SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

CERTAIN INVESTIGATION OF MALICIOUS AND NORMAL TRAFFIC INTERPRETATION

INT 400 – INTERNSHIP

PROJECT REPORT

Submitted by

AKISH RAJ A – E0222047

DHARSON RAM – E0222025

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Cyber Security & Internet of Things)

Sri Ramachandra Faculty of Engineering and Technology

Sri Ramachandra Institute of Higher Education and Research, Porur, Chennai - 600116

OCTOBER, 2024

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

CERTAIN INVESTIGATION OF MALICIOUS AND NORMAL TRAFFIC INTERPRETATION

INT 400 – INTERNSHIP

PROJECT REPORT

Submitted by

AKISH RAJ A – E0222047

DHARSON RAM – E0222025

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

(Cyber Security & Internet of Things)

Sri Ramachandra Faculty of Engineering and Technology

Sri Ramachandra Institute of Higher Education and Research, Porur, Chennai - 600116

OCTOBER, 2024

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

BONAFIDE CERTIFICATE

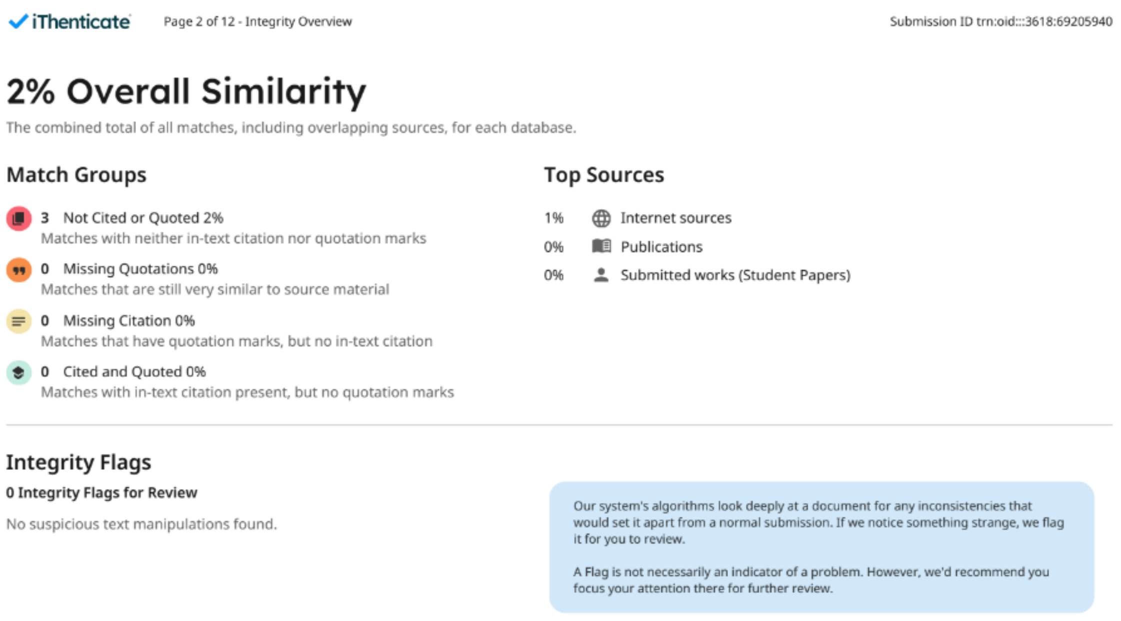
Certified that this project report **"CERTAIN INVESTIGATION OF MALICIOUS AND NORMAL TRAFFIC INTERPREATION"** is the bonafide record of work done by **"AKISH RAJ A – E0222047"** who carried out the internship work under my supervision.

Signature of the Supervisor	**Signature of Programme Coordinator**
Dr. ANAND J V	**Dr. Jayanthi G**
Assistant Professor,	**Associate Professor,**
Department of Cybersecurity and IoT	Department of Cybersecurity and IoT
Sri Ramachandra Faculty of Engineering and Technology,	Sri Ramachandra Faculty of Engineering and Technology,
SRIHER, Porur, Chennai-600 116.	SRIHER, Porur, Chennai-600 116.

Evaluation Date:

INTERNAL EXAMINER **EXTERNAL EXAMINER**

Plagiarism Report



iThenticate Page 2 of 12 - Integrity Overview Submission ID trn:oid:::3618:69205940

2% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- ● 3 Not Cited or Quoted 2%
  Matches with neither in-text citation nor quotation marks
- ● 0 Missing Quotations 0%
  Matches that are still very similar to source material
- ● 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- ● 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

1%	Internet sources
0%	Publications
0%	Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Signature of the Supervisor

SRI RAMACHANDRA
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Category - I Deemed to be University) Porur, Chennai
SRI RAMACHANDRA FACULTY OF ENGINEERING AND TECHNOLOGY

ACKNOWLEDGEMENT

I take this opportunity to express my gratitude to **Prof. T. Ragunathan, Dean, Sri Ramachandra Faculty of Engineering and Technology, SRIHER** for providing all the facilities to complete this Internship work successfully.

I express my sincere gratitude to **Dr. G. Jayanthi, Programme Coordinator, Department of Cybersecurity & Internet of Things, SRET** for their support and for providing the required facilities for carrying out this study.

I wish to thank my faculty supervisor(s), **Dr. ANAND J V,** Department of Cybersecurity and Internet of Things, SRET for extending help and encouragement throughout the project. Without his continuous guidance and persistent help, this project would not have been a success for me.

I am grateful to all the members of Sri Ramachandra Faculty of Engineering and Technology, my beloved parents and friends for extending the support, who helped us to overcome obstacles in the study.

TABLE OF CONTENTS

LIST FIGURES

ABSTRACT

The project Certain Investigation on Malicious and Normal Traffic Interpretation aimed to explore selected network problems by examining in the Wireshark traffic captured related with either malware, slow connection or other anomalies. etc Using various Wireshark filters, this project captures and analyzes the traffic to see different patterns in incoming data as normal versus malicious behavior. Here the Python is to process the packet capture (PCAP) files saved so far into a structured format destined for further analysis. The project consists of plotting IP address destinations on a map to visualize where traffic is going. The project also contains some automation scripts written in TShark (Wireshark's CLI version) to help with repetitive and simplified traffic analysis. Together, these build bottoms-up layer of rationale necessary to address the network security challenges in a holistic manner that includes automation at every level.

CHAPTER 1
INTRODUCTION

1.1. INTRODUCTION

Differentiating between regular and malicious traffic is required to protect against cyber-attacks, detect performance problems, and support stable operation. The project: Certain Investigation of Malicious and Normal Traffic Interpretation uses Wireshark, which is one among the various network protocol analyzers that allow logs to be viewed interactively or offline preprocess analysis on hosts in mannerable pattern for identification from malicious problems like malware and slow connections. Finally, the results are analyzed in python which takes PCAP files as input and based on destination IP provides good information on how traffic flows geographically. Also, TShark enables analyzer automation scripts of network traffic, it substitutes manual works and analyze processes greatly simplified. The sum of these parts provides a powerful source for monitoring and security investigations on your network in real time.

1.2. TECHNIQUES INVOLVED

1.2.1 Packet Capturing with Wireshark:

Wireshark is a network packet analyzer that enables you to capture and view packets traveling on a live network. This allows a detailed insight while looking into packets data for normal as well as abnormal activities. Variety: Various kinds of filters are added to segregate certain traffic such as Malware, Poor Connection and different out of the box patterns.

1.2.2 Filtering and Analysis:

With the help of Wireshark filter options, you can easily sift through large amounts of traffic and look specifically at what protocol is having trouble. This allows for profiling of packet information such as source, destination IP addresses; protocols and flags to identify possible anomalies or security events.

1.2.3 PCAP File Analysis Using Python:

There are pcap files that have been saved from Wireshark which need processing in Python. This data is then sniffed, pulled out via Python scripts (ZIP) and used to understand the network traffic by analyzing it. A major conclusion that can be drawn from this analysis is location of the destination IP geolocations to show on a map about where the traffic goes.

1.2.4 Automation with TShark:

TShark is used for traffic analysis tasks from automated scripts and is the command line version of Wireshark in some way. TShark is also scripted in small automation scripts which can be used to capture network traffic, filter it and then analyze it without human interaction making the whole process automated.

1.3. DATA COLLECTION

Wireshark is used to collect network traffic, which is saved in PCAP files. These filters are protocol-based or target a type of suspicious behavior It extracts key metadata such as source/destination IP, and timestamps for further analysis. Python scripts do this to convert the destination IPs into geolocations which it help visual mapping. With TShark being automated, you have an automatic traffic capture running in the background.

CHAPTER 2

LITERATURE SURVEY

1.Title: Network Traffic Analysis Using Wireshark and TShark Automation

Patel and Kumar (2023) propose the use of network traffic analysis methods utilizing Wireshark and describe its usefulness in cases of malicious activities. According to the authors, the program has the real-time packet-filtering function, as well as joining to TShark. Authors of this work also refer to the role of Python in location and graphical representation of traffic patterns and emphasize how seeing an IP address plotted on a map helps detect the threat. The paper ends discussing the need and possibility of automating some aspects of the network traffic analysis for the sake of improving network security in different environments.

2.Title: Automation of Network Traffic Monitoring Using Python and TShark

Singh et al. (2022) explain the need for the integration of TShark into Python scripts in an automated manner in order to perform continuous operational traffic Analysis. While performing this duty, they stress the application of automation tools in order to minimize on human effort. The investigation presented in this research was intended to explore how automated scripts could enhance monitoring of traffic and threats without extensive contact with geolocation data and machine learning predictive models based on existing scripts.

3.Title: Real-Time Network Traffic Analysis Using Machine Learning

Gomez et al. (2023) investigate the application of machine learning in real-time network traffic analysis. They have propose a model combining packet filtering and feature extraction techniques from traffic data using Wireshark and TShark. The authors demonstrate how automation tools improve detection of anomalies by integrating classification algorithms like Random Forest and SVM. Their research highlights the efficiency of using machine learning models for both normal and malicious traffic, emphasizing the increased detection accuracy with real-time monitoring**.**

4.Title: Detecting Network Anomalies Through Traffic Patterns

Zhao and Fernandez (2022) focus on detecting anomalies by analyzing traffic patterns from Wireshark-collected data. Their research outlines the use of Python to automate geolocation mapping, which tracks the origin of suspicious IP addresses. They emphasize the benefits of combining packet analysis with geolocation visualization in identifying malicious traffic. Additionally, the study provides insights into implementing advanced algorithms to classify the nature of traffic flow, helping reduce false positives in detection.

5.Title: Visualizing Malicious Network Traffic Using Geolocation Techniques

Iqbal and Rathi (2023) explore the use of geolocation mapping techniques to visualize malicious network traffic. The authors employ Wireshark to capture traffic and Python-based geolocation libraries to identify the origin of potential threats. Their study discusses how mapping can reveal geographic clusters of malicious activity, aiding cybersecurity efforts. The authors suggest combining machine learning techniques for deeper traffic analysis and identifying new attack vectors. The research is valuable for enhancing threat visualization capabilities.

6.Title: Automation of Network Traffic Capture for Cybersecurity

Khan et al. (2024) present a framework that automates network traffic capture using TShark and Python for continuous monitoring. Their research demonstrates how automation reduces the manual effort required for monitoring large networks while improving detection of anomalies in real time. They discuss strategies to integrate Python-based scripts to automate traffic collection and analysis, highlighting the importance of scalability for larger networks. Their study underlines the effectiveness of automation in identifying emerging cyber threats**.**

7.Title: Enhancing Network Security with Wireshark and Python Automation

Ramos and Ali (2022) examine the application of Wireshark and Python for enhancing network security by automating packet analysis. They present case studies demonstrating the detection of abnormal traffic and how geolocation techniques in Python help trace attacks back to their sources. Their findings suggest that automation is key in handling the growing complexity of modern network environments, improving the accuracy and speed of detecting security breaches.

8.Title: Anomaly Detection in Network Traffic Through Automated Scripting

Patel et al. (2024) evaluate the use of automated scripts in detecting anomalies within network traffic. Here they study combines Wireshark for packet filtering with TShark and Python for automating the process. The research also explores integrating machine learning models to further analyze traffic anomalies and improve detection rates. They have propose developing real-time monitoring systems using scripting techniques to handle various network conditions and threats.

CHAPTER 3

PROPOSED METHODOLOGY :

Data Collection: Capture real-time network traffic using Wireshark, storing it in PCAP files. This traffic will contain both normal and potentially harmful packets.

Data Filtering: Leverage Wireshark's advanced filtering capabilities to detect anomalies in the network, such as malware, suspicious behaviors, or slow connections.

Automated Traffic Capture: Use TShark to automate the traffic capture process, enabling continuous monitoring of network activity.

Geolocation Mapping: Utilize Python, along with libraries like Pyshark and Geopy, to analyze the PCAP data and visually map the destination IP addresses.

Testing and Validation: Perform testing across different network environments to assess the effectiveness of the methodology in detecting network issues and malicious activities.

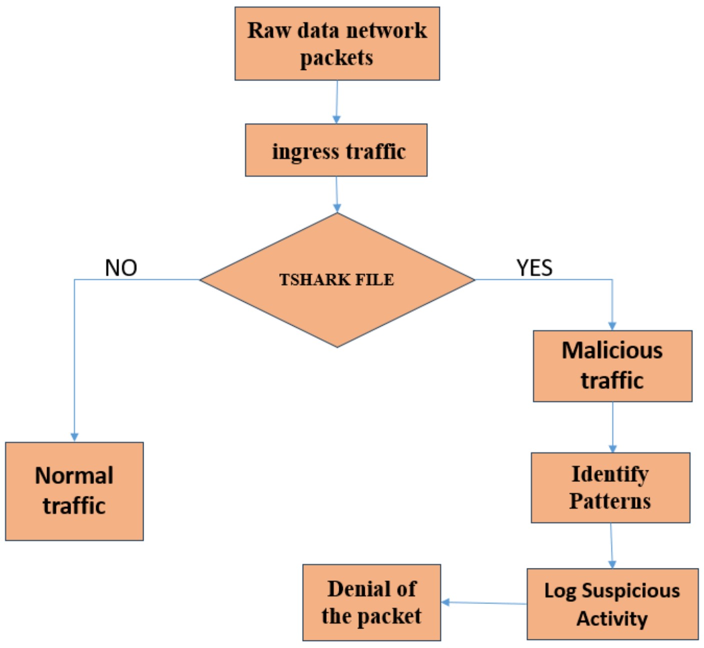


Fig 1 (WORKFLOW)

CHAPTER 4

IMPLEMENTATION :

1. **Setup Environment:** Install Wireshark and Tshark on your Linux machine, ensuring you have the necessary permissions to capture network traffic.

2. **Monitoring network traffic** : it provides detailed information such as timestamps, frame numbers, and packet sizes. At the network layer, it reveals source and destination MAC addresses, IP addresses, and protocols (like TCP, UDP, or ICMP). You can also see transport layer details like port numbers, packet flags, sequence numbers, and payload data, helping analyze network communication and detect issues like latency, dropped packets, or security threats

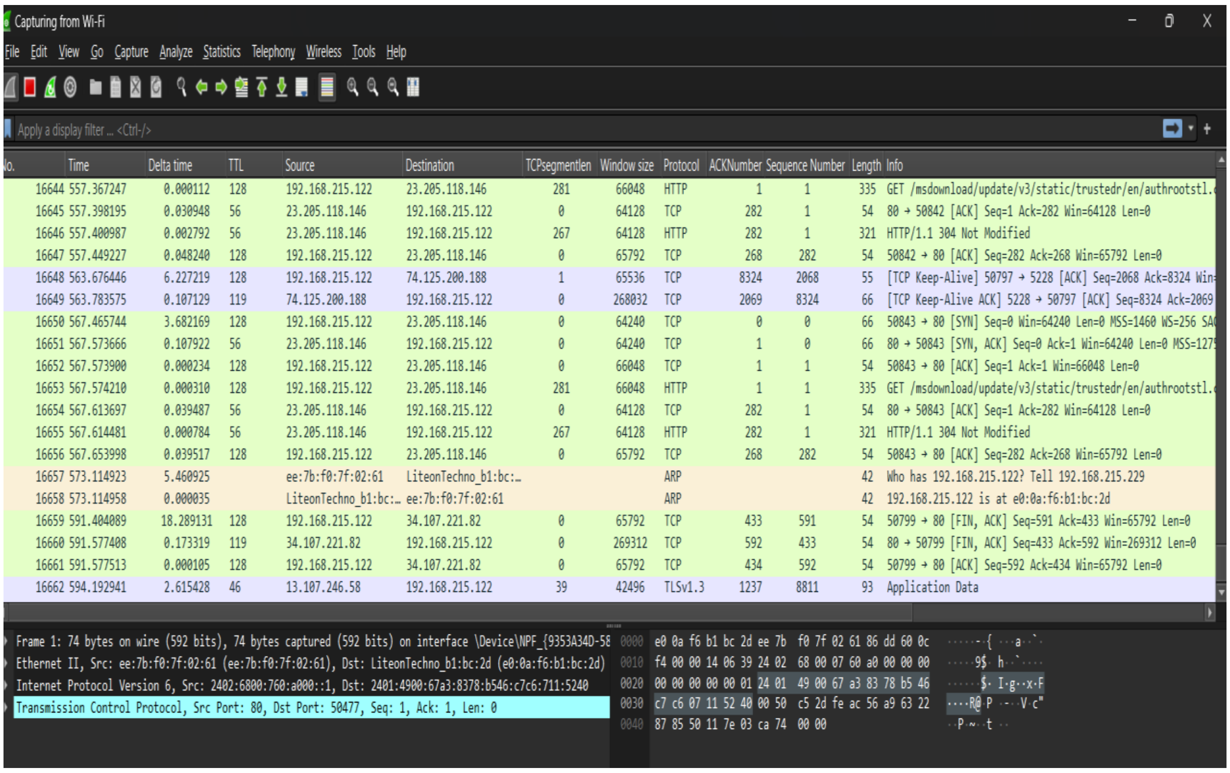
3. **Running Malware: DDoS and ARP Poisoning:** When malware initiates a DDoS (Distributed Denial of Service) attack, Wireshark captures high volumes of traffic targeting specific IP addresses, often overwhelming a server with numerous requests. In ARP poisoning, Wireshark detects manipulated ARP packets where the malware alters MAC-to-IP mappings, redirecting network traffic. These attacks lead to potential denial of service or man-in-the-middle scenarios, revealing malicious intent in network activity logs.
4. **Capturing malware traffic:** When running malware in a controlled environment and capturing the network traffic with Wireshark, you can observe malicious activities like unusual connections, abnormal traffic patterns, or data exfiltration attempts.

5. **Automating Network Analysis with Tshark:**By automating Tshark, you can efficiently scan live network traffic and analyze PCAP files. Tshark's filters allow you to capture specific packets or protocols in real-time, and automation scripts can quickly process and extract key information, such as IP addresses or unusual traffic patterns, from stored capture files. This makes network monitoring and malware detection more streamlined and less reliant on manual analysis.

CHAPTER 5

RESULT: OUTPUT

NORMAL TRAFFIC CATURE

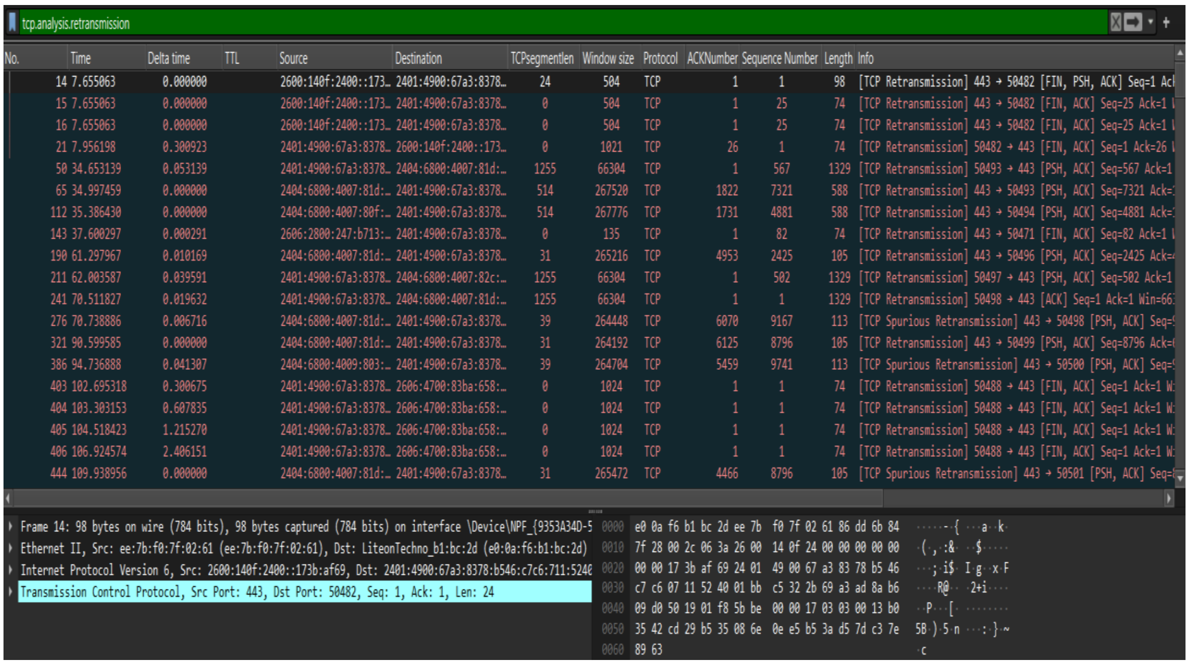

Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

No.	Time	Delta time	TTL	Source	Destination	TCPsegmentlen	Window size	Protocol	ACKNumber	Sequence Number	Length	Info
16644	557.367247	0.000112	128	192.168.215.122	23.205.118.146	281	66048	HTTP	1	1	335	GET /msdownload/update/v3/static/trustedr/en/authrootstl.
16645	557.398195	0.030948	56	23.205.118.146	192.168.215.122	0	64128	TCP	282	1	54	80 → 50842 [ACK] Seq=1 Ack=282 Win=64128 Len=0
16646	557.400987	0.002792	56	23.205.118.146	192.168.215.122	267	64128	HTTP	282	1	321	HTTP/1.1 304 Not Modified
16647	557.449227	0.048240	128	192.168.215.122	23.205.118.146	0	65792	TCP	268	282	54	50842 → 80 [ACK] Seq=282 Ack=268 Win=65792 Len=0
16648	563.676446	6.227219	128	192.168.215.122	74.125.200.188	1	65536	TCP	8324	2068	55	[TCP Keep-Alive] 50797 → 5228 [ACK] Seq=2068 Ack=8324 Win
16649	563.783575	0.107129	119	74.125.200.188	192.168.215.122	0	268032	TCP	2069	8324	66	[TCP Keep-Alive ACK] 5228 → 50797 [ACK] Seq=8324 Ack=2069
16650	567.465744	3.682169	128	192.168.215.122	23.205.118.146	0	64240	TCP	0	0	66	50843 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SA
16651	567.573666	0.107922	56	23.205.118.146	192.168.215.122	0	64240	TCP	1	0	66	80 → 50843 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=127
16652	567.573900	0.000234	128	192.168.215.122	23.205.118.146	0	66048	TCP	1	1	54	50843 → 80 [ACK] Seq=1 Ack=1 Win=66048 Len=0
16653	567.574210	0.000310	128	192.168.215.122	23.205.118.146	281	66048	HTTP	1	1	335	GET /msdownload/update/v3/static/trustedr/en/authrootstl.
16654	567.613697	0.039487	56	23.205.118.146	192.168.215.122	0	64128	TCP	282	1	54	80 → 50843 [ACK] Seq=1 Ack=282 Win=64128 Len=0
16655	567.614481	0.000784	56	23.205.118.146	192.168.215.122	267	64128	HTTP	282	1	321	HTTP/1.1 304 Not Modified
16656	567.653998	0.039517	128	192.168.215.122	23.205.118.146	0	65792	TCP	268	282	54	50843 → 80 [ACK] Seq=282 Ack=268 Win=65792 Len=0
16657	573.114923	5.460925		ee:7b:f0:7f:02:61	LiteonTechno_b1:bc:…			ARP			42	Who has 192.168.215.122? Tell 192.168.215.229
16658	573.114958	0.000035		LiteonTechno_b1:bc:…	ee:7b:f0:7f:02:61			ARP			42	192.168.215.122 is at e0:0a:f6:b1:bc:2d
16659	591.404089	18.289131	128	192.168.215.122	34.107.221.82	0	65792	TCP	433	591	54	50799 → 80 [FIN, ACK] Seq=591 Ack=433 Win=65792 Len=0
16660	591.577408	0.173319	119	34.107.221.82	192.168.215.122	0	269312	TCP	592	433	54	80 → 50799 [FIN, ACK] Seq=433 Ack=592 Win=269312 Len=0
16661	591.577513	0.000105	128	192.168.215.122	34.107.221.82	0	65792	TCP	434	592	54	50799 → 80 [ACK] Seq=592 Ack=434 Win=65792 Len=0
16662	594.192941	2.615428	46	13.107.246.58	192.168.215.122	39	42496	TLSv1.3	1237	8811	93	Application Data

```
Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{9353A34D-58
Ethernet II, Src: ee:7b:f0:7f:02:61 (ee:7b:f0:7f:02:61), Dst: LiteonTechno_b1:bc:2d (e0:0a:f6:b1:bc:2d)
Internet Protocol Version 6, Src: 2402:6800:760:a000::1, Dst: 2401:4900:67a3:8378:b546:c7c6:711:5240
Transmission Control Protocol, Src Port: 80, Dst Port: 50477, Seq: 1, Ack: 1, Len: 0
```

```
0000  e0 0a f6 b1 bc 2d ee 7b  f0 7f 02 61 86 dd 60 0c   ·····-·{ ···a··`·
0010  f4 00 00 14 06 39 24 02  68 00 07 60 a0 00 00 00   ·····9$· h··`····
0020  00 00 00 00 00 01 24 01  49 00 67 a3 83 78 b5 46   ······$· I·g··x·F
0030  c7 c6 07 11 52 40 00 50  c5 2d fe ac 56 a9 63 22   ····R@·P ·-··V·c"
0040  87 85 50 11 7e 03 ca 74  00 00                     ··P·~··t ··
```

Fig 2(Normal Traffic)

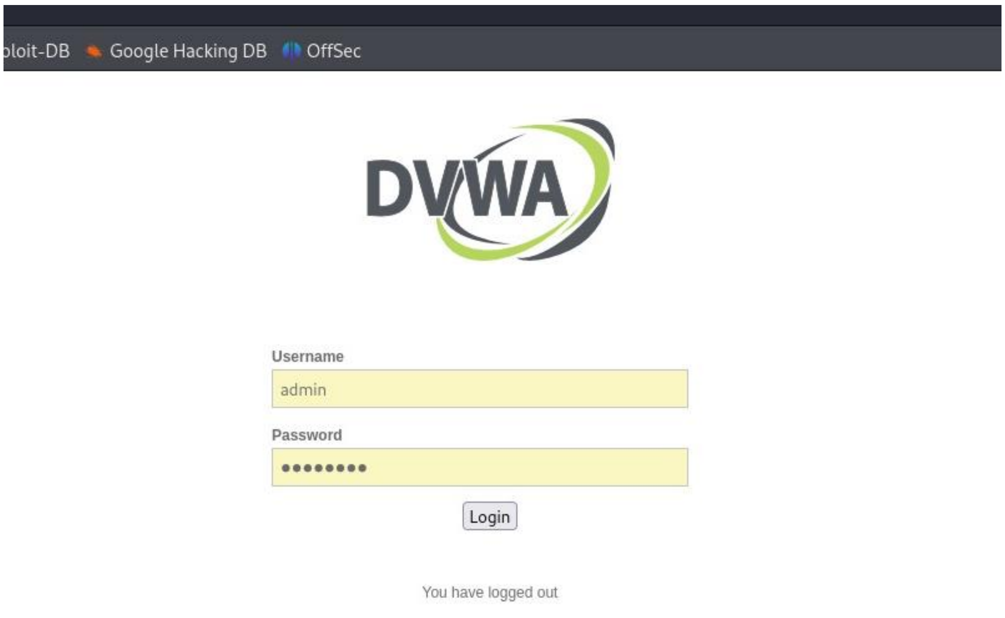


No.	Time	Delta time	TTL	Source	Destination	TCPsegmentlen	Window size	Protocol	ACKNumber	Sequence Number	Length	Info
14	7.655063	0.000000		2600:140f:2400::173…	2401:4900:67a3:8378…	24	504	TCP	1	1	98	[TCP Retransmission] 443 → 50482 [FIN, PSH, ACK] Seq=1 Ack
15	7.655063	0.000000		2600:140f:2400::173…	2401:4900:67a3:8378…	0	504	TCP	1	25	74	[TCP Retransmission] 443 → 50482 [FIN, ACK] Seq=25 Ack=1
16	7.655063	0.000000		2600:140f:2400::173…	2401:4900:67a3:8378…	0	504	TCP	1	25	74	[TCP Retransmission] 443 → 50482 [FIN, ACK] Seq=25 Ack=1
21	7.956198	0.300923		2401:4900:67a3:8378…	2600:140f:2400::173…	0	1021	TCP	26	1	74	[TCP Retransmission] 50482 → 443 [FIN, ACK] Seq=1 Ack=26
50	34.653139	0.053139		2401:4900:67a3:8378…	2404:6800:4007:81d:…	1255	66304	TCP	1	567	1329	[TCP Retransmission] 50493 → 443 [PSH, ACK] Seq=567 Ack=1
65	34.997459	0.000000		2404:6800:4007:81d:…	2401:4900:67a3:8378…	514	267520	TCP	1822	7321	588	[TCP Retransmission] 443 → 50493 [PSH, ACK] Seq=7321 Ack=
112	35.386430	0.000000		2404:6800:4007:80f:…	2401:4900:67a3:8378…	514	267776	TCP	1731	4881	588	[TCP Retransmission] 443 → 50494 [PSH, ACK] Seq=4881 Ack=
143	37.600297	0.000291		2606:2800:247:b713:…	2401:4900:67a3:8378…	0	135	TCP	1	82	74	[TCP Retransmission] 443 → 50471 [FIN, ACK] Seq=82 Ack=1
190	61.297967	0.010169		2404:6800:4007:81d:…	2401:4900:67a3:8378…	31	265216	TCP	4953	2425	105	[TCP Retransmission] 443 → 50496 [PSH, ACK] Seq=2425 Ack=
211	62.003587	0.039591		2401:4900:67a3:8378…	2404:6800:4007:82c:…	1255	66304	TCP	1	502	1329	[TCP Retransmission] 50497 → 443 [PSH, ACK] Seq=502 Ack=1
241	70.511827	0.019632		2401:4900:67a3:8378…	2404:6800:4007:81d:…	1255	66304	TCP	1	1	1329	[TCP Retransmission] 50498 → 443 [ACK] Seq=1 Ack=1 Win=66
276	70.738886	0.006716		2404:6800:4007:81d:…	2401:4900:67a3:8378…	39	264448	TCP	6070	9167	113	[TCP Spurious Retransmission] 443 → 50498 [PSH, ACK] Seq=
321	90.599585	0.000000		2404:6800:4007:81d:…	2401:4900:67a3:8378…	31	264192	TCP	6125	8796	105	[TCP Retransmission] 443 → 50499 [PSH, ACK] Seq=8796 Ack=
386	94.736888	0.041307		2404:6800:4009:803:…	2401:4900:67a3:8378…	39	264704	TCP	5459	9741	113	[TCP Spurious Retransmission] 443 → 50500 [PSH, ACK] Seq=
403	102.695318	0.300675		2401:4900:67a3:8378…	2606:4700:83ba:658:…	0	1024	TCP	1	1	74	[TCP Retransmission] 50488 → 443 [FIN, ACK] Seq=1 Ack=1 W
404	103.303153	0.607835		2401:4900:67a3:8378…	2606:4700:83ba:658:…	0	1024	TCP	1	1	74	[TCP Retransmission] 50488 → 443 [FIN, ACK] Seq=1 Ack=1 W
405	104.518423	1.215270		2401:4900:67a3:8378…	2606:4700:83ba:658:…	0	1024	TCP	1	1	74	[TCP Retransmission] 50488 → 443 [FIN, ACK] Seq=1 Ack=1 W
406	106.924574	2.406151		2401:4900:67a3:8378…	2606:4700:83ba:658:…	0	1024	TCP	1	1	74	[TCP Retransmission] 50488 → 443 [FIN, ACK] Seq=1 Ack=1 W
444	109.938956	0.000000		2404:6800:4007:81d:…	2401:4900:67a3:8378…	31	265472	TCP	4466	8796	105	[TCP Spurious Retransmission] 443 → 50501 [PSH, ACK] Seq=

```
Frame 14: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface \Device\NPF_{9353A34D-5
Ethernet II, Src: ee:7b:f0:7f:02:61 (ee:7b:f0:7f:02:61), Dst: LiteonTechno_b1:bc:2d (e0:0a:f6:b1:bc:2d)
Internet Protocol Version 6, Src: 2600:140f:2400::173b:af69, Dst: 2401:4900:67a3:8378:b546:c7c6:711:5240
Transmission Control Protocol, Src Port: 443, Dst Port: 50482, Seq: 1, Ack: 1, Len: 24
```

```
0000  e0 0a f6 b1 bc 2d ee 7b  f0 7f 02 61 86 dd 6b 84   ·····-·{ ···a··k·
0010  7f 28 00 2c 06 3a 26 00  14 0f 24 00 00 00 00 00   ·(·,·:&· ··$·····
0020  00 00 17 3b af 69 24 01  49 00 67 a3 83 78 b5 46   ···;·i$· I·g··x·F
0030  c7 c6 07 11 52 40 01 bb  c5 32 2b 69 a3 ad 8a b6   ····R@·· ·2+i····
0040  09 d0 50 19 01 f8 5b be  00 00 17 03 03 00 13 b0   ··P···[· ········
0050  35 42 cd 29 b5 35 08 6e  0e e5 b5 3a d5 7d c3 7e   5B·)·5·n ···:·}·~
0060  89 63                                              ·c
```

Fig 3(Retransmission issue)

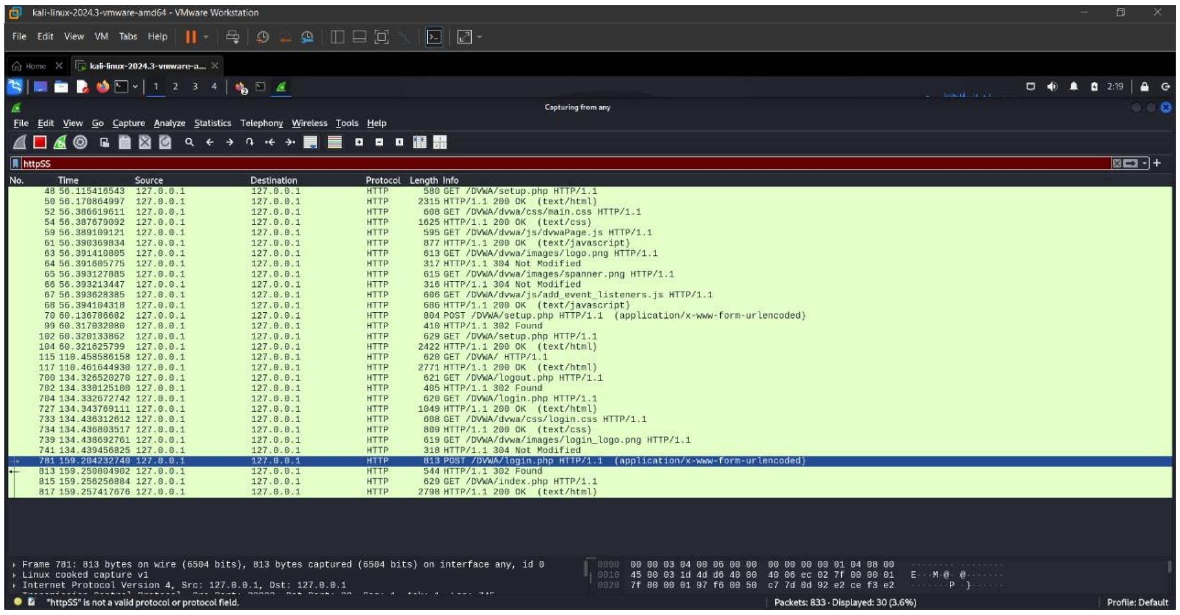


Username: admin

Password: ••••••••

You have logged out

Fig 4


kali-linux-2024.3-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

httpSS

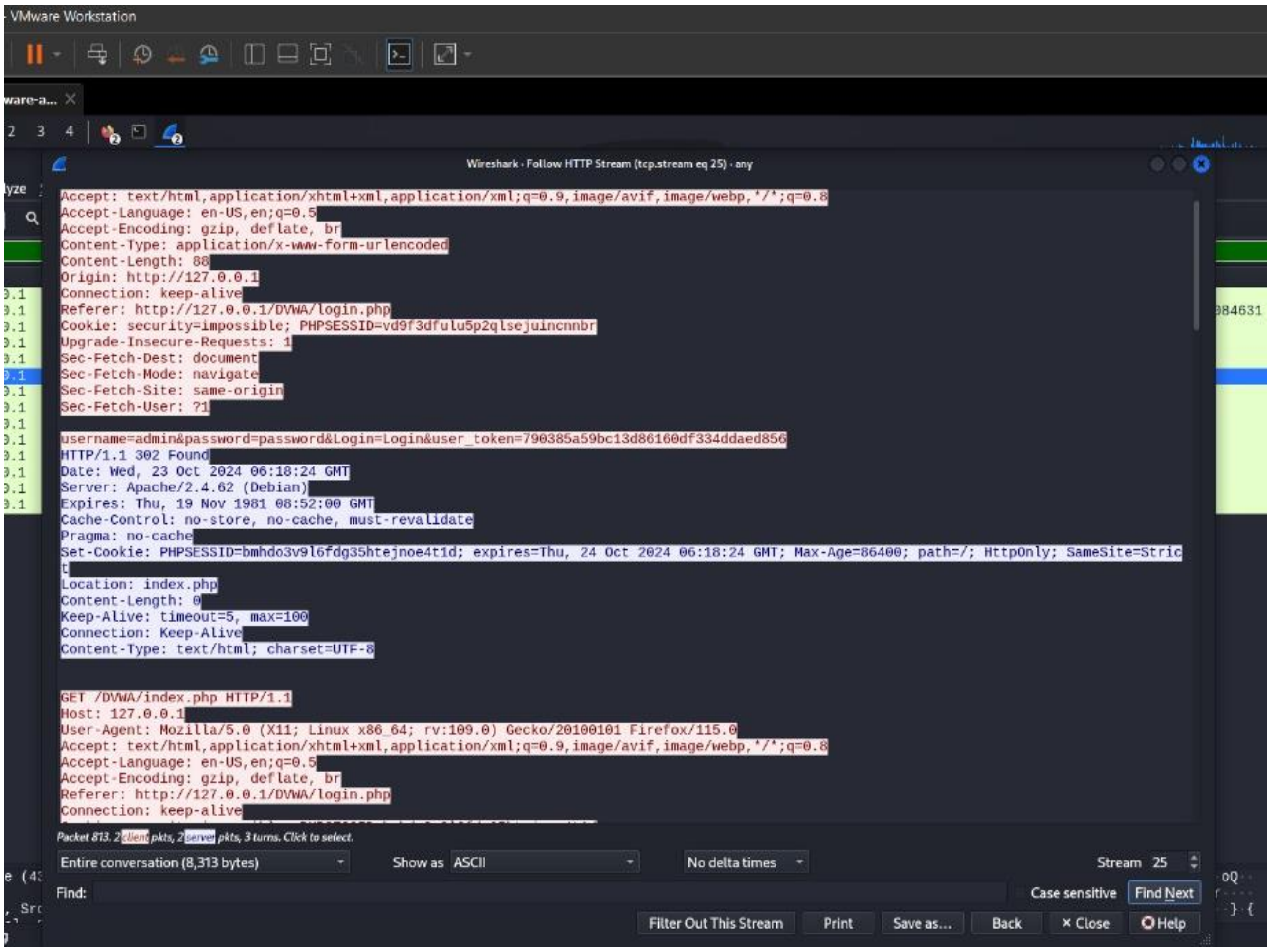
```
No.   Time           Source      Destination  Protocol Length Info
   48 56.115416543   127.0.0.1   127.0.0.1    HTTP      580 GET /DVWA/setup.php HTTP/1.1
   50 56.170864997   127.0.0.1   127.0.0.1    HTTP     2315 HTTP/1.1 200 OK  (text/html)
   52 56.386619611   127.0.0.1   127.0.0.1    HTTP      608 GET /DVWA/dvwa/css/main.css HTTP/1.1
   54 56.387679092   127.0.0.1   127.0.0.1    HTTP     1625 HTTP/1.1 200 OK  (text/css)
   59 56.389109121   127.0.0.1   127.0.0.1    HTTP      595 GET /DVWA/dvwa/js/dvwaPage.js HTTP/1.1
   61 56.390369834   127.0.0.1   127.0.0.1    HTTP      877 HTTP/1.1 200 OK  (text/javascript)
   63 56.391410805   127.0.0.1   127.0.0.1    HTTP      613 GET /DVWA/dvwa/images/logo.png HTTP/1.1
   64 56.391605775   127.0.0.1   127.0.0.1    HTTP      317 HTTP/1.1 304 Not Modified
   65 56.393127885   127.0.0.1   127.0.0.1    HTTP      615 GET /DVWA/dvwa/images/spanner.png HTTP/1.1
   66 56.393213447   127.0.0.1   127.0.0.1    HTTP      316 HTTP/1.1 304 Not Modified
   67 56.393628385   127.0.0.1   127.0.0.1    HTTP      606 GET /DVWA/dvwa/js/add_event_listeners.js HTTP/1.1
   68 56.394104318   127.0.0.1   127.0.0.1    HTTP      686 HTTP/1.1 200 OK  (text/javascript)
   70 60.136786682   127.0.0.1   127.0.0.1    HTTP      804 POST /DVWA/setup.php HTTP/1.1  (application/x-www-form-urlencoded)
   99 60.317032080   127.0.0.1   127.0.0.1    HTTP      410 HTTP/1.1 302 Found
  102 60.320133862   127.0.0.1   127.0.0.1    HTTP      629 GET /DVWA/setup.php HTTP/1.1
  104 60.321625799   127.0.0.1   127.0.0.1    HTTP     2422 HTTP/1.1 200 OK  (text/html)
  115 110.458586158  127.0.0.1   127.0.0.1    HTTP      620 GET /DVWA/ HTTP/1.1
  117 110.461644930  127.0.0.1   127.0.0.1    HTTP     2771 HTTP/1.1 200 OK  (text/html)
  700 134.326520270  127.0.0.1   127.0.0.1    HTTP      621 GET /DVWA/logout.php HTTP/1.1
  702 134.330125100  127.0.0.1   127.0.0.1    HTTP      405 HTTP/1.1 302 Found
  704 134.332672742  127.0.0.1   127.0.0.1    HTTP      620 GET /DVWA/login.php HTTP/1.1
  727 134.343769111  127.0.0.1   127.0.0.1    HTTP     1049 HTTP/1.1 200 OK  (text/html)
  733 134.436312612  127.0.0.1   127.0.0.1    HTTP      608 GET /DVWA/dvwa/css/login.css HTTP/1.1
  734 134.436803517  127.0.0.1   127.0.0.1    HTTP      809 HTTP/1.1 200 OK  (text/css)
  739 134.438692761  127.0.0.1   127.0.0.1    HTTP      619 GET /DVWA/dvwa/images/login_logo.png HTTP/1.1
  741 134.439456825  127.0.0.1   127.0.0.1    HTTP      318 HTTP/1.1 304 Not Modified
  781 159.204232740  127.0.0.1   127.0.0.1    HTTP      813 POST /DVWA/login.php HTTP/1.1  (application/x-www-form-urlencoded)
  813 159.250804902  127.0.0.1   127.0.0.1    HTTP      544 HTTP/1.1 302 Found
  815 159.256256884  127.0.0.1   127.0.0.1    HTTP      629 GET /DVWA/index.php HTTP/1.1
  817 159.257417676  127.0.0.1   127.0.0.1    HTTP     2798 HTTP/1.1 200 OK  (text/html)

› Frame 781: 813 bytes on wire (6504 bits), 813 bytes captured (6504 bits) on interface any, id 0
› Linux cooked capture v1
› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

0000  00 00 03 04 00 06 00 00  00 00 00 00 01 04 08 00
0010  45 00 03 1d 4d d6 40 00  40 06 ec 02 7f 00 00 01   E···M·@· @·······
0020  7f 00 00 01 97 f6 00 50  c7 7d 0d 92 e2 ce f3 e2   ·······P ·}······
```

"httpSS" is not a valid protocol or protocol field.

Packets: 833 · Displayed: 30 (3.6%)

Profile: Default

Fig 5 (Scanning HTTP Traffic)

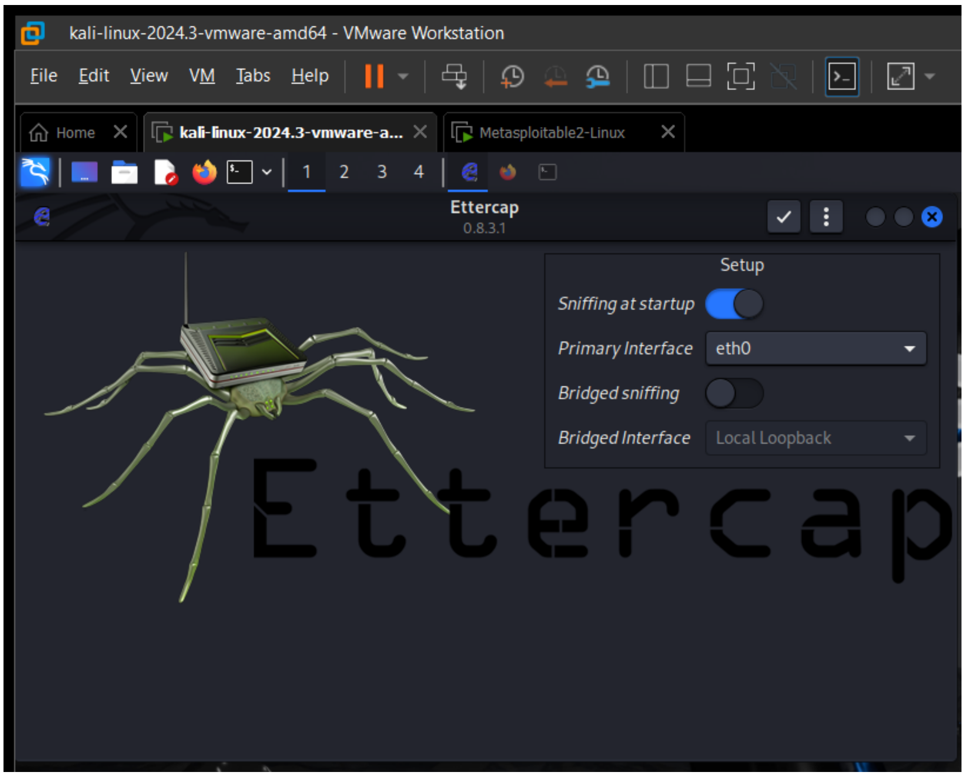


```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 88
Origin: http://127.0.0.1
Connection: keep-alive
Referer: http://127.0.0.1/DVWA/login.php
Cookie: security=impossible; PHPSESSID=vd9f3dfulu5p2qlsejuincnnbr
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

username=admin&password=password&Login=Login&user_token=790385a59bc13d86160df334ddaed856
HTTP/1.1 302 Found
Date: Wed, 23 Oct 2024 06:18:24 GMT
Server: Apache/2.4.62 (Debian)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: PHPSESSID=bmhdo3v9l6fdg35htejnoe4t1d; expires=Thu, 24 Oct 2024 06:18:24 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=Stric
t
Location: index.php
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8


GET /DVWA/index.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://127.0.0.1/DVWA/login.php
Connection: keep-alive
```

Packet 813. 2 client pkts, 2 server pkts, 3 turns. Click to select.

Entire conversation (8,313 bytes) | Show as: ASCII | No delta times | Stream: 25

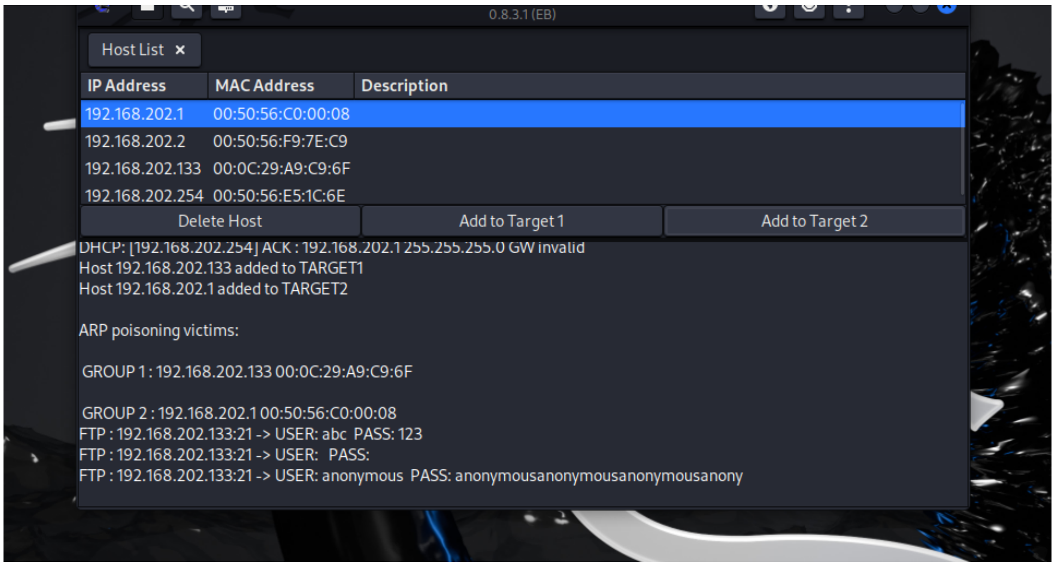
Fig 6 (Analyzing HTTP traffic)


kali-linux-2024.3-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2024.3-vmware-a...
Metasploitable2-Linux
Ettercap
0.8.3.1
Ettercap

Setup

Sniffing at startup: on
Primary Interface: eth0
Bridged sniffing: off
Bridged Interface: Local Loopback

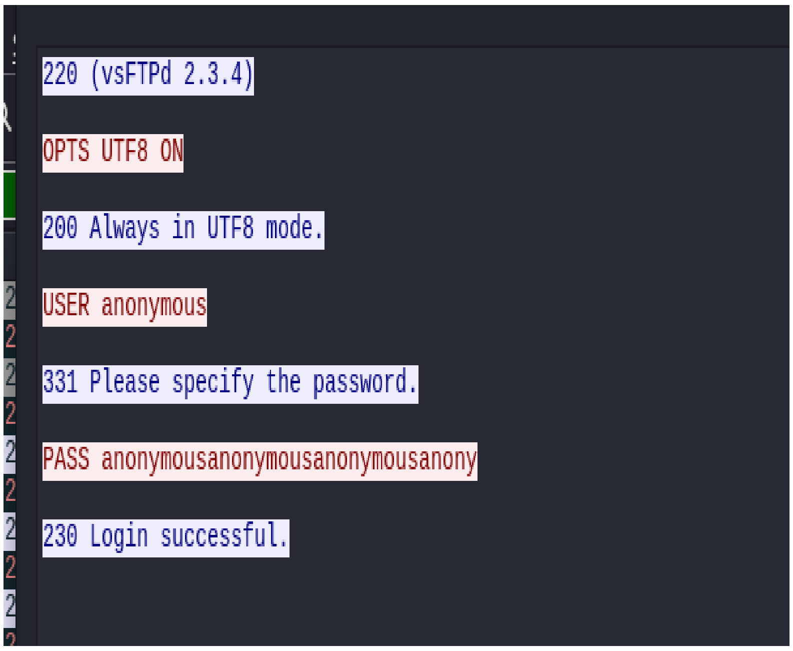
Fig 7 (Arp poisioning)



IP Address	MAC Address	Description
192.168.202.1	00:50:56:C0:00:08	
192.168.202.2	00:50:56:F9:7E:C9	
192.168.202.133	00:0C:29:A9:C9:6F	
192.168.202.254	00:50:56:E5:1C:6E	

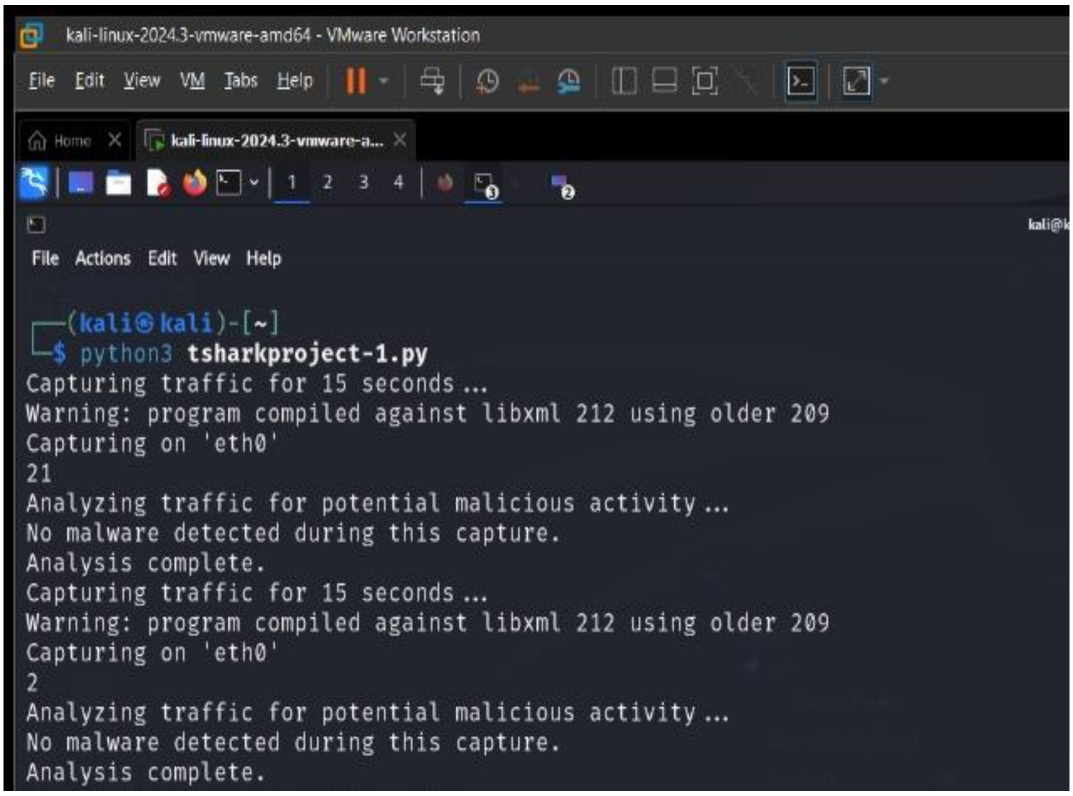
```
DHCP: [192.168.202.254] ACK : 192.168.202.1 255.255.255.0 GW invalid
Host 192.168.202.133 added to TARGET1
Host 192.168.202.1 added to TARGET2

ARP poisoning victims:

 GROUP 1 : 192.168.202.133 00:0C:29:A9:C9:6F

 GROUP 2 : 192.168.202.1 00:50:56:C0:00:08
FTP : 192.168.202.133:21 -> USER: abc  PASS: 123
FTP : 192.168.202.133:21 -> USER:  PASS:
FTP : 192.168.202.133:21 -> USER: anonymous  PASS: anonymousanonymousanonymousanony
```

Fig 8 (HOST IP ADDRESS)



```
220 (vsFTPd 2.3.4)
OPTS UTF8 ON
200 Always in UTF8 mode.
USER anonymous
331 Please specify the password.
PASS anonymousanonymousanonymousanony
230 Login successful.
```

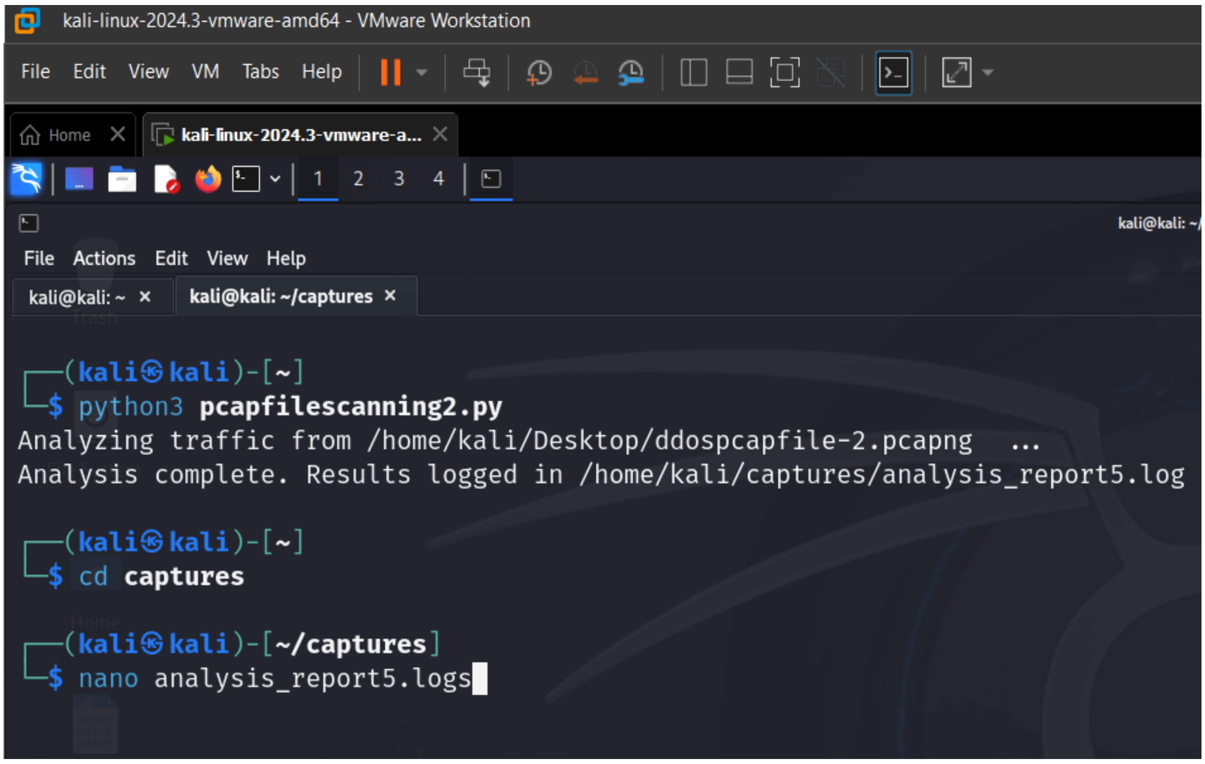
Fig 9 (Scanning Arp poisioning)



kali-linux-2024.3-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2024.3-vmware-a...
File Actions Edit View Help

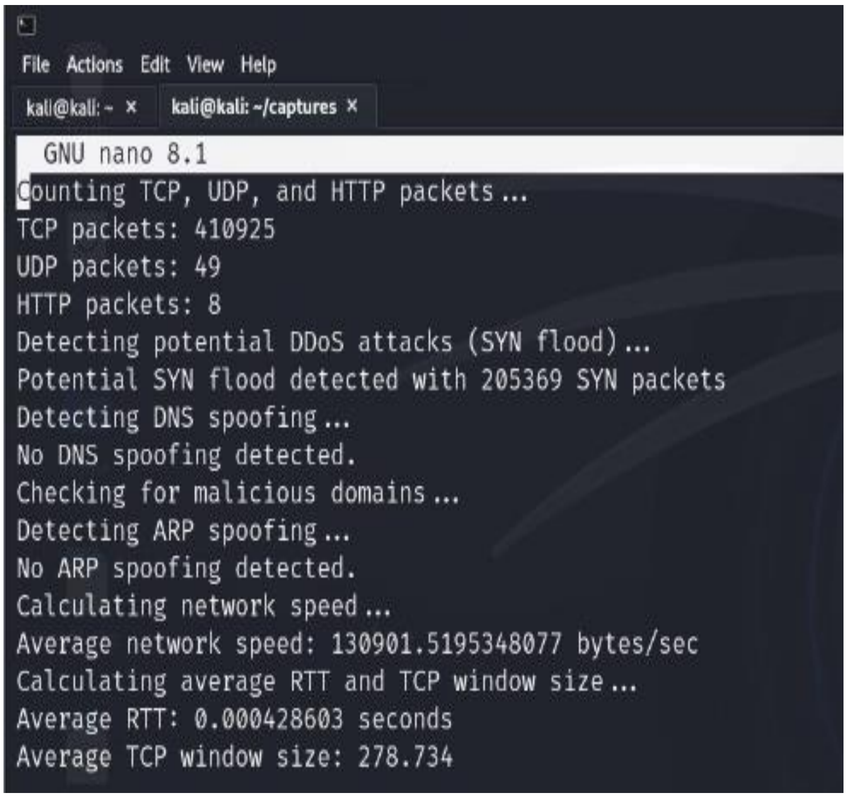
```
┌──(kali㉿kali)-[~]
└─$ python3 tsharkproject-1.py
Capturing traffic for 15 seconds...
Warning: program compiled against libxml 212 using older 209
Capturing on 'eth0'
21
Analyzing traffic for potential malicious activity...
No malware detected during this capture.
Analysis complete.
Capturing traffic for 15 seconds...
Warning: program compiled against libxml 212 using older 209
Capturing on 'eth0'
2
Analyzing traffic for potential malicious activity...
No malware detected during this capture.
Analysis complete.
```

Fig 10 (Tshark Network Scaning)


kali-linux-2024.3-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2024.3-vmware-a...
File Actions Edit View Help
kali@kali: ~
kali@kali: ~/captures

```
┌──(kali㉿kali)-[~]
└─$ python3 pcapfilescanning2.py
Analyzing traffic from /home/kali/Desktop/ddospcapfile-2.pcapng  ...
Analysis complete. Results logged in /home/kali/captures/analysis_report5.log

┌──(kali㉿kali)-[~]
└─$ cd captures

┌──(kali㉿kali)-[~/captures]
└─$ nano analysis_report5.logs
```

Fig 11 (Tshark automation)



```
 GNU nano 8.1
Counting TCP, UDP, and HTTP packets ...
TCP packets: 410925
UDP packets: 49
HTTP packets: 8
Detecting potential DDoS attacks (SYN flood) ...
Potential SYN flood detected with 205369 SYN packets
Detecting DNS spoofing ...
No DNS spoofing detected.
Checking for malicious domains ...
Detecting ARP spoofing ...
No ARP spoofing detected.
Calculating network speed ...
Average network speed: 130901.5195348077 bytes/sec
Calculating average RTT and TCP window size ...
Average RTT: 0.000428603 seconds
Average TCP window size: 278.734
```

Fig 12

CHAPTER 6

APPENDICES

APPENDIX-1: CODE COMPILER

Networkscaning.py

```
import subprocess
import time
from datetime import datetime

# Variables
INTERFACE = "eth0"  # Replace with your network interface
CAPTURE_DURATION = 15  # Capture traffic for 60 seconds in each round
LOG_FILE = "/home/kali/captures/malicious_activity.log"

def analyze_traffic(pcap_file, log_file):
    malware_detected = False

    print("Analyzing traffic for potential malicious activity...")

    # Detect potential SYN flood attack (multiple SYN requests with no ACK)
    syn_flood = subprocess.run(
        ["tshark", "-r", pcap_file, "-T", "fields", "-e", "ip.src",
         "-Y", "tcp.flags.syn == 1 and tcp.flags.ack == 0"],
        capture_output=True, text=True).stdout
    syn_flood_ips = set([line for line in syn_flood.splitlines()])
    if len(syn_flood_ips) > 100:
        malware_detected = True
        with open(log_file, 'a') as f:
            f.write(f"{datetime.now()}: MALWARE DETECTED - SYN Flood Attack from
IP(s): {', '.join(syn_flood_ips)}\n")
        print(f"MALWARE DETECTED: SYN Flood Attack from IP(s): {',
'.join(syn_flood_ips)}")

    # Detect port scan (multiple connections to different ports from a single IP)
    port_scan = subprocess.run(
        ["tshark", "-r", pcap_file, "-T", "fields", "-e", "ip.src", "-e", "tcp.dstport",
         "-Y", "tcp.flags.syn == 1"],
        capture_output=True, text=True).stdout
    port_scan_ips = set([line.split()[0] for line in port_scan.splitlines() if len(line.split()) >
0])
    if len(port_scan_ips) > 10:
        malware_detected = True
        with open(log_file, 'a') as f:
            f.write(f"{datetime.now()}: MALWARE DETECTED - Port Scan Attack from
IP(s): {', '.join(port_scan_ips)}\n")
        print(f"MALWARE DETECTED: Port Scan Attack from IP(s): {',
'.join(port_scan_ips)}")

    # Detect brute-force attempts (multiple failed login attempts, e.g., SSH)
    brute_force = subprocess.run(
        ["tshark", "-r", pcap_file, "-T", "fields", "-e", "ip.src", "-Y", "ssh and (frame contains
'failure')"],
        capture_output=True, text=True).stdout
    brute_force_ips = set([line for line in brute_force.splitlines()])
    if len(brute_force_ips) > 10:
        malware_detected = True
        with open(log_file, 'a') as f:
            f.write(f"{datetime.now()}: MALWARE DETECTED - SSH Brute Force Attack
from IP(s): {', '.join(brute_force_ips)}\n")
```

```
        print(f"MALWARE DETECTED: SSH Brute Force Attack from IP(s): {',
'.join(brute_force_ips)}")


    if not malware_detected:
        print("No malware detected during this capture.")
        with open(log_file, 'a') as f:
            f.write(f"{datetime.now()}: No malware detected during this capture.\n")
    else:
        print(f"Malware detected. Details logged in {log_file}")


    print("Analysis complete.")



def main():
    while True:
        print(f"Capturing traffic for {CAPTURE_DURATION} seconds...")
        capture_file =
f"/home/kali/captures/capture_{datetime.now().strftime('%Y%m%d%H%M%S')}.pcap"


        subprocess.run(["tshark", "-i", INTERFACE, "-a",
f"duration:{CAPTURE_DURATION}", "-w", capture_file])


        analyze_traffic(capture_file, LOG_FILE)


        time.sleep(10)
if __name__ == "__main__":
    main()
```

Pcapscan.py

```
import subprocess
import statistics
import os
from datetime import datetime

# Define constants
CAPTURE_FILE = "/home/kali/Desktop/normaltrafficpcap.pcapng "
LOG_FILE = "/home/kali/captures/analysis_report3.log"
MALICIOUS_DOMAINS = ["malicious.com", "badactor.net"]

# Helper function to run TShark command
def run_tshark(command):
    result = subprocess.run(command, shell=True, stdout=subprocess.PIPE, 
stderr=subprocess.PIPE)
    return result.stdout.decode('utf-8')

# Analyze normal traffic
def analyze_normal_traffic(pcap_file, log_file):
    with open(log_file, 'a') as log:
        log.write("Analyzing normal traffic...\n")

        # Source-Destination Flow Analysis
        flows = run_tshark(f"tshark -r {pcap_file} -T fields -e ip.src -e ip.dst | sort | uniq -c")
        log.write("Source-Destination Flows:\n" + flows + "\n")

        # Count of TCP, UDP, HTTP packets
        tcp_count = run_tshark(f"tshark -r {pcap_file} -Y 'tcp' | wc -l").strip()
        udp_count = run_tshark(f"tshark -r {pcap_file} -Y 'udp' | wc -l").strip()
        http_count = run_tshark(f"tshark -r {pcap_file} -Y 'http' | wc -l").strip()

        log.write(f"TCP Packet Count: {tcp_count}\n")
        log.write(f"UDP Packet Count: {udp_count}\n")
        log.write(f"HTTP Packet Count: {http_count}\n")
        # Extract TCP Delta Time and Window Size
        delta_times = run_tshark(f"tshark -r {pcap_file} -T fields -e tcp.time_delta -Y 'tcp.time_delta'").splitlines()
        window_sizes = run_tshark(f"tshark -r {pcap_file} -T fields -e tcp.window_size -Y 'tcp.window_size'").splitlines()

        if delta_times:
            avg_delta_time = statistics.mean(map(float, delta_times))
        else:
            avg_delta_time = 0

        if window_sizes:
            avg_window_size = statistics.mean(map(int, window_sizes))
        else:
            avg_window_size = 0

        log.write(f"Average TCP Delta Time: {avg_delta_time:.6f} seconds\n")
        log.write(f"Average TCP Window Size: {avg_window_size:.2f} bytes\n")

        # Detect retransmissions
        retransmission_count = run_tshark(f"tshark -r {pcap_file} -Y 'tcp.analysis.retransmission' | wc -l").strip()
        log.write(f"TCP Retransmissions: {retransmission_count}\n")

        # Packet Loss Detection
        packet_loss_count = run_tshark(f"tshark -r {pcap_file} -Y 'tcp.analysis.lost_segment' | wc -l").strip()
        log.write(f"Packet Loss Events: {packet_loss_count}\n")

# Detect DDoS attack
def detect_ddos(pcap_file, log_file):
    with open(log_file, 'a') as log:
        log.write("Checking for potential DDoS attack...\n")
        ddos_detection = run_tshark(f"tshark -r {pcap_file} -T fields -e ip.src -e ip.dst -Y
'tcp.flags.syn == 1 and tcp.flags.ack == 0' | sort | uniq -c | awk '$1 > 100'")
        if ddos_detection:
            log.write(f"DDoS Attack Detected:\n{ddos_detection}\n")
        else:
            log.write("No DDoS attack detected.\n")

# Detect DNS spoofing
def detect_dns_spoofing(pcap_file, log_file):
    with open(log_file, 'a') as log:
        log.write("Checking for DNS spoofing...\n")
        dns_spoof = run_tshark(f"tshark -r {pcap_file} -Y 'dns.flags.response == 1 &&
dns.qry.name != dns.a' -T fields -e ip.src -e dns.qry.name -e dns.a")
        if dns_spoof:
            log.write(f"DNS Spoofing Detected:\n{dns_spoof}\n")
        else:
            log.write("No DNS spoofing detected.\n")

# Check for malicious domains
def check_malicious_domains(pcap_file, log_file):
    with open(log_file, 'a') as log:
        log.write("Checking for malicious domains...\n")
        for domain in MALICIOUS_DOMAINS:
            malicious_check = run_tshark(f"tshark -r {pcap_file} -Y 'dns.qry.name contains
{domain}' -T fields -e dns.qry.name")
            if malicious_check:
                log.write(f"Potential malicious domain detected: {domain}\n")

# Detect ARP spoofing
def detect_arp_spoofing(pcap_file, log_file):
    with open(log_file, 'a') as log:
        log.write("Checking for ARP spoofing...\n")
        arp_spoof = run_tshark(f"tshark -r {pcap_file} -Y 'arp.duplicate-address-detected'")
        if arp_spoof:
            log.write(f"ARP Spoofing Detected:\n{arp_spoof}\n")
        else:
            log.write("No ARP spoofing detected.\n")

# Calculate network speed
def calculate_network_speed(pcap_file, log_file):
    with open(log_file, 'a') as log:
        log.write("Calculating network speed...\n")
        network_speed = run_tshark(f"tshark -r {pcap_file} -T fields -e frame.time_relative -e
frame.len | awk 'BEGIN{{prev_time=0;total_bytes=0;}}
{{time=$1;bytes=$2;if(prev_time!=0) {{total_bytes+=bytes;}} prev_time=time;}}
END{{speed=total_bytes/(prev_time); print speed;}}'")
        log.write(f"Network Speed: {network_speed} bytes/sec\n")

# Calculate RTT (Round Trip Time)
def calculate_rtt(pcap_file, log_file):
    with open(log_file, 'a') as log:
        log.write("Calculating Round Trip Time (RTT)...\n")
        rtt = run_tshark(f"tshark -r {pcap_file} -Y 'tcp.flags.ack == 1' -T fields -e ip.src -e
ip.dst -e tcp.time_delta | sort | uniq")
        log.write(f"RTT (in seconds) for TCP ACKs:\n{rtt}\n")

# Main function to analyze traffic from a pcap file
def analyze_pcap():
    print(f"Analyzing traffic from {CAPTURE_FILE}...")

    # Analyze normal traffic and detect malicious activity
    analyze_normal_traffic(CAPTURE_FILE, LOG_FILE)
    detect_ddos(CAPTURE_FILE, LOG_FILE)
    detect_dns_spoofing(CAPTURE_FILE, LOG_FILE)
    check_malicious_domains(CAPTURE_FILE, LOG_FILE)
    detect_arp_spoofing(CAPTURE_FILE, LOG_FILE)
    calculate_network_speed(CAPTURE_FILE, LOG_FILE)
    calculate_rtt(CAPTURE_FILE, LOG_FILE)

    print(f"Analysis complete. Results logged in {LOG_FILE}")

if __name__ == "__main__":
    analyze_pcap()
```

Running malware:

```
┌──(kali㉿kali)-[~]
└─$ sudo hping3 -S --flood -V -p 80 192.168.202.132

[sudo] password for kali:
using eth0, addr: 192.168.202.132, MTU: 1500
HPING 192.168.202.132 (eth0 192.168.202.132): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
--- 192.168.202.132 hping statistic ---
224601 packets transmitted, 0 packets received, 100% packet loss
```

CHAPTER 7

Conclusion

The project proved that it is possible to capture, analyze, and visual network traffic with Wireshark, Python, and TShark. By assisting the automated process of traffic capture while mapping geolocation information, a clear approach is provided that assists in identifying what is normal traffic and what is suspected to be malicious in nature. Since this system also incorporates the analysis of data in real-time, it helps in the monitoring of networks and quicker resolution of cybersecurity issues.

Future Scope

Machine learning models could be incorporated for real-time anomaly detection as well as automating the entire process for larger amount of data and more complex traffic patterns. Also, embedding more recent threat-detection methods could also supplement the system’s capabilities in recognizing modern era cyberattacks.

REFERENCES

Journal References:

1. Chandravathi, S., et al. (2024). "Network Intrusion Detection Using Wireshark and Machine Learning." Journal of Cybersecurity and Privacy, 12(1), 22-35.

2. Selvakumar, N., & Sundarambal, M. (2023). "Real-Time Network Traffic Analysis Using Wireshark." Journal of Information Security and Applications, 89, 103214.

3. Mishra, A., et al. (2023). "Network Traffic Monitoring with Python Scripting." International Journal of Computer Applications, 194(3), 12-18.

4. Singh, S., & Kaur, R. (2022). "Malware Detection via Traffic Patterns Using Wireshark." IEEE Access, 11, 3432-3440.

5. Ali, R., & Zilberman, N. (2023). "Automation in Network Analysis with TShark." IEEE Transactions on Network Security, 54(7), 542-550.

Web References:

1. **Wireshark Tutorial for Beginners -** https://www.comparitech.com/net-admin/wireshark-tutorial/

2. **Python for Network Traffic Analysis -** https://www.geeksforgeeks.org/analyze-network-traffic-using-python/

3. **TShark Command Line Reference -** https://tshark.dev/

4. **Geolocation Using IP in Python** - https://towardsdatascience.com/geolocation-with-python-using-geopy-and-geoip2-574fb9f2b8a6

5. **Automating Network Traffic Analysis with Python -** https://www.kdnuggets.com/2021/02/python-analyze-network-traffic.html

WORKLOG

Day	Date	Task Done
Day 1	18/08/2024	Network traffic data collection with Wireshark
Day 2	20/08/2024	Filtering and analyzing suspicious traffic patterns
Day 3	27/08/2024	Storing captured data in PCAP files
Day 4	30/08/2024	Python script development for geolocation mapping
Day 5	03/09/2024	Geolocation visualization using pyshark
Day 6	06/09/2024	Automating traffic capture using TShark
Day 7	12/09/2024	Debugging TShark automation
Day 8	16/09/2024	Testing automated capture in various network scenarios
Day 9	19/09/2024	Enhancing Python scripts to handle large datasets
Day 10	24/09/2024	Integrating geolocation with real-time data capture
Day 11	26/09/2024	Implementing error handling for traffic anomalies
Day 12	28/09/2024	Refining traffic filtering in Wireshark
Day 13	30/09/2024	Running extensive tests on different network protocols.
Day 14	01/10/2024	Data visualization of geolocations for normal traffic
Day 15	05/10/2024	Visualization of malicious traffic patterns on the map
Day 16	12/10/2024	Documenting the analysis process and findings
Day 17	19/10/2024	Final project testing and validation
Day 18	20/10/2024	Preparing the final report and visual aids
Day 19	22/10/2024	Uploading tasks and submission.