A Project Report on

Attack and Traffic Generator (ATG) for Cyber Drill Events

Submitted to

CDAC, Noida

Guide:
Mrs. Rekha Saraswat

Submitted by:
Arnav Raghav Sharma
Project Trainee,
(June-August) 2024

Centre for Development of Advanced Computing,

Noida, UP

CERTIFICATE

This is to certify that this project report entitled ‘Attack and Traffic Generator (ATG) for Cyber Drill Events’ submitted to CDAC Noida, is a Bonafide record of work done by Arnav Raghav Sharma under my supervision from 12.06.2024 to 12.08.2024.

Signature of the Guide

Guide: Mrs. Rekha Saraswat

Designation: Joint Director

Address: CDAC, Noida

Declaration by Author

This is to declare that this report has been written by me. No part of the report is plagiarized from other sources. All information included from other sources has been duly acknowledged. I aver that if any part of the report is found to be plagiarized, I shall take full responsibility for it.

Name of Author:______________________

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to everyone who supported and guided me throughout my internship at CDAC, Noida.

Firstly, I would like to extend my heartfelt gratitude and regards to my supervisor, Mrs. Rekha Saraswat, Joint Director, for her invaluable guidance, suggestions, encouragement, and endless support. Her expertise and insights greatly contributed to the successful completion of this project.

I am also immensely grateful to my mentor, Jyoti Pathak Ma'am, for her continuous assistance and mentorship in the lab. His knowledge and advice were crucial in helping me to navigate through the challenges of the project.

I am highly grateful to Dr. Arti Noor madam, Sr. Director of the E & T Department, for providing me with this opportunity and for her support throughout the internship. The experience gained in the E & T Department has been invaluable and will significantly contribute to my professional growth.

I am also thankful to Mr. V. K. Sharma sir, Chairman of the E & T Department for the whole process of internship in C-DAC. The internship programme would not have culminated without his support, co-operation, and advice during and before the course.

I would also like to thank Mr. Raja Deb, CDAC Noida, for his generous help and support on numerous occasions. His willingness to assist was greatly appreciated.

I would also like to express my gratitude to my parents for their constant encouragement and support. Their belief in me has always been a source of motivation.

Lastly, I extend my thanks to the entire team at CDAC, Noida, for creating a welcoming and conducive environment for learning and growth. The collective efforts and support of everyone at CDAC made this internship a memorable and rewarding experience.

Arnav Raghav Sharma

Date: 08.08.2024

TABLE OF CONTENTS

Attack and Traffic Generator (ATG) for Cyber Drill Events

1. INTRODUCTION OF THE PROJECT

1.1 About the Institute

The Centre for Development of Advanced Computing (C-DAC) is the premier Research & Development (R&D) organization of the Ministry of Electronics and Information Technology (MeitY) for carrying out R&D in IT, Electronics, and associated areas. Established in 1988, C-DAC has been at the forefront of cutting-edge technology, driving the national mission for software and hardware development in India.

CDAC, Noida is one of the key centers of C-DAC, known for its significant contributions to the advancement of information technology and related fields. Situated in the vibrant technological hub of Noida, this center has become a beacon of innovation and excellence in the IT landscape of India.

The mission of CDAC Noida is to become a center of excellence in IT and electronics by delivering state-of-the-art technologies and solutions. It aims to address the needs of various sectors, including government, industry, and academia, through its research and development initiatives. The vision is to empower India in the digital domain, fostering an environment where technological advancements drive socio-economic development.

CDAC Noida specializes in a wide range of research areas, including high-performance computing, grid computing, multilingual computing, professional electronics, VLSI and embedded systems, and software technologies. The center is renowned for its work in e-Governance, healthcare, cybersecurity, and digital forensics. The multidisciplinary approach ensures that CDAC Noida remains at the cutting edge of technology, continually pushing the boundaries of what is possible.

In addition to its R&D prowess, CDAC Noida is a hub for training and education in the field of advanced computing. The center offers a variety of specialized courses and training programs aimed at enhancing the skills of IT professionals and students. These programs are designed to keep pace with the rapidly evolving technological landscape, ensuring that participants are well-equipped to meet contemporary challenges.

CDAC Noida has been involved in numerous high-impact projects that have contributed significantly to India's technological capabilities. Some notable projects include the development of the PARAM series of supercomputers, which have placed India among the top nations in supercomputing. The center also engages in various collaborative efforts with international and national institutions, fostering a culture of knowledge exchange and innovation.

The state-of-the-art infrastructure at CDAC Noida supports its extensive R&D activities. The center is equipped with advanced laboratories, high-performance computing facilities, and sophisticated testing and development environments. This robust infrastructure enables researchers and developers to undertake complex projects and deliver high-quality solutions.

CDAC Noida has a profound impact on the local and national community. By driving technological innovation, the center contributes to the economic and social development of the region. Its initiatives in e-Governance have streamlined various government processes, making them more efficient and accessible to the public. Furthermore, the center's focus on cybersecurity and digital forensics plays a crucial role in safeguarding the nation's digital infrastructure.

In conclusion, CDAC Noida stands as a testament to India's capabilities in advanced computing and IT. Its commitment to research, development, and education has made it a pivotal player in the technological advancements of the country. As CDAC Noida continues to innovate and expand its horizons, it remains dedicated to its mission of empowering India through technology.

1.2 About the Project

The project, titled "Attack and Traffic Generator (ATG) for Cyber Drill Events," focuses on simulating attack traffic to enhance cybersecurity training and preparedness.

Attack traffic generator involves simulating network activities that mimic malicious behaviors observed during cyber-attacks. It encompasses generating various traffic patterns such as Distributed Denial of Service (DDoS), malware propagation, phishing attempts, and data exfiltration. The objective is to replicate real-world scenarios to accurately assess the resilience and responsiveness of cybersecurity defenses.

The objectives of Attack Traffic Generator can be summed up as follows:

1. Develop Simulation Tools: Create sophisticated tools capable of generating realistic attack traffic to simulate diverse cyber threats.
2. Enhance Training Effectiveness: Provide cybersecurity professionals with practical training scenarios to improve detection and response skills.
3. Support Preparedness: Assist organizations in preparing for cyber incidents by testing and validating their security measures.

The uses and benefits of Attack Traffic Generator in present technological scenario can be summed up as follows:

1. Training Aid: Facilitate hands-on training for cybersecurity teams to practice identifying and mitigating cyber threats.
2. System Validation: Test cybersecurity systems and infrastructure by simulating diverse attack scenarios.
3. Risk Mitigation: Identify vulnerabilities and weaknesses proactively to minimize the risk of actual cyber-attacks.
4. Standardized Training: Ensure consistent and uniform training across teams to achieve cohesive incident response capabilities.

In summary, "Development of Attack Traffic Generator (ATG) for Cyber Drill Events" addresses critical needs in cybersecurity training. By enabling realistic attack simulations and robust training scenarios, the project enhances cybersecurity readiness and resilience. It supports organizations in fortifying their defenses against evolving cyber threats, ultimately contributing to a safer digital landscape.

2. TOOLS & TECHNOLOGIES USED

2.1 Operating System for Security Testing

- **Kali Linux:** Kali Linux is a specialized, Debian-based Linux distribution designed for digital forensics, penetration testing, and cybersecurity. Developed and maintained by Offensive Security, Kali Linux comes pre-installed with a comprehensive suite of tools tailored for various information security tasks, including network analysis, vulnerability assessment, exploitation, forensics, and reverse engineering. It is widely used by ethical hackers, security professionals, and researchers due to its vast repository of over 600 tools, such as Nmap, Wireshark, Metasploit, and Burp Suite.

  Kali Linux is designed to be a highly versatile and customizable platform, offering a wide range of installation options, including running as a live system, on virtual machines, or as a persistent installation on a hard drive. The distribution also supports ARM devices and has a variant known as Kali NetHunter, which is designed specifically for mobile devices.

  The operating system is regularly updated to include the latest tools and features, making it a reliable and up-to-date resource for security testing and research. Kali Linux's emphasis on user accessibility and security, combined with extensive community support and detailed documentation, makes it an indispensable tool for anyone working in the field of cybersecurity.

2.2 Packet Crafting and Manipulation

- **Scapy:** Scapy is an advanced Python-based interactive packet manipulation program and library that enables users to craft, send, receive, and dissect network packets. It supports a wide range of network protocols, making it a versatile tool for tasks such as network scanning, tracerouting, probing, attacks, and network discovery. Scapy is particularly valuable for security testing and research, as it allows for the creation of custom packets and the manipulation of network traffic, enabling users to simulate a variety of attack scenarios, test firewall rules, and analyse network behaviour.

  In addition to its packet crafting capabilities, Scapy can also handle tasks such as sniffing, injection, and fuzzing. It can read packets from a file, send them over the wire, match requests and replies, and more. This flexibility makes Scapy a go-to tool for penetration testers, network engineers, and cybersecurity researchers.

  Scapy also integrates with other tools and libraries, providing additional functionality, such as plotting graphs or generating network maps. Its extensive documentation and active community support make it accessible even to those who are relatively new to network programming and security.

- **Hping:** Hping is a powerful command-line tool used for crafting and analyzing custom TCP/IP packets, making it essential for network security testing and performance analysis. It supports multiple protocols (TCP, UDP, ICMP) and allows precise control over packet attributes, enabling tasks like network scanning, firewall testing, DoS simulation, and vulnerability assessment. Security professionals use Hping to probe defenses, simulate attacks, and diagnose network issues, making it a versatile tool in network security and troubleshooting.

2.3 Network Analysis and Monitoring

- **Wireshark:** Wireshark is a powerful, open-source network protocol analyser that enables users to capture and interactively browse the traffic running on a computer network. Widely used by network administrators, security analysts, and developers, Wireshark provides deep inspection of hundreds of protocols, with the ability to capture live data from a network interface or read packets from a previously saved file. Its user-friendly interface displays packet data in detail, making it an essential tool for troubleshooting network issues, performing security audits, and analysing network performance.

  Wireshark offers advanced features such as filtering traffic, searching within packets, reconstructing TCP streams, and decrypting protocols like SSL/TLS, when keys are available. Its ability to dissect and analyse packets down to the individual field level allows for a thorough examination of network traffic, making it invaluable for detecting anomalies, identifying security threats, and understanding complex network behaviors.

  Wireshark is compatible with a wide range of operating systems and supports a variety of capture file formats. The tool also integrates with other network tools, offering enhanced functionality for tasks like network visualization and traffic generation. Its comprehensive documentation and active community support ensure that both novice and experienced users can effectively utilize Wireshark for their network analysis needs.

2.4 Performance and Load Testing

- **Apache JMeter:** Apache JMeter is an open-source, Java-based application designed primarily for performance testing and load testing of web applications. It allows users to simulate a heavy load on servers, networks, or objects to test their strength and analyze overall performance under different load types. Originally developed to test web applications, JMeter has evolved into a versatile tool that supports various protocols, including HTTP, HTTPS, FTP, JDBC, SOAP, REST, and more.

  JMeter enables testers to create comprehensive test plans using a graphical user interface (GUI), where they can define the number of users, the type of requests, and the timing of those requests. It also offers scripting capabilities for more complex test scenarios. The tool provides robust reporting features, including graphical charts, tables, and logs, allowing users to analyze test results in detail and identify potential bottlenecks or performance issues.

JMeter is widely used in the software development lifecycle to ensure that applications can handle anticipated traffic volumes before going live. Its open-source nature, combined with an active community and extensive documentation, makes it a popular choice for both beginners and experienced testers in performance and load testing.

2.5 Penetration Testing and Exploitation

- **Metasploit:** Metasploit is a powerful, open-source penetration testing framework widely used for security research, vulnerability assessment, and exploitation. Developed by Rapid7, Metasploit allows security professionals to discover, validate, and exploit vulnerabilities in various systems, networks, and applications.

  The framework includes a vast collection of exploits, payloads, and auxiliary modules that can be used to automate the process of testing and attacking systems. Metasploit's modular architecture enables users to customize attacks, choose specific payloads, and conduct post-exploitation activities such as privilege escalation, data extraction, and network pivoting.

  Metasploit supports a range of operating systems, including Windows, Linux, macOS, and more, making it versatile for different environments. It integrates well with other security tools, such as Nmap for network scanning and Wireshark for traffic analysis, enhancing its capability in comprehensive penetration testing.

  Security professionals and ethical hackers use Metasploit to simulate real-world attacks, helping organizations identify vulnerabilities and strengthen their defenses before malicious actors can exploit them. Its extensive community support and regularly updated database of exploits ensure that Metasploit remains a critical tool in the field of cybersecurity.

2.6 Network Scanning and Vulnerability Detection

- **Nmap:** Nmap, or Network Mapper, is a widely recognized and utilized network scanning tool that plays a crucial role in discovering hosts and services within a network. By sending specially crafted packets and analyzing the responses, Nmap allows users to gather detailed information about the network's topology, the services running on each host, and potential vulnerabilities.

  Nmap supports a broad range of scan types, including TCP, UDP, SYN, and ping sweeps, enabling it to perform in-depth assessments. It can identify open ports, detect running services, and even determine the operating systems of the devices within the network. This makes Nmap invaluable for network security auditing, vulnerability detection, and comprehensive mapping of network infrastructure.

  Due to its versatility and powerful capabilities, Nmap is a key tool for security professionals, network administrators, and ethical hackers. It provides critical insights into the security posture of a network, helping to identify potential vulnerabilities before they can be exploited by malicious actors. Nmap's adaptability and extensive feature set make it a staple in the toolkit of anyone involved in network security and analysis.

3. PURPOSE AND IMPLEMENTATION

The purpose of these scripts is to facilitate comprehensive network traffic generation and analysis for cybersecurity research and educational purposes. By utilizing a combination of Nmap and nping, these scripts enable the automated generation of diverse types of network traffic, such as TCP, ICMP, and SYN packets. This automation is critical for simulating real-world network conditions, testing network defences, and understanding how different types of traffic interact within a network.

A key feature of these scripts is the incorporation of IP spoofing, which allows the user to simulate traffic from a false or masked IP address. This is particularly valuable for studying the behaviour of network devices and security measures when faced with spoofed traffic, which is a common tactic used in various cyber-attacks.

Furthermore, these scripts are designed to work seamlessly with Wireshark, a powerful network protocol analyser, enabling the capture and detailed examination of the generated traffic. By capturing traffic with Wireshark, users can analyse packet-level details, gain insights into network performance, and detect potential vulnerabilities or anomalies.

Overall, these scripts are essential tools for anyone studying network security, providing hands-on experience with traffic generation, IP spoofing, and traffic analysis, all of which are crucial for understanding and mitigating cybersecurity threats.

4. SCRIPT DESCRIPTIONS AND WIRESHARK INSIGHTS

This section presents a comprehensive analysis of each script developed as part of this project. It outlines their specific purposes, key functionalities, and practical usage scenarios. Additionally, it provides insights into capturing and analyzing the generated network traffic using Wireshark. These scripts are integral to simulating various network traffic scenarios, essential for testing and evaluating network performance, security protocols, and system resilience.

4.1. send_web_traffic.py

Purpose:
This script is engineered to generate both HTTP and HTTPS web traffic toward a specified target IP address, replicating real-world web traffic scenarios. This simulation is crucial for stress-testing, analyzing the security posture, and evaluating the robustness of web servers under varying traffic loads.

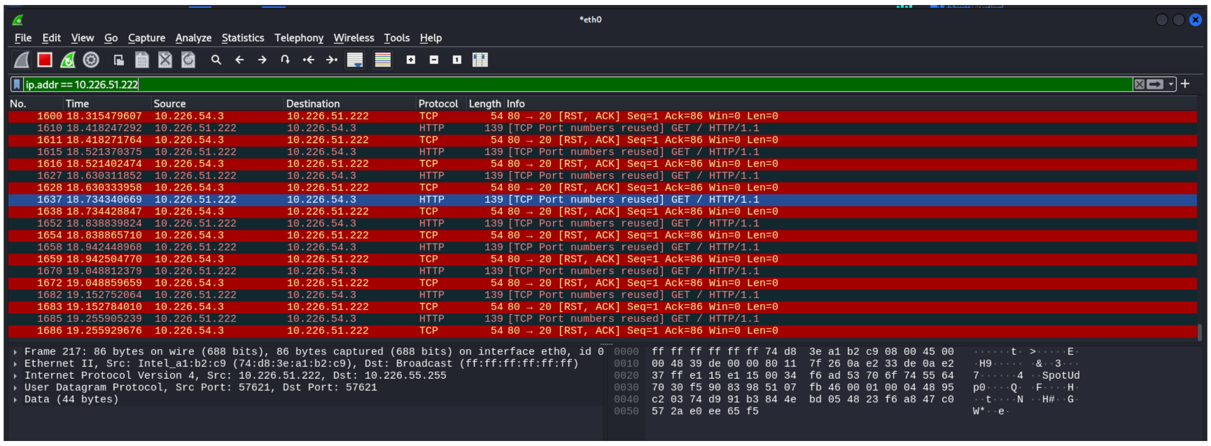
Technical Functionality:

- **Protocol Support:** The script accommodates both HTTP and HTTPS traffic, allowing for versatile testing scenarios. The distinction between the two lies in the port used, with HTTPS traffic routed through the standard port 443, albeit without true SSL/TLS encryption.
- **Packet Crafting:** Utilizing the Scapy library, the script crafts custom HTTP GET requests. For HTTPS, it mimics secure traffic by sending the same requests over the HTTPS port.
- **Traffic Control:** Users can define the number of GET requests and the interval between them, enabling controlled and repeatable test conditions. This feature is particularly useful for performance testing and benchmarking.
- **Traffic Patterns:** The script can simulate burst traffic patterns or sustained load conditions, depending on the configuration, to test how the server handles different traffic scenarios.

Usage:

- **Command:** `python send_web_traffic.py <target_ip> <protocol>`
- **Example:** `python send_web_traffic.py 10.226.54.3 http`

Wireshark Output:
In Wireshark, the generated traffic is visible as HTTP or HTTPS packets, depending on the protocol selected. This visibility allows for detailed packet-level analysis, such as examining request headers, payloads, and server responses, providing insights into server behavior under simulated traffic loads.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222
```

```
No.   Time         Source         Destination    Protocol Length Info
 1600 18.315479607 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
 1610 18.418247292 10.226.51.222  10.226.54.3    HTTP      139 [TCP Port numbers reused] GET / HTTP/1.1
 1611 18.418271764 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
 1615 18.521370375 10.226.51.222  10.226.54.3    HTTP      139 [TCP Port numbers reused] GET / HTTP/1.1
 1616 18.521402474 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
 1627 18.630311852 10.226.51.222  10.226.54.3    HTTP      139 [TCP Port numbers reused] GET / HTTP/1.1
 1628 18.630333958 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
 1637 18.734340669 10.226.51.222  10.226.54.3    HTTP      139 [TCP Port numbers reused] GET / HTTP/1.1
 1638 18.734428847 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
 1652 18.838839824 10.226.51.222  10.226.54.3    HTTP      139 [TCP Port numbers reused] GET / HTTP/1.1
 1654 18.838865710 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
 1658 18.942448968 10.226.51.222  10.226.54.3    HTTP      139 [TCP Port numbers reused] GET / HTTP/1.1
 1659 18.942504770 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
 1670 19.048812379 10.226.51.222  10.226.54.3    HTTP      139 [TCP Port numbers reused] GET / HTTP/1.1
 1672 19.048859659 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
 1682 19.152752064 10.226.51.222  10.226.54.3    HTTP      139 [TCP Port numbers reused] GET / HTTP/1.1
 1683 19.152784010 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
 1685 19.255905239 10.226.51.222  10.226.54.3    HTTP      139 [TCP Port numbers reused] GET / HTTP/1.1
 1686 19.255929676 10.226.54.3    10.226.51.222  TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=86 Win=0 Len=0
```

```
▸ Frame 217: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface eth0, id 0
▸ Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
▸ User Datagram Protocol, Src Port: 57621, Dst Port: 57621
▸ Data (44 bytes)
```

```
0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 48 39 de 00 00 80 11  7f 26 0a e2 33 de 0a e2   ·H9····· ·&··3···
0020  37 ff e1 15 e1 15 00 34  f6 ad 53 70 6f 74 55 64   7······4 ··SpotUd
0030  70 30 f5 90 83 98 51 07  fb 46 00 01 00 04 48 95   p0····Q· ·F····H·
0040  c2 03 74 d9 91 b3 84 4e  bd 05 48 23 f6 a8 47 c0   ··t····N ··H#··G·
0050  57 2a e0 ee 65 f5                                  W*··e·
```

4.2. send_video_streaming_traffic.py

Purpose:
This script simulates Real-time Transport Protocol (RTP) video streaming traffic aimed at a target IP address. It is designed to assess the handling and performance of video streaming data over a network, which is critical for understanding the impact of video traffic on network resources.

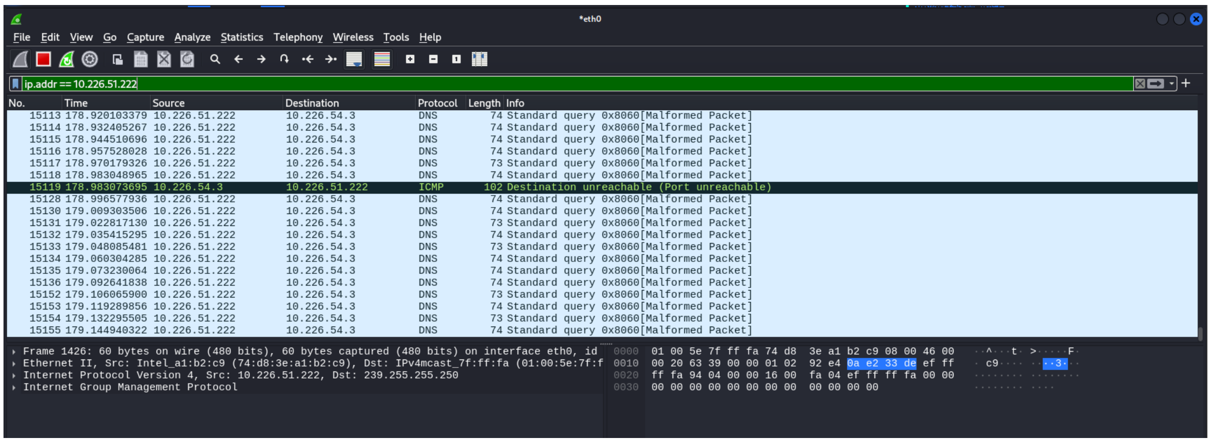
Technical Functionality:

- **RTP Packet Generation:** The script generates RTP packets with synthetic video frame data, including unique sequence numbers and timestamps to emulate live video streams. This is essential for testing jitter, latency, and packet loss in streaming scenarios.
- **Traffic Configuration:** Users can specify the target IP, the number of RTP packets to send, and the interval between packets. This flexibility allows for the simulation of various streaming conditions, from low-bitrate to high-definition video streams.
- **Network Impact Simulation:** The script can simulate varying video streaming conditions, helping to identify bottlenecks and optimize Quality of Service (QoS) parameters.

Usage:

- **Command:** `python send_video_streaming_traffic.py <target_ip>`
- **Example:** `python send_video_streaming_traffic.py 192.168.1.1`

Wireshark Output:
Wireshark will display the RTP traffic, where packet details such as sequence numbers, timestamps, and payload types can be analyzed. This information is crucial for diagnosing issues related to video quality, such as delays, out-of-order packets, or dropped frames.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222

No.   Time           Source          Destination     Protocol Length Info
15113 178.920103379 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15114 178.932405267 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15115 178.944510696 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15116 178.957528028 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15117 178.970179326 10.226.51.222   10.226.54.3     DNS       73 Standard query 0x8060[Malformed Packet]
15118 178.983048965 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15119 178.983073695 10.226.54.3     10.226.51.222   ICMP     102 Destination unreachable (Port unreachable)
15128 178.996577936 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15130 179.009303506 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15131 179.022817130 10.226.51.222   10.226.54.3     DNS       73 Standard query 0x8060[Malformed Packet]
15132 179.035415295 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15133 179.048085481 10.226.51.222   10.226.54.3     DNS       73 Standard query 0x8060[Malformed Packet]
15134 179.060304285 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15135 179.073230064 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15136 179.092641838 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15152 179.106065900 10.226.51.222   10.226.54.3     DNS       73 Standard query 0x8060[Malformed Packet]
15153 179.119289856 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]
15154 179.132295505 10.226.51.222   10.226.54.3     DNS       73 Standard query 0x8060[Malformed Packet]
15155 179.144940322 10.226.51.222   10.226.54.3     DNS       74 Standard query 0x8060[Malformed Packet]

› Frame 1426: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:f
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 239.255.255.250
› Internet Group Management Protocol

0000  01 00 5e 7f ff fa 74 d8  3e a1 b2 c9 08 00 46 00   ··^···t· >·····F·
0010  00 20 63 39 00 00 01 02  92 e4 0a e2 33 de ef ff   · c9···· ····3···
0020  ff fa 94 04 00 00 16 00  fa 04 ef ff ff fa 00 00   ········ ········
0030  00 00 00 00 00 00 00 00  00 00 00 00               ········ ····
```

4.3. send_udp.py

Purpose:
This script is designed to send User Datagram Protocol (UDP) traffic to a specified target IP and port. UDP is widely used in time-sensitive applications like gaming, VoIP, and streaming, making it essential to test network performance under UDP traffic.

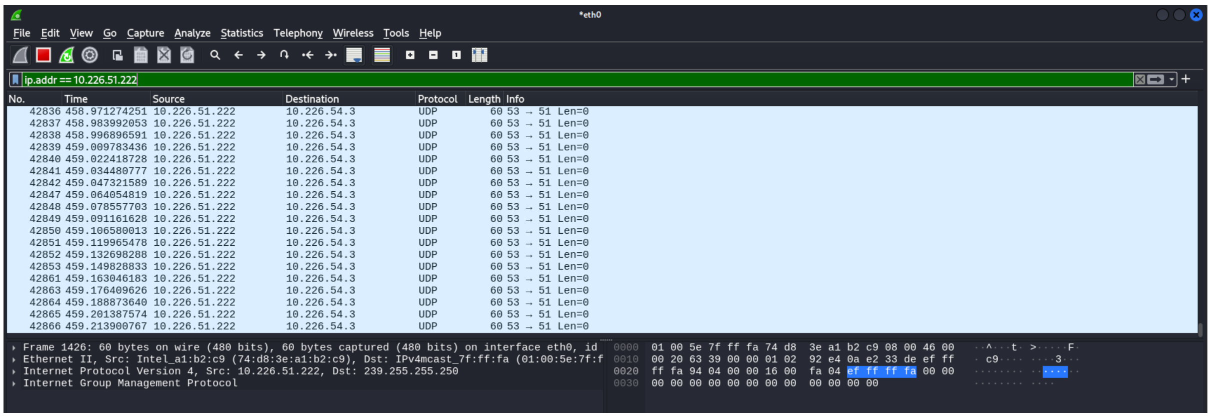
Technical Functionality:

- **Packet Generation:** The script crafts simple UDP packets, providing a lightweight, connectionless protocol for testing. This helps in assessing the network's ability to handle UDP traffic without the overhead of connection management.
- **Customizability:** Users can specify the target IP, port, number of packets, and the interval between sending them. This allows for precise control over the test conditions, enabling the simulation of both low and high traffic scenarios.
- **Application:** This script is ideal for testing network devices and servers that rely on or support UDP traffic, such as DNS servers, streaming services, and online gaming platforms.

Usage:

- **Command:** `python send_udp.py <target_ip> <target_port>`
- **Example:** `python send_udp.py 192.168.1.1 5004`

Wireshark Output:
UDP packets generated by this script will be visible in Wireshark, where you can inspect packet headers, payloads, and analyze packet flow to understand network behavior under UDP traffic conditions.



```
ip.addr == 10.226.51.222

No.   Time         Source          Destination    Protocol  Length Info
42836 458.971274251 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42837 458.983992053 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42838 458.996896591 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42839 459.009783436 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42840 459.022418728 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42841 459.034480777 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42842 459.047321589 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42847 459.064054819 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42848 459.078557703 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42849 459.091161628 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42850 459.106580013 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42851 459.119965478 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42852 459.132698288 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42853 459.149828833 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42861 459.163046183 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42863 459.176409626 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42864 459.188873640 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42865 459.201387574 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0
42866 459.213900767 10.226.51.222  10.226.54.3    UDP       60 53 → 51 Len=0

› Frame 1426: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:f
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 239.255.255.250
› Internet Group Management Protocol

0000  01 00 5e 7f ff fa 74 d8  3e a1 b2 c9 08 00 46 00   ··^···t· >·····F·
0010  00 20 63 39 00 00 01 02  92 e4 0a e2 33 de ef ff   · c9···· ····3···
0020  ff fa 94 04 00 00 16 00  fa 04 ef ff ff fa 00 00   ········ ········
0030  00 00 00 00 00 00 00 00  00 00 00 00               ········ ····
```

*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

4.4. send_tcp.py

Purpose:
This script sends Transmission Control Protocol (TCP) traffic to a designated target IP and port. TCP is the backbone of most internet traffic, making it vital to evaluate how a network handles TCP connections under various conditions.

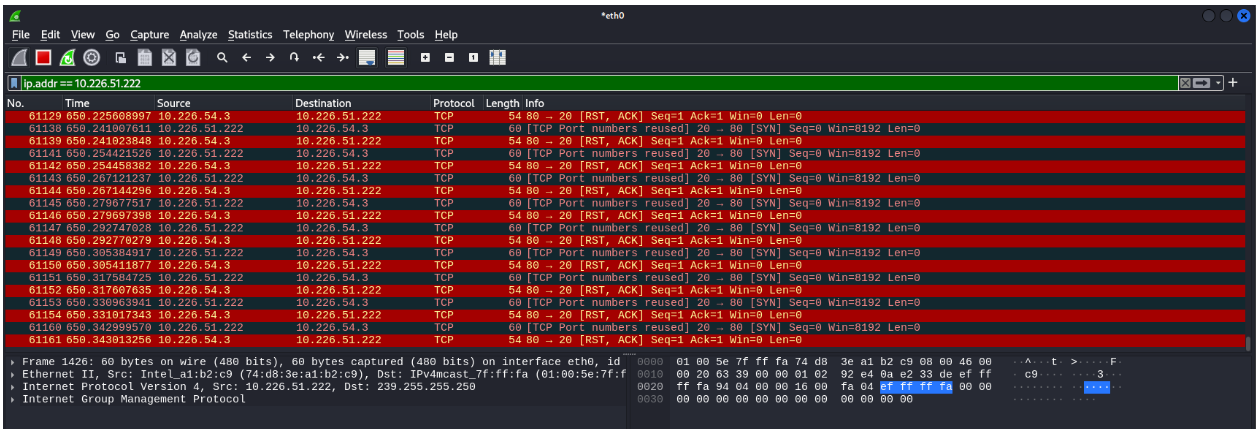
Technical Functionality:

- **TCP Segment Crafting:** The script generates TCP segments without establishing a full TCP connection (no SYN/ACK exchange). This approach is useful for testing firewalls, intrusion detection systems (IDS), and the initial handling of TCP segments by network devices.
- **Customization Options:** Users can specify the target IP, port, and parameters related to packet transmission, allowing for detailed testing scenarios such as slow or rapid packet sending.
- **Test Scenarios:** This script can be used to simulate potential TCP-based attacks (e.g., SYN floods) or to test the robustness of network services under abnormal TCP traffic.

Usage:

- **Command:** `python send_tcp.py <target_ip> <target_port>`
- **Example:** `python send_tcp.py 192.168.1.1 80`

Wireshark Output:
In Wireshark, the TCP segments will be visible, enabling analysis of the TCP header fields, sequence numbers, and payloads. This can provide insights into how the target system processes incomplete or malformed TCP segments.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ip.addr == 10.226.51.222

```
No.   Time             Source         Destination    Protocol Length Info
61129 650.225608997 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61138 650.241007611 10.226.51.222  10.226.54.3    TCP      60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
61139 650.241023848 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61141 650.254421526 10.226.51.222  10.226.54.3    TCP      60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
61142 650.254458382 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61143 650.267121237 10.226.51.222  10.226.54.3    TCP      60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
61144 650.267144296 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61145 650.279677517 10.226.51.222  10.226.54.3    TCP      60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
61146 650.279697398 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61147 650.292747028 10.226.51.222  10.226.54.3    TCP      60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
61148 650.292770279 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61149 650.305384917 10.226.51.222  10.226.54.3    TCP      60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
61150 650.305411877 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61151 650.317584725 10.226.51.222  10.226.54.3    TCP      60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
61152 650.317607635 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61153 650.330963941 10.226.51.222  10.226.54.3    TCP      60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
61154 650.331017343 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61160 650.342999570 10.226.51.222  10.226.54.3    TCP      60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
61161 650.343013256 10.226.54.3    10.226.51.222  TCP      54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

```
› Frame 1426: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:f
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 239.255.255.250
› Internet Group Management Protocol

0000  01 00 5e 7f ff fa 74 d8  3e a1 b2 c9 08 00 46 00   ··^···t· >·····F·
0010  00 20 63 39 00 00 01 02  92 e4 0a e2 33 de ef ff   · c9···· ····3···
0020  ff fa 94 04 00 00 16 00  fa 04 ef ff ff fa 00 00   ········ ········
0030  00 00 00 00 00 00 00 00  00 00 00 00               ········ ····
```

4.5. send_syn_scan.py

Purpose:
This script performs a SYN scan, a type of network reconnaissance technique, on a specified target IP. It sends SYN packets to a range of ports to identify open ports, which is a crucial step in vulnerability assessment and penetration testing.

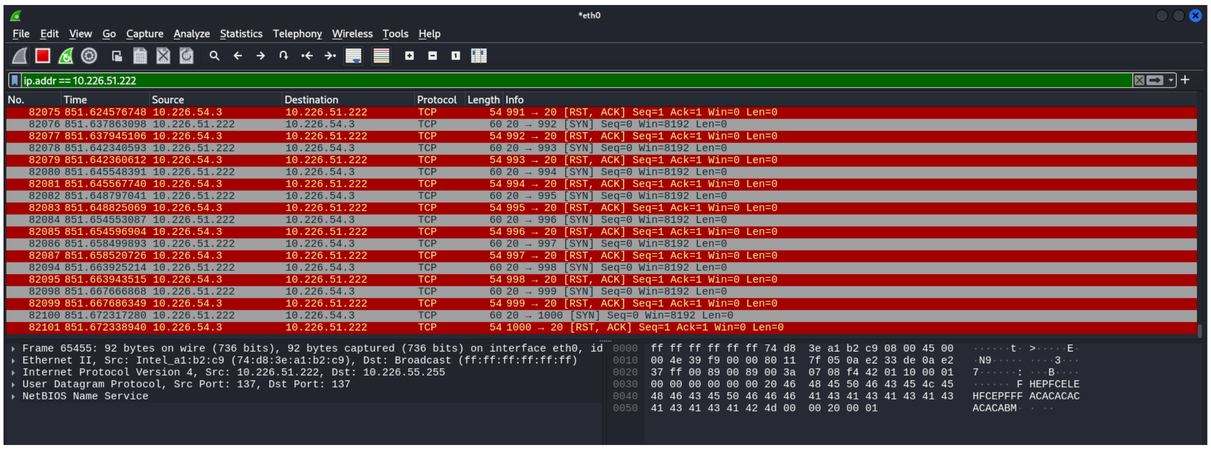
Technical Functionality:

- **SYN Packet Crafting:** The script crafts and sends TCP SYN packets to a specified range of ports on the target system. If a SYN-ACK is received, the port is considered open, which can then be logged or further analyzed.
- **Port Range Customization:** Users can define the start and end port for the scan, as well as the number of packets and interval between them, allowing for targeted or broad scanning.
- **Reconnaissance Use:** This script is particularly useful for identifying potentially vulnerable services running on a network, aiding in the early stages of a security assessment.

Usage:

- **Command:** `python send_syn_scan.py <target_ip>`
- **Example:** `python send_syn_scan.py 192.168.1.1`

Wireshark Output:
Wireshark will capture the SYN packets sent during the scan. By analyzing the responses (SYN-ACK, RST, etc.), one can identify open, closed, or filtered ports, providing a snapshot of the network's exposure to potential threats.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ip.addr == 10.226.51.222

```
No.   Time          Source         Destination    Protocol Length Info
82075 851.624576748 10.226.54.3    10.226.51.222  TCP      54 991 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
82076 851.637863098 10.226.51.222  10.226.54.3    TCP      60 20 → 992 [SYN] Seq=0 Win=8192 Len=0
82077 851.637945106 10.226.54.3    10.226.51.222  TCP      54 992 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
82078 851.642340593 10.226.51.222  10.226.54.3    TCP      60 20 → 993 [SYN] Seq=0 Win=8192 Len=0
82079 851.642360612 10.226.54.3    10.226.51.222  TCP      54 993 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
82080 851.645548391 10.226.51.222  10.226.54.3    TCP      60 20 → 994 [SYN] Seq=0 Win=8192 Len=0
82081 851.645567740 10.226.54.3    10.226.51.222  TCP      54 994 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
82082 851.648797041 10.226.51.222  10.226.54.3    TCP      60 20 → 995 [SYN] Seq=0 Win=8192 Len=0
82083 851.648825069 10.226.54.3    10.226.51.222  TCP      54 995 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
82084 851.654553087 10.226.51.222  10.226.54.3    TCP      60 20 → 996 [SYN] Seq=0 Win=8192 Len=0
82085 851.654596904 10.226.54.3    10.226.51.222  TCP      54 996 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
82086 851.658499893 10.226.51.222  10.226.54.3    TCP      60 20 → 997 [SYN] Seq=0 Win=8192 Len=0
82087 851.658520726 10.226.54.3    10.226.51.222  TCP      54 997 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
82094 851.663925214 10.226.51.222  10.226.54.3    TCP      60 20 → 998 [SYN] Seq=0 Win=8192 Len=0
82095 851.663943515 10.226.54.3    10.226.51.222  TCP      54 998 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
82098 851.667666868 10.226.51.222  10.226.54.3    TCP      60 20 → 999 [SYN] Seq=0 Win=8192 Len=0
82099 851.667686349 10.226.54.3    10.226.51.222  TCP      54 999 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
82100 851.672317280 10.226.51.222  10.226.54.3    TCP      60 20 → 1000 [SYN] Seq=0 Win=8192 Len=0
82101 851.672338940 10.226.54.3    10.226.51.222  TCP      54 1000 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

```
› Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
› User Datagram Protocol, Src Port: 137, Dst Port: 137
› NetBIOS Name Service
```

```
0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.6. send_pop3_traffic.py

Purpose:
This script simulates traffic using the Post Office Protocol version 3 (POP3), a protocol used by email clients to retrieve messages from a server. It is designed to test the security and performance of email servers and networks handling POP3 traffic.

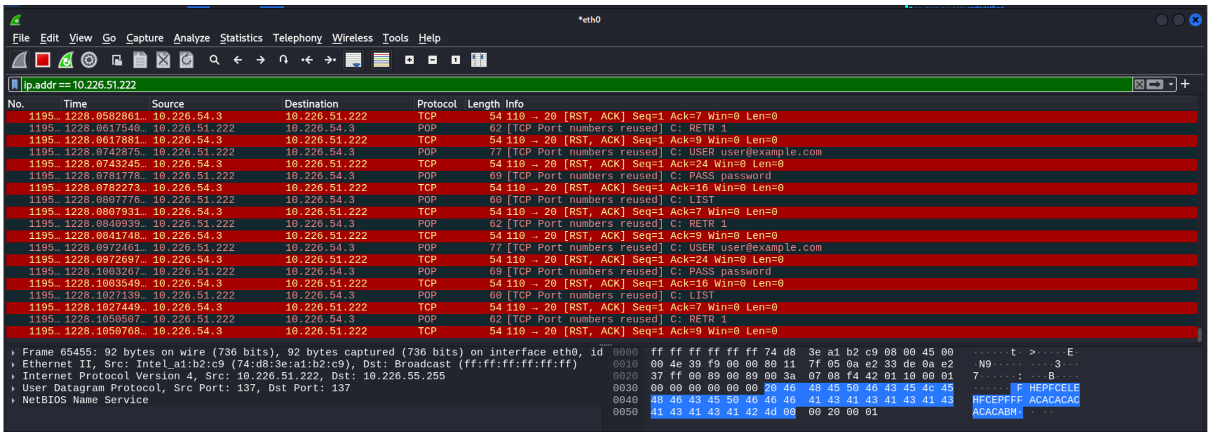
Technical Functionality:

- **POP3 Packet Crafting:** The script generates POP3 packets that include typical commands such as login, mailbox listing, and email retrieval. These packets simulate the actions of a legitimate email client.
- **Target Customization:** Users can specify the target IP, the number of packets, and the interval between them, allowing for detailed performance testing under varying load conditions.
- **Security Testing:** By simulating POP3 traffic, this script can help assess the effectiveness of security measures such as encryption, authentication, and intrusion detection.

Usage:

- **Command:** `python send_pop3_traffic.py <target_ip>`
- **Example:** `python send_pop3_traffic.py 192.168.1.1`

Wireshark Output:
In Wireshark, POP3 traffic will be visible, showing the client-server interaction. This allows for detailed analysis of command sequences, server responses, and the potential identification of security weaknesses in the email retrieval process.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ip.addr == 10.226.51.222

```
No.    Time            Source          Destination     Protocol Length Info
1195… 1228.0582861… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=7 Win=0 Len=0
1195… 1228.0617540… 10.226.51.222   10.226.54.3     POP      62 [TCP Port numbers reused] C: RETR 1
1195… 1228.0617881… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
1195… 1228.0742875… 10.226.51.222   10.226.54.3     POP      77 [TCP Port numbers reused] C: USER user@example.com
1195… 1228.0743245… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=24 Win=0 Len=0
1195… 1228.0781778… 10.226.51.222   10.226.54.3     POP      69 [TCP Port numbers reused] C: PASS password
1195… 1228.0782273… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=16 Win=0 Len=0
1195… 1228.0807776… 10.226.51.222   10.226.54.3     POP      60 [TCP Port numbers reused] C: LIST
1195… 1228.0807931… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=7 Win=0 Len=0
1195… 1228.0840939… 10.226.51.222   10.226.54.3     POP      62 [TCP Port numbers reused] C: RETR 1
1195… 1228.0841748… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
1195… 1228.0972461… 10.226.51.222   10.226.54.3     POP      77 [TCP Port numbers reused] C: USER user@example.com
1195… 1228.0972697… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=24 Win=0 Len=0
1195… 1228.1003267… 10.226.51.222   10.226.54.3     POP      69 [TCP Port numbers reused] C: PASS password
1195… 1228.1003549… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=16 Win=0 Len=0
1195… 1228.1027139… 10.226.51.222   10.226.54.3     POP      60 [TCP Port numbers reused] C: LIST
1195… 1228.1027449… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=7 Win=0 Len=0
1195… 1228.1050507… 10.226.51.222   10.226.54.3     POP      62 [TCP Port numbers reused] C: RETR 1
1195… 1228.1050768… 10.226.54.3     10.226.51.222   TCP      54 110 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
```

```
› Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
› User Datagram Protocol, Src Port: 137, Dst Port: 137
› NetBIOS Name Service
```

```
0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.7. send_imap_traffic.py

Purpose:
This script is designed to generate Internet Message Access Protocol (IMAP) traffic toward a specified target IP, simulating the behavior of an email client retrieving and managing messages from a server. IMAP is widely used in modern email systems, making this script essential for network and security testing.

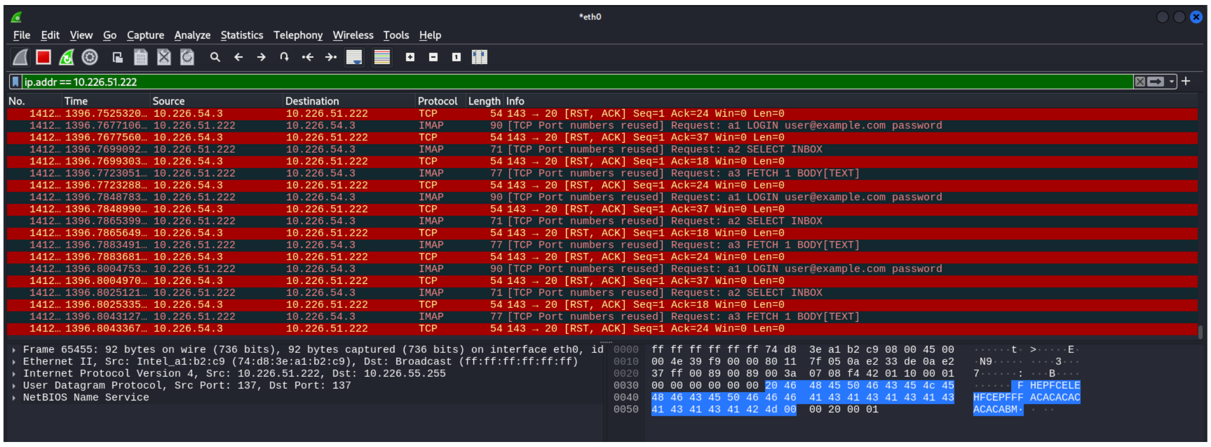
Technical Functionality:

- **IMAP Packet Crafting:** The script crafts IMAP packets that simulate various client commands such as login, mailbox selection, and email fetching. This helps in testing the server’s response to typical IMAP commands under different load conditions.
- **User-Controlled Testing:** The script allows users to specify the target IP, packet count, and interval, providing flexibility in testing different server loads and response times.
- **Security & Performance Evaluation:** This script is particularly useful for assessing the performance and security of IMAP servers, especially in environments where email traffic is a critical service.

Usage:

- **Command:** `python send_imap_traffic.py <target_ip>`
- **Example:** `python send_imap_traffic.py 192.168.1.1`

Wireshark Output:
IMAP packets generated by the script will be captured in Wireshark, providing visibility into the interactions between the client and server. This analysis can reveal potential performance issues, security vulnerabilities, and opportunities for optimizing server configurations.



```
ip.addr == 10.226.51.222

No.       Time            Source          Destination     Protocol  Length Info
1412…  1396.7525320…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=24 Win=0 Len=0
1412…  1396.7677106…  10.226.51.222   10.226.54.3     IMAP      90 [TCP Port numbers reused] Request: a1 LOGIN user@example.com password
1412…  1396.7677560…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=37 Win=0 Len=0
1412…  1396.7699092…  10.226.51.222   10.226.54.3     IMAP      71 [TCP Port numbers reused] Request: a2 SELECT INBOX
1412…  1396.7699303…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=18 Win=0 Len=0
1412…  1396.7723051…  10.226.51.222   10.226.54.3     IMAP      77 [TCP Port numbers reused] Request: a3 FETCH 1 BODY[TEXT]
1412…  1396.7723288…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=24 Win=0 Len=0
1412…  1396.7848783…  10.226.51.222   10.226.54.3     IMAP      90 [TCP Port numbers reused] Request: a1 LOGIN user@example.com password
1412…  1396.7848990…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=37 Win=0 Len=0
1412…  1396.7865399…  10.226.51.222   10.226.54.3     IMAP      71 [TCP Port numbers reused] Request: a2 SELECT INBOX
1412…  1396.7865649…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=18 Win=0 Len=0
1412…  1396.7883491…  10.226.51.222   10.226.54.3     IMAP      77 [TCP Port numbers reused] Request: a3 FETCH 1 BODY[TEXT]
1412…  1396.7883681…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=24 Win=0 Len=0
1412…  1396.8004753…  10.226.51.222   10.226.54.3     IMAP      90 [TCP Port numbers reused] Request: a1 LOGIN user@example.com password
1412…  1396.8004970…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=37 Win=0 Len=0
1412…  1396.8025121…  10.226.51.222   10.226.54.3     IMAP      71 [TCP Port numbers reused] Request: a2 SELECT INBOX
1412…  1396.8025335…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=18 Win=0 Len=0
1412…  1396.8043127…  10.226.51.222   10.226.54.3     IMAP      77 [TCP Port numbers reused] Request: a3 FETCH 1 BODY[TEXT]
1412…  1396.8043367…  10.226.54.3     10.226.51.222   TCP       54 143 → 20 [RST, ACK] Seq=1 Ack=24 Win=0 Len=0

Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
User Datagram Protocol, Src Port: 137, Dst Port: 137
NetBIOS Name Service

0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

4.8. send_p2p_traffic.py

Purpose:
This script simulates Peer-to-Peer (P2P) traffic, specifically focusing on BitTorrent, to test how a network handles P2P data exchanges. P2P traffic is often challenging to manage due to its decentralized nature, making this script valuable for testing network policies and bandwidth management.

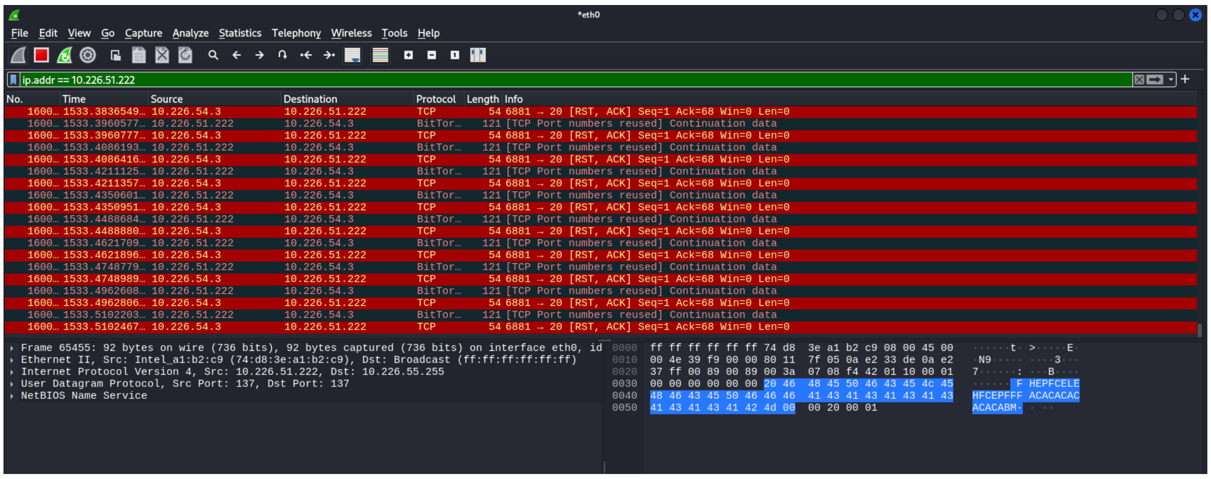
Technical Functionality:

- **BitTorrent Traffic Simulation:** The script generates and sends BitTorrent handshake messages to the target IP, simulating the initial steps of a P2P file-sharing session. This can help in testing network behavior under P2P traffic loads, which are often unpredictable and bursty.
- **Test Configurability:** Users can define the target IP, the number of packets, and the interval, allowing for simulations of different types of P2P traffic patterns, from light usage to heavy file sharing.
- **Network Impact Assessment:** This script helps in evaluating how P2P traffic impacts network performance, identifying potential bottlenecks, and testing the effectiveness of traffic shaping or throttling policies.

Usage:

- **Command:** `python send_p2p_traffic.py <target_ip>`
- **Example:** `python send_p2p_traffic.py 192.168.1.1`

Wireshark Output:
In Wireshark, BitTorrent packets will be visible, showing the handshake process and any subsequent data exchanges. This analysis is critical for understanding how P2P traffic is routed and managed within the network, and for detecting any potential security issues associated with P2P file sharing.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ip.addr == 10.226.51.222

```
No.     Time              Source          Destination     Protocol  Length Info
  1600… 1533.3836549… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
  1600… 1533.3960577… 10.226.51.222   10.226.54.3     BitTor…    121 [TCP Port numbers reused] Continuation data
  1600… 1533.3960777… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
  1600… 1533.4086193… 10.226.51.222   10.226.54.3     BitTor…    121 [TCP Port numbers reused] Continuation data
  1600… 1533.4086416… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
  1600… 1533.4211125… 10.226.51.222   10.226.54.3     BitTor…    121 [TCP Port numbers reused] Continuation data
  1600… 1533.4211357… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
  1600… 1533.4350601… 10.226.51.222   10.226.54.3     BitTor…    121 [TCP Port numbers reused] Continuation data
  1600… 1533.4350951… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
  1600… 1533.4488684… 10.226.51.222   10.226.54.3     BitTor…    121 [TCP Port numbers reused] Continuation data
  1600… 1533.4488880… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
  1600… 1533.4621709… 10.226.51.222   10.226.54.3     BitTor…    121 [TCP Port numbers reused] Continuation data
  1600… 1533.4621896… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
  1600… 1533.4748779… 10.226.51.222   10.226.54.3     BitTor…    121 [TCP Port numbers reused] Continuation data
  1600… 1533.4748989… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
  1600… 1533.4962608… 10.226.51.222   10.226.54.3     BitTor…    121 [TCP Port numbers reused] Continuation data
  1600… 1533.4962806… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
  1600… 1533.5102203… 10.226.51.222   10.226.54.3     BitTor…    121 [TCP Port numbers reused] Continuation data
  1600… 1533.5102467… 10.226.54.3     10.226.51.222   TCP         54 6881 → 20 [RST, ACK] Seq=1 Ack=68 Win=0 Len=0
```

```
› Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
› User Datagram Protocol, Src Port: 137, Dst Port: 137
› NetBIOS Name Service
```

```
0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.9. send_mysql_traffic.py

Purpose:
This script is designed to simulate MySQL traffic by sending SQL queries to a MySQL server, aiding in the evaluation of database performance and security under typical usage scenarios.

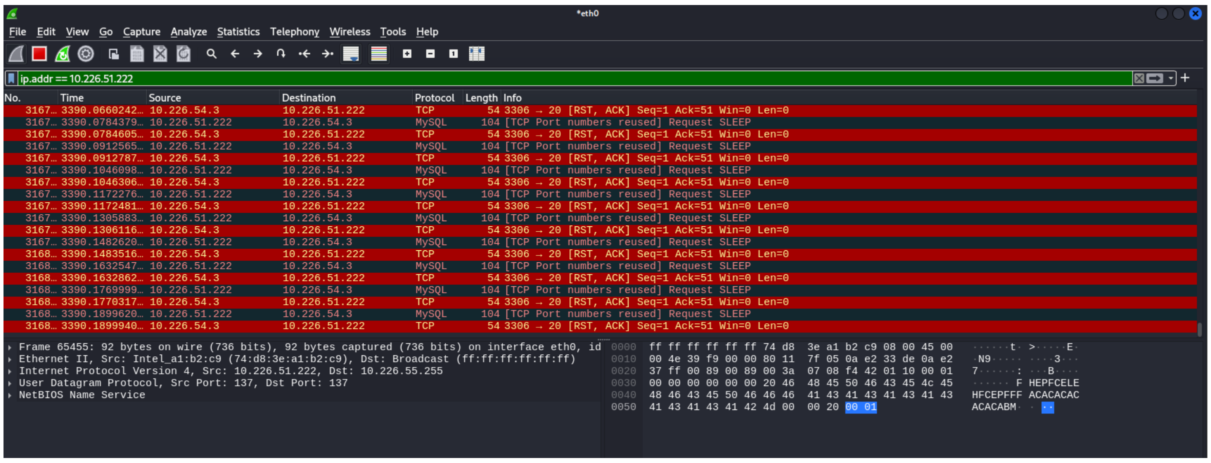
Technical Functionality:

- **SQL Query Simulation:** The script generates MySQL query packets that are sent to the target IP. These packets simulate basic database operations, providing insight into how the network and server handle database queries under load.
- **Configurable Parameters:** Users can specify the target IP, number of packets, interval, and port, enabling tailored tests that reflect different usage patterns, from light querying to heavy database interactions.
- **Application in Testing:** This script is ideal for testing MySQL server performance, network latency, and the impact of database queries on overall system performance. It can also be used to test the security of MySQL traffic, particularly in detecting vulnerabilities related to SQL injection or unauthorized access.

Usage:

- **Command:** `python send_mysql_traffic.py <target_ip>`
- **Example:** `python send_mysql_traffic.py 192.168.1.1`

Wireshark Output:
MySQL traffic will be captured in Wireshark, allowing for the analysis of SQL queries, responses, and any potential anomalies. This is particularly useful for detecting security issues or performance bottlenecks in the database communication process.


```
ip.addr == 10.226.51.222
No.     Time          Source         Destination    Protocol Length Info
3167…   3390.0660242… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0
3167…   3390.0784379… 10.226.51.222  10.226.54.3    MySQL    104 [TCP Port numbers reused] Request SLEEP
3167…   3390.0784605… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0
3167…   3390.0912565… 10.226.51.222  10.226.54.3    MySQL    104 [TCP Port numbers reused] Request SLEEP
3167…   3390.0912787… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0
3167…   3390.1046098… 10.226.51.222  10.226.54.3    MySQL    104 [TCP Port numbers reused] Request SLEEP
3167…   3390.1046306… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0
3167…   3390.1172276… 10.226.51.222  10.226.54.3    MySQL    104 [TCP Port numbers reused] Request SLEEP
3167…   3390.1172481… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0
3167…   3390.1305883… 10.226.51.222  10.226.54.3    MySQL    104 [TCP Port numbers reused] Request SLEEP
3167…   3390.1306116… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0
3167…   3390.1482620… 10.226.51.222  10.226.54.3    MySQL    104 [TCP Port numbers reused] Request SLEEP
3168…   3390.1483516… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0
3168…   3390.1632547… 10.226.51.222  10.226.54.3    MySQL    104 [TCP Port numbers reused] Request SLEEP
3168…   3390.1632862… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0
3168…   3390.1769999… 10.226.51.222  10.226.54.3    MySQL    104 [TCP Port numbers reused] Request SLEEP
3168…   3390.1770317… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0
3168…   3390.1899620… 10.226.51.222  10.226.54.3    MySQL    104 [TCP Port numbers reused] Request SLEEP
3168…   3390.1899940… 10.226.54.3    10.226.51.222  TCP       54 3306 → 20 [RST, ACK] Seq=1 Ack=51 Win=0 Len=0

› Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
› User Datagram Protocol, Src Port: 137, Dst Port: 137
› NetBIOS Name Service

0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.10. send_malware_traffic.py

Purpose:
This script simulates malware-related traffic by sending HTTP POST requests with malicious payloads. It is designed for testing the detection and response capabilities of security systems, such as intrusion detection systems (IDS) and firewalls.

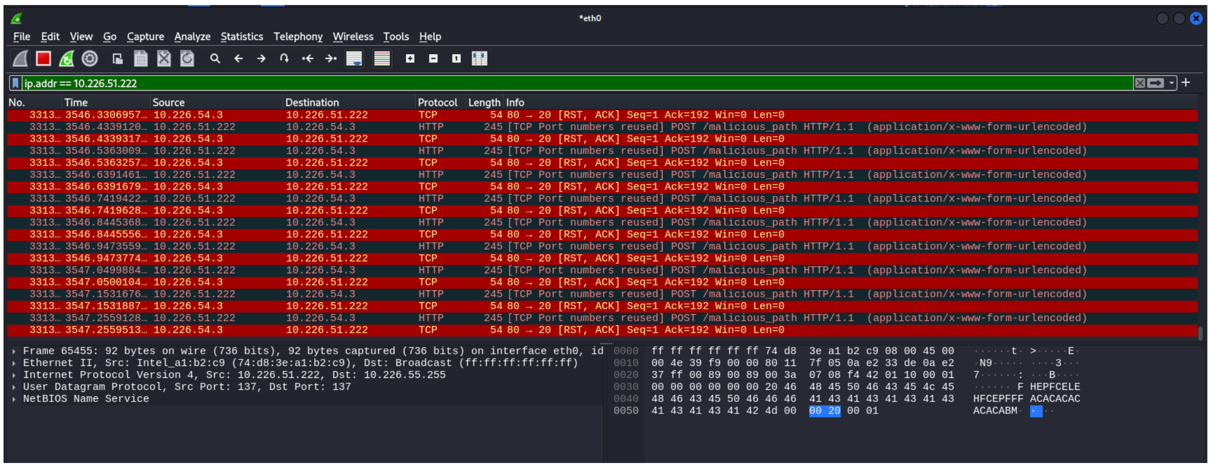
Technical Functionality:

- **Malicious Payload Simulation:** The script crafts HTTP POST requests containing payloads that mimic common malware behaviors, such as data exfiltration or command-and-control communication. This helps in testing the efficacy of security measures in place.
- **Targeted Testing:** Users can specify the target IP, the number of requests, and the interval, allowing for detailed testing of how different volumes and frequencies of malicious traffic are handled by the network.
- **Security Assessment:** This script is crucial for assessing the network's ability to detect and mitigate malware traffic. It helps identify potential vulnerabilities and areas where security defenses can be strengthened.

Usage:

- **Command:** `python send_malware_traffic.py <target_ip>`
- **Example:** `python send_malware_traffic.py 192.168.1.1`

Wireshark Output:
In Wireshark, the HTTP POST requests with the malicious payload will be visible. This allows security analysts to observe how the network and security systems respond to simulated malware traffic, providing valuable insights into the effectiveness of existing security controls.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222

No.      Time            Source          Destination     Protocol  Length Info
3313…    3546.3306957…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0
3313…    3546.4339120…   10.226.51.222   10.226.54.3     HTTP      245 [TCP Port numbers reused] POST /malicious_path HTTP/1.1  (application/x-www-form-urlencoded)
3313…    3546.4339317…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0
3313…    3546.5363009…   10.226.51.222   10.226.54.3     HTTP      245 [TCP Port numbers reused] POST /malicious_path HTTP/1.1  (application/x-www-form-urlencoded)
3313…    3546.5363257…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0
3313…    3546.6391461…   10.226.51.222   10.226.54.3     HTTP      245 [TCP Port numbers reused] POST /malicious_path HTTP/1.1  (application/x-www-form-urlencoded)
3313…    3546.6391679…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0
3313…    3546.7419422…   10.226.51.222   10.226.54.3     HTTP      245 [TCP Port numbers reused] POST /malicious_path HTTP/1.1  (application/x-www-form-urlencoded)
3313…    3546.7419628…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0
3313…    3546.8445368…   10.226.51.222   10.226.54.3     HTTP      245 [TCP Port numbers reused] POST /malicious_path HTTP/1.1  (application/x-www-form-urlencoded)
3313…    3546.8445556…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0
3313…    3546.9473559…   10.226.51.222   10.226.54.3     HTTP      245 [TCP Port numbers reused] POST /malicious_path HTTP/1.1  (application/x-www-form-urlencoded)
3313…    3546.9473774…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0
3313…    3547.0499884…   10.226.51.222   10.226.54.3     HTTP      245 [TCP Port numbers reused] POST /malicious_path HTTP/1.1  (application/x-www-form-urlencoded)
3313…    3547.0500104…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0
3313…    3547.1531676…   10.226.51.222   10.226.54.3     HTTP      245 [TCP Port numbers reused] POST /malicious_path HTTP/1.1  (application/x-www-form-urlencoded)
3313…    3547.1531887…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0
3313…    3547.2559128…   10.226.51.222   10.226.54.3     HTTP      245 [TCP Port numbers reused] POST /malicious_path HTTP/1.1  (application/x-www-form-urlencoded)
3313…    3547.2559513…   10.226.54.3     10.226.51.222   TCP        54 80 → 20 [RST, ACK] Seq=1 Ack=192 Win=0 Len=0

› Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
› User Datagram Protocol, Src Port: 137, Dst Port: 137
› NetBIOS Name Service

0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· ····
```

4.11. send_iot_traffic.py

Purpose: This script is engineered to emulate IoT (Internet of Things) device communication traffic, aimed at a specified target IP address. It serves as a robust tool for testing the network's capacity to manage the data traffic generated by multiple IoT devices, a scenario typical in smart homes, industrial automation, and large-scale IoT deployments.

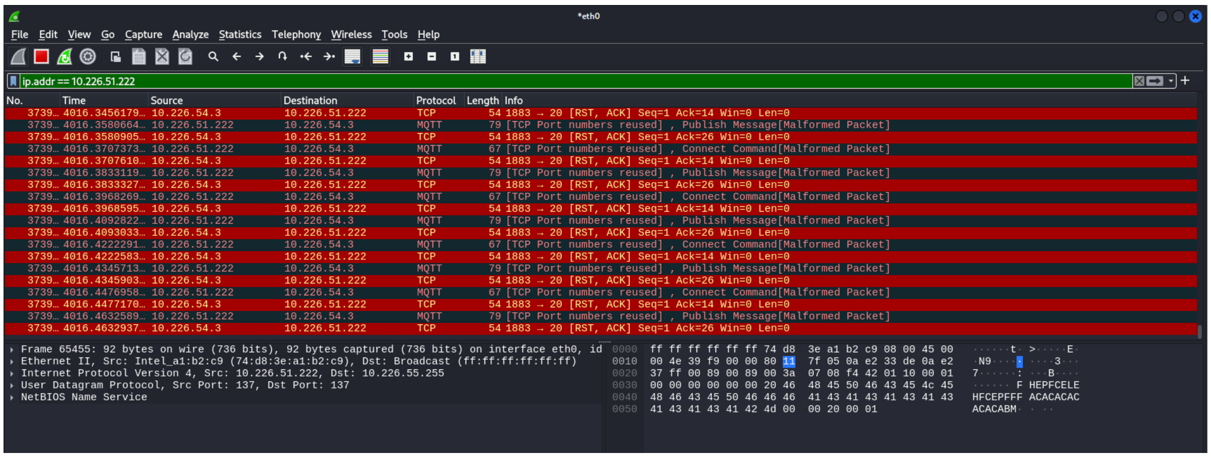
Functionality:

- **Multi-Protocol Simulation**: Supports the simulation of various IoT communication protocols such as MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), and HTTP. Each protocol reflects real-world IoT communication behavior.
- **Dynamic Packet Crafting**: Generates and dispatches packets that replicate the messaging patterns of different IoT devices, such as sensor data transmission or device status updates.
- **Customization Options**: Users can specify the target IP, simulate multiple IoT devices by adjusting parameters like the number of devices, and set the communication frequency (interval between messages).
- **Real-World Scenario Testing**: The script allows for testing network response to common IoT scenarios, such as device bootstrapping, data reporting, and command-and-control communication.

Usage:

- **Command**: python send_iot_traffic.py <target_ip> <protocol>
- **Example**: python send_iot_traffic.py 192.168.1.1 mqtt

Wireshark Output: IoT traffic will be visible in Wireshark, displaying protocol-specific packets, such as MQTT Publish messages or CoAP GET requests. This allows for a detailed examination of the IoT communication patterns and their impact on network performance.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222

No.      Time            Source         Destination     Protocol  Length Info
  3739… 4016.3456179… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=14 Win=0 Len=0
  3739… 4016.3580664… 10.226.51.222  10.226.54.3     MQTT      79 [TCP Port numbers reused] , Publish Message[Malformed Packet]
  3739… 4016.3580905… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=26 Win=0 Len=0
  3739… 4016.3707373… 10.226.51.222  10.226.54.3     MQTT      67 [TCP Port numbers reused] , Connect Command[Malformed Packet]
  3739… 4016.3707610… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=14 Win=0 Len=0
  3739… 4016.3833119… 10.226.51.222  10.226.54.3     MQTT      79 [TCP Port numbers reused] , Publish Message[Malformed Packet]
  3739… 4016.3833327… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=26 Win=0 Len=0
  3739… 4016.3968269… 10.226.51.222  10.226.54.3     MQTT      67 [TCP Port numbers reused] , Connect Command[Malformed Packet]
  3739… 4016.3968595… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=14 Win=0 Len=0
  3739… 4016.4092822… 10.226.51.222  10.226.54.3     MQTT      79 [TCP Port numbers reused] , Publish Message[Malformed Packet]
  3739… 4016.4093033… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=26 Win=0 Len=0
  3739… 4016.4222291… 10.226.51.222  10.226.54.3     MQTT      67 [TCP Port numbers reused] , Connect Command[Malformed Packet]
  3739… 4016.4222583… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=14 Win=0 Len=0
  3739… 4016.4345713… 10.226.51.222  10.226.54.3     MQTT      79 [TCP Port numbers reused] , Publish Message[Malformed Packet]
  3739… 4016.4345903… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=26 Win=0 Len=0
  3739… 4016.4476958… 10.226.51.222  10.226.54.3     MQTT      67 [TCP Port numbers reused] , Connect Command[Malformed Packet]
  3739… 4016.4477170… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=14 Win=0 Len=0
  3739… 4016.4632589… 10.226.51.222  10.226.54.3     MQTT      79 [TCP Port numbers reused] , Publish Message[Malformed Packet]
  3739… 4016.4632937… 10.226.54.3    10.226.51.222   TCP       54 1883 → 20 [RST, ACK] Seq=1 Ack=26 Win=0 Len=0

› Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
› User Datagram Protocol, Src Port: 137, Dst Port: 137
› NetBIOS Name Service

0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.12. send_icmp.py

Purpose: This script is crafted to generate ICMP (Internet Control Message Protocol) traffic, particularly useful for network diagnostics such as determining the reachability of a host and assessing network latency and packet loss.

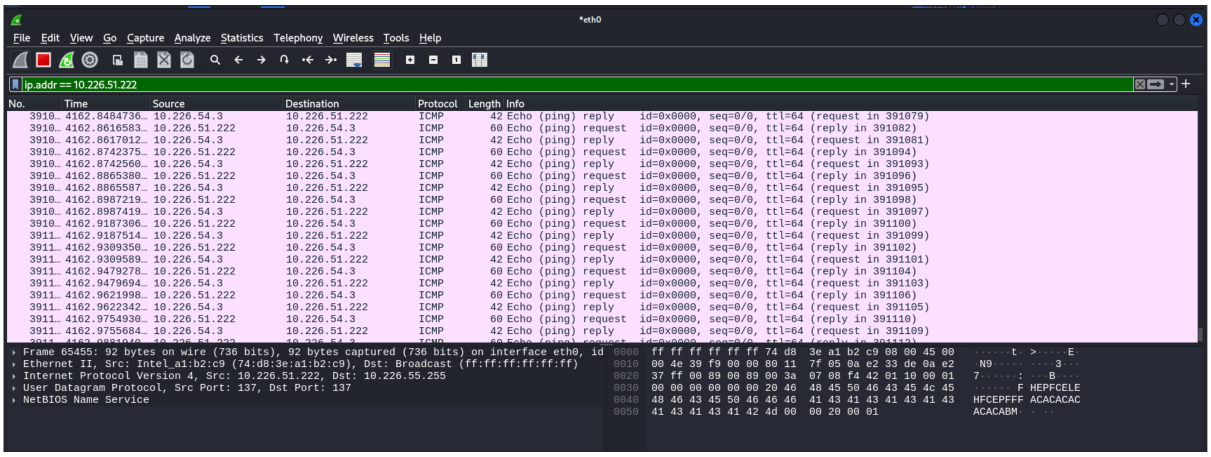
Functionality:

- **ICMP Echo Requests**: Primarily sends ICMP Echo Request packets (commonly known as ping) to a target IP, which is a fundamental method for testing network connectivity.
- **Configurable Parameters**: Users can adjust the number of ICMP packets sent, the interval between successive pings, and the size of the payload, allowing for tailored network diagnostics.
- **Comprehensive Network Testing**: The script can be used to test various aspects of network performance, including round-trip time, packet loss rate, and general network stability.

Usage:

- **Command**: python send_icmp.py <target_ip>
- **Example**: python send_icmp.py 192.168.1.1

Wireshark Output: In Wireshark, ICMP Echo Request and Echo Reply packets will be visible, providing insights into the round-trip time and any packet loss or delays, which are key indicators of network health.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222
No.     Time            Source          Destination     Protocol Length Info
 3910… 4162.8484736… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391079)
 3910… 4162.8616583… 10.226.51.222   10.226.54.3     ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 391082)
 3910… 4162.8617012… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391081)
 3910… 4162.8742375… 10.226.51.222   10.226.54.3     ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 391094)
 3910… 4162.8742560… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391093)
 3910… 4162.8865380… 10.226.51.222   10.226.54.3     ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 391096)
 3910… 4162.8865587… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391095)
 3910… 4162.8987219… 10.226.51.222   10.226.54.3     ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 391098)
 3910… 4162.8987419… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391097)
 3910… 4162.9187306… 10.226.51.222   10.226.54.3     ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 391100)
 3911… 4162.9187514… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391099)
 3911… 4162.9309350… 10.226.51.222   10.226.54.3     ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 391102)
 3911… 4162.9309589… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391101)
 3911… 4162.9479278… 10.226.51.222   10.226.54.3     ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 391104)
 3911… 4162.9479694… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391103)
 3911… 4162.9621998… 10.226.51.222   10.226.54.3     ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 391106)
 3911… 4162.9622342… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391105)
 3911… 4162.9754930… 10.226.51.222   10.226.54.3     ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 391110)
 3911… 4162.9755684… 10.226.54.3     10.226.51.222   ICMP      42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 391109)

▸ Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
▸ Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
▸ User Datagram Protocol, Src Port: 137, Dst Port: 137
▸ NetBIOS Name Service

0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.13. send_http_request.py

Purpose: This script is designed to generate and send HTTP requests to a specified web server. It is ideal for testing web server responsiveness, load handling, and security measures under various traffic conditions.

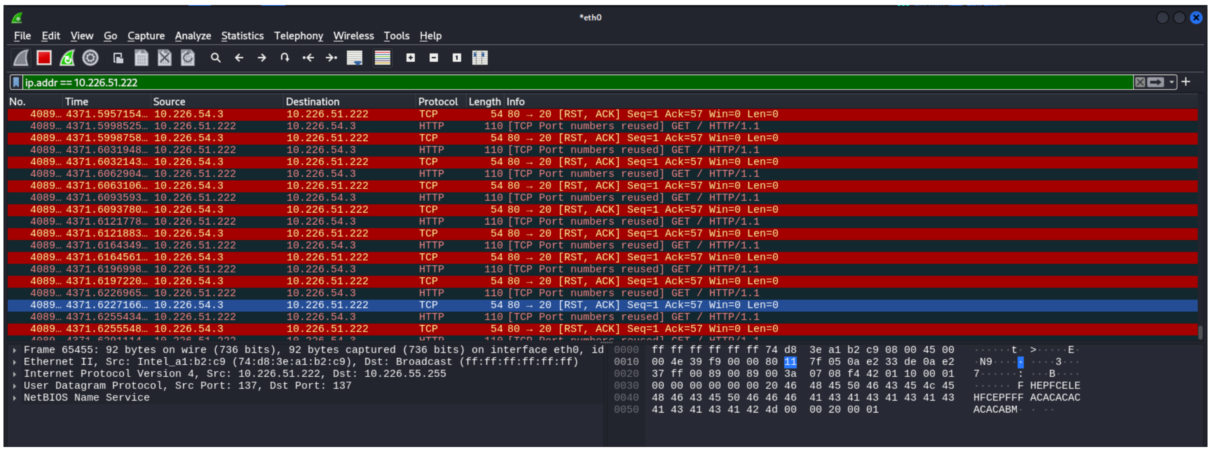
Functionality:

- **Supports Multiple HTTP Methods**: Capable of sending different types of HTTP requests, including GET, POST, and PUT, to mimic various client-server interactions.
- **Customizable Headers and Payloads**: Users can define custom headers, parameters, and payloads for each request, enabling simulations of real-world web traffic scenarios, such as form submissions or API calls.
- **Load Testing and Stress Testing**: The script allows for the generation of multiple requests in quick succession to test how the server handles high traffic volumes, simulating a load or stress testing environment.

Usage:

- **Command**: python send_http_request.py <target_ip> <request_type>
- **Example**: python send_http_request.py 192.168.1.1 GET

Wireshark Output: HTTP requests and responses can be inspected in Wireshark, where the full exchange between client and server, including headers and payloads, will be visible. This is useful for analyzing server behavior under various traffic conditions and for identifying potential security vulnerabilities.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222
No.      Time              Source          Destination     Protocol Length Info
4089…  4371.5957154…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0
4089…  4371.5998525…  10.226.51.222   10.226.54.3     HTTP     110 [TCP Port numbers reused] GET / HTTP/1.1
4089…  4371.5998758…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0
4089…  4371.6031948…  10.226.51.222   10.226.54.3     HTTP     110 [TCP Port numbers reused] GET / HTTP/1.1
4089…  4371.6032143…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0
4089…  4371.6062904…  10.226.51.222   10.226.54.3     HTTP     110 [TCP Port numbers reused] GET / HTTP/1.1
4089…  4371.6063106…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0
4089…  4371.6093593…  10.226.51.222   10.226.54.3     HTTP     110 [TCP Port numbers reused] GET / HTTP/1.1
4089…  4371.6093780…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0
4089…  4371.6121778…  10.226.51.222   10.226.54.3     HTTP     110 [TCP Port numbers reused] GET / HTTP/1.1
4089…  4371.6121883…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0
4089…  4371.6164349…  10.226.51.222   10.226.54.3     HTTP     110 [TCP Port numbers reused] GET / HTTP/1.1
4089…  4371.6164561…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0
4089…  4371.6196998…  10.226.51.222   10.226.54.3     HTTP     110 [TCP Port numbers reused] GET / HTTP/1.1
4089…  4371.6197220…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0
4089…  4371.6226965…  10.226.51.222   10.226.54.3     HTTP     110 [TCP Port numbers reused] GET / HTTP/1.1
4089…  4371.6227166…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0
4089…  4371.6255434…  10.226.51.222   10.226.54.3     HTTP     110 [TCP Port numbers reused] GET / HTTP/1.1
4089…  4371.6255548…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=57 Win=0 Len=0

› Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
› User Datagram Protocol, Src Port: 137, Dst Port: 137
› NetBIOS Name Service

0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.14. send_ftp_traffic.py

Purpose: This script is intended to simulate FTP (File Transfer Protocol) traffic, which is crucial for testing the performance, security, and functionality of FTP servers, particularly in scenarios involving large file transfers or multiple concurrent connections.

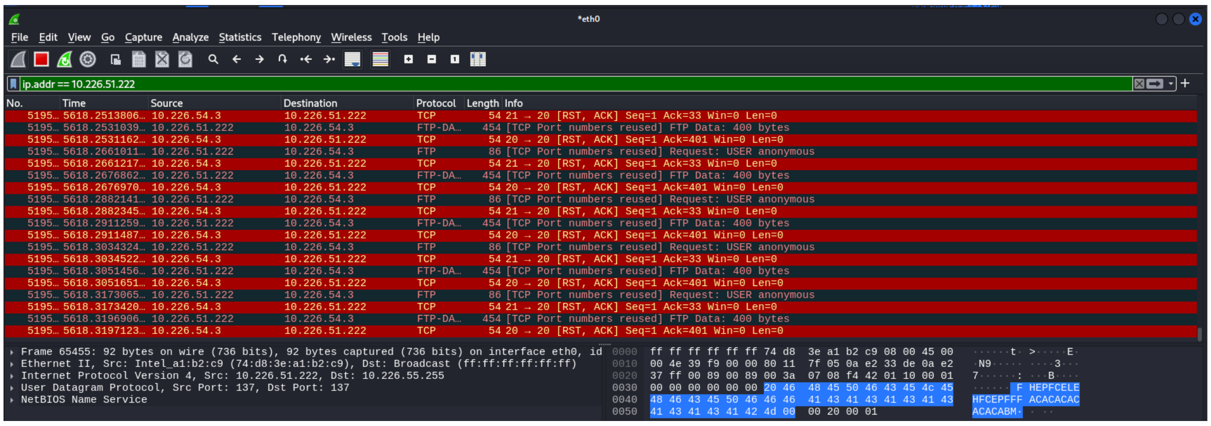
Functionality:

- **Comprehensive FTP Command Support**: Simulates various FTP operations, including login, file upload, and file download, allowing users to test different aspects of FTP server performance.
- **Active and Passive Modes**: Supports both active and passive FTP modes, offering flexibility in testing different network configurations and firewall settings.
- **Parameter Customization**: Users can specify the target IP, port, and types of FTP commands to be executed, enabling tailored testing for specific scenarios like stress testing or vulnerability assessments.

Usage:

- **Command**: python send_ftp_traffic.py <target_ip>
- **Example**: python send_ftp_traffic.py 192.168.1.1

Wireshark Output: In Wireshark, FTP commands and their corresponding data packets will be visible, allowing for detailed analysis of the FTP session, including command exchanges and data transfer efficiency.



```
ip.addr == 10.226.51.222

No.     Time             Source          Destination     Protocol  Length Info
5195…   5618.2513806…   10.226.54.3     10.226.51.222   TCP         54 21 → 20 [RST, ACK] Seq=1 Ack=33 Win=0 Len=0
5195…   5618.2531039…   10.226.51.222   10.226.54.3     FTP-DA…    454 [TCP Port numbers reused] FTP Data: 400 bytes
5195…   5618.2531162…   10.226.54.3     10.226.51.222   TCP         54 20 → 20 [RST, ACK] Seq=1 Ack=401 Win=0 Len=0
5195…   5618.2661011…   10.226.51.222   10.226.54.3     FTP         86 [TCP Port numbers reused] Request: USER anonymous
5195…   5618.2661217…   10.226.54.3     10.226.51.222   TCP         54 21 → 20 [RST, ACK] Seq=1 Ack=33 Win=0 Len=0
5195…   5618.2676862…   10.226.51.222   10.226.54.3     FTP-DA…    454 [TCP Port numbers reused] FTP Data: 400 bytes
5195…   5618.2676970…   10.226.54.3     10.226.51.222   TCP         54 20 → 20 [RST, ACK] Seq=1 Ack=401 Win=0 Len=0
5195…   5618.2882141…   10.226.51.222   10.226.54.3     FTP         86 [TCP Port numbers reused] Request: USER anonymous
5195…   5618.2882345…   10.226.54.3     10.226.51.222   TCP         54 21 → 20 [RST, ACK] Seq=1 Ack=33 Win=0 Len=0
5195…   5618.2911259…   10.226.51.222   10.226.54.3     FTP-DA…    454 [TCP Port numbers reused] FTP Data: 400 bytes
5195…   5618.2911487…   10.226.54.3     10.226.51.222   TCP         54 20 → 20 [RST, ACK] Seq=1 Ack=401 Win=0 Len=0
5195…   5618.3034324…   10.226.51.222   10.226.54.3     FTP         86 [TCP Port numbers reused] Request: USER anonymous
5195…   5618.3034522…   10.226.54.3     10.226.51.222   TCP         54 21 → 20 [RST, ACK] Seq=1 Ack=33 Win=0 Len=0
5195…   5618.3051456…   10.226.51.222   10.226.54.3     FTP-DA…    454 [TCP Port numbers reused] FTP Data: 400 bytes
5195…   5618.3051651…   10.226.54.3     10.226.51.222   TCP         54 20 → 20 [RST, ACK] Seq=1 Ack=401 Win=0 Len=0
5195…   5618.3173065…   10.226.51.222   10.226.54.3     FTP         86 [TCP Port numbers reused] Request: USER anonymous
5195…   5618.3173420…   10.226.54.3     10.226.51.222   TCP         54 21 → 20 [RST, ACK] Seq=1 Ack=33 Win=0 Len=0
5195…   5618.3196906…   10.226.51.222   10.226.54.3     FTP-DA…    454 [TCP Port numbers reused] FTP Data: 400 bytes
5195…   5618.3197123…   10.226.54.3     10.226.51.222   TCP         54 20 → 20 [RST, ACK] Seq=1 Ack=401 Win=0 Len=0

Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
User Datagram Protocol, Src Port: 137, Dst Port: 137
NetBIOS Name Service

0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.15. send_dns.py

Purpose: This script is used to generate DNS (Domain Name System) queries, which are essential for testing DNS server performance, response times, and the accuracy of DNS resolutions under varying loads and conditions.

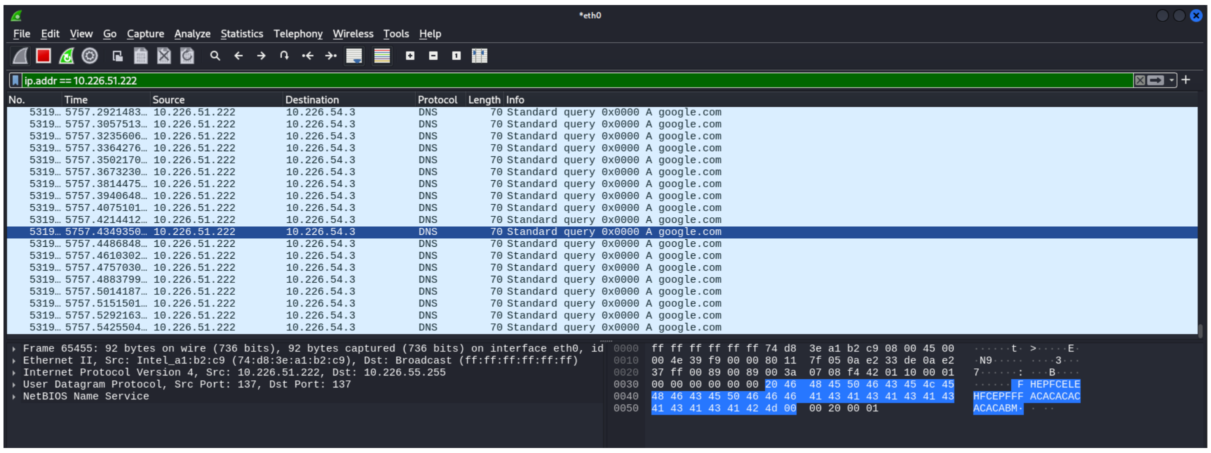
Functionality:

- **Versatile Query Generation**: Supports the creation of various types of DNS queries, including A (address), AAAA (IPv6 address), MX (mail exchange), and CNAME (canonical name) records.
- **TCP and UDP Support**: The script can send DNS queries over both UDP and TCP, providing a comprehensive tool for testing DNS servers under different network conditions.
- **Customizable Query Parameters**: Users can specify the DNS server, the domain name, and the type of DNS query, allowing for focused testing on specific DNS functionalities or potential vulnerabilities.

Usage:

- **Command**: python send_dns.py <dns_server_ip> <domain_name>
- **Example**: python send_dns.py 8.8.8.8 example.com

Wireshark Output: DNS queries and responses will be displayed in Wireshark, where the query type, response time, and the resolved IP addresses or records can be analyzed, helping to assess the DNS server's efficiency and reliability.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222
```

```
No.      Time          Source         Destination   Protocol Length Info
5319…  5757.2921483…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.3057513…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.3235606…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.3364276…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.3502170…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.3673230…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.3814475…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.3940648…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.4075101…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.4214412…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.4349350…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.4486848…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.4610302…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.4757030…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.4883799…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.5014187…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.5151501…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.5292163…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
5319…  5757.5425504…  10.226.51.222  10.226.54.3   DNS      70 Standard query 0x0000 A google.com
```

```
› Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
› User Datagram Protocol, Src Port: 137, Dst Port: 137
› NetBIOS Name Service
```

```
0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.16. send_ddos_traffic.py

Purpose: This script is designed to simulate a Distributed Denial of Service (DDoS) attack by generating a high volume of traffic towards a target IP. It is intended for testing the resilience and defensive capabilities of networks and systems against DDoS attacks.

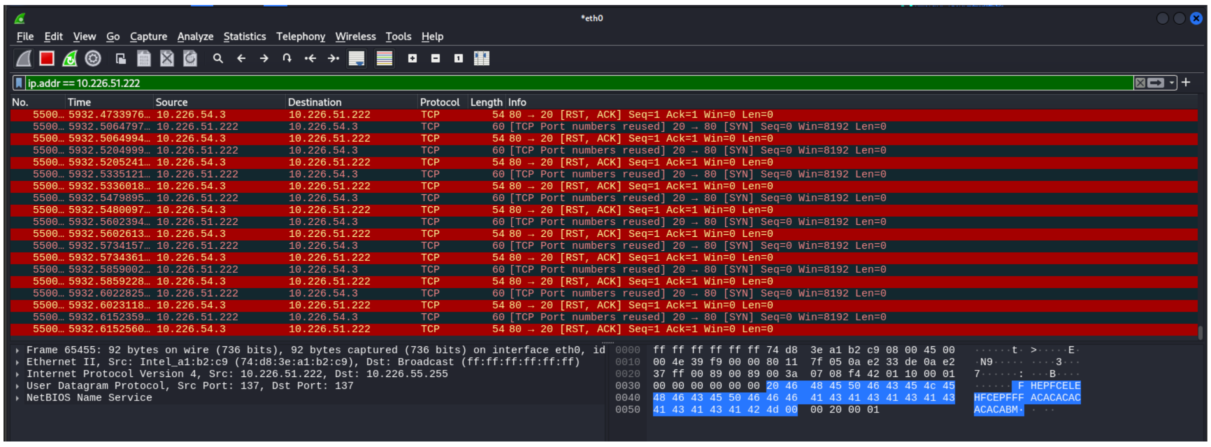
Functionality:

- **High-Volume Traffic Generation**: Capable of generating a massive number of TCP, UDP, or HTTP requests, overwhelming the target system to test its ability to withstand and mitigate DDoS attacks.
- **Traffic Type Customization**: Users can choose the type of traffic to be generated, such as SYN floods, UDP floods, or HTTP GET floods, each representing different types of DDoS attacks.
- **Duration and Volume Control**: The script allows users to set the volume of traffic and the duration of the attack, enabling realistic simulations of both short-burst and sustained DDoS attacks.

Usage:

- **Command**: python send_ddos_traffic.py <target_ip> <traffic_type>
- **Example**: python send_ddos_traffic.py 192.168.1.1 tcp

Wireshark Output: The generated traffic, such as SYN floods or HTTP request floods, will be highly visible in Wireshark, illustrating the overwhelming nature of the attack and its potential impact on the target system.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222
No.     Time           Source          Destination     Protocol  Length Info
5500…  5932.4733976…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5500…  5932.5064797…  10.226.51.222   10.226.54.3     TCP       60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
5500…  5932.5064994…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5500…  5932.5204999…  10.226.51.222   10.226.54.3     TCP       60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
5500…  5932.5205241…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5500…  5932.5335121…  10.226.51.222   10.226.54.3     TCP       60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
5500…  5932.5336018…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5500…  5932.5479895…  10.226.51.222   10.226.54.3     TCP       60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
5500…  5932.5480097…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5500…  5932.5602394…  10.226.51.222   10.226.54.3     TCP       60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
5500…  5932.5602613…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5500…  5932.5734157…  10.226.51.222   10.226.54.3     TCP       60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
5500…  5932.5734361…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5500…  5932.5859002…  10.226.51.222   10.226.54.3     TCP       60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
5500…  5932.5859228…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5500…  5932.6022825…  10.226.51.222   10.226.54.3     TCP       60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
5500…  5932.6023118…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5500…  5932.6152359…  10.226.51.222   10.226.54.3     TCP       60 [TCP Port numbers reused] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
5500…  5932.6152560…  10.226.54.3     10.226.51.222   TCP       54 80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

› Frame 65455: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface eth0, id
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.55.255
› User Datagram Protocol, Src Port: 137, Dst Port: 137
› NetBIOS Name Service

0000  ff ff ff ff ff ff 74 d8  3e a1 b2 c9 08 00 45 00   ······t· >·····E·
0010  00 4e 39 f9 00 00 80 11  7f 05 0a e2 33 de 0a e2   ·N9····· ····3···
0020  37 ff 00 89 00 89 00 3a  07 08 f4 42 01 10 00 01   7······: ···B····
0030  00 00 00 00 00 00 20 46  48 45 50 46 43 45 4c 45   ······ F HEPFCELE
0040  48 46 43 45 50 46 46 46  41 43 41 43 41 43 41 43   HFCEPFFF ACACACAC
0050  41 43 41 43 41 42 4d 00  00 20 00 01               ACACABM· · ··
```

4.17. send_rtp_traffic.py

Purpose: This script simulates RTP (Real-time Transport Protocol) traffic, which is commonly used in VoIP (Voice over IP) and video streaming applications. It is crucial for evaluating the Quality of Service (QoS) for real-time audio and video transmission across a network.

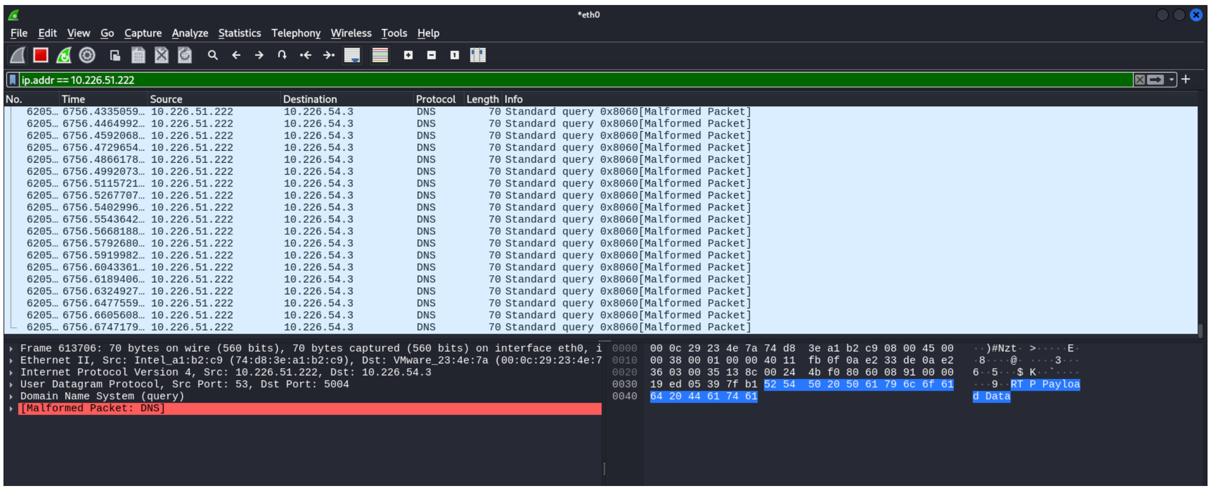
Functionality:

- **Simulated Media Streaming**: Generates RTP packets containing synthetic audio or video frame data, simulating the continuous media stream typical of VoIP calls or video conferencing.
- **Payload and Timing Customization**: Allows users to define the payload type, packet rate, and sequence numbers, enabling the creation of traffic that closely mimics real-world streaming scenarios.
- **QoS Testing**: The script is useful for testing network performance metrics such as jitter, packet loss, and latency, which are critical for maintaining the quality of real-time communication.

Usage:

- **Command**: python send_rtp_traffic.py <target_ip>
- **Example**: python send_rtp_traffic.py 192.168.1.1

Wireshark Output: RTP packets will be visible in Wireshark, where the media stream can be analyzed for key metrics like packet inter-arrival time, sequence integrity, and overall stream quality, providing valuable insights into the network's ability to support real-time communication.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222
No.     Time            Source          Destination     Protocol Length Info
  6205… 6756.4335059… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.4464992… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.4592068… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.4729654… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.4866178… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.4992073… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.5115721… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.5267707… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.5402996… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.5543642… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.5668188… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.5792680… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.5919982… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.6043361… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.6189406… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.6324927… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.6477559… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.6605608… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]
  6205… 6756.6747179… 10.226.51.222   10.226.54.3     DNS        70 Standard query 0x8060[Malformed Packet]

› Frame 613706: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface eth0, i
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: VMware_23:4e:7a (00:0c:29:23:4e:7
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.54.3
› User Datagram Protocol, Src Port: 53, Dst Port: 5004
› Domain Name System (query)
› [Malformed Packet: DNS]

0000  00 0c 29 23 4e 7a 74 d8  3e a1 b2 c9 08 00 45 00   ··)#Nzt· >·····E·
0010  00 38 00 01 00 00 40 11  fb 0f 0a e2 33 de 0a e2   ·8····@· ····3···
0020  36 03 00 35 13 8c 00 24  4b f0 80 60 08 91 00 00   6··5···$ K··`····
0030  19 ed 05 39 7f b1 52 54  50 20 50 61 79 6c 6f 61   ···9··RT P Payloa
0040  64 20 44 61 74 61                                  d Data
```

4.18. send_sip_traffic.py

Purpose: This script is designed to generate SIP (Session Initiation Protocol) traffic, which is fundamental in setting up, managing, and terminating VoIP (Voice over IP) calls. It is particularly useful for testing the performance, reliability, and security of SIP-based communication systems.

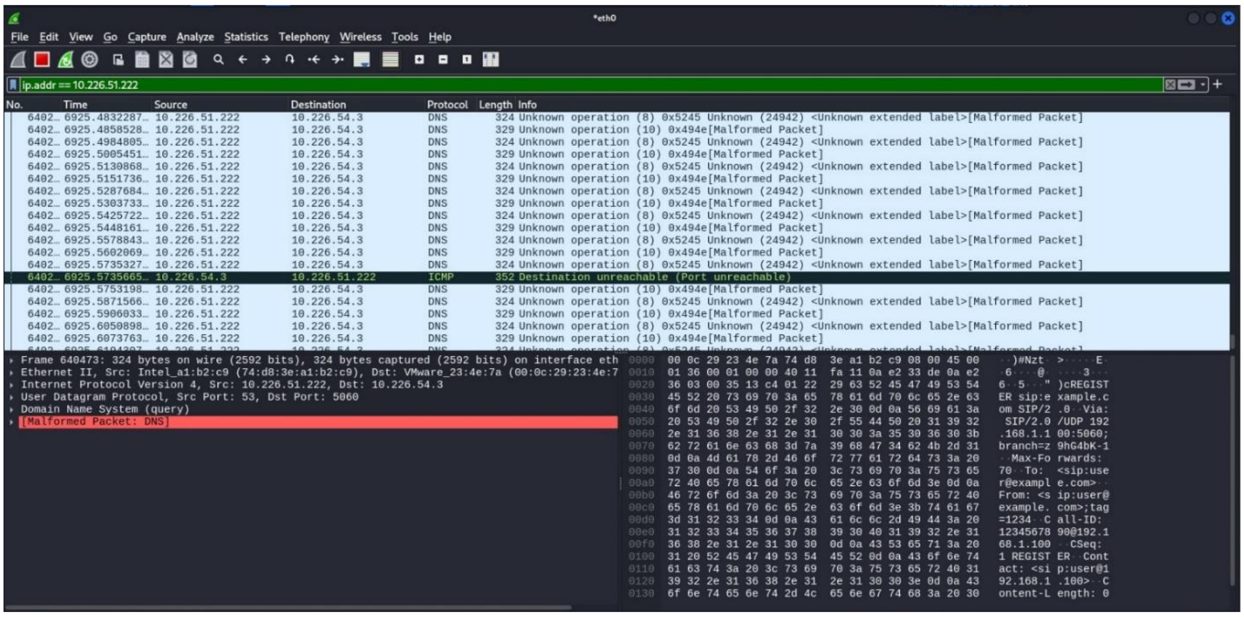
Functionality:

- **Comprehensive SIP Message Simulation**: The script supports the generation of various SIP messages, including INVITE, ACK, BYE, and REGISTER, which are essential for initiating, maintaining, and closing VoIP sessions.
- **Protocol Compliance**: It crafts SIP packets that adhere to RFC 3261, ensuring that the traffic is realistic and compliant with standard SIP protocol specifications.
- **Customizable Parameters**: Users can define the target IP, port, and SIP message type. The script also allows customization of headers, such as the Call-ID and CSeq, enabling detailed testing of SIP functionalities.
- **VoIP System Stress Testing**: The script can simulate multiple SIP calls in quick succession, useful for stress testing SIP servers or PBX systems to evaluate their capacity to handle large volumes of concurrent calls.

Usage:

- **Command**: python send_sip_traffic.py <target_ip> <sip_message_type>
- **Example**: python send_sip_traffic.py 192.168.1.1 INVITE

Wireshark Output: SIP traffic will be visible in Wireshark, where each SIP message, including its headers and payload, can be examined. This allows for a thorough analysis of the SIP signaling process, including call setup, negotiation, and termination, as well as potential issues such as call drops or registration failures.



*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 10.226.51.222
No.     Time            Source          Destination     Protocol  Length Info
 6402… 6925.4832287… 10.226.51.222  10.226.54.3     DNS       324 Unknown operation (8) 0x5245 Unknown (24942) <Unknown extended label>[Malformed Packet]
 6402… 6925.4858528… 10.226.51.222  10.226.54.3     DNS       329 Unknown operation (10) 0x494e[Malformed Packet]
 6402… 6925.4984805… 10.226.51.222  10.226.54.3     DNS       324 Unknown operation (8) 0x5245 Unknown (24942) <Unknown extended label>[Malformed Packet]
 6402… 6925.5005451… 10.226.51.222  10.226.54.3     DNS       329 Unknown operation (10) 0x494e[Malformed Packet]
 6402… 6925.5130868… 10.226.51.222  10.226.54.3     DNS       324 Unknown operation (8) 0x5245 Unknown (24942) <Unknown extended label>[Malformed Packet]
 6402… 6925.5151736… 10.226.51.222  10.226.54.3     DNS       329 Unknown operation (10) 0x494e[Malformed Packet]
 6402… 6925.5287684… 10.226.51.222  10.226.54.3     DNS       324 Unknown operation (8) 0x5245 Unknown (24942) <Unknown extended label>[Malformed Packet]
 6402… 6925.5303733… 10.226.51.222  10.226.54.3     DNS       329 Unknown operation (10) 0x494e[Malformed Packet]
 6402… 6925.5425722… 10.226.51.222  10.226.54.3     DNS       324 Unknown operation (8) 0x5245 Unknown (24942) <Unknown extended label>[Malformed Packet]
 6402… 6925.5448161… 10.226.51.222  10.226.54.3     DNS       329 Unknown operation (10) 0x494e[Malformed Packet]
 6402… 6925.5578843… 10.226.51.222  10.226.54.3     DNS       324 Unknown operation (8) 0x5245 Unknown (24942) <Unknown extended label>[Malformed Packet]
 6402… 6925.5602069… 10.226.51.222  10.226.54.3     DNS       329 Unknown operation (10) 0x494e[Malformed Packet]
 6402… 6925.5735327… 10.226.51.222  10.226.54.3     DNS       324 Unknown operation (8) 0x5245 Unknown (24942) <Unknown extended label>[Malformed Packet]
 6402… 6925.5735665… 10.226.54.3    10.226.51.222   ICMP      352 Destination unreachable (Port unreachable)
 6402… 6925.5753198… 10.226.51.222  10.226.54.3     DNS       329 Unknown operation (10) 0x494e[Malformed Packet]
 6402… 6925.5871566… 10.226.51.222  10.226.54.3     DNS       324 Unknown operation (8) 0x5245 Unknown (24942) <Unknown extended label>[Malformed Packet]
 6402… 6925.5906033… 10.226.51.222  10.226.54.3     DNS       329 Unknown operation (10) 0x494e[Malformed Packet]
 6402… 6925.6050898… 10.226.51.222  10.226.54.3     DNS       324 Unknown operation (8) 0x5245 Unknown (24942) <Unknown extended label>[Malformed Packet]
 6402… 6925.6073763… 10.226.51.222  10.226.54.3     DNS       329 Unknown operation (10) 0x494e[Malformed Packet]
```

```
› Frame 640473: 324 bytes on wire (2592 bits), 324 bytes captured (2592 bits) on interface eth
› Ethernet II, Src: Intel_a1:b2:c9 (74:d8:3e:a1:b2:c9), Dst: VMware_23:4e:7a (00:0c:29:23:4e:7
› Internet Protocol Version 4, Src: 10.226.51.222, Dst: 10.226.54.3
› User Datagram Protocol, Src Port: 53, Dst Port: 5060
› Domain Name System (query)
› [Malformed Packet: DNS]
```

```
0000  00 0c 29 23 4e 7a 74 d8  3e a1 b2 c9 08 00 45 00   ··)#Nzt· >·····E·
0010  01 36 00 01 00 00 40 11  fa 11 0a e2 33 de 0a e2   ·6····@· ····3···
0020  36 03 00 35 13 c4 01 22  29 63 52 45 47 49 53 54   6··5···" )cREGIST
0030  45 52 20 73 69 70 3a 65  78 61 6d 70 6c 65 2e 63   ER sip:e xample.c
0040  6f 6d 20 53 49 50 2f 32  2e 30 0d 0a 56 69 61 3a   om SIP/2 .0··Via:
0050  20 53 49 50 2f 32 2e 30  2f 55 44 50 20 31 39 32    SIP/2.0 /UDP 192
0060  2e 31 36 38 2e 31 2e 31  30 30 3a 35 30 36 30 3b   .168.1.1 00:5060;
0070  62 72 61 6e 63 68 3d 7a  39 68 47 34 62 4b 2d 31   branch=z 9hG4bK-1
0080  0d 0a 4d 61 78 2d 46 6f  72 77 61 72 64 73 3a 20   ··Max-Fo rwards: 
0090  37 30 0d 0a 54 6f 3a 20  3c 73 69 70 3a 75 73 65   70··To:  <sip:use
00a0  72 40 65 78 61 6d 70 6c  65 2e 63 6f 6d 3e 0d 0a   r@exampl e.com>··
00b0  46 72 6f 6d 3a 20 3c 73  69 70 3a 75 73 65 72 40   From: <s ip:user@
00c0  65 78 61 6d 70 6c 65 2e  63 6f 6d 3e 3b 74 61 67   example. com>;tag
00d0  3d 31 32 33 34 0d 0a 43  61 6c 6c 2d 49 44 3a 20   =1234··C all-ID: 
00e0  31 32 33 34 35 36 37 38  39 30 40 31 39 32 2e 31   12345678 90@192.1
00f0  36 38 2e 31 2e 31 30 30  0d 0a 43 53 65 71 3a 20   68.1.100 ··CSeq: 
0100  31 20 52 45 47 49 53 54  45 52 0d 0a 43 6f 6e 74   1 REGIST ER··Cont
0110  61 63 74 3a 20 3c 73 69  70 3a 75 73 65 72 40 31   act: <si p:user@1
0120  39 32 2e 31 36 38 2e 31  2e 31 30 30 3e 0d 0a 43   92.168.1 .100>··C
0130  6f 6e 74 65 6e 74 2d 4c  65 6e 67 74 68 3a 20 30   ontent-L ength: 0
```

4. CONCLUSION

This project provided a detailed exploration of network traffic generation, IP spoofing, and traffic analysis using various tools and techniques. Through the implementation of custom scripts, network administrators and cybersecurity professionals can simulate real-world network conditions, stress-test their environments, and understand the behavior of different types of traffic, including the challenges posed by malicious activities like IP spoofing.

The practical exercises demonstrated the effectiveness of tools like Nmap, nping, and Wireshark in generating, capturing, and analyzing network traffic. By utilizing a controlled virtual environment, the project allowed for safe experimentation with complex network scenarios that would otherwise be difficult to replicate in a real-world setting.

Additionally, the project highlighted the importance of understanding IP spoofing—a technique often used in cyber-attacks—to better defend against potential threats. The ability to generate and analyze spoofed traffic provided valuable insights into the mechanisms of network security and the significance of robust defensive measures.

In conclusion, the project not only served as a technical exercise in network traffic generation and analysis but also reinforced the critical importance of security in networked systems. The knowledge and skills acquired through this project will be invaluable in the continued study and practice of cybersecurity, ensuring that the next generation of network professionals is well-equipped to protect and manage modern IT infrastructures.

5. REFERENCES

- *Kali Linux Revealed*. Available at: https://kali.training/
- *Wireshark User Guide*. Available at: https://www.wireshark.org/docs/wsug_html_chunked/
- *Scapy Documentation*. Available at: https://scapy.readthedocs.io/
- Lyon, G. (2009). *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure.Org. Available at: https://nmap.org/book/
- *Python Software Foundation*. (2024). *Python Documentation*. Available at: https://docs.python.org/
- Postel, J. (1981). *RFC 791: Internet Protocol*. Available at: https://tools.ietf.org/html/rfc791
- *Mware Workstation Pro Documentation*. Available at: https://docs.vmware.com/en/VMware-Workstation-Pro/index.html
- Biondi, P. (2007). *Packet Crafting with Scapy*. Available at: https://www.secdev.org/conf/scapy