Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)
Vol. 10, No. 4, December 2024, pp. 1004-1020
ISSN: 2338-3070, DOI: 10.26555/jiteki.v10i4.30380

Enhancing Network Security Through Real-Time Threat Detection with Intrusion Prevention System (Case Study on Web Attack)

Tia Rahmawati[1], Nyoman Karna[2], Soo Young Shin[3], Made Adi Paramartha Putra[4]
[1]School of Electrical Engineering, Telkom University, Bandung 40257, Indonesia
[2]The University Center of Excellence for Intelligent Sensing-IoT, Telkom University, Indonesia
[3]Department of IT Convergence Engineering, Kumoh National Institute of Technology, Gumi, South Korea
[4]Faculty of Information Technology and Design, Primakara University, Denpasar, Indonesia

ARTICLE INFO

Article history:

Received December 02, 2024
Revised January 12, 2025
Accepted February 04, 2025

Keywords:

Cyberattacks;
Elasticsearch;
Intrusion Prevention System;
Suricata;
Web Attack

ABSTRACT

Cyberattacks on government websites in Indonesia have been steadily increasing, with over 109 million incidents recorded in 2023 by the National Cyber Security Operations Center (BSSN). A Netcraft survey revealed that more than one billion websites globally face similar threats, highlighting the urgent need for improved security measures, especially given infrastructure limitations and inadequate security implementations. Approximately 51% of Micro, Small, and Medium Enterprises in Indonesia reported experiencing web attacks, with 95% stating that these attacks severely disrupted their operations. This study implements a Suricata-based Intrusion Prevention System (IPS) to protect web servers from attacks such as SQL Injection, XSS, and command injection. Suricata monitors network traffic and blocks threats in real time. Detection logs in JSON format are managed through Filebeat, processed by Logstash, stored in Elasticsearch, and visualized using Kibana. The key contribution of this research lies in designing a comprehensive set of rules and integrating all components into a single Docker container, streamlining the deployment process. Testing confirmed that the designed rules effectively detect and block attack payloads by leveraging a rule structure in suricata and nfqueue capable of identifying all suspicious traffic. The novelty of this research lies in deploying a fully operational real-time security system on low-resource computers, demonstrating effective threat management under constrained conditions.

Corresponding Author:

Nyoman Karna, School of Electrical Engineering, Telkom University, Bandung 40257, Indonesia
Email: aditya@telkomuniversity.ac.id

1. INTRODUCTION

Government websites in Indonesia have become primary targets in cyberattacks. The National Cyber Security Operations Center (Pusopskamsinas) of the National Cyber and Crypto Agency (BSSN) recorded that a total of 109,379,790 incidents occurred on websites from January to December 2023. According to a survey conducted by Netcraft in December 2023, there were responses from 1,088,057,023 sites across 269,268,434 domains and 12,355,610 computers facing the web. Referring to the provided data, enhancing security becomes crucial to prevent, reduce, and address occurring attacks [1]. Some reasons behind frequent web application attacks include inadequate infrastructure usage and shortcomings in security concept implementation [2].

To gain a deeper understanding of this issue, a sample was taken from Micro, Small, and Medium Enterprises (UMKM) employees who use the web in their business transactions. The study results showed that 51% of 100 respondents experienced web attacks on their business systems. Furthermore, 95% considered web attacks highly disruptive to their daily business operations. This indicates that web security risks are significant issues that need to be addressed seriously. In efforts to enhance web system security, the implementation of

various appropriate strategies and technologies is required. One commonly used strategy is to build a robust security topology. Security topology encompasses various steps and technologies designed to protect web systems from various threats [3].

In a study titled "Centralized Log Management Using Elasticsearch, Logstash, and Kibana" by Farrukh Ahmed (2020), a centralized log management using ELK was conducted, providing centralized logging accessibility useful for identifying issues faced by servers or applications [4]. This will be continuous with this study, which utilizes Logstash, Elasticsearch, and Kibana in retrieving logs from IPS. The study titled "Integrated Security Information and Event Management (SIEM) with Intrusion Detection System (IDS) for Live Analysis based on Machine Learning" by Adabi Muhammad, Parman Sukarno, and Aulia Wardana (2023) focuses on developing a Security Information & Event Management (SIEM) system based on live analysis using machine learning integrated with an Intrusion Detection System (IDS) [5]. This research aims to build a system using commonly utilized open-source components for real-time analysis, detection, and monitoring of cyberattacks. The study employs a combination of Elastic (ELK) Stack, Slips, and Zeek IDS to construct the system. This study will be extended by implementing a similar framework but with an Intrusion Prevention System (IPS) Suricata integrated with the ELK Stack to enhance the detection and prevention capabilities of the system, ensuring comprehensive network security. The study titled "Automatic Decision Tree-Based NIDPS Ruleset Generation for DoS/DDoS Attacks" by Antonio Coscia and Vincenzo Dentamaro (2024) proposes an algorithm called Anomaly2Sign to automatically generate Suricata rules using a Decision Tree (DT) for detecting and blocking Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks [6]. This research focuses on optimizing Suricata rules for real-time blocking of SQL Injection, XSS, and Command Injection.

Among the commonly used technologies in security topology are Firewall, Intrusion Detection System (IDS), and Intrusion Prevention System (IPS). Firewall acts as the first barrier in preventing unauthorized access to the network [7]. however, without support from IDS and IPS, network security remains vulnerable to various complex attacks. IDS detects suspicious attacks or unusual traffic patterns in the network [8]. However, its weakness lies in its ability only to detect without directly stopping these attacks. This opens up opportunities for attacks to continue and potentially cause greater losses [9]. Conversely, IPS offers a more proactive solution by automatically stopping detected attacks [10]. This advantage makes IPS a superior choice compared to IDS in dealing with web security threats.

The main focus of this research is to design a rule structure within Suricata and NFQueue as a firewall capable of blocking all traffic indicating suspicious payloads, ensuring the system's effectiveness in addressing various types of attacks. The context of this research is to design an Intrusion Prevention System (IPS) where the designed rules can detect patterns of SQL Injection, Cross-Site Scripting (XSS), and Command Injection attacks. This research focuses on functionality validity testing, such as whether the created rules can be detected and matched with Suricata logs, and ensuring that these logs are successfully ingested into Elasticsearch and the attacks are detected by the IPS. In addition, reliability testing is conducted to evaluate the consistency of the IPS performance by testing attack payloads such as SQL Injection, XSS, and Command Injection [11]. Kibana will be used to visualize the log data stored in Elasticsearch through various types of graphs, diagrams, and interactive dashboards. The research results indicate, after conducting pentest trials with DVWA, the rules designed to block SQL Injection, XSS, and Command Injection attacks successfully detected and blocked all attack payloads with 100% effectiveness. All logs from Suricata were successfully integrated into Elasticsearch for data storage, while Logstash parsed and processed the raw logs to make them easier to query. Kibana also performed well in displaying data visualizations, enabling more effective and informative attack analysis.

2. METHODS

The methodology used in this research is the Network Development Life Cycle (NDLC) [12], which offers a step-by-step approach to designing and building networks. This method includes various stages such as needs analysis, design, implementation, testing, and maintenance. NDLC is considered an effective method for developing and analyzing network infrastructure, as it ensures that each step taken meets specific requirements and enhances network performance [12]. The system development method is shows on Fig. 1 carried out in several stages analysis, design, prototype simulation, implementation, monitoring, and management. The first stage is problem identification, which aims to address the high risks of web-based attacks, such as SQL Injection, XSS, and Command Injection, that threaten system security. Next, the problem statement is formulated with a focus on developing an Intrusion Prevention System (IPS) using Suricata to detect and block attacks in real-time and ensure log integration into Elasticsearch for further analysis. Suricata operates as an IPS engine that directly blocks traffic based on predefined rules designed to detect and stop payloads deemed suspicious or malicious. This approach ensures that only traffic meeting the criteria of the

rules is allowed, thereby reducing the risk of threats to the system. Suricata provides only an IPS engine that identifies packets strictly based on triggered rules, making its effectiveness highly dependent on the accuracy and comprehensiveness of these rules in detecting various threats. A literature review is conducted to understand attack detection techniques, optimal configurations, and the best log processing strategies [13].

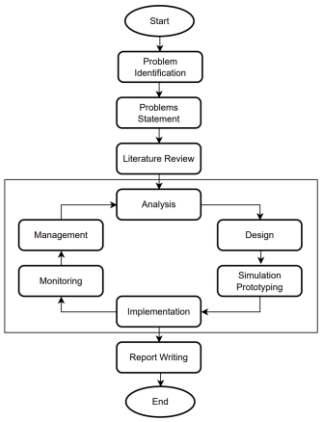


Fig. 1. NDLC Methods

The needs analysis phase of this research identified key threats, including SQL Injection, Cross-Site Scripting (XSS), and Command Injection, based on their frequent occurrence in web application attacks. Specific criteria were defined, such as the selection of keywords like group by, script, and whoami, which are commonly associated with these types of attacks. The analysis also focused on monitoring HTTP traffic directed to the server (flow:to_server) to enhance threat detection effectiveness. The HTTP protocol was prioritized as it is the primary vector for web-based attacks, with universal rule implementation (any any -> any any) to allow flexible detection without restricting specific IP addresses or ports. During the design phase, the drop action was implemented to block suspicious traffic immediately. Each rule included a descriptive message (msg) to assist administrators in identifying the detected threat, such as "TIA-SQL Injection Group by" or "TIA-XSS Injection Script." The content examined targeted the HTTP URI (http_uri) with specific keywords, while the nocase configuration ensured case insensitivity to account for variations in attack patterns. Metadata was incorporated to record the creation and update timestamps of each rule, ensuring their continued relevance to evolving threats. This design aims to detect and block threats in real-time while simplifying the processes of system maintenance and security analysis [14]. During the simulation phase, testing is conducted in a test environment to ensure that all components, from Suricata rules to log integration, function properly before deployment in the production environment. The implementation stage includes deploying the system in a production environment with full integration between Suricata, Filebeat, Logstash, Elasticsearch, and Kibana. After implementation, monitoring is carried out to ensure that the system effectively detects and mitigates threats and allows for adjustments if necessary. The final stage is continuous management, which involves updating rules, optimizing the log pipeline, and maintaining infrastructure. All these processes are then documented in a report containing test results and guidelines for future system development.

2.1. System Model

The proposed system model describes how Suricata is positioned as the primary line of defense for the web server, safeguarding data and maintaining system integrity against threats originating from the internet refer to the Fig. 2.

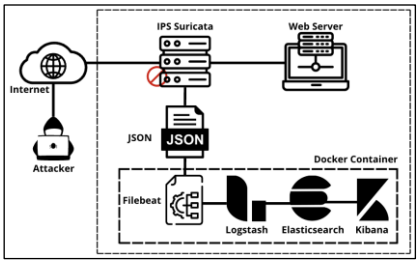


Fig. 2. System Model

This system model illustrates the implementation of an Intrusion Prevention System (IPS) using Suricata to protect a web server from attacks originating from the internet. On the far left, there is an attacker who attempts to access or attack the system through the internet, with attacks such as SQL injection, cross-site scripting (XSS), or command injection. The internet acts as the medium connecting the attacker to the system. In the middle, Suricata serves as the primary security layer, monitoring network traffic between the internet and the web server. By using various rules, Suricata can detect attack patterns and issue alerts or even block suspicious access, which is depicted by the red stop symbol on the connection line. Suricata ensures that only safe traffic reaches the web server, thus safeguarding the server's data and integrity, for hardware implemetation as shown on Fig. 3.

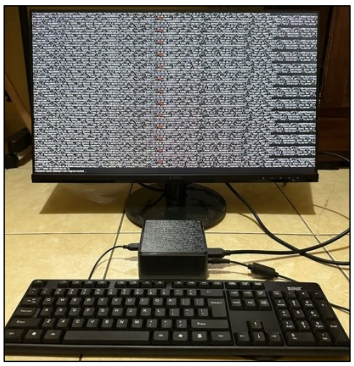


Fig. 3. Hardware Implementation

The detection logs from Suricata are sent in JSON format to Filebeat [15], which is responsible for forwarding the logs to Logstash [16]. Logstash then processes and parses the logs before storing them in Elasticsearch, which is used to efficiently store and search log data. Kibana connected to Elasticsearch, allows administrators to visualize and analyze the log data in the form of graphs or interactive dashboards. With Kibana, administrators can monitor attacks in real-time and gain clearer insights into network security [17], [18], [19]. Overall, this system works to ensure real-time security of the web server and provides valuable data for analysis and decision-making related to network security. The system specifications used for testing and implementing the Intrusion Prevention System (IPS) with Suricata, as described in this study, are outlined in Table 1.

Table 1. System Specification Information

System Specifications Information	
CPU	Intel Core i3-10110U @ 4x 4.1GHz
Kernel	x86_64 Linux 5.15.0-124-generic
RAM	7.34 GB used / 15.33 GB total
OS	Ubuntu 22.04

The proposed system model not only focuses on integrating Suricata with the ELK Stack to prevent attacks and monitor the network but is also designed to operate on low-end and resource-limited computers as shown on Fig. 3. This means the system can run on basic hardware with limited CPU, memory, and storage capacity. Despite using low-specification devices, the system is still capable of detecting, blocking, and logging attacks in real-time. This success is largely attributed to the use of Docker, which enables all components such as Filebeat, Logstash, Elasticsearch, and Kibana to run together efficiently without consuming excessive resources. Each component uses resources as needed, ensuring the system operates smoothly. For example, resource-intensive tasks such as log storage and data visualization can be efficiently managed by Elasticsearch and Kibana without placing an excessive load on the system. This approach demonstrates that robust security solutions can still be implemented on resource-constrained devices. This is particularly relevant for organizations or individuals with limited budgets that still require a reliable network security system. With this design, the system is capable of providing maximum protection without the need for expensive hardware. However, a key challenge in implementing this system lies in the limitations of low-specification hardware, which has the potential to cause performance bottlenecks when handling high traffic or large volumes of logs. The log pipeline involving Filebeat, Logstash, and Elasticsearch is also prone to bottlenecks if not properly managed. To address this issue, optimizing Suricata rules can enhance detection efficiency, while implementing a log rotation mechanism is essential to maintaining storage capacity. By leveraging

performance monitoring through Kibana and conducting load testing before implementation, the system can ensure reliability across various operational scenarios.

2.2. Intrusion Prevention System with Suricata

The flowchart illustrates the working process of an Intrusion Prevention System (IPS) using Suricata for detecting and preventing attacks, refer to Fig. 4. The process starts when data or network traffic enters the Suricata system. This data may contain potential attacks, such as SQL injection, XSS, or command injection. Suricata then checks each incoming data packet by comparing it against a set of predefined rules. These rules define specific attack patterns that the system needs to be aware of. If no attack is detected, the data is allowed to pass through without intervention. However, if an attack is identified, Suricata immediately blocks the data packet to prevent it from reaching the server or target system, as a precautionary measure. After blocking the attack, Suricata logs information related to the incident, such as the type of attack and the source of the traffic[20]. This data is useful for further analysis. Once all these steps are completed, the process ends. With this approach, Suricata acts as an active defense system that can detect and mitigate threats in real-time, thereby effectively securing the network [21], [22], [23].

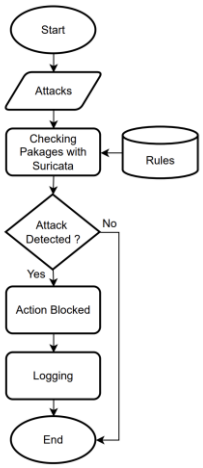


Fig. 4. IPS Flowchart

2.3. Suricata Rules Configuration

The initial step in building this system is installing Suricata in a prepared environment, serving as a crucial foundation for developing the threat detection system through proper configuration and rule implementation [24]. Since the default Ubuntu repository may not provide the latest version of Suricata, it is necessary to add the OISF (Open Information Security Foundation) repository to ensure access to the most recent officially supported version [25]. Adding the OISF repository updates the system's package list, enabling the download and installation of the latest stable version of Suricata, along with all necessary supporting components. Suricata is then configured to run automatically upon system startup using the command "sudo systemctl enable suricata.service", ensuring that the intrusion detection and prevention system remains active without requiring manual intervention [26], [27], [28].

By default, Suricata runs in IDS mode, which only detects and logs network activity without taking any blocking actions. Suricata can add a special ID, called the Community ID, to its JSON output. This Community ID helps match Suricata's event logs with datasets generated by other tools, like Elasticsearch. To do this, open the Suricata configuration file "/etc/suricata/suricata.yaml" and find the community-id setting. Enabling this ID makes it easier to integrate and correlate data across different network monitoring tools. A network interface is a device, either physical or virtual, that connects a computer to a network [29], [30]. In the context of Suricata, this interface is responsible for monitoring all incoming and outgoing network traffic.To enable IPS (Intrusion Prevention System) mode, adjustments need to be made so that Suricata not only logs harmful traffic but also stops or blocks it directly [31]. IPS mode is more proactive, as it allows Suricata to prevent attacks by stopping unwanted packets. nfqueue is a feature of the Netfilter subsystem in the Linux kernel that allows network traffic to be analyzed in user space through a queue. Suricata uses nfqueue to process network packets in real time, inspecting the traffic based on predefined rules, and then deciding whether the packet should be forwarded or blocked. When nfqueue is enabled, Suricata operates in IPS mode, meaning it not only detects threats but also blocks traffic deemed dangerous. Here are some important points to take note of from the output. The line

"Active: active (running)" indicates that Suricata has successfully restarted and is currently running. Additionally, the message "Starting suricata in IPS (nfqueue) mode… done." confirms that Suricata is now operating in IPS mode using nfqueue as shown on Fig. 5 [32], [33].

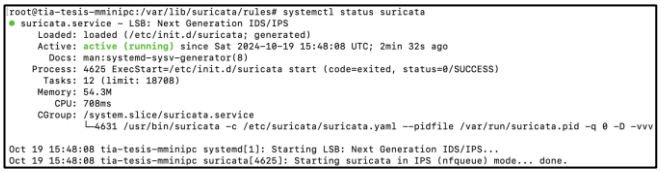


Fig. 5. Nfqueue IPS Mode

After IPS mode with nfqueue is activated on Suricata shows on Fig. 5. The next step is to ensure that network traffic is directed to Suricata for processing. Without this step, Suricata won't receive the traffic that needs to be inspected [34], [35], [36]. Uncomplicated Firewall (UFW) is a commonly used firewall in Ubuntu, and this configuration ensures that all incoming and outgoing traffic from the server is routed through Suricata via nfqueue refer to Fig. 6 and Fig. 7.

```
## Start Suricata NFQUEUE rules
-I INPUT 1 -p tcp --dport 22 -j NFQUEUE --queue-bypass
-I OUTPUT 1 -p tcp --sport 22 -j NFQUEUE --queue-bypass
-I FORWARD -j NFQUEUE
-I INPUT 2 -j NFQUEUE
-I OUTPUT 2 -j NFQUEUE
## End Suricata NFQUEUE rules
```

Fig. 6. Nfqueue Configuration

The rules on Fig. 6 mentioned above are the added NFQUEUE rules, allowing Suricata to effectively process network traffic [37], monitor and block any detected threats. Be sure to save these changes and restart the firewall service so that the new rules are fully applied. The first rule (-I INPUT 1 -p tcp –dport 22 -j NFQUEUE –queue-bypass) and the second rule (-I OUTPUT 1 -p tcp –sport 22 -j NFQUEUE –queue-bypass) act as a bypass for SSH connections (port 22), ensuring that SSH access to the server is not interrupted if Suricata fails or stops running. These rules are important because, without the bypass, SSH traffic could be blocked, cutting off remote access to the server. The next rule (-I FORWARD -j NFQUEUE) makes sure that all forwarded traffic passing through the server is also sent to Suricata for processing. Finally, the fourth (-I INPUT 2 -j NFQUEUE) and fifth (-I OUTPUT 2 -j NFQUEUE) rules direct all incoming and outgoing traffic, except SSH, to Suricata for analysis, ensuring that any non-bypassed packets are inspected by the system for threat detection and prevention.

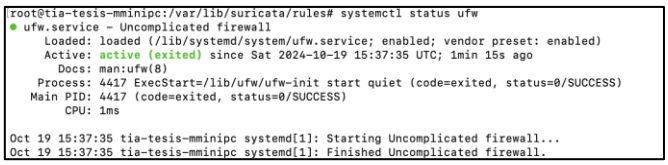


Fig. 7. UFW Status

Fig. 7 shows that the UFW (Uncomplicated Firewall) service has been successfully activated on the system. Its status is "active" which means the firewall is active and running properly. To protect web applications and networks from cyber threats, it's important to implement a security solution that can detect and stop attacks before they reach the server [38], [39]. Suricata is a powerful tool for this task, especially when configured with rules based on specific attack patterns. These rules are designed to detect common attack patterns, such as commands or terms typically used in injection or exploitation attacks on web applications. These rules can be created to detect various types of attacks, such as SQL Injection, Cross-Site Scripting (XSS), and Command Injection as shown on Fig. 8.

Fig. 8 displays a set of network security rules used in a Suricata-based Intrusion Prevention System (IPS). These rules are designed to detect and block SQL Injection attacks that could potentially harm web applications. Each rule instructs Suricata to monitor HTTP traffic and look for specific patterns commonly used in SQL Injection attacks, such as the "group", "order by", "union", "select" commands, among others. If these patterns are detected, the system will automatically block the traffic to protect the server from further exploitation. a set of rules used by Suricata to detect and block Cross-Site Scripting (XSS) attacks. These rules are designed to identify patterns commonly used in XSS attacks within HTTP traffic, where attackers inject

malicious code, such as JavaScript or specific HTML elements, to exploit vulnerabilities in web applications. Each rule is crafted to inspect URIs containing content like “script”, “src=”, “javascript”, “IMG”, “onerror” and “alert” and to block traffic that appears to be an XSS attack attempt. Fig.8 contains several rules configured in Suricata to detect and block Command Injection attacks. These rules work by analyzing incoming HTTP traffic to the server and looking for system command patterns commonly used in command injection attempts, such as “whoami,” “sleep,” “uname,” and others. If these patterns are detected in the HTTP request URI, the rules will block the request, protecting the server from executing malicious commands.

Fig. 8. Rules Configuration (SQL Injection, XSS and Command Injection)

These Suricata rules are designed to detect and block threats in HTTP traffic using the “drop” action, which automatically blocks malicious packets. These rules are universally applicable to all IP addresses and ports (any any -> any any) and focus on traffic directed to the server (to_server) [40]. Each rule includes a descriptive message (msg) to explain the identified threat, such as “TIA-SQL Injection Group by” for SQL Injection, “TIA-XSS Injection Script” for XSS, and “TIA-Command Injection Whoami” for Command Injection. Detection is based on specific content within the HTTP URI (http_uri), such as “group by,” “script,” or “whoami,” which are commonly used in attacks The rules are case-insensitive (nocase), have unique Signature IDs (SIDs), and revision numbers (rev) for tracking updates [41], [42].

2.4. ELK Stack

In network traffic monitoring and cybersecurity management, particularly in intrusion detection systems, the process of logging and data analysis is crucial. To effectively process the data generated by security systems, a clear workflow is necessary for data collection, processing, storage, and visualization. Suricata, an intrusion detection system, produces raw data logs that need further processing and analysis to be useful for network administrators in detecting threats and anomalies. For diagram outlines the flow refer to Fig. 9.

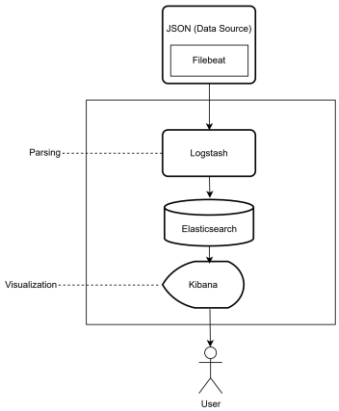


Fig. 9. ELK Stack Monitoring

The diagram outlines the flow of log from Suricata through the Elastic Stack for monitoring purposes shows on Fig. 9 Suricata generates logs in JSON format, recording network activity and potential security threats detected. These JSON logs, representing raw data from Suricata, are then collected by Filebeat, a lightweight data shipping tool. Filebeat is responsible for forwarding these logs to Logstash for further processing. Acting as a gateway in this pipeline, Filebeat ensures that the logs are efficiently transferred to the next stage for parsing [4]. For Docker status refer to Fig. 10.

Fig. 10 indicates that Docker is active and running, as shown by the status active (running). In the subsequent stage, the logs reach Logstash, which is designed for parsing. Within Logstash, the raw logs are processed, structured, and optimized for analysis by extracting useful information and applying necessary

transformations to make them more meaningful and easier to analyze. This parsing process is essential for transforming raw data into a more structured form, ready to be stored in Elasticsearch. Once the data is in Elasticsearch, users can utilize it for search, analysis, and reporting. At the end of the flow, Kibana provides a visual representation of the data stored in Elasticsearch, allowing users to view analysis results in an easy-to-understand format, such as graphs and interactive dashboards [43], [44], [45].

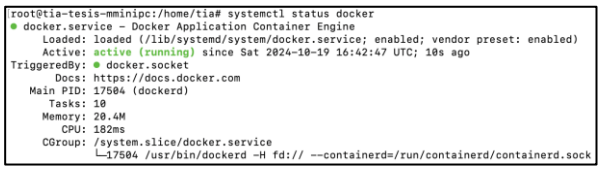


Fig. 10. Docker Status

3. RESULTS AND DISCUSSION

3.1. Validation and Analysis of Suricata JSON Logs and Elasticsearch Logs: Signature-Based Detection and Traffic Blocking Test

The aim of this test is to ensure that Suricata can detect and block harmful network attacks, such as SQL Injection, XSS, and command injection. The test is conducted by sending network traffic containing attack patterns and verifying if Suricata can recognize them according to the preconfigured rules. This test will evaluate suricata can detect the attacks based on the configured signatures and suricata successfully blocks the attacks according to the set rules shown on Fig. 11.

Fig. 11. Detection Test Based on Signatures in Suricata (SQL Injection Curl, XSS Curl and Command Injection Curl)

The attack attempt was made by accessing the URL 10.66.66.6 using a query that contained the pattern (+group+BY+1--+-), indicating an attempt at SQL Injection. This attack aims to exploit SQL queries through URL parameters shows on Fig. 11. The curl command was used to send a request to the address 10.66.66.6 with a querystring (%3Cimg%20src%3Dx%20onerror%3Djavascript%3Aalert%28%27XSS%27%29%3B%3E) that included an XSS (Cross-Site Scripting) attack. The attack pattern utilized a JavaScript script intended to trigger an alert in the browser. The curl command was used to send a request to the address 10.66.66.6, with a query that attempted to execute a command injection attack. The query was (?query=%3B%20whoami) aimed at running the whoami payload on the target server. The Fast log from Suricata can be seen in Fig. 12.

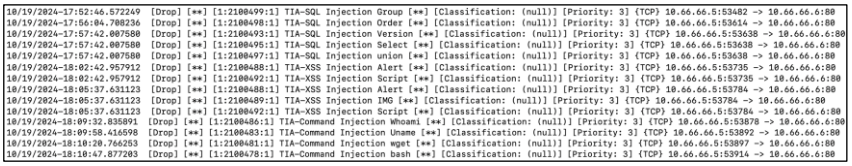


Fig. 12. Fast Log Suricata (SQL Injection, XSS and Command Injection)

Based on the logs shows on Fig. 12. The Suricata system has been properly configured to detect and block SQL Injection, XSS, and Command Injection attacks. All attack attempts recorded in the logs were successfully blocked according to the rules, demonstrating the system's effectiveness in preventing threats. This test ensures that Suricata's logs align with those stored in Elasticsearch by comparing key parameters such as timestamps,

alert signatures, HTTP URLs, and IP addresses. The JSON log from Suricata is verified against the Elasticsearch log to confirm the consistency of detection results, as shown in Fig. 13.

{"timestamp":"2024-10-19T17:56:04.708236+0000","flow_id":1233802994324942,"event_type":"alert","src_ip":"10.66.66.5","src_port":53614,"dest_ip":"10.66.66.6","dest_port":80,"proto":"TCP","pkt_src":"wire/pcap","community_id":"1:sqUrEfOZ4Lu/WnVY1jj5ch8N5VM=","tx_id":0,"alert":{"action":"blocked","gid":1,"signature_id":2100498,"rev":1,"signature":"TIA-SQL Injection Order","category":"","severity":3,"metadata":{"created_at":["2024_09_28"],"update_at":["2024_09_28"]}},"http":{"hostname":"10.66.66.6","url":"/?query='1'+ORDER+BY+1--+-","http_user_agent":"tesis","http_method":"GET","protocol":"HTTP/1.1","length":0},"app_proto":"http","direction":"to_server","flow":{"pkts_toserver":3,"pkts_toclient":1,"bytes_toserver":261,"bytes_toclient":60,"start":"2024-10-19T17:56:04.680483+0000","src_ip":"10.66.66.5","dest_ip":"10.66.66.6","src_port":53614,"dest_port":80}}

Fig. 13. JSON Log

JSON log from Suricata on Fig. 13 is compared with the log in Elasticsearch shows on Fig. 14 to verify if the example for Fig. 12 the SQL Injection attack detection results match. The Suricata log recorded the attack time on October 19, 2024, at 17:56:04 UTC, indicating that the system uses Universal Time Coordinated (UTC). Suricata detected the SQL Injection attack with the signature "TIA: SQL Injection Order," where the attacked URL contains the query /?query=1+ORDER+BY+1--. The source IP address of the attack was 10.66.66.5, and the destination IP was 10.66.66.6.

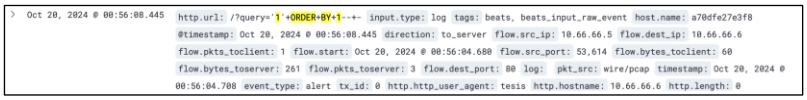


Fig. 14. Elasticsearch Log

In Elasticsearch, as shown in Fig. 14, the recorded time was October 20, 2024, at 00:56:08 GMT+7, with the log capturing the same attack signature (/?query=1+ORDER+BY+1–) and identical source and destination IP addresses (10.66.66.5 and 10.66.66.6). The test results confirm that Suricata's detection and logging system functions properly, and the logs forwarded to Elasticsearch are processed without any loss or alteration of data. Both logs consistently reflect the same attack, with the time difference solely caused by the difference in time zone settings.

3.2. Intrusion Prevention System (IPS) Suricata on Damn Vulnerable Web Application (DVWA)

DVWA (Damn Vulnerable Web Application) is a web application designed with security vulnerabilities such as SQL Injection, XSS, and Command Injection, allowing easy exploitation for testing purposes. Suricata is used to detect and monitor these attacks on the network. The testing aims to ensure that Suricata can detect, block, and log attacks through DVWA while providing protection for vulnerable applications. The testing steps include accessing DVWA through a browser connected to Suricata and setting the DVWA security level to low to facilitate exploitation, as shown in Fig. 15 [3].

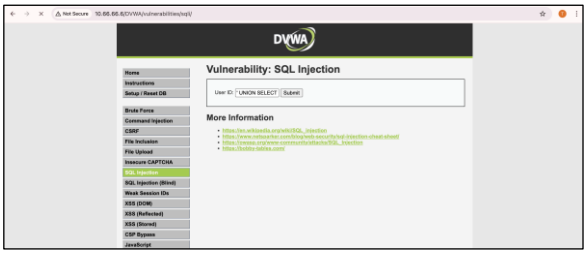


Fig. 15. DVWA (Vulnerability)

Once the configuration is complete, testing begins by selecting the type of vulnerability in DVWA as shown in Fig. 15. Payloads, such as SQL commands or malicious scripts, are entered into the provided fields. If the connection to DVWA is interrupted, it indicates that Suricata has successfully detected and blocked the attacking IP [46], [47]. The system's effectiveness is measured as the percentage of malicious payloads successfully blocked compared to the total payloads tested. This percentage reflects the system's ability to detect and mitigate threats, with higher values indicating better performance, calculated using the following formula:

$$Effectiveness = \frac{Total\ Blocked\ Payloads}{Total\ Payloads}\ x\ 100\% \quad (1)$$

Based on the testing of 300 payloads, consisting of 100 payloads for each type of attack SQL Injection, Cross-Site Scripting (XSS) and Command Injection the IPS was able to detect and block all attacks with 100% effectiveness shows on Fig. 16. In the SQL Injection test, every payload was successfully identified and stopped, demonstrating the IPS's ability to prevent exploits that could lead to data leaks or manipulation in the database. For XSS attacks the system successfully detected and blocked all attempts to execute malicious scripts in the browser, protecting users and applications from the threat of information theft or web page manipulation. Similarly, in the Command Injection test, the IPS successfully prevented each attempt to execute direct commands on the server, which could potentially jeopardize system control. Overall, these results show the IPS is highly effective at detecting and mitigating these three types of attacks with 100% effectiveness. The system is capable of maintaining the security and integrity of web applications, protecting sensitive data, and responding swiftly to threats

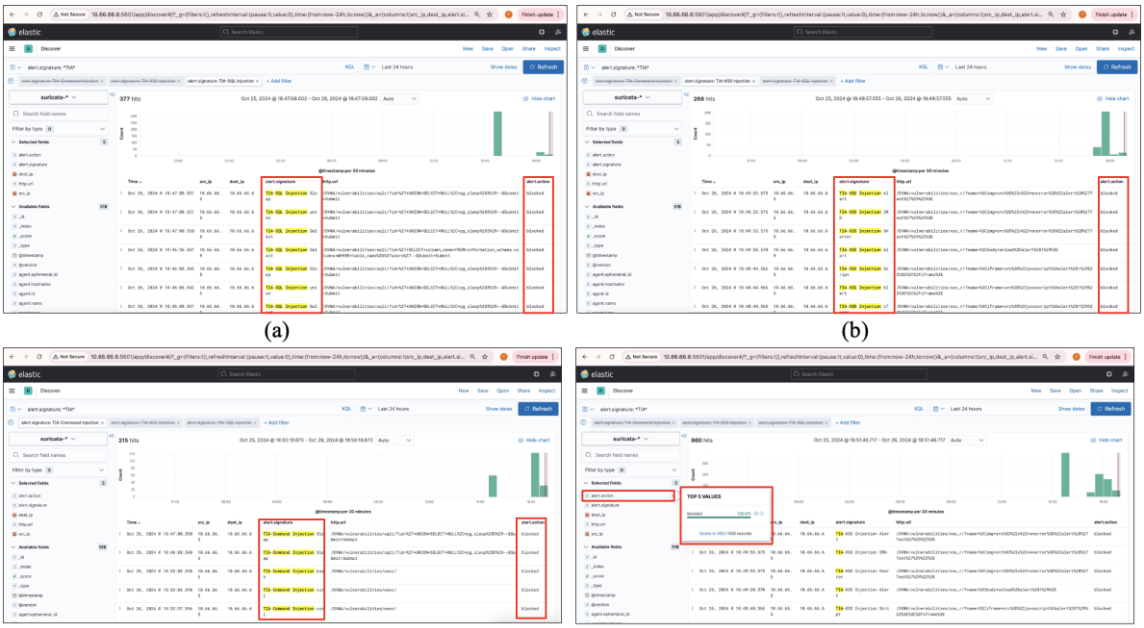


Fig. 16. Elasticsearch Logs Indicate All Traffic Blocked (a) SQL Injection (b) XSS (c) Command Injection (d) All Payloads Status

3.3. Analysis of Flow Bytes and Response Time on SQL Injection, XSS and Command Injection Payloads in DVWA

This testing aims to analyze network data flow (flow bytes) and system response times against SQL Injection, XSS, and Command Injection attacks using DVWA. The observed logs in Elasticsearch are utilized to understand byte patterns between the client and server and the execution time of each payload to identify system vulnerabilities [48]. The analyzed parameters include Flow Bytes To Server (bytes sent from the client to the server), Flow Bytes To Client (bytes in the server's response), Flow Start (time communication begins), Timestamp (time a threat is detected), and Time Differences (time gap between Flow Start and Timestamp), which indicates the system's efficiency in detecting threats, especially for complex payloads.

The test results show on Table 2 Suricata effectively detects and blocks all types of attack payloads, including SQL Injection, XSS, and Command Injection. Flow Bytes to Server demonstrates an increase in line with the complexity of the payload, where more complex payloads generate a higher volume of data sent to the server. Meanwhile, Flow Bytes to Client consistently remains low, with most payloads producing only 60 bytes, indicating that the system successfully limits server responses to protect sensitive data. Additionally, the average Time Differences of 0.902 seconds indicates that the system is capable of detecting threats quickly, even for payloads requiring more complex processing. The effectiveness of Suricata is evident in its ability to not only detect but also comprehensively block all payloads, ensuring network security and preventing data leaks.

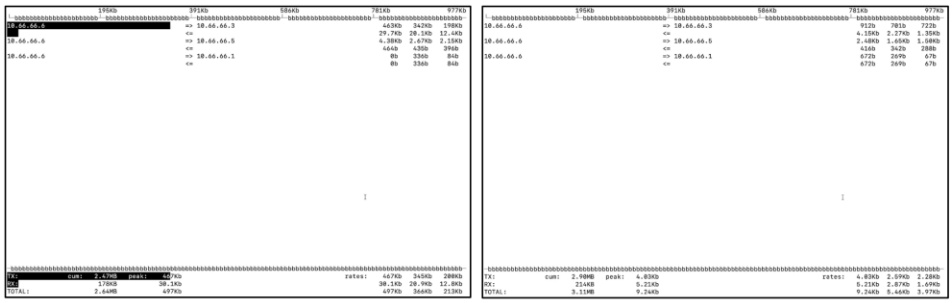
Table 2. SQL Injection, XSS and Command Injection Payloads on Flow Bytes and Response Time

No.	Payloads	Flow Bytes to Server	Flow Bytes to Client	Flow start	Timestamp	Time Differences (Seconds)
1	') GROUP BY id--	778	60	15:26:00.980	15:26:01.022	0.042
2	' OR 1=1 GROUP BY username, password--	802	60	15:42:45.144	15:42:45.433	0.289
3	' UNION SELECT 1,2, pg_sleep(5)--	798	60	16:00:06.790	16:00:07.852	1.062
4	' SELECT @@version--	781	60	16:17:04.755	16:17:06.000	1.245
5	' OR 1=1; pg_sleep(5)--	788	60	16:27:24.993	16:27:25.557	0.564
6	<script>console.log('XSS')</script>	801	60	19:36:27.184	19:36:28.341	1.157
7	javascript:alert(1)	773	60	19:44:03.989	19:44:04.550	0.561
8	<body onload=alert(1)>	780	60	21:18:48.288	21:18:49.998	1.710
9	<script>alert(document.cookie)</script>	801	60	19:31:05.844	19:31:06.199	0.355
10	<input onfocus="alert(1) ">	788	60	22:04:45.445	22:04:46.234	0.789
11	$(whoami) && sleep 10	911	60	21:45:24.458	21:45:25.427	0.969
12	sleep 10; uname -r	1003	120	23:29:34.522	23:29:35.751	1.229
13	uname -s && whoami	953	60	00:04:42.384	00:04:43.736	1.352
14	curl http://attacker.com && sleep 10	1030	120	11:30:55.859	11:30:57.111	1.252
15	wget http://attacker.com/shell.sh | bash	1110	120	11:01:02.301	11:01:03.261	0.960
					Average	0.902

3.4. Analysis of Network Aspects Using Suricata IPS and iftop for Traffic Monitoring

This testing aims to analyze network performance, security, and stability. Performance is measured by the volume of outgoing (TX) and incoming (RX) traffic during simple and complex HTTP requests. Network security is evaluated using Suricata IPS to detect threats such as SQL injection, while stability is assessed by comparing traffic patterns from both types of requests. The test environment includes a target server (IP 10.66.66.6), Suricata as the IPS, and iftop for traffic monitoring. The testing is conducted in two scenarios: a simple HTTP request using the command curl 10.66.66.6 --user-agent 0 and a complex request with suspicious queries using curl --url "10.66.66.6?query='+group BY+1--+-" --max-time 5 -v --user-agent "tesis". The observed parameters in this testing include outgoing traffic (TX), which is the volume of data sent from the server incoming traffic (RX), which is the volume of data received by the server and Suricata's response in detecting and mitigating threats. The observations from these two scenarios provide insights into the performance, security, and stability of the network when handling various types of HTTP requests as shown on Fig. 17.

Fig.17 (a) shows using User-Agent 0 with a simple HTTP request, the outgoing traffic (TX) volume was recorded at 2.47 MB, with a peak rate of 467 Kb. Meanwhile, the incoming traffic (RX) was relatively small, at only 178 KB, with a peak rate of 30.1 Kb. Observations indicate that this simple request generated predominantly outgoing traffic, while the server response was minimal (low RX traffic). Suricata likely did not log or intervene in this request as no suspicious patterns were detected. The traffic appeared stable with no significant spikes. Fig. 17 (b) shows using User-Agent "tesis" and a more complex query string, the test results showed an increase in traffic compared to the first scenario. The outgoing traffic (TX) volume was recorded at 2.99 MB, with a peak rate of 4.03 Kb, while incoming traffic (RX) increased to 214 KB, with a peak rate of 5.21 Kb. The increase in RX traffic indicates that the server provided a more complex response, possibly due to processing a heavy query like group BY, which Suricata may flag as an SQL Injection threat. TX traffic also increased as the request required additional data to be sent to the server.



(a) (b)

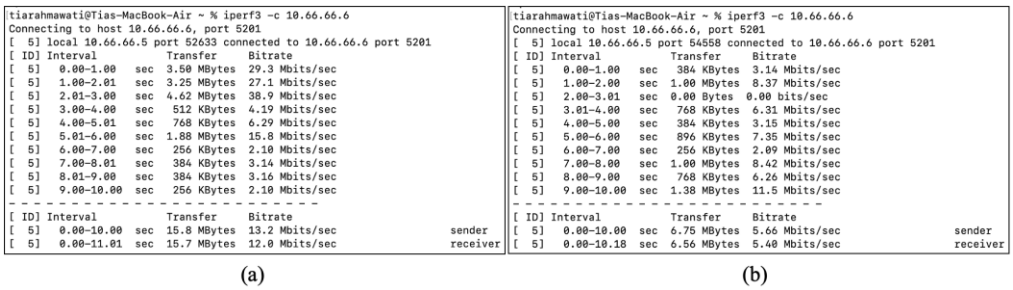
Fig. 17. Iftop (a) HTTP Request (b) Complex Query String

The analysis indicates the complex query can be identified by Suricata as a potential threat, such as an SQL injection. This enables Suricata to trigger preconfigured rules, either by blocking the request or logging it for further analysis. In contrast for HTTP Request, the traffic appears lighter and does not activate Suricata, demonstrating the system's ability to differentiate between normal and suspicious requests. This confirms Suricata's effectiveness in identifying suspicious request patterns.

3.5. Evaluation of Network Traffic Performance and IPS Effectiveness in Mitigating Slow HTTP Attacks

This Test aims to evaluate network performance under normal conditions and during an attack using the Slow HTTP Test method, as well as measure the effectiveness of IPS Suricata in detecting and mitigating attacks. The testing utilizes iperf3 to measure data transfer and network bitrate [49], [50], both before and after the attack. The testing environment consists of a Target Server (IP 10.66.66.6) running a vulnerable application (DVWA), IPS Suricata for detecting and blocking attacks, and a Client (IP 10.66.66.5) as the testing device. The Slow HTTP Test command (slowhttptest -c 1000 -H -g -o slowhttp -i 10 -r 200 -t GET -u http://10.66.66.6/DVWA/login.php -x 24 -p 3) is utilized to simulate high-traffic conditions or simultaneous attacks on the target server. This test exerts pressure on the server by opening up to 1000 simultaneous connections (-c 1000) and generating 200 new requests per second (-r 200), while maintaining the connections for 24 seconds (-x 24). The Slowloris attack technique used (-H) keeps the server occupied by slowly processing HTTP headers at 10-second intervals (-i 10), simulating a real-world attack on the target login page (http://10.66.66.6/DVWA/login.php). The login page is an ideal target because disruptions to authentication functionality can cripple system operations.

The testing process involves three stages, first measuring network performance without an attack, launching a Slow HTTP Test attack to disrupt the server and re-measuring network performance after attack mitigation by IPS Suricata. The observed parameters include Transfer Data (volume of data successfully sent), Bitrate (data transfer speed in Mbits/sec), and IPS Efficiency (the system's ability to block attacks without disrupting server functionality) refer to Fig. 18.



```
tiarahmawati@Tias-MacBook-Air ~ % iperf3 -c 10.66.66.6
Connecting to host 10.66.66.6, port 5201
[  5] local 10.66.66.5 port 52633 connected to 10.66.66.6 port 5201
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  3.50 MBytes  29.3 Mbits/sec
[  5]   1.00-2.01   sec  3.25 MBytes  27.1 Mbits/sec
[  5]   2.01-3.00   sec  4.62 MBytes  38.9 Mbits/sec
[  5]   3.00-4.00   sec   512 KBytes  4.19 Mbits/sec
[  5]   4.00-5.01   sec   768 KBytes  6.29 Mbits/sec
[  5]   5.01-6.00   sec  1.88 MBytes  15.8 Mbits/sec
[  5]   6.00-7.00   sec   256 KBytes  2.10 Mbits/sec
[  5]   7.00-8.01   sec   384 KBytes  3.14 Mbits/sec
[  5]   8.01-9.00   sec   384 KBytes  3.16 Mbits/sec
[  5]   9.00-10.00  sec   256 KBytes  2.10 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.00  sec  15.8 MBytes  13.2 Mbits/sec                  sender
[  5]   0.00-11.01  sec  15.7 MBytes  12.0 Mbits/sec                  receiver
```

(a)

```
tiarahmawati@Tias-MacBook-Air ~ % iperf3 -c 10.66.66.6
Connecting to host 10.66.66.6, port 5201
[  5] local 10.66.66.5 port 54558 connected to 10.66.66.6 port 5201
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec   384 KBytes  3.14 Mbits/sec
[  5]   1.00-2.00   sec  1.00 MBytes  8.37 Mbits/sec
[  5]   2.00-3.01   sec  0.00 Bytes  0.00 bits/sec
[  5]   3.01-4.00   sec   768 KBytes  6.31 Mbits/sec
[  5]   4.00-5.00   sec   384 KBytes  3.15 Mbits/sec
[  5]   5.00-6.00   sec   896 KBytes  7.35 Mbits/sec
[  5]   6.00-7.00   sec   256 KBytes  2.09 Mbits/sec
[  5]   7.00-8.00   sec  1.00 MBytes  8.42 Mbits/sec
[  5]   8.00-9.00   sec   768 KBytes  6.26 Mbits/sec
[  5]   9.00-10.00  sec  1.38 MBytes  11.5 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.00  sec  6.75 MBytes  5.66 Mbits/sec                  sender
[  5]   0.00-10.18  sec  6.56 MBytes  5.40 Mbits/sec                  receiver
```

(b)

Fig. 18. Iperf3 (a) Normal Network Traffic (b) During-Attack Traffic

When an attack using the Slow HTTP method was launched against the server protected by IPS Suricata, data transfer and bitrate dropped significantly shows on Fig. 18, the total data transfer decreased to 6.75 MB from the sender and 6.56 MB received by the receiver, with an average bitrate of 5.66 Mbits/sec for the sender and 5.40 Mbits/sec for the receiver, indicating a drop of more than 50% compared to normal conditions. During the 2.00–3.01 second interval, no data was transferred (0.00 bits/sec), indicating that Suricata successfully detected and completely blocked the attack. Afterward, the traffic resumed, although at a lower bitrate than in

normal conditions. The correlation between traffic reduction and Suricata's effectiveness is clearly evident. The IPS successfully blocked suspicious traffic, keeping the server operational without downtime. The decrease in bitrate and data transfer indicates disruption caused by the attack, but Suricata effectively minimized its impact. This testing proves that Suricata is effective in detecting and mitigating Slow HTTP attacks, although further system security enhancements and real-time monitoring are still recommended.

3.6. Performance Comparison of CPU and Memory Utilization

The CPU and memory testing on the web server aims to compare processor and memory usage between normal conditions and when the web server is under attack, specifically from SQL injection, XSS, and command injection attacks. The method involves monitoring resource usage (CPU and memory) on the web server when no attack is present and then comparing it to the conditions during an attack. The goal of this testing is to assess the impact of the attack on the web server's performance and to understand the extent of the increase in CPU and memory load during the attack.

Based on the test results in Table 3 CPU and memory usage increased significantly after the attack. Before the attack, the average CPU usage was only 0.7%, but it rose to an average of 20.7% after the attack, peaking at 61.1%. Memory usage also increased from 0.4% to 4.8%. This increase was due to the additional workload involved in threat detection and blocking processes. These results indicate that the attack had a substantial impact on resource consumption, particularly CPU usage, highlighting the need for increased capacity or system optimization to maintain performance when facing future attacks.

Table 3. Perfomance Results

Trial	Component	Pre-Attack (%)	During-Attack (%)	Trial	Component	Pre-Attack (%)	During-Attack (%)
1	Processor CPU	0.7	21.2	6	Processor CPU	0.7	38.8
	Memory	0.4	4.8		Memory	0.4	4.8
2	Processor CPU	0.7	23.3	7	Processor CPU	0.7	39.9
	Memory	0.4	4.8		Memory	0.4	4.8
3	Processor CPU	0.7	25.5	8	Processor CPU	0.7	45.3
	Memory	0.4	4.8		Memory	0.4	4.8
4	Processor CPU	0.7	27.3	9	Processor CPU	0.7	47.2
	Memory	0.4	4.8		Memory	0.4	4.8
5	Processor CPU	0.7	36.4	10	Processor CPU	0.7	61.1
	Memory	0.4	4.8		Memory	0.4	4.8
					Average	0.55%	20.7%

In the previous study titled "Integrated Security Information and Event Management (SIEM) with Intrusion Detection System (IDS) for Live Analysis based on Machine Learning" by Adabi Muhammad, Parman Sukarno, and Aulia Wardana [5], the system was capable of detecting attacks within network traffic but did not demonstrate the ability to prevent attacks directly. The system functioned solely to detect attack logs using IDS integrated with SIEM. This study required medium-to-high specification hardware due to the high resource consumption by Elasticsearch, which utilized up to 78% of CPU and 2300 MB of RAM. The system employed machine learning to detect and classify attacks based on IDS logs within the network. In contrast, the findings in this study demonstrated 100% effectiveness in blocking web application-based attacks, such as SQL Injection, XSS, and Command Injection, through specifically designed rules. Suricata was configured to recognize attack patterns directly, achieving an average detection time of 0.902 seconds. This system proved to be efficient and capable of operating on low-specification devices, thanks to the integration of Filebeat, Elasticsearch, Logstash, and Kibana within a single Docker container. This research focuses on preventing attacks through an IPS based on rules designed to detect malicious payloads in real-time. The CPU usage increased to an average of 20.7%, which remains within acceptable limits for low-resource devices.

4. CONCLUSION

This research successfully implemented an Intrusion Prevention System (IPS) using Suricata to enhance web server security against threats such as SQL Injection, Cross-Site Scripting (XSS), and Command Injection.

The rules configured in Suricata were able to detect and read attacks like SQL Injection, XSS, and Command Injection and effectively block these threats. These rules were specifically designed to recognize attack patterns in real-time and take preventive action to protect the system, with the average Time Differences demonstrating the system's efficiency in detecting and processing malicious payloads at 0.902 seconds.Following penetration testing using the Damn Vulnerable Web Application (DVWA), all SQL Injection, XSS, and Command Injection attack payloads were successfully blocked with 100% effectiveness. Flow Bytes To Server ranged between 773 and 1110 bytes, influenced by payload complexity, while Flow Bytes To Client remained consistent at 60 bytes, with occasional increases to 120 bytes. The system effectively restricted server responses, ensuring the prevention of sensitive data leaks.

The testing confirmed the rules were effective and consistent in detecting and blocking attacks. Integration of Filebeat, Logstash, Elasticsearch, and Kibana within a Docker container streamlined log management and real-time analysis. Suricata logs in JSON format were efficiently processed and visualized, enabling administrators to monitor threats seamlessly. Enhancing detection capabilities with adaptive rules using machine learning or behavioral analysis is recommended, along with real-time alert systems via email or messaging. Expanding Elasticsearch's historical log storage capacity is also suggested for comprehensive trend analysis and proactive security improvements.

Acknowledgments

The authors express their gratitude for the financial support provided by Indonesia's DRTPM, DITJEN DIKTIRISTEK, and KEMDIKBUDRISTEK through grants 043/SP2H/RT-MONO/LL4/2024 and 088/LIT07/PPM-LIT/2024. This support is gratefully acknowledged and appreciated.

REFERENCES

[1] R. A. Muzaki, O. C. Briliyant, M. A. Hasditama, and H. Ritchi, "Improving Security of Web-Based Application Using ModSecurity and Reverse Proxy in Web Application Firewall," in *2020 International Workshop on Big Data and Information Security, IWBIS*, pp. 85–90, Oct. 2020, https://doi.org/10.1109/IWBIS50925.2020.9255601.

[2] A. Fadhlillah, N. Karna, and A. Irawan, "IDS Performance Analysis using Anomaly-based Detection Method for DOS Attack," in *IoTaIS 2020 - Proceedings: 2020 IEEE International Conference on Internet of Things and Intelligence Systems*, pp. 18–22, Jan. 2021, https://doi.org/10.1109/IoTaIS50849.2021.9359719.

[3] T. Rahmawati, R. W. Shiddiq, M. Sumpena, S. Setiawan, N. Karna, and S. Hertiana, "Web Application Firewall Using Proxy and Security Information and Event Management for OWASP Cyber Attack Detection," *IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS))*, pp. 280–285, Nov. 2023, https://doi.org/10.1109/IoTaIS60147.2023.10346051.

[4] F. Ahmed, U. Jahangir, H. Rahim, and K. Ali, "Centralized Log Management Using Elasticsearch, Logstash and Kibana," *International Conference on Information Science and Communication Technology*, pp. 1–7, 2020, https://doi.org/10.1109/ICISCT49550.2020.9080053.

[5] A. R. Muhammad, P. Sukarno, and A. A. Wardana, "Integrated Security Information and Event Management (SIEM) with Intrusion Detection System (IDS) for Live Analysis based on Machine Learning," in *Procedia Computer Science*, pp. 1406–1415, 2022, https://doi.org/10.1016/j.procs.2022.12.339.

[6] A. Coscia, V. Dentamaro, S. Galantucci, A. Maci, and G. Pirlo, "Automatic decision tree-based NIDPS ruleset generation for DoS/DDoS attacks," *Journal of Information Security and Applications*, vol. 82, May 2024, https://doi.org/10.1016/j.jisa.2024.103736.

[7] S. Adiwal, B. Rajendran, P. S. D., and S. D. Sudarsan, "DNS Intrusion Detection (DID) — A SNORT-based solution to detect DNS Amplification and DNS Tunneling attacks," *Franklin Open*, vol. 2, p. 100010, Mar. 2023, https://doi.org/10.1016/j.fraope.2023.100010.

[8] A. Wiranata, N. Karna, A. Irawan, and A. I. Prakoso, "Implementation and Analysis of Network Security in Raspberry Pi against DOS Attack with HIPS Snort," *International Conference on Computer Science, Information Technology and Engineering (ICCoSITE)*, pp. 892-896, 2023, https://doi.org/https://doi.org/10.1109/ICCoSITE57641.2023.10127741.

[9] K. Barik and S. Misra, "IDS-Anta: An open-source code with a defense mechanism to detect adversarial attacks for intrusion detection system," *Software Impacts*, vol. 21, Sep. 2024, https://doi.org/10.1016/j.simpa.2024.100664.

[10] M. R. Ahmed and F. M. Ali, "Enhancing Hybrid Intrusion Detection and Prevention System for Flooding Attacks Using Decision Tree," *2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pp. 1–4, 2019, https://doi.org/10.1109/ICCCEEE46830.2019.9071191.

[11] A. Paul, V. Sharma, and O. Olukoya, "SQL injection attack: Detection, prioritization & prevention," *Journal of Information Security and Applications*, vol. 85, Sep. 2024, https://doi.org/10.1016/j.jisa.2024.103871.

[12] I. T. Wibowo, A. Kurniawan, N. F. Sulaiman, P. Oktivasari, "Design and Implementation of Cloud Computing Using the NDLC Method Combined with Tunnel Link Split," in *Proceeding - International Conference on Information Technology and Computing, ICITCOM*, pp. 131–135. 2023, https://doi.org/10.1109/ICITCOM60176.2023.10442875.

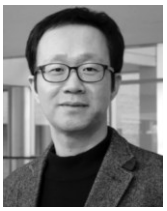
[13] Z. Noor, S. Hina, F. Hayat, and G. A. Shah, "An intelligent context-aware threat detection and response model for smart cyber-physical systems," *Internet of Things (Netherlands)*, vol. 23, Oct. 2023, https://doi.org/10.1016/j.iot.2023.100843.
[14] M. H. Nasir, J. Arshad, and M. M. Khan, "Collaborative device-level botnet detection for internet of things," *Comput Secur*, vol. 129, Jun. 2023, https://doi.org/10.1016/j.cose.2023.103172.
[15] D. Bhatnagar, R. J. Subalakshmi, and C. Vanmathi, "Twitter Sentiment Analysis Using Elasticsearch, LOGSTASH and KIBANA," in *International Conference on Emerging Trends in Information Technology and Engineering, ic-ETITE*, pp. 1-5, Feb. 2020. https://doi.org/10.1109/ic-ETITE47903.2020.351.
[16] M. M. Rahman, S. Al Shakil, and M. R. Mustakim, "A survey on intrusion detection system in IoT networks," *Cyber Security and Applications*, p. 100082, 2024, https://doi.org/10.1016/j.csa.2024.100082.
[17] M. A. Hossain and M. S. Islam, "Ensuring network security with a robust intrusion detection system using ensemble-based machine learning," *Array*, vol. 19, Sep. 2023, https://doi.org/10.1016/j.array.2023.100306.
[18] R. A. Abed, E. K. Hamza, and A. J. Humaidi, "A modified CNN-IDS model for enhancing the efficacy of intrusion detection system," *Measurement: Sensors*, vol. 35, Oct. 2024, https://doi.org/10.1016/j.measen.2024.101299.
[19] F. Younas, A. Raza, N. Thalji, L. Abualigah, R. A. Zitar, and H. Jia, "An efficient artificial intelligence approach for early detection of cross-site scripting attacks," *Decision Analytics Journal*, vol. 11, Jun. 2024, https://doi.org/10.1016/j.dajour.2024.100466.
[20] P. Nespoli, D. Díaz-López, and F. Gómez Mármol, "Cyberprotection in IoT environments: A dynamic rule-based solution to defend smart devices," *Journal of Information Security and Applications*, vol. 60, Aug. 2021, https://doi.org/10.1016/j.jisa.2021.102878.
[21] A. Bhardwaj, S. Bharany, A. Almogren, A. Ur Rehman, and H. Hamam, "Proactive threat hunting to detect persistent behaviour-based advanced adversaries," *Egyptian Informatics Journal*, vol. 27, Sep. 2024, https://doi.org/10.1016/j.eij.2024.100510.
[22] M. A. Hussain *et al.*, "Provably throttling SQLI using an enciphering query and secure matching," *Egyptian Informatics Journal*, vol. 23, no. 4, pp. 145–162, Dec. 2022, https://doi.org/10.1016/j.eij.2022.10.001.
[23] J. Jung, T. Oh, I. Kim, and S. Park, "Open-sourced real-time visualization platform for traffic simulation," in *Procedia Computer Science*, pp. 243–250, 2023, https://doi.org/10.1016/j.procs.2023.03.033.
[24] X. Huang *et al.*, "Clean: Minimize Switch Queue Length via Transparent ECN-proxy in Campus Networks," in *2021 IEEE/ACM 29th International Symposium on Quality of Service, IWQOS*, pp. 1-6, Jun. 2021. https://doi.org/10.1109/IWQOS52092.2021.9521295.
[25] S. Alem, D. Espes, L. Nana, E. Martin, and F. De Lamotte, "A novel bi-anomaly-based intrusion detection system approach for industry 4.0," *Future Generation Computer Systems*, vol. 145, pp. 267-283, 2023, https://doi.org/10.1016/j.future.2023.03.024.
[26] N. Negm *et al.*, "Tasmanian devil optimization with deep autoencoder for intrusion detection in IoT assisted unmanned aerial vehicle networks," *Ain Shams Engineering Journal*, vol. 15, no. 11, p. 102943, Nov. 2024, https://doi.org/10.1016/j.asej.2024.102943.
[27] F. Ullah, S. Ullah, G. Srivastava, and J. C. W. Lin, "IDS-INT: Intrusion detection system using transformer-based transfer learning for imbalanced network traffic," *Digital Communications and Networks*, vol. 10, no. 1, pp. 190–204, Feb. 2024, https://doi.org/10.1016/j.dcan.2023.03.008.
[28] I. S. Crespo-Martínez, A. Campazas-Vega, Á. M. Guerrero-Higueras, V. Riego-DelCastillo, C. Álvarez-Aparicio, and C. Fernández-Llamas, "SQL injection attack detection in network flow data," *Comput Secur*, vol. 127, Apr. 2023, https://doi.org/10.1016/j.cose.2023.103093.
[29] T. O. Ojewumi, G. O. Ogunleye, B. O. Oguntunde, O. Folorunsho, S. G. Fashoto, and N. Ogbu, "Performance evaluation of machine learning tools for detection of phishing attacks on web pages," *Sci Afr*, vol. 16, Jul. 2022, https://doi.org/10.1016/j.sciaf.2022.e01165.
[30] F. Wang, "Design of Computer Network Security Intrusion Prevention Strategy and Evaluation Algorithm Analysis Technology," in *Procedia Computer Science*, pp. 1270–1276, 2023, https://doi.org/10.1016/j.procs.2023.11.093.
[31] L. Shuai and S. Li, "Performance optimization of Snort based on DPDK and Hyperscan," in *Procedia Computer Science*, pp. 837–843, 2021, https://doi.org/10.1016/j.procs.2021.03.007.
[32] R. A. Abed, E. K. Hamza, and A. J. Humaidi, "A modified CNN-IDS model for enhancing the efficacy of intrusion detection system," *Measurement: Sensors*, vol. 35, Oct. 2024, https://doi.org/10.1016/j.measen.2024.101299.
[33] A. S. Alghawli, "Complex methods detect anomalies in real time based on time series analysis," *Alexandria Engineering Journal*, vol. 61, no. 1, pp. 549–561, Jan. 2022, https://doi.org/10.1016/j.aej.2021.06.033.
[34] M. Husák, M. Žádník, V. Bartoš, and P. Sokol, "Dataset of intrusion detection alerts from a sharing platform," *Data in Brief,* vol. 33, p. 106530*,* Nov. 2020, https://doi.org/10.17632/p6tym3fghz.1.
[35] J. Ye, W. Zhao, and D. Wang, "A Tool Design for SQL injection vulnerability detection based on improved crawler," in *Procedia Computer Science*, pp. 1331–1339, 2023, https://doi.org/10.1016/j.procs.2024.10.159.
[36] R. L. Alaoui and E. H. Nfaoui, "Web attacks detection using stacked generalization ensemble for LSTMs and word embedding," in *Procedia Computer Science*, pp. 687–696, 2022, https://doi.org/10.1016/j.procs.2022.12.070.
[37] A. Haydar and M. Ramparison, "Modeling Wazuh rules with Weighted Timed Automata," *Procedia Comput Sci*, vol. 251, pp. 75–82, 2024, https://doi.org/10.1016/j.procs.2024.11.086.

[38] H. Haugerud, H. N. Tran, N. Aitsaadi, and A. Yazidi, "A dynamic and scalable parallel Network Intrusion Detection System using intelligent rule ordering and Network Function Virtualization," *Future Generation Computer Systems*, vol. 124, pp. 254–267, Nov. 2021, https://doi.org/10.1016/j.future.2021.05.037.
[39] A. Adu-Kyere, E. Nigussie, and J. Isoaho, "Analyzing the effectiveness of IDS/IPS in real-time with a custom in-vehicle design," in *Procedia Computer Science*, pp. 175–183, 2024, https://doi.org/10.1016/j.procs.2024.06.013.
[40] T. S. Pooja, P. Shrinivasacharya, "Evaluating neural networks using Bi-Directional LSTM for network IDS (intrusion detection systems) in cyber security," *Global Transitions Proceedings*, vol. 2, no. 2, pp. 448–454, Nov. 2021, https://doi.org/10.1016/j.gltp.2021.08.017.
[41] Z. Chiba, N. Abghour, K. Moussaid, O. Lifandali, and R. Kinta, "A Deep Study of Novel Intrusion Detection Systems and Intrusion Prevention Systems for Internet of Things Networks," in *Procedia Computer Science*, pp. 94–103, 2022, https://doi.org/10.1016/j.procs.2022.10.124.
[42] D. Arnaldy and T. S. Hati, "Performance Analysis of Reverse Proxy and Web Application Firewall with Telegram Bot as Attack Notification on Web Server," in *2020 3rd International Conference on Computer and Informatics Engineering, IC2IE,* pp. 455–459, Sep. 2020, https://doi.org/10.1109/IC2IE50715.2020.9274592.
[43] T. Gaber, J. B. Awotunde, M. Torky, S. A. Ajagbe, M. Hammoudeh, and W. Li, "Metaverse-IDS: Deep learning-based intrusion detection system for Metaverse-IoT networks," *Internet of Things (Netherlands)*, vol. 24, Dec. 2023, https://doi.org/10.1016/j.iot.2023.100977.
[44] O. Nyarko-Boateng, I. K. Nti, A. A. Mensah, and E. K. Gyamfi, "Controlling user access with scripting to mitigate cyber-attacks," *Sci Afr*, vol. 26, Dec. 2024, https://doi.org/10.1016/j.sciaf.2024.e02355.
[45] A. C. Rus, M. El-Hajj, and D. K. Sarmah, "NAISS: A reverse proxy approach to mitigate MageCart's e-skimmers in e-commerce," *Comput Secur*, vol. 140, May 2024, https://doi.org/10.1016/j.cose.2024.103797.
[46] L. F. Sikos, "Packet analysis for network forensics: A comprehensive survey," *Forensic Science International: Digital Investigation*, vol. 32, p. 200892. 2020, https://doi.org/10.1016/j.fsidi.2019.200892.
[47] V. Devalla, S. Srinivasa Raghavan, S. Maste, J. D. Kotian, and D. Annapurna, "MURLi: A Tool for Detection of Malicious URLs and Injection Attacks," in *Procedia Computer Science*, pp. 662–676, 2022, https://doi.org/10.1016/j.procs.2022.12.068.
[48] O. Takaki, N. Hamamoto, A. Takefusa, S. Yokoyama, and K. Aida, "Implementation of Anonymization Algorithms for Log Data Analysis on a Cloud-Based Learning Management System," in *Procedia Computer Science*, pp. 3774–3784, 2023, https://doi.org/10.1016/j.procs.2023.10.373.
[49] M. A. Lawall, R. A. Shaikh, and S. R. Hassan, "A DDoS Attack Mitigation Framework for IoT Networks using Fog Computing," in *Procedia Computer Science*, pp. 13–20, 2021, https://doi.org/10.1016/j.procs.2021.02.003.
[50] T. Bajtoš, P. Sokol, and F. Kurimský, "Processing of IDS alerts in multi-step attacks [Formula presented]," *Software Impacts*, vol. 19, Mar. 2024, https://doi.org/10.1016/j.simpa.2024.100622.

BIOGRAPHY OF AUTHORS

Tia Rahmawati, received her Bachelor's degree in Telecommunication Engineering from Telkom University, Indonesia, in 2023, after earning a Diploma in Telecommunication Technology from the same institution in 2021. She is currently pursuing her Master's degree at the School of Electrical Engineering, Telkom University, Indonesia. She has been working as a Security Analyst at Defenxor. Her research interests include cybersecurity, image processing and microcontroller. Email: tiatrw@student.telkomuniversity.ac.id, Orcid: 0009-0008-4138-4756.

Nyoman Bogi Aditya Karna, received the Ph.D. degree in electrical engineering and computer science from Bandung Institute of Technology, West Java, Indonesia, in 2018. He has been a full-time Lecturer with the School of Electrical Engineering, Telkom Higher School of Technology (now Telkom University), West Java, since 1999. His research interests include the intelligent IoT, cybersecurity, and the Internet of Drone Things. Email: aditya@telkomuniversity.ac.id, Orcid: 0000-0002-0092-2692.



Soo Young Shin, received his Ph.D. degrees in electrical engineering and computer science from Seoul National University in 2006. He was with WiMAX Design Lab, Samsung Electronics, Suwon, South Korea, from 2007 to 2010. He is currently an associate professor in the Department of IT Convergence Engineering at Kumoh National Institute of Technology, Korea. He was a postdoctoral researcher at the University of Washington in 2007 and a visiting scholar at the University of British Columbia, Canada, in 2017. His research interests include 5G/6G wireless communications and networks, signal processing, the Internet of Things, mixed reality, drone applications, and more. Email: wdragon@kumoh.ac.kr, Orcid: 0000-0002-2526-2395.

Made Adi Paramartha Putra, received the Ph.D. degree in IT convergence engineering from the Kumoh National Institute of Technology, Gumi, South Korea, in 2024. He is currently a full-time Lecturer of informatics engineering with Primakara University, Bali, Indonesia, and also the Director of the Postgraduate Studies, in 2024. His research interests include named data networks (NDN), the real-time Internet of Things, federated learning optimization, blockchain, and energy efficient architecture.of Drone Things. Email: adi@primakara.ac.id, Orcid: 0000-0002-6024-941X.