RULE BASED INTRUSION DETECTION SYSTEM in PC

ST. TERESA'S COLLEGE(AUTONOMOUS)

AFFILIATED TO MAHATMA GANDHI UNIVERSITY

PROJECT REPORT

In partial fulfilment of the requirements for the award of the degree of

BCA (CLOUD TECHNOLOGY & INFORMATION SECURITY MANAGEMENT)

By

DEENA MORAIS- SB21BCA009

&

HIMAPRIYA M- SB21BCA018

III DC BCA (CLOUD TECHNOLOGY & INFORMATION SECURITY MANAGEMENT)

Under the guidance of

Ms Sreelakshmy I J

DEPARTMENT OF BCA (Cloud Technology & Information Security Management)

MARCH 2024

RULE BASED INTRUSION DETECTION SYSTEM in PC

ST. TERESA'S COLLEGE(AUTONOMOUS)

AFFILIATED TO MAHATMA GANDHI UNIVERSITY

PROJECT REPORT

In partial fulfilment of the requirements for the award of the degree of

BCA (CLOUD TECHNOLOGY & INFORMATION SECURITY MANAGEMENT)

By

DEENA MORAIS- SB21BCA009

&

HIMAPRIYA M- SB21BCA018

III DC BCA (CLOUD TECHNOLOGY & INFORMATION SECURITY MANAGEMENT)

Under the guidance of

Ms Sreelakshmy I J

DEPARTMENT OF BCA (Cloud Technology & Information Security Management)

MARCH 2024

DECLARATION

We, undersigned, hereby declare that the project report, **"RULE BASED INTRUSION DETECTION SYSTEM in PC"**, submitted for partial fulfillment of the requirements for the award of degree of BCA (Cloud Technology and Information Security Management)at St. Teresa's College (Autonomous),Ernakulam (Affiliated to Mahatma Gandhi University), Kerala, is a bonafide work done by us under the supervision of Ms Sreelakshmy I J. This submission represents our ideas in our own words and where ideas or words of others have not been included. We have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Ernakulam
March 2024

Deena Morais – SB21BCA009
Himapriya M – SB21BCA018

ST. TERESA'S COLLEGE (AUTONOMOUS), ERNAKULAM

BCA (CLOUD TECHNOLOGY & INFORMATION SECURITY MANAGEMENT)

DEPARTMENT OF BCA (Cloud Technology & Information Security Management)

ST. TERESA'S COLLEGE
(AUTONOMOUS)
ESTD 1925

CERTIFICATE

This is to certify that the report entitled "**RULE BASED INTRUSION DETECTION SYSTEM in PC**", submitted by Deena Morais to the Mahatma Gandhi University in partial fulfillment of the requirements for the award of the Degree of BCA (Cloud Technology & Information Security Management) is a bonafide record of the project work carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

20/03/2024

Ms. Sreelakshmy I J
Internal Supervisor

Ms. Archana Menon P
Head of the department

External Supervisor

ST. TERESA'S COLLEGE
DEPT. OF COMPUTERS
ERNAKULAM

ACKNOWLEDGEMENT

First and foremost we thank God Almighty for his blessings. We take this opportunity to express our gratitude to all those who helped us in completing this project successfully. I wish to express our sincere gratitude to the **Manager Rev. Dr. Sr. Vinitha CSST** and the Principal **Dr. Alphonsa Vijaya Joseph** for providing all the facilities.

We express our sincere gratitude towards the Head of the department **Ms. Archana Menon P** for the support. We deeply express sincere thanks to our project guide **Ms. Sreelakshmy I J** for her proper guidance and support throughout the project work.

We are indebted to our beloved teachers whose cooperation and suggestion throughout the project which helped us a lot. We thank all our friends and classmates for their support.

We convey our hearty thanks to our parents for the moral support, suggestion and encouragement.

ABSTRACT

In the realm of Internet of Things (IoT), ensuring the security of interconnected devices is paramount. This paper presents a Host-Based Intrusion Detection System (HIDS) crafted in Python and flask framework, designed specifically for real-time threat detection in IoT environments. The system encompasses lightweight sensor agents deployed on individual devices, a detection engine leveraging rule-based technique, an alerting system to notify administrators of potential threats, and logging/reporting capabilities for forensic analysis. By continuously monitoring device activities, analyzing behavior patterns, and promptly responding to detected threats, this Python-based HIDS fortifies the security posture of IoT devices, thereby mitigating risks and ensuring the robustness of IoT ecosystems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBRIVATIONS

CHAPTER 1
INTRODUCTION

1.1 HOST BASED INTRUSION DETECTION SYSTEM (HIDS)

HIDS stands for Host-based Intrusion Detection System. It's a type of security software that monitors and analyzes the internals of a computing system, including the operating system, applications, and file systems, for signs of malicious activities or policy violations.

Unlike network-based intrusion detection systems (NIDS) which monitor network traffic for suspicious activities, HIDS focuses on individual host systems, making it capable of detecting attacks that originate from both internal and external sources.

HIDS works by comparing observed system events and behaviors against a known database of signatures or predefined rules. When it identifies a pattern or behavior that matches those of known attacks or suspicious activities, it generates alerts or takes predefined actions to mitigate the threat, such as terminating a process, isolating the host, or alerting administrators.

HIDS can play a crucial role in maintaining the security and integrity of a system by providing real-time monitoring and detection capabilities to identify and respond to security incidents promptly.

1.2 INFORMATION SECURITY

Information security, often abbreviated as InfoSec, refers to the practice of protecting sensitive information from unauthorized access, disclosure, alteration, and destruction. It encompasses various measures, processes, and technologies designed to safeguard data, systems, networks, and other assets from security threat

1.3 PRINCIPLES OF INFORMATION SECURITY

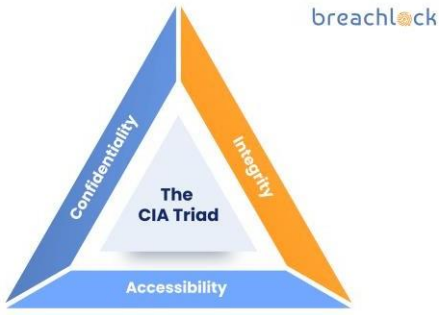


Fig: 1.1 CIA Triad

1.3.1 Confidentiality

Confidentiality, in the context of the CIA triad, refers to the principle of ensuring that information is only accessible to authorized individuals or entities. It involves protecting data from unauthorized disclosure or access.

Confidentiality measures aim to prevent sensitive information from falling into the wrong hands, whether intentionally or accidentally. This can include implementing access controls, encryption techniques, and secure communication protocols to safeguard data from unauthorized viewing, copying, or modification.

Maintaining confidentiality is crucial for protecting sensitive information such as personal data, trade secrets, financial records, and intellectual property. It helps preserve the privacy, trust, and integrity of individuals, organizations, and systems.

1.3.2 Integrity

In the context of the CIA triad, integrity refers to the principle of maintaining the accuracy, consistency, and trustworthiness of data and systems. It involves ensuring that information remains unaltered and reliable throughout its lifecycle, from creation to disposal.

Integrity measures aim to prevent unauthorized or malicious modification, deletion, or corruption of data. This can include implementing mechanisms such as digital signatures, checksums, access controls, and data validation techniques to detect and protect against unauthorized changes.

By upholding data integrity, organizations can trust that their information is accurate, reliable, and trustworthy. This is critical for ensuring the validity of transactions, protecting against data tampering or corruption, and maintaining the overall integrity and credibility of systems and processes.

1.3.3 Availability

Availability, within the CIA triad, refers to the principle of ensuring that information and resources are accessible and usable when needed by authorized users. It involves maintaining a reliable and timely access to data, systems, and services, even in the face of disruptions, failures, or attacks.

Availability measures aim to prevent or mitigate disruptions to access caused by factors such as hardware failures, software bugs, network outages, or malicious attacks. This can include implementing redundancy, fault tolerance, disaster recovery plans, and robust network architectures to ensure continuous operation and quick recovery from incidents.

Ensuring availability is crucial for supporting business operations, delivering services to users, and meeting service level agreements (SLAs). It helps prevent downtime, productivity losses, and service interruptions, ensuring that critical systems and information remain accessible and responsive to users' needs.

1.4 IoT Internet of things

The Internet of Things (IoT) refers to the network of physical objects or "things" embedded with sensors, software, and other technologies that enable them to connect and exchange data with other devices and systems over the internet. These objects can range from everyday items

like household appliances and wearable devices to industrial machinery and infrastructure components. IoT enables these objects to collect and exchange data, leading to increased efficiency, automation, and insights for various applications across industries.

The Internet of Things (IoT) refers to a network of physical devices, vehicles, appliances, and other physical objects that are embedded with sensors, software, and network connectivity, allowing them to collect and share data.

IoT devices—also known as "smart objects"—can range from simple "smart home" devices like smart thermostats, to wearables like smartwatches and RFID-enabled clothing, to complex industrial machinery and transportation systems. Technologists are even envisioning entire "smart cities" predicated on IoT technologies.

IoT enables these smart devices to communicate with each other and with other internet-enabled devices. Like smartphones and gateways, creating a vast network of interconnected devices that can exchange data and perform various tasks autonomously. This can include:

- monitoring environmental conditions in farms

- managing traffic patterns with smart cars and other smart automotive devices

- controlling machines and processes in factories

- tracking inventory and shipments in warehouses

The potential applications of IoT are vast and varied, and its impact is already being felt across a wide range of industries, including manufacturing, transportation, healthcare, and agriculture. As the number of internet-connected devices continues to grow, IoT is likely to play an increasingly important role in shaping our world. Transforming the way that we live, work, and interact with each other.

In an enterprise context, IoT devices are used to monitor a wide range of parameters such as temperature, humidity, air quality, energy consumption, and machine performance. This data can be analyzed in real time to identify patterns, trends, and anomalies that can help businesses optimize their operations and improve their bottom line.

1.5 Cyber attack

A cyber attack is any malicious attempt to gain unauthorized access to a computer, computing system or computer network with the intent to cause damage. Cyber attacks aim to disable, disrupt, destroy or control computer systems or to alter, block, delete, manipulate or steal the data held within these systems.

How do cyber attacks work?

Threat actors use various techniques to launch cyber attacks, depending in large part on whether they're attacking a targeted or an untargeted entity.

In an untargeted attack, where the bad actors are trying to break into as many devices or systems as possible, they generally look for vulnerabilities in software code that enable them to gain access without being detected or blocked. Or, they might employ a phishing attack, emailing large numbers of people with socially engineered messages crafted to entice recipients to click a link that downloads malicious code.

In a targeted attack, the threat actors are going after a specific organization and the methods used vary depending on the attack's objectives. The hacktivist group Anonymous, for example, was suspected in a 2020 distributed denial-of-service attack (DDoS) on the Minneapolis Police Department website after a man died while being arrested by Minneapolis officers. Hackers also use spear-phishing campaigns in a targeted attack, crafting emails to specific individuals who, if they click included links, would download malicious software designed to subvert the organization's technology or the sensitive data it holds.

Cybercriminals often create the software tools to use in their attacks, and they frequently share those on the dark web. Cyber attacks often happen in stages, starting with hackers surveying or scanning for vulnerabilities or access points, initiating the initial compromise and then executing the full attack -- whether it's stealing valuable data, disabling the computer systems or both

1.5.1 Types of cyber attack

ping Flood (ICMP Flood):

- In this type of attack, the attacker sends a large volume of ICMP echo request packets (ping) to a target system, overwhelming its resources and causing it to become unresponsive to legitimate traffic.

SYN Flood

- SYN flood attacks exploit the TCP three-way handshake process. The attacker sends a large number of TCP SYN packets to the target system, but does not complete the handshake by sending the final ACK packet. This causes the target system to consume resources while waiting for the final ACK, eventually leading to a denial of service.

ARP Spoofing (Address Resolution Protocol)

- ARP spoofing attacks involve sending falsified ARP messages over a local area network. By doing so, the attacker associates their MAC address with the IP address of a legitimate system on the network, causing traffic intended for that system to be redirected to the attacker's system.

DDoS (Distributed Denial of Service)

- DDoS attacks involve flooding a target system or network with a large volume of traffic from multiple sources, making it unavailable to handle legitimate requests. DDoS attacks can employ various techniques, including ICMP floods, SYN floods, UDP floods, and HTTP floods, among others.

DNS Spoofing (Domain Name System)

- DNS spoofing attacks manipulate DNS responses to redirect users to malicious websites or other unintended destinations. By poisoning the DNS cache with false mappings of domain names to IP addresses, attackers can intercept and redirect traffic intended for legitimate servers.

Man-in-the-Middle (MITM)

- MITM attacks occur when an attacker intercepts communication between two parties, often without their knowledge. By positioning themselves between the sender and receiver, the attacker can eavesdrop on sensitive information, modify data packets, or impersonate one of the parties involved.

Smurf Attack

- Smurf attacks involve sending a large number of ICMP echo request packets (pings) to the broadcast address of a network, with the source IP address spoofed to appear as the victim's IP address. This causes multiple hosts on the network to respond to the victim, overwhelming its resources.

Port Scanning:

- Port scanning attacks involve probing a target system or network to identify open ports and services. Attackers use port scanning to gather information about potential vulnerabilities that can be exploited to gain unauthorized access or launch further attacks.

These are just a few examples of network attacks, and there are many other types and variations, each with its own specific characteristics and methods of exploitation.

CHAPTER 2

LITERATURE SURVEY

With the escalating volume of network traffic and the ever-evolving landscape of security threats, the field of intrusion detection systems (IDSs) has garnered significant attention across the computer science domain. While existing IDSs face challenges stemming from diverse intrusion categories and demanding computational requirements, there exists a substantial body of literature addressing IDS issues. However, to provide a more comprehensive understanding, we aim to offer an elaborate depiction through an extensive survey and well-organized taxonomy.

Through our meticulous review, we propose a taxonomy to delineate modern IDSs, aiming to provide a structured framework for categorizing and understanding these systems. Our taxonomy, accompanied by tables and figures summarizing key insights, facilitates a clearer understanding of the diverse landscape of IDSs. [1]

Intrusion detection is a modern approach aimed at enhancing the security of existing computer systems and data networks, enabling them to maintain their current operational mode while safeguarding against unauthorized access and misuse. The primary objective of intrusion detection is to identify and thwart unauthorized activities perpetrated by both internal users and external attackers.

The proliferation of heterogeneous computer networks has made intrusion detection increasingly challenging. The expanded connectivity of computer systems provides intruders with greater access and opportunities to evade detection. Intrusion detection systems (IDSs) are built on the premise that an intruder's behavior differs significantly from that of a legitimate user, and many unauthorized actions leave detectable traces.

IDSs typically employ two main models for intrusion detection: statistical anomaly detection and rule-based misuse detection. Statistical anomaly detection involves establishing baseline behavior patterns and flagging deviations from these norms as potential intrusions. Rule-based misuse detection, on the other hand, relies on predefined rules or signatures to identify known malicious activities. [2]

Nowadays, cloud computing has become an emerging and widely used technology throughout the world on account of its dynamic, reliable and customizable quality of service. Because of its dynamic, dependable, and adaptable quality of service, cloud computing has emerged as a popular and frequently utilized technology in the modern world. At the same time, though, industry and academia are becoming more and more interested in the security issue surrounding cloud environments. Digital forensics is a popular topic in many cloud computing security concerns. It is more difficult to use digital forensic in the cloud than it is on standard digital devices and hardware since it is highly challenging to gather logs from the cloud environment. The goal of this study is to provide a novel system design for handling digital forensics in cloud

environments. We employ a novel architecture to assist investigators in gathering logs. The introduction of the host-based intrusion detection system (HIDS) protects cloud data from malevolent intrusion attempts. Subsequently, one web server generates email notifications and Secure Shell (SSH) messages to prevent additional suspicious activities based on the feedback results of HIDS. Ultimately, trustworthy proof of the accused user can be gathered by the digital forensic investigators. Thus, in a cloud context, HIDS and log gathering will play a major role in digital forensic.[3]

This study presented an efficient attack response mechanism against Advanced Persistent Threats (APTs). The NIST Cyber Security Framework (CRF) was mentioned in order to provide the most economical solutions. In order to avoid APT, it has created a system that recognizes and reacts to real-time AM-HIDS (Anti Malware Host Intrusion Detection System) monitoring of abnormal changes SW of PCs. It has shown that even with the greatest government-managed security controls, a good cost-effective environment may be created to thwart APT attacks.[4] .

Because cloud computing gives each client access to massive resources, software, and information, it is more efficient and convenient for users than traditional on-premise computing. Nonetheless, a variety of cyberattacks can readily endanger cloud computing infrastructure. For this reason, a cloud computing system absolutely needs an Intrusion Detection System (IDS). The fact that typical host-based intrusion detection systems for cloud computing use a lot of system resources is a major issue. To cut down on resource usage, we present a centralized host-based intrusion detection system in this research. gathering the system logs from every virtual machine using the logstash tool, then centrally storing them in the elasticsearch cluster. Subsequently, we examine every one of these logs in the detection center and forward the findings to every virtual machine. We have validated our framework in the openstack platform. The results show a good performance in reducing the CPU and memory usage.[5]

Intrusion Detection Systems (IDSs) are crucial for identifying and responding to attacks in computer networks. These systems gather network traffic data and employ various techniques to enhance network security. One distinction lies between misuse-detection IDSs, which identify known attack patterns, and anomaly detection IDSs, which detect deviations from normal behavior, potentially indicating novel attacks.

This paper proposes a hybrid IDS that integrates both approaches into a single system. The hybrid IDS combines Packet Header Anomaly Detection (PHAD) and Network Traffic Anomaly Detection (NETAD), both of which are anomaly-based IDSs, with Snort, an open-source misuse-based IDS. By merging these methodologies, the hybrid IDS can detect both known and novel attacks effectively.

To assess the efficacy of the hybrid IDS, we evaluate it using the MIT Lincoln Laboratories network traffic data (IDEVAL) as a testbed. The evaluation compares the performance of the misuse-based IDS alone against the hybrid IDS. Results demonstrate that the hybrid IDS

outperforms the standalone misuse-based IDS, indicating its superiority as a more robust and comprehensive intrusion detection system. [6]

Smart environments aim to enhance human life by optimizing comfort and efficiency, with the Internet of Things (IoT) emerging as a pivotal technology in their development. However, security and privacy pose significant challenges in IoT-based smart environments, as vulnerabilities within IoT systems can be exploited, leading to security threats. Consequently, there is a pressing need for specialized Intrusion Detection Systems (IDSs) tailored for IoT environments to mitigate these security risks effectively. Traditional IDSs may not be suitable due to the constrained computing and storage capabilities of IoT devices, as well as the unique protocols they employ.

This article conducts a thorough survey of the latest IDSs designed specifically for the IoT model, focusing on their methodologies, features, and mechanisms. It delves into the intricacies of IoT architecture, highlighting emerging security vulnerabilities and their correlation with different layers of the IoT architecture. Despite previous research efforts in designing IoT-oriented IDSs, the article underscores the ongoing importance of developing efficient, reliable, and robust IDSs for IoT-based smart environments. The survey concludes by outlining key considerations for the future development of such IDSs, emphasizing the critical nature of this ongoing endeavor. [7]

The proliferation of Internet of Things (IoT) applications has led to a significant surge in network data volume, along with heightened computational demands on interconnected devices. Despite their ability to capture valuable information for critical decision-making, many IoT devices face resource constraints, including low CPU power, limited memory, and energy storage. Consequently, they are susceptible to cyber-attacks due to their inability to support conventional security software, posing inherent risks to IoT networks. To address these challenges, the emergence of multi-access edge computing (MEC) platforms aims to alleviate device constraints by offloading complex computing tasks to the edge.

While existing research predominantly focuses on optimizing security solutions for protecting IoT devices, we advocate for greater emphasis on distributed solutions leveraging MEC. This paper conducts a comprehensive review of cutting-edge network intrusion detection systems (NIDS) and security practices tailored for IoT networks. Our analysis encompasses approaches utilizing MEC platforms and employing machine learning (ML) techniques. Additionally, we perform a comparative examination of publicly available datasets, evaluation metrics, and deployment strategies used in NIDS design. Ultimately, we propose an NIDS framework for IoT networks that leverages MEC, contributing to the ongoing discourse on bolstering IoT security. [8]

The proliferation of Internet-connected devices in the IoT landscape has raised significant concerns regarding security and privacy. These concerns act as major barriers to the widespread adoption of IoT technology, impacting consumers, organizations, and governmental entities alike. While the prevention of all attacks on IoT systems may be unattainable, the ability to

detect and respond to security breaches in real-time is crucial for effective defense. In response to this imperative, our paper presents a novel intrusion detection system (IDS) leveraging machine learning algorithms specifically tailored for detecting security anomalies within IoT networks.

This paper proposes IDS serves as a security-as-a-service platform, offering real-time detection capabilities and promoting interoperability among diverse network communication protocols utilized in IoT environments. The framework outlined in this paper elucidates the intricate process of intrusion detection within IoT networks, emphasizing the role of machine learning algorithms in identifying and mitigating security threats. By enhancing the ability to detect anomalies promptly, our system aims to bolster the security posture of IoT ecosystems, fostering greater confidence and trust in the deployment and utilization of IoT technology. [9]

Rule-based intrusion detection techniques rely on observing system events and applying predefined rules to determine whether certain patterns of activity are suspicious. These approaches can be categorized as either anomaly detection or penetration identification, although there is some overlap between them. Rule-based anomaly detection involves analyzing historical audit records to identify usage patterns and automatically generate rules that describe these patterns. Current behavior is then compared against these rules to detect deviations from normal behavior, similar to statistical anomaly detection methods. This approach doesn't require knowledge of specific security vulnerabilities within the system.

On the other hand, rule-based penetration identification employs expert system technology to identify known penetrations or behaviors indicative of potential attacks. This method relies on rules tailored to specific machines and operating systems, generated by experts through interviews with system administrators and security analysts. The effectiveness of this approach hinges on the expertise of those involved in crafting and refining the rules, making it crucial to have skilled individuals setting up these detection mechanisms. [10]

CHAPTER 3

EXISTING SYSTEM

The current landscape of host-based intrusion detection systems (HIDS) for IoT devices encompasses a variety of solutions developed by both commercial vendors and open-source communities. These systems aim to detect and mitigate security threats targeting individual IoT devices by monitoring and analyzing their internal activities. While these existing HIDS solutions offer valuable capabilities for detecting and responding to security threats in IoT environments, they also have their limitations and challenges, as outlined in the previous section. Developing more effective and resilient HIDS solutions requires addressing these shortcomings and leveraging advanced technologies such as machine learning, behavioral analytics, and threat intelligence integration.

3.1 DRAWBACKS OF EXISTING SYSTEM

- *Limited Resource Efficiency:*

Many existing HIDS solutions are resource-intensive, consuming significant memory and processing power. This can be problematic for resource-constrained IoT devices with limited computing capabilities.

-Lack of Scalability:

Traditional HIDS may struggle to scale effectively to accommodate the growing number of IoT devices in large-scale deployments. Managing and monitoring a vast number of devices can become cumbersome and inefficient.

-Signature-Based Detection:

Some HIDS rely heavily on signature-based detection techniques, which are effective against known threats but struggle to detect novel or zero-day attacks. This leaves IoT devices vulnerable to emerging threats that lack predefined signatures.

-High False Positive Rates:

Certain HIDS solutions suffer from high false positive rates, generating numerous alerts for benign activities. This inundates administrators with false alarms, leading to alert fatigue and potentially overlooking genuine security incidents.

-Limited Visibility:

Many existing HIDS provide limited visibility into the internal activities and processes of IoT devices. This lack of granular visibility makes it challenging to accurately detect and respond to sophisticated attacks that exploit internal system vulnerabilities.

-Complex Configuration and Management:

Configuring and managing HIDS can be complex and time-consuming, requiring expertise in cybersecurity and system administration. This complexity may deter organizations with limited resources or expertise from implementing robust security measures.

-Insufficient Forensic Capabilities:

Some HIDS lack comprehensive logging and reporting functionalities, making post-incident analysis and forensic investigations challenging. This hampers the ability to identify the root cause of security breaches and implement effective remediation measures.

-Incompatibility with Legacy Systems:

Integrating HIDS with legacy IoT devices and heterogeneous environments can pose compatibility challenges. Ensuring seamless interoperability across diverse device types and protocols remains a significant obstacle.

CHAPTER 4

PROPOSED SYSTEM

The proposed Host-Based Intrusion Detection System (HIDS) crafted in Python is designed to address the security challenges inherent in IoT environments. It offers a comprehensive solution for real-time threat detection and mitigation, aiming to enhance the security posture of interconnected IoT devices. The proposed HIDS in Python represents a proactive approach to security in IoT environments, providing administrators with the tools and capabilities needed to detect, mitigate, and respond to security threats in real-time, thereby enhancing the overall security posture of interconnected IoT devices.

4.1 FUNCTIONS & KEY COMPONENTS:

- *Lightweight Sensor Agents:*

-Deployed on individual IoT devices, the sensor agents are lightweight and resource-efficient, ensuring minimal impact on device performance.

- Responsible for monitoring system activities, such as file modifications, process executions, and network connections, in real-time.

- *Logging and Reporting:*

- Records detected events and system activities for audit and forensic analysis.

- Generates detailed reports on detected threats, including timestamps, severity levels, and affected devices, facilitating post-incident analysis and remediation.

- *Continuous Monitoring and Analysis:*

- Monitors device activities and network traffic continuously, ensuring timely detection of security threats.

- Analyzes behavior patterns and network communications to identify anomalous activities indicative of potential intrusions or malicious behavior.

- *Prompt Response Mechanisms:*

- Upon detecting a security threat, the system triggers automated responses or mitigation measures to contain and mitigate the impact of the threat.

- Examples include isolating compromised devices from the network, blocking malicious IP addresses, or terminating suspicious processes.

- *Detection Engine:*

-Utilizes a combination of rule-based and anomaly-based techniques to detect potential security threats.

- Rule-based detection: Matches patterns in data against predefined rules to identify known threats, such as malware signatures or suspicious network traffic.

- Anomaly-based detection: Identifies deviations from normal behavior, enabling the detection of novel or zero-day attacks.

- *Alerting System:*

- Notifies administrators or security personnel upon detection of suspicious activities or security breaches.

- Alerts can be delivered through various channels, including email, SMS, or push notifications, ensuring prompt response to detected threats.

- *Real-Time Threat Detection:*
  -The HIDS continuously monitors the activities of IoT devices in real-time, allowing for the immediate detection of suspicious or malicious behavior.

- *Flexible Configuration:*
  - A user-friendly UI provides flexible configuration options, allowing administrators to easily add or remove IP addresses, domains, services, and ports to the blocklist, configure reports, alerts, and other security settings.

- *Blocklist Management:*
  - The system maintains blocklists of known malicious IP addresses, domains, services, and ports, which are stored as JSON files in the config folder. These blocklists are dynamically loaded and updated by the packet-capture.py script, enabling effective blocking of malicious traffic.

- *Alerting and Reporting:*
  - In addition to blocking malicious traffic, the HIDS includes an alerting system to notify administrators of detected threats in real-time. It also provides logging and reporting capabilities for forensic analysis, enabling administrators to investigate security incidents and take appropriate action.

- *Integration with Firewall:*
  - While the system currently utilizes the iptables firewall for packet blocking, it is designed to support integration with other firewall solutions for enhanced security and flexibility.

- *Ongoing Improvement:*

-The development of the HIDS is an ongoing process, with plans to implement signature based detection and entropy-based DDoS detection to further strengthen its capabilities and effectiveness in mitigating security threats.

- *Comprehensive Analysis:*

-Utilizing the Scapy library for network packet analysis, the system performs comprehensive analysis of network traffic to identify potential security threats, including unauthorized access attempts, malware infections, and abnormal network behavior.

CHAPTER 5

SYSTEM ARCHITECTURE

5.1 ARCHITECTURE DIAGRAM

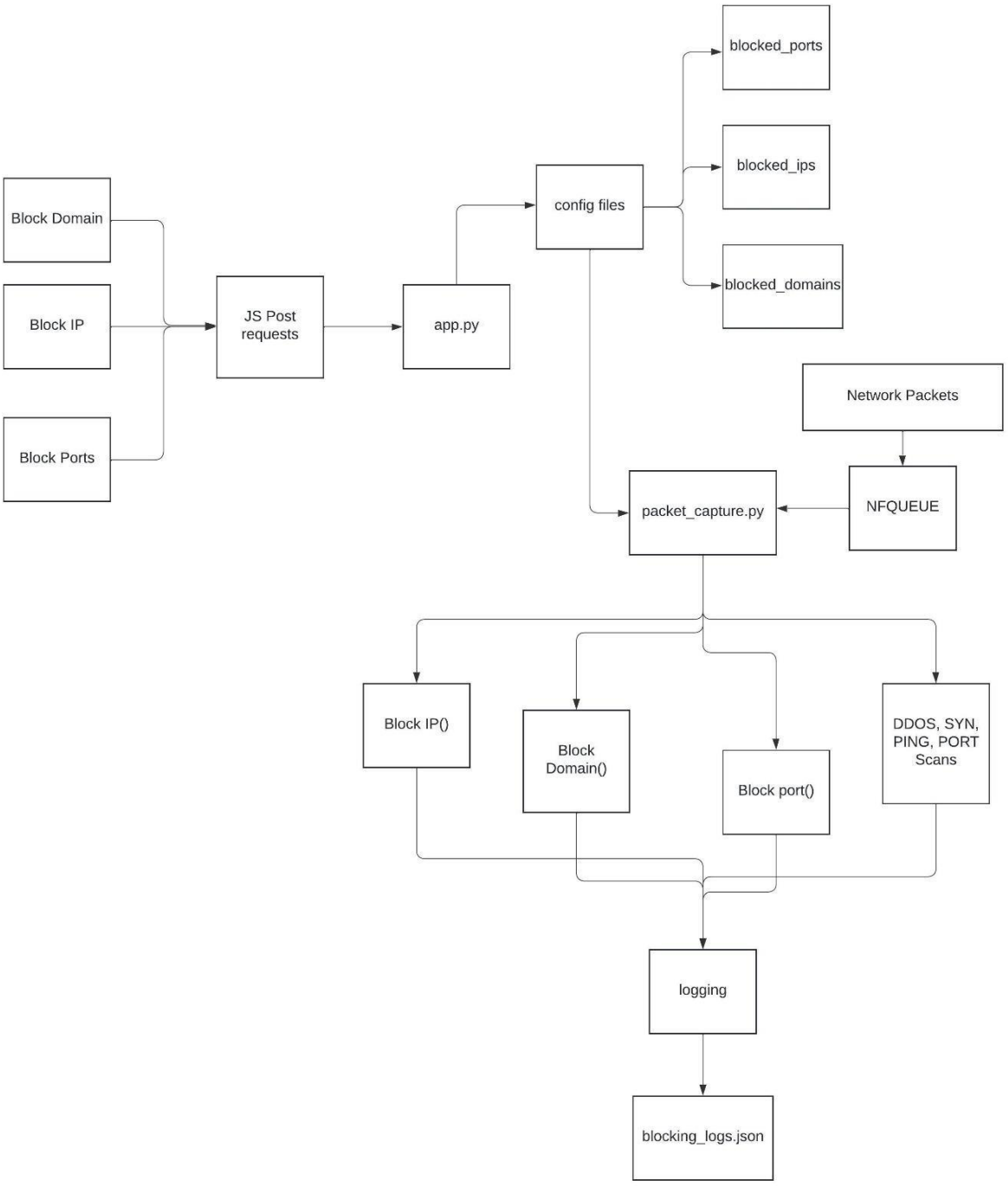


Fig 5.1 Proposed system

Blocked Ports:

Specific ports are designated for certain services (e.g., port 80 for web traffic, port 22 for Secure Shell). Blocking ports restricts access to those services.This can be helpful in preventing unauthorized access to vulnerable services running on the system.

Blocked IPs:

IP addresses are unique identifiers assigned to devices on a network. Blocking specific IPs prevents those devices from communicating with the system.This is commonly used to block known malicious actors or suspicious activity originating from specific IP ranges.

Blocked Domains:

Domain names are human-readable web addresses that translate to numerical IP addresses. Blocking domains prevents users from accessing websites associated with those names.This can be used to block malicious websites, phishing attempts, or inappropriate content.

Functionality Breakdown:

Block Domain/IP/Port: These represent software modules responsible for implementing the blocking logic. They likely interact with firewall rules or access control lists (ACLs) to define blocking criteria. These modules can be configured to block based on various factors like IP address range, specific domain names, or port numbers.

Packet Capture:

This involves capturing data packets traveling across the network. Captured packets can be analyzed for malicious content, suspicious activity patterns, or to identify the source of network attacks.

NFQUEUE:

As mentioned, it's a framework within the Linux kernel.

It allows the "Block Domain/IP/Port" modules to inspect and potentially modify network packets before they reach their destination. This enables more granular control over traffic filtering and potential threat mitigation.

Block IP() / Block Domain() / Block Port() functions:

These represent specific code sections within the blocking modules responsible for enforcing the defined blocking rules.They interact with the system's firewall or ACLs to implement the blocking action.

Logging:

System logs record important events, including information about blocked traffic. This data can be crucial for troubleshooting security incidents, analyzing attack patterns, and maintaining an audit trail.

Blocking logs.json:

This is likely a specific log file storing details about blocked traffic events. The information may include timestamps, blocked IP addresses/domains, and potentially reasons for blocking.

Additional Points:

- Configurability: These systems are often configurable, allowing administrators to define specific blocking criteria based on security policies and network requirements.
- False positives: While effective, blocking mechanisms can occasionally lead to unintended consequences.

For instance, accidentally blocking legitimate traffic can disrupt normal operations.

Maintaining an updated list of trusted sources and carefully defining blocking rules is crucial.

By understanding these components and their functionalities, you gain a deeper understanding of how network traffic filtering systems operate and contribute to overall network security.

Host-Based Intrusion Detection Systems (HIDS) are indispensable components of IoT security, particularly due to the unique challenges posed by the diverse and interconnected nature of IoT devices.

5.1.1. Comprehensive Device Monitoring:

- A HIDS designed for IoT devices provides comprehensive monitoring of device activities, including network traffic, system logs, file integrity, and application behavior. This holistic approach enables the detection of various types of threats, ranging from network-based attacks to malicious software exploitation.

5.1.2. Anomaly Detection:

- One of the primary functions of a HIDS is anomaly detection. By establishing baseline behavior profiles for IoT devices, the HIDS can identify deviations from normal patterns indicative of potential security incidents or intrusions. This proactive approach allows for the early detection and mitigation of emerging threats.

5.1.3. Real-time Threat Response:

- With the ability to detect anomalies in real-time, a HIDS can trigger immediate response actions to mitigate threats. This may include blocking suspicious network connections, isolating compromised devices from the network, or triggering alerts to notify administrators of security incidents.

5.1.4. Customized Security Policies:

- A HIDS tailored for IoT devices allows administrators to define customized security policies based on the specific requirements and characteristics of IoT deployments. This flexibility ensures that security measures are aligned with the unique challenges posed by IoT environments, such as resource-constrained devices and heterogeneous architectures.

5.1.5. Integration with IoT Platforms:

- Integrating the HIDS with IoT platforms and management systems enhances its capabilities by providing contextual information about device behavior and environmental factors. This integration enables more accurate threat detection and allows for automated response actions based on predefined policies.

5.1.6. Scalability and Adaptability:

- A HIDS tailored for IoT devices should be scalable and adaptable to accommodate the dynamic nature of IoT deployments. As the number and diversity of IoT devices grow, the HIDS should be able to scale its monitoring and detection capabilities accordingly, without compromising performance or accuracy.

CHAPTER 6

SYSTEM REQUIREMENTS

6.1 SOFTWARE REQUIREMENTS

i) Front end: HTML, Python

ii) Back end: Python

iii) Operating System:

* Client: Linux

iv) Platform: GitHub

6.2 HARDWARE REQUIREMENTS

System

*Linux/Ubuntu
*RAM: 4 GB
*System- type :64-bit
*Network Connectivity

CHAPTER 7

MODULE DESCRIPTION

7.1 ATTACK DETECTION MODULE

Like ddos, we use a rule based approach here. We can set the threshold for number of requests that can be accepted from IP or to an IP.

7.1.1 Threshold Setting:

- Administrators can configure the system to set thresholds for the number of requests that can be accepted from an IP address or directed towards an IP address within a certain time frame.
- These thresholds are adjustable parameters that can be fine-tuned based on the specific requirements and characteristics of the IoT environment.

7.1.2 Monitoring Incoming Traffic:

- The detection module continuously monitors incoming network traffic, specifically focusing on traffic patterns and request rates from individual IP addresses.
- It tracks the number of requests originating from each IP address and the rate at which these requests are being sent to identify anomalies indicative of potential DDoS attacks.

7.1.3 Detection Logic:

- If the number of requests from a single IP address exceeds the predefined threshold within the specified time window, the system flags it as suspicious activity.
- Similarly, if the number of requests directed towards a specific IP address surpasses the configured threshold, it triggers an alert indicating a potential DDoS attack targeting that IP address.

7.1.4 Mitigation Strategies:

- Upon detecting suspicious activity indicative of a DDoS attack, the system initiates mitigation strategies to mitigate the impact of the attack and maintain service availability.
- Mitigation techniques may include rate limiting, traffic filtering, IP blocking, or redirection of traffic to specialized DDoS protection services.

7.1.5 Real-time Alerting:

- The detection module generates real-time alerts to notify administrators or security personnel about detected DDoS attack attempts.
- Alerts provide details about the source IP addresses, target IP addresses, request rates, and other relevant information to facilitate prompt response and remediation.

7.1.6 Adaptive Thresholding:

- The system incorporates adaptive thresholding mechanisms to dynamically adjust the threshold values based on changing traffic patterns and attack conditions.
- Adaptive thresholding ensures that the detection module can effectively adapt to evolving DDoS attack tactics and maintain optimal performance in mitigating threats.

7.2 ALERT MODULE

When the ddos module detects attack, it forwards it to the alert module and alerts can be seen in the alert tabIn the reports module, we can see past attacks and IPs from the attack received or attack from our own systemIn . The alert module within the Host-Based Intrusion Detection System (HIDS) is a crucial component responsible for notifying administrators or security personnel about potential security threats detected within the IoT environment. The alert module plays a critical role in enhancing the visibility, responsiveness, and effectiveness of the HIDS by ensuring that administrators are promptly notified about security threats, enabling them to take proactive measures to safeguard the IoT environment.

7.3 REPORT MODULE

We can see past attacks and IPs from the attack received or attack frm our system

7.3.1 Historical Incident Logs:
The report module maintains a centralized repository of historical incident logs, capturing details of all security events, attacks, and anomalies detected by the HIDS over time. Each log entry contains relevant information such as timestamps, event types, affected devices, source IP addresses, target ports, and any other pertinent details.

7.3.2 Incident Summaries:
The report module generates summarized reports or summaries of past security incidents, allowing administrators to quickly review and analyze the overall security posture of the IoT environment. These summaries may include metrics such as the total number of attacks, the distribution of attack types, trends over time, and the impact on system performance.

7.3.3 Attack Attribution:

For each security incident logged in the report module, administrators can determine the attribution of attacks, distinguishing between external threats originating from outside the IoT environment and internal threats originating from within the network. This attribution helps in understanding the source and nature of security threats faced by the IoT ecosystem.

7.3.4 IP Reputation Analysis:

The report module facilitates IP reputation analysis by presenting information about IP addresses associated with detected attacks. Administrators can identify repeat offenders, malicious IP addresses, or suspicious network traffic patterns, enabling them to take proactive measures such as blacklisting or blocking malicious IPs.

7.3.5 Forensic Analysis:

In addition to providing insights into past security incidents, the report module supports forensic analysis of security events, allowing administrators to delve deeper into the details of specific incidents. Forensic analysis may involve examining packet captures, analyzing attack payloads, correlating events across multiple devices, and reconstructing attack scenarios for investigation purposes.

7.3.6 Compliance and Audit Trails:

The report module assists in compliance management and audit trail generation by documenting security incidents and responses in a structured format. This documentation helps organizations demonstrate regulatory compliance, track security incidents for audit purposes, and improve incident response procedures over time.

7.4 CONFIGURATION MODULE

we can set which protocols and services (eg ssh, Apache etc) in incoming and which in outgoing traffic. If anything else happens, that will also be reported. In the block module, we can block ip, domain, ports (incoming and outgoing)

7.4.1 Protocol and Service Configuration:

Administrators can specify which network protocols (e.g., TCP, UDP, ICMP) and services (e.g., SSH, HTTP, FTP) are permitted for incoming and outgoing traffic. By defining protocol and service rules, administrators can enforce security policies tailored to the specific requirements of the IoT environment. For example, they may allow incoming SSH connections for remote device management while blocking incoming traffic on non-standard ports.

7.4.2 Custom Rules for Traffic Management:

The configuration module allows administrators to define custom rules for traffic management based on criteria such as source IP addresses, destination IP addresses, port numbers, and

packet attributes. These rules enable administrators to segment network traffic, prioritize critical services, and restrict access to sensitive resources.

7.4.3 Blocking Functionality:

In addition to defining allowed protocols and services, the configuration module includes functionality for blocking specific IP addresses, domains, or port numbers. Administrators can configure blocklists to prevent communication with known malicious entities, suspicious domains, or unauthorized services. Blocked entities are denied access to the IoT environment, mitigating potential security risks and protecting IoT devices from external threats.

7.4.5 Real-Time Configuration Updates:

The configuration module supports real-time updates to protocol and service rules, allowing administrators to adapt quickly to changing security requirements or emerging threats. Administrators can modify rules, add new rules, or disable existing rules on-the-fly without disrupting network operations.

7.5 RULES MODULE

We add new rules and that will be handled by snort. Google snort if u don't know what it is. Basically it is an IDS where we can give rules and it will look for traffics that violate them

7.5.1 Rule Creation:

Administrators can define custom security rules specifying conditions that trigger alerts or actions when detected by Snort. These rules typically consist of a combination of match criteria, such as source IP addresses, destination IP addresses, port numbers, protocol types, and packet content patterns. By crafting rules tailored to the specific security requirements of the IoT environment, administrators can effectively detect and respond to potential security threats.

7.5.2 Rule Syntax and Structure:

Snort rules follow a specific syntax and structure defined by the Snort rule language. Each rule comprises multiple fields, including the action, protocol, source and destination addresses, port numbers, and content matching criteria. Administrators must adhere to the prescribed format and conventions when creating rules to ensure their proper interpretation and execution by the Snort IDS engine.

7.5.3 Rule Management:

The rules module provides functionality for managing and organizing security rules within the HIDS environment. Administrators can create rule sets, categorize rules based on their relevance or severity, and prioritize the enforcement of critical rules over less critical ones. Rule management capabilities enable administrators to maintain an organized and efficient rule base, facilitating effective threat detection and response operations.

7.5.4 Signature-Based Detection:

Snort primarily relies on signature-based detection to identify known threats and attack patterns present in network traffic. Administrators can leverage Snort's extensive rule repository, comprising thousands of pre-defined signatures for common network threats, vulnerabilities, and attack techniques. Additionally, administrators can create custom signatures tailored to the specific characteristics of the IoT environment to enhance detection accuracy and coverage.

7.5.5 Continuous Monitoring and Alerting:

Snort continuously monitors network traffic, analyzing packets against the configured rulesets in real-time. Upon detecting a rule match indicative of a potential security threat, Snort generates alerts, notifying administrators of the suspicious activity. Alerts include pertinent details such as the rule ID, timestamp, source and destination addresses, and a brief description of the detected event, enabling administrators to investigate and respond to security incidents promptly.

7.5.6 Integration with Snort:

The rules module seamlessly integrates with the Snort IDS engine, allowing administrators to deploy and enforce security rules across the IoT network. Snort continuously monitors network traffic, analyzing packets in real-time against the configured rulesets. When a packet matches a rule's criteria, Snort generates an alert or takes predefined actions, such as logging the event, blocking the traffic, or triggering an automated response.

7.5.7 Rule Updates and Maintenance:

The rules module facilitates the timely updating and maintenance of security rules to address emerging threats, vulnerabilities, and attack vectors. Administrators can regularly update Snort's rule sets using the latest threat intelligence feeds, community-contributed rules, and vendor-supplied updates. By staying abreast of evolving security threats and vulnerabilities, administrators can proactively strengthen the security defenses of the IoT environment and minimize the risk of successful attacks.

CHAPTER 8

IMPLEMENTATION

We use scapy, a python library for network packet analysis to monitor all packets going through the network. We have 2 python scripts running, one will Provide a UI for configuration like adding IP, domain,service, ports to blocklist, reports, alerts etc. The second one is solely for packet capture and analysis which utilizes the scapy library. The first one is just a flask app that provides endpoints for UI and other elements as mentioned before. The blocklists are written to files in config folder. The config folder will have Json files having information of blocked ips, ports, services etc.

The second file, which is a python script, named packet-capture.py will read from the config files and stores into variables dynamically. It will look for blocked contents in packets. If blocked contents are found, then it will write it into a log file and then drops the packet. To block packets,currently we use the iptables firewall for easiness. Else Low-level network programming and kernel level development will be required which takes time and lots of study. The remaining 2 topics are signature-based detection and entropy based DDOS detection. These were previously developed but logical errors were there, so we are improving the code.

The architecture of the Host-Based Intrusion Detection System (HIDS) involves two main components: the user interface (UI) for configuration and management, and the packet capture and analysis engine. The HIDS provides a comprehensive solution for detecting and mitigating security threats in IoT environments. By combining packet analysis, rule-based filtering, and advanced detection techniques, the system helps fortify the security posture of IoT devices and networks, thereby minimizing the risk of cyber attacks and ensuring the robustness of IoT ecosystems.

8.1. USER INTERFACE (UI):

- Developed using Flask, a Python web framework, the UI provides a user-friendly interface for configuring and managing the HIDS.
- Users can access various features such as adding IP addresses, domains, services, and ports to the blocklist, generating reports, configuring alerts, and performing other administrative tasks.
- Endpoints exposed by the Flask app enable interaction with the UI, allowing users to make changes to the HIDS configuration and view relevant information.

8.2.PACKET CAPTURE AND ANALYSIS ENGINE:

- Implemented as a separate Python script, named packet-capture.py, this component is responsible for capturing and analyzing network packets in real-time.

- Utilizes the Scapy library for network packet analysis, enabling comprehensive inspection of packet contents, headers, and protocols.
- Reads configuration files stored in the config folder, which contain information about blocked IPs, ports, services, and other relevant settings. These configurations are dynamically loaded into memory for efficient processing.
- Implements logic to examine incoming packets for blocked contents based on the configured rules and criteria. If a packet contains blocked content, it is logged and subsequently dropped to prevent further processing or execution.
- Currently relies on the iptables firewall for packet blocking, providing a straightforward approach to traffic management and enforcement. Alternatively, low-level network programming and kernel-level development could be explored for more advanced packet blocking capabilities.

8.3 SIGNATURE-BASED DETECTION AND ENTROPY-BASED DDOS DETECTION:

- These functionalities are part of the ongoing development efforts to enhance the HIDS capabilities.
- Signature-based detection involves the creation and utilization of signatures or patterns to identify known threats and attack patterns in network traffic. This approach leverages predefined rules or signatures to detect and mitigate common security threats effectively.
- Entropy-based DDoS detection focuses on analyzing the entropy of network traffic to detect distributed denial-of-service (DDoS) attacks. By monitoring changes in traffic patterns and identifying deviations from normal behavior, this approach helps detect and mitigate DDoS attacks in real-time.
- The development team is refining the code for these detection mechanisms to address logical errors and improve overall effectiveness in detecting and mitigating security threats.

8.4 PACKET CAPTURE AND ANALYSIS:

- Scapy, a Python library, is utilized for real-time network packet capture and analysis. It provides comprehensive capabilities for dissecting, forging, and manipulating packets at a low level. - The `packet-capture.py` script is dedicated to packet capture and analysis. It continuously monitors network traffic, inspects packet headers and payloads, and identifies any blocked contents based on predefined rules.

8.5 BLOCKLIST MANAGEMENT:

- Blocklists containing information about blocked IPs, ports, and services are stored as JSON files in the config folder. These files serve as the central repository for managing blocklist configurations. - The `packet-capture.py` script dynamically reads blocklist information from the JSON files and stores it in variables for efficient processing during packet analysis.

8.6 ENHANCEMENTS AND FUTURE DIRECTIONS:

- The system architecture can be further enhanced by incorporating additional features such as signature-based detection, entropy-based DDoS detection, and integration with threat

intelligence feeds. - Detailed diagrams or flowcharts can provide visual representations of the system architecture, illustrating the interactions between different components and data flow throughout the system.

8.7 ALERTING AND LOGGING:

- Detected security threats, such as packets containing blocked contents, are logged to a log file for record-keeping and analysis. - Additionally, the system can trigger alerts to notify administrators of potential security incidents, enabling timely response and mitigation actions.

CHAPTER 9

RESULTS

The system has undergone significant enhancements, enabling it to detect and mitigate various types of attacks, provide alerts, generate reports, and offer granular control over network traffic. The combination of attack detection, alerting, reporting, and granular traffic control functionalities empowers administrators to effectively safeguard their network infrastructure from malicious activities.

9.1. DDOS AND FLOODING ATTACK DETECTION:

- The system is now capable of detecting DDoS and flooding attacks in real-time. It monitors incoming traffic patterns and triggers alerts when it detects an unusually high volume of requests or packets, indicative of a potential DDoS or flooding attack.

9.2. ALERTING SYSTEM:

- Upon detecting a DDoS or flooding attack, the system promptly alerts administrators, providing them with real-time notifications to take immediate action and mitigate the impact of the attack.

9.3. REPORTING FUNCTIONALITY:

- The system generates detailed reports whenever a DDoS or flooding attack occurs. These reports include information about the attack, such as the time of occurrence, the source of the attack, and the affected devices or services.

9.4. BLOCKING DOMAINS, IPS, AND PORTS:

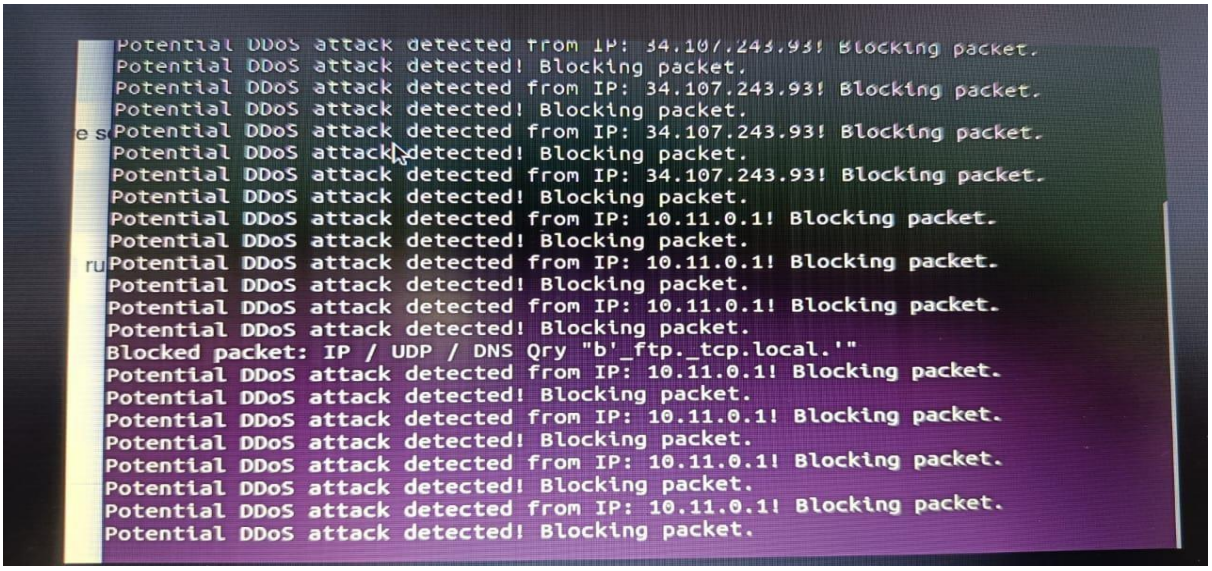
- Administrators can now configure the system to block specific domains, IP addresses, and ports, both incoming and outgoing. This capability allows for fine-grained control over network traffic, enabling the prevention of malicious connections and unauthorized access attempts.

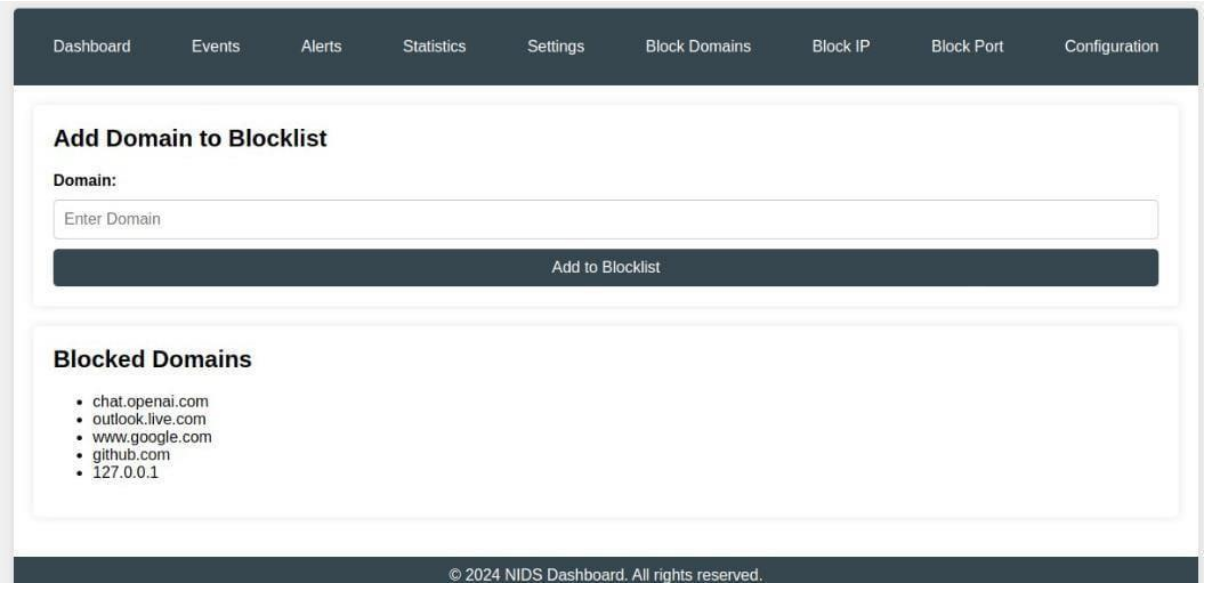
9.5. CONFIGURABLE NIDS RULES:

- The Network Intrusion Detection System (NIDS) component of the system can be configured to block specific services, such as SSH or Apache. Administrators can customize the NIDS rules via the UI, tailoring them to the specific security requirements of their environment.

9.6 CUSTOM RULES MANAGEMENT:

- Administrators have the flexibility to define custom rules via the UI, which are then integrated into the system's intrusion detection capabilities. These rules can be added to Snort, a popular open-source NIDS, allowing for the monitoring and blocking of traffic that violates the defined rules.





CHAPTER 10

CONCLUSION

Host-Based Intrusion Detection Systems (HIDS) are indispensable components of IoT security, particularly due to the unique challenges posed by the diverse and interconnected nature of IoT devices. A Host-Based Intrusion Detection System (HIDS) tailored for IoT devices plays a crucial role in maintaining the security and integrity of IoT ecosystems. By continuously monitoring device activities, detecting anomalies, and responding to threats in real-time, HIDS helps mitigate the evolving cybersecurity risks associated with the proliferation of IoT devices. One of the primary functions of a HIDS tailored for IoT devices is continuous monitoring of device activities. IoT devices, ranging from smart home appliances to industrial sensors, generate a vast amount of data and communicate with other devices and networks. By monitoring the behavior and communication patterns of these devices in real-time, a HIDS can detect any anomalies or suspicious activities that may indicate a security breach.

Furthermore, HIDS employ sophisticated detection mechanisms to identify potential threats. These mechanisms may include signature-based detection, anomaly detection, behavior analysis, and machine learning algorithms. Signature-based detection involves comparing network traffic and device activities against known patterns of malicious behavior, while anomaly detection identifies deviations from normal behavior that may indicate an intrusion.

In addition to detection, HIDS also play a crucial role in responding to security incidents in real-time. Upon detecting a potential threat or intrusion, the HIDS can trigger automated responses such as blocking suspicious network traffic, isolating compromised devices, or alerting system administrators. By taking proactive measures to mitigate threats as soon as they are detected, HIDS help prevent further damage to IoT devices and networks.

Moreover, HIDS provide valuable insights into the security posture of IoT ecosystems through comprehensive logging and reporting capabilities. System administrators can analyze security incidents, identify attack vectors, and implement remediation measures to strengthen the overall security posture of IoT deployments.

Overall, a well-designed and effectively deployed HIDS tailored for IoT devices is indispensable for safeguarding IoT ecosystems against cyber threats. By continuously monitoring device activities, detecting anomalies, and responding to security incidents in real-time, HIDS play a vital role in maintaining the security and integrity of IoT deployments in today's interconnected world.

REFERENCE

[1] Sekar, R., Guang, Y., Verma, S., & Shanbhag, T. (1999, November). A high-performance network intrusion detection system. In *Proceedings of the 6th ACM Conference on Computer and Communications Security* (pp. 8-17).

[2] Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, *36*(1), 16-24.

[3] Shaikh, A. A., Qi, H., Jiang, W., & Tahir, M. (2017, December). A novel HIDS and log collection based system for digital forensics in cloud environment. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)* (pp. 1434-1438). IEEE.

[4] Hong, S. P., Lim, C. H., & Lee, H. J. (2021, February). APT attack response system through AM-HIDS. In *2021 23rd International Conference on Advanced Communication Technology (ICACT)* (pp. 271-274). IEEE.

[5] Wang, Z., & Zhu, Y. (2017, June). A centralized HIDS framework for private cloud. In *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (pp. 115-120). IEEE.

[6] Aydın, M. A., Zaim, A. H., & Ceylan, K. G. (2009). A hybrid intrusion detection system design for computer network security. *Computers & Electrical Engineering*, *35*(3), 517-526.

[7] Elrawy, M. F., Awad, A. I., & Hamed, H. F. (2018). Intrusion detection systems for IoT-based smart environments: a survey. *Journal of Cloud Computing*, *7*(1), 1-20.

[8] Gyamfi, E., & Jurcut, A. (2022). Intrusion detection in internet of things systems: a review on design approaches leveraging multi-access edge computing, machine learning, and datasets. *Sensors*, *22*(10), 3744.

[9] Chawla, S., & Thamilarasu, G. (2018, April). Security as a service: real-time intrusion detection in internet of things. In *Proceedings of the Fifth Cybersecurity Symposium* (pp. 1-4).

[10] Ilgun, K., Kemmerer, R. A., & Porras, P. A. (1995). State transition analysis: A rule-based intrusion detection approach. *IEEE transactions on software engineering*, *21*(3), 181-199.

APPENDICES

SOURCE CODE

```
from flask import Flask, render_template, jsonify, request, redirect, url_for
import socket
import  json
import time
import os
from scapy.all import ARP, Ether, srp
from typing import List
import socket
import ipaddress
from collections import Counter
import netifaces
start_time = time.time()
BLOCKED_DOMAINS_FILE = 'config/blocked_domains.json'
BLOCKED_PORTS_FILE = 'config/blocked_ports.json'
app = Flask(_name_)
@app.route('/alerts')
def alerts():
    with open('config/blocking_logs.json', 'r') as f:
        logs = [json.loads(line) for line in f]
      ddos_alerts = [log for log in logs if 'DDoS' in log['message']]
      return render_template('alerts.html', ddos_alerts=ddos_alerts)
@app.route('/statistics')
def statistics():
    with open('config/blocking_logs.json', 'r') as f:
```

```
        logs = [json.loads(line) for line in f]
    ips_blocked = len(set(log['src'] for log in logs if 'Blocked IP' in log['message']))
    ports_blocked = len(set(log['dst'] for log in logs if 'Blocked Port' in log['message']))
    domains_blocked = len(set(log['dst'] for log in logs if 'Blocked Domain' in log['message']))
    ddos_detected = sum(1 for log in logs if 'DDoS' in log['message'])
    ddos_by_src = Counter(log['src'] for log in logs if 'DDoS' in log['message'])
    most_blocked_ips = Counter(log['src'] for log in logs if 'Blocked IP' in log['message']).most_common(10)
    most_blocked_ports = Counter(log['dst'] for log in logs if 'Blocked Port' in log['message']).most_common(10)
    most_blocked_domains = Counter(log['dst'] for log in logs if 'Blocked Domain' in log['message']).most_common(10)
     return  render_template(
        'statistics.html',
        ips_blocked=ips_blocked,
        ports_blocked=ports_blocked,
        domains_blocked=domains_blocked,
        ddos_detected=ddos_detected,
        ddos_by_src=ddos_by_src.items(),
        most_blocked_ips=most_blocked_ips,
        most_blocked_ports=most_blocked_ports,
        most_blocked_domains=most_blocked_domains
    )
def load_blocked_ports():
    try:
        with open(BLOCKED_PORTS_FILE, 'r') as f:
            return json.load(f)
    except json.JSONDecodeError:
```

```
        return []
    def save_blocked_ports(blocked_ports):
    with open(BLOCKED_PORTS_FILE, 'w') as f:
        json.dump(blocked_ports, f)
  @app.route('/block-port', methods=['POST'])
def block_port():
    blocked_ports = load_blocked_ports()
    port = request.form.get('port')
      if port not in blocked_ports:
        blocked_ports.append(port)
        save_blocked_ports(blocked_ports)
        return jsonify({'success': True})
      return jsonify({'success': False, 'error': 'Port is already blocked.'})
  @app.route('/unblock-port', methods=['POST'])
def unblock_port():
    blocked_ports = load_blocked_ports()
    port = request.form.get('port')
    if port in blocked_ports:
        blocked_ports.remove(port)
        save_blocked_ports(blocked_ports)
        return jsonify({'success': True})
      return jsonify({'success': False, 'error': 'Port is not blocked.'})
  @app.route('/block-ports')
def block_ports():
    blocked_ports = load_blocked_ports()
    return render_template('block-ports.html', blocked_ports=blocked_ports)
BLOCKED_SERVICES_FILE = 'config/blocked_services.json'
```

```
BLOCKED_IPS_FILE = 'config/blocked_ips.json'
# Function to read the list of blocked IP addresses from blocked_ips.json
def read_blocked_ips():
    try:
        with open(BLOCKED_IPS_FILE, 'r') as f:
            return json.load(f).get('blocked_ips', [])
    except FileNotFoundError:
        return []
# Function to write the list of blocked IP addresses to blocked_ips.json
def write_blocked_ips(blocked_ips):
    with open(BLOCKED_IPS_FILE, 'w') as f:
        json.dump({'blocked_ips': blocked_ips}, f, indent=4)
@app.route('/block-ip', methods=['POST'])
def block_ip():
    try:
        ip_address = request.form['ip']
        blocked_ips = read_blocked_ips()
        if ip_address not in blocked_ips:
            blocked_ips.append(ip_address)
            write_blocked_ips(blocked_ips)
            return jsonify({'success': True}), 200
        else:
            return jsonify({'success': False, 'error': 'IP address already blocked.'}), 400
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)}), 500
@app.route('/unblock-ip', methods=['POST'])
def unblock_ip():
```

```
    try:
        ip_address = request.form['ip']
        blocked_ips = read_blocked_ips()
        if ip_address in blocked_ips:
            blocked_ips.remove(ip_address)
            write_blocked_ips(blocked_ips)
            return jsonify({'success': True}), 200
        else:
            return jsonify({'success': False, 'error': 'IP address not found in the blocked list.'}), 404
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)}), 500
@app.route('/blocked-ips')
def get_blocked_ips():
    try:
        blocked_ips = read_blocked_ips()
        return jsonify({'success': True, 'blocked_ips': blocked_ips}), 200
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)}), 500
    @app.route('/block-ip-page')
def block_ip_page():
    blocked_ips = read_blocked_ips()
    return render_template('block-ip.html', blocked_ips=blocked_ips)
def load_blocked_domains():
    try:
        with open(BLOCKED_DOMAINS_FILE, 'r') as f:
            data = json.load(f)
            blocked_domains = data.get('blocked_domains', [])
```

```
            return set(blocked_domains)
    except FileNotFoundError:
        return set()


def save_blocked_domains(blocked_domains):
    with open(BLOCKED_DOMAINS_FILE, 'w') as f:
        json.dump({'blocked_domains': list(blocked_domains)}, f, indent=4)


@app.route('/delete-domain', methods=['POST'])
def delete_domain():
    try:
        domain_to_delete = request.form['domain']
        blocked_domains = load_blocked_domains()


        if domain_to_delete in blocked_domains:
            blocked_domains.remove(domain_to_delete)
            save_blocked_domains(blocked_domains)
            return jsonify({'success': True, 'message': f'Successfully deleted domain: {domain_to_delete}'}), 200
        else:
            return jsonify({'success': False, 'error': f'Domain "{domain_to_delete}" not found in blocked domains'}), 404
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)}), 500
@app.route('/configuration')
def configuration():
    try:
        with open('config/blocked_services.json', 'r') as f:
```

```
            data = json.load(f)
            blocked_services = data.get('blocked_services', [])
    except FileNotFoundError:
        blocked_services = []
return render_template('configuration.html', blocked_services=blocked_services)
@app.route('/block-domain')
def block_domain():
    domains = get_domains()
    return render_template('block-domain.html', domains=domains)
def read_blocked_domains():
    try:
        with open(BLOCKED_DOMAINS_FILE, 'r') as f:
            return set(json.load(f).get('blocked_domains', []))
    except FileNotFoundError:
        return set()
def write_blocked_domains(blocked_domains):
    with open(BLOCKED_DOMAINS_FILE, 'w') as f:
        json.dump({'blocked_domains': list(blocked_domains)}, f, indent=4)
@app.route('/add-domain', methods=['POST'])
def add_domain():
    try:
        domain = request.form['domain']
        blocked_domains = read_blocked_domains()
        blocked_domains.add(domain)
        write_blocked_domains(blocked_domains)
        return jsonify({'success': True}), 200
    except Exception as e:
```

```
        return jsonify({'success': False, 'error': str(e)}), 500
@app.route('/domains')
def get_domains():
    try:
        blocked_domains = read_blocked_domains()
        return blocked_domains
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)}), 500
@app.route('/uptime')
def get_uptime():
    uptime_seconds = int(time.time() - start_time)
    uptime = format_time(uptime_seconds)
    return jsonify(uptime=uptime)
def format_time(seconds):
    # Convert seconds to days, hours, minutes, and seconds
    minutes, seconds = divmod(seconds, 60)
    hours, minutes = divmod(minutes, 60)
    days, hours = divmod(hours, 24)
    return f"{int(days)} days, {int(hours)} hours, {int(minutes)} minutes, {int(seconds)}
seconds"
def get_local_ip():
    try:
        # Create a socket to get local IP address
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(("8.8.8.8", 80)) # Google's public DNS server
        local_ip = s.getsockname()[0]
      ]  s.close()
```

```
        return local_ip
    except Exception as e:
        print("Error fetching local IP:", e)
        return None
def get_internet_connected_interface():
    gateways = netifaces.gateways()
    default_gateway = gateways['default'][netifaces.AF_INET][1]
    return default_gateway
  def get_local_ip():
    connected_interface = get_internet_connected_interface()
    local_ip = netifaces.ifaddresses(connected_interface)[netifaces.AF_INET][0]['addr']
    return local_ip
  def get_connected_devices():
    all_connected_devices = []
  # Get the local IP address
  local_ip = get_local_ip()
   # Calculate the /24 subnet that the local IP is a part of
    ip_interface = ipaddress.ip_interface(f"{local_ip}/24")
    ip_network = ip_interface.network
    ip_range = str(ip_network)
   try:
        arp = ARP(pdst=ip_range)
        ether = Ether(dst="ff:ff:ff:ff:ff:ff")
        packet = ether/arp
   result = srp(packet, timeout=3, verbose=False)[0]
  connected_devices = [{'ip': received.psrc, 'mac': received.hwsrc} for sent, received in result]
all_connected_devices.extend(connected_devices)
```

```
    except Exception as e:
        print(f"An error occurred while scanning the IP range {ip_range}: {e}")
  return all_connected_devices
  @app.route('/')
def dashboard():
    local_ip = get_local_ip()
    connected_devices = get_connected_devices()
    return render_template('home.html', local_ip=local_ip, connected_devices=connected_devices)
if__name__== '_main_':
    app.run(debug=True)
```

PACKET CAPTURE

```
import json
import netfilterqueue
from datetime import datetime
from scapy.all import *
from scapy.layers.inet import IP, TCP, UDP, ICMP
from scapy.layers.dns import DNS, DNSQR, DNSRR
from collections import Counter
from math import log2
import socket
  # Get the IP address of your machine
def get_my_ip():
    return [(s.connect(('8.8.8.8', 53)), s.getsockname()[0], s.close()) for s in [socket.socket(socket.AF_INET, socket.SOCK_DGRAM)]][0][1]
  MY_IP = get_my_ip()
recent_src_ips = []
# Define maximum size for the queue
```

```
MAX_QUEUE_SIZE = 5000 # Adjust as needed
  # Global variables
src_ips_queue = deque(maxlen=1000) # Adjust queue size as needed
icmp_counter = 0
syn_counter = 0
ICMP_THRESHOLD = 1000 # Adjust as needed
SYN_THRESHOLD = 1000 # Adjust as needed
ENTROPY_THRESHOLD = 100.0 # Adjust as needed
src_ips_ports = defaultdict(set)
PORT_SCAN_THRESHOLD = 100  # Adjust as needed
def calculate_entropy(scapy_packet):
    global src_ips_queue
    global  icmp_counter
    global syn_counter
   # Check if the packet is incoming
    if scapy_packet.haslayer('IP') and scapy_packet['IP'].dst == MY_IP:
  # Add source IP to queue
        src_ips_queue.append(scapy_packet['IP'].src)
  # Check if the packet is ICMP
        if scapy_packet.haslayer(ICMP):
            icmp_counter += 1
            if icmp_counter > ICMP_THRESHOLD:
                print("Potential Ping Flood attack detected! Blocking packet.")
                icmp_counter = 0  # Reset counter
                return True


        # Check if the packet is TCP and has the SYN flag set
```

```
    if scapy_packet.haslayer(TCP) and 'S' in str(scapy_packet.sprintf('%TCP.flags%')):
        syn_counter += 1
        if syn_counter > SYN_THRESHOLD:
            print("Potential SYN Flood attack detected! Blocking packet.")
            syn_counter = 0  # Reset counter
            return True
     # Check if the packet is TCP
    if scapy_packet.haslayer(TCP):
        src_ip = scapy_packet['IP'].src
        dst_port = scapy_packet[TCP].dport
# Add the destination port to the set of ports accessed by the source IP
        src_ips_ports[src_ip].add(dst_port)
  # Check if the number of ports accessed by the source IP exceeds the threshold
        if len(src_ips_ports[src_ip]) > PORT_SCAN_THRESHOLD:
            print("Potential Port Scan detected! Blocking packet.")
            src_ips_ports[src_ip].clear() # Reset ports for the IP
            return True
  # Calculate frequency of each IP
    ip_freq = Counter(src_ips_queue)
  # Calculate probabilities
    probs = [f / len(src_ips_queue) for f in ip_freq.values()]
  # Calculate and return entropy
    entropy = -sum(p * log2(p) for p in probs)
    if entropy < ENTROPY_THRESHOLD:
        print(f"Potential DDoS attack detected from IP: {scapy_packet['IP'].src}! Blocking
packet.")
        return True
```

```
  return False
# Function to load blocked IPs from blocked_ips.json
def load_blocked_ips():
    try:
        with open('config/blocked_ips.json', 'r') as f:
            data = json.load(f)
            return set(data.get('blocked_ips', []))
    except FileNotFoundError:
        return set()
  # Function to load blocked URLs from blocked_domains.json
def load_blocked_urls():
    try:
        with open('config/blocked_domains.json', 'r') as f:
            data = json.load(f)
            return set(data.get('blocked_domains', []))
    except FileNotFoundError:
        return set()
  # Function to load blocked ports from blocked_ports.json
def load_blocked_ports():
    try:
        with open('config/blocked_ports.json', 'r') as f:
            data = json.load(f)
            blocked_ports = set()
            for entry in data:
                port_number = int(entry) # Convert port_number to integer directly
                blocked_ports.add(port_number)
            return blocked_ports
```

```
    except FileNotFoundError:
        return set()
# Function to load blocked services from blocked_services.json
def load_blocked_services():
    try:
        with open('config/blocked_services.json', 'r') as f:
            data = json.load(f)
            return set(data.get('blocked_services', []))
    except FileNotFoundError:
        return set()
  # Define blocked IPs, URLs, ports, and services
blocked_ips = load_blocked_ips()
blocked_urls = load_blocked_urls()
blocked_ports = load_blocked_ports()
blocked_services = load_blocked_services()
  # Function to check if a packet matches any blocked criteria
def is_blocked(scapy_packet):
    if IP in scapy_packet:
        src_ip = scapy_packet[IP].src
        dst_ip = scapy_packet[IP].dst
        if src_ip in blocked_ips or dst_ip in blocked_ips:
            return True, f"Blocked IP: {src_ip if src_ip in blocked_ips else dst_ip}", "IP"
    if DNS in scapy_packet and scapy_packet[DNS].qr == 0:  # Only check queries, not responses
        domain = scapy_packet[DNSQR].qname.decode("utf-8")
        if any(blocked_domain in domain for blocked_domain in blocked_urls):
            return True, f"Blocked URL: {domain}", "DNS"
```

```
    if TCP in scapy_packet:
        src_port = scapy_packet[TCP].sport
        dst_port = scapy_packet[TCP].dport
        if src_port in blocked_ports or dst_port in blocked_ports:
            return True, f"Blocked Port: {src_port if src_port in blocked_ports else dst_port}", "TCP"
    if UDP in scapy_packet:
        # Handle UDP payloads as binary data
        payload = bytes(scapy_packet[UDP].payload)
        # Convert blocked services to bytes-like objects
        blocked_services_bytes = [service.encode() for service in blocked_services]
        if any(service_bytes.lower() in payload.lower() for service_bytes in blocked_services_bytes):
            return True, f"Blocked Service in Payload", "UDP"
    return False, "", ""
# IP address of your "blocked" page
BLOCKED_PAGE_IP = "20.197.9.50" # Replace with the actual IP address
# Modified function to handle packets
def handle_packet(packet):
    global icmp_counter
    global syn_counter
 scapy_packet = IP(packet.get_payload())
   # Add source IP to recent IPs list
    if IP in scapy_packet:
        src_ip = scapy_packet[IP].src
        recent_src_ips.append(src_ip)
   # If the list gets too large, remove the oldest IP
        if len(recent_src_ips) > 10000:  # Adjust this number as needed
```

```
            recent_src_ips.pop(0)
    # Check for ICMP or SYN flood
     if ICMP in scapy_packet:
        icmp_counter += 1
        if icmp_counter > ICMP_THRESHOLD:
            print("Potential Ping Flood attack detected! Blocking packet.")
            packet.drop()
            return
    if TCP in scapy_packet and 'S' in str(scapy_packet.sprintf('%TCP.flags%')):
        syn_counter += 1
        if syn_counter > SYN_THRESHOLD:
            print("Potential SYN Flood attack detected! Blocking packet.")
            packet.drop()
            return
    # Check entropy
     entropy = calculate_entropy(scapy_packet)
     if entropy:
        print("Potential DDoS attack detected! Blocking packet.")
        packet.drop()
        return
   # Check if this is a DNS query
     if DNS in scapy_packet and scapy_packet[DNS].qr == 0:  # Only check queries, not
responses
        domain = scapy_packet[DNSQR].qname.decode("utf-8")
     # Check if the domain is blocked
        if any(blocked_domain in domain for blocked_domain in blocked_urls):
            # print(f"Blocked URL: {domain}")
```

```
  # Create a DNS answer packet
        dns_answer = DNSRR(rrname=domain, rdata=BLOCKED_PAGE_IP)
  # Modify the original packet to be our answer packet
        scapy_packet[DNS].an = dns_answer
        scapy_packet[DNS].ancount = 1
   # Delete checksums and lengths so they get recalculated
        del scapy_packet[IP].len
        del scapy_packet[IP].chksum
        del scapy_packet[UDP].len
        del scapy_packet[UDP].chksum
# Set the packet payload to our modified packet
        packet.set_payload(bytes(scapy_packet))
        packet.accept()
        return
  # Check other blocking criteria
    blocked, log_message, protocol = is_blocked(scapy_packet)
    if blocked:
        # Write log to file
        with open('config/blocking_logs.json', 'a') as log_file:
            log_entry = {
                "timestamp": str(datetime.now()),
                "message": log_message,
                "protocol": protocol,
                "src": scapy_packet[IP].src if IP in scapy_packet else None,
                "dst": scapy_packet[IP].dst if IP in scapy_packet else None
            }
            json.dump(log_entry, log_file)
```

```
            print("Blocked packet:", scapy_packet.summary())
            log_file.write('\n')
            # Drop the packet
            packet.drop()
    else:
        # Allow the packet to pass through
        packet.accept()
    queue = netfilterqueue.NetfilterQueue()
queue.bind(0, handle_packet) # Bind to queue number 0
try:
    print("[*] Waiting for packets...")
    queue.run()
except KeyboardInterrupt:
    print("[*] Stopping...")
```