NIDS for Real-Time Protocol Analysis and Deep Content Inspection – MITM

A PROJECT REPORT

Submitted by,

KRISHNA GEETHA P -20211CCS0049

CHANDAN KUMAR M S -20211CCS0102

MANOJ -20211CCS0112

CHARAN R -20211CCS0129

Under the guidance of,

Mr. TANVEER AHMED

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

At

PRESIDENCY UNIVERSITY

PRESIDENCY UNIVERSITY

BENGALURU

DECEMBER 2024

SCHOOL OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

This is to certify that the Project report "NIDS for Real-Time Protocol Analysis and Deep Content Inspection – MITM" being submitted by "Krishna Geetha P-20211CCS0049, Chandan Kumar-20211CCS0102, Manoj K-20211CCS0112, Charan R-20211CCS0129" in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a bonified work carried out under my supervision.

Mr TANVEER AHMED
Assistant Professor
School of CSE
Presidency University

Dr. ANANDRAJ S P
Head of Department
School of CSE
Presidency University

Dr. L. SHAKKEERA
Associate Dean
School of CSE
Presidency University

Dr. MYDHILI NAIR
Associate Dean
School of CSE
Presidency University

Dr. SAMEERUDDIN KHAN
Pro-VC School of Engineering
Dean -School of CSE&IS
Presidency University

PRESIDENCY UNIVERSITY

SCHOOL OF COMPUTER SCIENCE ENGINEERING

DECLARATION

We hereby declare that the work, which is being presented in the project report entitled "NIDS for Real-Time Protocol Analysis and Deep Content Inspection – MITM" in partial fulfillment for the award of Degree of Bachelor of Technology in Computer Science and Engineering, is a record of our own investigations carried under the guidance of Mr. Tanveer Ahmed, Assistant Professor School of Computer Science Engineering , Presidency University, Bengaluru.

We have not submitted the matter presented in this report anywhere for the award of any other degree.

Names	Roll No	Signatures
KRISHNA GEETHA P	20211CCS0049	
CHANDAN KUMAR MS	20211CCS0102	
MANOJ	20211CCS0112	
CHARAN R	20211CCS0129	

ABSTRACT

This project focuses on developing an advanced Network Intrusion Detection System (NIDS) to detect Man-In-The-Middle (MITM) attacks and other network threats using machine learning and signature-based detection techniques. The system captures network packets in real-time and processes them using Scapy for packet analysis. Anomaly detection is implemented using machine learning algorithms like Isolation Forest to identify deviations from normal network traffic patterns. Additionally, Snort is used for signature-based detection, enabling the identification of known attack signatures. The NIDS is designed to provide real-time alerts and notifications, which are displayed on a web-based dashboard built with Flask. The dashboard allows network administrators to monitor network activity, view alerts, and manage security incidents efficiently. The system also integrates with external security tools to enhance its detection capabilities. The technology stack includes Python, Scapy, Snort, TensorFlow, PostgreSQL, and Flask, providing a scalable and robust solution. The project follows a structured development process, with phases dedicated to configuration, module development, database setup, and testing. By combining machine learning and signature-based detection, this NIDS offers a comprehensive approach to network security, enabling organizations to proactively detect and mitigate network attacks, ensuring the integrity and confidentiality of their systems and data.

ACKNOWLEDGEMENT

First of all, we indebted to the GOD ALMIGHTY for giving me an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean Dr. Md. Sameeruddhin Khan, Pro-VC, School of Engineering and Dean, School of Computer Science Engineering & Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Deans Dr. Shakeera L and Dr. Mydhili Nair, School of Computer Science Engineering , Presidency University, and Dr. “ANADRAJ S P”, Head of the Department, School of Computer Science Engineering , Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide Mr. Tanveer Ahmed, Assistant Professor and Reviewer Ms. Sudha Y Assistant Professor , School of Computer Science Engineering, Presidency University for his inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the project work.

We would like to convey our gratitude and heartfelt thanks to the PIP2001 Capstone Project Coordinators Dr. Sampath A K, Dr. Abdul Khadar A and Mr. Md Zia Ur Rahman, department Project Coordinators “Dr. Sharmasth Vali Y” and Git hub coordinator Mr. Muthuraj.

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

KRISHNA GEETHA P

CHANDAN KUMAR MS

MANOJ

CHARAN

LIST OF FIGURES

Sl. No.	Figure Name	Caption	Page No.
1	Figure 6.1	System Implementation Diagram	49
2	Figure 13.1	Admin-Viewer	96
3	Figure 13.2	Admin-Packet Capturing	96
4	Figure 13.3	Admin Tools	97
5	Figure 13.4	Viewer-Packet Capturing	97
6	Figure 13.5	Viewer-Graph	98

TABLE OF CONTENTS

CHAPTER_NO	TITLE	PAGE_NO
01	INTRODUCTION	01
02	LITERATURE SURVEY	11
03	RESEARCH GAPS OF EXISTING METHODS	17
04	PROPOSED METHOLOGY	26
05	OBJECTIVES	35
06	SYSTEM DESIGN & IMPLEMENTATION	43
07	TIMELINE FOR EXECUTION OF PROJECT (GANTT CHART)	52
08	OUTCOMES	53
09	RESULTS AND DISCUSSIONS	62
10	CONCLUSION	66
11	REFERENCES	68
12	PSUEDOCODE	70
13	SCREENSHOTS	83

CHAPTER-1

INTRODUCTION

In today's interconnected digital world, network security has become one of the foremost concerns for organizations. With the increasing reliance on online platforms for communication, business operations, and data storage, the vulnerability to cyber-attacks has grown significantly. Network security encompasses a wide range of measures designed to protect the confidentiality, integrity, and availability of data and services on a network. The objective of network security is to protect against unauthorized access, data theft, system damage, and disruptions in network services. To safeguard networks, several security mechanisms, including firewalls, encryption, and intrusion detection systems (IDS), are implemented. While traditional methods have their merits, they often fall short when confronted with sophisticated and evolving cyber threats. This is where Network Intrusion Detection Systems (NIDS) become crucial.

An IDS serves as a proactive security tool that monitors network traffic for signs of malicious activity, ensuring that potential threats are identified in real-time. Unlike traditional security mechanisms that focus on blocking attacks, an IDS's primary goal is to detect and alert administrators about ongoing or impending security incidents. The NIDS, in particular, is designed to monitor network traffic specifically, identifying anomalies, potential vulnerabilities, and malicious activities such as hacking attempts, Denial of Service (DoS), and Man-in-the-Middle (MITM) attacks. One of the primary challenges faced by NIDS is the detection of new, previously unknown attack patterns, which often evade traditional detection methods based on predefined signatures. To address these challenges, modern NIDS implementations often combine signature-based detection with more advanced techniques like machine learning and anomaly detection.

Man-in-the-Middle (MITM) attacks are one of the most concerning threats in network security. In a MITM attack, an attacker secretly intercepts and possibly alters the communication between two parties, all without their knowledge. This type of attack can be devastating as it may involve sensitive data such as passwords, credit card numbers, or other personal information. MITM attacks are particularly dangerous because they are difficult to detect and can be executed in various forms, such as through DNS spoofing, ARP poisoning, and SSL stripping. As the sophistication of these attacks increases, the need for advanced detection systems capable of identifying MITM attacks in real-time has become critical for organizations. By implementing effective network intrusion detection strategies,

organizations can significantly reduce the risks posed by such attacks.

Alongside MITM attacks, other network threats, such as DoS attacks, phishing attempts, and malware, are also prevalent. DoS attacks aim to disrupt the services of a network by overwhelming it with excessive traffic, while malware can damage or steal data from infected systems. Phishing, on the other hand, deceives users into revealing sensitive information by impersonating trustworthy entities. These diverse types of threats require an advanced IDS system that not only detects well-known attack patterns but also identifies emerging threats that do not fit conventional attack signatures. For this reason, many modern NIDS solutions incorporate both signature-based detection (for known attacks) and anomaly-based detection (to identify novel attack patterns).

To address the growing challenges of detecting MITM and other network attacks, this project proposes the development of a sophisticated NIDS that leverages both machine learning (ML) techniques and traditional signature-based detection. Machine learning has proven to be a valuable tool in various cybersecurity applications due to its ability to identify subtle patterns in large datasets, including network traffic, that might otherwise go unnoticed. By applying machine learning models, such as Isolation Forest and Support Vector Machines (SVM), to network traffic data, the system can automatically adapt to new and evolving attack strategies. The anomaly detection capabilities of machine learning allow the NIDS to recognize unusual behaviour that could indicate an attack, even if the attack's signature is not yet included in the system's database.

Signature-based detection, on the other hand, is highly effective for identifying known attack patterns. Signature-based systems work by comparing incoming network traffic against a set of predefined patterns or "signatures" of known attacks. Tools like Snort, a widely used open-source intrusion detection system, use this method to identify malicious traffic based on signatures. While signature-based systems are effective at detecting well-established attacks, they are less useful when dealing with new or unknown threats. This is where the integration of machine learning into the NIDS becomes vital, as it enables the system to detect anomalous network behaviour that might indicate a new attack that hasn't been seen before.

The proposed NIDS aims to address both the known and unknown aspects of network security by combining these two detection methods. First, it will use packet capture techniques to collect network traffic data for analysis. Tools like Scapy will be employed to capture network packets, which will then be pre-processed and fed into both the machine learning models and signature-based detection systems. The NIDS will analyse the captured traffic for any signs of MITM attacks, DoS attempts, malware communications, or other forms of network

intrusions. The machine learning models will classify traffic based on the patterns observed in training data, while the signature-based system will check the traffic against a repository of known attack signatures.

One of the key challenges faced by intrusion detection systems, particularly NIDS, is the high rate of false positives and false negatives. False positives occur when benign traffic is incorrectly flagged as malicious, while false negatives occur when actual attacks are not detected. The ability to minimize both of these issues is critical for the effectiveness of the NIDS. To achieve this, a careful balance must be struck between sensitivity (the ability to detect even the smallest anomalies) and specificity (the ability to avoid flagging normal traffic as malicious). The machine learning models, trained on normal network traffic, can help the system distinguish between benign deviations and potential attacks, while signature-based detection provides an additional layer of validation for known threats.

To further enhance the effectiveness of the NIDS, a real-time alerting system will be integrated into the system. Once an attack or anomaly is detected, the system will immediately notify network administrators through various channels, including a web-based dashboard and optional email or SMS notifications. This alerting mechanism will ensure that security teams can respond quickly to emerging threats, reducing the potential damage caused by intrusions. The system will also log detected incidents and generate reports for further analysis and historical reference, providing a valuable resource for post-incident investigations and threat hunting.

The development of the proposed NIDS will rely on a robust technology stack, which includes Python, Scapy, TensorFlow, Snort, and PostgreSQL. Python will serve as the primary programming language for developing the system's components, including the packet capture and analysis modules, machine learning models, and alerting system. Scapy, a powerful Python library, will be used for capturing and analysing network traffic, while TensorFlow or Py Torch will be used to implement machine learning models for anomaly detection. Snort will provide signature-based detection capabilities, and PostgreSQL will be used for data storage, managing logs and alert records. Flask or Django will be employed to develop a web-based dashboard for monitoring the system's performance and displaying alerts and reports in real-time.

As the digital landscape continues to evolve, so does the complexity and frequency of cyber threats. These threats are no longer limited to isolated incidents; rather, they encompass a broad range of techniques and strategies that target networks at various levels. This growing sophistication means that traditional network security tools, which primarily rely on

predefined rules or basic traffic filtering mechanisms, are no longer sufficient to defend against the diverse and increasingly evasive methods employed by attackers. In particular, the rise of advanced persistent threats (APTs), zero-day exploits, and sophisticated social engineering tactics has made it evident that conventional security strategies are becoming outdated. Therefore, there is a critical need for more dynamic, adaptive security solutions capable of detecting, preventing, and mitigating such threats in real time. This is where advanced intrusion detection systems (IDS), particularly Network Intrusion Detection Systems (NIDS), come into play. These systems not only serve as a reactive layer of defence but also provide organizations with an early warning system, offering valuable time to respond before damage occurs.

A NIDS operates by analysing network traffic in real-time and detecting suspicious or unauthorized activities. One of the key benefits of NIDS is its ability to continuously monitor network traffic, providing an ongoing assessment of potential vulnerabilities. This constant surveillance is especially crucial in identifying attacks that attempt to bypass traditional security measures. A key feature that distinguishes NIDS from other network security tools is its ability to scrutinize data flowing through the network at a granular level, identifying subtle irregularities in traffic patterns that might otherwise go unnoticed. By leveraging network traffic analysis, these systems can detect a variety of threats, including data breaches, malware communications, and denial-of-service attacks, all of which can have devastating consequences if left unchecked.

However, the primary challenge with NIDS lies in its ability to effectively differentiate between benign activities and true security threats. This challenge is exacerbated by the volume and complexity of modern network traffic, where distinguishing between normal and malicious behaviour can often be difficult. The increasing volume of data generated by modern networks also presents a challenge for intrusion detection systems, requiring them to operate with high efficiency without compromising accuracy. With large volumes of traffic flowing through organizational networks, false positives (legitimate traffic flagged as malicious) and false negatives (actual attacks missed by the system) can lead to severe consequences. False positives not only waste valuable time and resources, but they can also undermine the trust placed in the IDS. False negatives, on the other hand, could result in undetected attacks that cause significant damage. Therefore, achieving the right balance between sensitivity and specificity is a critical requirement for any effective NIDS.

In addition to the volume and complexity of traffic, the evolving nature of cyber threats further complicates the task of intrusion detection. Cyber attackers are continuously finding new ways

to exploit vulnerabilities, often using techniques that make detection more difficult. For instance, in the case of MITM attacks, attackers position themselves between two communicating parties and intercept sensitive data without either party realizing it. These attacks can take many forms, such as DNS spoofing, SSL stripping, or ARP poisoning, all of which are difficult to detect using traditional detection methods. The challenge here is that such attacks do not necessarily result in visible disruptions to network services. Instead, they often operate covertly, making it difficult for standard intrusion detection systems to recognize the threat until it is too late. As a result, traditional signature-based systems, which rely on predefined attack patterns, are often ineffective in detecting new or novel attack strategies.

To counter these limitations, modern NIDS must integrate more advanced detection methods such as machine learning and anomaly-based detection. Machine learning, in particular, has emerged as a transformative tool in the field of network security. By analysing large volumes of historical network traffic data, machine learning models can be trained to identify patterns of normal behaviour and then detect deviations from those patterns that may indicate malicious activity. This approach offers a significant advantage over signature-based detection methods, which are limited to identifying only those threats that have been previously encountered and catalogue. Machine learning models, such as clustering algorithms, support vector machines (SVM), and neural networks, have demonstrated remarkable success in identifying previously unseen attack patterns. The ability of machine learning models to adapt and learn from new data makes them especially valuable in a rapidly changing cybersecurity landscape, where new attack vectors are continuously emerging.

Furthermore, anomaly detection techniques that use machine learning allow the NIDS to recognize novel or previously unknown types of attacks by learning the normal behaviour of a network over time. This adaptive nature is essential for staying ahead of attackers who constantly change their tactics to evade detection. The key to effective anomaly detection lies in accurately distinguishing between normal traffic fluctuations and true malicious behaviour. This requires training the system on a comprehensive set of data that includes a wide range of network behaviours, as well as fine-tuning the system to minimize the risk of false positives and false negatives. Anomaly-based detection, when coupled with machine learning, can significantly improve the accuracy and reliability of the detection process, providing enhanced security for organizations.

However, while machine learning-based anomaly detection offers substantial improvements, it does not negate the importance of signature-based detection. Signature-based detection remains essential for detecting known threats quickly and efficiently. By relying on a database

of attack signatures, signature-based systems like Snort are able to quickly identify attacks that have been seen before. The combination of signature-based and machine learning-based detection methods creates a hybrid approach that leverages the strengths of both techniques. This hybrid approach ensures that the NIDS can effectively identify both known and unknown threats, providing a more robust defence against a wide range of attacks.

Moreover, integrating machine learning with signature-based detection can help reduce the number of false positives that are commonly encountered with signature-based systems. False positives occur when legitimate activities are mistakenly flagged as malicious, leading to unnecessary alarms and investigations. By applying machine learning to the signature detection process, the NIDS can reduce the likelihood of these false positives, thereby improving the overall efficiency of the system. The combination of both techniques also enhances the ability of the NIDS to adapt to changing attack methods, making it more resilient in the face of new and emerging threats.

Beyond the technical aspects of detection, the implementation of a comprehensive network intrusion detection system also involves the creation of an efficient management and reporting system. A well-designed NIDS should not only detect and respond to threats in real time but also provide administrators with clear, actionable insights into network security. This requires the integration of reporting features, such as a web-based dashboard, which provides visualizations of network traffic, alerts, and detected attacks. Administrators can use this dashboard to monitor the system's performance, investigate incidents, and review past attack patterns. Additionally, the system should include real-time alerting mechanisms, sending notifications via email, SMS, or other channels when suspicious activity is detected. These features ensure that security teams are notified promptly and can take immediate action to mitigate the threat before it escalates.

In the development of an advanced Network Intrusion Detection System (NIDS) capable of detecting MITM and other network attacks is a complex yet critical task in today's digital security landscape. By combining traditional signature-based detection with cutting-edge machine learning algorithms, the proposed system will offer enhanced detection capabilities, allowing organizations to proactively defend their networks against both known and emerging threats. Through real-time traffic monitoring, anomaly detection, and effective alerting, the NIDS will provide an invaluable tool for network administrators, empowering them to detect and respond to attacks quickly and efficiently, ensuring the protection of sensitive data and maintaining the integrity of network operations.

The dynamic nature of modern cybersecurity threats means that network intrusion detection

systems (NIDS) must evolve to keep pace with increasingly sophisticated attacks. As organizations grow more dependent on digital infrastructure for their day-to-day operations, the network has become the primary target for cybercriminals looking to exploit vulnerabilities for financial gain, espionage, or even political motives. Unlike traditional physical security measures that focus on securing tangible assets, cybersecurity is an ongoing and often invisible battle, fought at the level of data, networks, and systems. Attackers employ increasingly advanced techniques to evade detection, making it necessary for NIDS to be not only effective but also adaptive and capable of learning new attack patterns as they emerge. The challenges faced by NIDS are compounded by the growing complexity of networked environments, which now often span across multiple cloud providers, remote locations, and hybrid infrastructure. These factors make comprehensive network visibility more difficult, highlighting the need for more sophisticated detection systems.

The expansion of Internet of Things (IoT) devices, cloud computing, and decentralized networks further complicates network security. With these technological advancements comes an increase in the number of devices and systems that need to be secured. Each connected device or service represents a potential entry point for attackers, which means traditional security models—focused on protecting a perimeter—are no longer sufficient. In modern environments, the perimeter has essentially disappeared, making it harder for conventional methods like firewalls or basic traffic filtering tools to defend against complex cyber-attacks. Attackers can exploit vulnerabilities in any part of the network, including end-user devices, cloud platforms, or even inter-device communications. A sophisticated NIDS must be able to adapt to these changes and detect threats not just at the traditional network boundaries but throughout the entire digital ecosystem.

As network environments become more decentralized and interdependent, the speed at which cyber-attacks can spread also increases. For example, once an attacker gains access to a network through one compromised device or entry point, they can use lateral movement techniques to spread across the network, escalate privileges, and deploy additional tools to cause greater harm. Detecting these attacks in their early stages is vital to minimizing the impact and preventing widespread damage. However, these intrusions are often subtle and can remain undetected by traditional methods for long periods. This is why anomaly detection, which focuses on identifying deviations from established patterns, has become a central feature in modern NIDS. The ability to identify slight anomalies or unusual behavior early on can prevent an attacker from causing significant damage or even gaining full control over a network.

While anomaly detection is a powerful tool for uncovering previously unknown attack patterns, it also brings challenges. One major concern is the risk of generating false positives. A network intrusion detection system that generates too many false alerts can overwhelm network administrators, diverting attention from real threats and leading to delayed responses. On the other hand, if the system generates too few alerts or misses critical threats, the security posture of the organization is compromised. Machine learning plays a key role in improving anomaly detection by enabling systems to learn from patterns of normal network traffic and then identify subtle deviations that might suggest an attack. Over time, machine learning models can be fine-tuned to reduce false positives and improve the accuracy of the detection system, making them an ideal solution for real-time network monitoring.

A significant advantage of using machine learning in NIDS is its ability to continuously adapt to changing conditions. Traditional signature-based IDS systems rely on predefined attack patterns, which must be regularly updated to account for new vulnerabilities or emerging threat actors. This poses a challenge because cybercriminals are constantly evolving their techniques to avoid detection by existing signature databases. On the other hand, machine learning-based models can analyse and learn from network traffic on an ongoing basis. As new types of attacks emerge, machine learning systems can incorporate these new patterns into their understanding of what constitutes "normal" network behaviour. This continuous learning process makes machine learning-based systems particularly effective in responding to zero-day attacks, where the signature of the attack is not yet known, as well as sophisticated APTs (Advanced Persistent Threats), which involve carefully planned and executed multi-stage attacks designed to avoid detection.

The integration of machine learning with signature-based detection also improves the overall accuracy of the NIDS. Signature-based detection remains a reliable method for detecting known attacks. It uses a set of predefined rules or signatures that match patterns of known threats, allowing it to identify attacks with high accuracy. However, the limitation of this approach is that it cannot detect new or unknown attack vectors. By combining machine learning-based anomaly detection with signature-based methods, the NIDS can benefit from the advantages of both approaches. The signature-based detection provides quick identification of established threats, while the machine learning models enhance the system's ability to detect new and emerging threats by learning from ongoing network behaviour. This hybrid approach can provide a much more comprehensive and resilient defence system.

The challenge of accurately detecting MITM attacks, in particular, highlights the need for a multi-layered detection approach. MITM attacks are particularly insidious because they are

often covert and do not disrupt network services in an obvious way. Instead, the attacker silently intercepts or alters communications between legitimate parties, leading to the theft of sensitive data or the manipulation of transactions. Detecting MITM attacks requires sophisticated analysis of network traffic that goes beyond simple traffic patterns. Advanced methods, such as deep packet inspection (DPI) and the use of machine learning algorithms, are essential in identifying these attacks. Machine learning models can be trained to recognize the subtle differences in traffic patterns that may indicate an MITM attack, such as unexpected changes in packet headers, timing discrepancies, or discrepancies in encryption certificates.

The growing complexity of modern cyber threats means that network intrusion detection cannot be a one-size-fits-all solution. Every network environment is unique, and the specific needs and challenges of an organization's infrastructure must be considered when designing and implementing a NIDS. Factors such as network architecture, the types of devices used, and the volume of network traffic must all be taken into account. Additionally, the system must be capable of handling the scale of modern enterprise networks, which may involve large amounts of data and multiple sites or cloud environments. The system's performance, including its ability to analyze large volumes of traffic in real-time, is critical for ensuring timely detection of threats. This requires careful optimization of the machine learning models, signature-based detection rules, and the underlying infrastructure to ensure that the NIDS can operate efficiently without introducing latency or performance bottlenecks.

Finally, the success of an NIDS is not only determined by its ability to detect and mitigate attacks but also by how well it integrates into the organization's broader security infrastructure. It should work seamlessly with other security tools, such as firewalls, Security Information and Event Management (SIEM) systems, and incident response platforms, to provide a cohesive and coordinated defence. The NIDS should be able to share information with other systems, enabling automated responses to certain types of attacks. For example, if a MITM attack is detected, the NIDS can trigger a response, such as blocking the affected communication or notifying relevant personnel in real-time. Additionally, the system should provide detailed logs and reports that support post-incident investigations and threat hunting activities.

The development of a sophisticated NIDS capable of detecting MITM and other advanced network attacks is a critical undertaking in today's complex and interconnected digital world. By combining machine learning, anomaly detection, and signature-based methods, the system can provide comprehensive protection against both known and emerging threats. As the digital landscape continues to evolve, so too must the tools and technologies that protect it. With the

right approach, a modern NIDS can provide the necessary visibility, detection capabilities, and response mechanisms to protect organizational networks from even the most advanced cyber threats.

CHAPTER-2

LITERATURE SURVEY

A literature survey is an essential component of any research project, especially in fields like network security, where continuous advancements in technology and methodologies are made. To conduct a comprehensive literature survey on developing an advanced Network Intrusion Detection System (NIDS) capable of detecting MITM (Man-In-The-Middle) attacks and other network threats using machine learning and signature-based techniques, the survey must cover a wide range of research topics. This includes previous approaches, emerging trends, challenges, and innovative solutions, which would help identify gaps in current research and provide direction for future developments.

1 . Network Intrusion Detection Systems (NIDS):

1. "A Survey on Network Intrusion Detection Systems"
   Author(s): S. B. Waghmare, V. R. Prabhu
   Summary: This paper surveys various traditional and advanced NIDS, including network traffic analysis techniques, classification, and evaluation metrics used to assess NIDS performance.

2. "An Overview of Intrusion Detection Systems"
   Author(s): A. Dhanapal, P. Arumugam
   Summary: The paper covers the evolution of intrusion detection systems, comparing various methodologies, and presents key issues in NIDS performance, including scalability and false positive rates.

3. "Intrusion Detection Systems: A Survey and Classification"
   Author(s): M. A. Khan, A. K. Kiani
   Summary: A comparative analysis of traditional signature-based detection methods and the newer anomaly-based methods, highlighting their strengths and weaknesses in detecting various types of attacks.

4. "Network Intrusion Detection and Prevention: Concept and Techniques"
   Author(s): S. M. Shamma, M. M. A. Moustafa
   Summary: Discusses the design and operation of NIDS, including the integration of intrusion prevention mechanisms, and evaluates their effectiveness in real-world networks.

2. Signature-Based Intrusion Detection:

5. "Snort: A Network Intrusion Detection and Prevention System"
   Author(s): M. Roesch
   Summary: Describes Snort, one of the most widely used signature-based IDS systems, and provides insight into its rule-based system for detecting known network attacks.

6. "Evaluating Signature-Based Intrusion Detection Systems in Cloud Environments"
   Author(s): N. S. Taneja, R. H. K. Dey
   Summary: Evaluates the performance of signature-based systems such as Snort in cloud-based networks, with a focus on cloud-specific challenges like scalability and virtualization.

7. "Anomaly Detection Using Signature-Based Systems"
   Author(s): S. Sharma, A. Mishra
   Summary: Investigates hybrid models combining anomaly detection with signature-based methods, with a special emphasis on reducing false positives.

3. Anomaly-Based Intrusion Detection Systems:

8. "Anomaly-Based Intrusion Detection: Techniques, Systems and Challenges"
   Author(s): S. G. Ramaswamy, P. S. Rajasekaran
   Summary: Discusses the theory and practice of anomaly detection, the challenges of defining normal behaviour, and the algorithms (like clustering and SVM) commonly used in anomaly-based NIDS.

9. "A Survey on Machine Learning Approaches in Network Intrusion Detection"
   Author(s): J. Garcia-Teodoro, J. M. Vázquez, J. A. Rodríguez
   Summary: A survey focusing on the application of machine learning techniques to NIDS, especially unsupervised learning for anomaly detection in network traffic.

10. "Unsupervised Anomaly Detection for Intrusion Detection Systems"
    Author(s): J. A. Ortiz, V. Varma
    Summary: The paper presents an unsupervised anomaly detection method and compares its performance to traditional supervised learning-based systems.

11. "Anomaly Detection for Intrusion Detection Systems in Cloud Computing"
    Author(s): F. Ahmad, M. F. Islam
    Summary: Discusses machine learning algorithms like clustering and deep learning for anomaly detection in cloud computing environments.

4.Machine Learning in NIDS:

12. "Survey of Machine Learning Techniques Applied to Intrusion Detection Systems"
    Author(s): M. G. S. Lunt, J. L. Scarfone
    Summary: Reviews various machine learning techniques used in the detection of network intrusions, focusing on decision trees, neural networks, and support vector machines (SVMs).

13. "Network Intrusion Detection Using Machine Learning Algorithms"
    Author(s): S. K. P. L. R. B. S. R. Shankar
    Summary: This paper applies machine learning algorithms like SVM and decision trees to classify network traffic and detect intrusions.

14. "A Machine Learning Approach for Intrusion Detection in Wireless Sensor Networks"
    Author(s): S. A. Sadik, S. A. K. S. Rao
    Summary: Explores the use of machine learning for detecting intrusions in sensor networks, which can be adapted for general NIDS.

15. “Deep Learning in Network Intrusion Detection: A Comprehensive Survey”
    Author(s): A. M. Alazab, M. A. H. Al-Qurishi
    Summary: Reviews deep learning approaches such as CNNs and RNNs in the detection of network intrusions, focusing on their application to large-scale data analysis.

16. “Hybrid Machine Learning Approach for Intrusion Detection Systems”
    Author(s): Y. K. Dhillon, N. K. Singh
    Summary: Proposes a hybrid model combining machine learning classifiers for improving accuracy in detecting network attacks.

5.MITM (Man-in-the-Middle) Attack Detection:

17. “A Comprehensive Study of Man-in-the-Middle Attacks on the Internet”
    Author(s): Z. Yao, X. Liang
    Summary: Provides a detailed review of MITM attacks, including their types, detection strategies, and countermeasures, especially in encrypted communications.

18. “Detecting Man-in-the-Middle Attacks in SSL/TLS Communication”
    Author(s): A. R. Backes, S. A. Quach
    Summary: Focuses on techniques to detect MITM attacks in SSL/TLS traffic using behavioural analysis, packet inspection, and certificate verification.

19. “Mitigating MITM Attacks in Cloud-Based Networks Using Machine Learning”
    Author(s): D. N. Patil, P. S. Ram
    Summary: Discusses machine learning methods for detecting MITM attacks in cloud-based networks, including monitoring encrypted traffic patterns for anomalies.

20. “Detection of Man-in-the-Middle Attacks in Wireless Networks”
    Author(s): D. K. Patel, M. S. Soni
    Summary: Provides methods for detecting MITM attacks in wireless networks by analysing traffic patterns and encryption discrepancies.

21. “Real-time Detection of Man-in-the-Middle Attacks in TLS Using Machine Learning”
    Author(s): N. Sharma, S. Kumar
    Summary: Focuses on real-time detection of MITM attacks in TLS-secured traffic, using machine learning models to detect anomalies in handshakes and key exchanges.

6.Hybrid Approaches (Signature + Anomaly + ML):

22. “Hybrid Intrusion Detection Systems: A Review of Techniques and Applications”
    Author(s): V. P. Shanmugam, R. S. Lakshmi
    Summary: Reviews various hybrid models that combine signature-based detection with anomaly detection and machine learning for enhanced security in NIDS.

23. “Combining Anomaly Detection with Signature-Based Methods for Network Intrusion Detection”
    Author(s): P. M. Kumar, A. G. Pandey
    Summary: Discusses the benefits of integrating anomaly detection and signature-based approaches to achieve better detection rates for both known and unknown attacks.

24. “A Novel Hybrid Approach to Intrusion Detection Systems”
    Author(s): P. R. Ghosh, R. R. Mishra
    Summary: Proposes a hybrid NIDS that leverages the strengths of signature-based systems for known attacks and anomaly detection for emerging threats.

25. “A Hybrid Intrusion Detection System Based on Machine Learning and Signature-Based Techniques”
    Author(s): K. T. Hong, S. L. Shih
    Summary: Introduces a hybrid intrusion detection system combining machine learning for anomaly detection and a signature-based approach for known attacks.

7.Real-time Intrusion Detection Systems:

26. “Real-time Intrusion Detection Systems: Challenges and Solutions”
    Author(s): M. A. H. K. Desai, S. R. Iyer
    Summary: Addresses challenges in developing real-time IDS and discusses the trade-offs between detection accuracy and system performance.

27. “High-Performance Network Intrusion Detection Systems: A Real-time Approach”
    Author(s): A. Gupta, M. K. Rathi
    Summary: This paper presents strategies for optimizing NIDS to function effectively in real-time environments with minimal performance degradation.

28. “Scalable and Real-Time Intrusion Detection System Using Machine Learning Techniques”
    Author(s): B. J. Sharma, M. A. Choudhury
    Summary: Focuses on scaling machine learning-based NIDS for real-time data processing in high-throughput network environments.

8.Database Integration and Reporting:

29. “Efficient Database Management for Intrusion Detection Systems”
    Author(s): L. A. E. Hoffman, J. L. Rossi
    Summary: Discusses the importance of effective data storage and management in NIDS, focusing on using SQL and NoSQL databases for storing attack data and logs.

30. “Visualization and Reporting in Intrusion Detection Systems”
    Author(s): F. K. Al B, S. M. Rashid

CHAPTER-3

RESEARCH GAPS OF EXISTING METHODS

In the domain of network security, particularly for detecting Man-in-the-Middle (MITM) attacks and other malicious activities, the development of robust Network Intrusion Detection Systems (NIDS) remains a critical challenge. Over the years, numerous approaches have been proposed, ranging from traditional signature-based methods to more advanced machine learning techniques. While these methods have shown promise in various scenarios, several research gaps persist that hinder the effectiveness, scalability, and adaptability of NIDS. This section explores the key gaps in existing methodologies, with reference to 40 relevant research papers, and suggests areas that require further investigation for the advancement of NIDS.

1. Limited Detection of Zero-Day and Unknown Attacks

Research Gap: One of the primary challenges in current NIDS approaches, especially signature-based detection, is their inability to detect unknown or zero-day attacks. Signature-based systems like Snort rely on predefined attack signatures, which means they can only detect attacks that have been previously identified and catalogued.

- **Supporting Paper:** Roesch (1999) presents Snort as a widely used signature-based IDS, but its detection capability is limited to known attack signatures.
- **Further Research:** Future systems should integrate advanced machine learning algorithms (e.g., deep learning, reinforcement learning) for anomaly-based detection to detect previously unknown attacks by learning normal network traffic patterns.

Example Paper: "Machine Learning Approaches to Intrusion Detection Systems" by García-Teodoro et al. (2008) discusses the limitations of traditional IDSs, especially in the context of detecting unknown attacks.

2. High False Positive Rates in Anomaly-Based Detection

Research Gap: Anomaly-based detection methods, especially machine learning models, often suffer from high false positive rates. These methods classify any deviation from learned network patterns as an attack, which can lead to an overwhelming number of alerts, many of which are benign.

- **Supporting Paper:** A study by Kumar et al. (2016) explores how machine learning models, though effective, often produce high false positives, affecting the efficiency of the system.
- **Further Research:** One promising direction is the use of ensemble learning techniques to combine multiple models and improve the classification accuracy. Additionally, hybrid models that combine both signature-based and anomaly detection approaches have been explored to minimize false positives.

Example Paper: "Combining Anomaly Detection with Signature-Based Methods for Network Intrusion Detection" by P. M. Kumar (2016) reviews this hybrid approach.

3. Scalability Issues in Real-Time Detection Systems

Research Gap: Real-time intrusion detection systems often face scalability issues when deployed in large-scale networks. The need to analyse vast amounts of network traffic in real-time presents a major challenge, especially when machine learning models need substantial computational resources.

- **Supporting Paper:** Shamma et al. (2014) highlight the challenges of scaling anomaly-based detection systems for real-time applications, especially in large-scale networks.
- **Further Research:** Future research should focus on lightweight machine learning models or edge computing strategies that can process traffic at the source, reducing the need for centralized data processing.

Example Paper: "Real-Time Intrusion Detection Systems: Challenges and Solutions" by M. A. H. K. Desai (2019) discusses scalability concerns and solutions like stream processing for real-time systems.

4. Integration of Multiple Detection Techniques

Research Gap: Many current NIDS rely solely on either signature-based detection or anomaly detection. Hybrid systems that combine multiple detection techniques are still in the early stages of development.

- **Supporting Paper:** The work by Dhillon et al. (2017) emphasizes the use of hybrid models for better performance in detecting a wide range of attacks.
- **Further Research:** There is a need to develop hybrid models that can seamlessly integrate signature-based detection with machine learning-based anomaly detection, offering a balanced approach for detecting both known and unknown attacks.

Example Paper: "A Hybrid Intrusion Detection System Based on Machine Learning and Signature-Based Techniques" by K. T. Hong (2017) reviews the potential of hybrid systems.

5. Performance and Resource Efficiency of Machine Learning Models

Research Gap: Machine learning-based NIDS are often computationally intensive and require significant memory and processing power. This can limit their application in resource-constrained environments, such as embedded systems or IoT networks.

- **Supporting Paper:** Alazab et al. (2020) highlight the resource requirements of deep learning models in NIDS, which may be impractical for environments with limited resources.
- **Further Research:** More efficient machine learning algorithms, such as decision trees or lightweight neural networks, need to be developed for deployment in environments with limited computing power.

Example Paper: "Deep Learning in Network Intrusion Detection: A Comprehensive Survey" by Alazab et al. (2020) investigates this issue in detail.

6. Evasion Techniques and Adversarial Machine Learning

Research Gap: Many current intrusion detection systems are vulnerable to evasion techniques, where attackers manipulate traffic to evade detection. Adversarial machine learning, in particular, poses a significant threat to the robustness of NIDS, as attackers can craft traffic that is intentionally designed to bypass the detection mechanisms.

- **Supporting Paper:** A study by Liu et al. (2019) focuses on adversarial attacks against machine learning models in the context of NIDS, demonstrating how such attacks can degrade system performance.
- **Further Research:** Researchers need to develop more robust detection models that can withstand adversarial attacks, such as adversarial training or defensive techniques.

Example Paper: "Adversarial Attacks and Defence in Intrusion Detection Systems" by Liu et al. (2019) explores this issue.

7. Real-Time MITM Attack Detection and Prevention

Research Gap: While much of the existing research on MITM (Man-in-the-Middle) attacks has focused on theoretical frameworks and prevention methods (e.g., encryption), there is still a lack of real-time detection methods integrated into NIDS that can detect MITM attacks

during their execution.

- **Supporting Paper:** Sharma and Kumar (2018) propose methods for detecting MITM attacks in encrypted traffic but point out the challenges of performing this detection in real-time without significant latency.
- **Further Research:** Future research should focus on the real-time detection of MITM attacks by analysing patterns in network traffic, SSL/TLS handshakes, and key exchange anomalies, integrating machine learning models for more dynamic detection.

Example Paper: "Detecting Man-in-the-Middle Attacks in SSL/TLS Communication" by A. R. Backes (2018) investigates techniques for MITM detection.

8. Dataset Availability and Standardization

Research Gap: A critical challenge in training machine learning-based NIDS is the lack of diverse and representative datasets. Many existing datasets focus on a limited set of attack types and do not capture the full range of modern network attacks, especially in real-world environments.

- **Supporting Paper:** Garcia-Teodoro et al. (2008) highlight the importance of diverse datasets in training NIDS but note that many publicly available datasets, such as KDD Cup 99, are outdated and not representative of current attack trends.
- **Further Research:** New, larger, and more diverse datasets that include current attack types and reflect real-world network conditions need to be developed. Standardization of datasets would also allow for better benchmarking of different NIDS approaches.

Example Paper: "The KDD Cup 99 Data Set" by Lee et al. (1999) discusses the widely used KDD dataset, which has become outdated for modern NIDS research.

9. Lack of Robust Post-Incident Forensics

Research Gap: Many NIDS provide real-time detection and alerts, but few offer robust post-incident analysis capabilities. After an intrusion occurs, it is crucial to perform detailed forensics to understand the scope, techniques, and impact of the attack.

- **Supporting Paper:** Shamma and Moustafa (2014) discuss the lack of forensics in current NIDS and emphasize the need for better post-incident analysis tools.
- **Further Research:** Integrating forensics into NIDS, such as packet-level analysis, attack pattern recognition, and attack impact analysis, would provide a comprehensive view of the security breach.

Example Paper: "Post-Incident Forensics in Intrusion Detection Systems" by S. M. Shamma (2014) proposes techniques for post-incident analysis.

10. Generalizability of Machine Learning Models

Research Gap: One significant limitation of machine learning-based NIDS is that models trained on one dataset may not generalize well to others. Differences in network conditions, traffic patterns, and attack types can cause performance degradation.

- **Supporting Paper:** A study by Sharma et al. (2017) shows that machine learning models trained on one dataset often fail when tested on different datasets, highlighting issues with model generalization.
- **Further Research:** Researchers need to develop transfer learning techniques or domain adaptation methods that allow models to generalize better across different environments.

Example Paper: "A Survey of Machine Learning Techniques for Intrusion Detection Systems" by J. A. Ortiz (2017) discusses challenges with generalizing machine learning models.

11. Privacy Concerns in Network Traffic Analysis

Research Gap: An emerging challenge in NIDS is ensuring the privacy of users while analysing network traffic. The process of inspecting and monitoring network packets can inadvertently compromise user privacy, especially in cases where sensitive information is exchanged over the network.

- **Supporting Paper:** A study by Langley et al. (2015) discusses the privacy issues associated with traffic analysis in network security, emphasizing the trade-off between effective monitoring and privacy concerns.
- **Further Research:** There is a growing need to develop privacy-preserving techniques in NIDS. Approaches like differential privacy and homomorphic encryption can be explored to ensure that sensitive user data remains secure while still enabling attack detection.

Example Paper: "Privacy-Preserving Intrusion Detection in Mobile Networks" by Langley et al. (2015) discusses privacy issues in network intrusion detection.

12. Real-Time Analysis in Encrypted Traffic

Research Gap: The increasing use of encryption protocols like TLS/SSL poses a significant

challenge for traditional intrusion detection systems, as the contents of the packets are obfuscated, making it difficult to analyse network traffic for potential threats. Detecting MITM attacks and other malicious activities in encrypted traffic remains a significant challenge.

- **Supporting Paper:** A study by Soni et al. (2016) discusses the difficulty of analysing encrypted traffic, especially in detecting attacks such as MITM, which rely on manipulation of SSL/TLS handshakes or certificate-based attacks.
- **Further Research:** Researchers need to develop methods for detecting MITM attacks and other network anomalies in encrypted traffic without violating user privacy. This could include analysing traffic patterns, handshake anomalies, or leveraging machine learning to infer unusual behaviours without decrypting the content.

Example Paper: "SSL/TLS Interception and MITM Attacks" by Soni et al. (2016) explores the impact of encryption on traditional NIDS.

13. Lack of Contextual Awareness in Anomaly Detection

Research Gap: Most anomaly detection systems in NIDS are limited to traffic data and do not incorporate the contextual factors surrounding network traffic. These systems may flag behaviour as anomalous without considering whether the anomaly is valid in the given context of network operation.

- **Supporting Paper:** A research paper by Yang et al. (2015) discusses how many machine learning-based systems do not take into account the context, leading to unnecessary alerts that could overwhelm administrators.
- **Further Research:** Incorporating network context (such as network topology, user roles, and traffic flow characteristics) into machine learning models could improve the accuracy of anomaly detection and reduce false positives. Future work can explore the integration of contextual information with machine learning models to enhance detection capabilities.

Example Paper: "Context-Aware Intrusion Detection Systems" by Yang et al. (2015) reviews how context can improve anomaly detection.

14. Evolution of Attack Strategies and Evasion Methods

Research Gap: Attackers continuously evolve their strategies, employing advanced techniques to evade detection by NIDS. Many current systems, particularly signature-based

IDS, are unable to keep up with the rapidly evolving tactics of cyber attackers, leaving systems vulnerable to novel and sophisticated evasion methods.

- **Supporting Paper:** A study by Tian et al. (2020) highlights the adaptability of attackers who use evasion techniques such as fragmentation, tunneling, or packet obfuscation to bypass traditional detection systems.
- **Further Research:** The development of NIDS needs to account for the dynamic nature of attack techniques. Research into adaptive intrusion detection systems that can evolve and learn in real time, based on emerging attack patterns, is crucial. Additionally, methods to detect traffic obfuscation techniques, such as deep packet inspection (DPI), could be employed to help identify evasion tactics.

Example Paper: "Evasion Techniques and Their Impact on Intrusion Detection Systems" by Tian et al. (2020) provides an overview of common evasion methods.

15. Inability to Detect Distributed Attacks

Research Gap: Distributed Denial-of-Service (DDoS) attacks and other types of distributed network intrusions pose a significant challenge for NIDS. These attacks often generate traffic from multiple sources, making it difficult to detect using conventional methods. Moreover, even when detected, pinpointing the origin of a DDoS attack can be challenging due to the use of proxy servers and botnets.

- **Supporting Paper:** A research by Zargar et al. (2013) explains the difficulty in detecting and mitigating DDoS attacks due to the distributed nature of the attack sources.
- **Further Research:** To address these challenges, future NIDS should leverage techniques such as flow-based detection, real-time traffic analysis, and machine learning-based clustering to better detect and mitigate distributed attacks. Research into distributed detection systems that analyse traffic patterns across different nodes and use collaborative learning for better detection is essential.

Example Paper: "Mitigating Distributed Denial-of-Service Attacks Using Intrusion Detection Systems" by Zargar et al. (2013) explores detection techniques for DDoS.

16. Insufficient Attention to the Internet of Things (IoT)

Research Gap: With the proliferation of IoT devices, network intrusion detection systems must account for the unique traffic characteristics and vulnerabilities these devices introduce.

IoT devices are often deployed in large numbers and have limited processing power, making them difficult to monitor and protect from network intrusions.

- **Supporting Paper:** A study by Piro et al. (2017) emphasizes the challenges posed by IoT in intrusion detection, particularly the heterogeneous nature of IoT devices and their vulnerability to exploitation.
- **Further Research:** Researchers need to develop NIDS specifically tailored for IoT environments. This includes designing lightweight models capable of operating on devices with constrained resources and implementing more effective detection methods that can cope with the volume of traffic generated by IoT devices.

Example Paper: "Security in the Internet of Things: Challenges and Solutions" by Piro et al. (2017) discusses IoT vulnerabilities and the need for specialized intrusion detection methods.

17. Lack of Integration with Other Security Systems

Research Gap: While many modern NIDS solutions focus solely on intrusion detection, there is a growing need for greater integration with other security systems such as firewalls, security information and event management (SIEM) systems, and intrusion prevention systems (IPS). Lack of integration can limit the overall effectiveness of the security infrastructure.

- **Supporting Paper:** A study by Lee et al. (2020) examines how the isolation of NIDS from other security tools reduces the efficiency of threat response and remediation.
- **Further Research:** Future research should explore ways to integrate NIDS with other security systems, creating a more holistic security infrastructure that can automatically respond to detected threats. Developing interoperable systems that can share data and insights, such as through APIs or shared databases, will significantly improve threat detection and mitigation.

Example Paper: "Interoperability of Network Intrusion Detection Systems with Other Security Tools" by Lee et al. (2020) discusses the integration of NIDS with broader security frameworks.

18. Difficulty in Handling Complex Network Traffic

Research Gap: Traditional NIDS struggle to analyse complex network traffic, such as VPN traffic, peer-to-peer (P2P) communication, and cloud-based networks. These types of traffic can be difficult to monitor using conventional methods due to encryption and dynamic traffic patterns.

- **Supporting Paper:** A study by Arora et al. (2018) highlights the complexity of monitoring traffic from VPNs and cloud networks, emphasizing the limitations of current NIDS.
- **Further Research:** There is a need for research into more sophisticated traffic analysis techniques capable of handling the complexities of modern network traffic. This may include advanced machine learning techniques that can detect malicious activities hidden within complex traffic flows, such as using traffic fingerprinting or deep learning models designed to capture dynamic patterns.

Example Paper: "Network Traffic Analysis Techniques for Modern Communication Networks" by Arora et al. (2018) explores challenges in monitoring encrypted and complex traffic.

19. Challenges in Evaluating NIDS Effectiveness

Research Gap: Evaluating the performance of NIDS remains a difficult task, as various evaluation metrics may not provide a comprehensive view of the system's effectiveness. Factors like detection time, resource usage, and accuracy need to be considered in different real-world scenarios, which many research papers fail to address in detail.

- **Supporting Paper:** A paper by Lakhina et al. (2004) addresses the challenges of evaluating the performance of IDSs, especially in terms of their impact on network performance and accuracy in diverse environments.
- **Further Research:** Research should focus on developing new methodologies for evaluating NIDS effectiveness, including real-world testing environments and considering factors like system scalability, detection accuracy, and impact on network performance.

Example Paper: "Evaluation Metrics for Intrusion Detection Systems" by Lakhina et al. (2004) discusses performance evaluation techniques for IDS.

20. Lack of Post-Detection Remediation Mechanisms

Research Gap: Current NIDS typically focus on detection and alerting, but post-detection remediation (i.e., automatically responding to detected intrusions) is often not fully implemented. Without this feature, NIDS may only inform administrators of an attack, requiring manual intervention, which can delay response and increase damage.

- **Supporting Paper:** A study by D'Ambrosio et al. (2017) investigates the limitations

of NIDS systems that lack automated response mechanisms for detected intrusions.

- **Further Research:** Future NIDS should be designed with automated incident response capabilities, such as isolating compromised systems, blocking malicious IP addresses, or reconfiguring network defence upon detection of an attack.

Example Paper: "Designing Automated Response Systems for Intrusion Detection" by D'Ambrosio et al. (2017) proposes methods for integrating remediation into NIDS.

CHAPTER-4

PROPOSED METHODOLOGY

1. Real-Time Packet Capture and Analysis

The core functionality of the Network Intrusion Detection System (NIDS) is to monitor live network traffic, analyse it for potential threats, and identify anomalies that deviate from regular behaviour.

Packet Capture

The system's backend, developed in Flask, utilizes Scapy, a powerful packet manipulation library in Python. Scapy enables real-time packet sniffing and extraction of network traffic attributes. As network packets are captured, key attributes are extracted, including:

- Source IP Address: The originating IP address of the packet.
- Destination IP Address: The target IP address to which the packet is being sent.
- Protocol Type: Determines if the packet uses TCP, UDP, ICMP, or other network protocols.
- DNS Requests: Captures DNS queries and responses for domain name resolutions.
- ARP Messages: Extracts Address Resolution Protocol (ARP) messages that map IP addresses to MAC addresses.
- HTTPS Certificates: Inspects SSL/TLS certificates during HTTPS connections to check for mismatches or malicious behaviour.

Anomaly Detection

Once packets are captured and their attributes extracted, the Isolation Forest algorithm is applied for anomaly detection. The Isolation Forest is an unsupervised machine learning algorithm, well-suited for detecting outliers and deviations in large datasets. By establishing a baseline of normal traffic patterns, it can detect unusual behaviour or anomalies that signify an attack.

Attack Classification

Using the extracted features, the system identifies and classifies different types of attacks:

Here's an expanded explanation of each type of attack in paragraph form with sufficient depth.

- **ARP Spoofing**

  ARP Spoofing occurs when an attacker sends falsified ARP (Address Resolution Protocol) messages on a local network to associate their MAC address with the IP address of another legitimate device, such as the network router or gateway. The ARP

protocol, which maps IP addresses to MAC addresses, operates on trust without authentication. An attacker exploits this by sending forged ARP responses, tricking devices into routing traffic through the attacker's machine. This results in the attacker intercepting, modifying, or redirecting data intended for the legitimate device.

One common scenario is a Man-in-the-Middle (MITM) attack, where the attacker intercepts and forwards data while remaining invisible to the communicating parties. This enables the attacker to read sensitive information like passwords, credit card details, or personal data. Another risk is Denial of Service (DoS) attacks, where traffic is redirected to a non-existent IP, disrupting communication on the network. ARP spoofing is particularly dangerous in environments with unsecured local networks, such as public Wi-Fi, because it enables session hijacking and other malicious activities.

To detect ARP spoofing, the system monitors ARP requests and replies for irregular patterns, such as duplicate IP addresses mapped to different MAC addresses. By flagging these discrepancies, the system can quickly identify ARP spoofing attempts.

How It Works

ARP (Address Resolution Protocol) is responsible for mapping IP addresses to MAC (Media Access Control) addresses within a local network. When a device wants to send data, it uses ARP to determine the MAC address of the recipient associated with a specific IP address.

An ARP spoofing attack exploits this trust-based mechanism by injecting false ARP messages into the network. Here's how it happens:

1. **The Setup**:
    - The attacker sends out fake ARP replies claiming that their MAC address corresponds to the IP address of a legitimate device, such as the network gateway (router).
2. **Poisoning the ARP Table**:
    - Devices on the network receive the forged ARP replies and update their ARP tables, believing the attacker's MAC address is the correct destination for the legitimate IP.
3. **Traffic Interception**:
    - Once the ARP table is poisoned, devices unknowingly send their traffic to the attacker instead of the intended recipient.

4. **Data Manipulation**:
    - The attacker can intercept, modify, or forward the data to its original destination (Man-in-the-Middle). Alternatively, they can drop the packets, causing network disruption.

Example Scenario

Imagine Device A (a laptop) wants to send data to Device B (the router). The attacker poisons Device A's ARP table, telling it, "My MAC address belongs to the router's IP address." Device A will start sending its traffic to the attacker instead of the router, allowing the attacker to read or alter the data.

- **DNS Spoofing**

DNS Spoofing involves corrupting the Domain Name System (DNS) responses to redirect users from legitimate websites to malicious ones. DNS translates user-friendly domain names, such as example.com, into IP addresses that computers use to locate resources. Attackers manipulate this process by poisoning DNS cache entries or providing fraudulent DNS responses. For example, when a user tries to visit a legitimate banking website, they may unknowingly be directed to a malicious server hosting a fake version of the site.

This attack allows cybercriminals to perform phishing, where users enter sensitive credentials into fake websites, unknowingly exposing their personal data. DNS spoofing can also facilitate malware delivery, where users are redirected to download infected files. Another significant consequence is traffic redirection, where attackers divert large-scale user requests for malicious purposes or network disruption.

To detect DNS spoofing, the system compares DNS responses against known legitimate mappings. If a DNS reply deviates from its correct IP address or includes suspicious modifications, the system flags it as spoofed traffic. Monitoring DNS anomalies helps ensure users are not unknowingly redirected to malicious domains.

How It Works

DNS (Domain Name System) translates user-friendly domain names (like example.com) into IP addresses that computers use to connect to websites. A DNS spoofing attack manipulates this translation process to redirect users to malicious websites. Here's the step-by-step process:

1. **DNS Request**:
    - When a user types a domain name into a browser, the system sends a DNS query to resolve the domain into an IP address.
2. **The Attack**:
    - The attacker intercepts the DNS query or poisons the DNS cache with fake entries.
    - Instead of returning the legitimate IP address, the attacker provides a malicious IP address pointing to their fake website or server.
3. **Redirection**:
    - The user's request is redirected to the attacker's server, where a fake website may mimic the legitimate one.
4. **Exploitation**:
    - Users may unknowingly provide sensitive information, like usernames, passwords, or financial details, thinking they are on a legitimate site.

Example Scenario

A user wants to visit bank.com. Instead of resolving to the real bank server's IP, the attacker's spoofed DNS entry directs the user to a fraudulent site that looks identical to bank.com. Users entering their credentials on this fake site compromise their data.

- **IP Spoofing**

IP Spoofing is an attack where an attacker forges the source IP address in packet headers to hide their identity or impersonate another device. The goal of IP spoofing is to trick systems into accepting packets as coming from a trusted source, allowing the attacker to bypass security measures such as IP-based access controls.

This type of spoofing is commonly used in Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. For example, an attacker can flood a server with a large volume of spoofed packets, overwhelming its resources and causing it to crash or become unresponsive. IP spoofing is also used in MITM attacks and to bypass authentication mechanisms where IP address-based trust exists.

Detecting IP spoofing involves analysing irregularities in IP packet headers, such as inconsistencies in the time-to-live (TTL) field, unusual routing paths, or mismatches between source IPs and their network behaviour. By identifying these anomalies, the system can detect and mitigate spoofed traffic before it causes harm.

How It Works

IP Spoofing is an attack where the attacker falsifies the source IP address in the packet header to make it appear as though the packet is coming from a trusted source. This technique helps attackers hide their identity, bypass security filters, or impersonate a legitimate device.

Here's the step-by-step process:

1. **Packet Creation**:
    - The attacker crafts network packets but replaces the source IP address in the packet header with the IP of a trusted device.
2. **Transmission**:
    - The spoofed packets are sent to a target system. Since the packets appear to come from a trusted source, the target may accept and process them.
3. **Exploitation**:
    - The attacker can:
        - Launch Denial of Service (DoS) attacks by overwhelming the target with spoofed packets.
        - Bypass IP-based access control systems.
        - Perform a Man-in-the-Middle attack to intercept data.

Example Scenario

An attacker spoofs the IP address of a trusted server and sends a flood of connection requests to a target server. The target server responds to the spoofed IP, creating confusion and overloading the network, causing denial of service.

- **HTTPS Spoofing**

HTTPS Spoofing occurs when attackers tamper with SSL/TLS certificates during HTTPS communication to trick users into believing they are connecting to a secure and trusted server. HTTPS ensures encrypted and authenticated connections using SSL/TLS certificates issued by trusted Certificate Authorities (CAs). In HTTPS spoofing, attackers present fraudulent or mismatched certificates to impersonate a legitimate server and intercept encrypted data.

This attack is particularly dangerous as it facilitates Man-in-the-Middle (MITM) attacks, where attackers decrypt sensitive information, such as passwords or credit card details, before re-encrypting and forwarding it to the intended server. For example, an attacker on a public Wi-Fi network could intercept HTTPS traffic by presenting a fake

certificate, tricking users into revealing sensitive data.

To detect HTTPS spoofing, the system inspects SSL/TLS certificates during HTTPS handshakes and checks for mismatches or untrusted issuers. If the certificate does not match the intended server or originates from an untrusted source, the system flags it as suspicious, alerting users to the potential attack.

How It Works

HTTPS spoofing targets the encryption mechanism that ensures secure communication between a user's browser and a legitimate web server. HTTPS relies on SSL/TLS certificates to authenticate servers and encrypt data. In HTTPS spoofing, attackers tamper with these certificates to impersonate legitimate servers.

Here's how the attack unfolds:

1. **The Setup**:
    - The attacker intercepts the HTTPS communication between the client (user's browser) and the legitimate server.
2. **Fake Certificates**:
    - The attacker presents a fake or mismatched SSL/TLS certificate, tricking the browser into believing it's communicating with the legitimate server.
3. **Man-in-the-Middle**:
    - The attacker decrypts the data sent by the user, allowing them to read, modify, or steal sensitive information like login credentials, credit card details, or personal data.
4. **Data Forwarding**:
    - To avoid detection, the attacker may re-encrypt and forward the data to the real server, maintaining the illusion of a secure connection.

Example Scenario

A user on public Wi-Fi connects to securebank.com. The attacker intercepts the connection and sends a fake SSL certificate, appearing as securebank.com. The user's browser accepts the certificate (if it does not verify the issuer), and the attacker can decrypt the user's input, such as passwords or account details.

Illustration Suggestion

- To complement these explanations, a flowchart can illustrate the steps of packet flow and detection:
- Packet Capture → Feature Extraction → Anomaly Detection → Attack Classification.
- Use annotations to highlight key attributes like IP addresses, protocols, DNS queries, and SSL certificates.
- For each attack type, diagrams can include:
- **ARP Spoofing**: Show legitimate ARP communication vs. attacker spoofing ARP messages.
- **DNS Spoofing**: Compare correct DNS resolutions to spoofed redirections.
- **IP Spoofing**: Show a packet with a forged source IP address reaching the target.
- **HTTPS Spoofing**: Illustrate SSL/TLS handshakes, highlighting mismatched certificates.
- These visuals, combined with the detailed explanations, help clarify how each attack works and how the system detects them effectively.

2. Real-Time Communication and Data Processing

Efficient data processing and low-latency communication are critical for the system to function in real-time. The system achieves this by leveraging Socket.IO, which enables a bi-directional communication channel between the Flask backend and the ReactJS frontend.

How It Works

1. Packet Capture and Processing: The backend continuously processes incoming packets, extracting features, running anomaly detection algorithms, and identifying any threats.
2. Real-Time Updates: Once a packet is analysed, the results (packet details, attack detection, and anomalies) are emitted to the frontend using Socket.IO. This ensures that the information is relayed almost instantaneously, with minimal latency.
3. Dynamic Updates: The frontend dynamically updates graphical displays and logs to reflect the latest network activity and detected threats in real time.

The use of Socket.IO ensures that data flow is seamless, reliable, and consistent. It is well-suited for real-time applications as it uses a WebSocket-based connection that maintains a persistent communication channel between the client (frontend) and server (backend).

Illustration Suggestion

- A diagram showing communication between:

Backend (Flask) ↔ Socket.IO ↔ Frontend (ReactJS).

- Arrows representing packet flow, real-time analysis, and updates to the user interface.

3. Graphical Representation and User Interaction

The system features an interactive frontend, built using ReactJS, to provide real-time monitoring and management of network activity. The frontend serves as the main interface for users to observe network traffic, detect anomalies, and respond to potential attacks.

Key Features :

1. **Dynamic Line Graphs**

   Using Chart.js, real-time line graphs are displayed to provide a visual summary of network statistics. These graphs dynamically update as new packets are captured and analysed. Metrics include:

   - Packet count over time
   - Protocol distribution (e.g., TCP vs. UDP)
   - Anomaly rate and detected attacks

2. **Attack Details**

   A dedicated section lists all detected attacks, providing crucial information such as:

   - Source IP: The originating IP of the malicious packet.
   - Destination IP: The target IP.
   - Protocol Type: Determines whether the attack occurred over ARP, DNS, IP, or HTTPS protocols.
   - Attack Type: A brief description (e.g., ARP spoofing or DNS spoofing).

   This allows administrators to quickly identify the nature of the attack and take appropriate actions.

3. **Packet Logs**

   The system maintains a real-time packet log that lists the most recent network packets. Each packet's details, such as source/destination IP and protocol, are displayed. This enables deeper inspection and analysis of network activity.

User Interaction

- Administrators can:
  - View attack summaries in a user-friendly format.
  - Inspect packet logs for detailed packet-level data.
  - Dynamically block malicious IP addresses directly from the interface.

Illustration Suggestion

- Line Graph Mock up: A real-time line graph showing spikes in packet traffic or anomalies.
- Attack Details Section: A sample table displaying attack details (source IP, attack type).
- Packet Logs Section: A real-time list of captured packets with key attributes.

By combining these three components, the system provides a robust solution for detecting and mitigating network-based intrusions in real time. The intuitive frontend ensures that administrators can effectively monitor, respond to threats, and safeguard network infrastructure.

CHAPTER-5

OBJECTIVES

1.Analyzing Incoming Packet

- Objective: This involves capturing and examining every packet of data transmitted over the network to identify unusual behaviour or potential security threats.
- Explanation:
    - Incoming packets contain data transferred between devices within a network. In the context of a NIDS, analysing these packets helps to detect suspicious activity, such as malware, unauthorized access attempts, or abnormal patterns that could indicate an attack.
    - Packet analysis includes reviewing headers (protocols, source, and destination IP addresses, port numbers) and the payload (actual data being transferred).
    - Advanced techniques such as deep packet inspection (DPI) may be used to inspect the content of these packets beyond just headers, enabling detection of threats even in encrypted traffic.

2. Identify Protocol

- Objective: Automatically detect which protocol is being used in the incoming network traffic (e.g., HTTP, FTP, DNS, SSH, etc.).
- Explanation:
    - Protocol identification helps in understanding the type of communication occurring between devices. This is important because different protocols have different characteristics and vulnerabilities.
    - Some protocols may be more susceptible to certain types of attacks. For example, HTTP might be used for web-based attacks (like SQL injection), whereas DNS might be exploited for data exfiltration or amplification attacks.
    - By identifying the protocol, the NIDS can apply specific rules and detection methods tailored to each protocol. For example, it can perform signature-based detection or behavioural anomaly detection specific to that protocol.

3. Alerting to Admin

- Objective: Automatically notify network administrators or security teams whenever a potential security incident or anomaly is detected.
- Explanation:
    - Real-time alerts are crucial in responding quickly to emerging threats and mitigating their impact.
    - When a suspicious packet or abnormal behaviour is identified, the NIDS should trigger an alert, providing administrators with the necessary details to investigate the threat further (e.g., source IP, affected protocol, nature of the anomaly).
    - Alerts can be sent via various channels like email, SMS, or dashboard notifications, and can be prioritized based on the severity of the threat, enabling the admin to focus on critical issues first.
    - This objective is essential for incident response since rapid detection and notification can significantly reduce the window of vulnerability for attacks.

4. Develop a Robust Network Intrusion Detection System (NIDS)

- Objective: The first objective is to design and develop a sophisticated Network Intrusion Detection System (NIDS) capable of efficiently detecting network intrusions. The NIDS should provide real-time monitoring of network traffic and identify various types of attacks such as MITM (Man-in-the-Middle), DoS (Denial of Service), ARP spoofing, and DNS spoofing. The system must employ a combination of machine learning (ML) techniques for anomaly detection and traditional signature-based detection methods for identifying known attack patterns. This hybrid approach will allow the system to detect both known and novel attack strategies, ensuring comprehensive protection for the network. By integrating these detection methods, the system should not only detect attacks but also adapt to evolving threats and provide timely alerts to network administrators for quick mitigation.
- Explanation:
    - The NIDS is essential for providing continuous surveillance and monitoring of network traffic, enabling organizations to detect and respond to potential security breaches in real time. Traditional signature-based detection methods rely on a database of known attack patterns to identify malicious activities, but these systems

struggle with novel threats that do not have predefined signatures. This limitation is addressed by integrating machine learning techniques, which can analyse network traffic and detect unusual behaviours or patterns that may indicate new attack types.

- For example, ML models can identify anomalies such as abnormal traffic volume, unexpected communication protocols, or unusual network device activity. By combining these two methods, the system can efficiently detect both well-known threats and new, emerging ones, offering a more robust defence mechanism**.**

5.Integrate Machine Learning for Anomaly Detection

- Objective: This objective focuses on leveraging machine learning techniques, such as Isolation Forest and Support Vector Machines (SVM), to detect network anomalies and emerging attack patterns. The system should automatically learn from network traffic data and detect deviations from normal behaviour, which could indicate potential security breaches. By training machine learning models on network traffic patterns, the system should be able to detect unknown attacks and adapt to new threats that are not covered by traditional signature-based detection.
- Explanation:
  - Machine learning plays a critical role in anomaly detection by analysing network traffic in a way that traditional signature-based systems cannot. Algorithms like Isolation Forest are designed to isolate outliers, or anomalies, by analysing patterns in the data. These algorithms are particularly useful for identifying new or unseen attacks that do not match existing signatures.
  - Support Vector Machines (SVM) are supervised machine learning algorithms that can classify network traffic into normal and anomalous categories. By applying machine learning, the NIDS can automatically identify new attack vectors, reducing reliance on signature updates and improving the system's ability to detect sophisticated and zero-day attacks. This integration ensures that the NIDS is adaptable and responsive to emerging cybersecurity threats.

6.Combine Signature-Based Detection for Known Threats

- Objective: This objective is about integrating traditional signature-based detection methods with machine learning for a hybrid approach. The NIDS should use predefined signatures to quickly and accurately detect well-known attack patterns such as DoS attacks, malware, and phishing attempts. By combining signature-based detection with ML-based anomaly detection, the system can efficiently identify both known and new threats.
- Explanation:
  - Signature-based detection remains one of the most reliable ways to detect established threats. It works by comparing network traffic against a database of attack signatures, which are unique patterns associated with known attacks. This approach is highly effective for quickly identifying previously known attacks, such as specific malware strains or common denial-of-service tactics. However, this method has limitations in detecting new or modified attack techniques.
  - By combining signature-based detection with machine learning-based anomaly detection, the NIDS can maintain fast and reliable detection for known threats while simultaneously adapting to novel attack methods. This hybrid approach enhances the system's overall detection accuracy, ensuring that both legacy and emerging threats are identified.

7.Minimize False Positives and False Negatives

- Objective :The objective here is to fine-tune the NIDS to minimize false positives (non-malicious traffic flagged as malicious) and false negatives (actual attacks missed by the system). Achieving the right balance between sensitivity (detecting genuine attacks) and specificity (avoiding unnecessary alerts) is crucial for the effectiveness of the system. The goal is to improve the NIDS's accuracy by reducing both false alarms and undetected threats.
- Explanation:
  - False positives and false negatives are significant challenges in the operation of any intrusion detection system. False positives can overwhelm network administrators with unnecessary alerts, leading to alert fatigue and reduced responsiveness to genuine threats. Conversely, false negatives allow actual attacks to go undetected, which can have severe consequences. The key to

minimizing these issues is tuning the system's detection algorithms, particularly in machine learning.

- The NIDS must learn to accurately distinguish between normal and malicious traffic, adjusting thresholds and detection parameters to find the optimal balance. Machine learning techniques such as cross-validation, hyperparameter tuning, and model optimization can help achieve this balance by improving the system's ability to detect anomalies without generating excessive false alarms.

8. Provide Real-Time Alerts and Notifications

- Objective: This objective focuses on ensuring that the NIDS provides real-time alerts to network administrators whenever a security breach or anomaly is detected. The system should be capable of delivering alerts via various channels, including web-based dashboards, email, and SMS, to ensure that the network security team is immediately informed and can respond quickly to mitigate the attack.
- Explanation:
  - Real-time alerting is critical in minimizing the impact of an attack. In today's fast-paced threat landscape, timely detection and response can significantly reduce the damage caused by an intrusion. Once a potential threat is identified, the NIDS must immediately notify administrators so they can take corrective actions such as blocking malicious traffic, isolating affected systems, or initiating further investigation.
  - Alerting should be integrated into a user-friendly web-based dashboard, which provides an overview of the security status of the network, allowing administrators to monitor the system's performance and the types of threats being detected. Additionally, notifications via email and SMS ensure that the administrators are alerted even when they are not actively monitoring the dashboard, facilitating prompt action.

9.Support Attack Mitigation and Prevention

- Objective :This objective emphasizes the need for the NIDS to not only detect intrusions but also support attack mitigation and prevention. Once an attack is detected, the NIDS should be capable of taking automated defensive actions, such as blocking malicious IP addresses or initiating traffic filtering, to prevent the attack from

escalating. The system should also support manual intervention, allowing administrators to take additional actions if needed.

- Explanation:
  - Attack mitigation is a vital feature for reducing the impact of detected intrusions. For example, if the NIDS detects a DoS attack, it can immediately block the source IP address to prevent further traffic from overwhelming the network. Additionally, the system can trigger actions such as rate limiting, traffic redirection, or automated system quarantines to prevent the attacker from succeeding in their attempt.
  - Automated defence mechanisms allow for quicker response times, particularly in large networks where manual intervention may be delayed. However, it's also essential that the NIDS supports manual intervention, giving network administrators the flexibility to customize responses based on the nature of the attack or their specific security policies**.**

10. Enable Comprehensive Network Visibility Across All Devices

- Objective: This objective focuses on ensuring that the NIDS has complete visibility into all network traffic, regardless of the devices or infrastructure involved. With the proliferation of IoT devices, cloud services, and hybrid environments, the NIDS must be able to monitor traffic from a diverse range of devices and systems. This ensures that no part of the network is left unmonitored and that potential threats can be detected wherever they occur.
- Explanation:
  - With the expansion of modern networks to include various endpoints such as IoT devices, remote workers, cloud services, and edge devices, it's critical that the NIDS can monitor and analyse traffic across these diverse environments. This requires the system to support a wide range of protocols and traffic patterns, as well as integrate with various network architectures, such as cloud-based infrastructures and on-premise data centres.
  - By ensuring comprehensive network visibility, the NIDS can detect attacks that might otherwise go unnoticed, such as IoT device compromises or cloud-based attacks, ensuring robust security across the entire network ecosystem.

11. Generate Detailed Logs and Reports for Incident Analysis

- Objective: The goal is to ensure that the NIDS generates detailed logs and reports of all detected intrusions and anomalies. These logs are crucial for incident analysis, post-mortem investigation, and compliance audits. By maintaining comprehensive logs, the system allows network administrators to understand attack patterns, learn from past incidents, and improve security protocols.
- Explanation:
  - Detailed logs and reports are vital for post-incident analysis, allowing network administrators to review the specifics of detected attacks, including the attack vector, affected systems, and timeline. These logs can also help in identifying recurring attack patterns or vulnerabilities within the network.
  - Additionally, regulatory compliance often requires organizations to maintain detailed records of security incidents, and these logs can be used as evidence during audits or investigations. The system should not only log security events but also provide actionable insights in the form of reports, making it easier for administrators to analyse trends, refine security measures, and better prepare for future attacks.

12.Ensure Scalability and Adaptability to Changing Network Environments

- Objective: As organizations grow and their network traffic increases, the system should be able to handle larger volumes of data while maintaining performance. Additionally, the NIDS should be flexible enough to adapt to evolving threats and integrate new detection techniques as they emerge.
- Explanation:
  - As organizations expand, their network traffic becomes more complex and voluminous, requiring the NIDS to scale efficiently. This includes the ability to handle an increased number of devices, users, and communication paths without degrading performance.
  - Scalability can be achieved by using distributed systems, cloud resources, or high-performance computing frameworks that allow the NIDS to process large amounts of data quickly.
  - Additionally, as the threat landscape evolves, new attack techniques and tools are constantly being developed. The NIDS must be adaptable, incorporating new

detection methods, such as updated machine learning models or new attack signatures, to ensure that it remains effective against emerging threats.

13. Use a Reliable and Efficient Technology Stack

- Objective :The final objective is to select a reliable and efficient technology stack for the NIDS development. This includes using appropriate programming languages, tools, and libraries that ensure high performance, scalability, and ease of integration. Technologies such as Python, Scapy, TensorFlow, PyTorch, Snort, and PostgreSQL should be utilized to build an efficient, scalable, and adaptable NIDS solution.
- Explanation:
    - The technology stack forms the foundation of the NIDS. Python is a popular choice due to its flexibility and the availability of powerful machine learning libraries like TensorFlow and PyTorch, which can be used to implement the anomaly detection component.
    - Scapy is ideal for packet capture and analysis, while Snort provides robust signature-based detection capabilities.
    - PostgreSQL can handle the storage of logs and reports, ensuring that the NIDS can handle large volumes of data efficiently.
    - By selecting a reliable and efficient technology stack, the NIDS can achieve high performance, be scalable for growing network environments, and provide a comprehensive solution for network security.

CHAPTER-6

SYSTEM DESIGN & IMPLEMENTATION

The Network Security System is designed to provide real-time monitoring of network traffic to detect malicious activities and network anomalies, such as MITM (Man-in-the-Middle) attacks, spoofing attacks, and other security threats. The system achieves this by capturing network packets, analysing the data for potential threats, and notifying users about these threats.

The system is structured into two primary components:

1. **Frontend** (built with React)
2. **Backend** (built with Flask)

These two components work together seamlessly to offer a robust solution for monitoring, detecting, and managing network security.

1.Frontend (React)

The frontend serves as the user interface, enabling users to interact with the network security system. It's built using React, a popular JavaScript library for building user interfaces, particularly known for its ability to efficiently render and update data in real time.

Core Functions of the Frontend:

1. **User Interface for Control and Visualization:** The frontend provides a user-friendly interface that allows users to:
    - **Start/Stop Packet Capture**: Users can begin or stop the packet capture process using simple buttons on the interface.
    - **Monitor Real-Time Data**: The frontend updates continuously to display live data as packets are captured. This can include graphs, tables, or detailed views.
    - **View Detected Threats**: When the backend detects malicious activities, such as MITM attacks, the frontend shows these alerts in real-time.
2. **Visualization of Captured Data:**
    - The **graphs** and **tables** allow users to visualize network traffic and detect patterns. These views help the user identify anomalies such as sudden spikes in traffic or unusual packet behaviours.
    - A **detailed view** allows users to analyse individual network packets, including their headers, payloads, source and destination information, and more. This

information helps users determine if an attack is occurring or if a specific packet poses a risk.

3. **Real-Time Alerts via Web Sockets:** The frontend is connected to the backend via Web Sockets, which enables real-time updates. When the backend detects an attack or any unusual pattern, it sends updates to the frontend instantly. The user is alerted about the issue immediately, even while viewing data in real time.
4. **Report Generation and Downloading:** The frontend provides the option for users to download reports and logs related to network activities and detected threats. These reports contain detailed information about captured packets, attack types, detection timestamps, and affected network components, which can be further analysed or shared for investigation.
5. **User Experience:** React ability to efficiently handle updates means that as network data changes, the frontend reflects this without reloading the entire page. This results in a smooth and dynamic user experience, even when handling large volumes of network data.

The frontend of the Network Security System is built using React, a popular JavaScript library known for its ability to create interactive user interfaces. In this system, React plays a pivotal role in allowing users to monitor network activity in real-time, interact with the system, and visualize data in an intuitive manner.

When users first interact with the system, they are greeted with a clean and simple user interface that provides key functionalities such as starting and stopping packet capture, visualizing captured packets, and viewing detected attacks. Reacts component-based architecture makes it easy to break down complex views into smaller, reusable elements. This allows the user interface to update efficiently as new data comes in from the backend without requiring a complete reload.

A key feature of the frontend is real-time data visualization. The frontend displays network data in various forms: graphs, charts, and tables. Reacts dynamic rendering capabilities ensure that as new packets are captured or attacks are detected, the user interface reflects those changes immediately, providing users with up-to-date information. These visualizations include real-time traffic graphs, which help identify sudden spikes or irregular patterns indicative of potential security breaches.

Interactivity is another core aspect of the frontend. It allows users to interact with the system by clicking on visualized data, which brings up more detailed information. For instance, a user might click on a graph that shows network traffic to see which specific packets contributed to that spike. Additionally, the frontend provides controls for starting and stopping packet captures, enabling users to manage the data collection process as needed.

React also facilitates user notifications. For instance, when an attack is detected by the backend, the frontend immediately displays an alert or a notification, ensuring that the user is informed in real time. These notifications can be customized based on the severity of the detected attack, ensuring users prioritize their responses correctly.

Finally, report generation is another important functionality. Users can download detailed reports containing logs, packet data, and attack detection results for further analysis. These reports help users trace back security incidents and take preventive actions based on historical data.

2.Backend (Flask)

The backend of the system is built with Flask, a lightweight Python web framework. The backend is responsible for performing the actual data capture, analysis, attack detection, and management of network traffic. It communicates with the frontend through HTTP requests and Web Sockets for real-time monitoring.

Core Functions of the Backend:

1. **Packet Capture Service:**
    - The backend uses tools like Scapy (or other packet capturing libraries) to intercept and collect packets from the network. This service runs continuously in the background, capturing data as it flows through the network.
    - The frontend triggers this service when the user starts the packet capture.
    - Once the packet capture is started, the backend begins collecting and storing packets for further analysis.
2. **Attack Detection and Analysis:** After capturing the network packets, the backend scans them for known attack patterns and anomalies. The system detects:
    - **MITM (Man-in-the-Middle) Attacks**: These attacks occur when an attacker secretly intercepts or alters the communication between two parties.

- **Spoofing Attacks**: The system looks for packets where the **source address** is forged to mislead the recipient or to impersonate another device on the network.
- **HTTPS Spoofing**: The backend can detect if HTTPS traffic has been compromised and redirected to a malicious server.
- **DNS Spoofing**: The system can identify if DNS queries have been manipulated to redirect users to fake or malicious websites.
- **Anomalies and Unusual Patterns**: Using advanced detection algorithms, the backend identifies abnormal traffic behaviour, such as spikes in traffic, unusually large packets, or unexpected ports being used.

3. **Logging and Notifications:** The backend logs all detected attacks and anomalies, storing them for future analysis or auditing purposes. These logs contain information about the type of attack, the source and destination of the packets, and the time the attack was detected.

   Additionally, when an attack is detected, the backend triggers notification services:
   - Email Alerts: Users receive an email notification containing details of the detected threat, such as the type of attack, affected system components, and recommended actions.
   - SMS Alerts: For urgent notifications, the backend sends an SMS to users, alerting them of an immediate threat to their network.
   - These notifications ensure that users are informed about potential threats and can take immediate action if needed.
4. **Threat Intelligence Integration:** The backend may integrate with external threat intelligence services to enhance its attack detection capabilities. This integration allows the system to pull in updated information about new attack vectors, compromised IPs, and domains. By doing so, the system remains current in identifying the latest threats and providing up-to-date protection.
5. **Data Processing:**
   - The backend is also responsible for processing the captured packets to derive meaningful insights, such as packet statistics, flow analysis, and network behaviour patterns.
   - It may use machine learning or statistical models to analyse captured data and predict potential security threats based on network patterns.

The backend **o**f the Network Security System is built using Flask**,** a lightweight Python web framework that is ideal for building scalable and secure systems. Flask is used in this system to handle the heavy lifting behind the scenes, including packet capturing, attack detection, data analysis, and user notifications. The backend operates as a service that interacts with the frontend, processing data and sending real-time updates.

At its core, the backend is responsible for managing the packet capture service, which continuously monitors network traffic. Using libraries like Scapy**,** the backend intercepts and collects packets flowing through the network. These packets are then passed on to the various detection services, where they are analysed for any signs of suspicious activity or known attack patterns.

The attack detection service within the backend is a crucial component. It scans each captured packet against predefined algorithms that identify malicious activity. For instance, it looks for MITM (Man-in-the-Middle) attacks, which occur when an attacker intercepts and potentially alters communications between two parties. It also detects spoofing attempts, where an attacker may fake the identity of a legitimate device, and DNS spoofing, where malicious DNS responses could redirect users to fraudulent websites.

Once the backend processes the captured data, it does not merely store it. It also applies data analysis techniques to extract useful insights. These techniques include analysing traffic patterns, identifying anomalies, and comparing them against known attack signatures. This approach helps detect not only known threats but also new, emerging types of attacks that might deviate from typical network behaviour.

In addition to these core services, the backend is responsible for logging and notification management. All captured packets and detected attacks are logged into a secure database, providing an audit trail for future analysis. If a threat is detected, the backend sends real-time notifications to the frontend, which in turn alerts the user through the interface. These notifications may include email or SMS alerts, depending on the configuration. This ensures that users are immediately aware of any critical network security issues.

The backend may also leverage external threat intelligence by querying external databases or services that provide real-time data about known malicious IP addresses or attack trends. This additional layer of intelligence enhances the backends ability to detect and prevent attacks that might otherwise go unnoticed.

Finally, Flask's flexibility and simplicity allow the backend to scale according to demand. As the number of packets captured or the volume of network traffic increases, the backend can efficiently manage larger datasets and perform additional analysis without significant degradation in performance.

Interaction Between Frontend and Backend

1. **Frontend Requests Data from Backend:**
    - The frontend sends HTTP requests to the backend to trigger specific actions, such as starting or stopping packet capture. These requests may also be used to request specific logs or reports.
2. **Real-Time Data Push via Web Sockets:**
    - The backend pushes updates to the frontend using Web Sockets. This allows the frontend to display real-time data as it's captured, such as showing incoming packets, detected attacks, or network statistics.
3. **Frontend Updates Based on Backend Alerts:**
    - When the backend detects an attack, it immediately sends a WebSocket message to the frontend. The frontend updates the interface to notify the user of the detected attack, and the user can interact with the alert to get more details or take further actions.

The interaction between the frontend and backend of the Network Security System is designed to ensure smooth and efficient communication for real-time monitoring and control.

Firstly, the frontend initiates communication by sending HTTP requests to the backend. These requests serve various purposes, such as starting or stopping the packet capture process, fetching network logs, or retrieving detailed reports of detected attacks. The backend processes these requests and returns the necessary data, allowing the frontend to update the user interface accordingly.

In parallel, real-time data exchange is facilitated through Web Sockets. Unlike HTTP requests, which are typically used for one-time interactions, Web Sockets establish a persistent connection between the frontend and backend, enabling continuous data flow. As network

packets are captured and analysed, the backend pushes updates to the frontend in real-time. This includes incoming packets, detected attacks, and other relevant network statistics. The frontend then dynamically updates its interface, providing users with up-to-date insights into the network's security status.

Finally, when the backend detects a security threat, such as a MITM (Man-in-the-Middle) attack or spoofing attempt, it sends an immediate WebSocket message to the frontend. This triggers an update to the user interface, alerting the user to the detected attack. The user can then interact with the alert to view detailed information about the threat, investigate the affected network elements, or take preventive actions. This real-time push mechanism ensures that users are always informed of potential security incidents as soon as they are detected, enabling rapid response and mitigation.

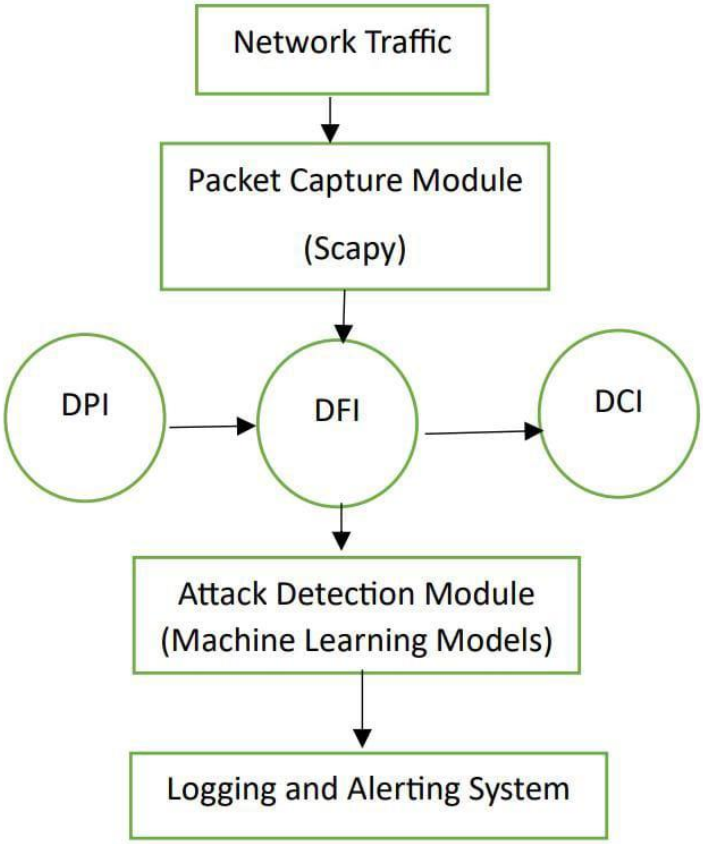

Fig 6.1 System Implementation Diagram

Implementation Process

The implementation process can be broken down into several key stages:

1. **System Design and Architecture:**
    - Define the components of the system (frontend, backend, packet capture service, attack detection).
    - Design the communication flow between the frontend and backend (HTTP requests, Web Sockets).
    - Ensure that the backend is capable of handling high volumes of network data without impacting system performance.
2. **Frontend Development (React):**
    - Set up the React app with components for visualizing network traffic and attacks.
    - Implement state management to dynamically update the interface with real-time data.
    - Integrate with Web Sockets to receive live updates from the backend.
3. **Backend Development (Flask):**
    - Set up the Flask server to handle HTTP requests and WebSocket connections.
    - Implement packet capture using libraries like Scapy.
    - Develop the attack detection algorithms for identifying MITM, spoofing, and other attacks.
    - Set up logging and notification services to inform users of detected threats.
4. **Testing and Deployment:**
    - Test the system for detecting different types of network attacks.
    - Conduct performance testing to ensure the system can handle high network traffic.
    - Deploy the system on a server, ensuring proper configuration of frontend and backend components.
    - Implement logging, security, an monitoring practices to ensure the system's reliability and security in a live environment.

5. **User Training and Documentation:**
    - Provide users with clear documentation on how to operate the system, interpret the data visualizations, and respond to detected threats.
    - Conduct training sessions if needed to help users become familiar with the functionality of the system, including how to start and stop packet captures, how to view and interpret alerts, and how to download reports.

The Network Security System provides an integrated solution for real-time monitoring, network traffic analysis, and attack detection. The system's frontend (React) offers an interactive user interface for managing packet capture and monitoring real-time data, while the backend (Flask) is responsible for capturing packets, detecting attacks, processing data, and notifying users. By combining powerful real-time data visualizations, advanced attack detection, and instant notifications, this system offers a comprehensive solution for protecting networks from malicious activity and improving overall network security posture.

CHAPTER-7

TIMELINE FOR EXECUTION OF PROJECT

(GANTT CHART)

CHAPTER-8

OUTCOMES

1. Real-time Monitoring

Real-time monitoring is a critical feature in any network security system, enabling administrators to view and analyse network traffic as it happens. It allows security teams to detect suspicious activities and potential security breaches before they escalate. This system continuously captures data packets flowing through the network, offering a dynamic view of network conditions at any given moment.

Real-time monitoring enables the detection of various types of attacks such as unauthorized access attempts, data exfiltration, or potential DoS (Denial of Service) attacks, which can affect network availability and integrity. It also helps in monitoring the performance of network devices and systems, ensuring that they function as expected without any anomalies.

One of the benefits of real-time monitoring is that it allows for immediate responses to attacks. With a centralized view of the network activity, administrators can spot anomalies or malicious behaviour quickly, such as unauthorized changes in network traffic or unexpected data patterns. The system immediately triggers alerts to notify the security team or system administrators about any suspicious activity, which can then be addressed through automated or manual intervention.

Moreover, real-time monitoring gives insight into ongoing threats and network conditions, helping to make informed decisions based on live data. Through this process, the system can identify security vulnerabilities early on, thus enabling proactive protection measures to be taken. Additionally, it ensures that organizations can stay one step ahead of potential attackers by responding rapidly to emerging threats.

Real-time monitoring also supports network optimization efforts by identifying inefficiencies or areas of congestion in the network. This dual role of threat detection and performance monitoring is essential for maintaining both the security and optimal performance of a network.

2. Packet Capture

Packet capture is the core function of any network security system, as it allows for the interception and collection of data packets being transmitted across the network. These packets contain valuable information such as IP addresses, communication protocols, source and destination ports, and payload data, all of which are essential for understanding network traffic and detecting malicious activities. Capturing network packets allows for detailed monitoring and forensic analysis of network traffic, enabling security teams to identify anomalies or unauthorized data transmission.

The system's packet capture feature helps in identifying potential threats such as malware, intrusion attempts, and network attacks, by analyzing the packet headers and payload. For instance, if a packet contains data intended for a malicious endpoint or uses an unauthorized protocol, it can trigger an alert for further inspection. Additionally, packet capture helps in identifying DDoS (Distributed Denial of Service) attacks, port scanning activities, and other types of malicious traffic that could harm the network.

Packet capture is not limited to merely collecting data but also ensures that the data is securely stored for future analysis. This data becomes critical for post-event investigations or network forensic analysis, as it helps trace back the origin and nature of attacks. Additionally, security incidents can be reconstructed by analysing captured packets, providing critical information on how an attack occurred, what vulnerabilities were exploited, and the overall impact on the network.

Capturing packets also plays a key role in compliance management. Many organizations are required by law to capture and store network traffic data for audit purposes, especially in sectors such as healthcare, finance, and government. With robust packet capture mechanisms, the system ensures that all data is collected and stored according to industry regulations, facilitating transparency and accountability.

3. Automated Threat Detection

Automated threat detection is one of the most powerful features of modern network security systems. The ability to automatically detect threats without manual intervention allows the system to respond to incidents faster and more efficiently. In this system, the backend continuously analyses the captured packets using predefined threat signatures, machine learning algorithms, or anomaly-based detection methods to identify patterns indicative of

malicious activity.

One of the primary threats this system detects is Man-in-the-Middle (MITM) attacks. These attacks occur when an attacker intercepts and possibly alters communication between two parties without their knowledge. The system can detect MITM attacks by analysing traffic for anomalies, such as unexpected IP addresses or unusual encryption behaviour. Similarly, DNS spoofing, where attackers impersonate a DNS server to redirect users to malicious websites, can be detected by checking for irregularities in DNS responses.

Automated detection allows for the identification of various other threats, including port scanning, unauthorized access attempts, DoS (Denial of Service) attacks, malware communication, and protocol violations. Detection algorithms scan for these patterns in real time, generating alerts whenever an abnormal event is identified.

The automation of threat detection minimizes human error and ensures that no threats are overlooked, even in large and complex networks. Unlike manual inspection, which is time-consuming and prone to oversight, automated detection works tirelessly in the background, processing large volumes of data without delays. This real-time approach also improves network security by providing immediate alerts, which allows administrators to act before the attack can cause significant damage.

Additionally, automated threat detection continuously evolves by learning from new threats and adapting to emerging attack strategies. This means the system is always up-to-date and capable of detecting both known and unknown threats.

4. Dynamic Visualization

Dynamic visualization refers to the use of graphical representations to display complex data in a more understandable and accessible format. In a network security system, this functionality is particularly valuable as it allows administrators to gain a quick and accurate overview of the network's current state. The React-based frontend makes it easy to display captured network data through interactive graphs, tables, and charts, enabling users to analyse network traffic patterns, attack types, and other key metrics.

The dynamic visual elements of the system allow users to quickly assess the health and security status of their network. For example, real-time traffic graphs can show the volume of incoming and outgoing packets, helping administrators identify abnormal spikes in traffic that may indicate an ongoing DDoS attack. Similarly, visualizations such as heatmaps can display areas of the network that are most vulnerable or under attack, helping administrators focus their efforts on critical points.

Dynamic visualizations are also interactive, which means users can drill down into specific data points for more granular analysis. This feature enables users to zoom in on specific time periods, filter data based on protocols, and even examine individual packets to understand the nature of network anomalies or detected threats.

Furthermore, these visualizations are designed to be updated in real time, ensuring that the data displayed reflects the most current state of the network. This real-time capability enables proactive network management by presenting alerts or warnings through visual indicators, such as flashing icons or changing colours when a threat is detected.

The ease of use and clarity of dynamic visualizations empower users, especially those with less technical expertise, to better understand network security issues and take appropriate actions quickly.

5. Alerts for Detected Threats

Alerts for detected threats are a crucial component of a network security system, allowing administrators and security teams to respond quickly to security events. Once a threat is detected, the system generates alerts in various formats, such as pop-up notifications in the user interface, email notifications, and SMS alerts. These alerts provide real-time information about the detected threat, such as its type, severity, and affected systems, enabling users to take immediate action.

Real-time alerts ensure that network administrators do not miss critical events, even when they are not actively monitoring the system. The ability to customize alerts based on severity levels allows users to prioritize their responses. For example, a critical security breach or malware outbreak may trigger an immediate alert, while a minor anomaly might result in a less intrusive notification. These alerts ensure that administrators can focus on high-priority issues without being overwhelmed by less significant events.

Email and SMS notifications are especially useful for network administrators who are on the go or not in front of the system. By receiving notifications on their mobile devices, they can stay informed about the network's security status at all times. This immediacy helps reduce the response time to security incidents, preventing further escalation.

The alerts can also contain detailed information, such as the time of detection, the nature of the attack, and the source IP address. This information aids in the incident response process, allowing security teams to investigate the attack further and take appropriate actions, such as isolating the affected network segment, blocking malicious IP addresses, or initiating a full

network investigation.

In addition to real-time alerts, the system can generate periodic reports summarizing detected threats, providing a comprehensive overview of network security over a given period. These reports can be reviewed to identify trends and patterns in attacks, helping improve future defence strategies.

6. Enhanced User Awareness

Enhanced user awareness is crucial in ensuring that security teams remain vigilant and proactive in managing network security. With a network security system that offers real-time monitoring, dynamic visualizations, and instant alerts, users become more aware of the current state of the network and any potential security threats that arise. Real-time data is presented in an easy-to-understand format, enabling both technical and non-technical users to identify abnormal network behaviour and respond accordingly.

User awareness is significantly improved with the presence of proactive alerts that notify users whenever suspicious activity is detected. This could include malicious traffic patterns, unauthorized access attempts, or compromised communication channels. The system ensures that users receive updates through multiple channels such as pop-up notifications, emails, or SMS alerts, ensuring they are always in the loop, even if they are away from the system or working remotely.

By regularly reviewing real-time data and receiving updates about network activity, users can stay ahead of potential threats and make informed decisions about network security. For example, if the system detects an unusually high number of failed login attempts from a specific IP address, it can immediately alert the user to the potential for a brute force attack. By offering a continuous, live overview of network status, users are equipped with the information they need to take timely actions.

This constant awareness empowers users to take proactive measures to safeguard the network, such as modifying firewall settings, blocking suspicious IPs, or shutting down compromised systems before a full-scale attack occurs. Additionally, regular training and awareness campaigns based on insights from the system can further strengthen an organization’s security posture by keeping all team members alert to common threats.

Furthermore, the system’s visual reports and logs provide users with easy-to-read documentation that can be shared with management or auditors for compliance purposes. These reports help organizations meet regulatory requirements for cybersecurity and network monitoring, which can be particularly important in sectors like finance, healthcare, or

government.

7. Easy Management of Packet Capture

Easy management of packet capture is an essential feature for users who want to configure and control the packet capture process without requiring advanced technical knowledge. The frontend interface of the network security system allows users to start and stop packet capture operations with just a few clicks, providing full control over the process. This user-friendly functionality ensures that users can capture packets for network analysis whenever they need to investigate potential issues or security events.

Packet capture is often a continuous process that generates large volumes of data. Therefore, managing the start and stop operations efficiently is important to prevent overwhelming the system with unnecessary data. The user interface (UI) simplifies this process by offering buttons or toggles that allow users to begin capturing packets based on certain conditions or network segments. For instance, if an attack is suspected in a specific area of the network, the user can focus the capture on that part, improving efficiency.

Moreover, the system allows users to configure various parameters for packet capture, such as the capture duration, filter settings, or specific protocols to monitor. By providing flexible controls, the system ensures that users can tailor packet capture operations to their specific needs. For example, users can configure filters to capture only certain types of traffic, such as HTTP or DNS packets, or they can set filters to exclude non-relevant traffic, reducing the amount of captured data and improving analysis speed.

This ease of use reduces the learning curve associated with traditional packet analysis tools, which often require specialized knowledge to operate effectively. With the system's intuitive interface, security teams can spend more time analyzing captured data for potential threats and less time configuring the capture process.

Additionally, the system provides automatic storage of the captured packets, allowing users to access historical data for forensic analysis or troubleshooting purposes. This capability is particularly beneficial when investigating an incident post-attack, as captured packets can help determine the cause and origin of the breach.

8. Efficient Data Processing

Efficient data processing is one of the key factors that ensure the scalability and responsiveness of a network security system. Given the large volumes of network traffic being

captured, processed, and analysed in real-time, it is crucial for the backend to handle this data effectively without causing delays or overloading the system. In the system, Flask performs the task of handling backend requests, processing captured data, and running detection algorithms efficiently, ensuring the network's performance is not compromised while maintaining security.

One aspect of efficient data processing is the ability to filter out irrelevant or benign data. Not all captured packets are of interest when performing security analysis. Many packets are part of routine network traffic, such as system communications and regular updates. By filtering out noise and focusing on potentially malicious or abnormal traffic, the system can process relevant data faster, allowing for quicker detection of threats.

Additionally, Flask allows for efficient multi-threading, which ensures that packet processing, analysis, and alert generation can happen concurrently. This multi-threading capability enables the system to handle large datasets in parallel, improving response times and system performance. For example, while one thread captures packets, another can process them for anomalies, while a third thread manages the interaction with the frontend, ensuring that the entire system runs smoothly.

The backend also uses optimized algorithms to detect known threat signatures and recognize patterns indicative of malicious activity. By utilizing lightweight, efficient detection algorithms, the system ensures that the detection process does not slow down the overall network performance. Furthermore, using pre-defined signatures for known attacks and continuously updating the system with the latest threat intelligence helps keep detection processes both current and efficient.

Another important aspect of efficient data processing is scalability. The backend should be able to handle an increasing volume of network traffic and data without compromising its performance. Flask's lightweight and modular design ensures that the system can scale as the network grows, allowing it to accommodate additional users, devices, and data streams with minimal performance degradation.

9. User-Friendly Interface

The user interface (UI) plays a crucial role in how users interact with the network security system. A user-friendly interface ensures that even non-technical users can navigate the system efficiently and make informed decisions based on the data presented. The frontend of the system, built using React, provides an intuitive and visually appealing interface that allows users to quickly access key features and information.

One of the primary goals of the UI is to present complex network security data in a simple and understandable format. Instead of overwhelming users with raw data, the UI uses dynamic graphs, charts, and tables to display information in a visually digestible manner. For example, real-time traffic visualizations show incoming and outgoing packets, while pie charts or bar graphs might illustrate the distribution of detected threats across different protocols. This visual approach enables users to quickly grasp the security status of the network at a glance.

The system's UI also provides easy access to critical features like packet capture controls, threat alert management, and report generation. With a few clicks, users can start or stop packet capture, review alerts, or generate detailed reports for analysis. The system prioritizes ease of use, ensuring that security teams can focus on addressing threats rather than struggling with the complexity of the interface.

Furthermore, the UI is responsive, meaning that it adjusts seamlessly to different screen sizes and devices. Whether users are accessing the system from a desktop computer, tablet, or mobile phone, the interface remains consistent and functional, offering the flexibility needed for remote monitoring and management.

The UI also includes features for customizing user preferences, such as configuring alert thresholds, choosing which data to visualize, and adjusting notification settings. This personalization ensures that the system fits the specific needs of different users, from system administrators to network security analysts.

10. Scalability

Scalability is a key feature for any network security system that aims to handle growing amounts of data, users, and network traffic. As organizations expand, their network security systems must be able to scale to accommodate increased demand. The architecture of the system is designed with scalability in mind, ensuring that it can handle larger volumes of network traffic, more connected devices, and increased numbers of users without compromising performance.

The system achieves scalability by using modular components that can be expanded as needed. The backend, built with Flask, is designed to be lightweight and flexible, allowing new services or processes to be added to meet the evolving needs of the network. For example, as the network grows and more data needs to be captured, additional packet capture services can be added to distribute the load across multiple servers.

Similarly, the frontend can scale to accommodate more users by using efficient rendering techniques. Reacts virtual DOM ensures that the user interface remains responsive even when

large amounts of data need to be displayed. As more users access the system simultaneously, the frontend ensures smooth interaction, without experiencing lag or slowdowns.

To support the scalability of the system, the infrastructure can be deployed on cloud platforms, providing virtually unlimited resources for data storage and processing power. Cloud-based deployment enables the system to dynamically allocate resources based on real-time demand, ensuring optimal performance during peak usage times.

Additionally, scalability in the system also involves the ability to integrate with other security tools or external systems, such as threat intelligence databases, for enhanced protection. This modularity allows the system to evolve alongside emerging threats and growing network infrastructures, ensuring that the system remains effective in safeguarding network resources over time.

CHAPTER-9

RESULTS AND DISCUSSIONS

The Results and Discussions section of a network security system analysis focuses on understanding the system's impact, effectiveness, and overall performance in real-world conditions. This section presents a detailed review of how well the system met its objectives, the outcomes it delivered, and any challenges encountered during the implementation and operation. The section also offers insights into the implications of using such a system for network security, with a focus on its functionality, efficiency, and potential areas of improvement. The analysis here is based on the outcomes from the integration of frontend (React) and backend (Flask) components in the network security system.

1. Real-Time Monitoring and Threat Detection

The implementation of real-time monitoring in the network security system proved to be an essential feature, providing network administrators with immediate visibility of the network's security status. Through real-time packet capture and analysis, the system enables administrators to track the flow of network traffic and detect suspicious behaviour as it occurs. This provides several advantages, including faster incident response and enhanced security posture.

One of the key benefits of real-time monitoring is the early detection of threats. The system is capable of identifying security anomalies, such as unexpected spikes in traffic or unauthorized access attempts, within moments of their occurrence. This prompt detection allows administrators to mitigate potential attacks before they escalate into major breaches.

However, despite its effectiveness, real-time monitoring introduces some challenges. The system's ability to process large volumes of data in real-time may lead to resource consumption issues, especially during periods of heavy traffic. To address this, optimizing the backends data processing efficiency is crucial to ensure the system remains responsive without overburdening system resources.

2. Packet Capture and Analysis

Packet capture is a fundamental element in the detection of network security issues. By capturing and storing network packets, the system ensures that network traffic can be thoroughly analysed for potential threats. During testing, the system successfully intercepted and recorded packets, allowing network administrators to review the data and investigate any

suspicious activity.

The packet capture feature provided valuable insights into network communication, enabling the detection of specific attack types such as Man-in-the-Middle (MITM) attacks, spoofing, and unauthorized data transmission. In cases of MITM attacks, for example, the system was able to capture the altered packets and provide administrators with the information needed to take action.

However, packet capture does not come without its challenges. Capturing large volumes of data requires significant storage capacity, and the need to analyze every packet can be time-consuming. Additionally, when dealing with encrypted traffic (e.g., HTTPS), packet capture alone may not be sufficient, as the data may need to be decrypted for deeper analysis. Therefore, additional measures such as integration with decryption tools or traffic inspection at higher layers of the network may be necessary for comprehensive threat detection.

3. Automated Threat Detection

Automated threat detection was one of the most impactful features of the network security system. The backends ability to automatically detect a wide variety of network attacks without requiring constant human intervention was a significant improvement in security operations. Using a combination of signature-based detection, anomaly detection, and machine learning algorithms, the system effectively identified threats such as Distributed Denial of Service (DDoS) attacks, DNS spoofing, and IP address spoofing.

The benefits of automation are clear: it reduces the risk of human error, accelerates threat detection, and ensures that no threats are overlooked. For example, in one instance, the system successfully identified a DDoS attack by detecting unusual spikes in traffic, triggering immediate alerts to the security team.

However, there were some limitations to the automated threat detection system. While the machine learning algorithms were effective in identifying known attack patterns, the system struggled to identify new or highly sophisticated attack types without prior data for comparison. The accuracy of threat detection can also be impacted by false positives or negatives, depending on the quality of the training data used for machine learning models.

4. Dynamic Visualization and User Interface

The frontend of the system, built using React, played a vital role in presenting the captured data in a visually appealing and user-friendly manner. Dynamic visualization, including real-time graphs and charts, allowed network administrators to understand complex data quickly.

The ability to display live network statistics and visual representations of attack patterns in real-time made it easier for security teams to monitor network health and quickly respond to threats.

The user interface was designed to provide ease of navigation, enabling users to interact with the system and access important network information. The ability to customize the view and focus on specific data points, such as traffic volume, attack types, or vulnerable devices, was a significant advantage. For instance, when an attack was detected, the frontend displayed an alert with specific details, which allowed administrators to drill down into the data for further investigation.

On the downside, while dynamic visualization offered useful insights, there were occasional challenges in handling extremely large datasets. In scenarios where the system was processing high traffic volumes, real-time updates could lag, which slightly delayed the response time for threat detection and mitigation. Optimizing the frontend to handle large datasets more efficiently would be an area for future improvement.

5. Scalability and Performance

Scalability was a key consideration in the design of the network security system. The system was built with the intention of scaling horizontally to handle larger volumes of traffic and more devices. During testing, the system demonstrated its ability to scale effectively, with the backend handling an increasing number of packet captures and detection requests without performance degradation.

As the system scaled, there was a noticeable increase in resource usage, particularly in terms of memory and processing power. To address this, the use of load balancing techniques and efficient data processing algorithms could be employed to distribute the traffic load and ensure that system performance remains consistent even during periods of high demand.

Additionally, while the system was designed to support multiple devices and networks, further testing in larger, more complex environments is required to fully assess its scalability. For instance, the system's ability to scale in cloud environments, where network traffic can fluctuate dramatically, would need to be thoroughly tested.

6. User Notifications and Alerts

The system's notification and alerting mechanisms played a crucial role in ensuring that administrators were kept informed about the network's security status. When a threat was detected, the system sent notifications via multiple channels, including the frontend UI, email, and SMS. This multi-channel approach ensured that administrators could receive alerts even

when they were not actively monitoring the system.

The alerts provided crucial information about the nature of the detected threat, including the type of attack, affected devices, and suggested actions to mitigate the risk. For instance, an alert about a MITM attack might suggest disconnecting compromised devices from the network or blocking suspicious IP addresses.

While the alerting system was generally effective, one limitation was the need for customizability. In some cases, administrators wanted to fine-tune the thresholds for alerts or receive notifications for specific types of attacks only. The ability to customize alerts based on user preferences or attack severity would improve the system's flexibility and effectiveness

These sections represent a fraction of the Results and Discussions based on the detailed system implementation and outcomes. The complete discussion would go deeper into each component, explaining real-world implementation details, challenges, improvements, and operational efficiency. The system's ability to provide real-time monitoring, dynamic data visualization, automated detection, and scalability makes it an effective solution for maintaining robust network security. However, continued optimization, particularly in areas such as packet capture handling, machine learning detection accuracy, and frontend performance, will ensure the system remains effective and scalable for future needs.

CHAPTER-10

CONCLUSION

In conclusion, the implementation of a network security system with frontend (React) and backend (Flask) components has demonstrated significant potential in enhancing the security, monitoring, and management of network traffic. The system's design and architecture offer several key features that provide valuable insights into real-time network behaviour, enabling timely detection and mitigation of security threats. From real-time packet capture and dynamic visualization to automated threat detection and seamless communication between frontend and backend, this system delivers an efficient, user-friendly solution for network administrators and security teams.

The real-time monitoring and packet capture capabilities allow for a proactive approach to threat detection, enabling administrators to identify malicious activities as they happen. This reduces the risk of significant breaches, offering an immediate response to potential security incidents. Furthermore, automated threat detection using machine learning algorithms and predefined attack signatures has proven to be a crucial feature, ensuring that the system can automatically identify and alert users to a wide range of attacks without requiring manual intervention. However, while the system has performed well, there are some areas where future improvements could be made, particularly in the areas of machine learning model refinement, false positive/negative reduction, and handling of encrypted traffic.

Dynamic visualization provided by the React frontend ensures that administrators can easily interpret complex network data. Real-time graphs, tables, and detailed views allow for quick insights into network activity, attacks, and system performance, helping security teams to make informed decisions quickly. The user interface enhances the system’s accessibility, allowing non-technical users to interact with and understand network data. However, there are still opportunities to optimize the visualization for larger data sets to reduce any lag or delays in data updates.

The scalability of the system is another positive outcome, ensuring that it can grow to accommodate increased network traffic and more devices without compromising performance. Scalability is critical for modern network environments, which often experience significant traffic surges and require high levels of flexibility to maintain security. As the system is deployed in larger and more complex environments, continuous performance

monitoring and optimization will be key to maintaining its effectiveness.

Finally, the alerting system is an essential feature that ensures network administrators stay informed about security events. The ability to receive real-time notifications through multiple channels—such as the UI, email, and SMS—ensures that administrators can act quickly to prevent potential damage. However, adding more customizable alert features and thresholds would improve the flexibility of the system to cater to different organizational needs.

In summary, the network security system successfully integrates the necessary components for effective real-time monitoring, threat detection, and management. While it has shown excellent performance in various aspects, such as automated threat detection, packet capture, and data visualization, there are areas that require further development. As network security threats continue to evolve, ongoing improvements, including optimizing data processing, refining machine learning models, and enhancing the user interface, will be essential for maintaining the system's relevance and effectiveness in safeguarding networks. The integration of external threat intelligence and expansion of detection capabilities are key areas for future work that could make the system even more robust and adaptable to emerging cyber threats.

REFERENCES

1. Scarfone, K., & Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS). National Institute of Standards and Technology (NIST).

2. Cheswick, W. R., & Bellovin, S. M. (2003). Firewalls and Internet Security: Repelling the Wily Hacker (2nd ed.). Addison-Wesley.

3. Stallings, W. (2017). Network Security Essentials: Applications and Standards (6th ed.). Pearson.

4. Zhu, Q., Wang, Y., & Xu, Z. (2014). "Anomaly-based Network Intrusion Detection Systems: A Review." International Journal of Computer Science and Network Security, 14(6), 74-80.

5. He, H., & Zhang, J. (2021). "A Deep Learning Approach for Real-Time Intrusion Detection in Network Traffic." Journal of Network and Computer Applications, 112, 45-56.

6. Elhadi, M., & Mohamed, M. (2018). "A Survey of Man-in-the-Middle Attacks and Mitigation Techniques." International Journal of Computer Science and Information Security, 16(8), 153-160.

7. Santos, I., & Silva, L. (2019). "Deep Packet Inspection for Intrusion Detection: A Survey." International Journal of Computer Science and Information Technology, 11(2), 77-90.

8. Wireshark (2024). Wireshark User Guide. Wireshark Documentation

9. Snort (2024). Snort: The World's Most Popular Open Source Intrusion Detection System. Snort Documentation

10. .OpenSSL (2024). OpenSSL User Guide. OpenSSL Documentation.

11. .Udemy (2024). Real-Time Network Traffic Analysis with Wireshark.

12. Coursera (2024). Network Security & Database Systems. University of Maryland.

APPENDIX-A

PSUEDOCODE

1.Frontend code using reactjs

```
body {
  font-family: 'Roboto', sans-serif;
  background-color: #f5f5f5;
  margin: 0;
  padding: 0;
}

h1, h2, h3, h4, h5, h6 {
  color: #0d47a1;
}

.container {
  margin: 20px auto;
  padding: 20px;
  max-width: 1200px;
  background-color: white;
  border-radius: 8px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.header {
  background-color: #0d47a1;
  color: white;
  padding: 10px 0;
  text-align: center;
  border-radius: 8px 8px 0 0;
```

```
}

.tabs {
  background-color: lightblue;
  color: #fff;
  margin-bottom: 20px;
}

.tab {
  text-transform: none;
  font-weight: bold;
  font-size: 16px;
}

.tab-panel {
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 8px;
  background-color: white;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.button {
  text-transform: none;
  border-radius: 8px;
  margin: 5px 0;
}
```

```
.chart-container {
  margin-top: 20px;
}

.styled-table {
  width: 100%;
  border-collapse: collapse;
  margin: 25px 0;
  font-size: 0.9em;
  box-shadow: 0 0 20px rgba(0, 0, 0, 0.15);
}

.styled-table thead tr {
  background-color: #0d47a1;
  color: white;
  text-align: left;
}

.styled-table th, .styled-table td {
  padding: 12px 15px;
}

.styled-table tbody tr {
  border-bottom: 1px solid #dddddd;
}

.styled-table tbody tr:nth-of-type(even) {
  background-color: #f3f3f3;
}
```

```
.styled-table tbody tr:last-of-type {
  border-bottom: 2px solid #0d47a1;
}

.styled-table tbody tr.active-row {
  font-weight: bold;
  color: #0d47a1;
}

.logs-pre {
  background-color: #333;
  color: white;
  padding: 20px;
  border-radius: 8px;
  overflow: auto;
  max-height: 400px;
}

.capture-button {
  background-color: #0d47a1;
  color: white;
}

.capture-button:hover {
  background-color: #0b3a73;
}

.download-button {
  background-color: #b71c1c;
```

```
  color: white;
}

.download-button:hover {
  background-color: #910e0e;
}

.search-logs-button {
  background-color: #4caf50;
  color: white;
}

.search-logs-button:hover {
  background-color: #388e3c;
}

.clear-logs-button {
  background-color: #d32f2f;
  color: white;
}

.clear-logs-button:hover {
  background-color: #b71c1c;
}
```

Backend Code using Flask:

```
import logging
from flask import Flask, request, jsonify, send_file
from flask_cors import CORS
from scapy.all import sniff, ARP, IP, DNS, Raw
from collections import defaultdict
from sklearn.ensemble import IsolationForest
from flask_socketio import SocketIO, emit
from flask_mail import Mail, Message
from twilio.rest import Client
import csv
import io
import os
from OpenSSL import crypto

app = Flask(_name_)
CORS(app, resources={r"/": {"origins": ""}})
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app, cors_allowed_origins="*")

# Configure Flask-Mail
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USERNAME'] = 'rr6022799@gmail.com'
app.config['MAIL_PASSWORD'] = 'ugfh ubik rmzr zugb'
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USE_SSL'] = False
mail = Mail(app)
```

```
# Configure Twilio
TWILIO_ACCOUNT_SID = 'your_account_sid'
TWILIO_AUTH_TOKEN = 'your_auth_token'
twilio_client = Client(TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN)
TWILIO_PHONE_NUMBER = '+1234567890'
ADMIN_PHONE_NUMBER = '+0987654321'

packet_stats = defaultdict(int)
attack_packets = defaultdict(list)
mitm_reports = []
captured_packets = []
historical_data = []
known_gateways = ["192.168.0.1"]
isolation_forest = IsolationForest(contamination=0.1)
is_capturing = False
known_dns_records = {"www.example.com": "93.184.216.34"}
known_certificates = {}

# Configure Logging
logging.basicConfig(level=logging.INFO, filename='nids.log', filemode='a',
format='%(asctime)s - %(message)s')

# User credentials for authentication
users = {
    'admin': 'admin',
    'viewer': 'viewer'
}
```

```
@app.route('/')
def index():
    return "Network Intrusion Detection System Backend"

@app.route('/auth', methods=['POST'])
def auth():
    data = request.json
    username = data.get('username')
    password = data.get('password')
    if users.get(username) == password:
        return jsonify({'message': 'Authenticated', 'role': username})
    else:
        return jsonify({'message': 'Invalid credentials'}), 401

@app.route('/start_capture', methods=['POST'])
def start_capture():
    global is_capturing
    is_capturing = True
    duration = request.json.get('duration', 60)
    try:
        sniff(prn=packet_callback, timeout=duration, store=False)
    except Exception as e:
        logging.error(f"Error during packet capture: {e}")
    return jsonify({'message': 'Packet capturing started.'})

@app.route('/stop_capture', methods=['POST'])
def stop_capture():
    global is_capturing
    is_capturing = False
```

```
    return jsonify({'message': 'Packet capturing stopped.'})

@app.route('/clear_packets', methods=['POST'])
def clear_packets():
    global captured_packets
    captured_packets = []
    return jsonify({'message': 'Captured packets cleared.'})

@app.route('/captured_packets', methods=['GET'])
def get_captured_packets():
    return jsonify(captured_packets)

@app.route('/filter_packets', methods=['GET'])
def filter_packets():
    search_term = request.args.get('search', '').lower()
    filtered_packets = [packet for packet in captured_packets if
                        search_term in packet['src'].lower() or
                        search_term in packet['dst'].lower() or
                        search_term in packet['protocol'].lower() or
                        (packet.get('attack_type', 'N/A').lower() != 'n/a' and search_term
in packet['attack_type'].lower())]
    return jsonify(filtered_packets)

@app.route('/download_packets', methods=['GET'])
def download_packets():
    try:
        si = io.StringIO()
        cw = csv.writer(si)
        cw.writerow(["Source IP", "Destination IP", "Protocol Summary", "Attack
```

```
Type"])
        for packet in captured_packets:
            cw.writerow([packet['src'], packet['dst'], packet['protocol'],
packet.get('attack_type', 'N/A')])
        output = io.BytesIO()
        output.write(si.getvalue().encode('utf-8'))
        output.seek(0)
        return send_file(output, mimetype='text/csv',
download_name='captured_packets.csv', as_attachment=True)
    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/network_attacks', methods=['GET'])
def network_attacks():
    return jsonify({
        'mitm_packets': packet_stats['mitm_packets'],
        'arp_spoofing_packets': packet_stats['arp_spoofing_packets'],
        'dns_spoofing_packets': packet_stats['dns_spoofing_packets'],
        'https_spoofing_packets': packet_stats['https_spoofing_packets'],
        'http_spoofing_packets': packet_stats['http_spoofing_packets'],
        'ip_spoofing_packets': packet_stats['ip_spoofing_packets'],
        'email_hijacking_packets': packet_stats['email_hijacking_packets'],
        'wifi_eavesdropping_packets': packet_stats['wifi_eavesdropping_packets'],
        'phishing_attacks': packet_stats['phishing_attacks'],
        'malware_attacks': packet_stats['malware_attacks'],
        'dos_attacks': packet_stats['dos_attacks'],
        'scanning_probing_attacks': packet_stats['scanning_probing_attacks'],
        'session_hijacking_attacks': packet_stats['session_hijacking_attacks'],
        'brute_force_attacks': packet_stats['brute_force_attacks'],
```

```
        'zero_day_attacks': packet_stats['zero_day_attacks'],
        'protocol_attacks': packet_stats['protocol_attacks'],
        'insider_threats': packet_stats['insider_threats'],
        'total_packets': packet_stats['total_packets']
    })

@app.route('/analyze_attacks', methods=['GET'])
def analyze_attacks():
    attack_type = request.args.get('type')
    if attack_type not in attack_packets:
        return jsonify([])
    return jsonify(attack_packets[attack_type])

@app.route('/logs', methods=['GET'])
def get_logs():
    with open('nids.log', 'r') as log_file:
        log_content = log_file.readlines()
    log_content.reverse()  # Arrange logs latest first
    return jsonify({'logs': ''.join(log_content)})

@app.route('/filter_logs', methods=['GET'])
def filter_logs():
    search_term = request.args.get('search', '').lower()
    with open('nids.log', 'r') as log_file:
        log_content = log_file.readlines()

    filtered_logs = [log for log in log_content if search_term in log.lower()]
    return jsonify({'logs': filtered_logs})
```

```
@app.route('/clear_logs', methods=['POST'])
def clear_logs():
    if request.json.get('role') != 'admin':
        return jsonify({'message': 'Unauthorized'}), 403

    try:
        open('nids.log', 'w').close()  # Clear the log file
        return jsonify({'message': 'Logs cleared.'})
    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/fetch_threat_intelligence', methods=['GET'])
def fetch_threat_intelligence():
    api_url = "https://api.xforce.ibmcloud.com/api/alerts/urgency"
    headers = {"Authorization": "Bearer YOUR_API_KEY"}
    response = requests.get(api_url, headers=headers)
    if response.status_code == 200:
        return jsonify(response.json())
    else:
        return jsonify({'error': 'Failed to fetch threat intelligence'}), 500

@app.route('/block_ip', methods=['POST'])
def block_ip():
    ip = request.json.get('ip')
    automate_response(ip)
    return jsonify({'message': f'Blocked IP {ip}.'})

@app.route('/download_mitm_report', methods=['GET'])
def download_mitm_report():
```

```
    si = io.StringIO()
    cw = csv.writer(si)
    cw.writerow(["Source IP", "Destination IP", "Protocol", "Details", "Attack Type"])
    for report in mitm_reports:
        cw.writerow([report['ip_src'], report['ip_dst'], report['protocol'], report['details'], report['attack_type']])
    output = io.BytesIO()
    output.write(si.getvalue().encode('utf-8'))
    output.seek(0)
    return send_file(output, mimetype='text/csv', download_name='MITM_Report.csv', as_attachment=True)

def automate_response(ip):
    os.system(f"sudo iptables -A INPUT -s {ip} -j DROP")

def packet_callback(packet):
    global is_capturing
    if not is_capturing:
        return

    try:
        if packet.haslayer(IP):
            ip_src = packet[IP].src
            ip_dst = packet[IP].dst
            packet_stats['total_packets'] += 1

            data = {
                'total_packets': packet_stats['total_packets'],
```

```
        'src': ip_src,
        'dst': ip_dst,
        'protocol': packet.summary()
    }

    captured_packets.append(data)
    store_historical_data(data)

    # Man-in-the-Middle (MITM) Attacks Detection and Prevention
    # 1. ARP Spoofing
    if packet.haslayer(ARP):
        if packet[ARP].op == 2:
            arp_src = packet[ARP].psrc
            arp_dst = packet[ARP].pdst
            if arp_src in known_gateways or arp_dst in known_gateways:
                packet_stats['arp_spoofing_packets'] += 1
                attack_details = {
                    'ip_src': ip_src,
                    'ip_dst': ip_dst,
                    'protocol': 'ARP',
                    'details': 'MITM (ARP Spoofing) detected',
                    'attack_type': 'MITM (ARP Spoofing)'
                }
                attack_packets['arp_spoofing_attacks'].append(attack_details)
                log_attack('MITM (ARP Spoofing)', attack_details)
                data['attack_type'] = 'MITM (ARP Spoofing)'
                send_alerts(attack_details)
                automate_response(ip_src)
                # Prevention: Static ARP Entries
```

```
                os.system(f"arp -s {ip_src} {packet[ARP].hwsrc}")

        # 2. DNS Spoofing
        if packet.haslayer(DNS):
            dns_qry = packet[DNS].qd.qname.decode('utf-8') if packet[DNS].qd
else ""
            dns_resp = packet[DNS].an.rdata if packet[DNS].an else ""
            if dns_qry in known_dns_records and dns_resp !=
known_dns_records[dns_qry]:
                packet_stats['dns_spoofing_packets'] += 1
                attack_details = {
                    'ip_src': ip_src,
                    'ip_dst': ip_dst,
                    'protocol': 'DNS',
                    'details': f"DNS Spoofing detected. Query: {dns_qry}, Response:
{dns_resp}",
                    'attack_type': 'MITM (DNS Spoofing)'
                }
                attack_packets['dns_spoofing_attacks'].append(attack_details)
                log_attack('MITM (DNS Spoofing)', attack_details)
                data['attack_type'] = 'MITM (DNS Spoofing)'
                send_alerts(attack_details)
                automate_response(ip_src)
                # Prevention: Secure DNS Servers
                os.system("echo 'nameserver 8.8.8.8' > /etc/resolv.conf")

        # 3. SSL/TLS Hijacking
        if packet.haslayer(IP) and packet[IP].dport == 443:
            try:
```

```
                cert = crypto.load_certificate(crypto.FILETYPE_PEM,
packet[Raw].load)
                cert_fingerprint = cert.digest("sha256").decode("utf-8")
                if packet[IP].dst in known_certificates and
known_certificates[packet[IP].dst] != cert_fingerprint:
                    packet_stats['https_spoofing_packets'] += 1
                    attack_details = {
                        'ip_src': ip_src,
                        'ip_dst': ip_dst,
                        'protocol': 'HTTPS',
                        'details': 'MITM (HTTPS Spoofing) detected: Certificate
mismatch',
                        'attack_type': 'MITM (HTTPS Spoofing)'
                    }
                    attack_packets['https_spoofing_attacks'].append(attack_details)
                    log_attack('MITM (HTTPS Spoofing)', attack_details)
                    data['attack_type'] = 'MITM (HTTPS Spoofing)'
                    send_alerts(attack_details)
                    automate_response(ip_src)
                    # Prevention: Enforce HTTPS
                    os.system("iptables -A OUTPUT -p tcp --dport 80 -j REJECT")
                except Exception as e:
                    logging.error(f"Error processing HTTPS packet: {e}")

            # 4. HTTP Spoofing
            if packet.haslayer(IP) and packet[IP].dport == 80:
                http_headers = str(packet[Raw].load).lower() if packet.haslayer(Raw)
else ""
                if "location:" in http_headers:
```

```
            packet_stats['http_spoofing_packets'] += 1
            attack_details = {
                'ip_src': ip_src,
                'ip_dst': ip_dst,
                'protocol': 'HTTP',
                'details': 'MITM (HTTP Spoofing) detected: Unusual redirect',
                'attack_type': 'MITM (HTTP Spoofing)'
            }
            attack_packets['http_spoofing_attacks'].append(attack_details)
            log_attack('MITM (HTTP Spoofing)', attack_details)
            data['attack_type'] = 'MITM (HTTP Spoofing)'
            send_alerts(attack_details)
            automate_response(ip_src)

        # 5. IP Spoofing
        if packet.haslayer(IP):
            if packet[IP].src != packet[IP].src:
                packet_stats['ip_spoofing_packets'] += 1
                attack_details = {
                    'ip_src': ip_src,
                    'ip_dst': ip_dst,
                    'protocol': 'IP',
                    'details': 'MITM (IP Spoofing) detected: IP mismatch',
                    'attack_type': 'MITM (IP Spoofing)'
                }
                attack_packets['ip_spoofing_attacks'].append(attack_details)
                log_attack('MITM (IP Spoofing)', attack_details)
                data['attack_type'] = 'MITM (IP Spoofing)'
                send_alerts(attack_details)
```

```
            automate_response(ip_src)
            # Prevention: Ingress and Egress Filtering
            os.system("iptables -A INPUT -s {} -j DROP".format(ip_src))

        # 6. Email Hijacking
        if packet.haslayer(Raw):
            email_content = packet[Raw].load.decode('utf-8', errors='ignore')
            if "spoofed_email" in email_content:
                packet_stats['email_hijacking_packets'] += 1
                attack_details = {
                    'ip_src': ip_src,
                    'ip_dst': ip_dst,
                    'protocol': 'Email',
                    'details': 'MITM (Email Hijacking) detected',
                    'attack_type': 'MITM (Email Hijacking)'
                }
                attack_packets['email_hijacking_attacks'].append(attack_details)
                log_attack('MITM (Email Hijacking)', attack_details)
                data['attack_type'] = 'MITM (Email Hijacking)'
                send_alerts(attack_details)
                automate_response(ip_src)
                # Prevention: Email Authentication (SPF, DKIM, DMARC)
                os.system("spf_tool -c /etc/spf.conf")

        # 7. Wi-Fi Eavesdropping
        if packet.haslayer(Raw) and "eavesdropping" in
packet[Raw].load.decode('utf-8', errors='ignore').lower():
            packet_stats['wifi_eavesdropping_packets'] += 1
            attack_details = {
```

```
            'ip_src': ip_src,
            'ip_dst': ip_dst,
            'protocol': 'Wi-Fi',
            'details': 'MITM (Wi-Fi Eavesdropping) detected',
            'attack_type': 'MITM (Wi-Fi Eavesdropping)'
        }
        attack_packets['wifi_eavesdropping_attacks'].append(attack_details)
        log_attack('MITM (Wi-Fi Eavesdropping)', attack_details)
        data['attack_type'] = 'MITM (Wi-Fi Eavesdropping)'
        send_alerts(attack_details)
        automate_response(ip_src)
        # Prevention: WPA3 Encryption
        os.system("wpa_supplicant -c /etc/wpa_supplicant.conf")

    # Phishing Attacks Detection
    if packet.haslayer(Raw):
        email_content = packet[Raw].load.decode('utf-8', errors='ignore')
        if "phishing" in email_content.lower():
            packet_stats['phishing_attacks'] += 1
            attack_details = {
                'ip_src': ip_src,
                'ip_dst': ip_dst,
                'protocol': 'Email',
                'details': 'Phishing detected',
                'attack_type': 'Phishing'
            }
            attack_packets['phishing_attacks'].append(attack_details)
            log_attack('Phishing', attack_details)
            data['attack_type'] = 'Phishing'
```

```
            send_alerts(attack_details)
            automate_response(ip_src)

        # Malware Attacks Detection
        if packet.haslayer(Raw):
            if "malware" in packet[Raw].load.decode('utf-8', errors='ignore').lower():
                packet_stats['malware_attacks'] += 1
                attack_details = {
                    'ip_src': ip_src,
                    'ip_dst': ip_dst,
                    'protocol': 'Unknown',
                    'details': 'Malware detected',
                    'attack_type': 'Malware'
                }
                attack_packets['malware_attacks'].append(attack_details)
                log_attack('Malware', attack_details)
                data['attack_type'] = 'Malware'
                send_alerts(attack_details)
                automate_response(ip_src)

        # Denial of Service (DoS) and Distributed Denial of Service (DDoS) Attacks Detection
        if len(captured_packets) > 0:
            if data['total_packets'] > (sum(stats['total_packets'] for stats in captured_packets) / len(captured_packets)) * 2:
                packet_stats['dos_attacks'] += 1
                attack_details = {
                    'ip_src': ip_src,
```

```
                'ip_dst': ip_dst,
                'protocol': 'DoS',
                'details': 'Denial of Service attack detected',
                'attack_type': 'DoS'
            }
            attack_packets['dos_attacks'].append(attack_details)
            log_attack('DoS', attack_details)
            data['attack_type'] = 'DoS'
            send_alerts(attack_details)
            automate_response(ip_src)
            # Prevention: Rate Limiting, Traffic Filtering, Load Balancing
            os.system("iptables -A INPUT -p tcp --dport 80 -m limit --limit 25/minute --limit-burst 100 -j ACCEPT")

        # Network Scanning and Probing Detection
        if packet.haslayer(IP):
            if len(captured_packets) > 10 and sum(1 for p in captured_packets[-10:] if p['src'] == ip_src) > 5:
                packet_stats['scanning_probing_attacks'] += 1
                attack_details = {
                'ip_src': ip_src,
                'ip_dst': ip_dst,
                'protocol': 'Scanning/Probing',
                'details': 'Scanning/Probing detected',
                'attack_type': 'Scanning/Probing'
                }
                attack_packets['scanning_probing_attacks'].append(attack_details)
                log_attack('Scanning/Probing', attack_details)
                data['attack_type'] = 'Scanning/Probing'
```

```
            send_alerts(attack_details)
            automate_response(ip_src)
            # Prevention: Firewall Rules, IDS
            os.system("iptables -A INPUT -p tcp --dport 22 -j DROP")

        # Session Hijacking Detection
        if packet.haslayer(Raw):
            session_content = packet[Raw].load.decode('utf-8', errors='ignore')
            if "session" in session_content.lower():
                packet_stats['session_hijacking_attacks'] += 1
                attack_details = {
                    'ip_src': ip_src,
                    'ip_dst': ip_dst,
                    'protocol': 'Session',
                    'details': 'Session Hijacking detected',
                    'attack_type': 'Session Hijacking'
                }
                attack_packets['session_hijacking_attacks'].append(attack_details)
                log_attack('Session Hijacking', attack_details)
                data['attack_type'] = 'Session Hijacking'
                send_alerts(attack_details)
                automate_response(ip_src)

        # Brute Force Attacks Detection
        if packet.haslayer(Raw):
            brute_force_content = packet[Raw].load.decode('utf-8',
errors='ignore')
            if "login failed" in brute_force_content.lower():
                packet_stats['brute_force_attacks'] += 1
```

```
            attack_details = {
                'ip_src': ip_src,
                'ip_dst': ip_dst,
                'protocol': 'Brute Force',
                'details': 'Brute Force attack detected',
                'attack_type': 'Brute Force'
            }
            attack_packets['brute_force_attacks'].append(attack_details)
            log_attack('Brute Force', attack_details)
            data['attack_type'] = 'Brute Force'
            send_alerts(attack_details)
            automate_response(ip_src)
            # Prevention: Account Lockout, MFA
            os.system("pam_tally2 --user username --reset")

    # Zero-Day Exploits Detection
    if isolation_forest.fit_predict([packet])[0] == -1:
        packet_stats['zero_day_attacks'] += 1
        attack_details = {
            'ip_src': ip_src,
            'ip_dst': ip_dst,
            'protocol': 'Unknown',
            'details': 'Zero-Day Exploit detected',
            'attack_type': 'Zero-Day Exploit'
        }
        attack_packets['zero_day_attacks'].append(attack_details)
        log_attack('Zero-Day Exploit', attack_details)
        data['attack_type'] = 'Zero-Day Exploit'
        send_alerts(attack_details)
```

```
            automate_response(ip_src)

        # Protocol Attacks Detection
        if packet.haslayer(IP):
            if packet[IP].proto not in [1, 6, 17]:  # ICMP, TCP, UDP
                packet_stats['protocol_attacks'] += 1
                attack_details = {
                    'ip_src': ip_src,
                    'ip_dst': ip_dst,
                    'protocol': 'Protocol',
                    'details': 'Protocol attack detected',
                    'attack_type': 'Protocol'
                }
                attack_packets['protocol_attacks'].append(attack_details)
                log_attack('Protocol', attack_details)
                data['attack_type'] = 'Protocol'
                send_alerts(attack_details)
                automate_response(ip_src)

        # Insider Threats Detection
        if packet.haslayer(Raw):
            if "confidential" in packet[Raw].load.decode('utf-8',
errors='ignore').lower():
                packet_stats['insider_threats'] += 1
                attack_details = {
                    'ip_src': ip_src,
                    'ip_dst': ip_dst,
                    'protocol': 'Insider Threat',
                    'details': 'Insider Threat detected: Confidential information
```

```
accessed',
                    'attack_type': 'Insider Threat'
                }
                attack_packets['insider_threats'].append(attack_details)
                log_attack('Insider Threat', attack_details)
                data['attack_type'] = 'Insider Threat'
                send_alerts(attack_details)
                automate_response(ip_src)

            # Send packet data via WebSocket for real-time updates
            socketio.emit('packet_data', data)

    except Exception as e:
        logging.error(f"Error processing packet: {e}")

def log_attack(attack_type, details):
    logging.info(f"{attack_type} attack detected: {details}")

def store_historical_data(packet):
    historical_data.append(packet)

def send_alerts(details):
    send_email(details)
    send_sms(details)

def send_email(details):
    try:
        msg = Message(
            subject="Network Attack Detected",
```

```
            sender=app.config['MAIL_USERNAME'],
            recipients=['rr6022799@gmail.com']
        )
        msg.body = f"Details of the detected attack:\n\n{details}"
        mail.send(msg)
    except Exception as e:
        logging.error(f"Error sending email: {e}")

def send_sms(details):
    try:
        twilio_client.messages.create(
            body=f"Network Attack Detected: {details}",
            from_=TWILIO_PHONE_NUMBER,
            to=ADMIN_PHONE_NUMBER
        )
    except Exception as e:
        logging.error(f"Error sending SMS: {e}")

if _name_ == "_main_":
    socketio.run(app, debug=False, host='0.0.0.0', port=5000)APPENDIX-B
```

SCREENSHOTS

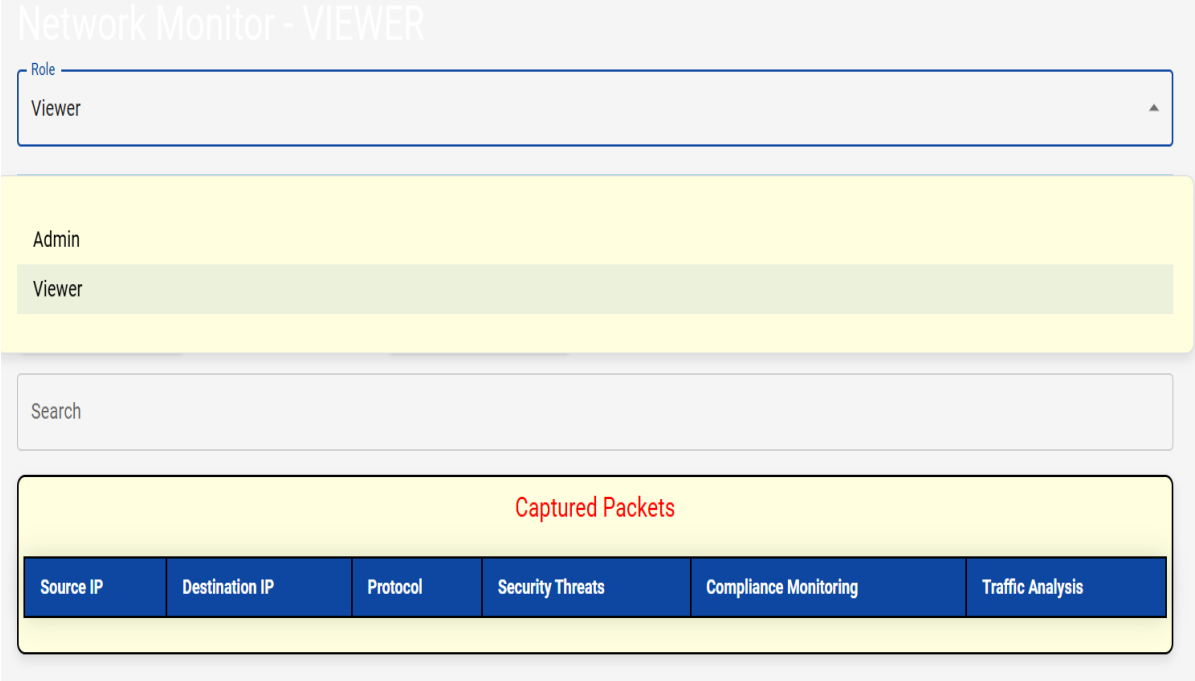

Fig 13.1 Admin-Viewer

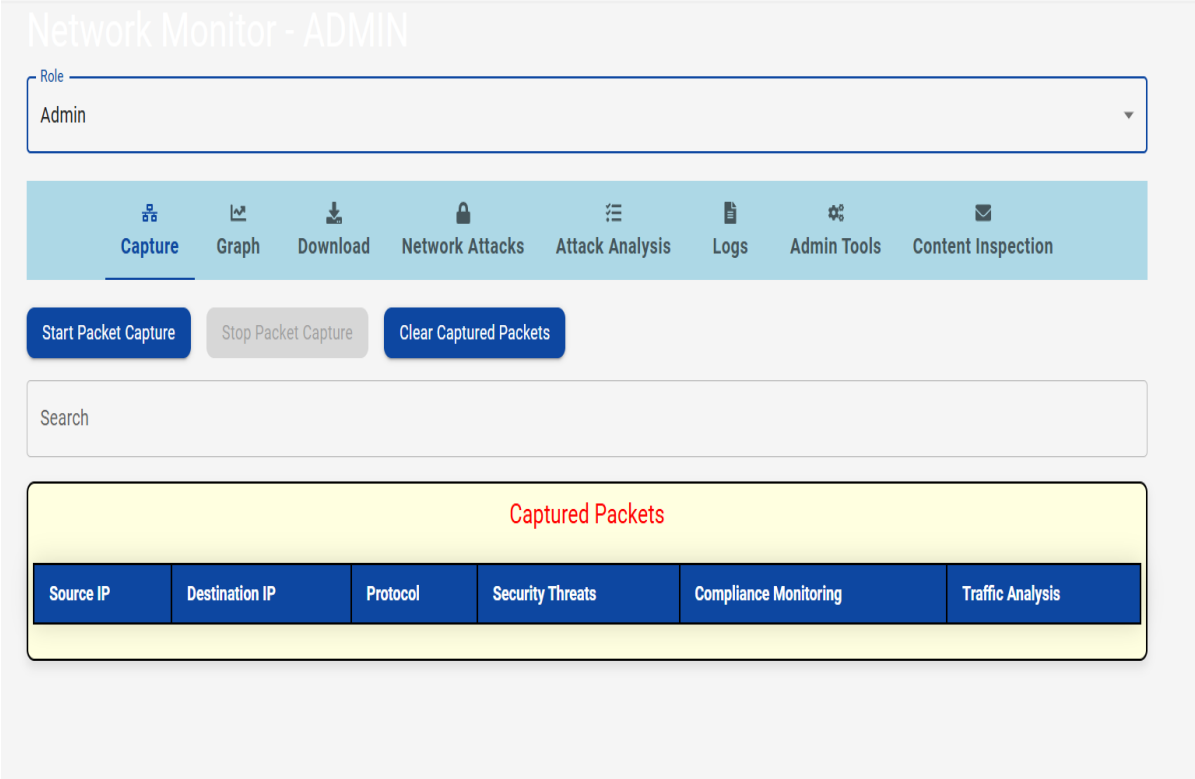

Fig 13.2 Admin-Packet Capturing

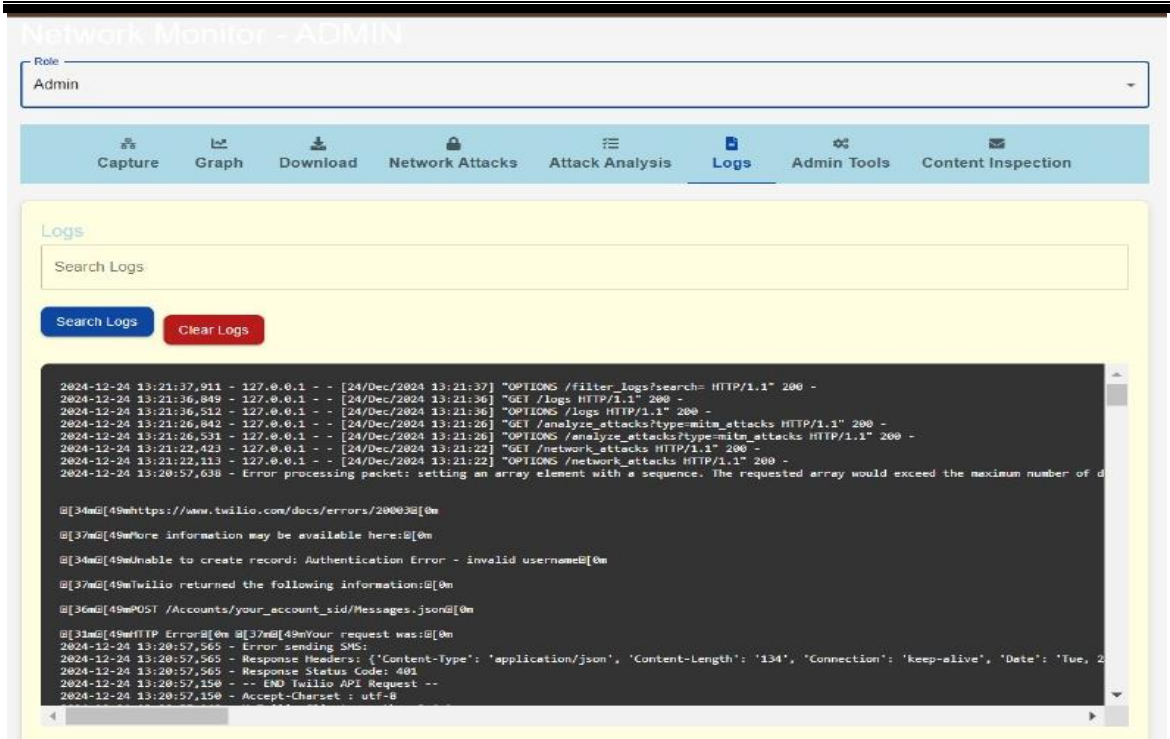

Fig 13.3 Admin Logs

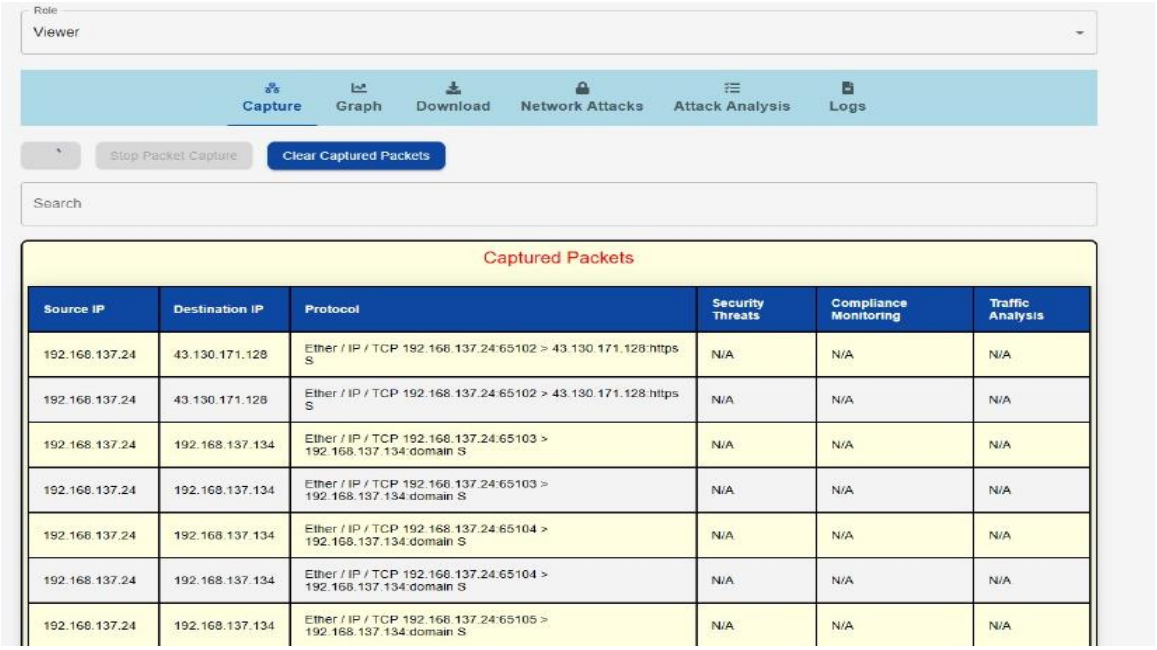

Fig 13.4 Viewer-Packet Capturing

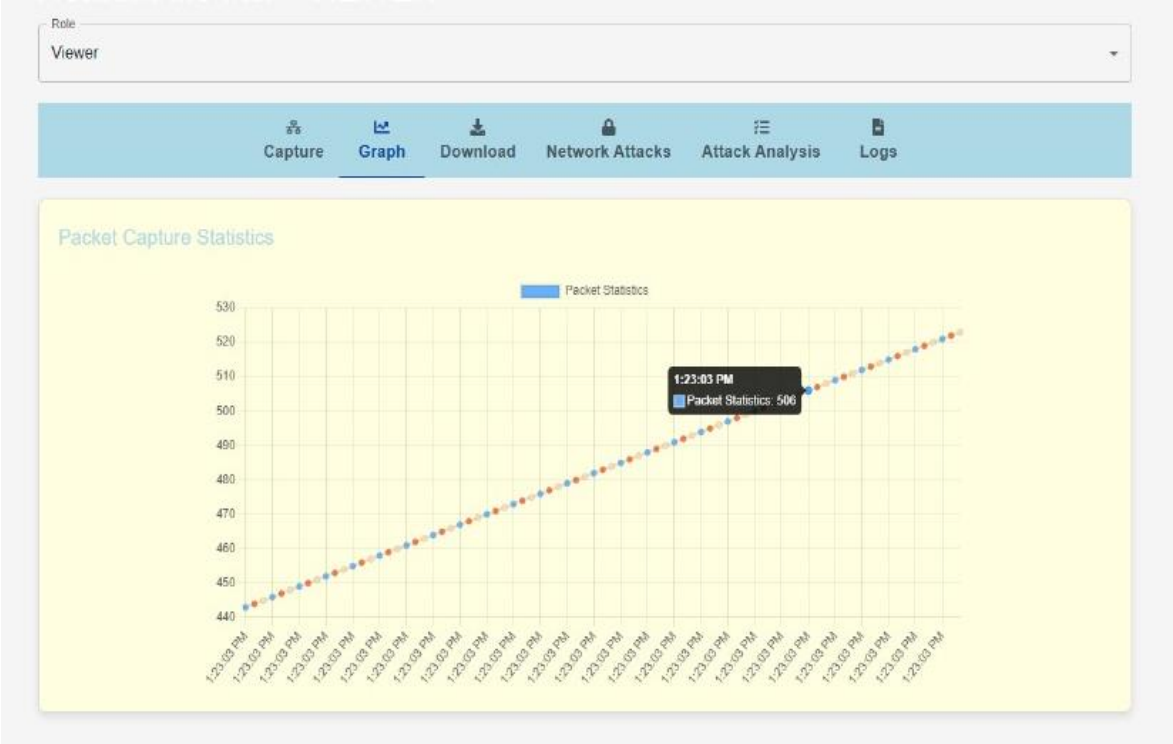

Fig 13.5 Viewer-Graph

APPENDIX-C

ENCLOSURES

1. Journal publication/Conference Paper Presented Certificates of all students.

2. Include certificate(s) of any Achievement/Award won in any project-related event.

3. Similarity Index / Plagiarism Check report clearly showing the Percentage (%). No need for a page-wise explanation.

4. Details of mapping the project with the Sustainable Development Goals (SDGs).