

Zeek Network Traffic Analysis and Security Detection

CS6903 - Network Security Assignment

Task 1: Network Traffic Collection and Analysis

Task 1A: Capturing Network Traffic on Personal Laptop

This task involved capturing real-time network traffic from my laptop and identifying the most active source IPs. Capturing live traffic helps in understanding network patterns, identifying common communication endpoints, and detecting potential anomalies.

To achieve this, I used tcpdump, a powerful packet capture tool, to collect network data from my active interface. The captured packets were then analyzed using Zeek to extract meaningful insights, such as the most frequently communicating IPs.

Capturing Network Traffic:

```
sudo tcpdump -i en0 -w my_traffic.pcap -s 0
```

```
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1A % sudo tcpdump -i en0 -w my_traffic.pcap -s 0
Password:
tcpdump: listening on en0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C58568 packets captured
58581 packets received by filter
0 packets dropped by kernel
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1A %
```

- Tcpcap captures network packets in real-time.
- -i en0 specifies the network interface (Wi-Fi in this case).
- -w my_traffic.pcap saves the captured traffic to a file for further analysis.
- -s 0 ensures the full packet headers are stored without truncation.
- The process was terminated manually using Ctrl+C after 10 minutes to prevent excessive data collection.

Processing PCAP file with Zeek:

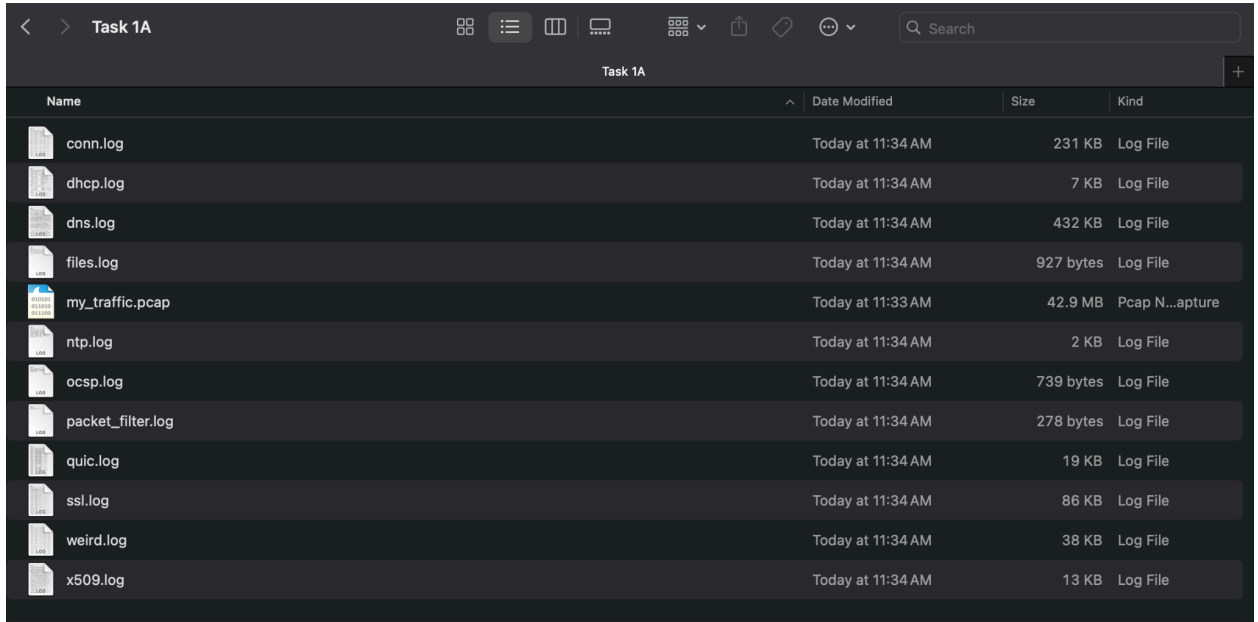
```
zeek -r my_traffic.pcap
```

```
(base) gulshanhatzade@Gulshans-MacBook-Air Zeek % zeek -r my_traffic.pcap
(base) gulshanhatzade@Gulshans-MacBook-Air Zeek %
```

- This command reads the captured packet file and processes it with Zeek.

- Zeek generates multiple logs (conn.log, dns.log, etc.), which store extracted metadata about the captured traffic.

Generated files-



Name	Date Modified	Size	Kind
conn.log	Today at 11:34 AM	231 KB	Log File
dhcp.log	Today at 11:34 AM	7 KB	Log File
dns.log	Today at 11:34 AM	432 KB	Log File
files.log	Today at 11:34 AM	927 bytes	Log File
my_traffic.pcap	Today at 11:33 AM	42.9 MB	Pcap N...apture
ntp.log	Today at 11:34 AM	2 KB	Log File
ocsf.log	Today at 11:34 AM	739 bytes	Log File
packet_filter.log	Today at 11:34 AM	278 bytes	Log File
quic.log	Today at 11:34 AM	19 KB	Log File
ssl.log	Today at 11:34 AM	86 KB	Log File
weird.log	Today at 11:34 AM	38 KB	Log File
x509.log	Today at 11:34 AM	13 KB	Log File

Extracting Most Active Source IPs:

```
cat conn.log | zeek-cut id.orig_h | sort | uniq -c | sort -nr | head -10
```

```
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1A % cat conn.log | zeek-cut id.orig_h | sort | uniq -c |
sort -nr | head -10
1323 172.21.149.18
38 172.21.148.197
21 172.21.148.180
18 172.21.148.247
17 172.21.149.24
14 fe80::ce45:6bb8:6ff5:fb00
12 172.21.149.12
11 172.21.149.20
11 172.21.148.235
9 172.21.148.183
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1A %
```

- cat conn.log prints the contents of the connection log.
- zeek-cut id.orig_h extracts only the source IP addresses from the log.
- sort | uniq -c | sort -nr counts occurrences and sorts them in descending order.
- head -10 displays the top 10 most active IPs based on connection frequency.

Findings:

- **Captured 58,568 packets** using tcpdump, with no packet loss by the kernel.

- **Top active source IP:** 172.21.149.18 with **1,323 connections**, significantly higher than others.
- Other active IPs had **38 or fewer connections**, suggesting 172.21.149.18 was the dominant communication source.
- The presence of **IPv6 address (fe80::ce45:6bb8:6ff5:fb00)** indicates local network communication.
- This pattern suggests a **server, frequent client-server interaction, or potentially an automated process generating traffic.**

Task 1B: Public PCAP Dataset Analysis

For this subtask, I analyzed a publicly available PCAP dataset to compare its network traffic with my locally captured data. This dataset comes from a real-world scenario and provides insights into active network entities and their behavior.

By processing the dataset using Zeek, I extracted the most active source IPs, similar to Task 1A. This allows for understanding how real network environments generate traffic and whether certain IPs dominate communication.

Link of pcap file -

https://mcfp.felk.cvut.cz/publicDatasets/CTU-Mixed-Capture-5/2015-03-19_winnormal.pcap

Processing the PCAP file:

zeek -r 2015-03-19_winnormal.pcap

```
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1B % zeek -r 2015-03-19_winnormal.pcap
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1B % █
```

- -r 2015-03-19_winnormal.pcap tells Zeek to analyze the dataset and generate log files.
- Produces logs such as conn.log, dns.log, http.log, etc.

Generated files-

Task 1B			
Name	Date Modified	Size	Kind
2015-03-19_winnormal.pcap	Today at 9:39 PM	181.9 MB	Pcap N...apture
analyzer.log	Today at 9:40 PM	3 KB	Log File
conn.log	Today at 9:40 PM	252 KB	Log File
dns.log	Today at 9:40 PM	142 KB	Log File
dpd.log	Today at 9:40 PM	2 KB	Log File
files.log	Today at 9:40 PM	97 KB	Log File
http.log	Today at 9:40 PM	439 KB	Log File
ocsp.log	Today at 9:40 PM	5 KB	Log File
packet_filter.log	Today at 9:40 PM	278 bytes	Log File
pe.log	Today at 9:40 PM	794 bytes	Log File
ssl.log	Today at 9:40 PM	98 KB	Log File
weird.log	Today at 9:40 PM	625 bytes	Log File
x509.log	Today at 9:40 PM	28 KB	Log File

Extracting Most Active Source IPs:

```
cat conn.log | zeek-cut id.orig_h | sort | uniq -c | sort -nr | head -10
```

```
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1B % cat conn.log | zeek-cut id.orig_h | sort | u
niq -c | sort -nr | head -10

2103 10.0.2.200
25 10.0.2.2
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1B %
```

- cat conn.log prints the connection log.
- zeek-cut id.orig_h extracts only the source IP addresses.
- sort | uniq -c | sort -nr counts and sorts IPs by activity.
- head -10 displays the top 10 active IPs.

Findings:

- The most active IP was 10.0.2.200 with 2103 connections.
- This suggests either a heavily used server or an automated script performing repeated network operations.

Task 2: Destination Port Analysis

Task 2A: Top 10 Destination Ports from Captured Data

In this subtask, I analyzed the captured network traffic from Task 1A to identify the most frequently accessed destination ports. Destination ports indicate which network services (such as web browsing, DNS resolution, or remote access) were most actively used.

By processing the conn.log file using Zeek, I extracted the top 10 most accessed destination ports. This helps in understanding common communication patterns and detecting any unusual port usage that might indicate unauthorized services or network misconfigurations.

Extracting Top Destination Ports:

```
cat conn.log | zeek-cut id.resp_p | sort | uniq -c | sort -nr | head -10
```

```
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1A % cat conn.log | zeek-cut id.resp_p | sort | uniq -c |
sort -nr | head -10

705 53
591 443
364 5353
30 1947
17 135
13 137
9 3
8 57621
8 22222
4 68
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1A %
```

- cat conn.log prints the connection log.
- zeek-cut id.resp_p extracts only the destination port numbers.
- sort | uniq -c | sort -nr counts and sorts ports based on frequency.
- head -10 displays the top 10 most accessed ports.

Findings:

- Most accessed destination port: 53 (DNS) with 705 requests, indicating frequent domain name resolution.
 - Port 443 (HTTPS) received 591 connections, confirming secure web browsing as a major activity.
 - Port 5353 (Multicast DNS) with 364 requests suggests local network service discovery.
- Uncommon ports detected:
- 1947 (Sentinel Licensing Service) – Could be related to software license verification.
 - 22222 – Often used for custom SSH or administrative access, requiring further inspection.

- 57621 and 68 (DHCP-related) – May indicate dynamic IP assignments or internal device communications.

Task 2B: Top 10 Destination Ports from Public Dataset

For this subtask, I analyzed the public dataset from Task 1B to extract the most accessed destination ports. This helps in comparing real-world network behavior with my locally captured data.

Link of pcap file -

https://mcfp.felk.cvut.cz/publicDatasets/CTU-Mixed-Capture-5/2015-03-19_winnormal.pcap

Extracting Top Destination Ports:

```
cat conn.log | zeek-cut id.resp_p | sort | uniq -c | sort -nr | head -10
```

```
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1B % cat conn.log | zeek-cut id.resp_p | sort | u  
niq -c | sort -nr | head -10  
  
853 443  
525 53  
402 80  
30 5355  
26 0  
23 40009  
21 12350  
18 40014  
17 40015  
15 40023  
(base) gulshanhatzade@Gulshans-MacBook-Air Task 1B %
```

Findings:

- **Port 443 (HTTPS) was the most accessed with 853 connections**, indicating a high volume of encrypted web traffic.
- **Port 53 (DNS) received 525 requests**, showing frequent domain name lookups.
- **Port 80 (HTTP) had 402 connections**, suggesting unencrypted web browsing or API calls.
- **Unusual ports detected:**
 - **40009, 40014, 40023** – These could be associated with **custom services, internal applications, or automated network processes**.
 - The dataset reflects **web-heavy traffic patterns**, likely including **browsing, API requests, or automated tasks**.

- The presence of **non-standard ports** may indicate **proprietary services, internal communications, or potential security risks** that require further inspection.

Task 3: Identifying Self-Signed Certificates

Self-signed certificates are commonly used in internal networks but can also be a sign of security risks when seen in external traffic. Attackers use them in phishing and **man-in-the-middle attacks**.

To detect them, I created a Zeek script that checks if the certificate's issuer and subject fields are the same.

Step 1: Created a Zeek Script

The Zeek script analyzes SSL/TLS traffic to detect self-signed certificates by checking if the issuer and subject fields are the same (`rec$issuer == rec$subject`). If a match is found, it prints the server name, subject, and issuer details. The script also includes `zeek_init` to indicate the start of analysis and `zeek_done` to signal completion. While it effectively detects self-signed certificates, adding a check for `rec$cert_chain` could improve accuracy by ensuring valid certificate data is available.

Then visiting <https://self-signed.badssl.com/>

Step 2: Capture SSL Traffic

`sudo tcpdump -i en0 port 443 -w ssl_traffic.pcap -s 0`

```
(base) gulshanhatzade@Gulshans-MacBook-Air (base) gulshanhatzade@Gulshans-MacBook-Air (base) guls(base) gulshanhatzade@Gulshans-MacBook-Air Task 3 % sudo tcpdump -i en0 port 443 -w ssl_traffic.pcap -s 0
Password:
tcpdump: listening on en0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C411 packets captured
1000 packets received by filter
0 packets dropped by kernel
```

- Captures all HTTPS (SSL/TLS) traffic on port **443**.
- `-w ssl_traffic.pcap` saves the packets to a file for later analysis.
- The packet capture was stopped manually after visiting a test website.

Testing the Script:

`zeek -r ssl_traffic.pcap self_signed_cert.zeek`

```

Now analyzing the S.S.L. Traffic for the Self Signed Certificate . . .
Self signed certificate is detected for this task 3.
Printing the server Name - self-signed.badssl.com
Printing the Subject - CN=*.badssl.com,O=BadSSL,L=San Francisco,ST=California,C=US
Printing the Issuer - CN=*.badssl.com,O=BadSSL,L=San Francisco,ST=California,C=US

Self signed certificate is detected for this task 3.
Printing the server Name - self-signed.badssl.com
Printing the Subject - CN=*.badssl.com,O=BadSSL,L=San Francisco,ST=California,C=US
Printing the Issuer - CN=*.badssl.com,O=BadSSL,L=San Francisco,ST=California,C=US

Self signed certificate is detected for this task 3.
Printing the server Name - self-signed.badssl.com
Printing the Subject - CN=*.badssl.com,O=BadSSL,L=San Francisco,ST=California,C=US
Printing the Issuer - CN=*.badssl.com,O=BadSSL,L=San Francisco,ST=California,C=US

Self signed certificate is detected for this task 3.
Printing the server Name - self-signed.badssl.com
Printing the Subject - CN=*.badssl.com,O=BadSSL,L=San Francisco,ST=California,C=US
Printing the Issuer - CN=*.badssl.com,O=BadSSL,L=San Francisco,ST=California,C=US

Finally SSL Analysis is completed.

```

- Processes the captured SSL traffic using Zeek.
- Applies the self-signed certificate detection script.

Findings:

- Successfully detected a self-signed certificate from badssl.com.
- The analysis of SSL traffic revealed the presence of a self-signed certificate issued by BadSSL. The server name detected was self-signed.badssl.com, with both the subject and issuer fields being identical, confirming that the certificate is self-signed. The details indicate that the certificate belongs to CN=*.badssl.com, O=BadSSL, L=San Francisco, ST=California, C=US.
- Since self-signed certificates do not have a trusted Certificate Authority (CA), they can pose security risks, especially in public-facing applications. Attackers often use such certificates in man-in-the-middle (MITM) attacks to intercept encrypted communications. The repeated detection of the same self-signed certificate suggests that multiple connections were established using it.

Task 4: Detecting SSH Brute-Force Attacks

Brute-force attacks involve repeated login attempts to guess SSH credentials. This can be detected by monitoring failed authentication attempts.

I created a Zeek script that logs any SSH client making more than **10 failed login attempts** from the same IP.

Step 1: Download the SSH Brute-Force PCAP File

Downloaded pcap file from given link

<https://github.com/bro/bro/raw/master/testing/btest/Traces/ssh/sshguess.pcap>

Step 2: Writing the Zeek Script

Wrote script for zeek

The script tracks SSH login attempts per IP and logs any that exceed 10 failed attempts, flagging them as brute-force attacks. It then prints a summary of detected attacks at the end.

Step 3: Run Zeek on the PCAP File

zeek -C -r sshguess.pcap detect_ssh_bruteforce.zeek

```
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 0 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 1 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 2 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 3 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 4 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 5 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 6 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 7 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 8 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 9 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[DEBUG] The Failed login attempt detecting from 192.168.56.1
[DEBUG] Now 10 failed attempts is recorded for 192.168.56.1 in the last 60 seconds
[ALERT] The SSH Brute-Forcing Detected from 192.168.56.1! 10 attempts in 60 seconds - [Name: Gulshan Hatzade, Roll No: CS24MTECH14006]
```

This command runs **Zeek** to analyze the given **SSH traffic capture file (sshguess.pcap)** while executing the **custom brute-force detection script (detect_ssh_bruteforce.zeek)**.

- **-C** : Disables checksum validation, ensuring that Zeek processes all packets, even if they have invalid checksums. This is useful when working with capture files from different network environments.
- **-r sshguess.pcap** : Reads and analyzes the given **packet capture (PCAP) file**, which contains recorded SSH traffic.
- **detect_ssh_bruteforce.zeek** : Runs the **custom Zeek script** designed to detect brute-force attacks by tracking repeated failed login attempts from the same IP.

By executing this command, Zeek inspects **SSH authentication attempts** in the PCAP file and logs any IPs that exceed the predefined **brute-force threshold**, flagging them as potential attackers.

Findings:

- The Zeek script successfully detected a **brute-force attack** originating from **IP 192.168.56.1**, which made **10 failed SSH login attempts within 60 seconds**. The **debug logs** confirm that each failed attempt was recorded incrementally until the **threshold of 10 attempts** was reached. Once the threshold was exceeded, the script **triggered an alert**, indicating that a brute-force attack was detected.
- This finding highlights the effectiveness of the **time-based brute-force detection mechanism**, ensuring that rapid failed login attempts are flagged while ignoring slow, distributed attacks over a long period. To further mitigate such attacks, implementing **fail2ban, IP blocking, or disabling password-based SSH authentication** would enhance security.

The detection confirms that the **Zeek script successfully identified the brute-force attack**, making it useful for intrusion detection and prevention systems

Anti-Plagiarism Statement

I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been

granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarized the work of other students and or persons. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honor violations by other students if I become aware of it.

Name: CS24MTECH14006

Date: 07/03/2025

Signature: Gulshan Hatzade