Ensemble Machine Learning Approach For Identifying Real-Time Threats In Security Operations Center

Favour Femi-Oyewole, Victor Osamor, Daniel Okunbor

Abstract— **Cyberattacks can be avoided if threats are identified in advance and robust cybersecurity measures are in place to protect infrastructures. However, in recent years, cyber threats and data breaches have become more prevalent, exploiting vulnerabilities and causing significant financial damage and organizational harm. This often involves compromising sensitive personal information, emphasizing the need for proactive defence strategies led by experienced security professionals. Traditional methods of threat detection involve laborious log analysis due to the multitude of logs generated by network devices. However, ensemble machine learning techniques offer automation within intrusion detection systems, streamlining the threat detection process. This study investigates various ensemble methods, such as blending and stacking, to enhance detection capabilities, both manually and automatically identifying potential cyber threats. The methodology involves implementing a stacking blending ensemble model and conducting feature selection to improve performance. Additionally, a web application interface is developed using the Python Flask web framework to facilitate model deployment and management. Evaluation includes testing on real production network traffic and the CICIDS2017 Thursday-WorkingHours-Morning dataset, with intentional web attacks executed to assess system effectiveness. The ensemble model is evaluated using the Thursday Morning Dataset, achieving high precision, recall, and F1-score of 0.99, with an overall accuracy of 99% in binary classification tasks. These results validate the model's robustness and effectiveness in identifying real-time network traffic patterns and potential security incidents, demonstrating its potential to enhance cybersecurity measures.**

Index Terms—**Cybersecurity, Intrusion Detection System, Machine Learning, Stacking, Blending Ensemble Model**

I. INTRODUCTION

As technology advances, so do the complexities of cyber security risks and attack methods [1], [2]. Malicious actors now employ sophisticated tools and technologies for swift, targeted assaults that can yield significant harm and gather vast amounts of data [3], [4], [5]. In this dynamic landscape, the Security Operations Center (SOC) plays a central role, continuously monitoring an organization's IT infrastructure to swiftly detect and mitigate cybersecurity threats. Whether managed internally or outsourced, SOC teams are entrusted with overseeing and maintaining cybersecurity technologies and evaluating threat data to bolster the organization's security posture [6]. They employ an array of tools, including firewalls, intrusion detection and prevention systems, and security information and event management (SIEM) systems, while adhering to security best practices such as robust password policies, multi-factor authentication, and regular security audits [7].

In the realm of cybersecurity, Security Operations Centers (SOCs) leverage advanced technology and sophisticated computer forensics tools to detect, prevent, and respond to issues related to cyber threats and cyberattacks [8]. The effectiveness of a SOC hinges on its ability to efficiently scrutinize and analyse vast volumes of data, enabling the identification of malicious event patterns [9]. At its core, the SOC meticulously monitors and classifies a multitude of network events, encompassing both benign and malicious activities.

SOCs are the recommended best practice on which large and medium-sized organisations depend for cybersecurity incident detection, notification, and response. Many organisations or enterprises have established SOCs as efficient solutions for monitoring cybersecurity [10]. SOCs function as a core unit within security operations, typically perceived not as a singular entity or system but as an intricate structure responsible for managing and improving an organisation's overall security posture. SOCs serve as centralized defence units within medium or large organizations. The function of SOCs involves identifying, analysing, and addressing cybersecurity threats and incidents using personnel, procedures, and technology as shown in Fig 1 [11], [12], [10]. The effectiveness of SOCs has been demonstrated in enhancing an organization's security posture by proactively addressing, identifying, analyzing, and responding to cybersecurity incidents [13].

SOCs are indispensable strategic resources for organizations, actively identifying, preventing, and facilitating the swift recovery from cyberattacks. Given the heightened prevalence of cybercrimes and cyberattacks, the role of SOCs is pivotal for organizations. The year 2017, in particular, witnessed some of the most severe and notable security breaches, exemplified by the Equifax breach, where hackers gained access to approximately 145.5 million user account credentials.

The stolen information included personal identities, account numbers, vehicle license numbers, and banking information of around 5 million individuals. This breach had significant financial implications, but perhaps even more damaging was the loss of consumer trust and the

Manuscript received April 5, 2024; revised September 27, 2024.

F. Femi-Oyewole is a PhD Candidate at Covenant University Km 10 Ota. Ogun State Nigeria. (corresponding author; phone: +234 80 98186081; e-mail: Favour.Femi-Oyewole@Stu.cu.edu.ng).

V. Osamor is a Professor of Computer and Information Sciences at Covenant University, KM 10 Ota, Ogun State, Nigeria (e-mail: victor.osamor@covenantuniversity.edu.ng).

D. Okunbor is a Professor of Mathematics and Computer Science at Rice University, Fayetteville State University, Fayetteville, California USA. (e-mail: diokunbor@uncfsu.edu).

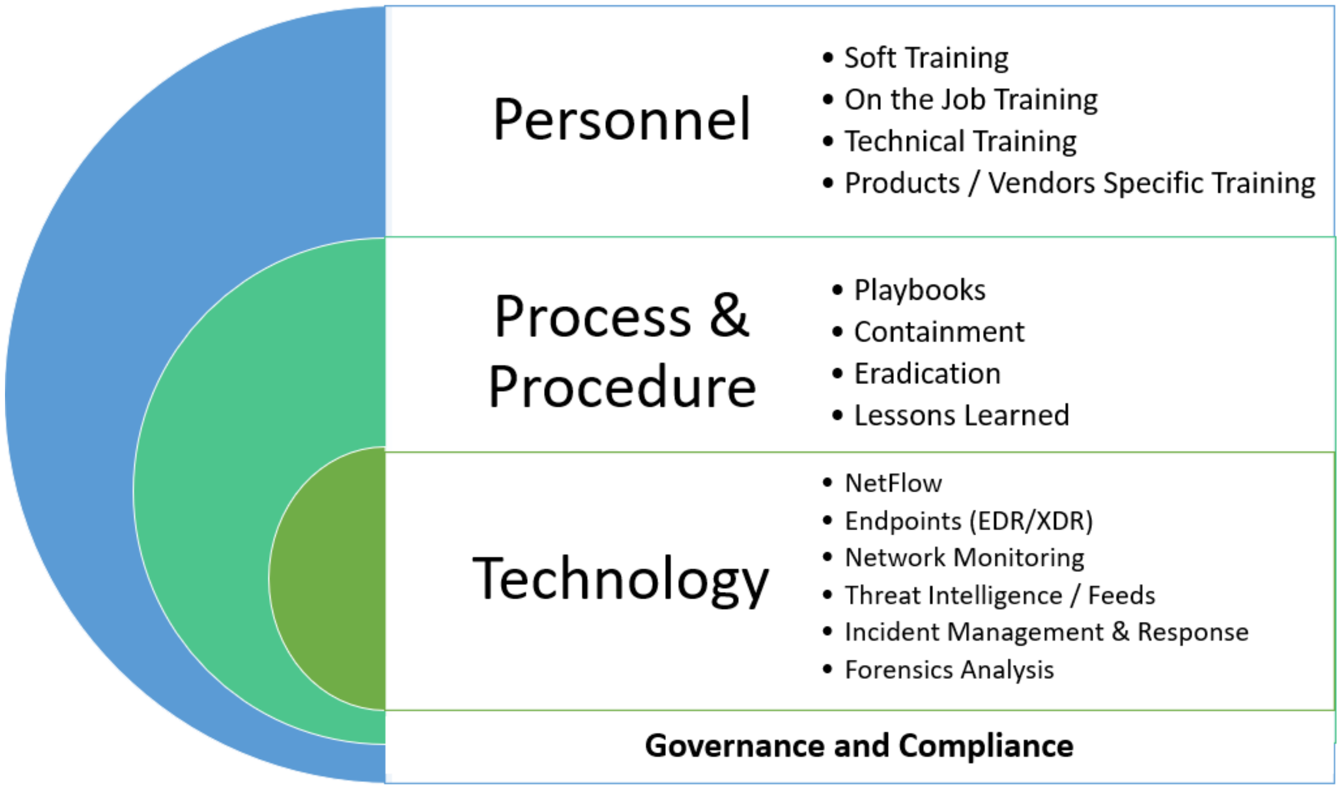


Fig. 1. SOC Highlevel Architecture ([14] and owned by the authors).

erosion of the company's brand which had been built over many years [15]. Incidents like the Equifax breach have compelled businesses to recognize the rapidly changing threat landscape and attack strategies, necessitating a shift away from basic firewall and antivirus measures toward more comprehensive and resilient processes, such as establishing a dedicated Security Operations Center [16]. To prevent sensitive information breaches, companies must proactively identify vulnerabilities and threats, enabling early incident detection for rapid response and, in worst-case scenarios, faster recovery. Establishing a Security Operations Center (SOC) equipped with intrusion detection systems and security information and event management (SIEM) is the most effective way to strategically oversee, analyse, and manage an organization's security strategy [17].

The challenges and drawbacks of SOCs include:

1) Detecting Sophisticated Attacks: Detecting sophisticated attacks poses a challenge for analysts. The complex nature and stealthy techniques employed in modern attacks make them particularly difficult to detect. According to [18], the complexity of these attacks presents difficulties for analysts in identifying them within extensive and intricate datasets. Analysts often have to depend on their experience to counter these attacks, especially when the attacker successfully bypasses existing technical controls.

2) Large Volume of Alerts: SOC analysts struggle with a significant challenge related to the large number of alerts they encounter. The volume of alerts often makes it difficult for analysts to detect actual attacks amidst the multitude of notifications. [19] highlighted the presence of false incidents (false positives), emphasizing that not all alerts translate into legitimate incidents. Researchers propose several techniques to tackle this issue, including limiting log collection to essential assets/devices, adjusting policies, and filtering out unnecessary alerts (noise) [19]. Despite these recommendations, literature evidence indicates that analysts persistently face difficulties in managing the overwhelming volume of alerts they handle.

3) Incident Management and Incident Handling Complexity: SOC analyst's primary responsibility is to respond to incidents, and all incidents must be handled in a way that minimizes further damage [20], [21]. In addition, analysts are under time constraints when dealing with incidents [21]. Detecting an attack is one thing; dealing with it through the incident-handling process is quite another. Detecting and managing security incidents (attacks) is critical to a SOC's success. Incident management involves collaborating with (or escalating to) other teams, either internal or external to the organization, to reduce or eliminate the impact of any attack [22]. Since the complexity of incident handling can be intimidating for less experienced analysts, they must rely on the guidance of more experienced analysts. With literature suggesting that SOCs are having difficulty retaining experienced analysts, dealing with complex incidents becomes difficult.

4) Analyst Burnout: The issue of burnout represents a significant challenge for SOC analysts and has become a focal point for researchers [23]. Researchers widely agree that burnout is the outcome of various organizational, environmental, and human factors, including alert fatigue, stress, workload, and anxiety. These factors, in turn, contribute to turnover among analysts [21], [24].

5) False Positives: False positives (FP) are among the numerous challenges confronting SOC analysts. FP refers to the instances of 'false alarms' presented to analysts [18], [24]. These alarms, while not legitimate attacks (true

positives), misleadingly prompt analysts to initiate investigations, ultimately resulting in a waste of analyst time. FP contributes to the high volume of alerts faced by analysts. Similar to the alert volume, FPs also arise from collecting excessive logs. [20] caution that high FPs may lead to an IDS operator completely disregarding alerts. Factors contributing to FPs include system misconfigurations and weak signature/detection strings designed by the vendor or the analysts, which inaccurately match legitimate network traffic. Recommendations for reducing FPs include policy tuning and the application of machine learning [25]. Businesses can also invest more effort in addressing misconfiguration issues to alleviate the burden on analysts. Tuning policies and filtering out false positives are integral aspects of an analyst's responsibilities [19].

6) False Negatives: A challenge encountered by SOC analysts is false negatives (FN). FNs typically occur when a detection system fails to identify legitimate security events, presenting a difficulty for analysts [26], [18]. Analysts must then resort to alternative indicators on the network, such as employing behavioural analytical techniques, to identify malicious activity.

7) Assessment Methods: SOC managers typically anticipate high operational performance from their analysts [27], [23]. Nevertheless, research indicates a deficiency in appropriate metrics and criteria for evaluating the analysts' performance. The challenge lies in cases where analysts and managers do not align on how performance should be measured, and managers encounter difficulty in formulating effective metrics. [23] assert that analysts derive benefits from well-defined metrics as their bonuses and promotions are determined based on these metrics. Analysts seek objective metrics that encompass various aspects of their work.

8) Workloads: [28] argue that SOC analysts struggle with more alerts than they can effectively investigate. This is in line with [25], who attribute the heightened workload primarily to the overwhelming number of alerts that analysts need to handle. Workload is a significant challenge faced by SOC analysts. Previous research indicates that an increased workload also hampers analysts' ability to maintain situational awareness, vigilance, and attention [20]. [20] posit that workload adversely affects analyst's monitoring, analysis, and response capabilities. The authors further argue that attention has an impact on the response time of operators (analysts), and the same applies to eye movement in terms of vigilance. These human factors contributing to suboptimal performance are often underexplored or insufficiently measured to track how performance can be monitored and enhanced.

9) Tacit Knowledge: The challenge associated with tacit knowledge extends beyond the SOC environment, as many professions encounter similar issues. Nonetheless, within the SOC environment, tacit knowledge can impede or delay investigations. A key problem with tacit knowledge is that experienced analysts may struggle to articulate the rationale behind their actions, making it difficult for less experienced analysts to learn. One potential solution to this challenge is for SOCs to implement playbooks or run books, accompanied by thoroughly documented processes that less experienced analysts can refer to for decision-making support within SOCs.

10) Experienced Shortage: Analysts encounter challenges related to a shortage of experience and skills. The frequent turnover of analysts poses difficulties for SOCs in retaining their most experienced and qualified professionals [29]. This presents a dilemma for less experienced analysts, as the chance to learn from experienced counterparts becomes limited. Given the ever-evolving nature of cyber threats, analysts need periodic training, whether through in-house programs or external paid training, to ensure they possess the skills necessary to counteract attackers [27].

11) Repetitive and Manual Processes: The utilization of repetitive, manual, and monotonous step-by-step processes in certain SOCs has been identified as a source of dissatisfaction among SOC analysts [28], [27], [24]. Adhering strictly to these rigid processes may impede the creativity of analysts within the SOC [23]. Moreover, relying on manual processes and analyses proves highly inadequate for enterprise organizations, emphasizing the need for a shift toward automation [23], [19]. The integration of SIEM technologies, coupled with security event visualization systems, can alleviate some of the burdens associated with manual and repetitive procedures [29], [30].

12) Communication issues Between Teams: Inefficient communication among analysts is identified as a challenge in SOCs [27]. [21] emphasize the importance of communication and information sharing for SOC success, noting that team members often struggle to find time for communication when under pressure. The lack of effective communication within the team has an impact on overall performance [21]. [23] assert that SOCs should prioritize addressing communication gaps to ensure analysts do not feel isolated or left behind.

To address these challenges, machine learning models have been integrated to predict malicious attacks, mitigate the risk of false positives, optimize efforts, and reduce costs and time. False negatives, which pose substantial risks to organizations and can result in significant damage, are also a concern. Additionally, the extensive involvement of security analysts and other personnel in SOCs exacerbates the challenges of false positives and negatives, potentially introducing human errors into SOC processes. This research proposes a system that leverages various machine learning techniques to effectively classify threats, address false positives and negatives, minimise model overfitting, and identify new patterns in incoming traffic. Moreover, the proposed system aims to reduce human errors stemming from increased human interactions within SOCs [31].

Machine learning (ML) is reshaping the cybersecurity landscape, serving as a potent tool against the ever-evolving threat landscape. These algorithms excel in processing vast datasets at speeds beyond human capability, unveiling concealed patterns, anomalies, and compromise indicators [32], [33]. One of ML's pivotal roles in cybersecurity is its capacity to augment threat detection by discerning subtle deviations from established patterns and identifying anomalies that may signal potential attacks, including previously unseen ones. Additionally, ML streamlines incident response by automating labour-intensive tasks like log analysis and alert prioritization, thus reducing false positives and negatives. Its adaptability allows it to evolve alongside the threat landscape, learning from each encounter and progressively enhancing its capabilities [34], [35], [36].

This research focuses on the detection of internal and external network threats using traditional intrusion detection systems (IDS), a critical component of the SOC. In the realm of threat detection within network traffic, the challenges of false positives and false negatives are prominent, consuming substantial time and resources [37], [38]. Existing network threat detection criteria or signatures often prove inefficient, leading to elevated false positive rates (Kumar, 2007) [39].

This research employs machine learning techniques for developing threat detection systems, a choice grounded in the rationale outlined in subsequent subsections, which also elucidate the critical role of the SOC. Machine learning-driven SOC systems leverage advanced analytics to proactively identify real-time cyberattack patterns, thereby mitigating potential harm. Within the SOC framework, SIEM plays a crucial role by integrating security information management (SIM) and security event management (SEM). SIM collates data from diverse sources, while SEM identifies and responds to security incidents, collectively reinforcing the organization's security posture [40]. SIEM systems aggregate data from various sources, including network devices, servers, applications, and security equipment, facilitating precise data analysis for informed decision-making. Leveraging advanced analytics such as machine learning and artificial intelligence, SIEM identifies unusual behaviour and triggers alerts upon detecting anomalies. These alerts empower SOC analysts to swiftly investigate and neutralize potential threats. In essence, the SOC serves as a vigilant guardian of an organization's digital assets, employing a range of tools and strategies, with SIEM and machine learning augmenting its capabilities to promptly and accurately detect and respond to cybersecurity incidents.

This study aims to enhance the effectiveness of Security Operations Centers (SOCs) in identifying and mitigating security threats by developing and evaluating an ensemble machine learning approach tailored for network threat detection. The aim of this study was achieved with the following specific objectives:

1) To collect and preprocess relevant data from various sources for analysis;
2) perform data aggregation and exploratory data analysis formulate a predictive model using ensemble machine learning techniques;
3) develop an application programming interface (API) to provide intelligent security information;
4) evaluate the proposed ensemble machine learning model using real-world network traffic data.

II. Related Works

[41] introduced the use of the Bryant Kill Chain method for intrusion detection. A virtual Windows 7 workstation was used to simulate a corporate environment, and pen testing was done using the Kali Linux default toolset on 26 security test cases picked by the researcher. Default LogRhythm SIEM and the Deconstructed Kill chain method were used for evaluation. Their framework was able to detect 25 out of their 26 test cases (achieving a 96% detection rate), while the default LogRhythm detected 7 out of 26 (achieving a 26.9% detection rate). Their framework demonstrates the ability to align log data or SIEM events with more descriptive event categories. The test cases were limited to attacks coming from the remote shell (rsh) and malware installations. Also, real-time user behaviour (server-clients) was not included in the virtual environment simulation.

[25] presented a user-centric machine learning system that leverages big data of various security logs, alert information, and analyst insights to the identification of risky users. The system provides a complete framework and solution to risky user detection for enterprise security operation centers. The authors also demonstrated that the learning system can learn more insights from the data with highly unbalanced and limited labels, even with simple machine learning algorithms. The authors used a Multi-Layer Neural Network (MNN) with two hidden layers, a Random Forest (RF) with 100 Gini Split trees, a Support Vector Machine (SVM) with a radial basis function kernel, and a Logistic Regression (LR). Their model though intelligent, achieved a low detection accuracy.

[42] proposed a new network forensics framework that uses fully adaptive and computational intelligence approaches to enhance the security operating centers. This model is an accurate and useful ensemble machine learning tool that analyses network flow in real time to detect encrypted or malware traffic with low computing power usage. The authors utilized an ensemble architecture, combining SVM, ANN, Random Forest (RF), and k-Nearest Neighbors (k-NN) to identify malicious activities from data streams. Despite performing similarly or slightly less accurately than other models on all datasets, the novel framework presents a promising approach for the timely detection of malicious traffic in computer networks.

[43] proposed a novel visualization system for finding out network-based Underneath attacks (VISNU), which can help security experts of the CSOC to analyse security events more effectively. The VISNU classifies the security events according to each organization and displays them based on both real-time and accumulated information, such as appearance patterns and history. The proposed algorithm represented security occurrences in cubes, and the height of accumulated cubes and colours was used as an indicator for a security threat. The authors developed the formula height of the cubes. The experimental results demonstrated that it is beneficial for finding abnormal activities from the security events and provides a better understanding and insights for analyzing them. Their method is limited to some abnormalities and needs improvement.

[44] proposed a study that aimed to solve the problem of identifying and ranking malicious activities in enterprise networks based on their level of risk. They developed a system called MADE that uses machine learning techniques with security log data to detect suspicious communication. Unlike other methods which use detection methods, MADE relies on supervised learning to prioritise the most critical areas for enterprise hosts contacted in one month while detecting previously unnoticed malicious activities.

However, it has some limitations such as not being able to detect HTTPS malicious communication appropriately. The system also risks being compromised by adversaries who could exploit its features through adversarial attacks, negatively affecting overall performance. Despite these challenges, authors demonstrated MADE's effectiveness in detecting previously unnoticed malicious activities (18 out of 100), showing the system's potential usefulness in protecting enterprise networks from cyber threats while boasting high precision rates and limited false positives.

[45] introduced a novel intelligence-driven cognitive computing SOC that is based exclusively on progressive, fully automatic procedures. The SOC implements the Lambda machine learning architecture that can analyse a mixture of batch and streaming data, using an Extreme Learning Machine neural network with Gaussian Radial Basis Function kernel (ELM/GRBFk) for the batch data analysis and a Self-Adjusting Memory k-Nearest Neighbors classifier (SAM/k-NN) to examine patterns from real-time streams. The method performs poorly with huge volumes of real-time data. Also, the model introduces latency.

[46] presented an automated malware screening and pattern identification model using three machine-learning approaches. To test the proposed approach, the authors employed several well-known malware assaults such as WannaCry, DBGer, Cerber, Defray, GandCrab, Locky, and nRansom. The training logs and pattern extraction were obtained from the Cuckoo sandbox environment. Their investigations revealed that the TF-IDF approach is capable of identifying most of the malware detection characteristics when compared to other techniques like ET (randomized trees). However, the ET technique is more robust in handling input volatility and dynamic changes in data flow patterns. Overall, the study aims to address security challenges handled by intelligent security.

[47] utilised Gaussian mixture models (GMMs) to detect XSS through web requests and web responses representing the integration of the dual-stage model. Word2Vec model was used for feature engineering on normal and XSS payloads on web traffic datasets (response and requests). Then trained using GMMs, and their model was evaluated using ROC (Receiver operating characteristic). Their model successfully used both response and requests for analytics and prediction of Cross-Site Scripting (XSS) events instead of single stage most current research use. The evaluation metrics showed that their dual-stage model was more effective than a single model. The model was not evaluated on real-time traffic data and real-time XSS payloads. 2. The proposed model was not integrated into a SIEM for real-time alarm generation and prevention mechanism.

[48] presented a revolutionary graphical system that enables security analysts to grasp a comprehensive picture of SIEM rule execution & alert scenarios that might occur in preparation depending on the closest condition graphically and in real-time. Aside from actual rule monitoring, it also allowed security analysts to investigate the reasons behind alarms in an orderly and effective manner using online accounts. The created ruleset was connected to create a graphic representation of the reasons behind protection warnings. Their suggested solution allows security analysts to examine the status of several rules in real-time, analyse susceptible rules, and undertake discovery using login details. In other phrases, a security researcher may profit from concentrating just on the constraints that may cause an alarm.

[49] presented a novel SOC incident categorization and prioritizing strategy that utilizes graph monitoring to identify a selection of ML features. By adopting diagram characteristics, their research shows improved accuracy rates with multiple ML classification approaches. The investigation used three statistical algorithms: LR, XGBoost, and DNN. Results revealed DNN as the highest-performing clustering algorithm with an 8% improvement in the classifier for both threshold and magnified feature packages that include diagram functionalities showing area under curve values of 91% and 99%, respectively.

[50] introduced a new ensemble learning classifier to detect wireless networking assaults. The method combines SVM (Support Vector Machine) and LR (Logistic Regression) models using a pooling algorithm to increase efficiency, reduce misclassification, and improve overall recognition accuracy. By analyzing the data inside the KDD CUP sample that has 41 characteristics, this approach should immediately catch any attack. Further examination utilizing the Spyder IDE revealed that the EL ensemble methods achieved a remarkable prediction performance of 99.2% with a confidence interval of only 0.8 percent. The current body of research on SOC and ensemble learning techniques falls short. To address this gap, we conducted a thorough literature review, and, in this section, we will examine a few relevant articles that shed light on the behavior of SOC and ensemble learning.

[51] proposed a unique approach for approximate representations of skewed datasets using positive sequence approximations. The improved depictions were then fed into an attack detection model specifically designed for industrial control system settings. This model, which employs DNNs and DT classifiers, effectively identifies information security through the novel representation. To assess the technique's performance, researchers evaluated it using ten cross-validations with two real ICS datasets and compared it to Random Forest, Deep Neural Network, and AdaBoost, as well as traditional classifications and existing systems in research. The suggested method exceeded expectations with its exceptional performance while being universally applicable that can readily be integrated into existing ICS systems.

[52] presented evaluation methodology in SOCs. The author asserted that SOCs are a solution for companies seeking to handle regulation through threat monitoring. However, there is currently no conceptual method that encompasses procedures, staff, and technology despite the paradigms covering the technical aspects of these operations. Therefore, it would be beneficial for organizations and stakeholders considering developing, purchasing, or selling similar services to assess the efficacy and development of the products supplied. This study suggests a categorization or evaluation methodology for SOC operations that considers both skills and maturity of benefit rendered.

[19] proposed a comprehensive framework that comprises log capture, evaluation, incident response,

notification, management as well as regular. This approach also entails a Cyber Battle Plan backed by the CSOC architecture and overlaid on top of Protective Management Controls (PMCs). Additionally, the challenges and benefits of implementing SOC are examined.

[12] conducted 18 moderate discussions with SOC researchers and managers from various industrial sectors to identify and address challenges faced by SOC. The analysis of interview data revealed both technical and non-technical obstacles in SOC operations. Moreover, the study uncovered inherent conflicts between SOC directors and analysts that could compromise productivity and quality if left unresolved. From these findings, the authors distilled lessons applicable to both future educational research industries and SOC administration. They call for further studies aimed at improving the effectiveness and efficiency of SOCs.

[31] performed a more profound, discovery-oriented participant observation with a security expert on system security limits, including the accuracy & integrity of their alerts. Their findings show that, notwithstanding all FPs, the vast majority are caused by innocuous impulse alarms described by genuine activity in the external structure, which experts may choose to avoid. To virtually assess the adequacy and efficacy of security solutions, suppliers and investigators must be able to distinguish between different sorts of FP. Warning verification is a time-consuming process that might lead to alarm exhaustion &, ultimately, desensitization.

[53] proposed a framework for secure information exchange in power generation. Their suggested approach monitors security conditions within electricity networks and issues alerts to companies upon detecting suspicious threats. The proposed system entails collecting public data linked to electromagnetic safety and scanning available power control technology on the Internet resulting in an understanding of global security architecture and overpower systems. An e-xtendable honeypot technology is subsequently installed to detect real-time security and intelligence states of power systems. Additionally, cyber risks are combined using a data structure around the transmission lines to capture holistic security consciousness. Investigation findings reveal that their framework can identify control efficacy in real-time hence successfully protecting it from infiltration and incursion.

A. Summary of Gaps Identified in Literature

There is a lack of literature on security operations centers. Only a few studies on security operations centers have been published recently. Most security operations center material is based on best practices and security vendor blogs and presentations. There are few research publications on security operations centers using machine learning. This Chapter reviewed the essential concepts of NN and DL that lay the groundwork for the work presented in the following chapter, which will provide a detailed explanation of the proposed system and how the data flows through the HMM architectures. In this work, the research study explores the implementation of an XGBoost and LSTM for information extraction; both of these architectures have been used for tasks with significant success rates. As described in the project, there is previous work on obtaining security-related information in the Security Operation Center. However, after careful consideration of the literature review, it seems as though there is a lack of research on utilizing ML methods for SOC implementation. There have been many studies focusing on machine learning and other aspects of cyber security, but none specifically speak to integrating a SOC. Moreover, to the best of the knowledge gathered by conducting the comprehensive literature review, currently, there is almost zero Deep learning-based end-to-end approach exists for the SOC. Hence, an end-to-end fully functional ensemble machine learning approach for the SOC is provided as the main project outcome. The ensemble-based model implemented is based on HMM with gradient boosting (XGBoost) and LSTM working together for better predictions. All in all, it's an unprecedented deep learning ensemble-based model for SOC. To further clarify this specific combination of the HMM, LSTM, and XGBoost ensemble model, it hasn't been implemented yet when it comes to predicting cyber threats or the SOC itself both in the research purposes and the industrial level. Again, there are several papers focused on stock market analysis using a similar ensemble approach, but non-existent for the cyber security or SOC domain. Thus, using this particular methodology or ensemble approach for the security operations center is unprecedented and should provide a huge value and a great impact on the fields of cyber security, the security operations center, and hence, the IT security domain in general.

The effectiveness of SOC technologies stands as a linchpin in thwarting modern threats that continually evolve in sophistication and scale. However, despite their vital role, a conspicuous gap in the existing literature becomes apparent when one seeks research specifically tailored to Security Operation Center technologies. This gap presents a compelling opportunity for further investigation and exploration into the intricate workings of SOC technologies and their optimization.

Bridging the Gap with Emerging Technologies: The landscape of cybersecurity technologies is in a perpetual state of flux, with the rapid emergence of new tools and innovative approaches. Notably, technologies such as Artificial Intelligence (AI), Machine Learning (ML), and Big Data analytics have gained prominence for their potential to revolutionize SOC capabilities. However, there exists a noticeable void in research dedicated to probing the depths of these emerging technologies and their transformative potential within SOC environments. Delving into how AI, ML, and Big Data analytics can be harnessed to elevate detection accuracy, expedite response times, and fortify the overall security posture within the SOC represents a significant avenue for scholarly exploration and practical implementation.

There is a research gap in the cyber security field when it comes to effectively deploying publicly available real-world or synthetic datasets. Most of the datasets available are imbalanced and directly affect the end –results, thus the traditional usage of dataset-balancing techniques needs more research, and new techniques are welcomed. An imbalanced dataset occurs when the number of instances belonging to one class is significantly higher than the other(s). This can make it difficult for machine learning algorithms to accurately classify data, as they may become biased toward the majority class and neglect the minority class. In this context, the minority class often represents the class of interest, such as a cyber-attack or malicious behaviour. Because the overwhelming amount and sizes of daily network logs are mainly related to “BENIGN” activities.

III. Research Methodology

The solution proposed in this study addresses the challenges by introducing an ensemble model based on the Hidden Markov model, XGBoost and LSTM to enhance the security posture of SOC. Additionally, it provides solutions for addressing the class imbalance issues, aiming to implement an effective threat detection model in SOC. This section discusses the design of the proposed framework as well as the core logic of the proposed model. The proposed model is committed to detecting malicious threats in real-time while reducing fatigue on the SOC analyst to the barest minimum. This includes the design of a Hidden Markov model to solve the feature/pattern extraction issues to reduce computational time and space.

The proposed model is designed to address the challenges of the existing SOC security posture. It integrates an ensemble model based on the Hidden Markov model, Random Forest, XGBoost, and LSTM to achieve real-time detection of malicious patterns in SOC infrastructure. The proposed model includes several important elements that collaborate to ensure the system's effectiveness, as illustrated in Fig 2. Firstly, a comprehensive dataset of network logs, intrusion alerts, and system logs was collected from various sources. To ensure the quality of the data, a thorough data cleaning process was executed, eliminating any inconsistencies, errors, or noise that may exist in the dataset. The data balancing technique was then implemented to tackle the class imbalance problem. Additionally, the framework integrates an ensemble model for the classification task. The Hidden Markov model, Random Forest, XGBoost, and LSTM models were combined to enhance the accuracy of the classification process.

A. Data Collection

In this section, the data collection process is explored as a critical phase in preparing the dataset for comprehensive analysis. The dataset under consideration, "Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv," originates from the esteemed "CICIDS2017 Thursday Morning Hours Dataset," recognized for its significance in the field of cybersecurity research. The data collection endeavors aimed to encompass a wide spectrum of network traffic scenarios and potential security incidents. The selected dataset comprises a total of 458,968 records and an intricate structure comprising 85 columns, including the pivotal label column, as thoughtfully summarized in Table I.

TABLE I
DATASET INFORMATION

Dataset Name	Total Records	Total Columns	Non-Null Values	Missing Values	Data Types
Thursday WorkingHours-Morning-WebAttacks	458,968	85	170,366	288,602	Float/ObJ

B. Data Preprocessing

The crucial phase in machine learning is the data preparation and cleaning process, where the raw network logs are transformed into a suitable format for machine learning algorithms to detect malicious patterns [58]. Before proceeding with experiments to develop the proposed SOC cybersecurity detection model with optimal detection rates, a few steps must be taken to drop unused columns, and missing values and eliminate duplicated features and columns. as they can impact the effectiveness of the proposed model. The next step involves transforming categorical features into numerical representations and encoding non-numerical string values as integers for utilization in training the proposed model. In the context of network log datasets, this transformation is achieved using a Label Encoder known for its effectiveness in encoding string and categorical features into numerical values [54]. Through this process, non-numerical values are assigned integer values ranging from 0 to $n - 1$, rendering them suitable for preprocessing by machine learning algorithms. Despite the class label being categorical features, it remains unaltered, as the original categories are crucial during processing for classifying different types of attacks in various forms and testing different approaches. Since the LSTM and HMM are part of the proposed ensemble model, it is essential to convert the network log datasets into sequences necessary for training the LSTM and HMM models.

After the label encoder has been applied to the network log datasets, the next step involves normalizing the data. Failure to normalize the data can result in a situation where one feature dominates others, even if the dataset possesses numerous advantageous features. To preprocess the data, both min–max normalization and Z-score normalization are employed. The choice of normalization technique for detecting cyberattacks in SOC relies on the specific characteristics of the data and the algorithm in use.

For the CIC-IDS-2017 dataset, the min–max normalization was utilised due to the widely varying ranges of its features. This technique ensures that all features share the same scale by transforming numeric column values ranging from 10,000 to 100,000 into a numeric range from 0 to 1. Importantly, this transformation maintains the distinctions in value ranges without sacrificing information. The implementation of min–max normalization utilises a scaling formula known as "min–max scaling," as normalized data often enhances the efficiency of training a machine learning algorithm [55]. While the min–max method may eliminate some outliers, its impact on system performance is negligible, as the detection task is specifically designed to identify long-term attacks [56], [57].

The important background for a machine learning model involves developing an algorithm that enables the model to learn from patterns once the dataset has undergone preprocessing. While some datasets consist of both training and test data, the network log datasets utilized in this study are individual, undivided datasets. Consequently, following normalization, the model is prepared in a state that can be utilised by the proposed algorithm by partitioning the datasets into two segments: 70% for the training set and 30% for the testing set.

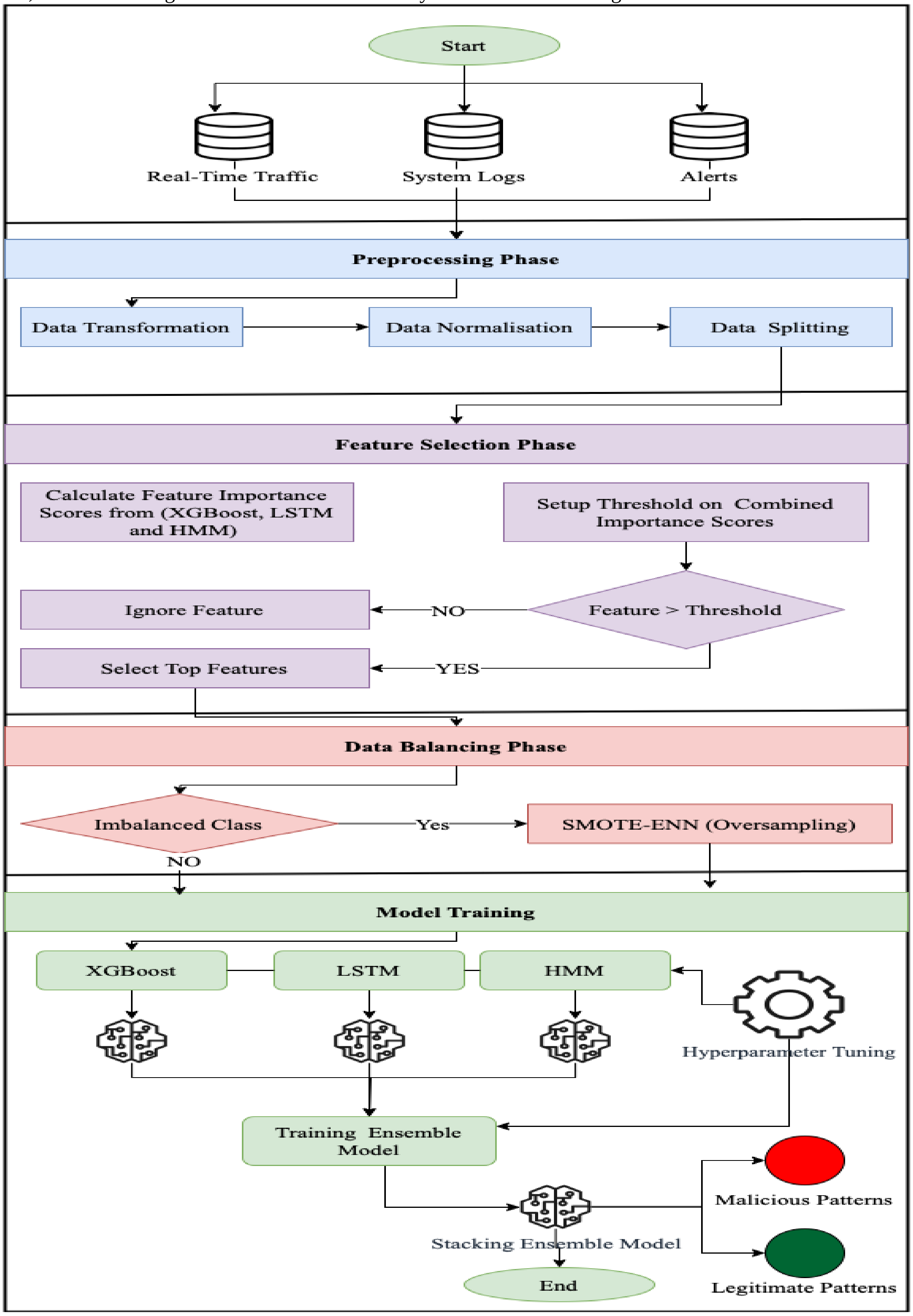


Fig. 2. Structure of the Proposed Model

C. Feature Selection

Feature selection techniques aim to streamline the model's intricacy by reducing the input feature while retaining only those that hold significance and relevance for implementing the model. This process serves the purpose of decreasing execution time and enhancing the performance of the model. The effectiveness of any model, given a high-quality dataset, hinges on the process of feature selection. In this study, the focus is to extract the essential independent features that exhibit robust relationships with the dependent features. Otherwise, the model might attain an accuracy performance below 75%, which is deemed unacceptable for a cybersecurity model intended to counter sophisticated threats and enhance the efficiency of SOCs infrastructure.

In this study, the feature selection process involves leveraging feature importance scores obtained from three different models: XGBoost, Hidden Markov Model (HMM), and LSTM. Each of these models indicates the significance of each feature in contributing to the detection of malicious patterns. The XGBoost model is trained on the network log datasets, and the important scores are extracted. Following that, the HMM is also trained on the sequential dataset to capture hidden states, and emission probabilities are then used as features. Additionally, an LSTM model is designed for sequence modeling, and it is trained on sequential data. Each model provides a ranking or score for each feature based on its contribution to the model's performance. The scores obtained from different models, including XGBoost, HMM, and LSTM, are then combined. This is achieved by using an aggregation technique. The next phase involves setting up a threshold based on their importance scores. Features that surpass the threshold are considered relevant and selected for the next stage. The final selected features from the reduced feature set are then used for training the proposed ensemble model.

D. Data Balancing and Splitting

Imbalanced datasets refer to datasets where the distribution of observations is uneven, indicating that one class label may have a substantial number of instances, while others have fewer. This situation is prevalent in classification problems. Various techniques can address this issue, such as oversampling or undersampling the majority class or a combination of both. Upon analysing the network log datasets, an imbalanced class distribution was identified, primarily driven by its heightened occurrence in the dataset. In the case of CIC-IDS-2017, the number of legitimate samples exceeds the proportion of malicious samples. This could result in a biased cyberattack detection model and negatively affect the detection rate. In this study, the Edited Nearest Neighbors (ENN) algorithm, specifically the Synthetic Minority Over-sampling Technique combined with ENN (SMOTE-ENN), was applied to the network log datasets to establish a balanced distribution among the classes. The SMOTE-ENN technique works by augmenting the minority class samples through linear interpolation, while simultaneously reducing the majority class samples using ENN, which eliminates noisy instances. Additionally, any sample with a class label differing from at least two of its three closest neighbors is removed by SMOTE-ENN.

The network log datasets were divided into a 70% training set and a 30% test set ratio, with a majority of the data assigned to the training set and a smaller portion set aside for testing purposes.

E. Designing the Machine Learning Model

Three machine learning algorithms were utilized in the study to implement the proposed cyberattack detection model within SOC infrastructure. The approach involved leveraging the strengths of each model and employing the stacking ensemble technique to make the final prediction to detect unknown attacks or new instances. The machine learning algorithms utilised in this research include HMM, XGBoost and LSTM.

1) HMM Model: The HMM algorithm is characterized as a doubly stochastic process, comprising an underlying stochastic process that is unobservable and can be investigated through another set of stochastic processes. In a Markov model, the state is directly observable, whereas in an HMM, the state is associated with a probability distribution across a set of outputs (observations). Hence, a sequence of observations produced by an HMM does not directly reveal the sequence of states. A representation of a Hidden Markov Model is expressed in Equation 1.1.

$$\lambda=[A,B,\pi] \tag{1.1}$$

where A denotes the state transition matrix, B denotes the matrix of observation probabilities, and π denotes the initial state probabilities in the Hidden Markov Model.

Three fundamental problems that are to be solved include: In the first problem, considering a set of observations in Equation 1.2 and the HMM in Equation 1.1, the probability of the given observation sequence is computed in Equation 3.3.

$$O = \{O_1, O_2, \dots, O_T\} \tag{1.2}$$

$$\Pr(O|\lambda) \tag{1.3}$$

In the second problem, considering a set of observations in Equation 1.2 and the HMM in Equation 1.1, an optimal state sequence is computed in Equation 1.4.

$$I = \{i_1, i_2, \dots, i_T\} \tag{1.4}$$

In the third problem, considering a set of observations in Equation 1.2, the parameters of the HMM model in Equation 1.1 are adjusted such that Equation 1.3 is maximized.

The first problem can be addressed through either the forward method or the backward method; the second problem is solved by employing the Viterbi algorithm, and the third problem can be addressed using the Baum-Welch algorithm (BW). To estimate the parameters of the HMM, the first problem must be solved by determining the probability value of an observation sequence. Once the parameters are estimated, the model can be trained using either the forward or backward method. The forward variable in Equation 1.5, denotes the probability of the partial observation sequence when the current state, qt, is produced with the state s_i at time t, given the model λ.

$$\alpha_t(i) = P(O, q_t = s_i) \quad (1.5)$$

After initializing the forward variable as expressed in Equation 1.6 and applying the induction formula in Equation 1.7, an optimal model is obtained, until Equation 1.8 converges and is maximized, where Equation 1.9 denotes the probability of state s_i at time t moving to s_j at time $t+1$ and $b_j(O_{t+!})$ is the probability of the observable state at time $t+1$ given the hidden state at j.

$$\alpha_t(i) = \pi_t b_t(O_t) \quad (1.6)$$
$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \times a_{ij}\right] \times b_j(O_{t+1}) \quad (1.7)$$
$$\Pr(O|\lambda) = \sum_{i=1}^{N} \alpha_t(i) \quad (1.8)$$
$$a_{ij} = P\left(q_{t+1} = s_j \middle| q_t = s_i\right), i, j = 1, 2, \dots, N \quad (1.9)$$

The next problem involves determining the sequence of hidden states that is most probable, considering the HMM model and the observation sequence expressed in Equation 1.2.

The Viterbi algorithm, a dynamic programming approach, is utilized to identify the most probable sequence of states, known as the Viterbi path, based on a given sequence of observations. Equation 1.10 introduces the dynamic programming approach for Hidden Markov Models (HMMs), emphasizing the Viterbi algorithm. The initial condition and the induction formula are expressed in Equation 1.10 and Equation 1.11.

$$\delta_1(i) = \pi_i b_i(O_i) \quad (1.10)$$
$$\delta_t(j) = max_1 \le i \le N\ \delta_{t-1}(i) a_{ij} b_j(O_t)\ for\ 2 \le t \le T\ and\ 1 \le j \le N \quad (1.11)$$

where $\delta_t(j)$ denotes the probability of the most likelihood state sequence of the first t observations and j as its final state. By utilizing the dynamic programming algorithm, the Viterbi algorithm identifies the most probable hidden state at time T, as expressed in Equation 1.12 for a given observation sequence, when Equation 1.13 is maximized.

$$q_t^* = args \max_{1<i<N} \delta_T(i) \quad (1.12)$$
$$P_t^* = \max_{1<i<N} \delta_T(i) \quad (1.13)$$

The Baum-Welch algorithm enhances the optimization of the HMM model by re-estimating the parameters recursively until the model converges when Equation 1.3 is maximized, as expressed in Equation 1.14.

$$\lambda' = [A', B', \pi'] \text{ as } \pi'_t = \gamma_1(1), a'_{ij} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)} \text{ nd } b'_j(k) = \frac{\sum_{t=1\, s.t. o_t = k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \quad (1.14)$$

Modern sophisticated attacks follow a series of cybersecurity kill chains such Reconnaissance (State R): Observation: System X is under low-frequency scans from various sources; Weaponization (State W): Observation: Development or acquisition of malicious tools; Delivery (State D): Observation: Delivery of the weaponized payload; Exploitation (State E): Observation: Exploitation of vulnerabilities in the target system (System X); Installation (State I): Observation: Installation of malware or malicious components; Actions on Objectives (State O): Observation: Achieving the objectives, such as data theft or system manipulation.

The proposed model employs a classification technique that considers the states to detect the stages of the cybersecurity kill chain. Since the malicious stages (states) are concealed within the event logs, the proposed model adopts the HMM. In this model, the sequence of transitions between malicious states is concealed, and it is observed through a sequence of emitted observations. The actions observed in event logs serve as emitted observations, while the sequence of concealed states forms a series of attack steps displayed in the upper layer. The lower layer shows the corresponding observations ss depicted in Fig 3. This mapping allows for the representation of each cybersecurity kill chain stage as a distinct state in the HMM. The observations associated with each state provide a way to capture the characteristics or indicators of the attack at that specific stage.

The Baum-Welch algorithm enhances the optimization of the HMM model by re-estimating the parameters recursively until the model converges when Equation 1.3 is maximized, as expressed in Equation 1.14.

$$\lambda' = [A', B', \pi'] \text{ as } \pi'_t = \gamma_1(1), a'_{ij} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)} \text{ nd } b'_j(k) = \frac{\sum_{t=1\, s.t. o_t = k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \quad (1.14)$$

Modern sophisticated attacks follow a series of cybersecurity kill chains such Reconnaissance (State R): Observation: System X is under low-frequency scans from various sources; Weaponization (State W): Observation: Development or acquisition of malicious tools; Delivery (State D): Observation: Delivery of the weaponized payload; Exploitation (State E): Observation: Exploitation of vulnerabilities in the target system (System X); Installation (State I): Observation: Installation of malware or malicious components; Actions on Objectives (State O): Observation: Achieving the objectives, such as data theft or system manipulation.

The proposed model employs a classification technique that considers the states to detect the stages of the cybersecurity kill chain. Since the malicious stages (states) are concealed within

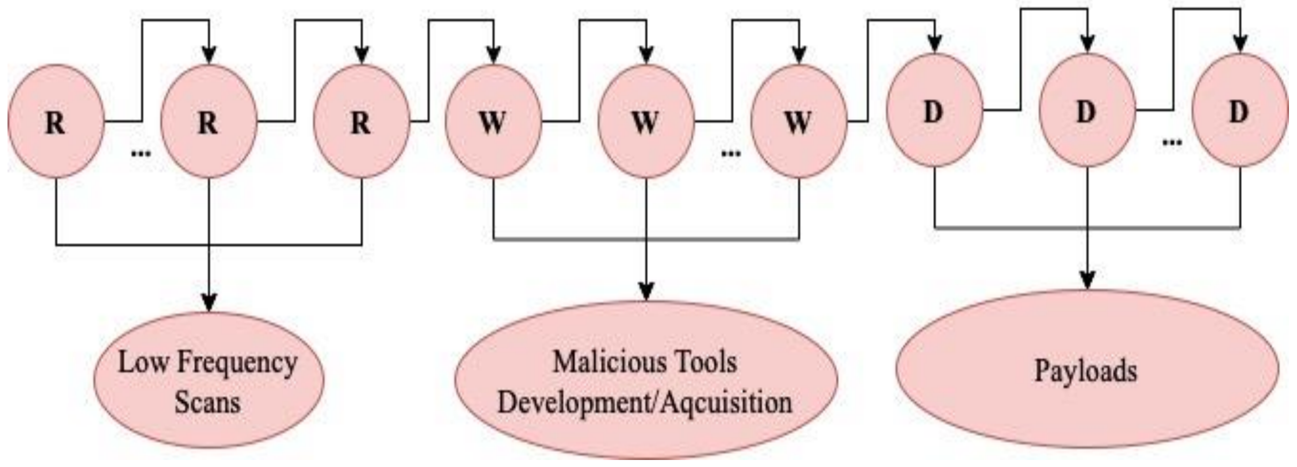


Fig. 3. Attack Stages in the Cybersecurity Kill Chain

the event logs, the proposed model adopts the HMM. In this model, the sequence of transitions between malicious states is concealed, and it is observed through a sequence of emitted observations. The actions observed in event logs serve as emitted observations, while the sequence of concealed states forms a series of attack steps displayed in the upper layer. The lower layer shows the corresponding observations ss depicted in Fig 3. This mapping allows for the representation of each cybersecurity kill chain stage as a distinct state in the HMM. The observations associated with each state provide a way to capture the characteristics or indicators of the attack at that specific stage.

2) LSTM Model: A Recurrent Neural Network (RNN) is a neural network structure designed for tasks involving sequence learning. Within the category of RNNs, LSTM stands out as a specialized variant. The Simple Recurrent Neural Network (RNN) iterates through sequence components, relying solely on the information from the preceding sequence component to handle the current timestep. Hence, encountering the challenge of gradient vanishing. LSTM addresses this limitation by introducing a memory cell state that can retain information over multiple timesteps, ensuring a persistent record of previous information. The cell states (C_{t-1}, C_t) run concurrently with the hidden states (h_{t-1}, h_t), which receive input sequences at each timestep. Within LSTM, three mechanisms are employed to govern the interaction with the cell state, determining which elements of the sequence to retain, forget, or update. The first step in LSTM is to identify the information that is not necessary and decide on what information is going to be thrown away from the cell state. This decision is formed by a sigmoid layer called the 'forget gate layer as expressed in Equation 1.15.

$$f_i^{(t)} = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{1.15}$$

where h_{t-1} denotes the output from the preceding timestamp, x_t denotes the new input data, and b_f denotes the bias. The second phase involves determining the new information to be stored in the cell state. This consists of two components: first, a sigmoid layer known as the input gate layer which determines the values to be updated, and second, a 'tanh layer' that is responsible for generating a vector of new candidate values that can be added to the state as expressed in Equation 1.16 and Equation 1.17.

$$n_i^{(t)} = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{1.16}$$
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{1.17}$$

In the third phase, Equation 1.16 and Equation 1.17 are combined to generate an update to the state as expressed in Equation 1.18. This stage is responsible for updating the previous cell state, C(t-1), into the new cell state. The previous steps have already determined what to do and are now being executed. First, the old state is multiplied, effectively forgetting the elements determined to be forgotten previously. Second, the result is added to the cell state, incorporating the new candidate values scaled by the proportion determined for updating each state value.

$$C_t = f_i^{(t)} + C_{t-1} + n_i^{(t)} * \tilde{C}_t \tag{1.18}$$

The final stage is the output stage, where the output is designed to support the cell state in a refined manner. First, a sigmoid layer is applied, which determines the segments of the cell state that will be included in the output. Second, the cell state undergoes a tanh operation (to constrain values between −1 and 1) and is multiplied by the output of the sigmoid gate. This ensures that only the part decided upon are included in the final output as expressed in Equation 1.19 and Equation 1.20.

$$O_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{1.19}$$
$$h_t = O_t * \tanh(C_t) \tag{1.20}$$

3) **XGBoost Model**: XGBoost was primarily developed for enhanced speed and performance through the utilization of gradient-boosted decision trees. It serves as a tool for machine boosting. XGBoost, short for eXtreme Gradient Boosting, is efficient in maximizing memory and hardware resources for tree boosting algorithms. It offers advantages such as algorithm improvement, model tuning, and deployability in various computing environments. XGBoost excels in executing three major gradient boosting

techniques: Gradient Boosting, Regularized Boosting, and Stochastic Boosting. Additionally, it facilitates the incorporation and fine-tuning of regularization parameters, setting it apart from other algorithms. XGBoost proves highly effective in reducing computing time while ensuring optimal utilization of memory resources. It exhibits Sparse Awareness, meaning it can handle missing values, support the parallel structure in tree construction, and possesses the distinctive ability to perform boosting on additional data that has already been integrated into the trained model.

Additionally, XGBoost has various configurations to reduce overfitting and enhance the overall performance of detecting malicious patterns, resulting in improved accuracy, efficiency, and feasibility. One of these configurations involves adding regularization to the loss function, resulting in the objective function expressed in Equation 1.21.

$$X^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{1.21}$$

where $X^{(t)}$ denotes the objective function, n denotes the number of predicted malicious threats, $L(y_i, \hat{y}_i)$ denotes the training error of the ith sample, and Ω denotes the regularization function.

4) Random Forest Model: The RF model consists of decision trees and serves purposes of both classification and regression problems. When used for classification, the predictions are made by taking a majority vote of the decision tree predictions. On the other hand, for regression, the final result is obtained by averaging the outputs of the individual trees. In the intrusion detection system training phase, a separate training real-time network traffic dataset T_i is generated for each tree using samples from the original network traffic training dataset, T. To create each tree split, a subset of m features is randomly selected, and a measurement is applied to determine the feature that should be used for the split. Due to this randomization, multiple trees are generated, which typically leads to improved prediction performance when combined. RF models offer several advantages over commonly used machine learning methods, including shorter model training time, robustness in handling inconsistent datasets, the ability to incorporate feature importance in classification, and internal metrics for assessing feature impact. RF is trained for intrusion detection by using different feature sets.

F. Hyperparameter Tuning

The model's parameters are learned from the data, and hyperparameter tuning is employed to determine the optimal best-fit for the learning algorithm. It can be conceptualized as a configuration for the model that must be fine-tuned, as the optimal values for one dataset may differ from those of other datasets to attain high precision and accuracy. These values are set before the initiation of the learning process.

The random search cross-validation was employed in this study to identify the best hyperparameters for the attack detection models. This approach combines the advantages of grid search, such as conceptual simplicity, ease of implementation, and trivial parallelism, with the advantage of sampling random points. Unlike grid search, random search does not assign equal importance to every combination of hyperparameters. Due to the significance of multiple hyperparameters in terms of accuracy, speed, and the handling of false negatives and false positives, the following hyperparameters were selected in this study to mitigate overfitting.

1. In the XGBoost algorithm, the feature subsampling for constructing individual trees is controlled through column subsampling by the tree. The learning rate determines the step size in each iteration, and regularization terms were applied to the feature weights (L1 Regularization Term on Weights). The subsample parameter regulates part of the training data used to develop the tree, while the column subsampling by tree parameter controls the selected features for choosing a split candidate.
2. In the HMM algorithm, the utilization of the hidden state approach is essential. The selection of this technique directly impacts the model's efficacy in capturing the underlying patterns within the dataset. Additionally, it dictates the number of states within the HMM, influencing the model's ability to identify malicious patterns in the data. This approach strikes a balance between conceptual simplicity, ease of implementation, and the ability to explore the hyperparameter space efficiently. The focus on mitigating overfitting through the application of the hidden state technique underscores the study's commitment to enhancing the model's generalization to new, unseen attacks, contributing to the overall effectiveness of detecting sophisticated attacks.
3. The random search cross-validation technique was used to determine the learning rate in the LSTM algorithm. Using the random search cross-validation technique to explore learning rates for LSTM models on the CSE-CIC-IDS2017 dataset ensures an optimal configuration that maximizes the model's effectiveness in cyberattack detection. This approach aligns with the primary objective of efficiently dealing with the hyperparameter space to achieve superior model performance.

G. Proposed Ensemble Model

The proposed ensemble model combines three high-performing algorithms to enhance threat detection in SOCs. It aggregates the predictions from each model and uses a stacking mechanism to make the final prediction for the class label of a new instance. An ensemble model that combines HMM, LSTM, and XGBoost for cyberattack detection in SOC infrastructure provides an in-depth solution. The ability of the ensemble to leverage the strengths of different models improves accuracy, and robustness, resulting in a more effective defence against a wide range of cyberattacks.

The function of the HMM algorithm in the ensemble is to detect temporal dependencies and patterns in sequential data. The HMM algorithm contributes to the ensemble by simulating normal network or system activity. The HMM algorithm performs well at detecting deviations and malicious patterns from learned patterns. Its strength lies in capturing the sequential nature of malicious threats, making it useful for understanding attack progressions.

The LSTM algorithm excels at capturing long-term dependencies in sequential data, which is essential for understanding how malicious threats evolve. LSTM contributes to the ensemble by providing insights into subtle and complex patterns. The ability of LSTM to identify malicious patterns over long periods contributes to the ensemble's understanding of the temporal aspects of malicious threats. The algorithm's ability to learn from historical network log datasets makes it advantageous in detecting sophisticated attacks.

XGBoost, as a decision tree ensemble, provides a new perspective by paying attention to feature interactions and classifications. XGBoost contributes to the ensemble by robustly classifying instances based on various features. XGBoost can perform an array of features and capture complex relationships between them. It excels at classifying instances, making it useful for detecting patterns associated with cyberattacks.

The combination of these models increases the likelihood of detecting a wide range of malicious threats, thus enhancing overall detection ability. When compared to individual models, the proposed model is inherently more robust and adaptable. When one model fails to detect a specific type of attack, others may compensate. This ability enables it to adapt to changing threat landscapes, providing a more reliable defence against evolving malicious threats. The proposed ensemble model is less prone to overfitting in the training data due to the combination of HMM, LSTM, and XGBoost. Due to the different strengths of these algorithms, the proposed model can balance biases and reduce the risk of overfitting, resulting in a more generalized and efficient cyberattack detection model in SOC infrastructure.

The ensemble model is developed through a comprehensive set of steps that incorporate the strengths of HMM, LSTM, and XGBoost models, as depicted in Appendix A. The steps for implementing the blending ensemble model include:

1) Feature Extraction: HMM is utilized to create new features from the existing data, generating different components ranging from 1 to 5. These features are then used for subsequent training and testing.
2) Data Splitting**:** The enhanced dataset is split into training and testing sets in an 80:20 ratio, and the training set is further divided into a smaller training set and a validation set.
3) Normalization and Label Encoding: The features in the training, validation, and testing sets are normalized using a StandardScaler to have zero mean and unit variance. The labels (y values) are encoded using a LabelEncoder, converting categorical data into numerical values.
4) Model Training: Two different models are trained: an XGBoost model with varying numbers of trees (100, 200, 300, 400, 500) and an LSTM model with LSTM layers and a Dense output layer. The LSTM model is compiled using the Adam optimizer and the categorical cross-entropy loss function.
5) Model Prediction: Both LSTM and XGBoost models make predictions on the test set. These predictions are then inverse transformed using the LabelEncoder to return them to their original format.
6) Ensemble Learning-Stacking: The script employs a blending ensemble strategy by using predictions from the LSTM and XGBoost models, along with the original features from the validation and test sets. These new datasets are used to train a meta-classifier, which, in this case, is a Random Forest model.
7) Evaluation: The performance of the Meta Classifier is evaluated on the test set, and predictions are obtained for the test set and inversely transformed using the LabelEncoder. The script presents the confusion matrix and classification report for the meta-classifier's predictions.

H. Integration of the SOC Application Programming Interface for Intelligent Security Information

One of the objectives of the study is to improve existing SOCs by integrating a novel stacking ensemble model based on the combination of HMM, LSTM and XGBoost for intrusion detection system (IDS). The SOC analyzes both external and internal threats by monitoring network traffic logs in real-time with the developed ensemble ML IDS. The proposed model was tested for performance evaluation using standard machine learning evaluation metrics after accounting for overfitting. The system was deployed in a real-world environment where it successfully detected web attacks as expected. The testing was conducted on the local host before deploying it in cloud systems.

Digital Ocean cloud service was utilized to create cloud-based scenarios for testing and evaluating the proposed system using real network traffic data. Several other cloud service providers were considered, such as AWS and IBM Cloud; however, Digital Ocean was chosen mostly due to its appealing subscription. Digital Ocean exclusively offers Linux servers as their droplets; therefore, the models, intrusion detection system, and security operations center had to be deployed in a Linux-compatible mode. The system was originally developed on a Windows platform. Most of the libraries and packages that had been previously used had to be adapted to be compatible with Linux.

The latest version of "CICFLOWMETER," a software tool designed for flow-based analysis of network traffic, commonly used in intrusion detection and network security, was leveraged with specific modifications, including feature renaming. The feature names created in version four were different from an older version that had been used to generate the original CICIDS2017 dataset. Missing features in the newest CICIDS2017 flowmeter were created within the intrusion detection system prediction logic using feature/data aggregation methods.

The study utilized the Flask web framework to create a user-friendly interface for integrating the proposed model into the API. Flask is a lightweight, easy-to-use web framework that enables the speedy implementation of both web applications and APIs. It is built on two reliable components, namely, the Werkzeug WSGI toolkit and the Jinja2 template engine.

To implement the API, it is necessary to import the trained models alongside their associated objects, such as the Standard Scaler and Label Encoder. The Python code imports both models and objects, utilizing them to generate predictions based on the incoming network traffic feed. As the API operates, the implemented model and objects are loaded into memory.

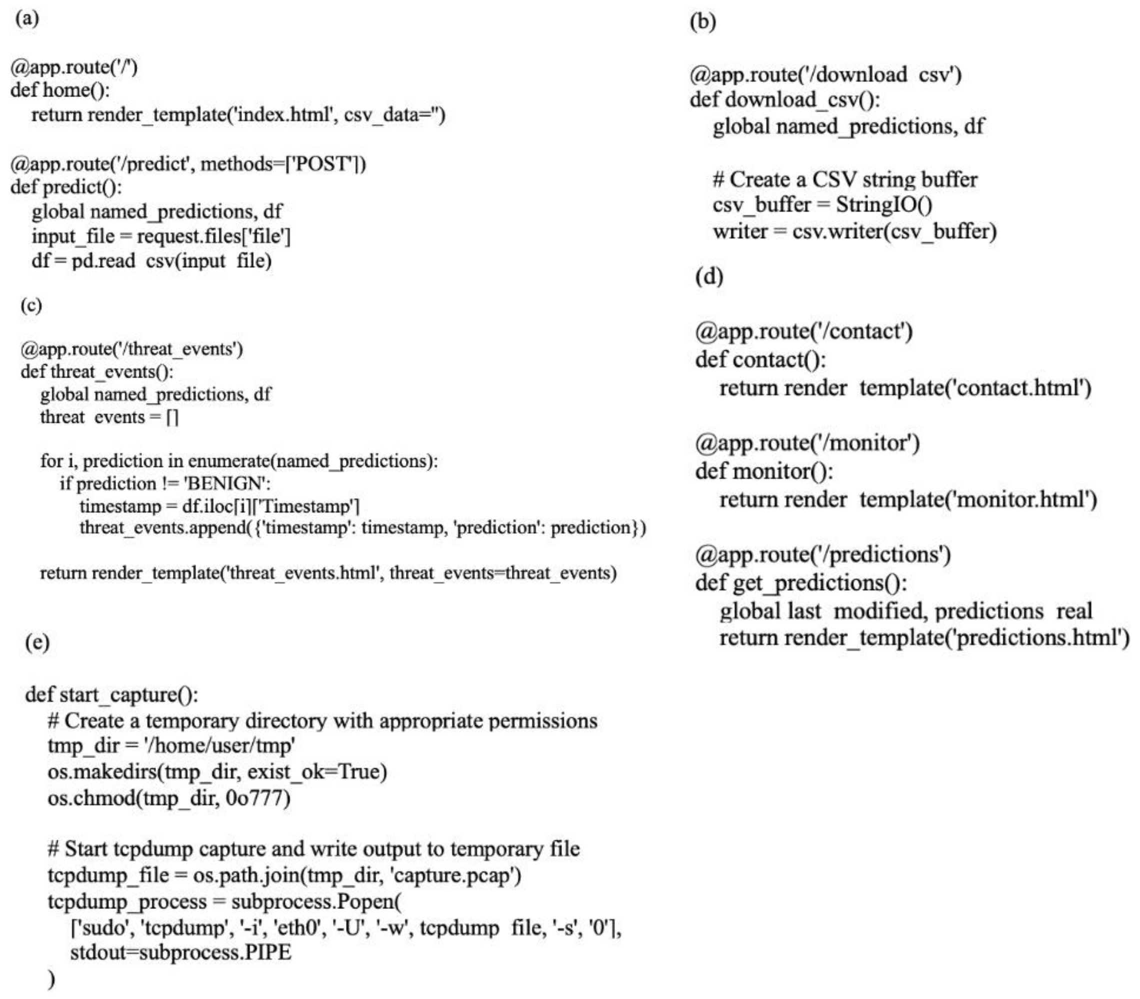
Fig 4a depicts the index route that is responsible for returning the index page of the project. The predicted route depicted in Fig 4b is a POST request endpoint that is used for predicting malicious files, with "files" as the route argument or API payload. The Threat_Events route in Fig 4c is a GET request endpoint that returns predicted threat instances stored in the system, while the predictions route in Fig 4d shows previous predictions stored for various monitoring on the system. Additionally, there are other routes, as depicted in Fig 4e, which represent various modular functionalities for different functions in the system.

To define the various routes for an API, we utilized the Flask web framework, which creates distinct endpoints to handle different tasks. These endpoints, in Figures 4a to 4e, are unique URLs that allow users to access specific functionalities offered by the API.

To create different pathways, the web framework provides routing functionality. In Flask, for instance, one can define routes using the @app.route() decorator which is a special type of function that modifies the behaviour of another function. When user requests are sent to a particular URL, the framework redirects them to an appropriate function corresponding to the endpoint associated with that URL. Afterwards, this function handles and processes the request and issues an output response back to the user.

Dividing an API into different routes results in a system that is modular and well-organized. Each route can be designated to perform a specific function or a group of related functions, facilitating smoother management and maintenance for the API over time. The typical number of API routes ranges from 5 and above. These routes essentially serve as API endpoints for monitoring threat events, sending alerts, retrieving data, connecting to pickled models for prediction, and other modular functionalities used in this implementation.

Moreover, utilizing a web framework with routing functionality provides certain benefits, such as pre-implemented security protocols and error-handling features. These benefits guarantee reliable operation and safeguard against potential threats, enhancing the overall security and stability of the API.


(a)

```
@app.route('/')
def home():
    return render_template('index.html', csv_data='')

@app.route('/predict', methods=['POST'])
def predict():
    global named_predictions, df
    input_file = request.files['file']
    df = pd.read  csv(input  file)
```

(b)

```
@app.route('/download  csv')
def download_csv():
    global named_predictions, df

    # Create a CSV string buffer
    csv_buffer = StringIO()
    writer = csv.writer(csv_buffer)
```

(c)

```
@app.route('/threat_events')
def threat_events():
    global named_predictions, df
    threat  events = []

    for i, prediction in enumerate(named_predictions):
        if prediction != 'BENIGN':
            timestamp = df.iloc[i]['Timestamp']
            threat_events.append({'timestamp': timestamp, 'prediction': prediction})

    return render_template('threat_events.html', threat_events=threat_events)
```

(d)

```
@app.route('/contact')
def contact():
    return render  template('contact.html')

@app.route('/monitor')
def monitor():
    return render  template('monitor.html')

@app.route('/predictions')
def get_predictions():
    global last  modified, predictions  real
    return render_template('predictions.html')
```

(e)

```
def start_capture():
    # Create a temporary directory with appropriate permissions
    tmp_dir = '/home/user/tmp'
    os.makedirs(tmp_dir, exist_ok=True)
    os.chmod(tmp_dir, 0o777)

    # Start tcpdump capture and write output to temporary file
    tcpdump_file = os.path.join(tmp_dir, 'capture.pcap')
    tcpdump_process = subprocess.Popen(
        ['sudo', 'tcpdump', '-i', 'eth0', '-U', '-w', tcpdump  file, '-s', '0'],
        stdout=subprocess.PIPE
    )
```

Fig. 4. API Endpoints

IV. IMPLEMENTATION

A. Model Evaluation Metrics

The study measured precision, recall, and f1 score to evaluate the proposed model's attack detection accuracy. True positive (TP), true negative (TN), false positive (FP), and false negative (FN) values are utilized to determine these parameters. They can be formalized in the context of detecting malicious threats as follows.

1) TP: The total number of correctly predicted malicious instances.
2) TN: The total number of correctly predicted legitimate instances.
3) FP: The total number of incorrect predictions of legitimate instances.
4) FN: The total number of malicious instances that were incorrectly predicted.

Accuracy measures the proportion of accurate predictions made by the model for detecting real-time threats in SOC. This ratio is determined by dividing the total number of correct predictions by the overall number of predictions, which includes both cyber attacks and benign ones. The calculation for accuracy is depicted in Equation 1.22.

Precision measures the proportion of correctly classified instances among the overall predicted instances to be malicious threats, as shown in Equation 1.23. In Equation 1.24 the recall is determined by the proportion of malicious threat instances out of all compromised instances. The F1-score is the harmonic mean of precision and recall as expressed in Equation 1.25.

$$\text{Accuracy} = (TN+TP)/(TN+TP+FN+FP) \quad (1.22)$$
$$\text{Precision} = TP/(TP+FP) \quad (1.23)$$
$$\text{Recall} = TP/(TP+FN) \quad (1.24)$$
$$\text{F1-score} = 2\times(\text{recall}\times\text{precision})/(\text{recall}+\text{precision}) \quad (1.25)$$

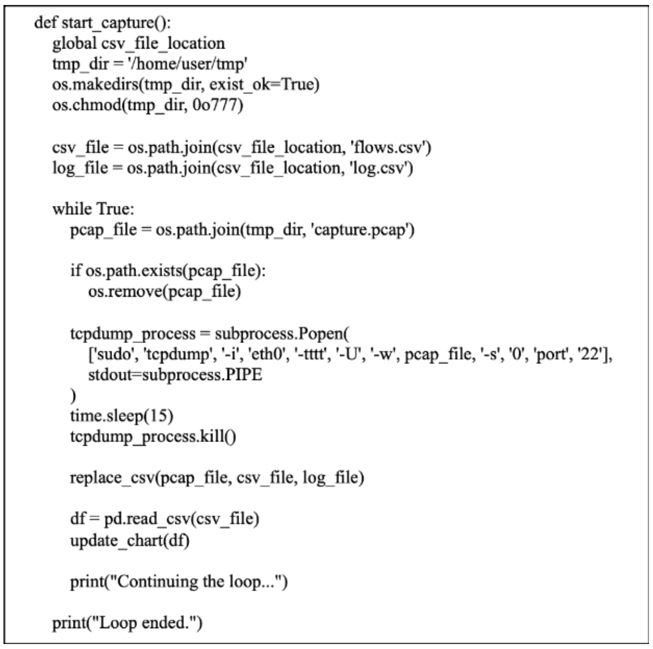
B. Evaluating the Proposed Model with Real-World Network Traffic Data

Evaluating the proposed model with real-world network traffic data was achieved using two approaches. The first approach involves using standard machine learning evaluation metrics as discussed in Section 4.1. The second approach was executed within the API. This approach involves using real-world traffic data to simulate attacks. This means that actual web attacks were conducted on the network where the API was deployed, and the ability to detect these attacks by the integrated ensemble model was tested. This involved no labels as the real, raw network traffic flow logs captured by tcdump and cicflowmeter were passed into the API for detection.

The evaluation metrics in the second approach include:

1) Mean Time to Detect: The Mean Time to Detect (MTTD) is used to measure the performance of a security operations center. MTTD measures the average time taken to detect or predict a security incident also known as a cyberattack. In the developed system Time To Detect or TTD is continuously calculated and monitored during the continuous analysis of the ongoing network traffic. The TTD is determined by the timestamps associated with the start time (when the network flows in question are captured during the tcpdump) and end time (the timestamp when the attack is identified), especially in this case of identifying brute force IPs.

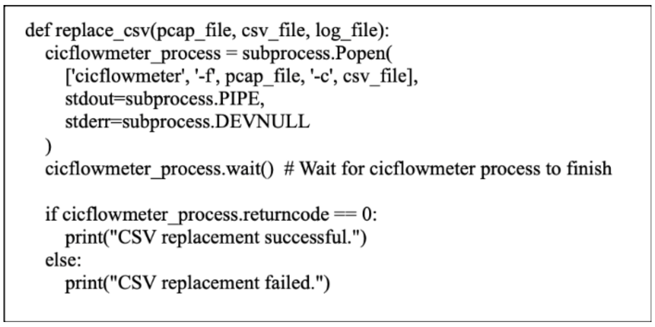
To capture the network traffic continuously the developed system follows an approach of capturing consecutive PCAP files using a loop with each PCAP representing a 15 seconds of network activity. This is done within the Start_Capture function as shown in Fig 5.



```
def start_capture():
    global csv_file_location
    tmp_dir = '/home/user/tmp'
    os.makedirs(tmp_dir, exist_ok=True)
    os.chmod(tmp_dir, 0o777)

    csv_file = os.path.join(csv_file_location, 'flows.csv')
    log_file = os.path.join(csv_file_location, 'log.csv')

    while True:
        pcap_file = os.path.join(tmp_dir, 'capture.pcap')

        if os.path.exists(pcap_file):
            os.remove(pcap_file)

        tcpdump_process = subprocess.Popen(
            ['sudo', 'tcpdump', '-i', 'eth0', '-tttt', '-U', '-w', pcap_file, '-s', '0', 'port', '22'],
            stdout=subprocess.PIPE
        )
        time.sleep(15)
        tcpdump_process.kill()

        replace_csv(pcap_file, csv_file, log_file)

        df = pd.read_csv(csv_file)
        update_chart(df)

        print("Continuing the loop...")

    print("Loop ended.")
```

Fig. 5. Network Capturing

Within the loop, these PCAP files are converted into CSV inside the replace_csv function (using the CICFLOWMETER and are subsequently loaded into the data frame called df as shown in Fig 6. As these data frames are loaded, the update_chart(df) function is called for each data frame. This function is responsible for processing each df dataset using the developed ensemble model, enabling continuous monitoring of the ongoing network traffic.



```
def replace_csv(pcap_file, csv_file, log_file):
    cicflowmeter_process = subprocess.Popen(
        ['cicflowmeter', '-f', pcap_file, '-c', csv_file],
        stdout=subprocess.PIPE,
        stderr=subprocess.DEVNULL
    )
    cicflowmeter_process.wait()  # Wait for cicflowmeter process to finish

    if cicflowmeter_process.returncode == 0:
        print("CSV replacement successful.")
    else:
        print("CSV replacement failed.")
```

Fig. 6. Converting PCAP File to CSV

In the network analysis phase based on the TD and MTTD calculation, if a brute force attack is detected, the TTD is calculated for each run as new "df" datasets are processed by the update_chart() function. As mentioned earlier, the end_time for TTD calculation is recorded by the developed system when it predicts an IP address as a brute force IP as shown in Figure 6. The current timestamp when the detection is done is recorded for this purpose. On the other hand, the start_time for TTD calculation is derived from the timestamp of the brute force IP from the "df" itself. This timestamp is recorded when the PCAP file is generated from the tcpdump network traffic capturing. Furthermore, the earliest occurrence of the corresponding brute force IP is selected for retrieving its timestamp as the start_time as shown in Fig 7.

```
if len(brute_force_ips) > 0:
  brute_force_detected = True

  for ip_address in brute_force_ips:
    ip_flows = df[df['Source IP'] == ip_address]
    start_time = ip_flows['timestamp'].min()
    end_time = datetime.now()
    calculate_ttd(start_time, end_time)
    print("IP Address:", ip_address)
    apply_firewall_rule(ip_address, end_time)
```

Fig. 7. Evaluating Brute Force Attack Based on TTD and MTTD

The TTD is then calculated by subtracting the start_time from the end_time. This TTD calculation provides an estimate of the time taken to predict/detect the corresponding brute force attack. Then the MTTD is calculated as follows. Note that this is done in the front-end programming for displaying it to the user who is using the SOC as expressed in Equation 1.26.

$$\mathrm{MTTD} = \frac{(TTD_1 + TTD_2, \ldots, TTD_n)}{n} \qquad (1.26)$$

First, for n detections n amount of TTDs is calculated and summed together. Then this value is divided by the number n to calculate the MTTD. The formula is applied using a custom function in the Java scripting section in the monitor.html file.

As shown in Fig 8, the MTTD is calculated for consecutive 5 TTD values. The TTD values obtained in different runs can vary depending on the number of records included in the Dataframe. For example, the df Dataframe instances with a higher number of records may result in longer TTD values. Such occasions are during active and aggressive brute force attacks. These may result in comparatively longer TTD values and ultimately longer MTTD values. Conversely, Dataframe instances with fewer records may lead to shorter MTTD values. Moreover, TTD is also highly dependent on its start_time which is recorded during the tcpdump as shown in Fig 9.

To clarify, the TTD value is higher for the malicious IP flows which are timestamped during the starting of the 15-second PCAP capture, than the ones timestamped during the end of the 15 second PCAP capture. However, as long as the tcpdump is the first contact of the attacks with the system, it is still important to use the tcpdump timestamps of the network flows as the start_time for the TTD calculation.

```
if (consecutiveCounter === 5) {
  mttdList.push(mttd);
  if (mttdList.length > 5) {
    mttdList.shift(); // Keep only the last 5 values
  }
}

if (mttdList.length === 5) {
  var mttdMean = calculateMean(mttdList);
  mttdMeanValueSpan.innerText = mttdMean.toFixed(2);
}
```

Fig. 8. MTTD Calculation for Five Consecutive TTD Values

```
function calculateMean(arr) {
  if (arr.length === 0) return 0;
  const sum = arr.reduce((acc, val) => acc + val, 0);
  return sum / arr.length;
}
```

Fig. 9. Custom Mean Function

2) Mean Time to Respond: Like the MTTD, the Mean Time To Respond (MTTR) is another important performance metric that can be used to measure the performance of a security operations center. MTTR simply defines the average time taken to respond to and mitigate a security incident once it has been detected or predicted by a system. In the developed system, first, the TTR or Time To Response is calculated based on two sets of timestamps, the recorded current time set as the detection of a brute force IP being the start_time of the TTR and the recorded current time of an application of a firewall rule being the end_time of the TTR.

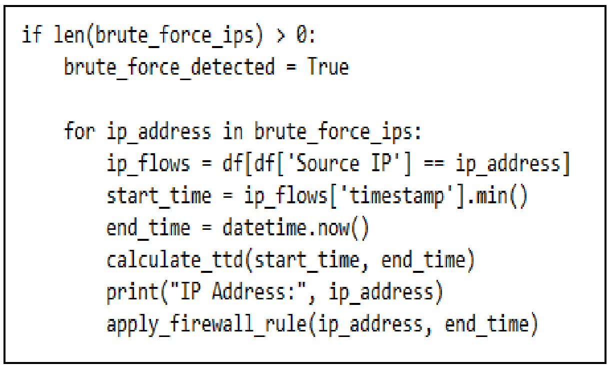
As discussed earlier, the proposed model flags brute force attacks with specific IP addresses associated with these attacks. Following the detection, the system responds to block the brute force attack by applying a new firewall rule to block the corresponding malicious IP address. The firewall rule application is done inside the apply_firewall_rule function using the UFW (Uncomplicated Firewall) which is a netfilter firewall designed with a command-line interface as shown in Fig 10. This response action is critical to prevent any unauthorized access attempts and protect the network.

```
def apply_firewall_rule(ip_address, end_time):
    global ttr_value
    print("apply_firewall_rule() called")

    command = ["ufw", "deny", "from", ip_address]
```

Fig. 10. Firewall Deny Rule

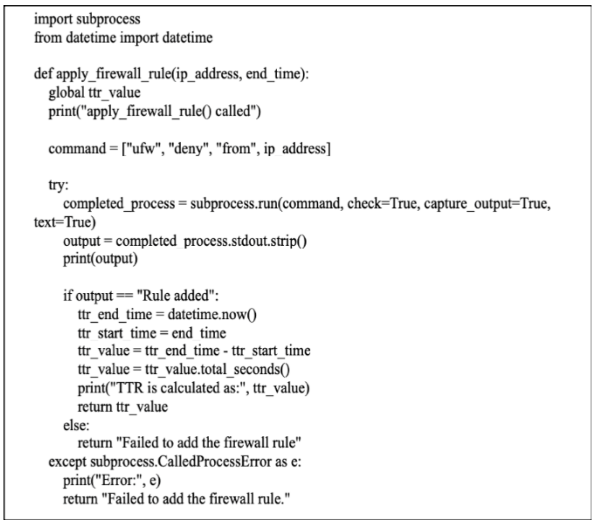
When a brute force IP is detected during the processing of a Dataframe (df), the system records the current timestamp as the end_time of the TTD as shown in Fig 11.

This end_time becomes the start_time of the TTR calculation and is used as a parameter of the apply_firewall_rule function. The end_time for TTR is taken when the firewall rule is successfully applied to block the brute force IP, indicating the completion of the incident response process, though the difference between the start_time of TTR and the end_time of TTR provides an estimate of the time it takes the developed system to respond and implement the necessary mitigation measures for a detected brute force attack as shown in Fig 12.



```
if len(brute_force_ips) > 0:
    brute_force_detected = True

    for ip_address in brute_force_ips:
        ip_flows = df[df['Source IP'] == ip_address]
        start_time = ip_flows['timestamp'].min()
        end_time = datetime.now()
        calculate_ttd(start_time, end_time)
        print("IP Address:", ip_address)
        apply_firewall_rule(ip_address, end_time)
```

Fig. 11. Brute Force IP Detection Based on MTTR Calculation



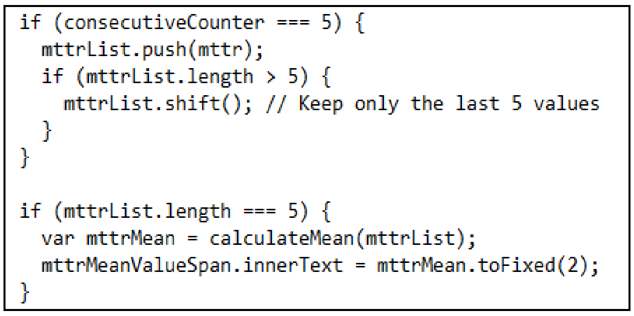
```
import subprocess
from datetime import datetime

def apply_firewall_rule(ip_address, end_time):
  global ttr_value
  print("apply_firewall_rule() called")

  command = ["ufw", "deny", "from", ip_address]

  try:
    completed_process = subprocess.run(command, check=True, capture_output=True,
text=True)
    output = completed_process.stdout.strip()
    print(output)

    if output == "Rule added":
      ttr_end_time = datetime.now()
      ttr_start_time = end_time
      ttr_value = ttr_end_time - ttr_start_time
      ttr_value = ttr_value.total_seconds()
      print("TTR is calculated as:", ttr_value)
      return ttr_value
    else:
      return "Failed to add the firewall rule"
  except subprocess.CalledProcessError as e:
    print("Error:", e)
    return "Failed to add the firewall rule."
```

Fig. 12. Firewall Deny Rule and Calculating TTR

Like in the MTTD, MTTR is calculated by calculating the average time taken for multiple TTR actions in the front-end as expressed in Equation 1.27.

$$\mathrm{MTTR} = \frac{(TTR_1 + TTR_2, \ldots, TTR_n)}{n} \qquad (1.27)$$

First, for n firewall rule applications n amount of TTRs are calculated and summed together. Then this summed value is divided by the number n to calculate the MTTR as shown in Fig 13.



```
if (consecutiveCounter === 5) {
  mttrList.push(mttr);
  if (mttrList.length > 5) {
    mttrList.shift(); // Keep only the last 5 values
  }
}

if (mttrList.length === 5) {
  var mttrMean = calculateMean(mttrList);
  mttrMeanValueSpan.innerText = mttrMean.toFixed(2);
}
```

Fig. 13. MTTR Calculation for Consecutive Five TTR Values

V. Result and Discussion

This section presents the experimental results of the proposed ensemble intrusion detection model, which incorporates stacking, bagging, and majority voting models. The chapter also discusses the experimental results obtained from undersampled and minority-sampled datasets using ensemble model techniques. A comparison was executed to determine the most efficient model for real-time detection of malicious threats in a network.

A. Result of the Proposed Model for the Multi-Class Classification

The confusion matrix in Fig 14 shows that 1018 instances belong to the Benign class, and the model correctly predicted all of them as Benign. There are no instances that were incorrectly predicted as Class Benign. 1017 instances belong to Class Brute Force, and the model correctly predicted 928 of them as Class Brute Force. However, 5 instances were predicted as Class SQL Injection and 84 were predicted as Class XSS when they belong to Class Brute Force. 1018 instances belong to Class SQL Injection, and the model correctly predicted all of them as Class SQL Injection. There are no instances that were incorrectly predicted as Class SQL Injection. 1017 instances belong to Class XSS, and the model correctly predicted 978 as Class XSS. However, there is 1 instance that was predicted as Class Benign, 36 were predicted as Class Brute Force, and 2 were predicted as Class SQL Injection when they belong to Class XSS. The classification report for the stacking blending ensemble is shown in Table II.

Table II
CLASSIFICATION REPORT OF THE PROPOSED STACKING BLENDING MODEL FOR THE MULTI-CLASS CLASSIFICATION

	precision	recall	f1-score	support
BENIGN	1.00	1.00	1.00	1018
Web Attack ? Brute Force	0.96	0.91	0.94	1017
Web Attack ? Sql Injection	1.00	1.00	1.00	1018
Web Attack ? XSS	0.92	0.96	0.94	1017
accuracy			0.97	4070
macro avg	0.97	0.97	0.97	4070
weighted avg	0.97	0.97	0.97	4070

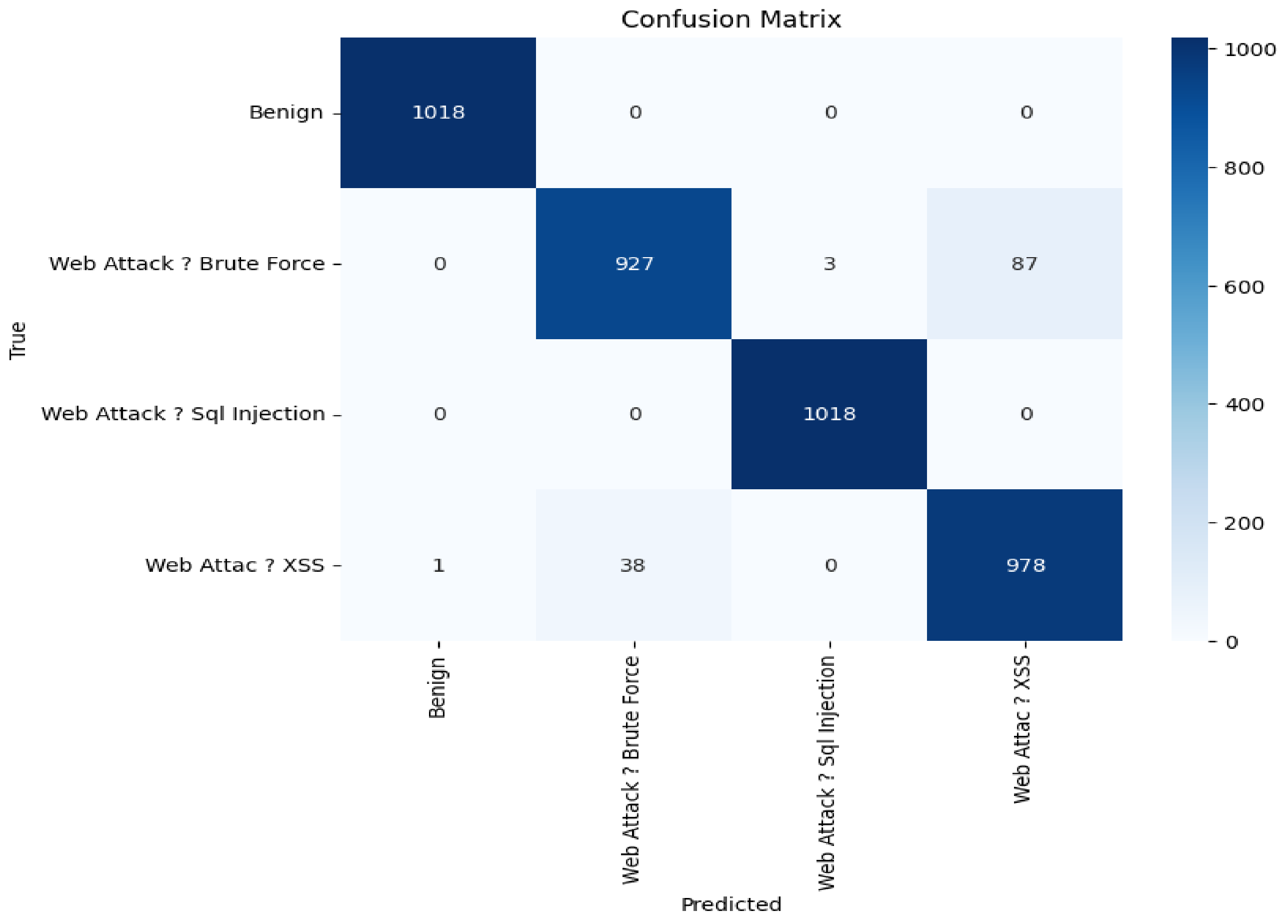


Fig. 14. Confusion Matrix of the Proposed Stacking Blending Ensemble Model for the Multi-Class Classification

The class-wise evaluations and the overall evaluation include:

1) Class-Wise Evaluation: In the test set, there are 1018 instances of the first class, which is BENIGN. The model performed well in this class with precision, recall, and F1 scores of 1.00, indicating that all instances of this class were correctly predicted by the model. The second class, called Brute Force, contains 1017 samples in the test set. According to the evaluation metrics, this model classifies instances belonging to this category with high precision, recall, and F1 scores of 0.96, 0.91, and 0.94, respectively. This indicates that the model correctly identifies 96% of the instances within this class and accurately categorizes approximately 91% of them. The test set contains a third-class of 1018 instances. The precision, recall, and F1 scores of this class are all perfect with a value of 1.00. This indicates that the model accurately predicted all the instances involved in this particular class. The test set has 1017 instances in the XSS class. It achieved precision, recall, and F1 scores of 0.92, 0.96, and 0.94, respectively. Thus, the model accurately predicted 92% of the examples belonging to this class and recognized precisely 96% of the instances that belong to it as expected.

2) Overall Evaluation: Precision is the proportion of true and positive predictions among all positive predictions. A high precision score indicates that the model is making fewer false positive predictions. The report shows that for all classes, the model has a precision score ranging from 0.92 to 1.00, indicating high precision overall. Recall measures how well a model predicts the positive instances in a dataset. A high score indicates that the model accurately identifies a large proportion of actual positives. In this report, recall scores between 0.91 and 1.00 show that the model has high recall across all classes. F1-score: The F1-score is a statistical tool used to balance precision and recall by taking the harmonic mean between them. Scores obtained from 0.94 to 1.00 in this report suggest excellent performance across all classes for the model employed here. Accuracy is the proportion of accurately classified instances among all instances. According to this report, the model's overall accuracy stands at 0.97, indicating that it is performing well. The macro average is a performance metric that averages the precision, recall, and F1-score of each class. It provides an overall measure of the model's performance without considering the imbalance in instances across different classes. This report indicates good overall performance with a macro average of 0.97 for all metrics used in the evaluation. The report calculates the weighted average by combining metrics such as precision, recall, and F1-score for every class based on the number of instances in each category. This measure indicates the model's overall performance while accounting for unequal instance

distribution across different classes. The resulting weighted average of 0.97 implies a good overall performance of the model.

B. Result of the Proposed Model for the Binary Class Classification

For the malicious threat (class 0), 99% of instances predicted as malicious threats are indeed malicious. This indicates a low rate of false positives. The model also captures 99% of all actual instances of malicious threats (class 0), demonstrating high sensitivity. The model achieved an F1-score of 99%, suggesting an outstanding overall performance in classifying malicious threats (class 0) as depicted in Table III

For the legitimate traffic (class 1), among the instances predicted as legitimate traffic, 99% are truly legitimate. This implies an extremely low rate of false negatives. The model identifies 99% of all actual instances of legitimate traffic (class 1), indicating high sensitivity. The model achieved an F1-score of 99%, signifying an effective balance between precision and recall for classifying legitimate traffic (class 1).

TABLE III
CLASSIFICATION REPORT OF THE PROPOSED STACKING BLENDING MODEL FOR THE BINARY-CLASS CLASSIFICATION

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1552
1	0.99	0.99	0.99	1501
accuracy			0.99	3053
macro avg	0.99	0.99	0.99	3053
weighted avg	0.99	0.99	0.99	3053

The result in Figure 15 depicts the Area Under the Receiver Operating Characteristic Curve (AUC_ROC score). The objective of the AUC-ROC score is to detect and respond to cyberattacks efficiently. Fig 15 shows that the model achieved an AUC-ROC score of 0.99. This result indicates that the model is accurate and capable of classifying benign network activities from malicious patterns associated with cyberattacks. The model's high true positive rate (Sensitivity) combined with an AUC of 0.99 indicates that it is capable of detecting malicious patterns effectively. This is critical for efficient detecting and responding to security incidents, thereby reducing potential damage from cyber threats.

A model with a high AUC-ROC score generally has an optimized detection threshold. Therefore, the model is tuned to achieve a balance between detecting as many attacks as possible (high recall) and minimizing false alarm rate (high precision). As a result, SOC teams can rely on the proposed model for continuous security monitoring, as well as have confidence in the model's ability to accurately flag suspicious activities, enabling SOC analysts to focus on genuine threats and take prompt action to protect the organization's resources.

Fig 16 depicts the outcome of the cross-validation. The cross-validation results show consistently high accuracy across five-folds, with an average accuracy of around 0.999. This result shows that the model performs consistently well across various subsets of the training data. Consistency is important in machine learning because it indicates that the model's performance is stable and not heavily influenced by specific data points or subsets. An average accuracy of 0.999 is high, indicating that the model is efficient at correctly classifying malicious network patterns in real-time. This degree of accuracy is frequently indicative of a well-trained model with outstanding predictive abilities. As a result, there is high confidence in the model's ability to generalize well to new data points (unseen attacks). This is essential in real-world applications where the model must perform well with new, previously unseen attacks

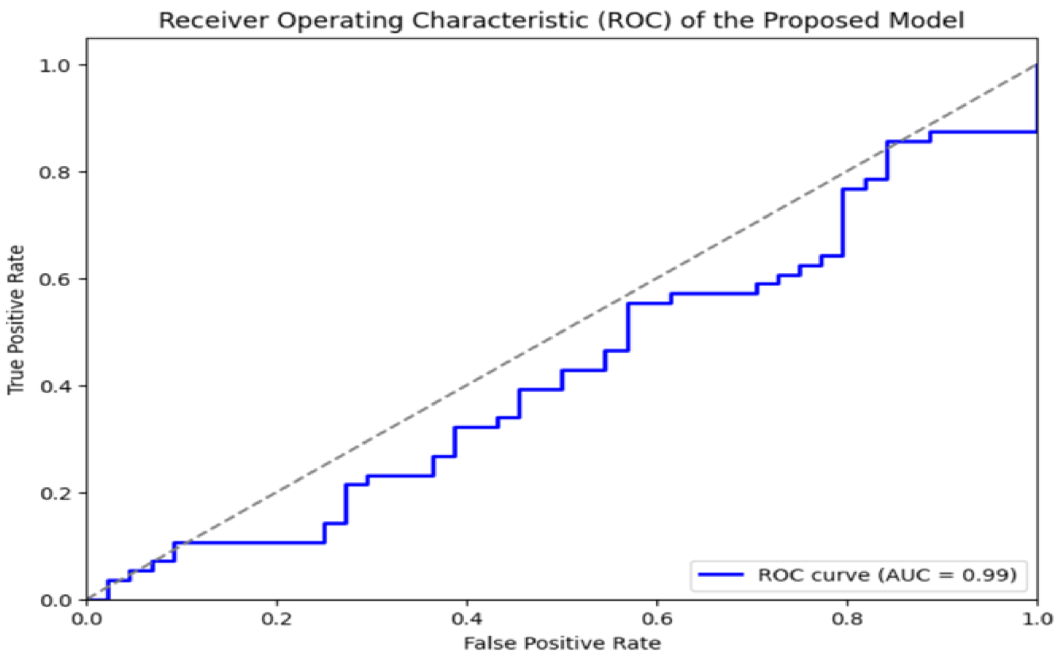


Fig. 15. AUC-ROC Score of the Proposed Model

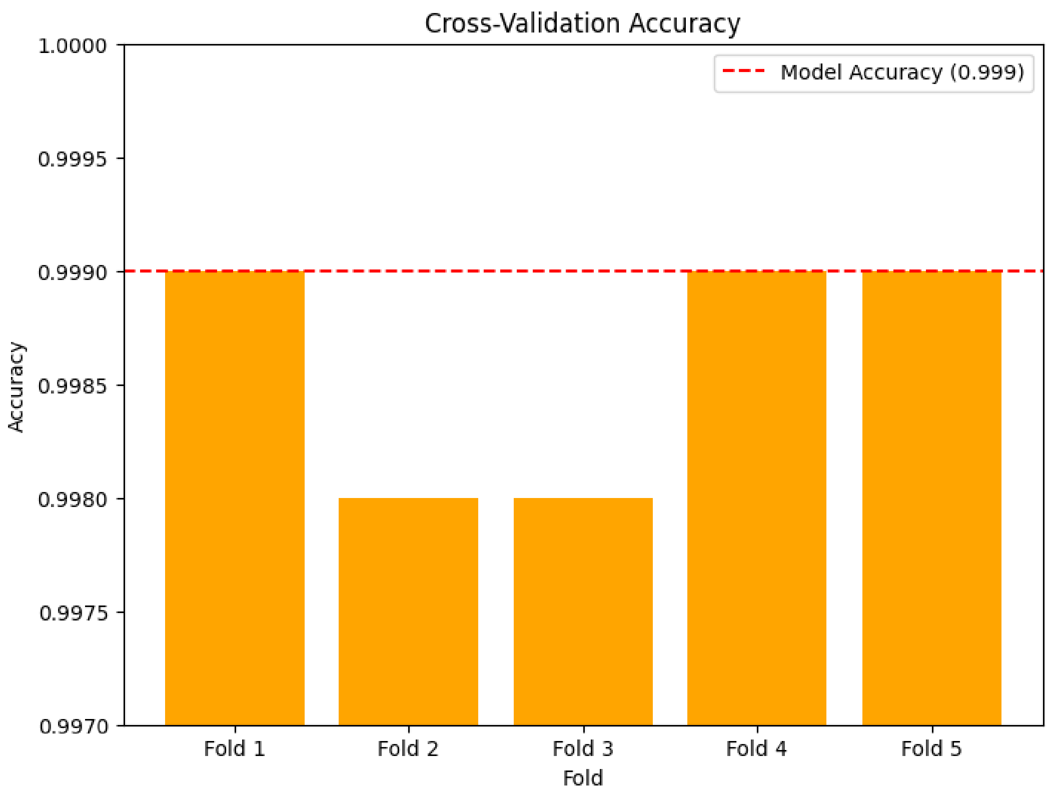


Fig. 16. Cross-Validation of the Proposed Model

C. Benchmarking with Existing Models in Literature

A comparison of the proposed model to those reviewed in the literature reveals that it outperforms all other AI models for binary classifications, as shown in Table IV because the datasets used in this work have not been explored in the literature, the proposed model was trained on the same dataset used by the previous researcher to ensure a fair comparison.

TABLE IV
BENCHMARKING OF THE PROPOSED MODEL WITH EXISTING MODELS

Authors	**Models**	**Accuracy**	**Precision**	**Recall**	**F1-Score**
[59]	AE	98.5%	98%	98%	98%
[59]	KNN	99%	98%	99%	98%
[59]	LDA	96%	96%	96%	96%
[59]	LSTM	99.2%	99%	99%	99%
[58]	L-SVM	99%	98%	98%	98%
[59]	MLP	94.6%	94%	94%	96%
[59]	QDA	99%	99%	98%	98%
[59]	Q-SVM	64%	64%	64%	63%
Ours	Proposed Model	99%	99%	99%	99%

D. Practicability and Generalization to Real-World Environments

The effectiveness of any machine learning model, particularly in cybersecurity, hinges not just on its performance metrics in controlled environments but also on its practicability and generalization to real-world scenarios. This is particularly true for AI-driven cybersecurity models. This section evaluates the proposed model's ability to meet these requirements and compares it to state-of-the-art approaches that make use of the KDD CUP dataset, which is an established benchmark for intrusion detection.

The KDD CUP dataset has been extensively utilized to evaluate machine learning models designed for detecting various cyber threats, including DDoS attacks. For instance, [60] combined logistic regression and support vector machine (SVM) as an ensemble learning technique to detect DDoS attacks in real-time. This study achieved a remarkable 99.2% accuracy in detecting DDoS attacks, setting a benchmark in the field due to its high accuracy and strong generalization capabilities. However, when the proposed stacking blending model was applied to the same KDD CUP dataset, it achieved an improved accuracy of 99.6%. This represents a significant improvement, with an absolute increase of 0.4% in accuracy. While this improvement might seem marginal, even small gains in detection accuracy can have a significant impact on cybersecurity, particularly when handling large volumes of network traffic where false positives or missed detections could lead to severe consequences. This result indicates that the model's strong generalization across different environments enables it to be adapted to various network configurations and threat models with minimal retraining. This adaptability makes it a valuable tool for organizations aiming to strengthen their cybersecurity defences. The result also suggests that the proposed model will not only perform well in controlled test environments but also has the potential for deployment in practical settings where the unpredictability and diversity of threats are significantly higher.

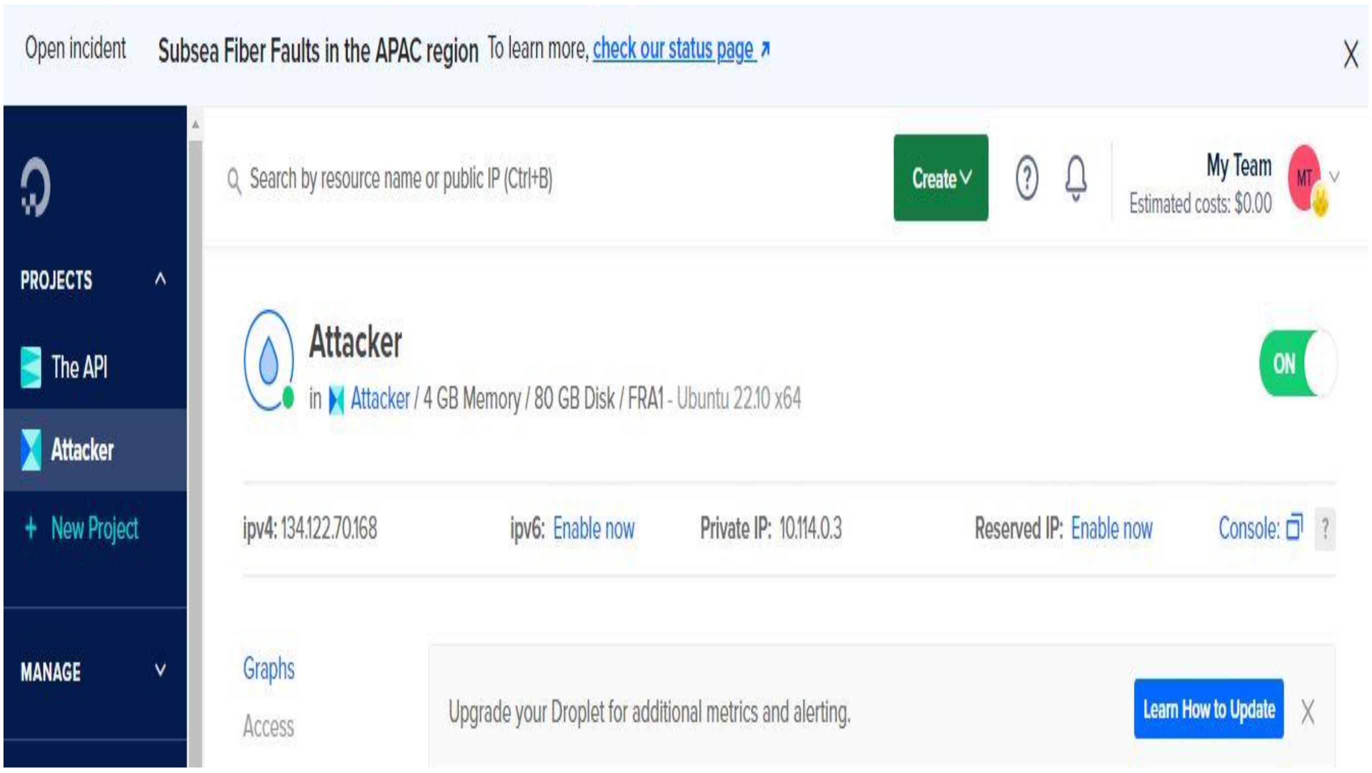
This means that the proposed model's consistently high detection rates and minimal false positives are essential for reducing the operational burden on SOC analysts. In addition, the model's ability to reduce the number of false alarms allows security personnel to concentrate on genuine threats. Hence, enhancing the overall efficiency of the SOC infrastructure.

E. Environmental Setup of the Testing and Evaluation in Controlled and Real-World Environments

This section discusses the evaluation and testing procedures to assess the effectiveness and reliability of the developed Intrusion Detection System (IDS) and its associated API. Two key aspects are explored: evaluation in a production environment with real network traffic data and control testing with simulated attacks.

1) Evaluation in the Production Environment (No Labels): The deployed model in the SOC environment was evaluated using real network traffic data converted to a continuously updating CSV dataset using tcpdump and CICFlowMeter. The ongoing network traffic in the network was captured in real-time and fed into the ensemble model.
2) Creating A Linux Attacker PC in Digital Ocean: An Ubuntu-based attacker PC was installed to validate the API using actual network traffic to replicate and simulate attacks, such as Brute Force attacks on the web server, as shown in Fig 17.
3) Attacking the API with Brute Force Attack (Real-Time Monitoring): Fig 18 depicts the execution of a brute force attack using the Hydra technique. The IDS/API detects brute force attacks in "real-time" (with about a 10-second lag; this lag is for the prediction logic to take place for the updated CSV records upon the fetch calls from the front end, which is inevitable), as shown in Fig 19.

A brute force attack involves attempting various username and password combinations until the attacker gains entry into the targeted system or application. The attacker uses automation tools like Hydra to quickly guess thousands of login credentials from their list, testing the Security Operation Center's Intrusion Detection System (IDS) effectiveness. Often, an external source, like a password.txt file, contains credential pairs that are tried out by the tool on multiple systems in rapid succession. Hydra is a widely used software for brute force attacks. This malicious tactic involves repeatedly guessing the username and password combinations until the correct match is found. Hydra automates this process by rapidly trying different username and password combo variations, significantly reducing the time it would take to discover the right credentials.

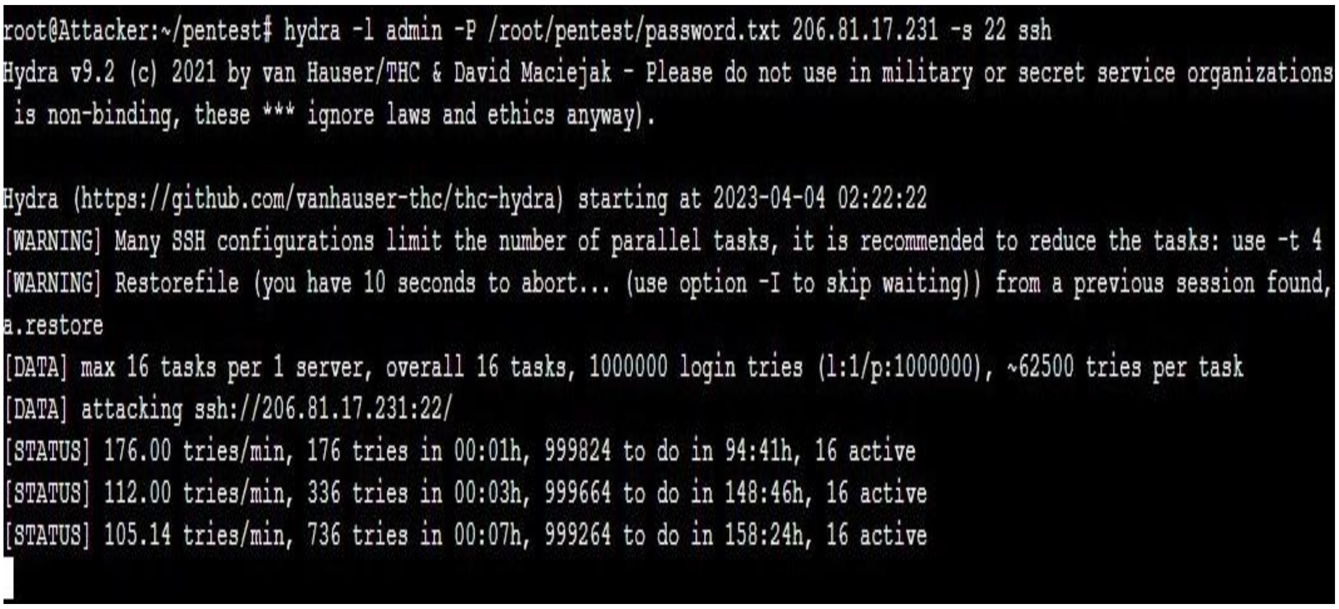

Open incident Subsea Fiber Faults in the APAC region To learn more, check our status page

Attacker
in Attacker / 4 GB Memory / 80 GB Disk / FRA1 - Ubuntu 22.10 x64

ipv4: 134.122.70.168 ipv6: Enable now Private IP: 10.114.0.3 Reserved IP: Enable now Console:

Upgrade your Droplet for additional metrics and alerting.

Fig. 17. The Linux Attacker PC



```
root@Attacker:~/pentest# hydra -l admin -P /root/pentest/password.txt 206.81.17.231 -s 22 ssh
Hydra v9.2 (c) 2021 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations
 is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-04-04 02:22:22
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found,
a.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 1000000 login tries (l:1/p:1000000), ~62500 tries per task
[DATA] attacking ssh://206.81.17.231:22/
[STATUS] 176.00 tries/min, 176 tries in 00:01h, 999824 to do in 94:41h, 16 active
[STATUS] 112.00 tries/min, 336 tries in 00:03h, 999664 to do in 148:46h, 16 active
[STATUS] 105.14 tries/min, 736 tries in 00:07h, 999264 to do in 158:24h, 16 active
```

Fig. 18: Conducting Brute Force Attack

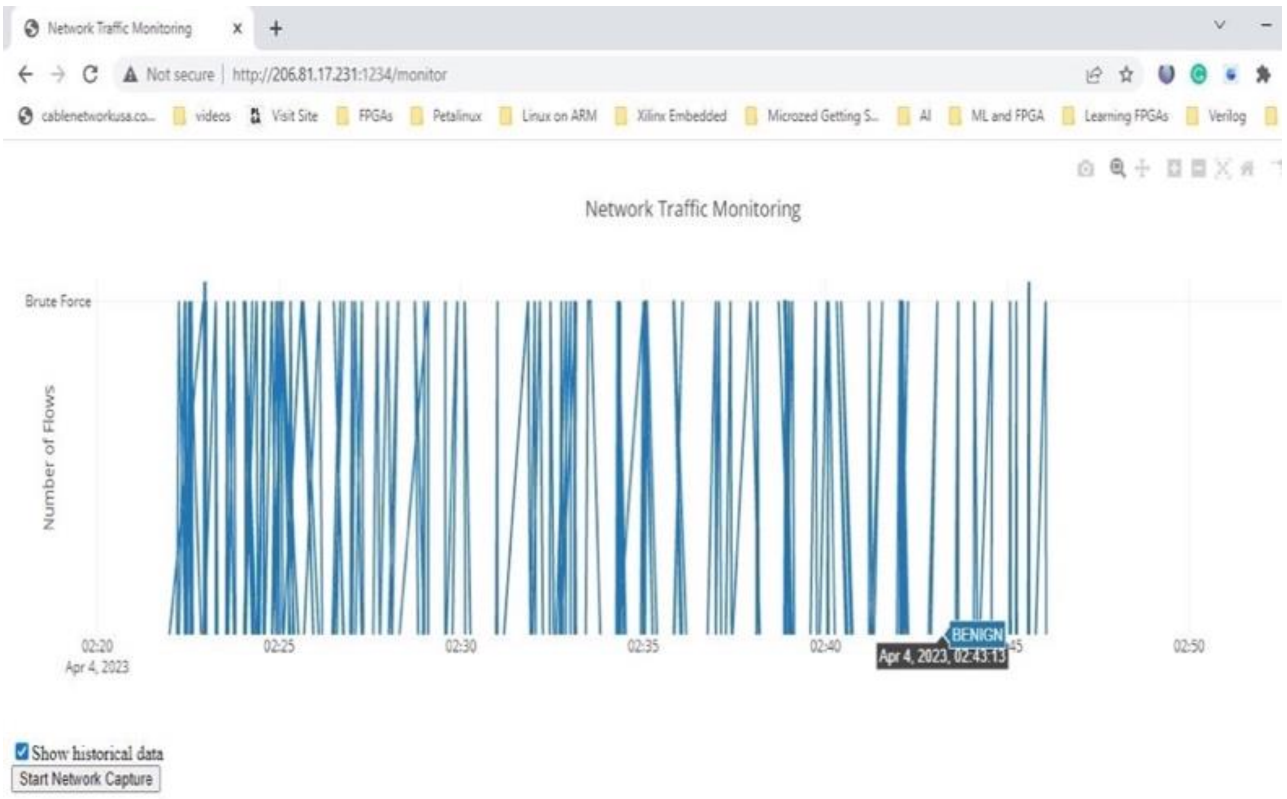


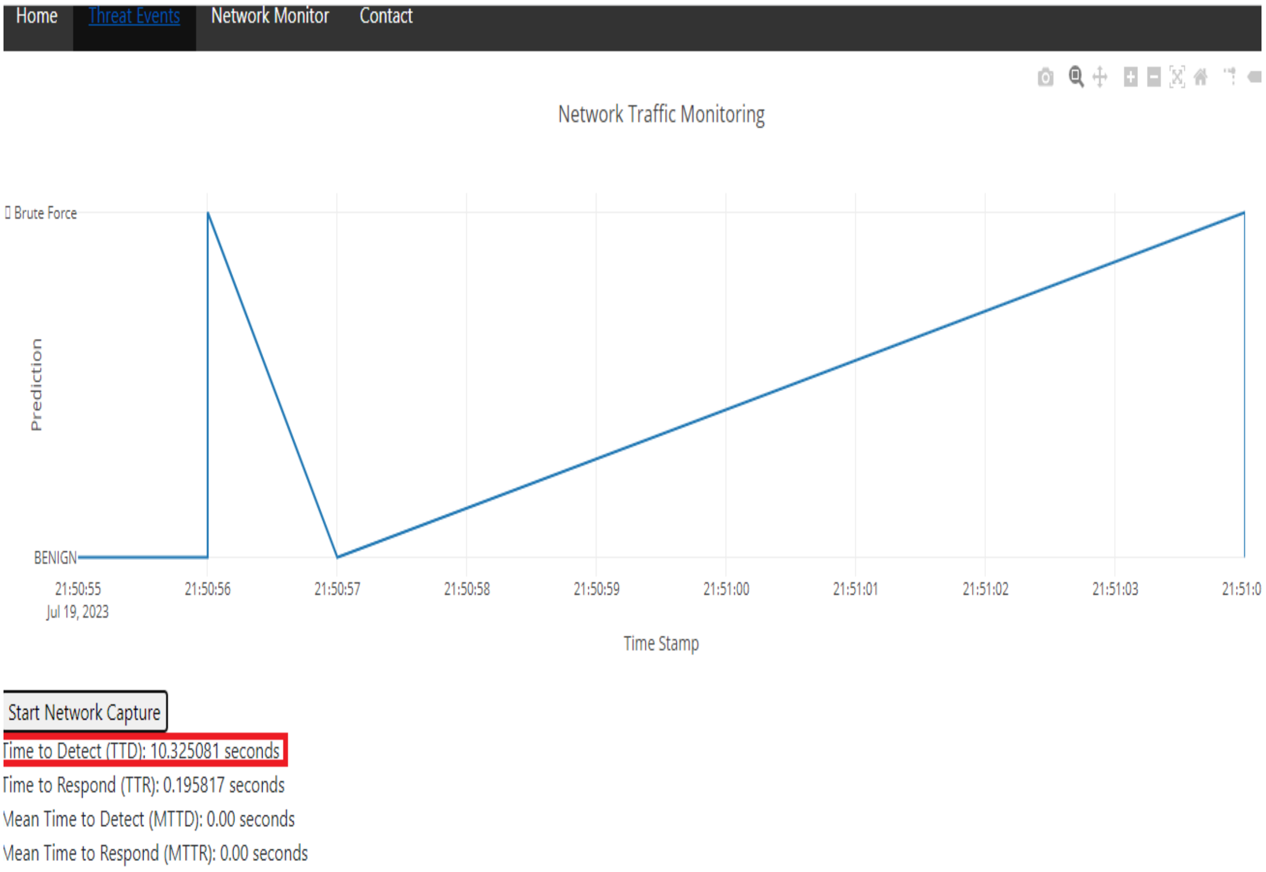
Fig. 19. Real-Time Detection for Brute Force Attacks

F. Results of System Effectiveness for Mean Time to Detect (MTTD) and Mean Time to Respond (MTTR)

This section presents the outcomes of evaluating the system's efficiency in promptly detecting security incidents, measured by Mean Time to Detect (MTTD), and responding effectively, assessed through Mean Time to Respond (MTTR). The MTTD metric reflects the system's ability to identify threats swiftly, particularly in the case of brute force attacks, while a lower MTTR indicates the system's agility in implementing mitigation measures.

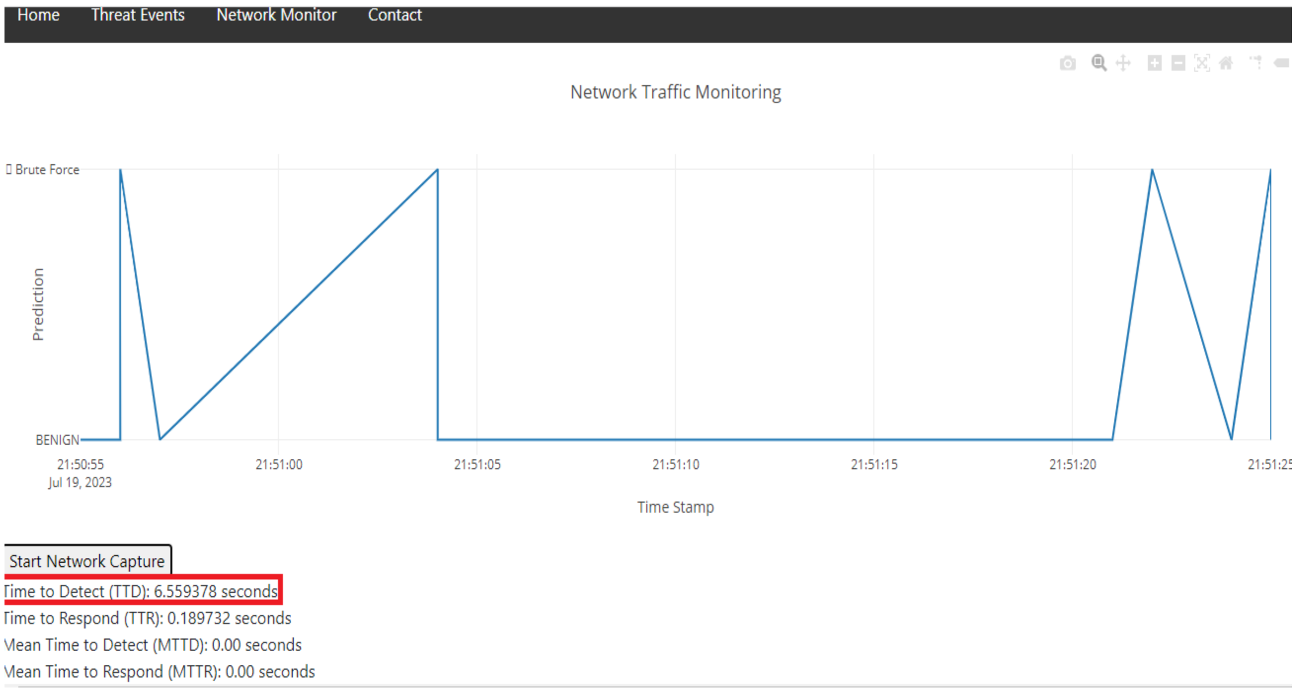
1) Results of System Effectiveness for Mean Time to Detect (MTTD): Mean Time to Detect (MTTD) is a crucial metric that measures the system's ability to promptly identify and respond to security incidents, such as brute force attacks. We gained valuable insights into the ensemble model's performance by continuously calculating Time-to-Detect (TTD) values and obtaining MTTD values. Fig 20 and Fig 21 display TTD values captured from the front end. Fig 22 depicts a TTD value capture within the backend

console. These values represent the time taken to detect the brute-force IPs after the attacks were initiated. Using these TTD values, we computed the MTTD, as depicted in Fig 23. A lower MTTD indicates that the system predicts and responds to brute force IPs more promptly, minimizing the potential damage caused by malicious actors. By blocking the malicious IPs upon detection, the system prevents further compromise of the network and reinforces the SOC's defensive capabilities.



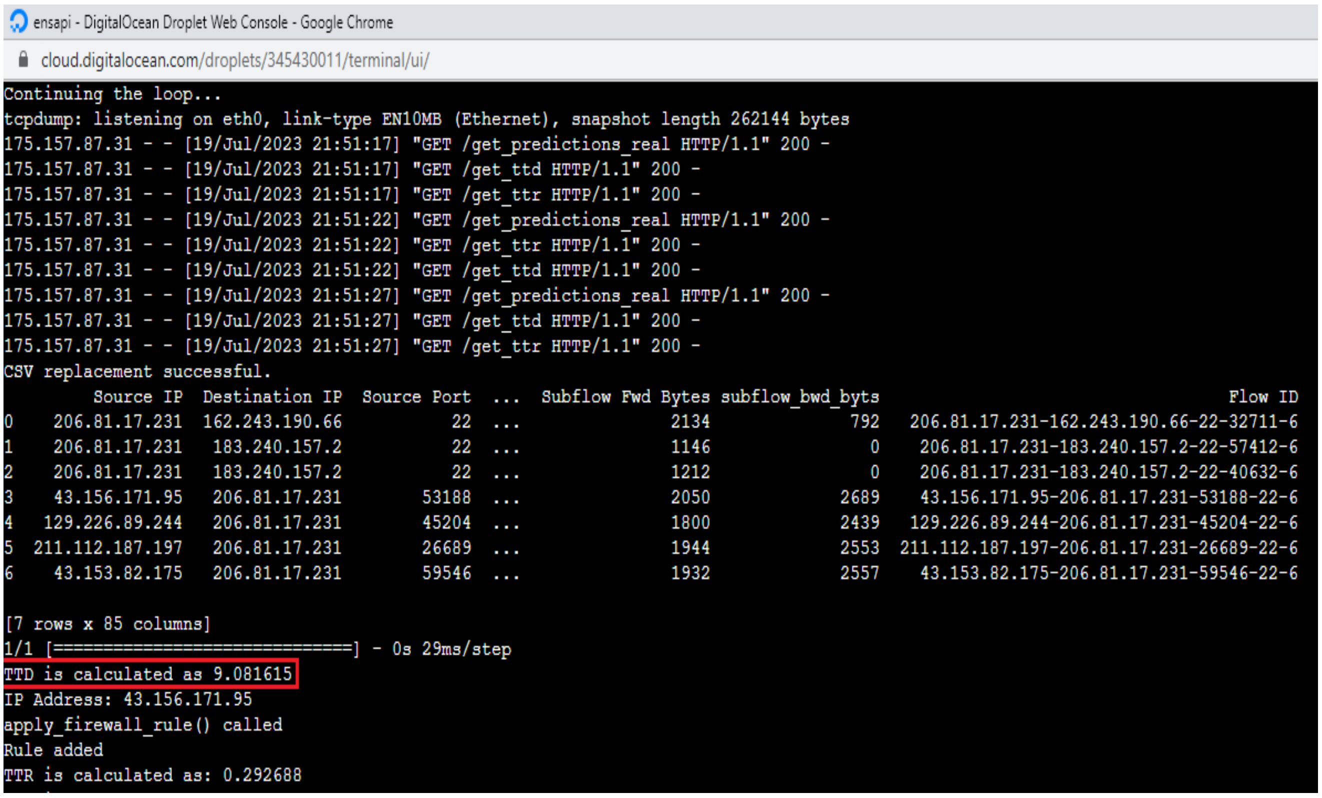
Time to Detect (TTD): 10.325081 seconds
Time to Respond (TTR): 0.195817 seconds
Mean Time to Detect (MTTD): 0.00 seconds
Mean Time to Respond (MTTR): 0.00 seconds

Fig. 20. A TTD Value Captured from the Frontend



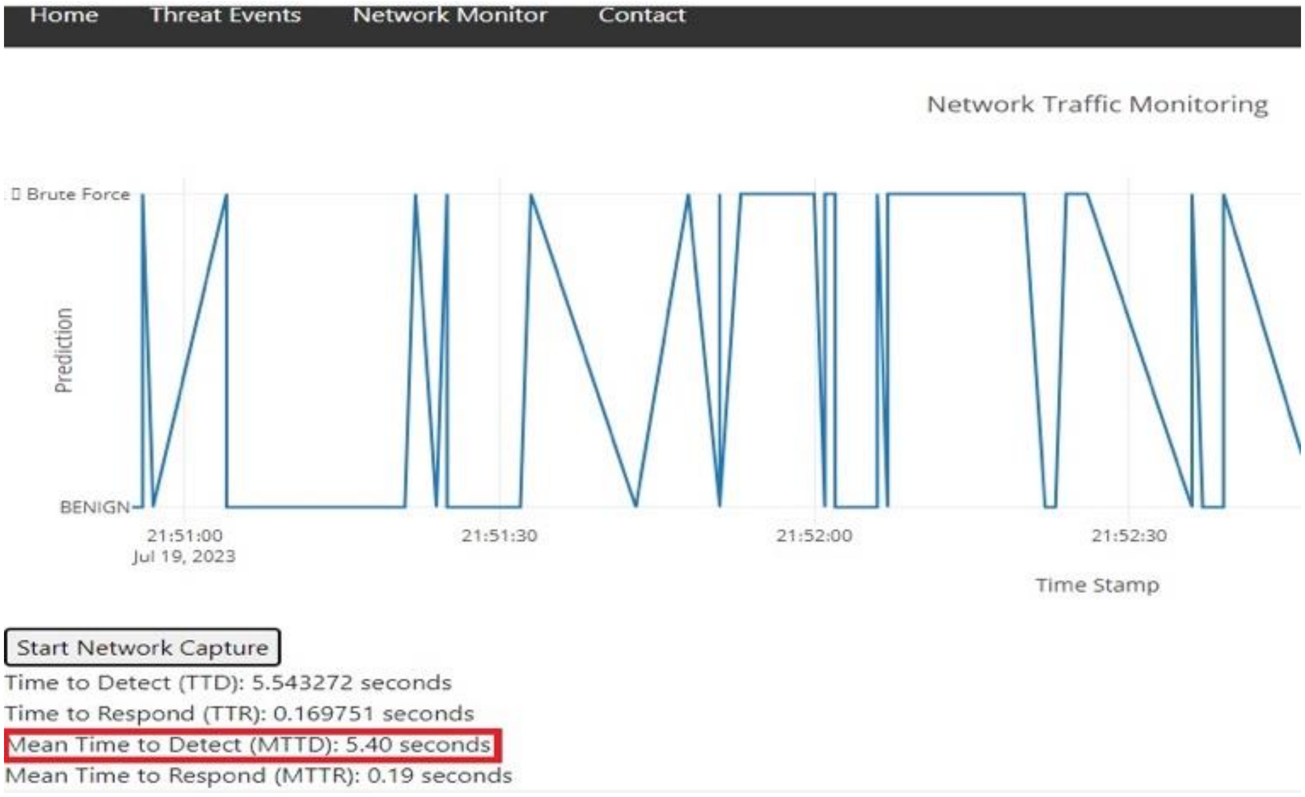
Time to Detect (TTD): 6.559378 seconds
Time to Respond (TTR): 0.189732 seconds
Mean Time to Detect (MTTD): 0.00 seconds
Mean Time to Respond (MTTR): 0.00 seconds

Fig. 21. A TTD Value Captured from the Frontend


ensapi - DigitalOcean Droplet Web Console - Google Chrome
cloud.digitalocean.com/droplets/345430011/terminal/ui/

```
Continuing the loop...
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
175.157.87.31 - - [19/Jul/2023 21:51:17] "GET /get_predictions_real HTTP/1.1" 200 -
175.157.87.31 - - [19/Jul/2023 21:51:17] "GET /get_ttd HTTP/1.1" 200 -
175.157.87.31 - - [19/Jul/2023 21:51:17] "GET /get_ttr HTTP/1.1" 200 -
175.157.87.31 - - [19/Jul/2023 21:51:22] "GET /get_predictions_real HTTP/1.1" 200 -
175.157.87.31 - - [19/Jul/2023 21:51:22] "GET /get_ttr HTTP/1.1" 200 -
175.157.87.31 - - [19/Jul/2023 21:51:22] "GET /get_ttd HTTP/1.1" 200 -
175.157.87.31 - - [19/Jul/2023 21:51:27] "GET /get_predictions_real HTTP/1.1" 200 -
175.157.87.31 - - [19/Jul/2023 21:51:27] "GET /get_ttd HTTP/1.1" 200 -
175.157.87.31 - - [19/Jul/2023 21:51:27] "GET /get_ttr HTTP/1.1" 200 -
CSV replacement successful.
         Source IP  Destination IP  Source Port  ...  Subflow Fwd Bytes subflow_bwd_byts                              Flow ID
0    206.81.17.231  162.243.190.66           22  ...               2134              792  206.81.17.231-162.243.190.66-22-32711-6
1    206.81.17.231   183.240.157.2           22  ...               1146                0   206.81.17.231-183.240.157.2-22-57412-6
2    206.81.17.231   183.240.157.2           22  ...               1212                0   206.81.17.231-183.240.157.2-22-40632-6
3    43.156.171.95   206.81.17.231        53188  ...               2050             2689   43.156.171.95-206.81.17.231-53188-22-6
4   129.226.89.244   206.81.17.231        45204  ...               1800             2439  129.226.89.244-206.81.17.231-45204-22-6
5  211.112.187.197   206.81.17.231        26689  ...               1944             2553 211.112.187.197-206.81.17.231-26689-22-6
6    43.153.82.175   206.81.17.231        59546  ...               1932             2557   43.153.82.175-206.81.17.231-59546-22-6

[7 rows x 85 columns]
1/1 [==============================] - 0s 29ms/step
TTD is calculated as 9.081615
IP Address: 43.156.171.95
apply_firewall_rule() called
Rule added
TTR is calculated as: 0.292688
```

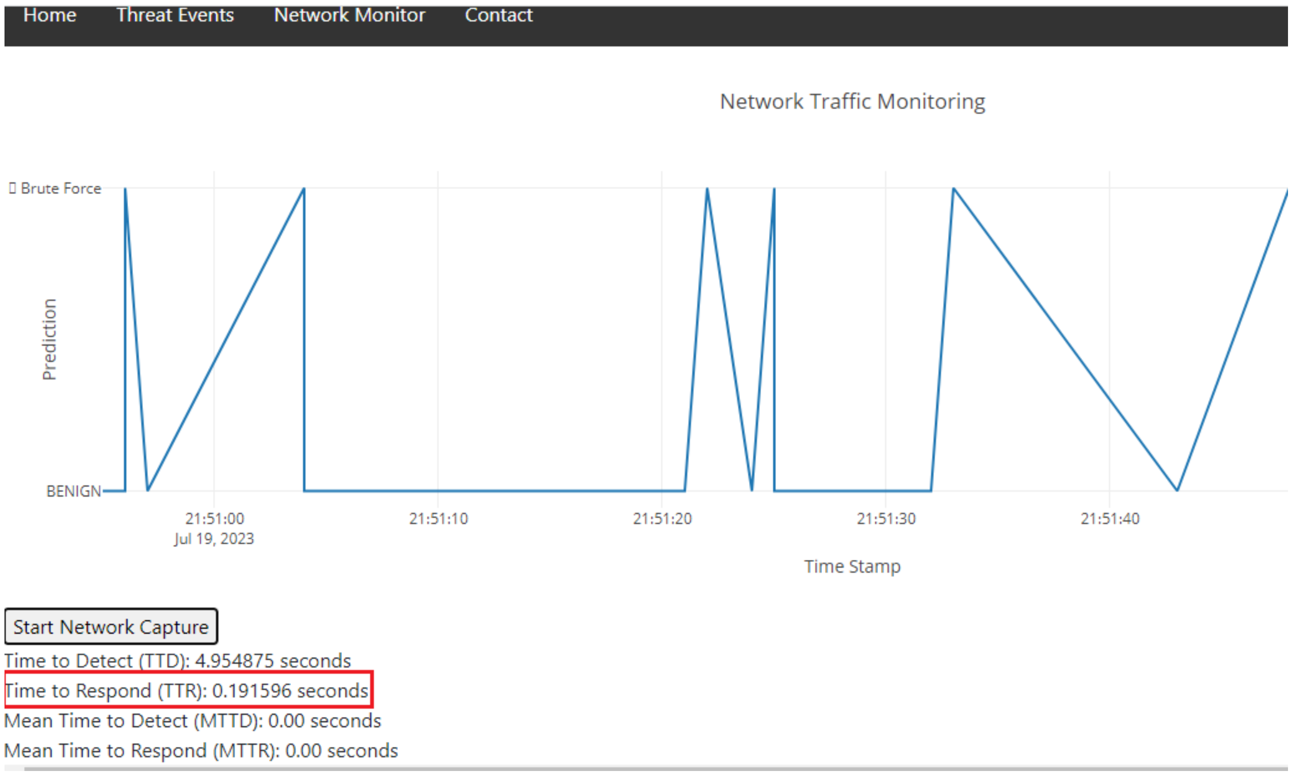
Fig. 22. A TTD Value Captured within the Backend Console



Time to Detect (TTD): 5.543272 seconds
Time to Respond (TTR): 0.169751 seconds
Mean Time to Detect (MTTD): 5.40 seconds
Mean Time to Respond (MTTR): 0.19 seconds

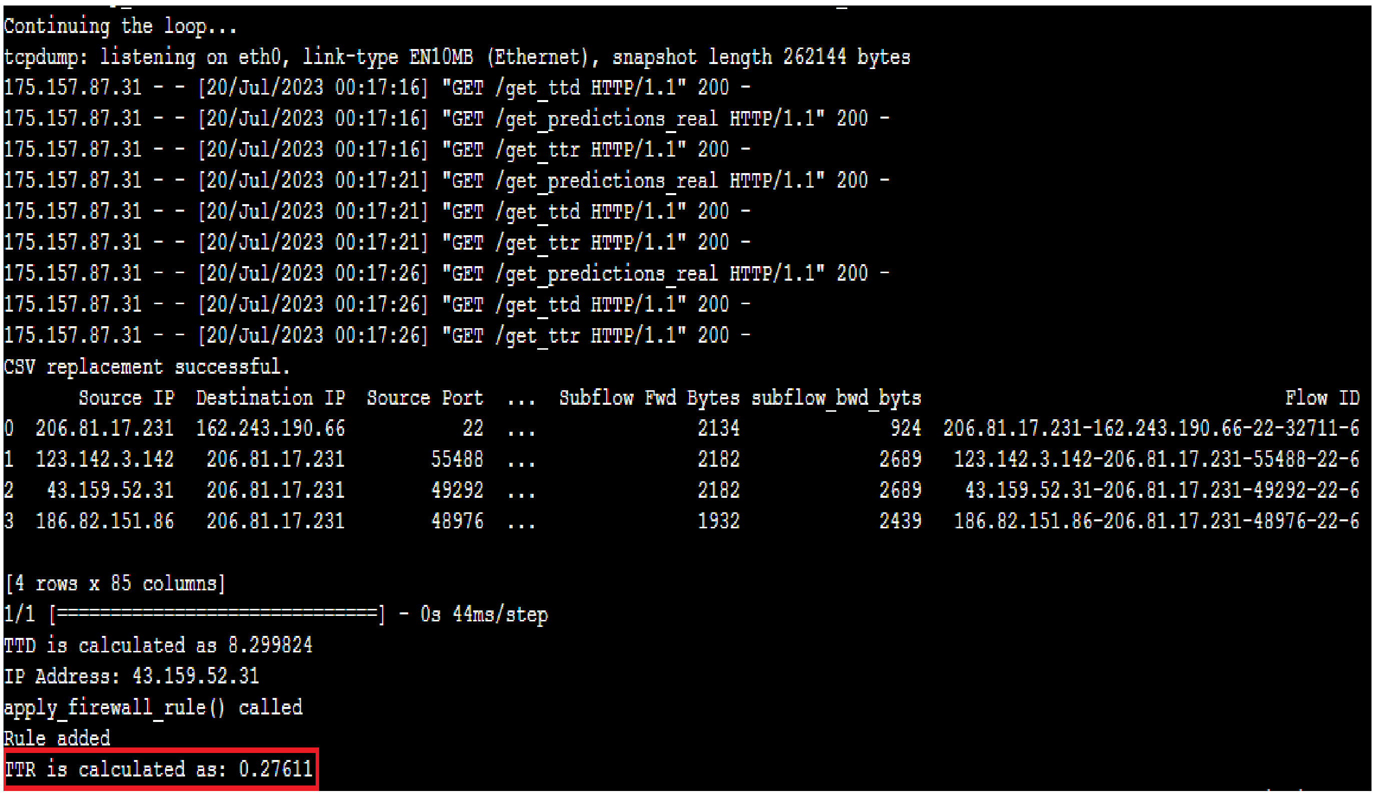
Fig. 23. MTTD Calculation from the Frontend

2) Results of System Effectiveness for Mean Time to Respond (MTTR): Mean Time to Respond (MTTR) is a critical performance metric that evaluates how swiftly the system implements appropriate mitigation measures upon detecting security incidents. In the developed system, the primary focus is on minimizing the MTTR to block the brute-force IPs in the firewall as soon as they are detected. Fig 24 shows a TTR value captured in the front end, representing the time taken to respond to the detected attacks. Fig 25 depicts a TTD value capture within the backend console. These values enable us to calculate the MTTR, as illustrated in Fig 26.



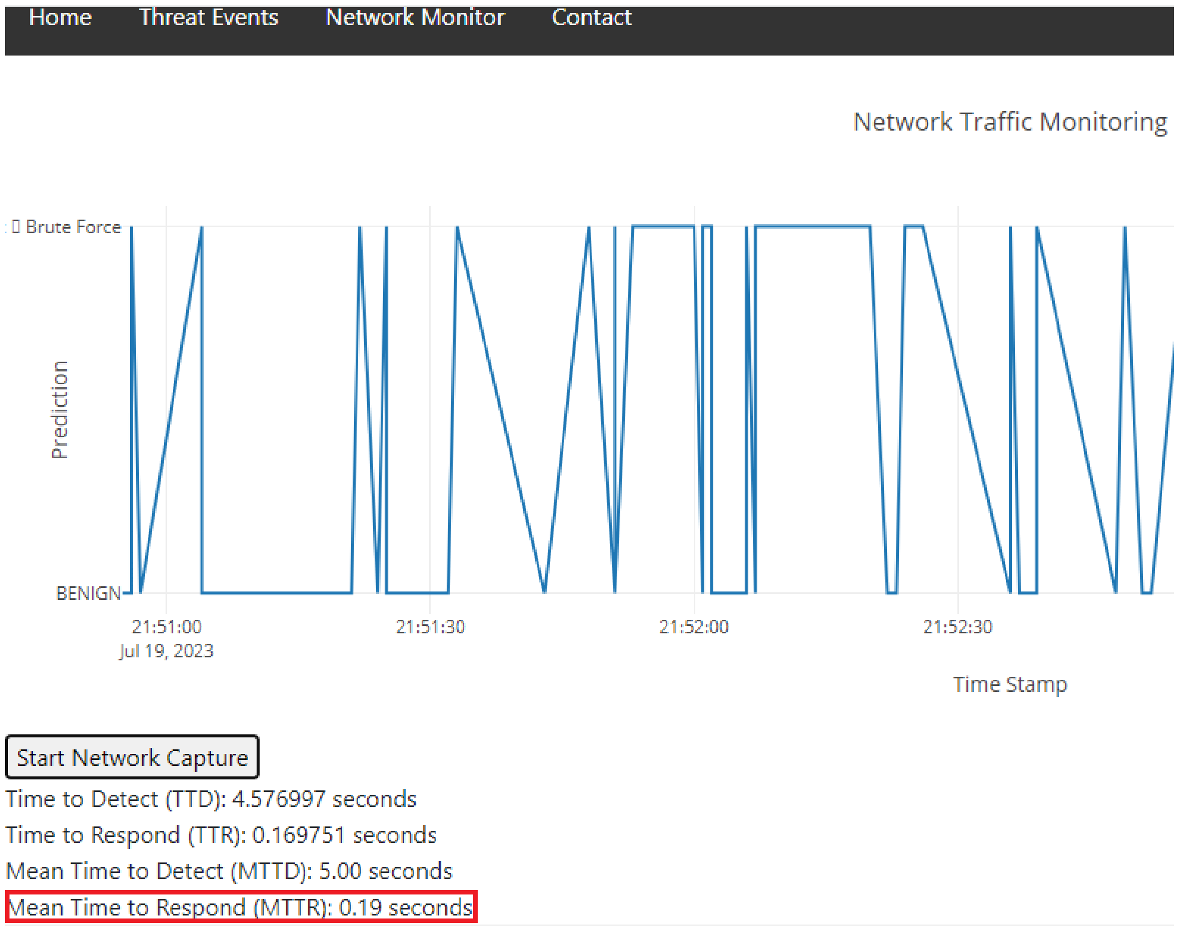
Time to Detect (TTD): 4.954875 seconds
Time to Respond (TTR): 0.191596 seconds
Mean Time to Detect (MTTD): 0.00 seconds
Mean Time to Respond (MTTR): 0.00 seconds

Fig. 24. A TTR Value Captured in the Frontend



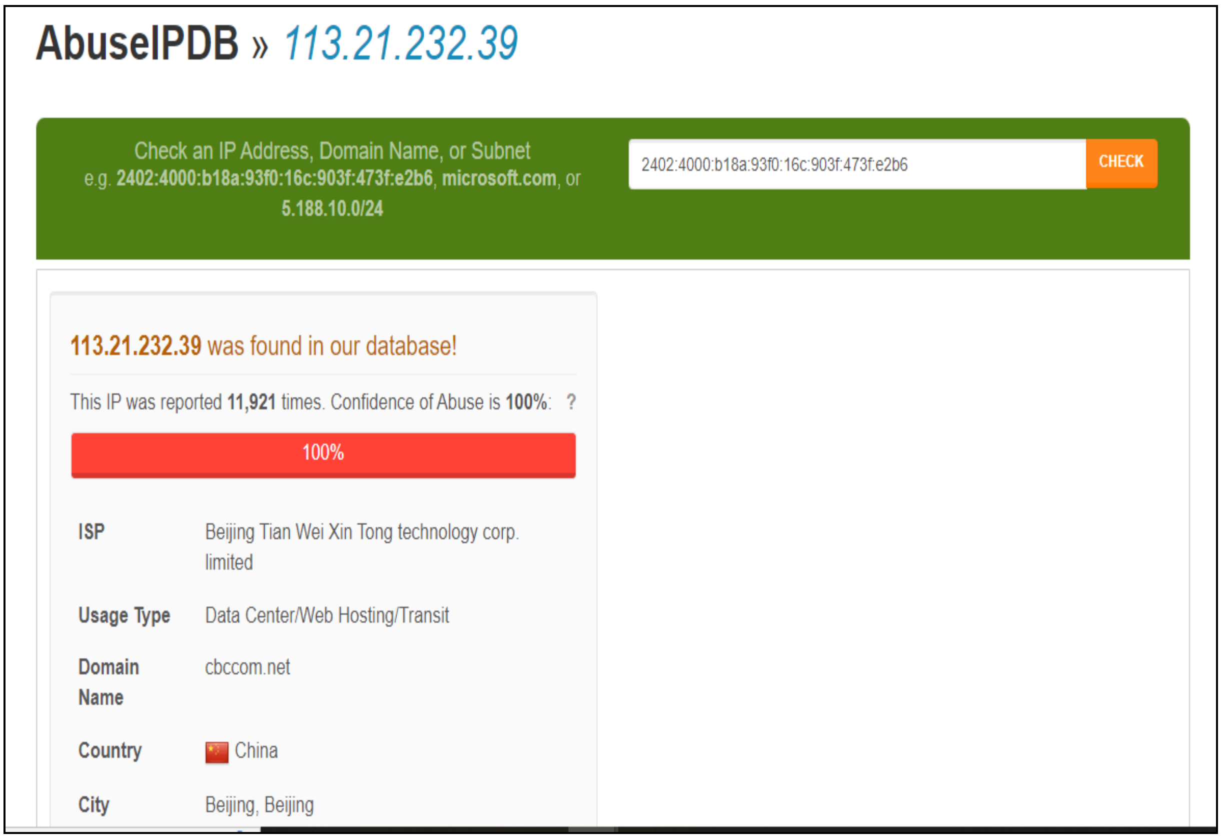
```
Continuing the loop...
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
175.157.87.31 - - [20/Jul/2023 00:17:16] "GET /get_ttd HTTP/1.1" 200 -
175.157.87.31 - - [20/Jul/2023 00:17:16] "GET /get_predictions_real HTTP/1.1" 200 -
175.157.87.31 - - [20/Jul/2023 00:17:16] "GET /get_ttr HTTP/1.1" 200 -
175.157.87.31 - - [20/Jul/2023 00:17:21] "GET /get_predictions_real HTTP/1.1" 200 -
175.157.87.31 - - [20/Jul/2023 00:17:21] "GET /get_ttd HTTP/1.1" 200 -
175.157.87.31 - - [20/Jul/2023 00:17:21] "GET /get_ttr HTTP/1.1" 200 -
175.157.87.31 - - [20/Jul/2023 00:17:26] "GET /get_predictions_real HTTP/1.1" 200 -
175.157.87.31 - - [20/Jul/2023 00:17:26] "GET /get_ttd HTTP/1.1" 200 -
175.157.87.31 - - [20/Jul/2023 00:17:26] "GET /get_ttr HTTP/1.1" 200 -
CSV replacement successful.
       Source IP  Destination IP  Source Port  ...  Subflow Fwd Bytes subflow_bwd_byts                             Flow ID
0  206.81.17.231  162.243.190.66           22  ...               2134              924  206.81.17.231-162.243.190.66-22-32711-6
1  123.142.3.142   206.81.17.231        55488  ...               2182             2689   123.142.3.142-206.81.17.231-55488-22-6
2   43.159.52.31   206.81.17.231        49292  ...               2182             2689    43.159.52.31-206.81.17.231-49292-22-6
3  186.82.151.86   206.81.17.231        48976  ...               1932             2439   186.82.151.86-206.81.17.231-48976-22-6

[4 rows x 85 columns]
1/1 [==============================] - 0s 44ms/step
TTD is calculated as 8.299824
IP Address: 43.159.52.31
apply_firewall_rule() called
Rule added
TTR is calculated as: 0.27611
```

Fig. 25. A TTD Value Captured within the Backend Console



Time to Detect (TTD): 4.576997 seconds
Time to Respond (TTR): 0.169751 seconds
Mean Time to Detect (MTTD): 5.00 seconds
Mean Time to Respond (MTTR): 0.19 seconds

Fig. 26. MTTR Calculation from the Frontend


AbuseIPDB » 113.21.232.39
Check an IP Address, Domain Name, or Subnet
e.g. 2402:4000:b18a:93f0:16c:903f:473f:e2b6, microsoft.com, or 5.188.10.0/24
2402:4000:b18a:93f0:16c:903f:473f:e2b6
CHECK

113.21.232.39 was found in our database!

This IP was reported **11,921** times. Confidence of Abuse is **100%**: ?

100%

ISP	Beijing Tian Wei Xin Tong technology corp. limited
Usage Type	Data Center/Web Hosting/Transit
Domain Name	cbccom.net
Country	China
City	Beijing, Beijing

Fig. 27. Reputation Check for Blocked IP (113.21.232.39)

By carefully checking the console messages and analyzing the MTTD and MTTR values, it is confirmed that the attacks are blocked at their first attempts, proving the system's effectiveness in blocking IPs before causing any damage. Furthermore, the developed SOC successfully classifies both Digital Ocean dedicated attacker's brute force attacks and random brute force attacks from around the world, showcasing the system's effectiveness in generalizing and classifying different attack types correctly. IPs from these attacks are added to the firewall deny list, preventing them from conducting further attacks. Fig 27 is a reputation check of one of those blocked IPs (randomly selected for checking), which is 113.21.232.39

In conclusion, the results demonstrate the strong performance and effectiveness of the ensemble machine learning system in detecting and classifying cyber threats in a security operations center.

G. Research Challenges and Threats to Proposed Model Validity

This section sheds light on the challenges encountered during the research journey, primarily in the realms of cybersecurity and machine learning. These hurdles were not only anticipated but also essential for refining the developed ensemble model. The researchers overcame these challenges by applying innovative solutions and methodologies, ultimately contributing to the advancement of the field.

The research encountered several challenges inherent to the cybersecurity and machine learning domains. Addressing these challenges was critical to obtaining accurate and reliable results. The lack of comprehensive literature discussing Hidden Markov Models in the context of cybersecurity posed a significant obstacle. Nevertheless, we proceeded with the research, aiming to address this identified knowledge gap and provide a novel contribution to the field. Additionally, obtaining a suitable and diverse cybersecurity dataset with sufficient labeled data was challenging. However, we applied careful data preparation techniques and used stratified sampling to preserve the original class distribution, ensuring balanced and representative datasets for training and evaluation. The compatibility issues between different model architectures, specifically LSTM and XGBoost, demanded innovative solutions. By aligning and combining the outputs of the two models, we successfully integrated them into the ensemble system.

During the development of the unique ensemble model combining HMM, LSTM, and XGBoost for cybersecurity anomaly detection, we encountered significant challenges. One major obstacle was the lack of comprehensive literature specifically discussing Hidden Markov Models in the context of cybersecurity anomalies. HMMs are unique generative machine learning models, and their application in the cybersecurity domain is relatively unexplored. Nevertheless, we proceeded with the project to address this knowledge gap and make a novel contribution to the field. Another challenge was the scarcity of publicly available datasets that fulfil the requirements for the research. The ideal dataset should follow the Markov property and contain features relevant to cybersecurity anomalies. Unfortunately, finding such a dataset proved to be a difficult task within the time constraints of the project.

Data preparation is crucial in developing machine learning-based intrusion detection systems. During this phase, we encountered several challenges, including:

1) Unbalanced Data: The accuracy of machine learning models can be significantly affected by an unbalanced dataset, where normal traffic far outweighs malicious activity. To address this, we applied balancing techniques to ensure accurate model training.
2) Outdated Data: To effectively detect evolving attacks, intrusion detection systems require training on current datasets. Using outdated data could negatively impact the model's performance, so we ensured the dataset was up to date.
3) Lack of Diversity: Effective intrusion detection systems should be able to detect a wide range of attacks. We addressed this challenge by curating a diverse dataset with various attack types, targets, and methodologies.
4) Limited Data Size: Machine learning models require a large volume of data to generalize effectively. Although we had a limited dataset, we used stratified sampling and other techniques to make the most of the available data.
5) Label Quality: Accurate labeling of data is essential for training reliable machine learning models. We ensured high-quality labeling standards to improve model accuracy and reliability.

Overcoming these data preparation challenges was critical to developing an accurate and effective ensemble model for cybersecurity anomaly detection. The modeling phase presented its own set of challenges, and they include:

1) Completely Different Model Architectures: One of the significant hurdles in the modeling phase was dealing with completely different model architectures for the LSTM and XGBoost models. As mentioned earlier, the LSTM model requires sequential data in the form of 3-dimensional arrays, while the XGBoost model expects tabular data in 2-dimensional arrays. To address this, we carefully preprocessed the data and ensured that the input data were appropriately formatted for each model.
2) Compatibility of Predictions: Integrating the outputs of the LSTM and XGBoost models into the ensemble model required handling different prediction formats. The LSTM model produced probabilities for each class, while the XGBoost model generated class labels. To combine these predictions effectively, we developed a mechanism to align and harmonize the outputs, ensuring seamless integration in the ensemble model.
3) The Nature of the Dataset vs. the Approach Taken**:** Working with a limited dataset posed challenges in designing the ensemble model. We had to carefully allocate the available data for training, testing, and validation sets while ensuring a justified distribution of classes among them. Only 7267 records remained after the reduction. Careful utilization was crucial for HMM, LSTM, XGBoost, and ensemble model training and testing. We utilized various techniques, such as stratified sampling in sklearn train_test_split, to preserve the original class distribution and prevent data leakage.

Despite these challenges, the ensemble model proved effective in identifying and classifying various cyber threats accurately, showcasing its robustness and reliability in real-world scenarios.

VI. Conclusion

This study makes significant contributions to the field of cybersecurity and threat detection within Security Operations Centers (SOCs). The study introduces and successfully demonstrates an ensemble machine learning approach that combines Gaussian Hidden Markov Models (HMM), Long Short-Term Memory (LSTM), and XGBoost algorithms, achieving an impressive average F1-score of 99% in identifying major web threats, including Brute Force Attacks, Cross Site Scripting (XSS), and SQL Injection, while minimizing false detections. The study evaluates the ensemble model's real-world efficacy using actual network traffic data and controlled testing with simulated attacks, presenting a comprehensive assessment of its operational efficiency and effectiveness within SOC environments. These contributions collectively advance the state of the art in cybersecurity practices and threat detection, providing valuable insights for future research endeavours and practical applications.

A. Future Works and Research Directions

Future works and research directions stemming from this study offer exciting prospects for further advancing cybersecurity practices and threat detection methodologies in Security Operations Centers (SOCs). Firstly, exploring the integration of emerging machine learning techniques, such as deep learning models and reinforcement learning algorithms, could enhance the ensemble model's predictive capabilities, potentially increasing detection accuracy and adaptability to evolving threats. Secondly, delving deeper into real-time threat monitoring systems and the incorporation of anomaly detection methods, coupled with the analysis of encrypted traffic, could further strengthen SOC defences against sophisticated attacks. Thirdly, investigating the scalability and resource efficiency of the ensemble model, particularly in large-scale network environments, will be crucial for its practical implementation. Additionally, research efforts may focus on the development of user-friendly interfaces and visualization tools that empower SOC analysts in interpreting model outputs and facilitating more informed decision-making. Lastly, fostering interdisciplinary collaborations with experts in network security, cryptography, and artificial intelligence can foster innovative solutions and cross-pollination of ideas to address emerging cybersecurity challenges effectively. These future works and research directions collectively aim to propel the field of cybersecurity forward, ensuring the continued adaptability and robustness of SOC operations in an ever-evolving threat landscape.

References

[1] M. Abomhara, G. Køien, and M. Alghamdi, "Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks," Journal of Cyber Security, vol. 4, pp. 65-88, 2021, doi: 10.13052/jcsm2245-1439.414.

[2] F. O. Sveen, J. M. Torres, and J. M. Sarriegi, "Blind information security strategy," International Journal of Critical Infrastructure Protection, vol. 2, no. 3, pp. 95-109, 2009, doi: 10.1016/j.ijcip.2009.07.003.

[3] F. G. Bîrleanu, P. Anghelescu, N. Bizon, and E. Pricop, "Cyber security objectives and requirements for smart grid," Energy Systems in Electrical Engineering, pp. 607–634, 2018, doi:10.1007/978-981-13-1768-2_17.

[4] M. Lehto, "Cyber-attacks against critical infrastructure," Computational Methods in Applied Sciences, pp. 3–42, 2022, doi:10.1007/978-3-030-91293-2_1.

[5] J. Kennedy, T. Holt, and B. Cheng, "Automotive cybersecurity: Assessing a new platform for cybercrime and malicious hacking," Journal of Crime and Justice, vol. 42, no. 5, pp. 632–645, 2019, doi:10.1080/0735648x.2019.1692425.

[6] L. Aijaz, B. Aslam and U. Khalid, "Security operations center — A need for an academic environment," *2015 World Symposium on Computer Networks and Information Security (WSCNIS)*, Hammamet, Tunisia, 2015, pp. 1-7, doi: 10.1109/WSCNIS.2015.7368297.

[7] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in Cloud," Journal of Network and Computer Applications, vol. 36, pp. 42-57, 2013, doi:10.1016/j.jnca.2012.05.003.

[8] K. Demertzis, P. Kikiras, N. Tziritas, S. L. Sanchez, and L. Iliadis, "The Next Generation Cognitive Security Operations Center: Network Flow Forensics Using Cybersecurity Intelligence," Big Data and Cognitive Computing, vol. 2, no. 4, pp. 35, 2018. doi:10.3390/bdcc2040035.

[9] D. Schlette, M. Vielberth, and G. Pernul, "CTI-SOC2M2 – the quest for mature, intelligence-driven security operations and incident response capabilities," Computers & Security, vol. 111, p. 102482, 2021. doi:10.1016/j.cose.2021.102482.

[10] Wang, T. Yan, D. An, Z. Liang, C. Guo, H. Hu, Q. Luo, H. Li, H. Wang, S. Zeng, C. Zhou, L. Ma, and F. Qi, "A comprehensive security operation center based on Big Data Analytics and threat intelligence," in Proceedings of International Symposium on Grids & Clouds 2021 — PoS(ISGC2021), 2021, doi:10.22323/1.378.0028.

[11] J. Muniz, N. AlFardan, and G. McIntyre, Security Operations Center: Building, Operating, and Maintaining Your SOC, Hoboken, NJ, USA: Cisco Press, 2015.

[12] F. B. Kokulu, A. Soneji, T. Bao, Y. Shoshitaishvili, Z. Zhao, A. Doupé, and G.-J. Ahn, "Matched and Mismatched SOCs: A Qualitative Study on Security Operations Center Issues," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 2019, pp. 1955-1970, doi: 10.1145/3319535.3354239.

[13] C. Hill, "Security Operation Center (SOC) – NASCIO," 2017. [Online]. Available: https://www.nascio.org/wp-content/uploads/2020/09/NASCIO-IL-2018-Cybersecurity-SOC.pdf.

[14] M. Vielberth, F. Bohm, I. Fichtinger, and G. Pernul, "Security Operations Center: A systematic study and open challenges," IEEE Access, vol. 8, pp. 227756–227779, 2020, doi:10.1109/access.2020.3045514.

[15] D. J. Marcus, "The Data Breach Dilemma: Proactive Solutions for Protecting Consumers' Personal Information," Duke L.J., vol. 68, p. 555, 2018. [Online]. Available: https://scholarship.law.duke.edu/dlj/vol68/iss3/3

[16] R. Syed, "Enterprise reputation threats on social media: A case of data breach framing," The Journal of Strategic Information Systems, vol. 28, no. 3, pp. 257-274, Sep. 2019. doi: 10.1016/j.jsis.2018.12.001

[17] B. Faulkner, "Hacking into Data Breach Notification Laws," Fla. L. Rev., vol. 59, pp. 1097, 2007. [Online]. Available: https://scholarship.law.ufl.edu/flr/vol59/iss5/5

[18] C. Zhong, J. Yen, P. Liu, and R. F. Erbacher, "Automate cybersecurity data triage by leveraging human analysts' cognitive process," in IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2016, pp. 357–363, doi:10.1109/bigdatasecurity-hpsc-ids.2016.41.

[19] C. Onwubiko, "Cyber security operations centre: Security monitoring for protecting business and supporting cyber defense strategy," in International Conference on Cyber Situational Awareness, Data Analytics and Assessment, London, UK, 2015, pp. 1-10. https://doi.org/10.1109/CyberSA.2015.7166125.

[20] P. Lif and T. Sommestad, "Human factors related to the performance of intrusion detection operators," in Proceedings of the Ninth International Symposium on Human Aspects of Information Security & Assurance (HAISA), Lesvos, Greece, 2015, pp. 265-275.

[21] B. P. Hámornik and C. Krasznay, "A team-level perspective of human factors in Cyber Security: Security Operations Centers," in Advances in Intelligent Systems and Computing, 2017, pp. 224–236, doi:10.1007/978-3-319-60585-2_21.

[22] S. Kowtha, L. A. Nolan, and R. A. Daley, "Cyber security operations center characterization model and analysis," in IEEE Conference on Technologies for Homeland Security (HST), Waltham, MA, USA, 2012, pp. 470-475. https://doi.org/10.1109/THS.2012.6459894.

[23] S. C. Sundaramurthy, A. G. Bardas, J. Case, X. Ou, M. Wesch, J. McHugh, and S. R. Rajagopalan, "A Human Capital Model for Mitigating Security Analyst Burnout," in Eleventh Symposium On Usable Privacy and Security (SOUPS), pp. 347–359, USENIX Association, 2015. [Online]. https://www.usenix.org/conference/soups2015/proceedings/presentation/sundaramurth

[24] C. Zhong, J. Yen, P. Liu, and R. F. Erbacher, "Learning from experts' experience: Toward Automated Cyber Security Data Triage," IEEE Systems Journal, vol. 13, no. 1, pp. 603–614, 2019. doi:10.1109/jsyst.2018.2828832.

[25] C. Feng, S. Wu, and N. Liu, "A user-centric machine learning framework for cybersecurity operations center," in IEEE International Conference on Intelligence and Security Informatics (ISI), Beijing, China, 2017, pp. 173–175. DOI: 10.1109/ISI.2017.8004902.

[26] A. Shah, R. Ganesan, and S. Jajodia, "A methodology for ensuring fair allocation of CSOC effort for alert investigation," International Journal of Information Security, vol. 18, pp. 199–218, 2019. https://doi.org/10.1007/s10207-018-0407-3.

[27] S. C. Sundaramurthy et al., "A tale of three security operation centers," in Proceedings of the 2014 ACM Workshop on Security Information Workers, 2014, pp. 43–50, doi:10.1145/2663887.2663904.

[28] S. C. Sundaramurthy et al., "Turning Contradictions into Innovations or: How We Learned to Stop Whining and Improve Security Operations," in Twelfth Symposium on Usable Privacy and Security (SOUPS), USENIX Association, pp. 237–251, 2016. [Online]. Available: https://www.usenix.org/conference/soups2016/technical-sessions/presentation/sundaramurthy.

[29] S. Schinagl, K. Schoon, and R. Paans, "A Framework for Designing a Security Operations Centre (SOC)," in Hawaii International Conference on System Sciences, Kauai, HI, USA, 2015, pp. 2253-2262. https://doi.org/10.1109/HICSS.2015.270.

[30] P. Jacobs, A. Arnab, and B. Irwin, "Classification of security operation centers," in Information Security for South Africa, 2013, pp. 1–7. https://doi.org/10.1109/ISSA.2013.6641054.

[31] B. A. Alahmadi, "99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms," in Proceedings of the 31st USENIX Security Symposium (USENIX Security), Boston, MA, USA, 2022.

[32] A. Angelopoulos, E. T. Michailidis, N. Nomikos, P. Trakadas, A. Hatziefremidis, S. Voliotis, and T. Zahariadis, "Tackling Faults in the Industry 4.0 Era—A Survey of Machine-Learning Solutions and Key Aspects," Sensors, vol. 20, no. 1, p. 109, 2020. doi:10.3390/s20010109

[33] V. Naganathan, "Comparative Analysis of Big Data, Big Data Analytics: Challenges and Trends," International Research Journal of Engineering and Technology (IRJET), vol. 05, no. 05, pp. 2378-2382, 2018.

[34] N. Kumar, A. C. Sen, V. Hordiichuk, M. T. Espinosa Jaramillo, B. Molodetskyi, and A. B. Kasture, "AI in Cybersecurity: Threat Detection and Response with Machine Learning," Journal of Propulsion Technology, vol. 44, no. 3, 2023. [Online]. Available: https://www.propulsiontechjournal.com/index.php/journal/article/view/237

[35] S. K. Hassan and A. Ibrahim, "The role of Artificial Intelligence in cyber security and incident response," International Journal for Electronic Crime Investigation, vol. 7, no. 2, pp. 1-8, 2023. doi:https://doi.org/10.54692/ijeci.2023.0702154

[36] A. Diro, S. Kaisar, A. V. Vasilakos, A. Anwar, A. Nasirian, and G. Olani, "Anomaly Detection for Space Information Networks: A Survey of Challenges, Schemes, and Recommendations," 2023. [Online]. Available: https://doi.org/10.36227/techrxiv.23584530.v1

[37] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," Computers & Security, vol. 28, no. 1–2, pp. 18–28, 2009. doi:10.1016/j.cose.2008.08.003

[38] A. Mishra, K. Nadkarni and A. Patcha, "Intrusion detection in wireless ad hoc networks," in *IEEE Wireless Communications*, vol. 11, no. 1, pp. 48-60, Feb. 2004, doi: 10.1109/MWC.2004.1269717.

[39] A. Khraisat, I. Gondal, P. Vamplew, et al., "Survey of intrusion detection systems: techniques, datasets and challenges," Cybersecur, vol. 2, no. 20, 2019. doi:10.1186/s42400-019-0038-7.

[40] D. S. Sharma and D. S. Srivastava, "An overview on e-learning and information security management," International Journal for Research in Applied Science and Engineering Technology, vol. 10, no. 9, pp. 427–435, 2022. doi:10.22214/ijraset.2022.46531

[41] B. D. Bryant and H. Saiedian, "A novel kill-chain framework for remote security log analysis with SIEM software," Computers & Security, vol. 67, pp. 198-210, Jun. 2017. doi:10.1016/j.cose.2017.03.003.

[42] K. Demertzis, P. Kikiras, N. Tziritas, S. L. Sanchez, and L. Iliadis, "The Next Generation Cognitive Security Operations Center: Network Flow Forensics Using Cybersecurity Intelligence," Big Data and Cognitive Computing, vol. 2, no. 4, pp. 35, 2018. doi:10.3390/bdcc2040035.

[43] M.-J. Kwak, H. G. Kong, K. Choi, S.-K. Kwon, J. Y. Song, J. Lee, P. A. Lee, S. Y. Choi, M. Seo, H. J. Lee, E. J. Jung, H. Park, N. Roy, H. Kim, M. M. Lee, E. M. Rubin, S.-W. Lee, and J. F. Kim, "Rhizosphere microbiome structure alters to enable wilt resistance in tomato," Nature Biotechnology, vol. 36, pp. 1100–1109, 2018. doi:10.1038/nbt.4232

[44] A. Oprea, Z. Li, R. Norris, and K. Bowers, "MADE: Security Analytics for Enterprise Threat Detection," Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18), pp. 124–136, Dec. 2018. doi:10.1145/3274694.3274710

[45] K. Demertzis, N. Tziritas, P. Kikiras, S. Sanchez, and L. Iliadis, "The Next Generation Cognitive Security Operations Center: Adaptive Analytic Lambda Architecture for Efficient Defense against Adversarial Attacks," Big Data and Cognitive Computing, vol. 3, no. 6, 2019, doi:10.3390/bdcc3010006.

[46] Q. Chen, S. R. Islam, H. Haswell, and R. A. Bridges, "Automated Ransomware Behavior Analysis: Pattern Extraction and Early Detection," Science of Cyber Security, pp. 199–214, 2019. doi:10.1007/978-3-030-34637-9_15

[47] J. Zhang, Y. Jou, and X. Li, "Cross-Site Scripting (XSS) Detection Integrating Evidences in Multiple Stages," Proceedings of the 52nd Hawaii International Conference on System Sciences, 2019. [Online]. Available: http://hdl.handle.net/10125/60153.

[48] A. Majeed, R. ur Rasool, F. Ahmad, M. Alam, and N. Javaid, "Near-miss situation based visual analysis of SIEM rules for Real Time Network Security Monitoring," Journal of Ambient Intelligence and Humanized Computing, vol. 10, no. 4, pp. 1509–1526, 2018. doi:10.1007/s12652-018-0936-7.

[49] N. Jindal, "Automated event prioritization for security operation center using graph-based features and deep learning," PhD diss., 2020.

[50] M. Beulah and B. Pitchai Manickam, "Detection of DDoS Attack Using Ensemble Machine Learning Techniques," in Advances in Intelligent Systems and Computing, vol. 1397, Springer, Singapore, 2022, pp. 333-344. doi:10.1007/978-981-16-5301-8_62.

[51] A. Al-Abassi, H. Karimipour, A. Dehghantanha and R. M. Parizi, "An Ensemble Deep Learning-Based Cyber-Attack Detection in Industrial Control System," in *IEEE Access*, vol. 8, pp. 83965-83973, 2020, doi: 10.1109/ACCESS.2020.2992249.

[52] E. Agyepong, Y. Cherdantseva, P. Reinecke and P. Burnap, "Towards a Framework for Measuring the Performance of a Security Operations Center Analyst," *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, Dublin, Ireland, 2020, pp. 1-8, doi: 10.1109/CyberSecurity49315.2020.9138872.

[53] J. Ding, C. Lu, and B. Li, "A Data-Driven Based Security Situational Awareness Framework for Power Systems," Journal of Signal Processing Systems, vol. 94, no. 11, pp. 1159–1168, 2022. doi:10.1007/s11265-022-01741-y

[54] E. Bisong, "Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners (1st ed.)," pp. 215-229, Apress Berkelely, CA, 2019. doi:10.1007/978-1-4842-4470-8

[55] K. M. Ali Alheeti and K. McDonald-Maier, "Intelligent intrusion detection in external communication systems for Autonomous Vehicles," Systems Science & Control Engineering, vol. 6, no. 1, 2018, doi:10.1080/21642583.2018.1440260.

[56] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, "Tree-based intelligent intrusion detection system in internet of vehicles," in 2019 IEEE Global Communications Conference (GLOBECOM), 2019, doi:10.1109/globecom38437.2019.9013892.

[57] M. S. Korium, M. Saber, A. Beattie, A. Narayanan, S. Sahoo, and P. H. J. Nardelli, "Intrusion detection system for cyberattacks in the internet of vehicles environment," Procedia Computer Science, vol. 115, pp. 588-595, 2017, doi:10.1016/j.procs.2017.09.169.

[58] Y. Chen, C. M. Poskitt, and J. Sun, "Learning from Mutants: Using Code Mutation to Learn and Monitor Invariants of a Cyber-Physical System," in Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2018, doi:10.1109/SP.2018.00016.

[59] A. Bibi, G. A. Sampedro, A. Almadhor, A. R. Javed, and T.-h. Kim, "A Hypertuned Lightweight and Scalable LSTM Model for Hybrid Network Intrusion Detection," Technologies, vol. 11, no. 5, p. 121, 2023, https://doi.org/10.3390/technologies11050121.

[60] Beulah, M., & Pitchai Manickam, B. (2022). Detection of DDoS attack using ensemble machine learning techniques. Advances in Intelligent Systems and Computing, 1397, 739-752. Springer, Singapore. https://doi.org/10.1007/978-981-16-5301-8_62