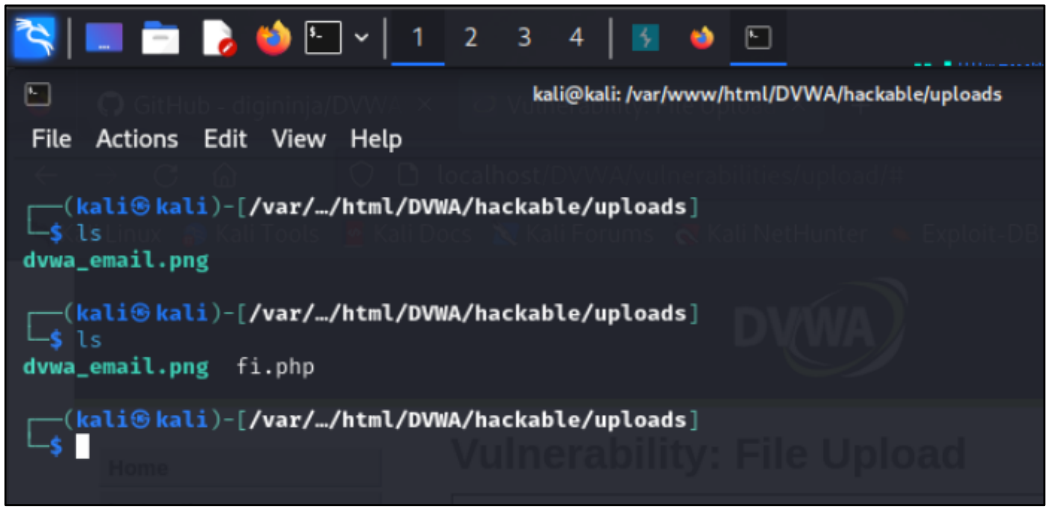
The file upload module in DVWA is then used to upload a PHP file containing malicious code. This file is crafted to execute specific commands upon being accessed, enabling the attacker to perform remote code execution.



After uploading, the fi.php file from the previous point (containing the quotes) is accessed to confirm that the upload functionality is indeed processing PHP files. This test demonstrates the server's vulnerability in executing scripts from uploaded files.

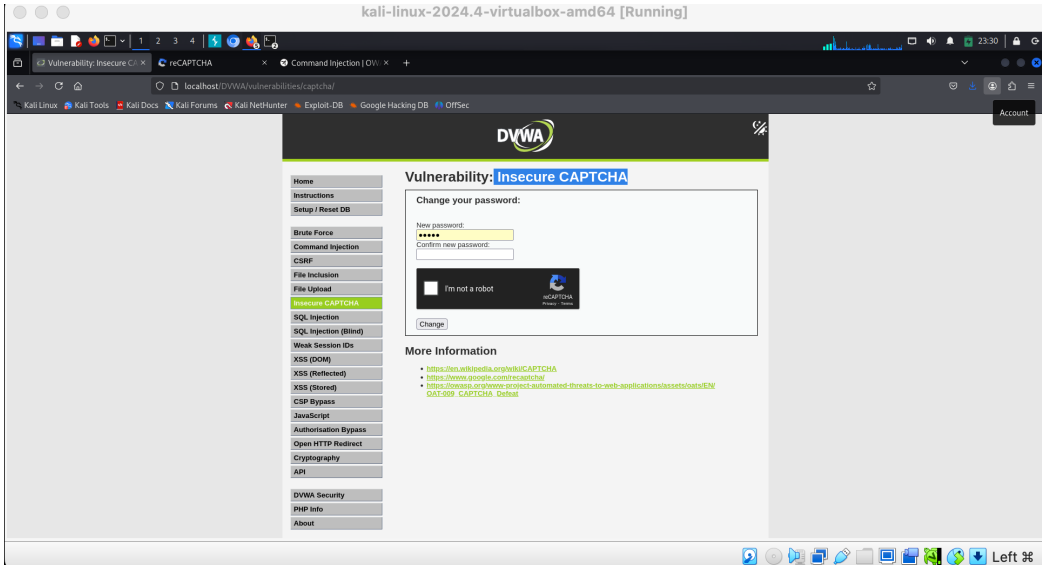


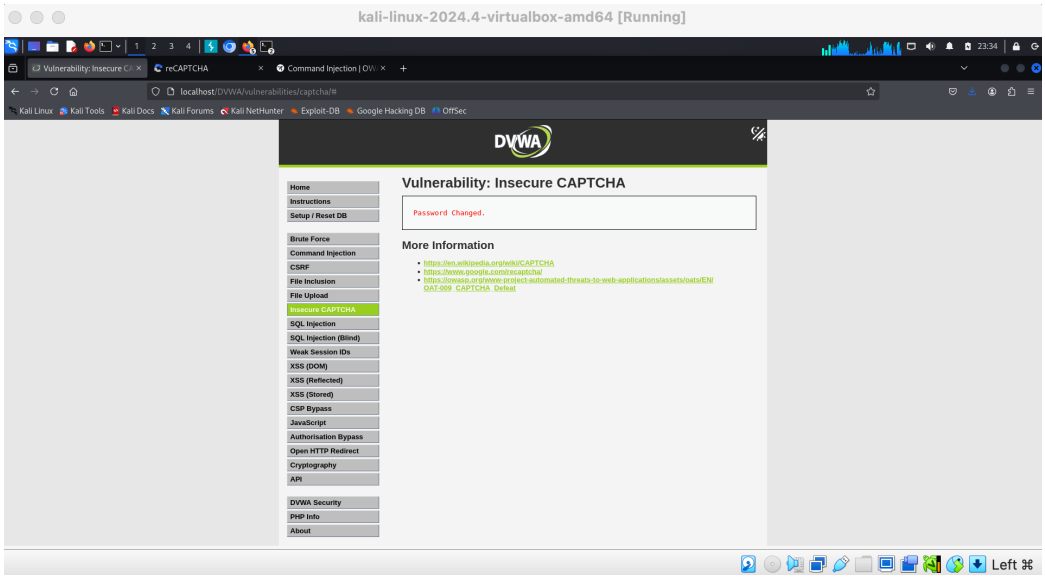
Finally, the /var/www/html/DVWA/hackable/uploads directory is revisited to confirm that the uploaded PHP file is present and accessible. By successfully uploading and accessing this malicious file, remote command execution on the server is verified, demonstrating the critical risk posed by unrestricted file upload vulnerabilities.



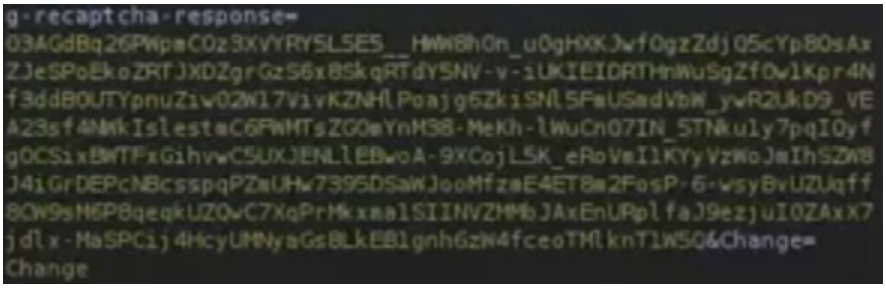
- ***Insecure CAPTCHA***

First what we will do is see how it works, so we will proceed to change the password and complete the captcha.





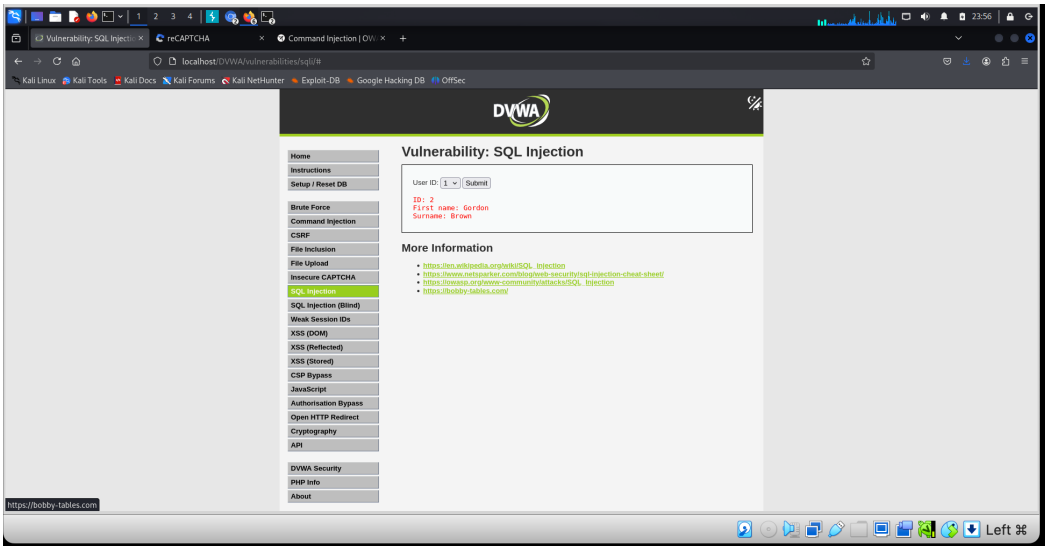
After we have completed the captcha using Burp Suite we first look at the code generated by the captcha. And as we can see at the bottom "change" which tells us that the password was successfully changed. This redirected to another page that through a "password changed" confirms that the request was made correctly.



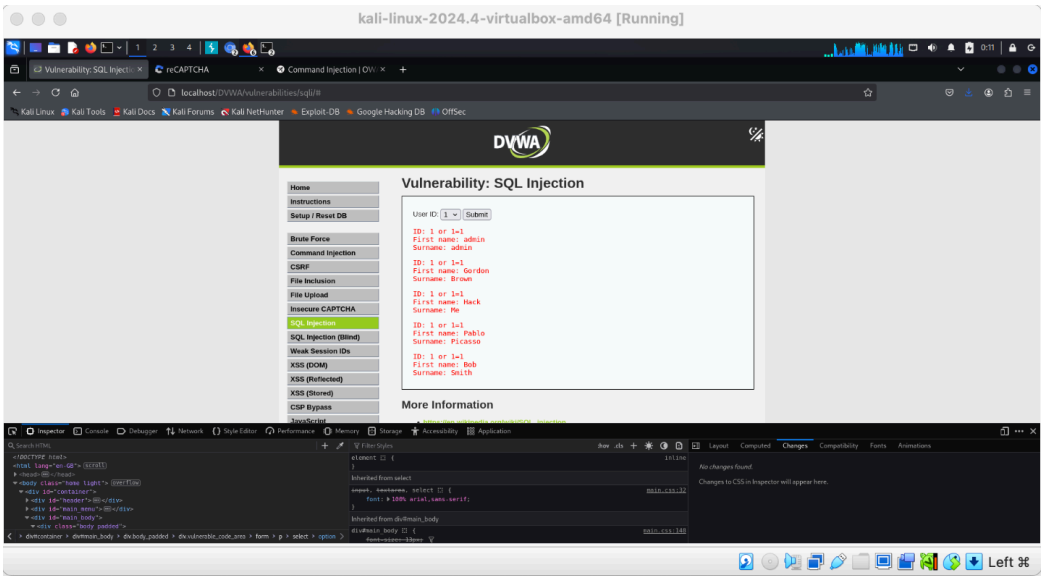
Now making use of the same tool what we will do is look for a similar POST request. However, in this case what we will do is turn it to repeat and change both variables "password_new" and "password_conf" so that they are not the password that was sent but the one that the attacker wants.

- ***SQL Injection***

SQL Injection allows the execution of arbitrary SQL queries in the database, accessing or manipulating sensitive information without authorization. This is one of the most critical vulnerabilities in web applications.

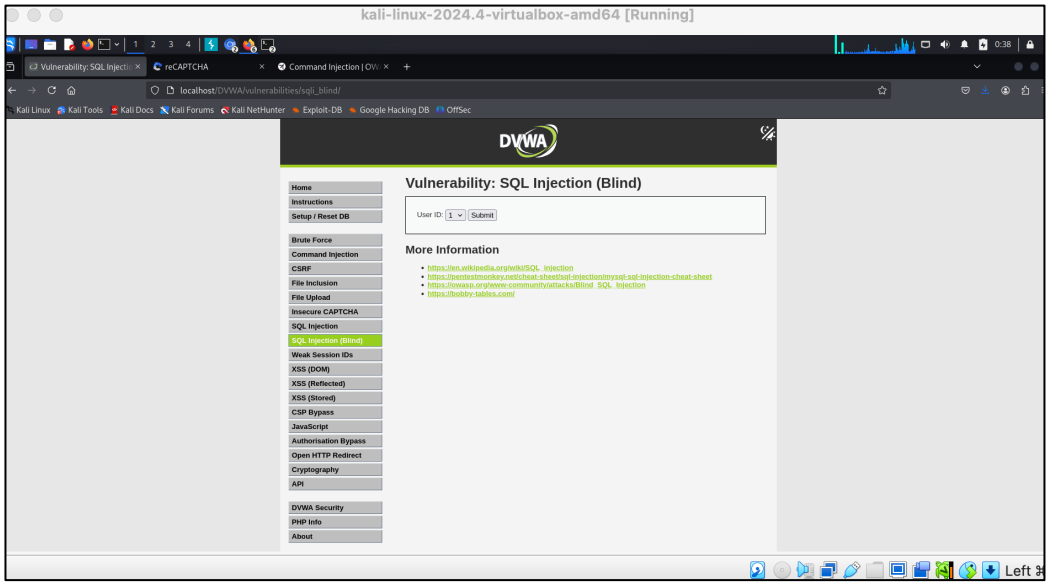


With what is captured in the burpSuite we will modify the query a bit using command such as or to concatenate the queries. And with this we modified the HTML code from the selection box and added things about the queries obtained, one of the queries that we got successful was when we put ***value="1or 1=1"***

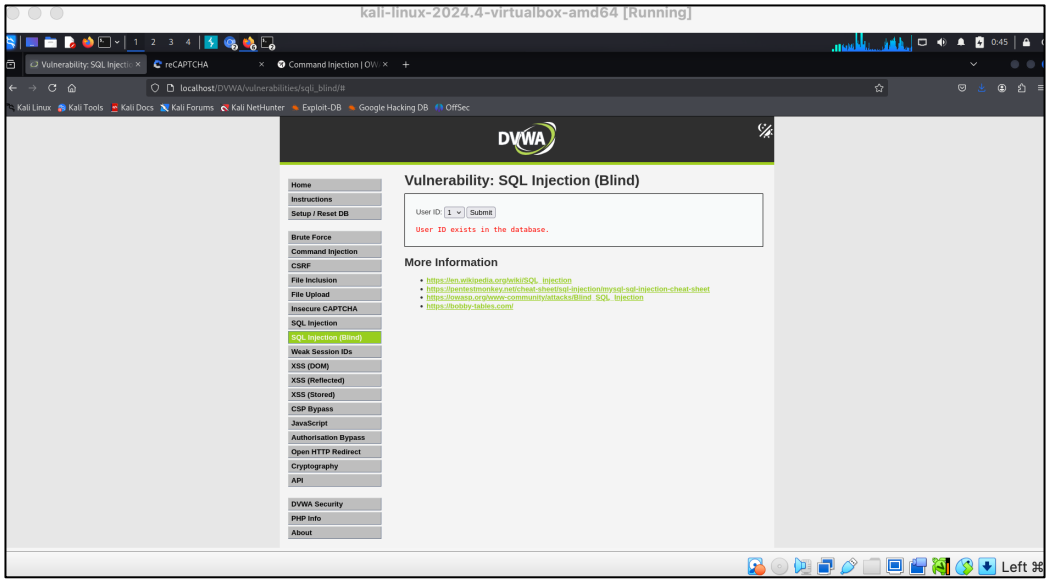


- ***Blind SQL Injection***

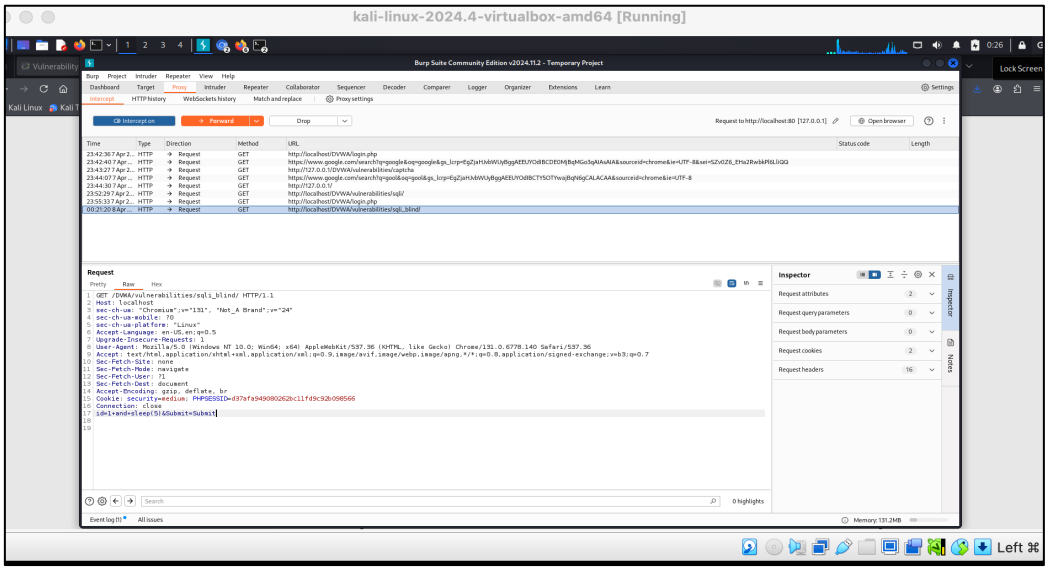
Blind SQL Injection is similar to regular SQL Injection, but in this case, the application does not return useful error messages. Boolean-based or time-based attacks allow an attacker to deduce information by observing the response time or application behavior.



First what we will do is try to identify through the answers that the requests give us what information is useful about the database and how to violate it, but this time in a hurry since we will not have the command box to be able to enter them.



Once this is done, we will move on to trying to enter sql commands using the Burp suite. This way we can modify this request and make an injection into the database.


kali-linux-2024.4-virtualbox-amd64 [Running]
Burp Suite Community Edition v2024.11.2 - Temporary Project
Request
Inspector
Left ⌘