Network Intrusion Detection System (NIDS)

Sumit Chauhan (22103146)
Nit Jalndhar (CSE)

Contents

Abstract

In our project we are looking forward to developing a Network Intrusion Detection System (NIDS) which it is a small cybersecurity project aimed at protecting, and monitoring small networks, like that of homes or small office networks. Where nowadays cyberattacks are increasing in both frequency and sophistication, and there are many entry points into networks that are insufficiently locked down. In this project we have developed intrusion detection system using raspberry pi which is fast and convenient detection as well as response system which is able to detect multiple types of threats or attacks. In this project we will be familiar with tools such as scapy which provide traffic analysis and iptables which store and block automatic IP blocking.

1.0 Introduction

In our project, we are looking forward to developing a Network Intrusion Detection System (NIDS) which is a small cybersecurity project aimed at protecting and monitoring small networks, that of homes or small office networks. Nowadays cyberattacks have become the most dangerous part of the network and every day the number of attacks that occur increases since the network isn't secure enough. Therefore, for our project, we have developed an intrusion detection system using Raspberry Pi which is fast and convenient in detecting various types of attacks when they occur as well as a response system which blocks the attacker's IP address to prevent it from continuing the attack. In this project, we will be familiar with tools such as scapy which provides traffic analysis, iptables which stores and blocks automatic IP blocking, and Hydra and Ettercap which helped us simulate the attacks.

1.1 Motivation

The motivation behind this project is to develop an effective tool that can detect and log some common cyber-attacks. To improve network security against suspicious activities in real-time and risk mitigation. It will identify threats like unauthorized access through SSH, malicious scans, brute-force attempts, as well as ARP poisoning attacks simply by monitoring the network traffic. Once a threat is detected, it can automatically block the attack source, log this on an event log file, and alert a network administrator. This NIDS is user-friendly to even users with minimal technical experience. Its menu-driven interface for easy administrative use in security tasks (e.g., viewing logs, blocking / unblocking IPs, starting packet monitoring).

1.2 Problem Statement

The key issue our team is concerned with is unauthorized access or rather cyber threats in computer networks. While most of these cyber-attacks have now become intelligent, traditional methods of security are incapable of detecting such emerging threats and mostly lead to network breaches. It is in this regard that the project entitled **"Network Intrusion Detection System"** is being proposed for its enhancement to continuously monitor the network traffic for any abnormality, and detect any probable intrusion in real-time, thereby providing proactive alerts. This will, therefore, reduce the risk of cyber-attacks, hence protecting sensitive data and reducing potential financial and operational damage by enhancing network security.

1.3 Literature Review

Network Intrusion Detection Systems (NIDS) play a vital role in ensuring network security. Multiple research studies have explored various methodologies and assessed the effectiveness of different NIDS implementations. Existing systems typically use either signature-based or anomaly-based approaches. Signature-based systems, such as Snort and Suricata, rely on predefined patterns to detect known threats with high accuracy, making them reliable for familiar attack scenarios. However, their dependence on frequent updates and predefined rules renders them ineffective against zero-day attacks and novel threats.

In contrast, anomaly-based systems leverage statistical models or machine learning algorithms to identify deviations from typical network behaviour. These systems are highly adaptable and capable of detecting unknown threats, which makes them suitable for dynamic and complex network environments. Nevertheless, they are often prone to high false-positive rates and require extensive datasets for training, which can be resource-intensive. Hybrid systems, combining signature-based and anomaly-based methods, aim to overcome individual limitations by improving detection accuracy and flexibility. However, these systems face challenges such as increased computational demands and complexity in deployment, particularly in resource-constrained settings.

2.0 Design

2.1 Background

Security from the network perspective is about detecting and analysing the network traffic, where the administrator can gain useful information by detecting and analysing all possible threats. In this project, the focus is on three major attacks that may occur in the network:

1. SSH Brute Force Attacks:

A brute force attack that focuses on trying all possible combinations where the attacker will try everything possible to guess the username and password to gain unauthorized access to the victim's view. Particularly ssh on port 22.

2. Nmap Scans:

Nmap is a tool used for ethical hacking and attacking networks, where it's used to identify and scan open ports in the network, also the attacker can techniques such as SYN, Null, and XMAS scans to gather information about the victim.

3. ARP Poisoning:

Address Resolution Protocol (ARP) poisoning it's a method used by the attacker where the goal is to intercept the traffics and spoof the MAC address of the victim, where it's considered as a weakness in local networks, and it can result a man-in-the-middle (MITM) attack. Tools such as scapy will be used to analyse network traffics. Also, iptables to update firewall rules. To ensure real-time detection and mitigation of threats.

2.2 Experimental Setup

The experimental setup includes a Raspberry Pi acting as an intrusion detection unit and two machines, an attacker machine and a victim machine. The goal is to have a complete environment to detect and understand the attacks.

2.3 Hardware

Raspberry Pi: The Raspberry Pi 4 was used to develop our intrusion detection system. Where it will provide us with maintainability and sufficient power for small security applications.

In the raspberry pi, there will be 2 taps or machines which are:

1. **Attacker Machine:** In this machine, the attack will be launched which will target the victim machine.
2. **Attacker Machine:** In this machine, the victim will be targeted by the attacker machine.

2.4 Software

Python: Our system uses Python 3 programming language, where we used in Python a library called Scapy that can capture the packets.

iptables: A Linux firewall used to block and detect specific malicious IP addresses.

Operating System: The used OS is Linux-based which is used by Raspberry Pi.

2.5 Configuration

In the configuration process, the Raspberry Pi was placed in the same subnet of the attacker machine and the victim machine, to allow the Raspberry Pi to capture the packets. Also, the configuration was made to ensure proper packet routing between devices. Lastly, static IP addresses were assigned to all devices.

3.0 Experimental Testing and Results

Detail the procedure used to experiment step-by-step. The instruction sheet, together with the instructions given to you by the laboratory instructor, will be of help here. Sufficient information should be provided to allow the reader to repeat the experiment identically. Special procedures used to ensure specific experimental conditions, or to maintain a desired accuracy in the information obtained should be described.

3.1 First Exercise: Detecting SSH Brute Force Attacks

In the implementation section using Python 3, there is a dictionary to track failed login attempts by looking for the IP address. If the IP address exceeds the maximum attempts for instance 5 attempts, that IP will be blocked using iptables. The following code achieves this logic:

```
if dst_port == 22:
    clean_expired_logins()
    if src_ip not in failed_logins:
        failed_logins[src_ip] = {'count': 0, 'last_attempt':
datetime.now()}
    failed_logins[src_ip]['count'] += 1
    failed_logins[src_ip]['last_attempt'] = datetime.now()
    log_threat(f"{src_ip} -> {dst_ip}, Port: {dst_port}",
"Potential SSH Brute Force")
```

This code ensures that the secure and non-malicious traffic will not be detected by resetting outdated entries in the failed_logins log file. And ensures that it will detect malicious behaviour.

3.2 Second Exercise: Detecting Nmap Scans

In the second exercise, Nmap scans will be done by using different types of scans such as (SYN, Null, and XMAS) where it has been run from the attacker machine. Where the detection analyses the TCP flag in the packets that have been captured to identify these scans. The following code shows the implementation:

```
if tcp_flags == "S":  # SYN scan
    log_threat(f"SYN scan detected from {src_ip} to port
{dst_port}", "Nmap SYN Scan")
    #block_ip(src_ip)
elif tcp_flags == "FPU":  # XMAS scan (FIN + PUSH + URG flags)
    log_threat(f"XMAS scan detected from {src_ip}", "Nmap XMAS
Scan")
    #block_ip(src_ip)
elif tcp_flags == "":  # Null scan (no flags set)
    log_threat(f"Null scan detected from {src_ip}", "Nmap Null
Scan")
```

There are different types of scans and each one as shown has different and unique flag combinations, to enable and ensure accurate detection. Moreover, the system will log these activities and incidents, which aim to provide valuable insights into the captured network activities.

3.3 Third Exercise: Detecting ARP Poisoning

In the third exercise, an ARP poisoning has been simulated using tools such as ARP-Spoof. In which the detection system compared incoming ARP packets with the predefined legitimate MAC-IP pairs. And any discrepancy will be flagged as a potential attack.

```
if packet.haslayer(scapy.ARP):
arp_src_ip = packet[scapy.ARP].psrc
arp_src_mac = packet[scapy.ARP].hwsrc

# If the source IP is already in the ARP table, check for a MAC
address mismatch
if arp_src_ip in arp_table:
    if arp_table[arp_src_ip] != arp_src_mac:
        log_threat(f"ARP Poisoning detected: {arp_src_ip} is
claiming to be {arp_src_mac}", "ARP Poisoning")
Else:
    # Add the IP and MAC to the ARP table
    arp_table[arp_src_ip] = arp_src_mac
```

4.0 Results and Discussion

In this section of the report, the results of the in-depth analysis that was obtained during the experiments will be provided. To test the system's reliability and accuracy all the exercises were tested several times under different conditions. In addition to that, to identify the strengths, limitations, and areas for improvement all the data that was collected was analysed.

4.1 First Exercise: Detecting SSH Brute Force Attacks

The NIDS has successfully identified and mitigated the brute force attack that was targeting the victim machine's SSH service. Brute force attacks could be simulated using tools such as Hydra where it will repeatedly guess the login credentials. The system detected the attack while monitoring the traffic on port 22 where it identified the failed login attempts and as a mitigation blocked the malicious IPs dynamically.

Observed Workflow and Results

In our testing process the system was recording incorrect attempts rate, where it was approximately five attempts per second. Moreover, the system recorded the logs of the failed attempts associated with source IP address and timestamp. The IP address will be blocked if 5 attempts within a 5-minute has been done, where it will be blocked using the following iptables command:

```
sudo iptables -A INPUT -s <malicious_ip> -j DROP
```

Example log entry for a detected brute force attack:

```
[2024-11-15 12:01:00] 192.168.1.102 -> 192.168.1.24, Port: 22
| Attack Type: SSH Brute Force

[2024-11-15 12:01:05] Blocked IP: 192.168.1.102
```

Strengths of the Detection System

After testing our system shows high accuracy in differentiating the brute force attack from the legitimate traffic, where the goal has been accomplished which shows how the implementation threshold for failed login attempts.

Our intrusion detection system shows the ability to automatically block any malicious IP addresses, in which these IP addresses have been detected because they were initiating the brute force attack.

Challenges and Limitations:

The threshold should be chosen carefully, because any legitimate user may get blocked if they reach the maximum number of failed login attempts.

System efficiency and reliability may decrease if many traffic, or high rate of traffic has been done.

Potential Improvements:

- Doing a whitelist which will store the trusted IP address in the network, to prevent the possibility of blocking trusted IP address.

- Applying a mechanism in which the adaptive thresholds based on traffic patterns, to enhance the overall intrusion detection.

4.2 Second Exercise: Detecting Nmap Scans

Our intrusion detection system shows the ability of detecting and identifying the use of NMAP tool which it includes SYN, Null, and XMAS scans. Where these scans can be runed by the attacker machine to scan any open ports in the victim machine, to detect such event, we need to analyse TCP flags in network packets, and finally detect which time is it and log the incident.

Observed Workflow and Results:

Each scan type was executed separately, and the system's ability to detect and log them was evaluated. The detection logic categorised scans as follows:

The scans each one has been executed separately, and the system able to evaluate and understand this type. Based on the following:

- **SYN Scanning:** It can be identified by the TCP packets in which the SYN flag is present.
- **Null Scanning:** If there is no flag or it's not sited in the tcp header it can be detected.
- **XMAS Scanning:** If the combination of FIN, PUSH, and URG flags was there, then it can be detected.

Example log entries for detected scans:

```
[2024-11-15 12:02:00] SYN scan detected from 192.168.1.103 to port 22 | Attack Type: Nmap SYN Scan

[2024-11-15 12:03:00] XMAS scan detected from 192.168.1.104 | Attack Type: Nmap XMAS Scan

[2024-11-15 12:04:00] Null scan detected from 192.168.1.105 | Attack Type: Nmap Null Scan
```

Strengths of the Detection System:

- The system provides accurate log for the scans in which it will display and identify their execution within seconds, which it is considered one of the most important points in any intrusion detection system.
- Our intrusion detection system can differentiate between scanning types which enhances the system's ability to monitor traffic and networking events.

Challenges and Limitations:

- The professional attacker can encrypt or fragment packets to evade detection. This issue can be addressed by using deeper inspection techniques to network packets.
- Continually capturing and analysing TCP packets can lead to resource consuming and waste on the Raspberry Pi.

Potential Improvements:

- Including in the detection mechanism other protocols such as UDP or ACK scan, which it will improve the detection
- Integrating machine learning to detect and identify anomalous traffic patterns that can be deviate from normal behaviour.

4.3 Third Exercise: Detecting ARP Poisoning

The detection of ARP poisoning attempts involved monitoring ARP packets and comparing MAC-IP mappings against a legitimate table. During testing, the attacker used ARP-spoof to impersonate the victim machine, attempting to redirect traffic through the attacker's machine.

Observed Workflow and Results:

The system shows the ability of detecting anomalies in ARP packets in which the MAC address did not map with the expected source IP address. The user was alerted and announced for each detected anomaly. Example log entry:

```
[2024-11-15 12:05:00] ARP Poisoning detected: 192.168.1.24 is claiming to be MAC 00:11:22:33:44:55 | Attack Type: ARP Spoofing
```

Strengths of the Detection System:

- The system could identify spoofing attempts, where the reason is having an accurate ARP table.
- The system will store Legitimate MAC-IP address pairs in the ARP table dynamically, which will reduce any type of manual configuration.

Challenges and Limitations:

- Initially the ARP table must be initialized with the right and accurate data to avoid any type of errors or false positives.
- Maintaining and verifying ARP tables in any large network will be both resource-intensive and time-consuming.

Potential Improvements:

- Applying a mechanism to initialize and prepare ARP table configuration automatically, by learning mappings over time.
- Applying an active type of defence such as ARP packet rejection, to prevent and neutralize any type of spoofing attempts.

5.0 Conclusion

5.1 Summary

In Summary, it proves that developing a NIDS using a Raspberry Pi is effective and practical for small networks. It provides an affordable way for users to monitor network traffic, detect common threats such as Nmap scans, SSH brute-force attacks, and ARP poisoning, and log these events. The system features automated responses, such as blocking malicious IPs and logging traffic, ensuring proactive security without manual intervention. Its simplicity and user-friendly interface make it ideal for small environments, like homes or small offices, where it enhances network security with minimal setup. While the system shows high speed, reliability, and accuracy, its drawbacks include limitations in handling advanced attacks, scalability, and high attack traffic, reducing the effectiveness of the system. What's more, setting the predefined thresholds manually needs attention with care. However, this limited design of the system itself accommodates easy updates and developments to overcome such challenges.

Key Takeaways

1. The system can monitor network traffic and detect events in real time, making it a solid security solution for small-area networks.
2. Automated responses enhance security by minimizing manual configuration and enabling automatic threat detection.
3. The system's simple design allows for easy updates and modifications.

Limitations:

The system faces challenges with advanced attacks, scalability, and handling of high-attack traffic. The need for manual configuration for predefined thresholds and tables requires careful management.

5.2 Future Work

- **Security Upgrades:** A packet inspection mechanism can enhance the overall security of the intrusion detection system which will increase the ability to filter the packets in a detailed manner and provide accurate outputs. Which the end will let it able to manage a large network area.

- **Machine Learning Development**: Configuring new algorithms that can understand and identify unknown attack patterns and include new attack detection.
- **User-Friendly Interface**: building a simple user-friendly interface to be suitable for all users even if they don't have a huge knowledge of networking and network security.

5.3 Lessons Learned

During this project, we have learned different aspects of cybersecurity which will help us in our future carers whenever we work on securing a network. We have done thorough research to understand how someone can target a network and what are some of the common attacks that could be launched on a network. In Addition, since we couldn't use Wire Shark for network traffic analysis we had to learn and understand how the scapy library works so we can integrate it into our NIDS. Overall, the project had lots of lessons that we learned, and it has had a huge impact on our future.

5.4 Team Dynamics

Our team was made up of three cybersecurity students and we distributed the work between us equally so that each one of us would choose a type of network attack, conduct research about how it works, how to prevent it and how we can launch an attack to test if our program works.

The Team Leader for the project was Ameen who guided us on what should be the first course of action we should take to start the project, Mohamed Ibrahim

was the backbone of the Team where he monitored and oversaw what type of articles or websites would be helpful to help us in creating our NIDS and Abed Alhameed was responsible for researching how we recreate an attack to test our program.

Overall, we all worked on creating the program for the system and all of us worked to achieve only one goal which was securing the network.

5.5 Impact Statement

The proposed Network Intrusion Detection System (NIDS) significantly benefits society and the economy:

Social Impact:

Operational, affordable, and available, the NIDS offers better security and privacy for individuals and small businesses. It puts users in a position to safeguard their networks against malicious activities, building trust in the digital systems. This encourages safer interactions over the internet and lowers the chances of financial and personal losses resulting from cyberattacks. Additionally, the system is built in a user-friendly fashion to be accessible by a non-technical person to close the awareness gap and accessibility of cybersecurity in the underserved community.

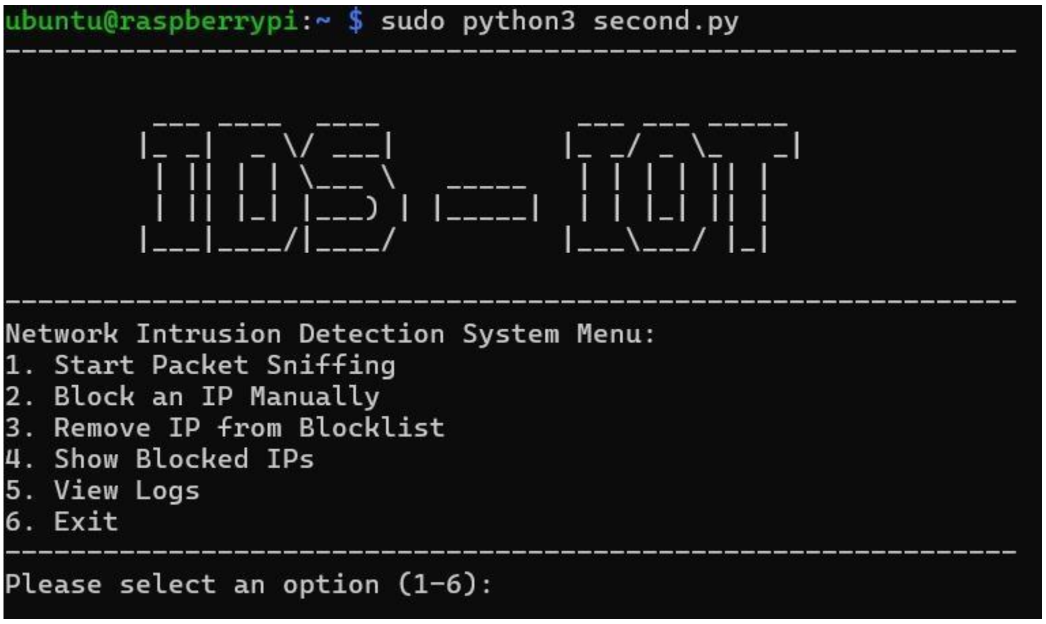
Economic Impact:

Low cost for the system will make it very appealing to small businesses and startups by allowing them to secure their operations without bearing the cost associated with expensive security solutions. Indirectly, it contributes to economic growth through decreased chances of cyberattacks and ensuing financial losses with disruption to operations. Though the system is not of a kind to provide direct employment opportunities, it supports small enterprises to keep their operations running securely and thereby sustains jobs for the economy. Besides, its use of inexpensive hardware, like the Raspberry Pi, reduces capital expenditure and makes advanced security technologies accessible to a wider audience.

This solution represents a socially and economically impactful approach to enhancing network security while remaining accessible to those with limited resources.

6.0 Appendix

6.1 Output Look



```
ubuntu@raspberrypi:~ $ sudo python3 second.py
--------------------------------------------------------------

   ___ ____  ____            ___ ___ _____
  |_ _|  _ \/ ___|          |_ _/ _ \_   _|
   | || | | \___ \   _____   | | | | || |
   | || |_| |___) | |_____|  | | |_| || |
  |___|____/|____/          |___\___/ |_|

--------------------------------------------------------------
Network Intrusion Detection System Menu:
1. Start Packet Sniffing
2. Block an IP Manually
3. Remove IP from Blocklist
4. Show Blocked IPs
5. View Logs
6. Exit
--------------------------------------------------------------
Please select an option (1-6):
```

6.2 Full Code

```python
import scapy.all as scapy  # The tool that's like Wireshark
import time
import os
from datetime import datetime

failed_logins = {}  # To record failed logins
blocked_ips = set()  # Record blocked IPs (to prevent duplicate
entries)
arp_table = {}  # To store expected MAC addresses for IPs
```

```
# Log detected threats to a file and print to the console
def log_threat(message, attack_type):
    log_entry = f"[{time.strftime('%Y-%m-%d %H:%M:%S')}] {message}
| Attack Type: {attack_type}"
    with open("logs.txt", "a") as log_file:
        log_file.write(log_entry + "\n")
    print(log_entry)


# Block an IP using the system firewall
def block_ip(ip):
    if ip not in blocked_ips:
        os.system(f"sudo iptables -A INPUT -s {ip} -j DROP")  #
Uses iptables in firewall to block the IP
        blocked_ips.add(ip)
        log_threat(f"Blocked IP: {ip}", "IP Blocked")


# Unblock an IP
def unblock_ip(ip):
    if ip in blocked_ips:
        os.system(f"sudo iptables -D INPUT -s {ip} -j DROP")
        blocked_ips.remove(ip)
        print(f"Unblocked IP: {ip}")
    else:
        print(f"IP {ip} is not blocked.")

    # Used to clear memory from old login attempts


def clean_expired_logins():
    current_time = datetime.now()
    for ip in list(failed_logins):
        if (current_time -
failed_logins[ip]['last_attempt']).total_seconds() > 300:
            del failed_logins[ip]


# Reset iptables and clear old blocked IPs
def reset_firewall():
    print("Resetting firewall rules...")
    os.system("sudo iptables -F")  # Flush all iptables rules
    blocked_ips.clear()
    print("Firewall rules cleared.")


# Detect potential threats, including SSH, ARP Poising, Nmap scans

def detect_threat(packet):
    if packet.haslayer(scapy.IP) and packet.haslayer(scapy.TCP):  #
If belonging to TCP IP
        src_ip = packet[scapy.IP].src
        dst_ip = packet[scapy.IP].dst
        dst_port = packet[scapy.TCP].dport
        tcp_flags = packet[scapy.TCP].flags

        # Ignore packets from blocked IPs
        if src_ip in blocked_ips:
            return

        # Check for Nmap-specific scan types (there are three types
of NMAP Scans: SYN, XMAS, Null)
        if tcp_flags == "S":  # SYN scan
            log_threat(f"SYN scan detected from {src_ip} to port
 {dst_port}", "Nmap SYN Scan")
            #block_ip(src_ip)
        elif tcp_flags == "FPU":  # XMAS scan (FIN + PUSH + URG
flags)
            log_threat(f"XMAS scan detected from {src_ip}", "Nmap
XMAS Scan")
            #block_ip(src_ip)
        elif tcp_flags == "":  # Null scan (no flags set)
            log_threat(f"Null scan detected from {src_ip}", "Nmap
Null Scan")
            # block_ip(src_ip)

        # Check if destination port was 22 (SSH) for brute-force
attempts
        if dst_port == 22:
            clean_expired_logins()
            if src_ip not in failed_logins:
                failed_logins[src_ip] = {'count': 0,
'last_attempt': datetime.now()}
            failed_logins[src_ip]['count'] += 1
            failed_logins[src_ip]['last_attempt'] = datetime.now()
            log_threat(f"{src_ip} -> {dst_ip}, Port: {dst_port}",
"Potential SSH Brute Force")

    # Check for ARP Poisoning
    if  packet.haslayer(scapy.ARP):
        arp_src_ip = packet[scapy.ARP].psrc
        arp_src_mac = packet[scapy.ARP].hwsrc

        # If the source IP is already in the ARP table, check for a
MAC address mismatch
        if arp_src_ip in arp_table:
            if arp_table[arp_src_ip] != arp_src_mac:
                log_threat(f"ARP Poisoning detected: {arp_src_ip}
is claiming to be {arp_src_mac}", "ARP Poisoning")
        else:
            # Add the IP and MAC to the ARP table
            arp_table[arp_src_ip] = arp_src_mac


# Start sniffing on a specified network interface
def start_sniffing(interface):
    global failed_logins
    reset_firewall()  # Reset iptables rules and clear blocked IPs
    failed_logins = {}  # Reset failed logins when sniffing starts
    print("Starting packet sniffing on interface: " + interface)
    scapy.sniff(iface=interface, store=False, prn=detect_threat)


def show_blocked_ips():
    if blocked_ips:
        print("Blocked IPs:")
        for ip in blocked_ips:
            print(ip)
    else:
        print("No IPs are currently blocked.")


def view_logs():
    try:
        with open("logs.txt", "r") as log_file:
            for log in log_file:
                print(log.strip())
    except FileNotFoundError:
        print("No logs found.")

def show_menu():
    while True:

        print("------------------------------------------------------------------------------------------------------
----------")
        print("""

         ___ ____  ____            ___ ___ _____
        |_ _|  _ \/ ___|          |_ _/ _ \_   _|
         | || | | \____\   ______  | | | | || |
         | || |_| |___) | |_____|  | | |_| || |
        |__|_____/|____/          |__\___/ |_|
        """)
        print("------------------------------------------------------------------------------------------------------
----------")
        print("Network Intrusion Detection System Menu:")
        print("1. Start Packet Sniffing")
        print("2. Block an IP Manually")
        print("3. Remove IP from Blocklist")
        print("4. Show Blocked IPs")
        print("5. View Logs")
        print("6. Exit")
        print("------------------------------------------------------------------------------------------------------
----------")

        choice = input("Please select an option (1-6): ")

        if choice == "1":
            interface = input("Enter the network interface to
monitor (e.g., eth0, wlan0): ")
            start_sniffing(interface)
        elif choice == "2":
            ip_to_block = input("Enter the IP address to block: ")
            block_ip(ip_to_block)
        elif choice == "3":
            ip_to_unblock = input("Enter the IP address to unblock:
")
            unblock_ip(ip_to_unblock)
        elif choice == "4":
            show_blocked_ips()
        elif choice == "5":
            view_logs()
        elif choice == "6":
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please select a valid option.")

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