6500CSMM Project

School of Computer Science and Mathematics

Final Report Submitted by

Khaing Zin Thein

PN1140819

B.Sc. (Computer Networks)

"Implementation of A Resilient Campus Network with Suricata IDS and Integrated SIEM"

Supervised by

U Thet Zaw Aye

Submitted on

June, 2025

Table of Contents Pages

Abstract

This project outlines the full design and implementation of a resilient campus network topology with the capability of supporting at least 200 users within the various academic and administrative departments. This network topology utilizes VLAN segmentation and inter-VLAN routing in a way that maintains efficient use of bandwidth and logical separation of traffic, providing both security and performance benefits. At the center of this project is the implementation of Suricata IDS to provide proactive intrusion detection and the ELK Stack (Elasticsearch, Logstash and Kibana) to perform the centralized logging and real-time analysis of logs. These two utilities allow for not only continuous monitoring and detection of security incidents, but also the ability to respond quickly to incidents. As a result, the network has the tools needed to significantly reduce the risk of unauthorized access and downtime to services while also being flexible enough to support future growth and expansion within the university environment. The findings from this project highlight the importance of structured network segmentation and awareness-based monitoring in building a secure and highly available digital environment.

Acknowledgements

I would first like to thank Liverpool John Moores University for the opportunity to undertake this project, which has enabled me to enhance my knowledge of resilient networking design and security systems. I would also like to thank Auston University for their support of this project which has included access to the resources I needed, laboratory space and the tools to help me complete this work.

My sincere thanks to my supervisor, U Thet Zaw Aye, who supported me by providing guidance, motivation and feedback throughout each stage of the project. Many thanks also to Dr Nyein Aye Maung Maung for her helpful discussions that were informative and technical to further enhance my research, I sincerely appreciate it. I would also like to thank Dr Sandar Myo Myint, Head of Academic, for her continuous support, motivation, encouragement and for providing me with an environment conducive to my academic growth.

Thank you to Dr Su Thawda Win, Dr Kalyar Myo San and Dr Lae Win Thwin for their lectures on cybersecurity and for the valuable knowledge they contributed to assist me in understand cyber intrusion detection systems and the best practices in security. I also want to thank Sayar Khant Phyo, for the comprehensive instruction on networking, from the initial principles to far advanced configurations, he helped my confidence in building VLAN segmentation and securing network topologies.

Lastly, I want to say a heartfelt thank you to my friends, family, and colleagues for their consistent encouragement and emotional support which I had throughout all the struggles in completing this project.

Chapter 1

Introduction

1.1 Background and Context of Resilient Campus Network Design

With the digital landscape evolving so rapidly today, the reliance of educational institutions upon a dependable, responsive network architecture has significantly evolved. Today's campus networks form the foundation of a larger, more complex ecosystem of essential campus-specific academic and administrative applications, ranging from student information systems, to cloud-based learning management systems, to digital assessment applications, to collaborative research spaces. This complexity requires not only continuous network uptime, but successful, seamless connectivity for geographically disparate departments, classroom/labs, and business offices.

For many, the design of traditional campus network architecture does not support these evolving changing demands. Unfortunately, many legacy models are at risk of single points of failure, and lack scalable network architecture to support larger classes and data-intensive applications. Even worse, they often do not include any proactive security architectures needed in defending against ever-evolving sophisticated cyber threats. Diversity in campus users (faculty/staff, students, guests) further complicates the risk equation. In some cases, capable and technically savvy students may unintentionally and/or intentionally traffic potentially harmful penetration tools or vulnerabilities, resulting in insider threats that are uncommon with other standard commercial enterprises.

To tackle these complex adversities, resilient campus network design must adopt fault tolerance, scalability, and security integration as core design principles. This means applying logical network segmentation using VLANs to create broadcast domain boundaries and security boundaries. VLANs, by virtue of limiting broadcast domains, gives a level of assurance regarding the risk of a lateral threat propagation. Implementing Intrusion Detection in the perimeter of the network, such as Suricata, with its ability to aggregate all logs for a holistic view and network inspection capability in real-time will allow for greater visibility over malicious activity in the network and

response controls. In addition to Suricata, the log management and analytics capabilities of the ELK Stack lend itself as a Security Information and Event Management (SIEM) solution for the aggregation of logs, monitoring, visualization and analytics required to provide effective threat detection and forensics.

This idea will be realized in this project using the theoretical and pragmatic construct for designing and implementing a secure, scalable, and fault-resilient campus network. The planned design of the network model embraces VLAN segmentation, Suricata IDS/IPS, and use of the ELK Stack for active monitoring in order to achieve a suitable defence-in-depth posture that provides systems with high availability and high-performance responsiveness, with the unique requirements of educational institutions for proactive cybersecurity operations required.

1.2 Identification of the Research Problem

As academic institutions increasingly rely on digital systems to facilitate education, research, and administration, the security resilience of campus networks becomes a critical concern. Unfortunately, many university environments still operate with reactive, fragmented, and outdated security postures, leaving them vulnerable to a broad spectrum of internal and external cyber threats. The campus network assessed in this project exhibits several structural and operational deficiencies that compromise its ability to defend against attacks and maintain service continuity:

1.2.1 Lack of Network Redundancy

A key architectural weakness is the lack of redundancy features in core network components. The current topology contains single points of failure with core routers and distribution switches that, in the event of failure (hardware or cyberattack), disrupt many users and can cause drastic downtime. In any cyber event, attackers take advantage of these weak points in a network to cause maximum disruption, and we have experienced first-hand the impact of failure during high-value events such as online examinations, deadlines for coursework, and administrative operations. Row and core switches without redundancy do not only threaten uptime as a single point of failure, but also incident response and network resilience during active threats.

1.2.2 Absence of Real-Time Threat Detection and Centralized Monitoring

The campus network currently also does not support any consolidated Intrusion Detection or Prevention System (IDS/IPS) to report unusual traffic patterns, nor does it provide any capability for responding to an attack once it is underway. This means that actions such as port scanning, brute-force logons, lateral movement, and malware calling home are not detected until damage is done. There is also no central log aggregation tool, or even a Security Information and Event Management (SIEM) solution enabling usable visibility of the security posture of the network. Consequently, forensic investigation, compliance auditing, and threat correlation are impossible or delayed - which increase the mean time to detect (MTTD) and mean time to respond (MTTR) drastically.

1.2.3 Poor Network Segmentation and Access Control Enforcement

The network architecture does not enforce strict segmentation and isolation between the sensitive VLANs (e.g. administration, faculty, students, and guest devices). This flat or loosely segmented network architecture facilitates unregulated east-west traffic and increases the chances for attackers or malicious insiders to pivot across the network once initial access has been achieved. In addition, services and servers that are critical to the organization are exposed to users that do not require access to those services when the appropriate ACLs or user-based access policies are not in place, which violates the principle of least privilege and creates additional risk of insider threats.

1.2.4 Unsecured Endpoints and IoT Proliferation

The growing adoption of bring-your-own-device (BYOD) policies and IoT devices (e.g., smart boards, surveillance cameras, lab sensors) are adding numerous unmanaged and insecure endpoints onto the network. These devices are often poorly patched and insecure by default which make them attractive entry points for attackers. Once compromised, these devices can directly launch attacks internally, be used as compromised components of a botnet, and/or be leveraged to exfiltrate data. These rogue endpoints are undetectable and unmonitored without network access control (NAC) and/or anomaly detection.

All these vulnerabilities endanger the confidentiality, integrity, and availability (CIA) of academic data, staff records, student data, and institutional resources. Hence, this project proposes the need for a secure, monitored, and resilient campus network infrastructure where the University can detect, mitigate, and respond to modern cyber threats in real-time.

Year	Date	Attack	Victim University	Attack Behavior / Details
2020	May 28, 2020	NetWalker Ransomware	Michigan State University	Ransomware encrypted critical data, disrupting university services; demanded ransom for decryption keys.
2020	June 1, 2020	NetWalker Ransomware	UCSF School of Medicine	Targeted medical school data, encrypted files; ransom demand included threat of data leak to dark web if unpaid.
2020	Oct 1, 2020	Ransomware (unspecified)	California State University San Marcos	Attack caused system lockdown; university paid ransom to regain access.
2023	Feb 7, 2023	BlackCat Ransomware	Munster Technological University	Sophisticated ransomware attack; exfiltrated sensitive data before encryption; attackers leaked sample data publicly.
2023	June 2023–Mar 2024	Phishing & Data Breach	Western Sydney University	Prolonged breach involving phishing emails leading to credential theft and unauthorized access to sensitive data.
2024	May 21, 2024	Microsoft 365 Account Breach	Western Sydney University	Breach via compromised Microsoft 365 accounts affecting ~7,500 users;

				attackers accessed emails and personal info.
2025	Jan/Feb 2025	Single Sign-On Data Theft	Western Sydney University	Attackers exploited SSO vulnerabilities, stealing credentials and sensitive student/staff data.
2025	Jan 5, 2025	Rhysida Ransomware & Data Leak	University of the West of Scotland	Attackers encrypted systems and leaked 363 GB of data online; ransom demand accompanied leak threats.
2025	Feb 16, 2025	“fsociety” Ransomware Claim	Australian National University	Ongoing investigation; attackers claimed ransomware infection and data exfiltration; university alerted community.

Documented Cyberattacks Targeting Campus and University Networks from 2020 to 2025

1.3 Project Aim: Designing a Secure, Resilient, and Scalable Campus Network

This major undertaking is fundamentally a construction and implementation project for a robust campus networking infrastructure that addresses existing gaps in function and reliability, security, and availability. One primary outcome of the design project is high availability of networks with minimal downtime during and following equipment failure and/or cyber incursion.

A design tenet is VLAN segmentation, where users and departments have a logical separation into individual broadcast domains. VLAN segmentation affords improvements to performance from reduced unwanted traffic and enhances security by limiting exposure between network zones. VLANs support the justification of traffic isolation and access control in administrative areas such as Administration, Human Resources, Management, Students, Teachers and guests.

Additionally, the project utilizes the Suricata IDS, which is an advanced open-source detection engine that performs deep packet inspection (DPI) as well as real-time alerting. Also, as the project intends to use a full-fledged Security Information and Event Management (SIEM) platform, the Institute is deploying the ELK Stack (Elasticsearch, Logstash, Kibana) for centralized monitoring and analysis. With the ELK Stack, the Administrator/Users were able to centralized security logs across the network and correlate events and visualize them in dashboards for situational awareness. Designed collectively, the campus network is transformed into a proactive, and scalable and threat-aware network, where its infrastructure can respond to increases in user growth and changing threats.

1.4 Project Objectives: Enhancing Security, Visibility, Redundancy, and Scalability

This project is designed to transform a traditional campus network into a resilient, security-based, and scalable architecture that withstands modern cyber threats while transitioning to accommodate the adaptive requirements of most academic environments. The goal of this project is to reduce the risks of systemic network failures through the redundancy of the architectural design on both the routing and switching layer of the campus network. By implementing a central distribution switch (Central Hub) to route, monitor, and load share the network, the chance of failure in the event of upstream links or devices failing is greatly reduced. The design is set up for redundancy when it comes to single points of failure for any of the core services needed for the campus network to support the essential requirements of the academic institution.

In support of reinforcing internal segmentation and access control, the network internally separates each user community through the logical construction of user groups using VLANs - i.e., administration, human resources, students, guest users, teachers, and management in dedicated Layer 3 domains. In addition to internal segmentation, VLANs also contribute to performance improvements through multiple attached devices with Layer 2 through minimizing broadcast domains and limiting movements through lateral propagation, especially in the case of breach, as it aligns itself to a zero-trust model and the principle of least privilege .

A key aspect of the implementation is the deployment of Suricata, a strong open-source intrusion detection system (IDS), in the heart of the network. Suricata receives data from the central hub, monitoring inter-VLAN, internal-to-external, and east-west traffic in real time. Suricata monitors the highest value traffic flows, examining for signs of malicious activity (scans, exploits, policy violations), and acts as the baseline protection within a multi-layered security architecture through the mechanism of SPAN (port mirroring).

To produce actionable intelligence from raw detections, this project utilized the ELK Stack (Elasticsearch, Logstash, and Kibana) as the centralized SIEM. The ELK Stack is hosted on a dedicated virtual machine in a logically isolated cloud segment (Cloud1/vmnet2) and ingests logs as Suricata's output in Filebeat to allow normalization, correlation and visualization in near real time. Together, ELK and Suricata increased situational awareness, aided forensic investigations, and sped up incident response to internal and external incidents by creating a central dashboard for monitoring.

1.6 Project Methodology: A Phased, Layered Implementation Approach

This project utilized a phased and layered approach that incorporated secure network design with modern threat detection and monitoring procedures using a virtualized simulation environment. All the approach reflects a defense in depth strategy with segmentation, redundancy, detection, and centralized visibility, which is contextualized to a university network. The process began with an analysis of the existing architecture to identify primary flaws with the former network, namely; failover capabilities, VLAN segmentation and visibility of threats as they occurred in real-time. All observations in the analysis were directly related to the design blueprint and deployment.

The start of the project consisted of implementing the design and virtualization of the network infrastructure utilizing EVE-NG. Here a logical view of a modern campus topology was created. Appropriate VLANs were configured to represent different organizational units, Admin, HR, Management, Teachers, Students, and Guest VLANs were mapped against different IP subnets, towards creating segmentation and access control. All VLANs were given Layer 2 switches that connected to Layer 3 switches

using trunk links. A central hub was added in between the core router and the switching layer to provide an aggregated traffic flow point that could be monitored centrally. These design changes allowed for modularity, clearer traffic flows for analysis and an ideal location for monitoring and threat detection. The use of SPAN configuration with some of the access switches need improvement and did not function as intended. Under the monitored traffic flowing from the access switches was successfully mirrored to the central hub, and this should allow for the aggregation of traffic that would then be redirected to Suricata IDS for analysis and detection.

In the following step, Suricata was deployed as a standalone Ubuntu virtual machine and connected directly to the central hub switch. By placing the IDS in this position, the IDS was able to passively monitor mirrored traffic across all VLANs no matter the originating switch. Suricata was configured using custom detection rules, and community signatures that allowed the network-based detection of a variety of attacks like port scanning, ARP spoofing, malformed packets, and brute force authentication attempts, with outputs from Suricata formatted into structured JSON logs that are compatible with SIEM components and kept semantic integrity downstream.

Following the successful integration of the previously implemented IDS, my focus for further research lay in centralized monitoring with ELK. The ELK system was hosted on a different Kali Linux VM connected to a logically isolated cloud bridge (Cloud1/VMnet2). This ELK system was designed to receive, index, and visualize logs from Suricata. Filebeat was configured on the Suricata node to send logs securely and efficiently to Elasticsearch. As log source option, Logstash was still an optional part of the pipeline; although the option to use Logstash existed, it provided increased flexibility on areas of the ELK workflow that we would use parsing, if we needed to. Kibana would be the interface for threat visualization allowing real-time event analysis using type, severity, source/destination IP and triggered rules as display parameters. The centralized system exhibited risk-awareness improvements and allowed an obvious analytics option, when the need for deeper investigation arose.

The last phase was validation and iterative improvement. We tested connectivity on the network with inter-VLAN ping tests and verified default gateways to trace routing. We enabled promiscuous mode in Suricata to capture traffic properly.

Security testing was done with controlled Nmap scans, protocol fuzzing, and synthetic attack payloads. We confirmed that the alerts from Suricata detected, were cross-validated with the raw events received from the event logs, and the visual representations from the Kibana dashboards. Misconfigurations were discovered, including incomplete sets of Suricata rules and issues related to our Filebeat repository. Our tests located and corrected the misconfigurations. This iterative debugging and validation process allowed for further fortification of the detection pipeline, improved operational performance, and best of all, the environment closely represented operational network conditions, i.e. "the real world."

Overall, this methodology aligns with real-world workflows for network hardening and monitoring processes used in enterprise-grade deployments, assuring systematic placement of software; staged validation; and a complete stack visibility for the network security life cycle—design, instrument, detection, and analyzed centrally.

Chapter 2

Literature Review

2.1 Network Redundancy and High Availability

Network redundancy and high availability (HA) are two central concepts in designing fault-tolerant networks. They mitigate the effects of hardware and/or software failure by providing features that allow service to continue and minimize recovery time. Redundancy involves redundancy of component and the backup system (routers, switches, transmission paths, power systems) that ensures, that if a hardware failure occurs a single point of failure does not disrupt the communication of the network . HA is redundancy plus emphasis on keeping network services operable and providing minimal downtime during failure events. HA typically accomplishes this functionality through failover, heartbeat monitoring, load balancing, clustering, and other forms of redundancy (meeting, restores vast amounts of service quickly) in service delivery. Most of the redundancy models used to provide various types HA are classified as either active-active (both nodes configured to support traffic simultaneously); or active-passive -where the active node services traffic and when it needs fails the backup system then becomes active. These types of redundancy configurations require architectural support or reliability architecture to accomplish to

keep MTBF and MTTR reliable, when service delivery produces system disasters with continuous service in mind.

Looking at a layered architecture, redundancy at the network level applies at different layers of the OSI model.. At the Data Link Layer (Layer 2), we usually could implement redundancy with the help of the Spanning Tree Protocol (STP) and the Rapid Spanning Tree Protocol (RSTP) by preventing broadcast storms/leaving loops and creating loop-free Ethernet topologies across redundant spans of data connections. The STP family can dynamically monitor the topology, link or switch status, and manage a fast reconvergence to maintain communication without breaking loops or limiting redundancy; creating and maintaining reliability. In the Network Layer (Layer 3) we could apply redundancy with the help of dynamic routing protocols such as Open Shortest Path First (OSPF) and Enhanced Interior Gateway Routing Protocol (EIGRP). The adaptive routing nature of these protocols is possible because of the advertising and monitoring of Link-State Advertisements (LSAs) for OSPF, and via the Diffusing Update Algorithm (DUAL) in EIGRP. When a link fails these protocols invoke a rapid adjustment to routing paths with minimal delay. In the majority of cases, their convergence properties permit quick recovery and rerouting of traffic to establish connectivity and availability required in highly available routing environments.

In mission-critical environments—such as academic institutions, hospitals, and data centers—where uptime is essential, these principles become not only desirable but necessary. High availability and redundancy are increasingly seen as prerequisites for supporting modern workloads, cloud services, and security platforms, which depend heavily on uninterrupted connectivity and real-time responsiveness.Thus, the literature strongly supports the integration of multi-layered redundancy and high availability strategies as essential pillars of robust network architecture. The practical implementation of these strategies enables networks to recover swiftly from failures and to deliver reliable service, even under adverse conditions.

2.2 VLAN Security Vulnerabilities and Mitigation

Although VLAN segmentation greatly improves modularity, performance, and inherent isolation, it can also create security vulnerabilities when not properly configured and monitored. These vulnerabilities can arise from the technical limitations of VLAN protocols as well as from human error when configuring VLANs, which can

lead to unintended lateral movement, unauthorized access, and further expandability of an attack.

One of the main vulnerabilities that affect VLAN security is VLAN Hopping, where an attacker is somehow able to access traffic that does not belong to the VLAN they were legitimately placed in. This process, for example, is commonly achieved by two distinct methods: Switch Spoofing and Double Tagging. In switch spoofing, the attacker sets up their device to act like a trunking switch by using 802.1Q negotiation to access all VLAN traffic allowed on the trunk. Double tagging occurs when the attacker inserts two VLAN tags in a packet, one for the native VLAN and the other for the desired VLAN, causing the packet to be forwarded incorrectly by certain switches, allowing the packet to hop from the source VLAN to an unauthorized VLAN.

Another common problem is due to poorly defined native VLANs. IEEE 802.1Q allows a default VLAN (often VLAN 1) on trunk links to carry untagged traffic. When an attacker originates malicious traffic on this native VLAN that is improperly segregated or is unused, it can be a strong mechanism for them to launch cross-VLAN attacks. Likewise, with inconsistent VLAN assignments across switch ports, and an improper application of VLAN pruning, and by not disabling unused ports, they can broaden the attack surface that permits unauthorized VLAN access or lays down an entry for an insider.

Theoretically, these concepts reflect the idea of implicit trust in internal segments which poorly align with modern Zero Trust Architecture (ZTA) which assumes that no network segment is inherently secure. VLANs provide some level of segmentation, but not true isolation (or encryption), and thus without appropriate controls in place, the security boundary gets fuzzy.

To mitigate these threats, network administrators must follow best-practice VLAN hardening techniques:

- DTP (Dynamic Trunking Protocol) on all non-trunk ports to prevent switch spoofing.
- Set unused switch ports to an unused VLAN and administratively shut them down.
- Avoid using VLAN 1 or the default/native VLAN for any production traffic.

- Explicitly configure allowed VLANs on trunk ports (VLAN pruning) rather than relying on automatic negotiation.
- Deploy Private VLANs (PVLANs) to isolate devices within the same VLAN where needed.
- Use Access Control Lists (ACLs) and firewalls to control inter-VLAN traffic, ensuring communication policies are enforced at Layer 3.
- Monitor VLAN configurations with network scanning tools, SIEM systems, and periodic audits to detect misconfigurations and anomalous traffic patterns.

Current thinking about VLAN segmentation indicates that a defence-in-depth strategy or approach should be used in conjunction with VLANs, including some combination of endpoint protection, network access controls (NAC), and IDS/IPS tools such as Suricata that may detect VLAN hopping signatures to alert administrators.

In conclusion, all though VLANs are great segmentation tools on their own, they were not designed as a security solution. Attack paths remain present when VLANs are misconfigured, creating further exposure for an organization. As a result, through the use of both technical controls, continuous policy enforcement, secure practices will ensure VLANs preserve logical isolation and reduce risk.

2.3 Threat Intelligence and Rule Management in IDS

An Intrusion Detection System (IDS) like Suricata can only be effective if its ruleset is accurate, relevant and timely. Rules drive a detection system's ability to identify and generate attribution, and thus rule management based on contextualized threat intelligence will play a major role in how an IDS can identify and respond to both known and emergent threats, in a continually evolving network landscape.

Suricata's detection, signature-based engine has the ability to compare packets of network traffic against known signatures of malicious behavior. Network traffic signatures can be sourced from curated repositories, such as the Emerging Threats (ET) ruleset sponsored by Proofpoint, and the Suricata-Community Rules which have free and paid variants. The curated rules contain large collections of a variety of indicators of compromise (IOCs) for example IP addresses, domain names, byte sequences, protocol anomalies, or application-specific signature of an exploit.

This model is based on pattern matching and regular expression evaluation, both of which are incredibly accurate with known attacks, but consistently need updating to remain effective. This is where threat intelligence comes into play. Threat intelligence is information collected and analyzed from global threat feeds, honeypots, malware databases, and behavioral analytics platforms in order to identify new adversary tactics, techniques and procedures (TTPs). This intelligence can then be converted into new IDS rules that encompass new threats such as zero-day exploits, advanced persistent threats (APTs), and command-and-control (C2) communications.

In order to effectively manage the update of these rules, network security administrators usually pursue an automated update workflow such as PulledPork or Suricata-Update to pull the latest rules down from trusted feed and place it reliably into the IDS workflow. In practice, these tools provide a blend of classification of rules along with ability to prioritize and tune the rules, such that an organization can choose to suppress false positive alerts based on environment, or focus alerts based on threats more aligned with their local environment, or finely tune alerts as required for performance. For example, an educational institution network may focus on detection rules that pertain to malware spreading via peer to peer traffic or unauthorized VPN tunneling while suppressing rules that trigger on preferably enterprise only services.

In addition, rule management includes tuning and contextual enrichment, aligning rule actions (alert, drop, pass) with the organization’s security policies. By leveraging metadata tags, flowbits, and thresholding, security teams can refine detection logic, reduce alert fatigue, and improve response prioritization. Integration with a SIEM (e.g., ELK Stack) allows rule-triggered events to be correlated with host and application logs, further enhancing contextual awareness and enabling faster threat validation.

Ultimately, effective rule management in Suricata is not a one-time configuration task but an ongoing lifecycle process that mirrors the pace of global cyber threat evolution. By aligning IDS rules with live threat intelligence feeds and organizational risk profiles, security teams transform Suricata from a static monitoring tool into a dynamic threat prevention system capable of adapting to both known and unknown adversarial behavior.

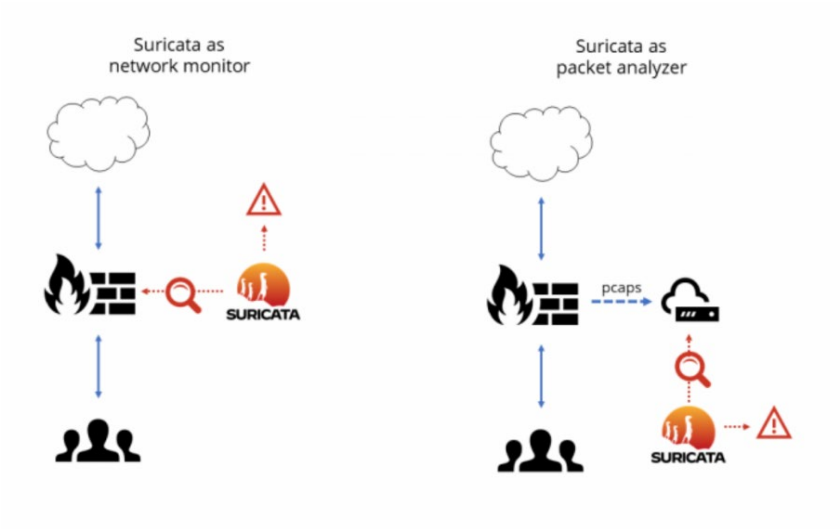


Figure 1 : Suricata Detection Process

2.4 Signature-Based vs. Anomaly-Based Detection Models

Anomaly-based and signature-based detection models are two primary models for use in an intrusion detection and prevention system (IDPS). Anomaly-based and signature-based models are based on different theories and implementation techniques. Signature-based detection uses known attack patterns or "signatures" to compare observed data against a collection of signatures in a database. Specifically, signature-based detection is deterministic and works well when identifying documented threats, including malware with a known payload (the exploit) or a specific sequence of commands and payloads provided in an exploit kit. Suricata uses the signature-based model, with a multi-featured rule-based language where signatures are updated continuously through sources like Emerging Threats. The leading advantage of signature-based detection is its precision and low false positive rate, assuming that signatures are accurate. The principal disadvantage of signature detection is its reactive nature—zero-day attitude does not work against zero-day exploits and variants that lack an adequate signature.

Anomaly based detection uses statistical, heuristic or machine learning algorithms to create a baseline of normal system behaviour. Any deviances from it (excessive volume of traffic, unexpected access times, abnormal ports or protocols, etc.) are flagged as a potential intrusion. This model is a reflection of behavioural analysis theory, which focuses on identifying threats by behavior, rather than

indicators. Anomaly detection can be effective to identify novel or targeted attacks, such as advanced persistent threats (APTs) or insider threats. However, this model can suffer from high false positive events, particularly in dynamic environments where normal behavior is frequently changing. The Elastic ML plugin added capabilities to the ELK Stack for anomaly based detection using unsupervised techniques against traffic and logging data, adding another layer of detection to the signature defender.

In practice, modern security operations can gain the most benefit by using a hybrid model that takes advantages of both the specificity of signature based, and the flexibility and adaptability of anomaly based detection. For example, Suricata can be used in parallel with all the ML capabilities of the ELK Stack to correlate known threats and identify suspicious behavior that deviates from the observed baseline, creating a layered approach against defense-in-depth and zero-trust frameworks.

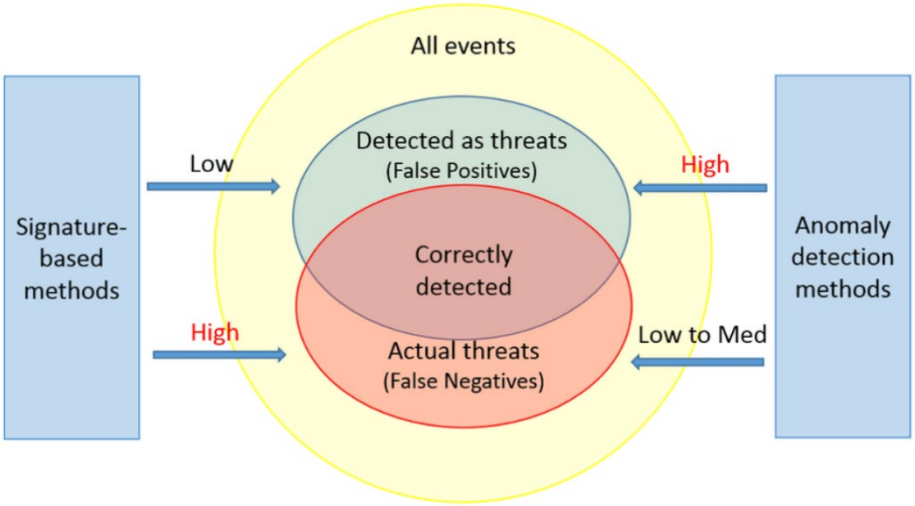


Figure 2 : Signature-based vs. Anomaly-based methods of intrusion detection

2.5 Real-Time Threat Detection and Monitoring in Modern Network Environments

Real-time threat detection and monitoring are foundational components of modern cybersecurity strategy that rely on the theoretical frameworks of continuous monitoring, situational awareness, and defense-in-depth. This proactive approach focuses on seeing security incidents occur through the immediate chance to take reduction actions to maintain critical systems' Confidentiality, Integrity, and Availability

(CIA). As opposed to reacting to a detected anomaly (and the annoyances that may have arisen from it), real-time detection allows organizations to rapidly respond to anomalies when they happen in order to mitigate the potential risk damage from threats such as malware, lateral movement, and zero-day attacks.

Theoretically, real-time detection is predicated on the Observe-Orient-Decide-Act (OODA) loop model that emphasizes rapid situational responses based on iterative observation and decision-making about a threat . Operationally, the OODA loop is possible through the use of Intrusion Detection and Prevention Systems (IDS/IPS) such as Suricata, and Security Information and Event Management (SIEM) tools such as the ELK Stack. Suricata uses a combination of Deep Packet Inspection (DPI) and stateful inspection to monitor data flows in real time, and identifies suspicious events through signature-based or behavioural heuristics such as brute-force attempts or command-and-control traffic .

Once events are identified, they flow into the ELK Stack for processing and analytics. Logstash conduct data normalization, parsing different structured log formats to formulate a common meaningful schema that embodies the theoretical framework supporting consistency in data structure for large scale event correlation and audit processes. Elasticsearch employs distributed and horizontally scalable big data theory to store, index, and query vast amounts of security logs with minimal latency which enables low-latency analytics as is commonplace in high-speed networks where immediate insights present a byproduct of speed with time being a critical element for incident response.

Kibana provides the visualization and interaction layer the where it transforms data into interactive dashboards and visual alerts. The basis of data abstraction and human-centered visualization underpins Kibana, allowing security analysts to visualize trends, outliers, and events chains, which improves response and decision accuracy. Dashboards often map out MITRE ATT&CK frameworks, enabling a contextual connection between observed behaviors and known adversarial tactics.

The integration of Suricata and the ELK Stack reflects the layered security model, where multiple detection and analysis points work together to provide redundancy and depth. This multi-tiered approach aligns with defense-in-depth strategy, ensuring that even if one detection layer fails, others remain active to mitigate

threats. Furthermore, this synergy supports the broader goal of threat intelligence-driven defense, allowing for adaptive rule updates, historical pattern recognition, and proactive blocking.

In conclusion, real-time threat detection and monitoring represent a theoretically rigorous and practically effective cybersecurity capability. The combination of Suricata's real-time traffic inspection and the ELK Stack's scalable log analysis infrastructure provides a comprehensive defense framework. Together, they support rapid detection, contextual analysis, and informed decision-making, all of which are crucial for maintaining resilience in today's evolving cyber threat landscape.

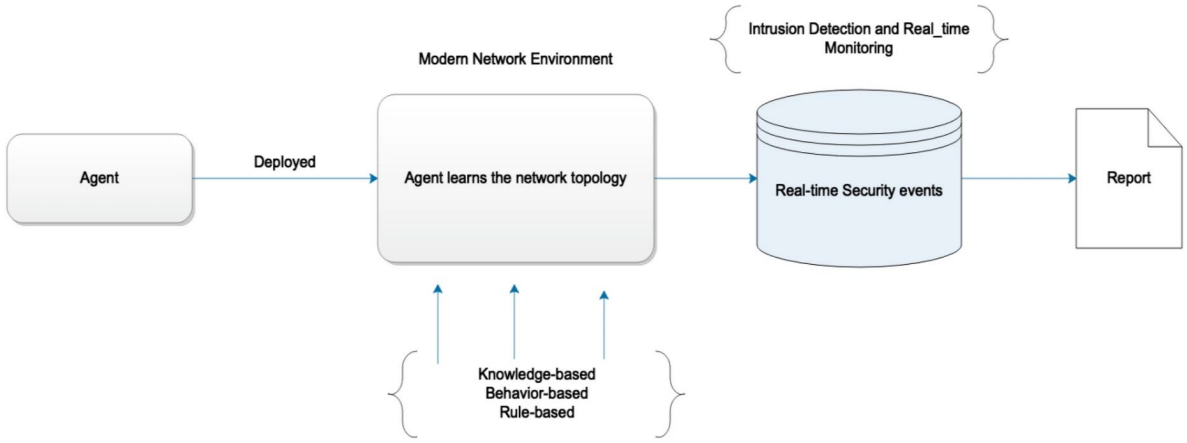


Figure 3 : Agent in Modern learning Environment

2.6 Network Visibility and Traffic Analysis

Network visibility and traffic analysis are fundamental for good cybersecurity monitoring, situational awareness and incident response. In theory, visibility in the context of network traffic means observation of all data, network interactions of devices and their behavior in an infrastructure. This connects strongly with observability and network situational awareness in that we want to be able to demonstrate that we know where every packet traversing through the network is and we can log, see, and analyze it. Visibility is particularly important in an environment which uses a combination of Intrusion Detection/Prevention Systems (IDS/IPS) such as Suricata and Security Information and Event Management (SIEM) like ELK Stack, which are dependent upon thorough, complete, and accurate traffic data to sound security events and/or mitigations.

Technologies at the heart of acquiring network visibility include SPAN (Switched Port Analyzer) or port mirroring, which provide passive copy traffic from one or more switch ports to the monitoring port. The copy is then fed to a monitoring tool-type Suricata for deep packet inspection. SPAN fits nicely with the observer effect in network monitoring for which the observation process does not affect production traffic. SPAN is also specific to virtual environments. Virtual switches (vSwitches) have the ability to be programmed to mirror traffic to a monitoring interface, providing visibility as long as it is all one segment or was VLAN segregated at the time of capture. Limitations of SPAN include packet loss in high traffic conditions and potential inefficiency in distributed networks.

NetFlow is a Cisco protocol that is similar to SPAN, but NetFlow abstracts the IP traffic flow into Flow Records based on source/destination IP, source/destination ports, protocols, and byte count. Conceptually, NetFlow supports the flow-based traffic analysis model which is a Mode of analysis that reduces monitoring overhead by abstracting the raw packets into flow records that can then be analyzed for trends, anomalies, and potential Indicators of Compromise (IOCs) without the need of storing each individual packet. NetFlow and its derivatives (i.e., sFlow, IPFIX,) are an especially valuable utility for capacity planning, DDoS attack detection, and application usage. It is not as granular as full packet capture for incident investigations, but the tradeoff is the scalability simulation of a packet capture or full packet analysis that NetFlow can deliver. Its architectural support is typically provided for SIEM solutions as a mechanism to monitor high-level events and generate alerts based on anomalies.

Full packet capture is another fundamental way to derive visibility from network traffic. Full packet capture records all packets, including payloads, and keeps them for historical forensic analysis. This is consistent with the theoretical construct in retrospective security, that is network data may be useful to reanalyze as new, previously unknow threats are discovered. Tools like Suricata in IDS mode rely on packet captures (or interfaces) using pcap libraries and network monitoring will average the last 2-3%, allowing it to check the traffic in real time for rules violations. Packet captures are key to understanding substantial threats like Advanced Persistent Threats (APTs), lateral movement, or exfiltration attempts which often do not invoke a signature prior to detecting, and then identifying them during post-event analysis.

These types of visibility methods are elemental to getting traffic into SIEMs like the ELK Stack. Its Logstash collection layer can intake either packet or flow log metadata, normalize or enrich the event, and send structured data to a data store like Elasticsearch. Kibana then can present the events in a visual manner, with dashboards available for analysts examining traffic, suspicious flows, and historical evolution. Ultimately, the integration of traffic visibility with real time data gives organizations the ability to support the observe-orient-decide-act (OODA) loop in incident response processes, and provide overall improvement to the challenges of cyber defence.

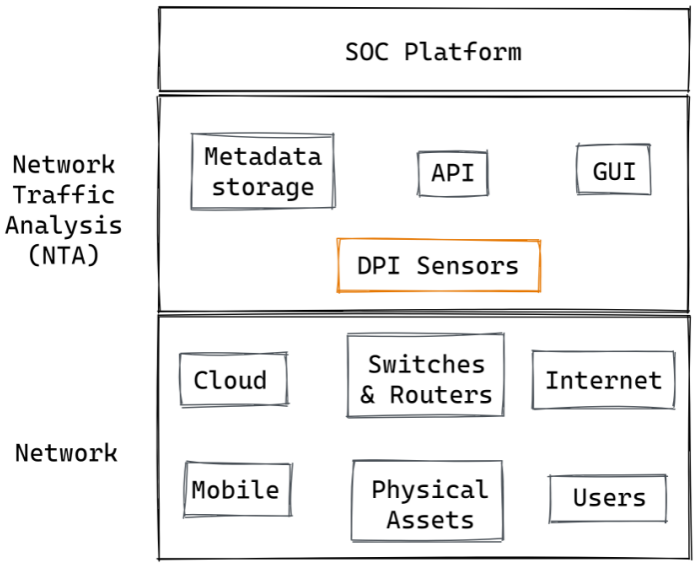


Figure 4 : An Overview of Network Traffic Analysis

2.7 Integration of SIEM with IDS for Proactive Defense

The association of Intrusion Detection Systems (IDS) such as Suricata with Security Information and Event Management (SIEM) platforms such as the ELK Stack (Elasticsearch, Logstash, Kibana) is a compelling measure when establishing a proactive, layered defense model within the architecture of your cybersecurity. Seeing the integration of IDS and SIEM platforms exemplifies the layered defense or defense-in-depth architecture and requirements offering respective capabilities to detect threats, analyze scope and contextualize relevant information and ultimately respond to threats at every stage of the attack lifecycle.

Conceptually, proactive defense not only involves real-time detection of threats, but also automated correlation of threats, contextualization of those threats, and

responding primitively to those threats. Suricata in IDS mode captures the packet and describes the flow in data format through deep packet inspection (DPI) and signature-based detection. Suricata alerts on suspicious activity (port scanning, brute force attempts, and malware communications) and those alerts are available in structured language such as EVE JSON and sent off to Logstash for ingest in the SIEM pipeline.

Within the ELK Stack, each component serves a critical role in enabling proactive security:

- Logstash acts as the event pipeline, parsing, enriching, and normalizing Suricata's output using filter plugins to prepare data for correlation.
- Elasticsearch indexes and stores the structured event data, allowing fast querying/performance and real-time search - both essential for timely detection of incidents and historical forensic investigations.
- Kibana presents these visual insights in interactive dashboards, highlighting patterns such as alert spikes, repeated attack vectors, or anomalies across subnets and VLANs. This enables analysts to identify patterns and threat-hunt using both a macro (strategic) view, as well as a micro (tactical) perspective.

From an operational perspective, the combination provides real-time threat correlation. Here, you take the data from disparate sources (e.g., firewall logs, endpoint telemetry, Suricata alerts) and pull it into the SIEM, where it all gets correlated and analyzed. Analysts can then take seemingly isolated events and link them together to form a coherent narrative of an ongoing attack (e.g., phishing email resulting in a connected outbound malicious connection and subsequent privilege escalation attempt).

Additionally, when integrated with alerting functions like ElastAlert or Elastic's built-in Watcher, such a system could also execute automated responses (e.g., blocklists, scripts, or send alerts to a SOC), drastically decreasing MTTD and MTTR.

In addition, this system allows compliance monitoring, Elastic ML for anomaly detection, and incident triaging workflows based on severity or asset criticality, amongst other capabilities. For example, Suricata can identify lateral movement attempts across VLANs, and the SIEM can aid in identifying the asset, the source of the connection, and any historical attempts that may have occurred as well.

In essence, the Suricata + ELK integration is more than a log aggregation pipeline—it is an intelligence-driven security framework that aligns with cyber kill chain modeling, OODA loop, and NIST incident response lifecycle. It moves the organization from reactive log reviews to automated, contextual threat intelligence and action.

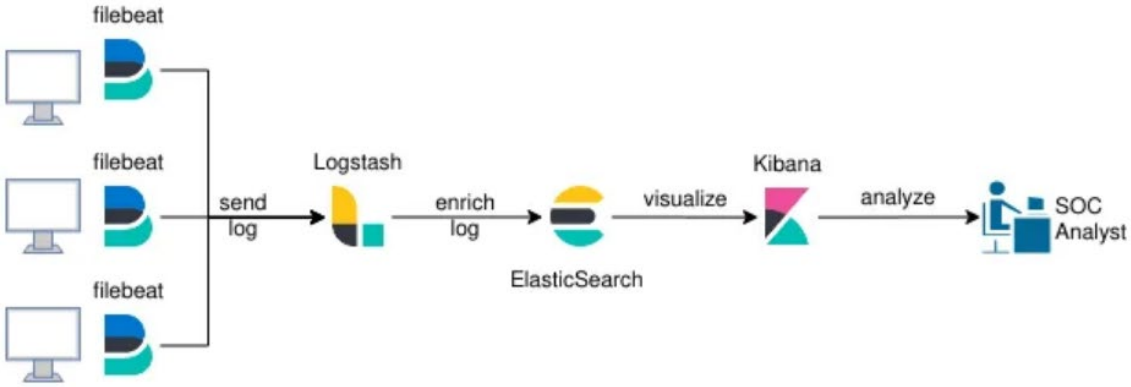


Figure 5 : File Iteration of SIEM Architecture

Chapter 3

Demonstration and Implementation

3.1 Overview

This chapter provides a description of how a virtual campus network with Suricata IDS and ELK-based SIEM systems was implemented in practice. The EVE-NG virtualization platform was used to design a practical network implementing segmented VLANs, inter-VLAN routing, and traffic monitoring in real-time.

The demonstration was structured in four key phases:

- Network Topology Design and VLAN Segmentation
- Suricata Deployment on Ubuntu VM
- Filebeat Log Forwarding to ELK Stack
- ELK Stack Installation and Configuration on Kali VM

3.2 Phase I: Virtual Network Design and VLAN Configuration

3.2.1 Network Topology and Virtual Infrastructure

In order to create and test a robust and security-informed campus network, the complete system was designed and modelled in the EVE-NG virtualization environment. This virtual testbed allowed for the construction of a scalable, modular, and highly dynamic representation of a modern university network. Central to the design was a centralized architecture, which included one main router (R1), a core distribution switch, and two distribution switches (Switch1 and Switch2)—which were serving as aggregation points for VLAN traffic from the access switches (S1-S6) assigned in each department.

Each access switch hosted one or more end-host devices (virtual PCs) assigning themselves to specific VLANs based on realistic organizational roles, such as AUS_ADMIN (VLAN 10), TEACHER (VLAN 20), STUDENT (VLAN 30), GUEST (VLAN 40), AUS_HR (VLAN 50), and AUS_MGMT (VLAN 60). Each VLAN is assigned its own subnet and default gateway (for example, 172.18.10.0/24 for VLAN 10 and 172.18.20.0/24 for VLAN 20), allowing for logical segmentation and layer 3 isolation of broadcast domains.

Traffic from each VLAN flowed through its respective Layer 3 switches with 802.1Q trunking and was sent to the central hub. The central hub acted as a logical traffic aggregation point, directing outbound communications and inter-VLAN communications to the main router (R1), which connected to an external network cloud (Cloud0) to simulate Internet connectivity.

An important improvement in this new topology was also the direct connection of a Suricata-based intrusion detection system (IDS) to the central hub. This placement meant Suricata could passively analyse mirrored VLAN traffic from all over the network. Instead of configuring per-switch SPANs as done in traditional topologies, the central hub allowed for greatly simplified exposure to traffic across multiple segments, which significantly improved scalability and precision of detection.

At the same time, a dedicated ELK monitoring stack was hosted on a separate Kali Linux VM segmented by Cloud1 (VMnet2). Just as before, this ensured that the monitoring systems were both logically and operationally isolated from our production paths. The use of EVE-NG for this virtual simulation allowed for efficient emulation of switching, routing, and security behaviour; which allowed multiple sets of testing, debugging, and validating of complex enterprise network configurations.

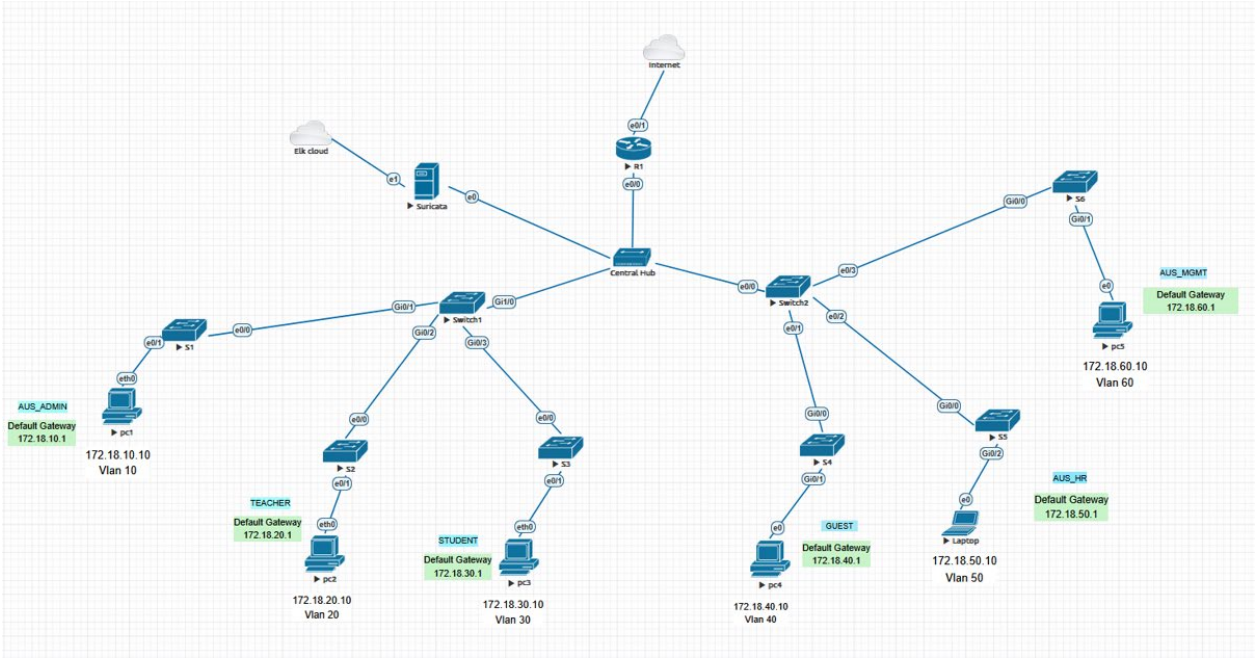


Figure 6 : Network Topology Design of Campus Network

3.2.2 VLAN Segmentation and Device Configuration

The VLAN segmentation configuration was thoroughly put into place using the EVE-NG environment. On Router R1, a subinterface was configured with encapsulation dot1Q for each VLAN, for example interface Ethernet0/2.60 was configured with 172.18.60.1 for TEACHER VLAN which allowed for isolated broadcast domains while enabling restrictive control of traffic. Identical configurations were made for the other VLANs, from VLAN 10 through to VLAN 50.

The central switches (Switch 1 and Switch 2) were configured to enable VLAN trunking on their respective Gigabit interfaces, so tagged VLAN traffic could pass between the switches and the router. Each of the access switches (S1-S6) were assigned to respective VLANs, in access mode on the correct ports, and with the other port configured to trunk toward the central switch. This enabled consistent and isolated

routing, while allowing for inter-VLAN routed traffic when necessary, via R1. In addition to the logical separation the VLAN architecture provided, it also provided a base for security monitoring via Suricata and the SIEM.

i. Router (R1) Subinterface Configuration

- On Router R1, subinterfaces were configured to handle traffic for each VLAN using IEEE 802.1Q encapsulation.

```
Router(config)#interface e0/0.10
Router(config-subif)#encapsulation dot1Q 10
Router(config-subif)#ip address 172.18.10.1 255.255.255.0
Router(config-subif)#no shut
Router(config-subif)#interface e0/0
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#interface e0/0.20
Router(config-subif)#encapsulation dot1Q 20
Router(config-subif)#ip address 172.18.20.1 255.255.255.0
Router(config-subif)#no shut
Router(config-subif)#interface e0/0
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#interface e0/0.30
Router(config-subif)#encapsulation dot1Q 30
Router(config-subif)#ip address 172.18.30.1 255.255.255.0
Router(config-subif)#no shut
Router(config-subif)#interface e0/0
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#do show vlan brief
                     ^
```

```
Router(config)#interface e0/2.40
Router(config-subif)#encapsulation dot1Q 40
Router(config-subif)#ip address 172.18.40.1  255.255.255.0
Router(config-subif)#no shut
Router(config-subif)#interface e0/2
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#inte
*Jun  7 05:01:54.586: %LINK-3-UPDOWN: Interface Ethernet0/2, changed state to up
*Jun  7 05:01:55.595: %LINEPROTO-5-UPDOWN: Line protocol on Interface Ethernet0/
2, changed state to up
Router(config)#interface e0/2.50
Router(config-subif)#encapsulation dot1Q 50
Router(config-subif)#ip address 172.18.50.1 255.255.255.0
Router(config-subif)#no shut
Router(config-subif)#interface e0/2
Router(config-if)#no shut
Router(config-if)#exit
Router(config)#interface e0/2.60
Router(config-subif)#encapsulation dot1Q 60
Router(config-subif)#ip address 172.18.60.1 255.255.255.0
Router(config-subif)#no shut
Router(config-subif)#interface e0/2
Router(config-if)#no shut
Router(config-if)#exit
```

Each subinterface (e.g., Ethernet0/0.30) was assigned to a specific VLAN using encapsulation dot1Q <VLAN_ID>. This allowed tagged VLAN traffic to be routed, providing a default gateway for devices in each VLAN. The IP address assigned (e.g.,

172.18.30.1) corresponds to that VLAN's default gateway. The no shutdown command activates the subinterface.

```
Router(config-if)#interface e0/1
Router(config-if)#ip address 192.168.91.100 255.255.255.0
Router(config-if)#ip default-gateway 192.168.91.2
Router(config)#no shut
```

```
Router(config)#do show ip interface brief
Interface                  IP-Address      OK? Method Status                Prot
ocol
Ethernet0/0                unassigned      YES NVRAM  up                    up

Ethernet0/0.10             172.18.10.1     YES NVRAM  up                    up

Ethernet0/0.20             172.18.20.1     YES NVRAM  up                    up

Ethernet0/0.30             172.18.30.1     YES NVRAM  up                    up

Ethernet0/1                192.168.91.100  YES NVRAM  up                    up

Ethernet0/2                unassigned      YES NVRAM  up                    up

Ethernet0/2.40             172.18.40.1     YES NVRAM  up                    up

Ethernet0/2.50             172.18.50.1     YES NVRAM  up                    up

Ethernet0/2.60             172.18.60.1     YES NVRAM  up                    up

Ethernet0/3                unassigned      YES NVRAM  up                    up
```

ii. Central Switch Trunk Configuration (Switch 1 and Switch 2)

- On the central switches, trunk links were configured to allow VLAN-tagged traffic between the switches and the router.

Example configuration for Central Switch 1:

```
Switch(config-if)#no shut
Switch(config-if)#int g1/0
Switch(config-if)#switchport trunk allowed vlan 10,20,30
Switch(config-if)# switchport trunk encapsulation dot1q
Switch(config-if)# switchport mode trunk
Switch(config-if)#
Switch(config-if)#int g0/1
Switch(config-if)# switchport trunk allowed vlan 10
Switch(config-if)# switchport trunk encapsulation dot1q
Switch(config-if)# switchport mode trunk
Switch(config-if)#int g0/2
Switch(config-if)#switchport trunk allowed vlan 20
Switch(config-if)# switchport trunk encapsulation dot1q
Switch(config-if)# switchport mode trunk
Switch(config-if)#
Switch(config-if)#int g0/3
Switch(config-if)#switchport trunk allowed vlan 30
Switch(config-if)# switchport trunk encapsulation dot1q
Switch(config-if)# switchport mode trunk
Switch(config-if)#
Switch(config-if)#
Switch(config-if)#do sh vlan
```

- VLAN 10 , VLAN 20 and VLAN 30 are created and named for clarity. The trunk interfaces are then configured with dot1q encapsulation and set to trunk mode.

The allowed vlans command ensures only specific VLANs are permitted on the trunk link.

```
Switch(config-if)#do sh vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active
10   VLAN0010                         active    Gi0/1
20   VLAN0020                         active    Gi0/2
30   VLAN0030                         active    Gi0/3
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup
Switch(config-if)#do sh int trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       on               802.1q         trunking      1
Gi1/0       on               802.1q         trunking      1

Port        Vlans allowed on trunk
Gi0/0       10,20,30
Gi1/0       10,20,30

Port        Vlans allowed and active in management domain
Gi0/0       10,20,30
Gi1/0       10,20,30

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/0       10
Gi1/0       10
```

iii. Access Switch Configuration (e.g., S6 for VLAN 60)

- Each access switch was configured with access mode on the port facing the client PC and trunk mode on the uplink port towards the central switch. Example configuration on S1 , S2 , S3 :

S1

```
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#interface e0/1
Switch(config-if)#switchport mode access
Switch(config-if)#switchport access vlan 10
% Access VLAN does not exist. Creating vlan 10
Switch(config-if)#switchport access vlan 10
Switch(config-if)#no shut
Switch(config-if)#interface e0/0
Switch(config-if)#switchport trunk encapsulation dot1q
Switch(config-if)#switchport mode trunk
Switch(config-if)#switchport trunk allowed vlan 10
Switch(config-if)#no shut
Switch(config-if)#do show vlan brief
```

```
S2(config)#interface e0/0
S2(config-if)#switchport trunk allowed vlan 20
S2(config-if)#switchport trunk encapsulation dot1q
S2(config-if)#switchport mode trunk
S2(config-if)#
*Jun 13 13:58:51.543: %CDP-4-DUPLEX_MISMATCH: duplex mismatch discovered on Ethernet0/0 (not full duplex),
S2(config-if)#no shut
S2(config-if)#interface e0/1
S2(config-if)#switchport trunk allowed vlan 20
S2(config-if)#switchport mode trunk
Command rejected: An interface whose trunk encapsulation is "Auto" can not be configured to "trunk" mode.
S2(config-if)#switchport mode trunk
Command rejected: An interface whose trunk encapsulation is "Auto" can not be configured to "trunk" mode.
S2(config-if)#switchport trunk encapsulation dot1q
S2(config-if)#switchport mode trunk
```

```
S3(config)#int e0/0
S3(config-if)#switchport trunk allowed vlan 30
S3(config-if)#switchport trunk encapsulation dot1q
S3(config-if)#switchport mode t
*Jun 13 14:01:54.411: %CDP-4-DUPLEX_MISMATCH: duplex mismatch discovered on Ethernet0/0 (not full duplex),
S3(config-if)#switchport mode trunk
S3(config-if)#no shut
S3(config-if)#int e0/1
S3(config-if)#switchport trunk allowed vlan 30
S3(config-if)#switchport trunk encapsulation dot1q
S3(config-if)#switchport mode trunk
```

All interfaces facing the end devices are set to access mode and assigned to specific VLANs. The uplink interface (e0/0) is configured as trunk to carry VLAN-tagged traffic back toward the centre switch.

iv. IP Assignment for VPCS in VLANs

- To enable each virtual PC (VPCS) to communicate within its VLAN and route traffic via its respective gateway, static IP addresses were manually assigned. Each host uses a /24 subnet and the first usable IP as its default gateway.

```
VPCS> ip 172.18.10.10 255.255.255.0 172.18.10.1
Checking for duplicate address...
PC1 : 172.18.10.10 255.255.255.0 gateway 172.18.10.1

VPCS> show ip

NAME        : VPCS[1]
IP/MASK     : 172.18.10.10/24
GATEWAY     : 172.18.10.1
DNS         :
MAC         : 00:50:79:66:68:0a
LPORT       : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU         : 1500
```

This command assigns PC1 the IP 172.18.10.10 with a subnet mask of /24. The default gateway is set to 172.18.10.1, which is typically configured on the router sub interface for VLAN 10.

```
VPCS> ip 172.18.20.10 255.255.255.0 172.18.20.1
Checking for duplicate address...
PC1 : 172.18.20.10 255.255.255.0 gateway 172.18.20.1

VPCS> show ip

NAME        : VPCS[1]
IP/MASK     : 172.18.20.10/24
GATEWAY     : 172.18.20.1
DNS         :
MAC         : 00:50:79:66:68:07
LPORT       : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU         : 1500
```

PC2 is given the IP 172.18.20.10, with a subnet mask for VLAN 20. The gateway 172.18.20.1 allows PC2 to send traffic beyond its VLAN if routing is enabled.

```
VPCS> ip 172.18.30.10 255.255.255.0 172.18.30.1
Checking for duplicate address...
PC1 : 172.18.30.10 255.255.255.0 gateway 172.18.30.1

VPCS> show ip

NAME        : VPCS[1]
IP/MASK     : 172.18.30.10/24
GATEWAY     : 172.18.30.1
DNS         :
MAC         : 00:50:79:66:68:0f
LPORT       : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU         : 1500
```

This assigns PC3 its VLAN 30 address and sets 172.18.30.1 as its gateway, enabling it to reach external networks via inter-VLAN routing.

3.3 Phase II: Suricata IDS Deployment on Ubuntu VM

Suricata was installed on a Ubuntu virtual machine (IP: 192.168.20.128) on the same VMnet2 network to provide advanced intrusion detection and monitoring. The installation proceeded by first running an apt-get update and apt-get install suricata, and editing the configuration files, including suricata. yaml. Suricata was configured to monitor the defined network interface (in this environment it was ens3) and we set the HOME_NET and EXTERNAL_NET variables to VLAN IP ranges.

Rules sets were downloaded and updated for Suricata, using relevant signatures to detect various attacks including DoS, DDoS, and port scans. Suricata service was enabled and started with systemctl enable suricata and systemctl start suricata, so it could monitor mirrored VLAN traffic indefinitely. Logs were configured to be saved in /var/log/suricata/, where Suricata captured events in JSON format to integrate with the ELK Stack.

3.3.1 Ubuntu VM Configuration

- Assigns a dynamic IP address to the ens4 interface so it can communicate within the 192.168.20.0/24 subnet.

```
khine@khine-virtual-machine:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou
p default qlen 1000
    link/ether 50:00:00:14:00:00 brd ff:ff:ff:ff:ff:ff
    altname enp0s3
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP grou
p default qlen 1000
    link/ether 50:00:00:14:00:01 brd ff:ff:ff:ff:ff:ff
    altname enp0s4
    inet 192.168.20.128/24 brd 192.168.20.255 scope global dynamic noprefixroute
 ens4
       valid_lft 1573sec preferred_lft 1573sec
    inet6 fe80::b416:73c5:fd1b:d267/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

- Run sudo ip link set ens3 promisc on to activate promiscuous mode on the ens3 interface, forcing it to process all network traffic regardless of destination .

```
khine@khine-virtual-machine:/etc/netplan$ cd
khine@khine-virtual-machine:~$ sudo ip link set ens3 promisc on
```

3. 3.2 Installing Suricata

- The sudo apt update command refreshes the system's package index by downloading the latest package information from all configured software repositories.

```
khine@khine-virtual-machine:~$ sudo apt update
[sudo] password for khine:
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:5 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [868 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [160 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21.5 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/main Icons (48x48) [13.4 kB]
Get:10 http://security.ubuntu.com/ubuntu noble-security/main Icons (64x64) [20.0 kB]
Get:11 http://security.ubuntu.com/ubuntu noble-security/main amd64 c-n-f Metadata [7,068 B]
Get:12 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [1,138 kB]
Get:13 http://security.ubuntu.com/ubuntu noble-security/restricted Translation-en [236 kB]
Get:14 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:15 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 c-n-f Metadata [468 B
]
Get:16 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [851 kB]
Get:17 http://security.ubuntu.com/ubuntu noble-security/universe Translation-en [187 kB]
Get:18 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [52.2 kB]
Get:19 http://security.ubuntu.com/ubuntu noble-security/universe Icons (48x48) [45.0 kB]
Get:20 http://security.ubuntu.com/ubuntu noble-security/universe Icons (64x64) [70.7 kB]
Get:21 http://security.ubuntu.com/ubuntu noble-security/universe Icons (64x64@2) [29 B]
Get:22 http://security.ubuntu.com/ubuntu noble-security/universe amd64 c-n-f Metadata [17.0 kB
]
Get:23 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [17.7 kB]
Get:24 http://security.ubuntu.com/ubuntu noble-security/multiverse Translation-en [3,792 B]
```

This necessary maintenance job guarantees the package manager has up to date knowledge of the available software versions and dependencies, and that will not modify any installed packages.

- Then run " sudo add-apt-repository ppa:oisf/suricata-stable" to install the Suricata intrusion detection/prevention system from the newly added otsf/suricata-stable repository. This is the latest stable version of Suricata for network security monitoring.

```
khine@khine-virtual-machine:~$ sudo add-apt-repository ppa:oisf/suricata-stable
Repository: 'Types: deb
URIs: https://ppa.launchpadcontent.net/oisf/suricata-stable/ubuntu/
Suites: noble
Components: main
'
Description:
Suricata IDS/IPS/NSM stable packages
https://suricata.io/
https://oisf.net/
```

- Use sudo apt install suricata to install Suricata, the intrusion detection/prevention system.

```
khine@khine-virtual-machine:~$ sudo apt install suricata
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  isa-support libevent-2.1-7t64 libevent-core-2.1-7t64 libevent-pthreads-2.1-7t64
  libhiredis1.1.0 libhtp2 libhyperscan5 libluajit-5.1-2 libluajit-5.1-common liblzma-dev
  liblzma5 libnet1 libnetfilter-queue1 sse3-support xz-utils
Suggested packages:
  liblzma-doc
The following NEW packages will be installed:
  isa-support libevent-2.1-7t64 libevent-core-2.1-7t64 libevent-pthreads-2.1-7t64
  libhiredis1.1.0 libhtp2 libhyperscan5 libluajit-5.1-2 libluajit-5.1-common liblzma-dev
  libnet1 libnetfilter-queue1 sse3-support suricata
The following packages will be upgraded:
  liblzma5 xz-utils
2 upgraded, 14 newly installed, 0 to remove and 217 not upgraded.
Need to get 7,296 kB of archives.
Show Apps operation, 29.6 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

- Opens the Netplan configuration file so we can set a static IP and DNS.

```
Unpacking suricata-update (1.2.3-1) ...
Setting up libnetfilter-log1:amd64 (1.0.2-1) ...
Setting up oinkmaster (2.0-4.1) ...
Setting up libhtp2 (1:0.5.39-1) ...
Setting up libnet1:amd64 (1.1.6+dfsg-3.1build3) ...
Setting up libhyperscan5 (5.4.0-2) ...
Setting up python3-simplejson (3.17.6-1build1) ...
Setting up libluajit-5.1-common (2.1.0~beta3+dfsg-6) ...
Setting up libevent-core-2.1-7:amd64 (2.1.12-stable-1build3) ...
Setting up suricata-update (1.2.3-1) ...
Setting up libnetfilter-queue1:amd64 (1.0.5-2) ...
Setting up snort-rules-default (2.9.15.1-6build1) ...
Setting up libevent-pthreads-2.1-7:amd64 (2.1.12-stable-1build3) ...
Setting up libhiredis0.14:amd64 (0.14.1-2) ...
Setting up libluajit-5.1-2:amd64 (2.1.0~beta3+dfsg-6) ...
Setting up suricata (1:6.0.4-3) ...
Created symlink /etc/systemd/system/multi-user.target.wants/suricata.service → /lib/systemd/system/suricata.service.
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.10) ...
khine@khine-virtual-machine:~$ sudo nano /etc/netplan/01-netcfg.yaml
```

Inside Netplan.yaml :

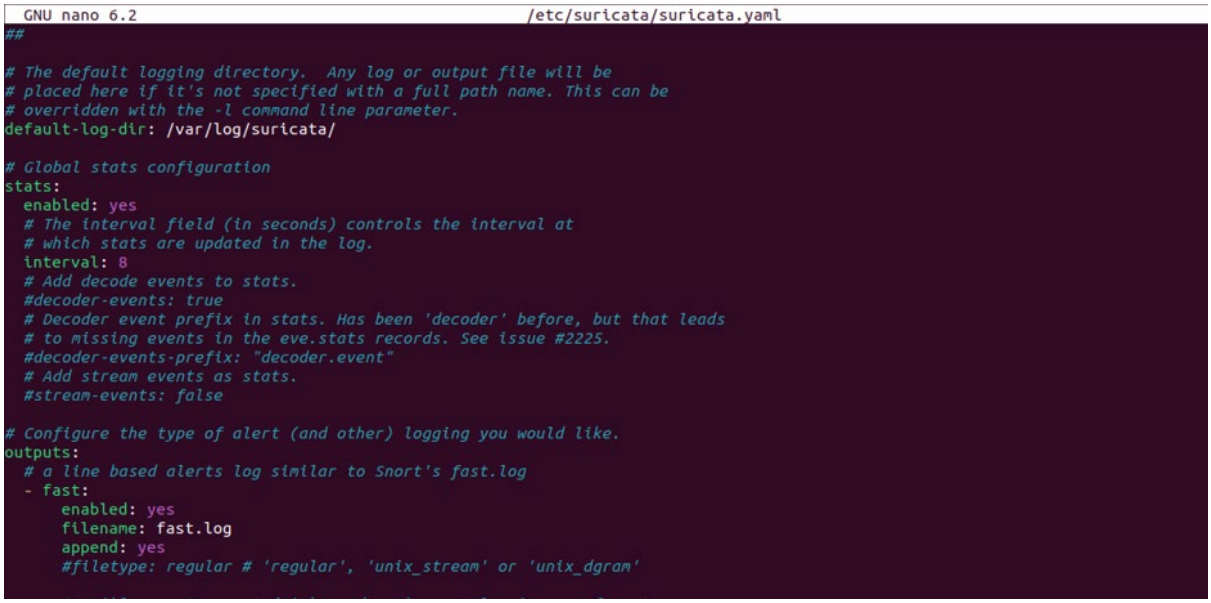
```
  GNU nano 6.2                                    /etc/netplan/01-netcfg.yaml *
network:
  version: 2
  ethernets:
    ens33:
      dhcp4: no
      addresses:
        - 192.168.20.128/24
      gateway4: 192.168.20.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 1.1.1.1
```

This configuration assigns the static IP, default gateway, and DNS servers. The gateway allows the VM to access other networks or the internet.

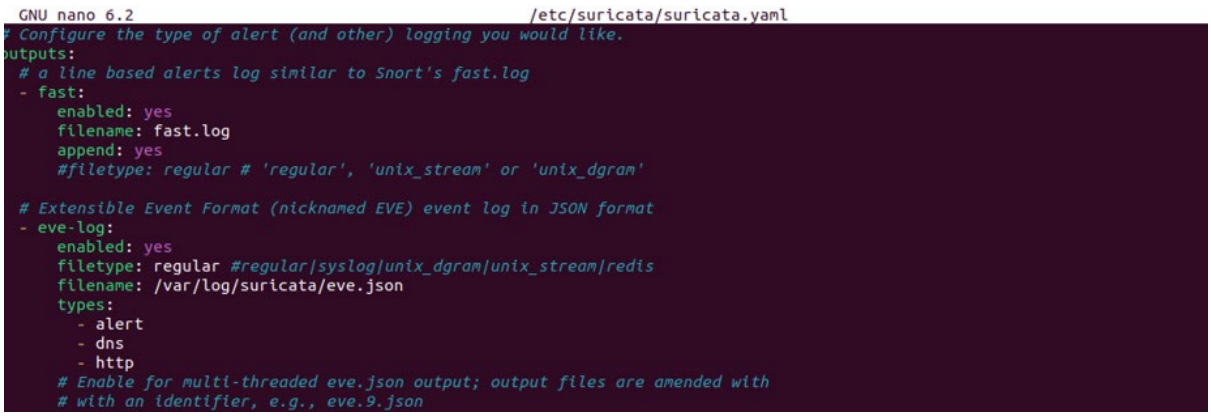
- After that , open the Suricata configuration file using the command sudo nano /etc/suricata/suricata.yaml.
- In the af-packet section, change the interface to ens3 to match VM's network interface. If eth0 is listed, replace it with ens3.
- Then, uncomment the lines for cluster-id, cluster-type, and defrag to enable multi-core performance tuning.
- Add vlan: use-for-tracking: yes under the appropriate section to ensure VLAN-tagged traffic is properly tracked.

```
  GNU nano 6.2                                    /etc/suricata/suricata.yaml *
# Linux high speed capture support
af-packet:
  - interface: ens3
    # Number of receive threads. "auto" uses the number of cores
    #threads: auto
    # Default clusterid. AF_PACKET will load balance packets based on flow.
    cluster-id: 99
    # Default AF_PACKET cluster type. AF_PACKET can load balance per flow or per hash.
    # This is only supported for Linux kernel > 3.1
    # possible value are:
    #  * cluster_flow: all packets of a given flow are sent to the same socket
    #  * cluster_cpu: all packets treated in kernel by a CPU are sent to the same socket
    #  * cluster_qm: all packets linked by network card to a RSS queue are sent to the same
    #  socket. Requires at least Linux 3.14.
    #  * cluster_ebpf: eBPF file load balancing. See doc/userguide/capture-hardware/ebpf-xdp.rst for
    #  more info.
    # Recommended modes are cluster_flow on most boxes and cluster_cpu or cluster_qm on system
    # with capture card using RSS (requires cpu affinity tuning and system IRQ tuning)
    cluster-type: cluster_flow
    # In some fragmentation cases, the hash can not be computed. If "defrag" is set
    # to yes, the kernel will do the needed defragmentation before sending the packets.
    defrag: yes
    # To use the ring feature of AF_PACKET, set 'use-mmap' to yes
    use-mmap: yes
    # Lock memory map to avoid it being swapped. Be careful that over
    # subscribing could lock your system
    #mmap-locked: yes
    # Use tpacket_v3 capture mode, only active if use-mmap is true
    # Don't use it in IPS or TAP mode as it causes severe latency
    tpacket-v3: yes
    vlan:
     use-for-tracking: true
    # Ring size will be computed with respect to "max-pending-packets" and number
```

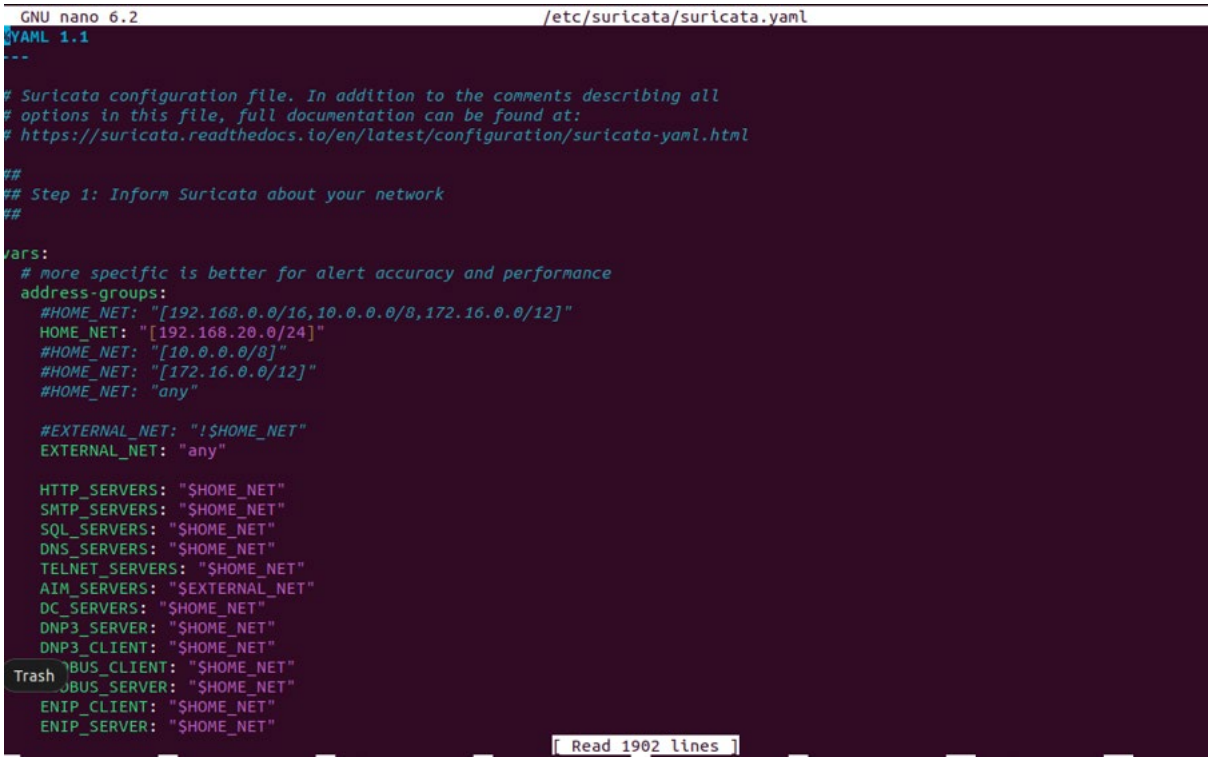
- Edit the Suricata configuration file using sudo nano /etc/suricata/suricata.yaml.
- First, change the default-log-dir to /var/log/suricata/ to define where Suricata will store its log files.
- In the outputs: section, locate and uncomment the line filename: fast.log to enable fast alert logging for quick reference.



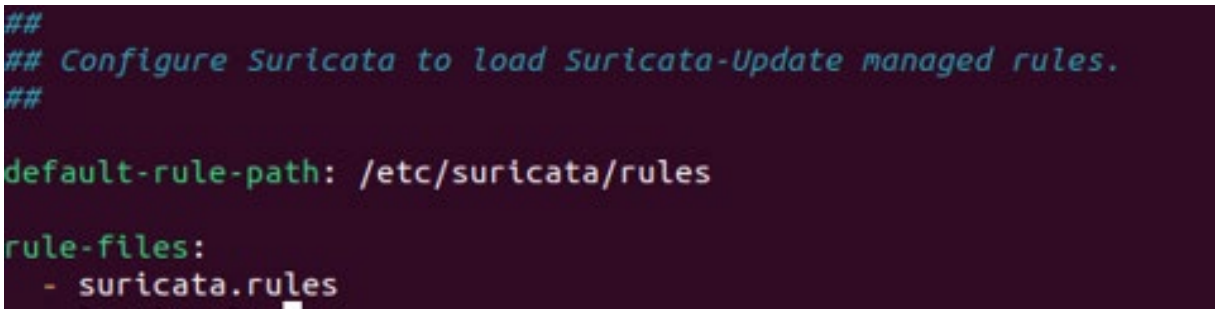
- Then, in the eve-log: section, make sure JSON logging is enabled by setting the filename to /var/log/suricata/eve.json and including types: [alert, dns, http] to log detailed information such as alerts, DNS queries, and HTTP traffic.



- Next, scroll to the vars: section and set HOME_NET to 192.168.20.0/24, defining the internal network range that Suricata should monitor.
- Finally, ensure that EXTERNAL_NET is set to "any" so all traffic outside the home network is treated as potentially suspicious

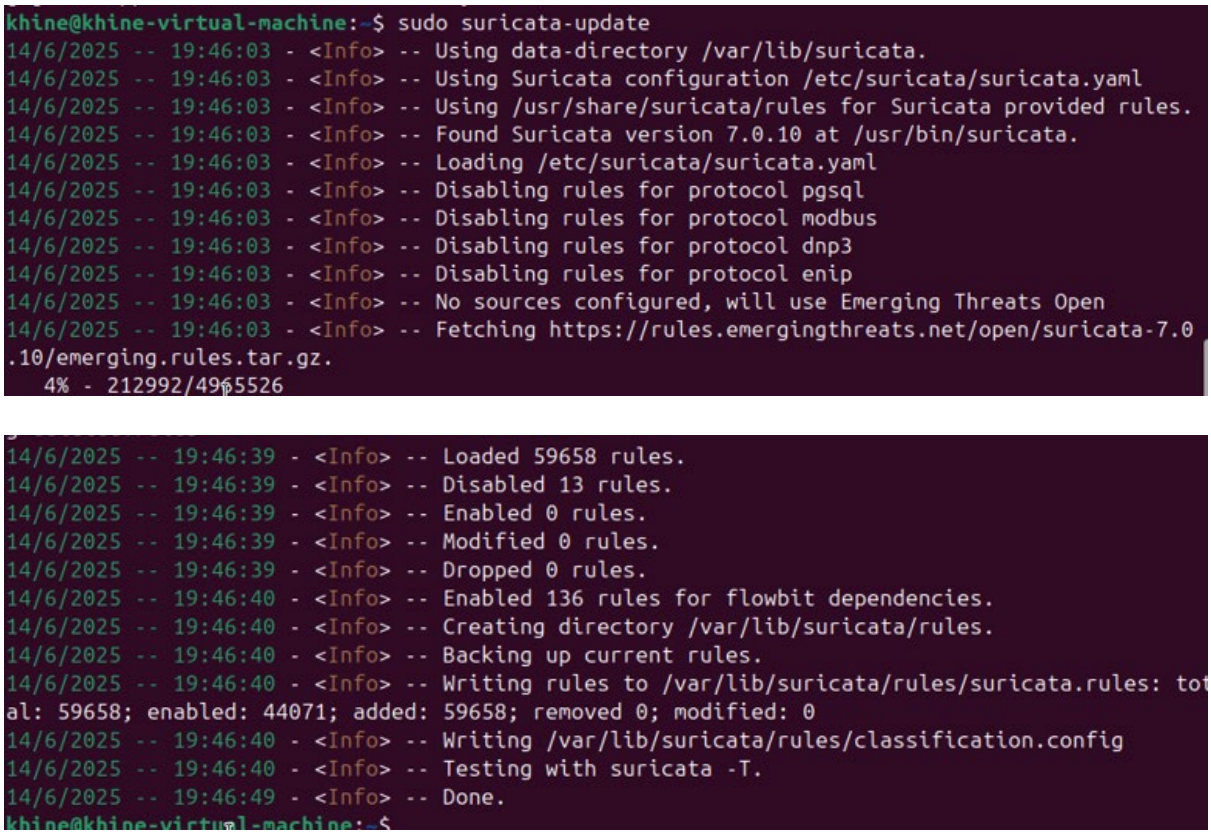


- Ensure default-rule-path points to /var/lib/suricata/rules, where detection rules are stored.



To download the latest ruleset for Suricata, use the built-in tool suricata-update by running the command sudo suricata-update. This will fetch rule sets from external providers, specifically the free Emerging Threats (ET) Open Rules.

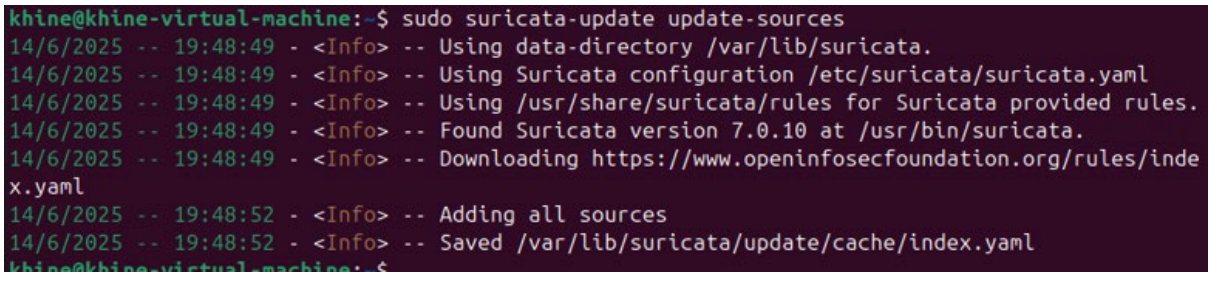
- Once the update is complete, Suricata will save the rules to /var/lib/suricata/rules/suricata.rules.



To view the list of available rule sources, use the following command:

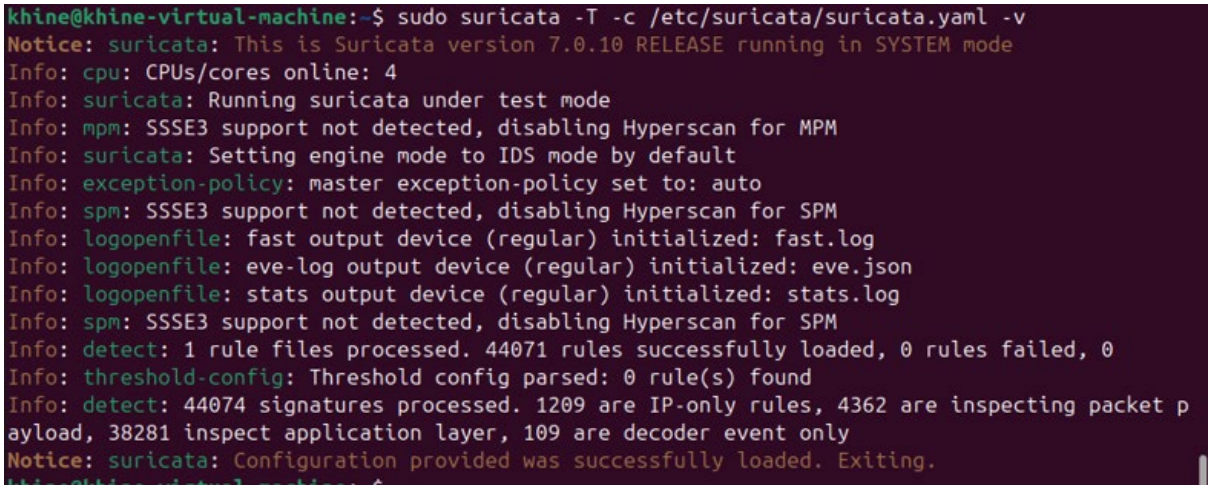
- sudo suricata-update list-sources

This will display the default set of supported rule providers, allowing to choose and enable additional sources as needed.

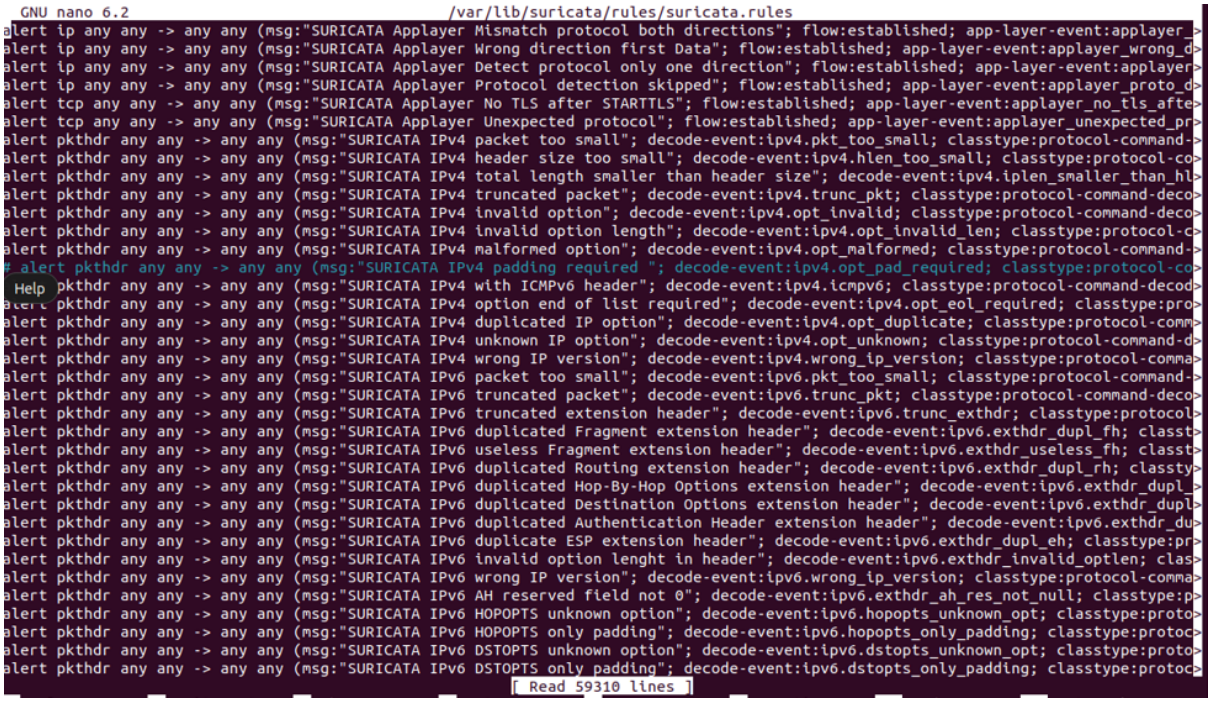


- Suricata has a built-in test mode that will check the configuration file and any included rules for validity.

- Validate the changes from the previous section using the -T flag to run Suricata in test mode. The -v flag will print some additional information, and the -c flag tells Suricata where to find its configuration file:



- Edit the Suricata rules file with sudo nano /var/lib/suricata/rules/suricata.rules to add or customize detection rules as needed.

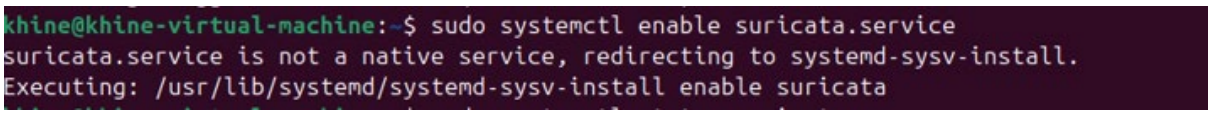


This result shows a portion of the Suricata IDS ruleset in the terminal, specifically highlighting protocol anomaly detection rules for IPv4 and IPv6 headers using decode-event triggers.

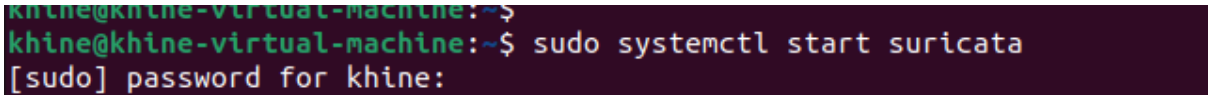
3.3.3 Testing Suricata Setup

- Then restart the Suricata service using command " sudo systemctl restart suricata" .

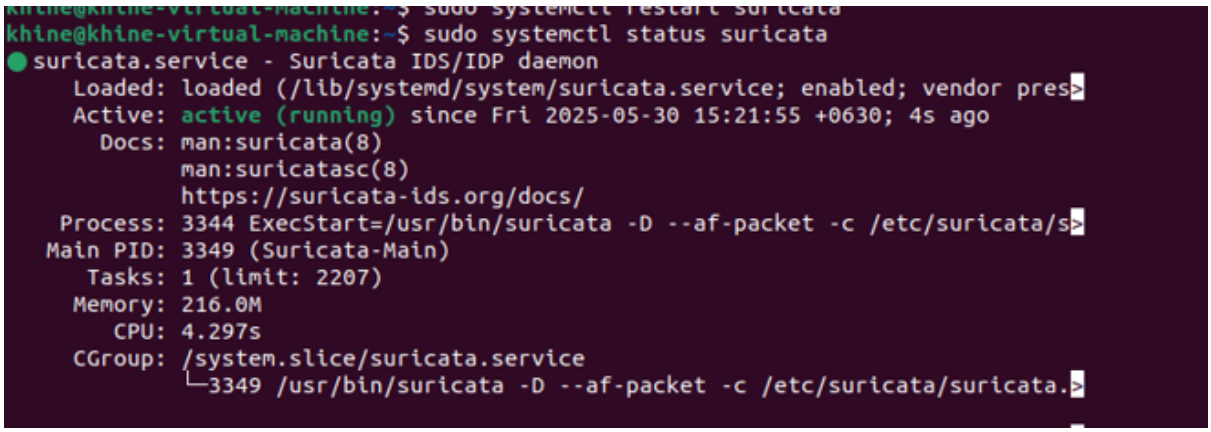
Now that the Suricata configuration and ruleset are properly set up, we can start the Suricata server. First, enable the service to start automatically at boot by running sudo systemctl enable suricata.service.



- Then, start the service using sudo systemctl start suricata.



- To verify Suricata is running properly, check the service status by using sudo systemctl status suricata. You want to see that the service is listed as "active (running)" along with the process memory, process ID and name of the Suricata binary that is being run.



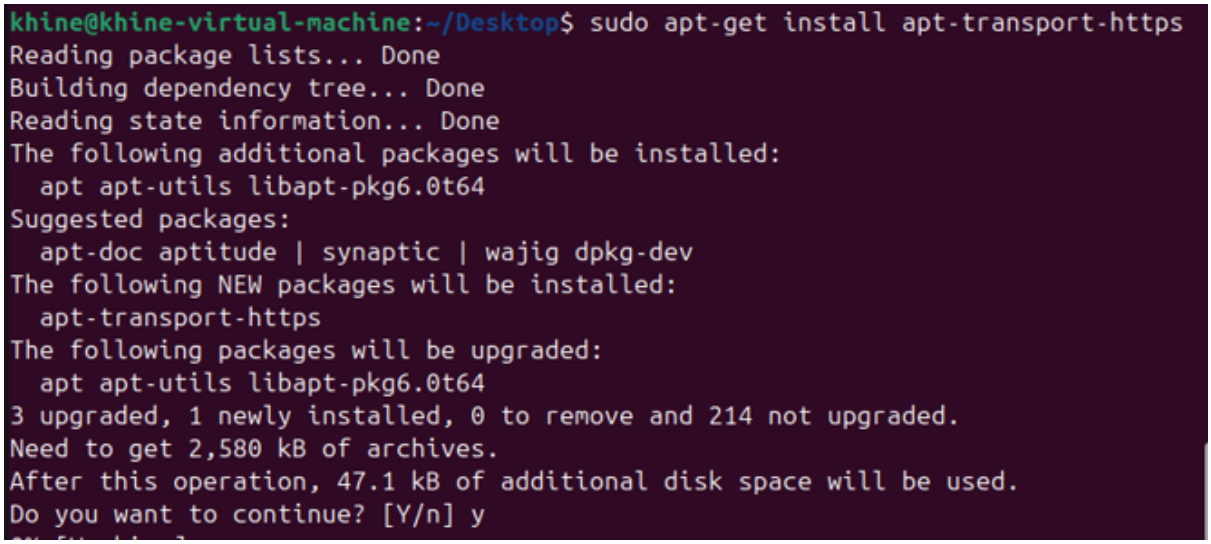
3.4 Phase III: Filebeat Installation for Log Forwarding

To perform log forwarding from Suricata to the ELK Stack, Filebeat was installed on the same Ubuntu VM. The installation was completed with the installation of the Filebeat package and editing the configuration file filebeat.yml. The main points of configuration included the Suricata log directory location, which was specified under the filebeat.inputs section, and the output for the ELK server which was set to 192.168.20.130:5044. The Filebeat service was enabled, started and was set up to ship both Suricata alerts and logs into the centralized SIEM solution for real-time analysis and visualizations.

3.4.1 Installing Filebeat on Ubuntu

To install and set up Filebeat version 7.17.18, begin by installing the necessary dependencies such as artifacts, curl, and wget to enable downloading and managing packages.

- Once the dependencies are installed, use wget or curl to download the Filebeat package version 7.17.18 from the official Elastic website or repository. This ensures the correct version of Filebeat for compatibility with the Elasticsearch stack before proceeding with installation and configuration.



Installing dependencies :

```
khine@khine-virtual-machine:~/Desktop$ sudo apt-get install curl
Reading package lists... Done
Building dependency tree... Done
```

```
khine@khine-virtual-machine:~/Desktop$ wget https://artifacts.elastic.co/downloads/beats/fileb
eat/filebeat-7.17.28-amd64.deb
--2025-06-15 13:25:08--  https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-7.17.2
8-amd64.deb
Resolving artifacts.elastic.co (artifacts.elastic.co)... 34.120.127.130, 2600:1901:0:1d7::
Connecting to artifacts.elastic.co (artifacts.elastic.co)|34.120.127.130|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 37407308 (36M) [application/vnd.debian.binary-package]
Saving to: ‘filebeat-7.17.28-amd64.deb’

filebeat-7.17.28-amd64. 100%[============================>]  35.67M   736KB/s    in 46s

2025-06-15 13:25:55 (788 KB/s) - ‘filebeat-7.17.28-amd64.deb’ saved [37407308/37407308]
```

After downloading the Filebeat package, install it using the command

- sudo dpkg -i filebeat-7.17.28-amd64.deb.

This command installs Filebeat on our system, preparing it for configuration and integration with the monitoring stack.

```
khine@khine-virtual-machine:~/Desktop$ sudo dpkg -i filebeat-7.17.28-amd64.deb
Selecting previously unselected package filebeat.
(Reading database ... 149379 files and directories currently installed.)
Preparing to unpack filebeat-7.17.28-amd64.deb ...
Unpacking filebeat (7.17.28) ...
Setting up filebeat (7.17.28) ...
```

We started the installation process by first updating the package lists using sudo apt update to ensure all repositories are up to date. Then we installed Filebeat using the command sudo apt install filebeat. This allows the Filebeat instance to ship Suricata logs directly to the ELK server, making it easy to reside the Suricata logs on a centralized source for monitoring and analysis.

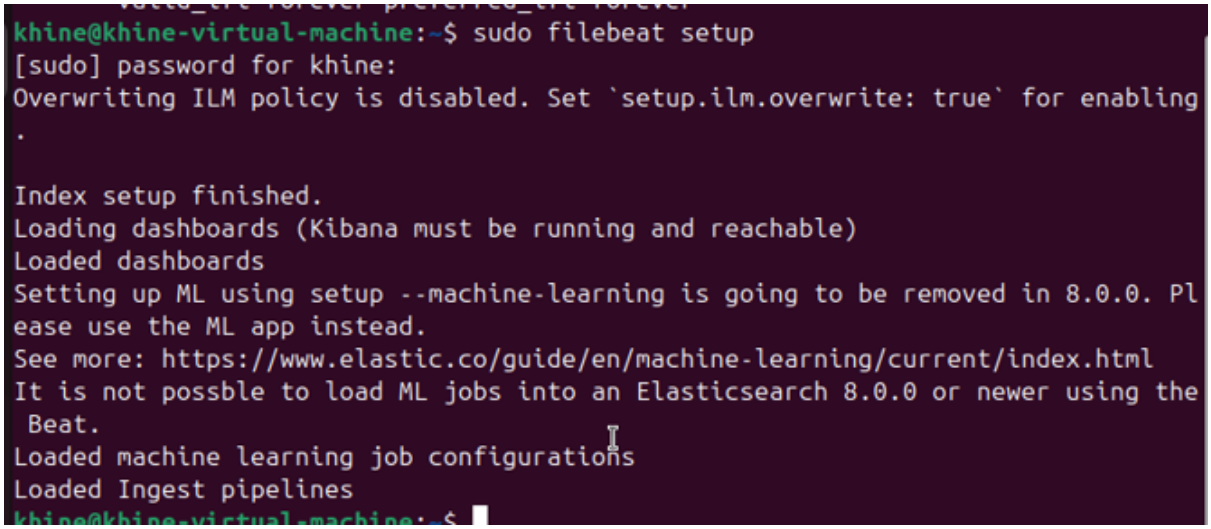
```
khine@khine-virtual-machine:~$ sudo apt update
Hit:1 http://mm.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 https://artifacts.elastic.co/packages/8.x/apt stable InRelease [3,248 B]
Hit:3 http://security.ubuntu.com/ubuntu jammy-security InRelease
Get:4 https://artifacts.elastic.co/packages/8.x/apt stable/main amd64 Packages [75.7 kB]
Hit:5 http://mm.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:6 http://mm.archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:7 https://artifacts.elastic.co/packages/8.x/apt stable/main i386 Packages [3,396 B]
Fetched 82.4 kB in 1s (60.5 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
6 packages can be upgraded. Run 'apt list --upgradable' to see them.
khine@khine-virtual-machine:~$ sudo apt install filebeat
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  filebeat
0 upgraded, 1 newly installed, 0 to remove and 6 not upgraded.
Need to get 63.9 MB of archives.
After this operation, 239 MB of additional disk space will be used.
Get:1 https://artifacts.elastic.co/packages/8.x/apt stable/main amd64 filebeat amd64 8.18.2 [63.9 MB]
Fetched 63.9 MB in 2min 39s (401 kB/s)
Selecting previously unselected package filebeat.
(Reading database ... 202173 files and directories currently installed.)
Preparing to unpack .../filebeat_8.18.2_amd64.deb ...
Unpacking filebeat (8.18.2) ...
Setting up filebeat (8.18.2) ...
khine@khine-virtual-machine:~$
```

- After installation, start the Filebeat and the status of the Filebeat service was checked using sudo systemctl status filebeat.

```
khine@khine-virtual-machine:~/Desktop$ sudo systemctl start filebeat
khine@khine-virtual-machine:~/Desktop$ sudo systemctl status filebeat
● filebeat.service - Filebeat sends log files to Logstash or directly to Elasticsearch.
     Loaded: loaded (/usr/lib/systemd/system/filebeat.service; enabled; preset: enabled)
     Active: active (running) since Sun 2025-06-15 13:27:22 +07; 5s ago
       Docs: https://www.elastic.co/beats/filebeat
   Main PID: 5042 (filebeat)
      Tasks: 8 (limit: 4611)
     Memory: 36.1M (peak: 36.8M)
        CPU: 385ms
     CGroup: /system.slice/filebeat.service
             └─5042 /usr/share/filebeat/bin/filebeat --environment systemd -c /etc/filebeat/f>

Jun 15 13:27:23 khine-virtual-machine filebeat[5042]: 2025-06-15T13:27:23.206+0700        INF>
Jun 15 13:27:23 khine-virtual-machine filebeat[5042]: 2025-06-15T13:27:23.212+0700        INF>
Jun 15 13:27:23 khine-virtual-machine filebeat[5042]: 2025-06-15T13:27:23.213+0700        INF>
Jun 15 13:27:23 khine-virtual-machine filebeat[5042]: 2025-06-15T13:27:23.213+0700        INF>
Jun 15 13:27:23 khine-virtual-machine filebeat[5042]: 2025-06-15T13:27:23.214+0700        INF>
Jun 15 13:27:23 khine-virtual-machine filebeat[5042]: 2025-06-15T13:27:23.214+0700        INF>
Jun 15 13:27:23 khine-virtual-machine filebeat[5042]: 2025-06-15T13:27:23.214+0700        INF>
Jun 15 13:27:23 khine-virtual-machine filebeat[5042]: 2025-06-15T13:27:23.214+0700        INF>
Jun 15 13:27:23 khine-virtual-machine filebeat[5042]: 2025-06-15T13:27:23.214+0700        INF>
lines 1-21/21 (END)
```

The output confirmed that Filebeat was running and active, indicating that the service had started successfully and was ready to forward logs to the ELK stack.

- Next, the command sudo filebeat setup was executed to initialize Filebeat's index templates, dashboards, and other necessary components in Elasticsearch and Kibana.



This setup process completed successfully, confirming that the Filebeat dashboards were loaded and ready for use in visualizing Suricata logs.

3.4.2 Integrating Filebeat for Log Forwarding

- Open the Filebeat configuration file by running sudo nano /etc/filebeat/filebeat.yml.
- Within this file, configure Filebeat to ingest Suricata's EVE JSON logs by setting the input type to log, enabling it, and specifying the log path as "/var/log/suricata/eve.json".

Additionally, enable JSON parsing with the options json.key_under_root: true and json.add_error_key: true to ensure proper extraction of log fields. To allow broader log collection, uncomment or add the line paths: ["/var/log/*.log"], enabling Filebeat to collect logs from all files in the /var/log/ directory.

This setup ensures Suricata logs are correctly ingested and parsed by Filebeat.

```
  GNU nano 6.2                                  /etc/filebeat/filebeat.yml
# ============================== Filebeat inputs ===============================

filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input specific configurations.

# filestream is an input for collecting log messages from files.
- type: log
  enable: true
  paths:
    - /var/log/suricata/eve.json
  json.keys_under_root: true
  json.add_error_key: true

  # Unique ID among all inputs, an ID is required.
  id: my-filestream-id

  # Change to true to enable this input configuration.
  enabled: false

  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /var/log/*.log
    #- c:\programdata\elasticsearch\logs\*
```

This ensures Filebeat reads the JSON output from Suricata for ingestion into the ELK stack.

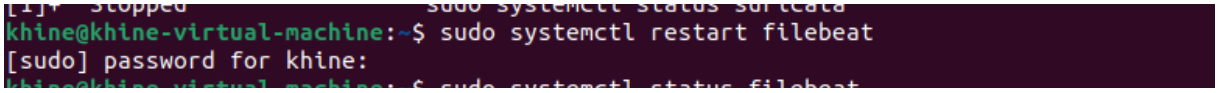
- In output.elasticsearch, set hosts: ["192.168.20.130:9200"]
- Configures Filebeat to send logs to Elasticsearch on the Kali VM.

```
# ================================== Outputs ===================================

# Configure what output to use when sending the data collected by the beat.

# ---------------------------- Elasticsearch Output ----------------------------
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["http://192.168.20.130:9200"]

  # Protocol - either `http` (default) or `https`.
  #protocol: "https"

  # Authentication credentials - either API key or username/password.
  #api_key: "id:api_key"
  #username: "elastic"
  #password: "changeme"
```

This setting directs Filebeat to forward logs to Elasticsearch on the Kali VM (IP 192.168.20.130).

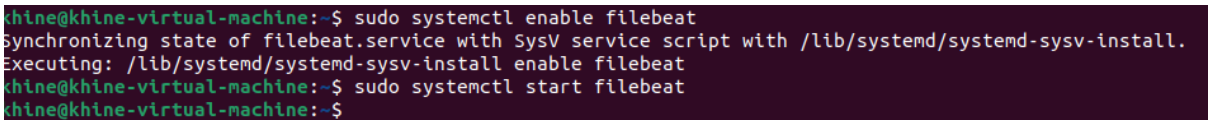
```
# ================================== Kibana ====================================

# Starting with Beats version 6.0.0, the dashboards are loaded via the Kibana API.
# This requires a Kibana endpoint configuration.
setup.kibana:

  # Kibana Host
  # Scheme and port can be left out and will be set to the default (http and 5601)
  # In case you specify and additional path, the scheme is required: http://localhost:5601/path
  # IPv6 addresses should always be defined as: https://[2001:db8::1]:5601
  host: "192.168.20.130:5601"

  # Kibana Space ID
```

- Restart Filebeat with sudo systemctl restart filebeat command .

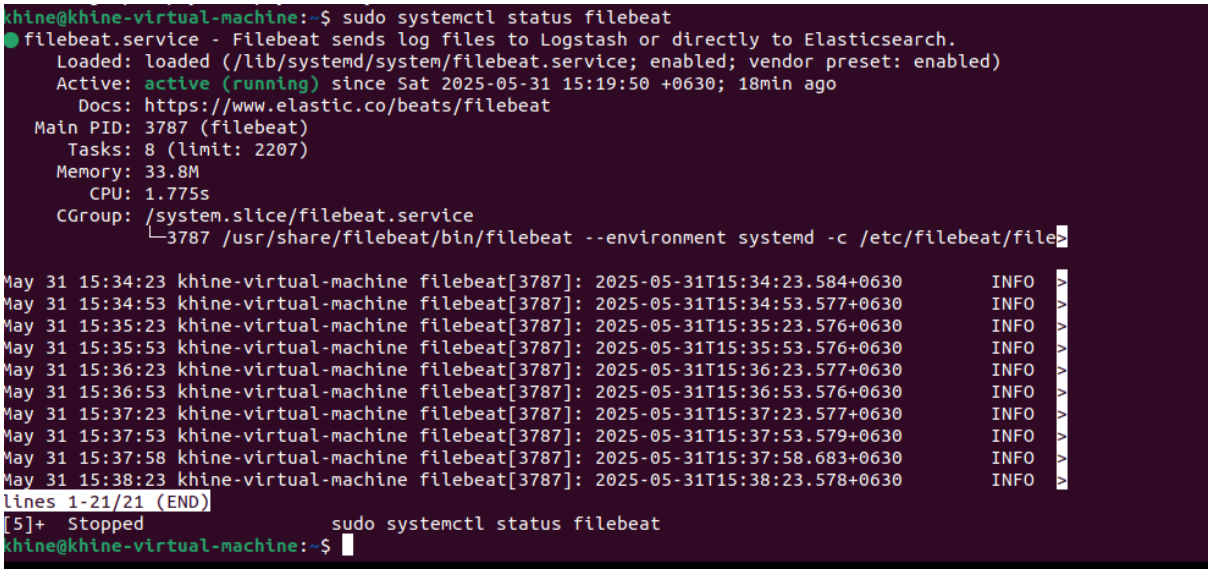


3.4.3 Enable and Monitor Filebeat

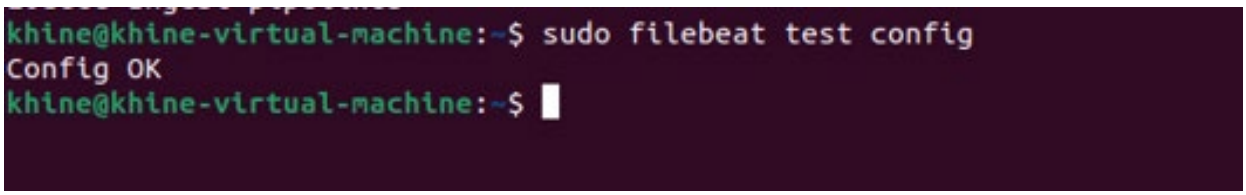
- Finally, Filebeat was enabled and started:



- Run sudo systemctl status filebeat to check that Filebeat is running.



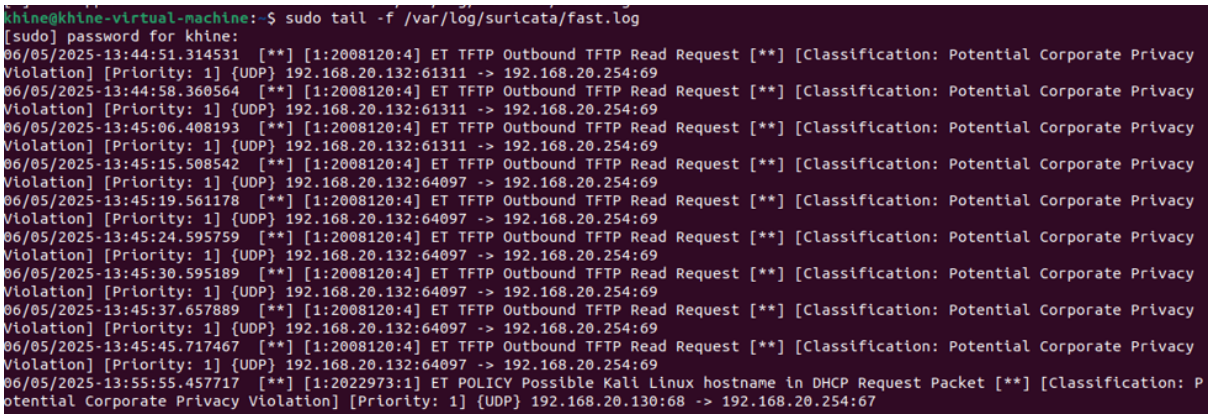
These commands ensure filebeat runs on boot and confirm that the service is active and running without errors.



These steps ensure Filebeat starts on boot and runs continuously, forwarding logs to the ELK stack for analysis.

- To monitor Suricata alerts in real-time, run the command sudo tail -f /var/log/suricata/fast.log.

- This continuously displays new entries in the fast.log file, allowing immediate visibility into detected network events and alerts as they occur.



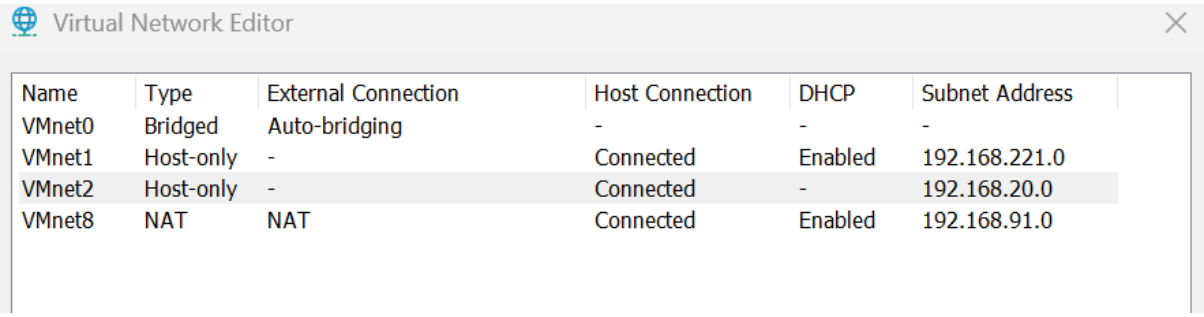
3.5 Phase IV: ELK Stack Installation on Kali VM

The ELK stack, which includes Elasticsearch and Logstash and Kibana, was deployed on Kali Linux onto a virtual machine (192.168.20.130). The deployment began where Java Runtime Environment was installed first and was followed by downloading and installing each component of the stack(Elasticsearch, Logstash, and Kibana). The installation of Elasticsearch included configuring Elasticsearch, such that Elasticsearch would listen on the local network and to always index the incoming data. Once I configured Elasticsearch, I set up Logstash, where it was configured to receive beats input on port 5044(logging port for Suricata) and it was also configured to parse Suricata logs, that are in JSON-format, into structured data that could be analyzed.

Kibana was then configured to utilize data from Elasticsearch, so it would act like a more user-friendly interface of the ELK stack which could be easily accessed through a web browser by reading the URL: http://192.168.20.130:5601. Within Kibana, an index pattern was created for filebeat-* which made it easy to visualize Suricata alerts using time-based filtering on the dashboards. The dashboards were configured to emphasize security events relevant to critical security situational awareness, such as traffic anomalies, attacks, and network activity.

3.5.1 Kali VM Network Setup

A host-only network adapter (VMnet2) with the subnet 192.168.20.0 was added to both the Ubuntu and Kali VMs to enable isolated communication. In EVE-NG, each VM has two NICs: one NAT (VMnet8) for internet access and one host-only (VMnet2) for internal network traffic. Ubuntu runs Suricata IDS, and Kali runs the ELK stack, allowing them to communicate securely within the same subnet.



- First , assign the dynamic IP address 192.168.20.130 with a subnet mask of /24 (meaning 255.255.255.0) to the network interface eth0 on the Kali VM. This allows the Kali VM to communicate with other devices on the same subnet (such as the Suricata machine).
- After that run “ sudo ip link set eth0 up “ to activate the eth0 network interface, making it ready to send and receive packets.

```
┌──(kali㉿kali)-[~]
└─$ sudo ip link set eth0 up

┌──(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.20.130  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 00:0c:29:ac:49:8b  txqueuelen 1000  (Ethernet)
        RX packets 345  bytes 20883 (20.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 64  bytes 11635 (11.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0×10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 7031  bytes 3151672 (3.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7031  bytes 3151672 (3.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- Sends ICMP echo requests to the Suricata machine at 192.168.20.150 to verify that the network connectivity between Kali and Suricata is working.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:ac:49:8b brd ff:ff:ff:ff:ff:ff
    inet 192.168.20.130/24 brd 192.168.20.255 scope global dynamic noprefixroute eth0
       valid_lft 1649sec preferred_lft 1649sec
    inet6 fe80::7a4f:abf7:cd4c:4e31/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

┌──(kali㉿kali)-[~/Desktop]
└─$ ping 192.168.20.128
PING 192.168.20.128 (192.168.20.128) 56(84) bytes of data.
64 bytes from 192.168.20.128: icmp_seq=1 ttl=64 time=6.67 ms
64 bytes from 192.168.20.128: icmp_seq=2 ttl=64 time=1.01 ms
64 bytes from 192.168.20.128: icmp_seq=3 ttl=64 time=0.928 ms
64 bytes from 192.168.20.128: icmp_seq=4 ttl=64 time=2.22 ms
```

- Alternatively sent ping request from Suricata vm to kali using kali IP .

```
khine@khine-virtual-machine:~$ ping 192.168.20.130
PING 192.168.20.130 (192.168.20.130) 56(84) bytes of data.
64 bytes from 192.168.20.130: icmp_seq=1 ttl=64 time=0.445 ms
64 bytes from 192.168.20.130: icmp_seq=2 ttl=64 time=0.400 ms
64 bytes from 192.168.20.130: icmp_seq=3 ttl=64 time=1.48 ms
64 bytes from 192.168.20.130: icmp_seq=4 ttl=64 time=1.16 ms
64 bytes from 192.168.20.130: icmp_seq=5 ttl=64 time=0.347 ms
64 bytes from 192.168.20.130: icmp_seq=6 ttl=64 time=0.400 ms
64 bytes from 192.168.20.130: icmp_seq=7 ttl=64 time=1.07 ms
64 bytes from 192.168.20.130: icmp_seq=8 ttl=64 time=0.471 ms
```

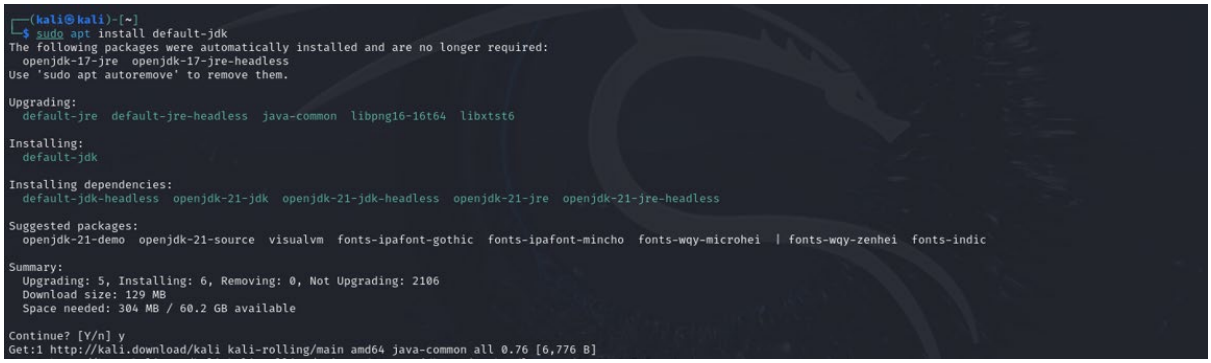
The two IP addresses can successfully communicate with each other within the vmnet2 network, as confirmed by successful bidirectional ping requests.

3.5.2 Installing Elasticsearch

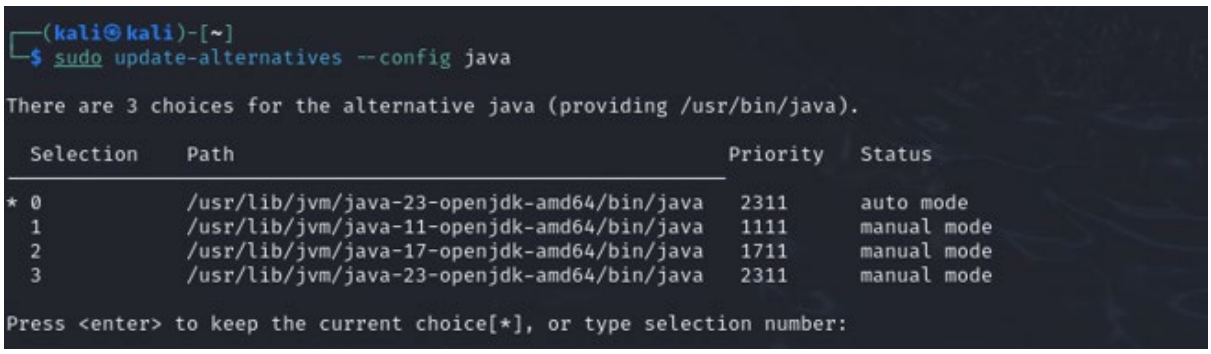
Before installation, the Kali VM was updated to ensure compatibility with the latest package repositories:

- Run sudo apt update command

- Since Elasticsearch depends on Java to run, the default Java Development Kit (JDK) was installed using the command :

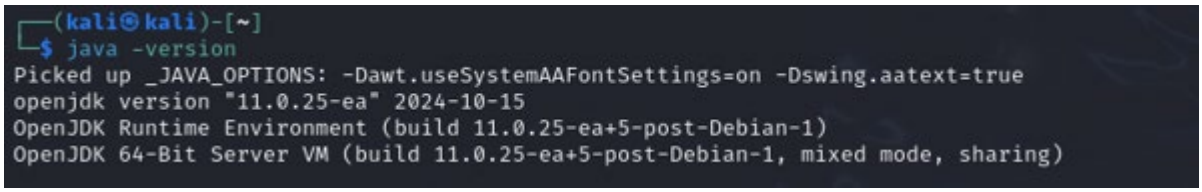


Java provides the necessary runtime environment for Elasticsearch and Logstash components.



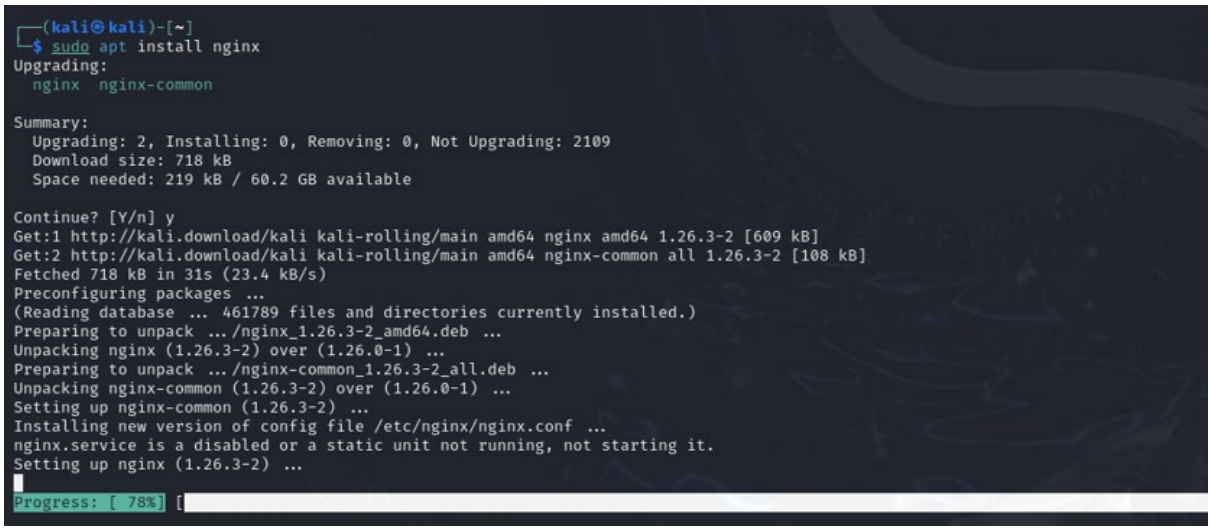
The command sudo update-alternatives --config java brings up a list of all installed Java binaries managed under the update-alternatives system.

- After installation, the command java --version can be used to confirm that Java has been properly installed .

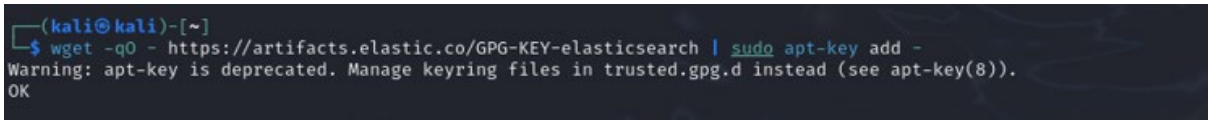


In this case, the output shows OpenJDK version 17.0.9, confirming a successful setup.

- Next, the Nginx web server is installed using sudo apt install nginx, which will later be used as a reverse proxy for Kibana to allow HTTP access via a browser.



- To ensure the authenticity of Elastic packages, the Elasticsearch GPG key was first imported using the command :



This securely downloads and adds Elastic's GPG key to the system's trusted keyring.

- Following that, sudo apt install apt-transport-https is used to enable the system to retrieve packages over HTTPS.

```
┌──(kali㉿kali)-[~]
└─$ sudo apt install apt-transport-https
Upgrading:
  apt  apt-utils

Installing:
  apt-transport-https

Installing dependencies:
  libapt-pkg7.0  sqv

Summary:
  Upgrading: 2, Installing: 3, Removing: 0, Not Upgrading: 2107
  Download size: 3,552 kB
  Space needed: 5,808 kB / 60.2 GB available
```

Next, the Elastic repository was added to APT's sources list using :

```
┌──(kali㉿kali)-[~]
└─$ echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee -a /etc/apt/sources.list.d/elastic-7.x.list
deb https://artifacts.elastic.co/packages/7.x/apt stable main

┌──(kali㉿kali)-[~]
```

This step enables the system to install Elastic components like Elasticsearch, Kibana, and Logstash directly via the package manager.

- After adding the Elastic repository, the package index was refreshed using sudo apt update to ensure the system recognized the newly added source.

```
┌──(kali㉿kali)-[~]
└─$ sudo apt update
[sudo] password for kali:
Hit:1 https://artifacts.elastic.co/packages/7.x/apt stable InRelease
Get:3 https://artifacts.elastic.co/packages/8.x/apt stable InRelease [3,248 B]
Get:2 http://kali.download/kali kali-rolling InRelease [41.5 kB]
```

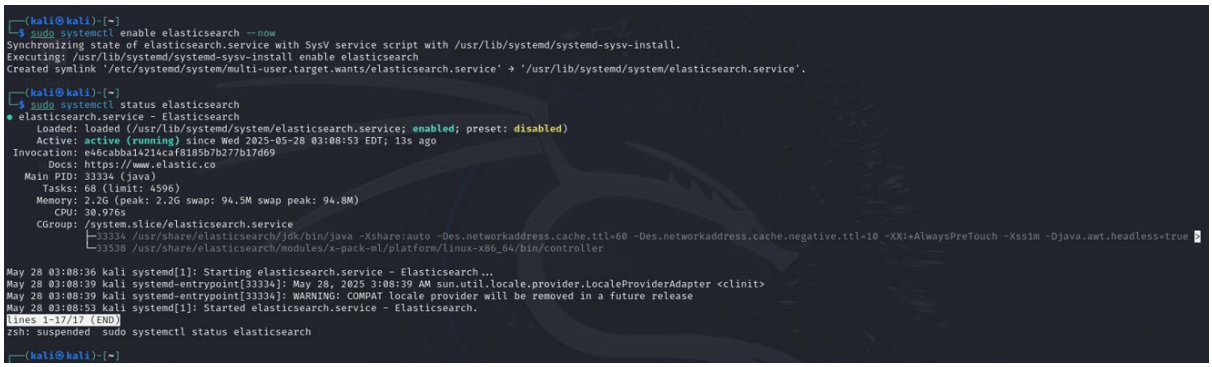
3.5.3 Downloading and Installing Elasticsearch

- After updating package lists, Elasticsearch was installed using "sudo apt install -y elasticsearch" .

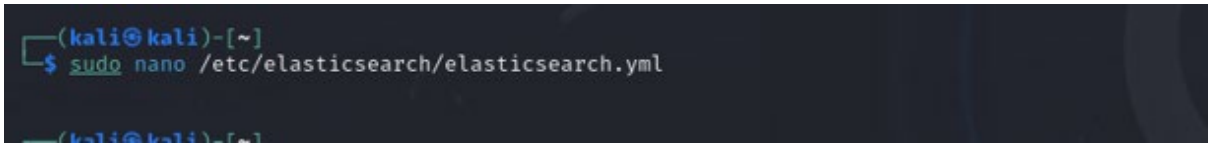
```
┌──(kali㉿kali)-[~]
└─$ sudo apt install -y elasticsearch
Installing:
  elasticsearch

Summary:
  Upgrading: 0, Installing: 1, Removing: 0, Not Upgrading: 2111
  Download size: 0 B / 325 MB
  Space needed: 542 MB / 59.4 GB available

Selecting previously unselected package elasticsearch.
(Reading database ... 414468 files and directories currently installed.)
Preparing to unpack .../elasticsearch_7.17.28_amd64.deb ...
Creating elasticsearch group ... OK
Creating elasticsearch user ... OK
Unpacking elasticsearch (7.17.28) ...
Setting up elasticsearch (7.17.28) ...
### NOT starting on installation, please execute the following statements to configure elasticsearch service to start automatically using systemd
 sudo systemctl daemon-reload
 sudo systemctl enable elasticsearch.service
### You can start elasticsearch service by executing
 sudo systemctl start elasticsearch.service
Created elasticsearch keystore in /etc/elasticsearch/elasticsearch.keystore

┌──(kali㉿kali)-[~]
└─$ sudo ls /etc/elasticsearch

elasticsearch.keystore  elasticsearch-plugins.example.yml  elasticsearch.yml  jvm.options  jvm.options.d  log4j2.properties  role_mapping.yml  roles.yml  users  users_roles
```

These steps install Elasticsearch and allow it to run as a service, forming the database layer of the ELK stack.Then check elasticsearch config directory using ls command and ensure that the configuration files such as elasticsearch.yaml where we can make changes .

- Enabling and starting the service ensures Elasticsearch runs on boot and is active for log ingestion.

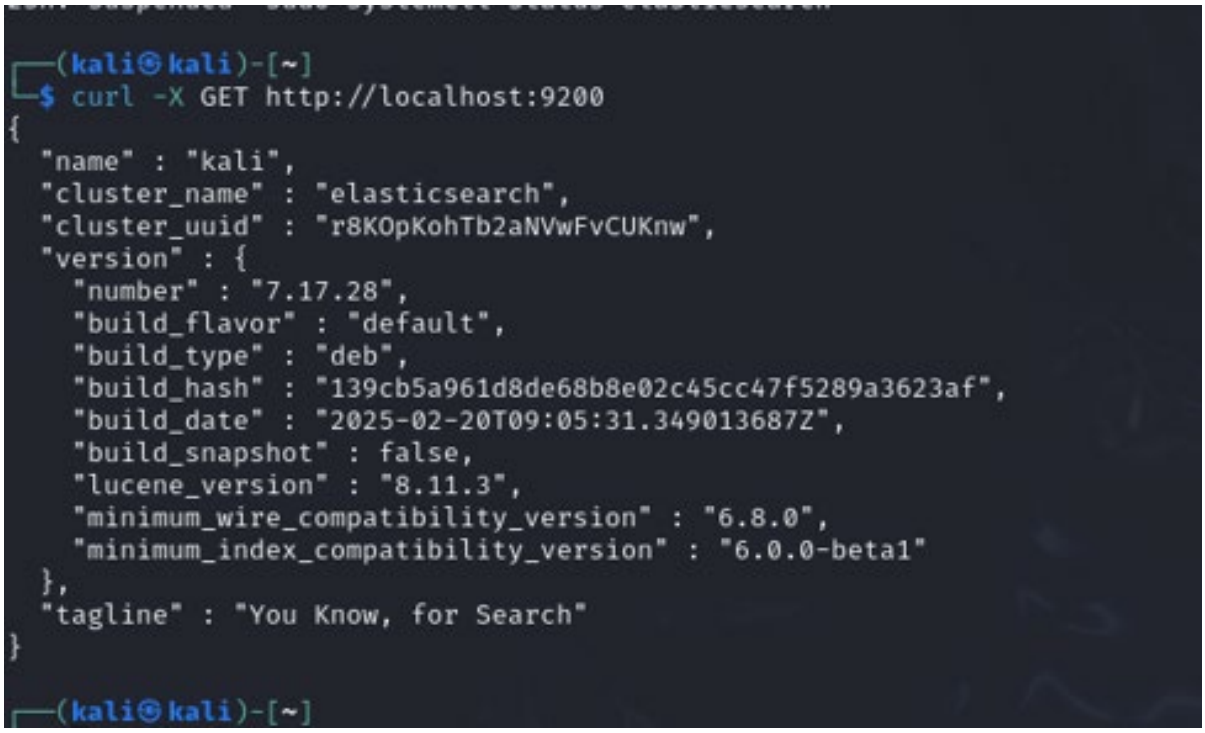


- Run sudo nano /etc/elasticsearch/elasticsearch.yml to open Elasticsearch config file.



- Opens the main config file. Set the network.host to localhost to bind elasticsearch to only listen to localhost for security (so no one from outside can directly access it) .

- To verify that Elasticsearch was successfully installed and running, the command curl -X GET http://localhost:9200 was executed.



Sends a GET request to check if Elasticsearch is reachable .

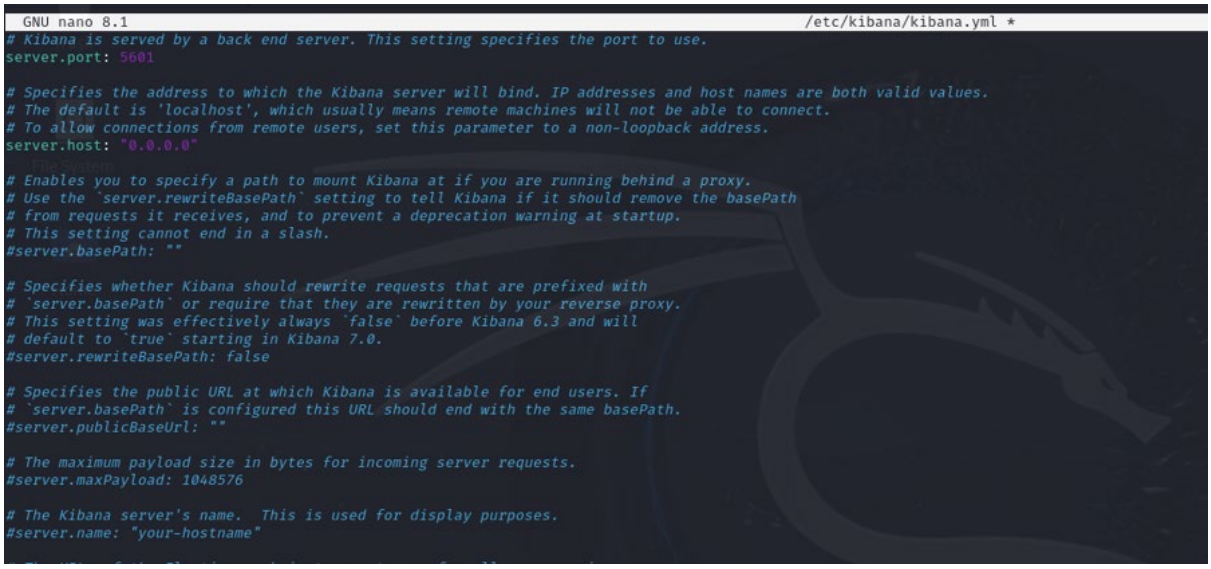
3.5.3 Installing Kibana

- With Elasticsearch up and running, Kibana was installed using apt .

- In cases where the installation failed or was interrupted, the command sudo dpkg --configure -a was used to fix any broken or partially installed packages.

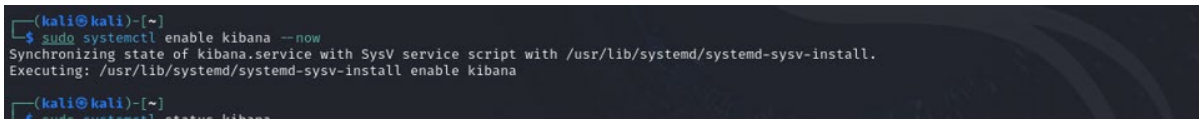
- After installation, the Kibana configuration file was edited to bind the service to the correct IP address.

The following line was added or updated:

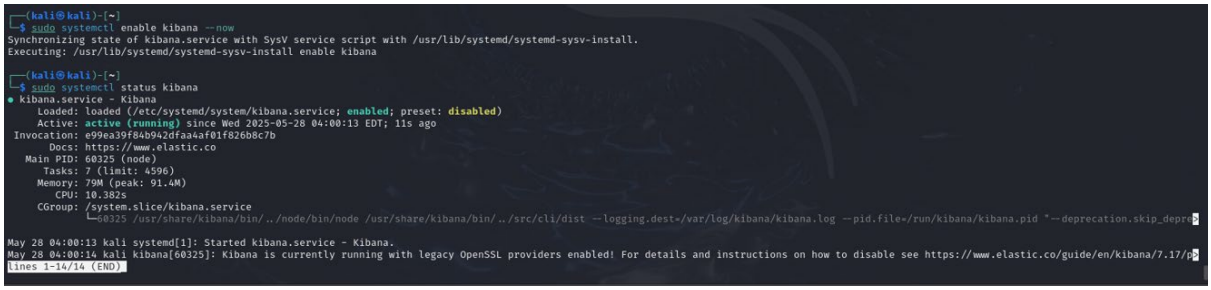


Specifically, the line server.host: "0.0.0.0" was added or updated to allow Kibana to accept connections from any network interface, including the VM's IP.

- Finally, Kibana was enabled and started using sudo systemctl enable kibana followed by sudo systemctl start kibana.



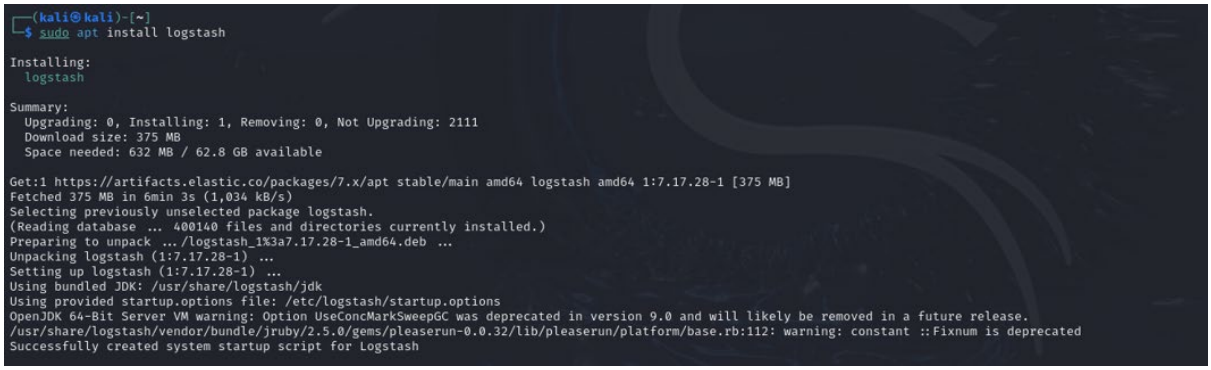
- To confirm that Suricata is running properly after all configurations and integrations, the command sudo systemctl status suricata was executed.
- The output indicated that the Suricata service was active and running, verifying that it was successfully monitoring network traffic and ready to forward alerts to the ELK stack for analysis.



This confirms Kibana is active and ready to receive logs from Filebeat.

3.5.4 Installing Logstash (Optional for Parsing)

- Although not strictly required for Filebeat Elasticsearch ingestion, Logstash was installed for potential future enrichment or parsing.



- After completing the Logstash installation, the ls command was run to verify that the logstash.yml configuration file was properly installed and located in the expected directory.

- To confirm the installation further, a pipeline configuration directory was created at /etc/logstash/conf.d .
- Within this directory, necessary configuration files were added or updated to define Logstash’s data processing pipelines, ensuring it can correctly receive, process, and forward logs as required.

```
┌──(kali㉿kali)-[~]
└─$ sudo apt list --installed | grep logstash

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

logstash/stable,now 1:7.17.28-1 amd64 [installed]

┌──(kali㉿kali)-[~]
└─$ sudo mkdir -p /etc/logstash/conf.d

┌──(kali㉿kali)-[~]
└─$ sudo nano /etc/logstash/conf.d/suricata.conf
```

In suricata.conf file , add :

```
File  Actions  Edit  View  Help
  GNU nano 8.1                                  /etc/logstash/conf.d/suricata.conf *
input {
  file {
    path ⇒ "/var/log/suricata/eve.json"
    start_position ⇒ "beginning"
    codec ⇒ "json"
  }
}

filter {
  if [event_type] == "alert" {
    mutate {
      add_field ⇒ { "alert_msg" ⇒ "%{[alert][signature]}" }
    }
  }
}

output {
  elasticsearch {
    hosts ⇒ ["http://localhost:9200"]
    index ⇒ "suricata-%{+YYYY.MM.dd}"
  }
}
```

Logstash can be configured to filter and transform logs before they're stored in Elasticsearch.

3.5.5 Add Firewall Rules (Optional)

Optionally, firewall rules were added to allow incoming connections to port 5601, which is used by Kibana's web interface. This was done by running the command sudo ufw allow 5601/tcp, permitting TCP traffic on that port to reach the VM server.

```
┌──(kali㉿kali)-[~]
└─$ sudo ufw allow 5601/tcp
Rule added
Rule added (v6)
```

- Check firewall to verify allowed ports .

```
┌──(kali㉿kali)-[~]
└─$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
Anywhere on eth0           ALLOW       Anywhere
9200/tcp                   ALLOW       Anywhere
5601                       ALLOW       Anywhere
5601/tcp                   ALLOW       Anywhere
Anywhere (v6) on eth0      ALLOW       Anywhere (v6)
9200/tcp (v6)              ALLOW       Anywhere (v6)
5601 (v6)                  ALLOW       Anywhere (v6)
5601/tcp (v6)              ALLOW       Anywhere (v6)

Anywhere                   ALLOW OUT   Anywhere on eth0
Anywhere (v6)              ALLOW OUT   Anywhere (v6) on eth0
```

The status of Logstash was confirmed by executing sudo systemctl status logstash, which showed that Logstash was active and ready to receive logs forwarded from Filebeat for further processing.

```
┌──(kali㉿kali)-[~]
└─$ sudo systemctl status logstash
● logstash.service - logstash
     Loaded: loaded (/etc/systemd/system/logstash.service; enabled; preset: disabled)
     Active: active (running) since Wed 2025-05-28 03:41:36 EDT; 10s ago
 Invocation: 9b8acac453b840de93e32aa28da25e30
   Main PID: 50836 (java)
      Tasks: 15 (limit: 4596)
     Memory: 307.8M (peak: 308.2M)
        CPU: 16.590s
     CGroup: /system.slice/logstash.service
             └─50836 /usr/share/logstash/jdk/bin/java -Xms1g -Xmx1g -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:+UseCMSInitiatingOccupancyOnly -Djava.awt.headless=true -Dfile.en

May 28 03:41:36 kali systemd[1]: Started logstash.service - logstash.
May 28 03:41:36 kali logstash[50836]: Using bundled JDK: /usr/share/logstash/jdk
May 28 03:41:36 kali logstash[50836]: OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a future release.
lines 1-14/14 (END)
```

- Once configuration was complete, Suricata was started and enabled:

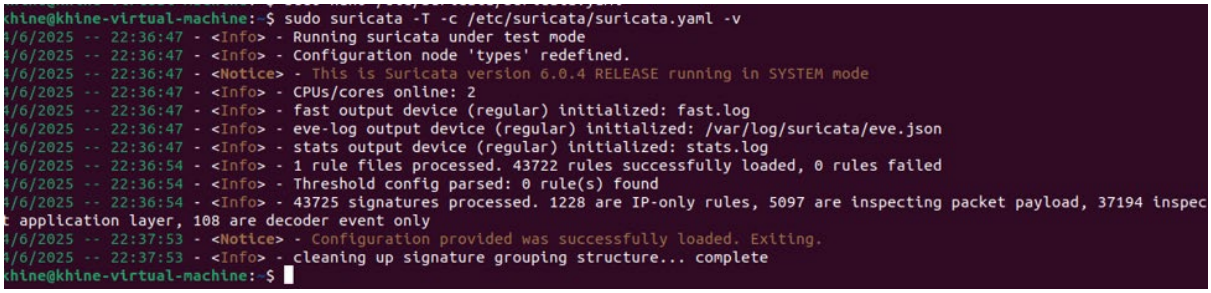
```
khine@khine-virtual-machine:~$ sudo systemctl enable suricata
[sudo] password for khine:
suricata.service is not a native service, redirecting to systemd-sysv-install
Executing: /usr/lib/systemd/systemd-sysv-install enable suricata
khine@khine-virtual-machine:~$ sudo systemctl start suricata
```

- Start the Suricata service, enabling it to begin monitoring traffic based on the configuration file (usually /etc/suricata/suricata.yaml).
- It loads the rule set and interfaces we defined, ensuring Suricata is actively analyzing packets in real-time.

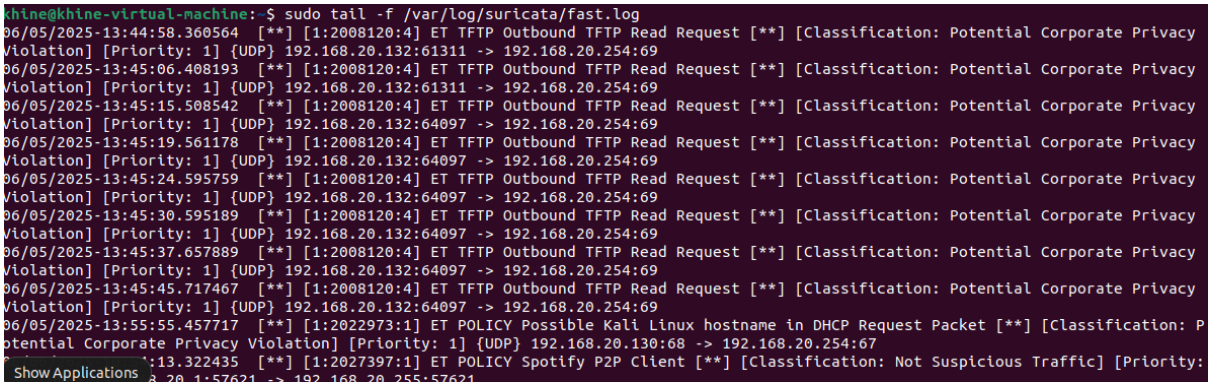
```
khine@khine-virtual-machine:~$ sudo systemctl status suricata
● suricata.service - Suricata IDS/IDP daemon
     Loaded: loaded (/lib/systemd/system/suricata.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2025-06-14 14:16:08 +0630; 4min 32s ago
       Docs: man:suricata(8)
             man:suricatasc(8)
             https://suricata-ids.org/docs/
   Main PID: 904 (Suricata-Main)
      Tasks: 8 (limit: 2207)
     Memory: 315.0M
        CPU: 1min 26.274s
     CGroup: /system.slice/suricata.service
             └─904 /usr/bin/suricata -D --af-packet -c /etc/suricata/suricata.yaml --pidfile /run/suricata.pid

Jun 14 14:16:07 khine-virtual-machine systemd[1]: Starting Suricata IDS/IDP daemon...
```

This indicates whether Suricata is currently running and its status (active). It is a good practice to check the service status for debugging purposes. If Suricata is unable to be started due to some sort of configuration error like bad interface arguments or rule file syntax, etc. This helps confirm that the service is working.

- Before starting Suricata fully, the configuration was tested using the command suricata -T -c /etc/suricata/suricata.yaml, which validates the syntax and rule loading without running the service.
- This test step is essential to ensure the configuration is correct and to prevent failures during actual deployment.

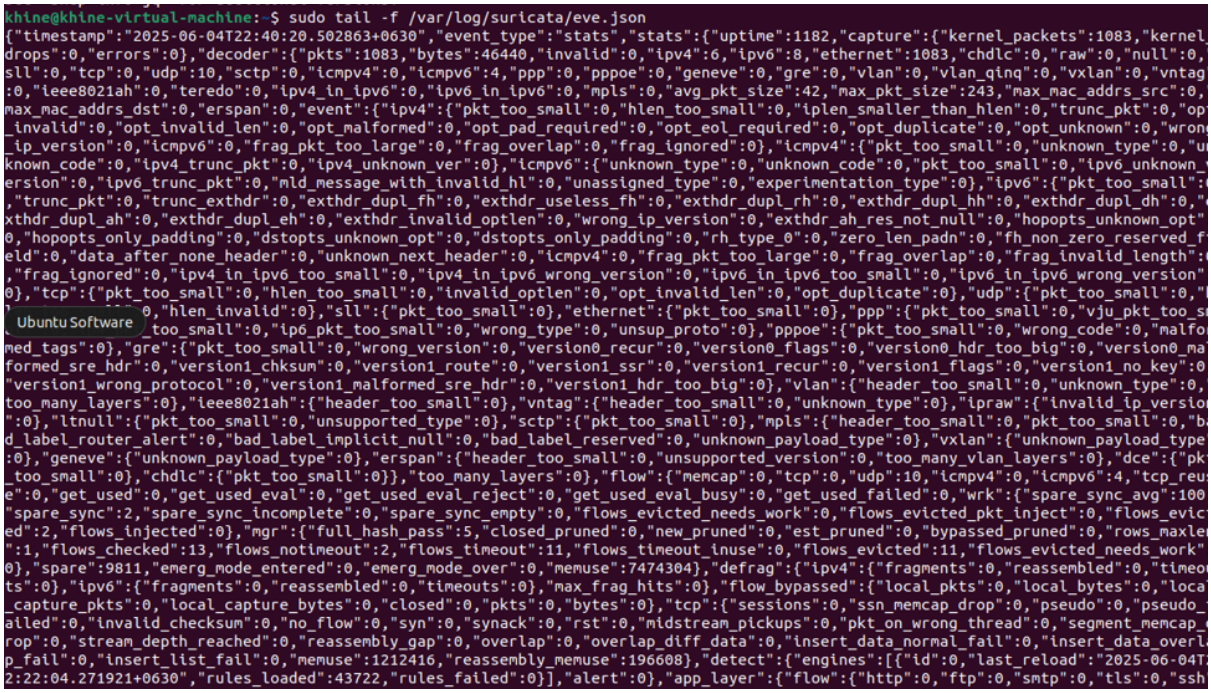


- Streams live logs from Suricata's fast.log file, which records alerts in a concise, human-readable format.



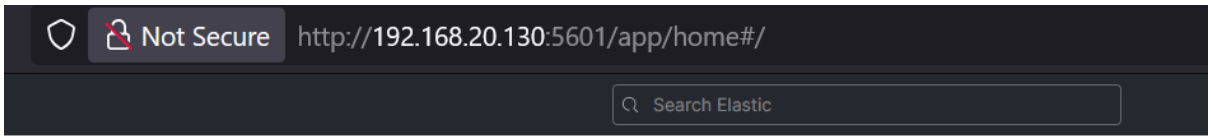
Monitoring logs helps verify that Suricata is actively analyzing traffic and detecting suspicious activities. It also provides immediate feedback, allowing us to confirm that the rule set is generating alerts as expected. This step is especially useful during testing or troubleshooting to ensure real-time detection is working.

- To monitor Suricata's detailed event logs in real-time, the command tail -f /var/log/suricata/eve.json was used.

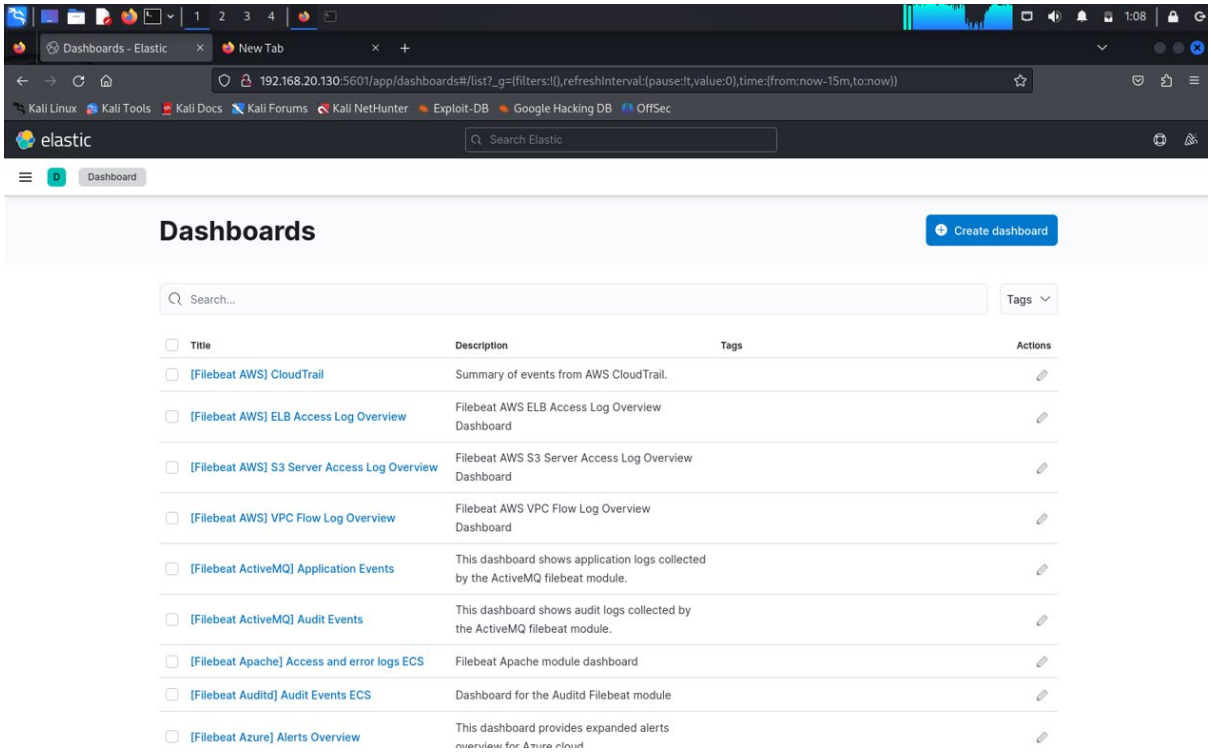


The eve.json file includes detailed event data such as DNS queries, HTTP requests, and alerts. This is especially important for integrating with log management tools like Filebeat and ELK Stack for centralized analysis and visualization.

- Run http://192.168.20.130:5601 on browser to open the Kibana web interface, allowing us to visualize and analyze Suricata alerts and logs.



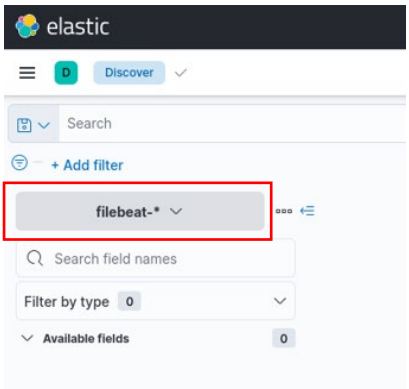
Kibana connects to Elasticsearch and provides a user-friendly interface for querying, filtering, and visualizing network security data collected by Suricata and Filebeat.



Then, in Kibana click Dashboard to see the existing visualizations, titles, descriptions, and tags (created by Filebeat). This dashboard itself provides a summary of security events and metrics such as number of alerts, event types, source and destination IPs, etc. This allows you to quickly identify threats and trends in your network traffic.

3.5.6 Finalizing Integration with Filebeat

- In Kibana's Discover app, use the search bar to add a filter:

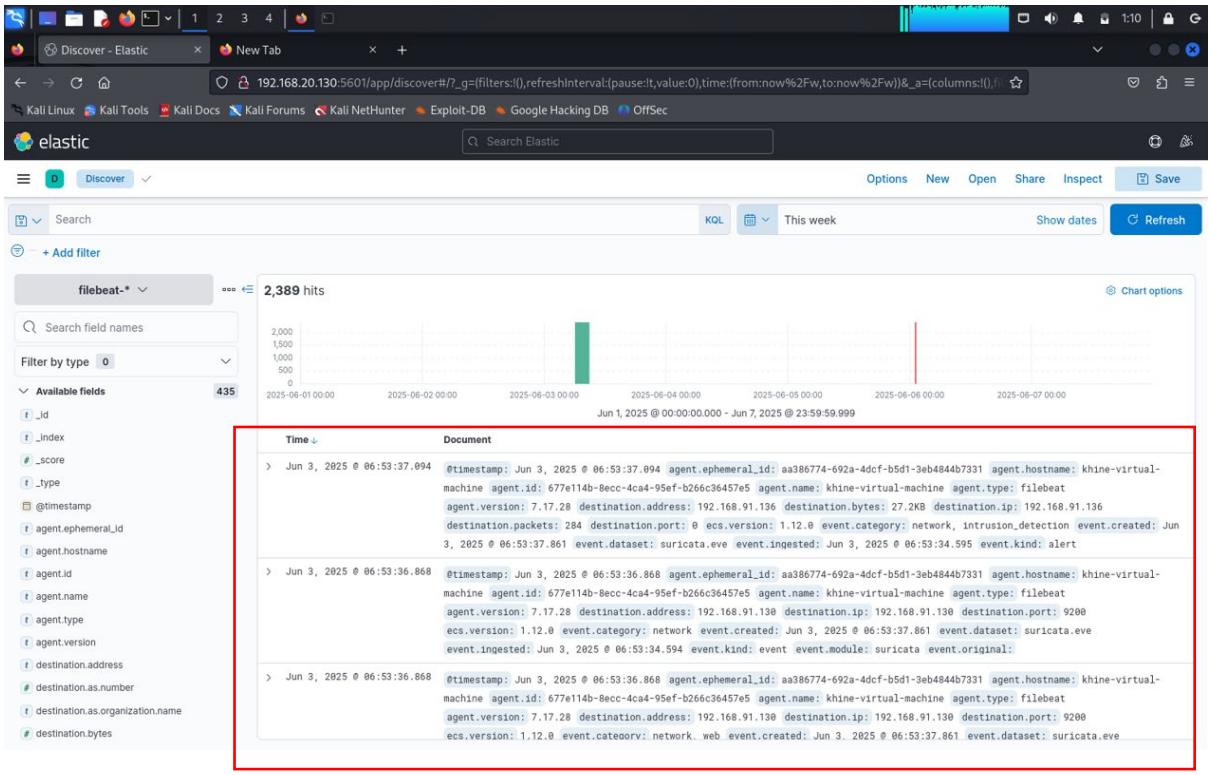


This index pattern is specifically for the suricata logs from the filebeat shipper for use with Kibana. Filtering by filebeat-* ensures we are only seeing the logs that

were collected from Suricata's eve.json file using Filebeat. This helps us to narrow down by time, event type, or network segments for investigation or threat hunting.

3.5.7 Verifying Kibana Dashboard

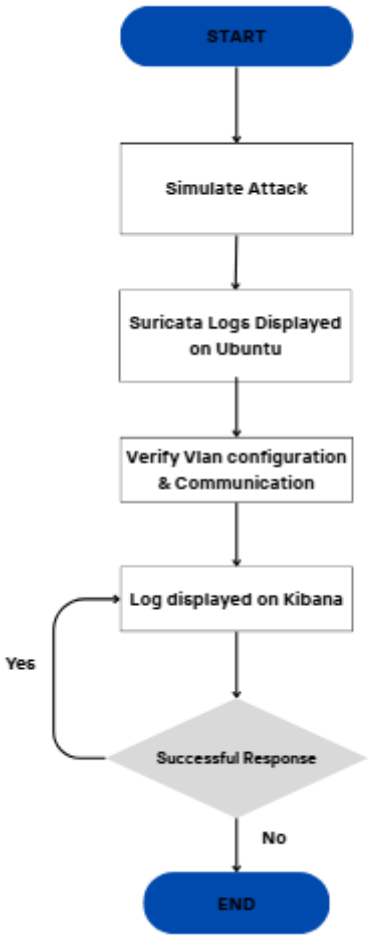
- The Kibana dashboard was accessed via a web browser using the assigned IP and port 5601:



The dashboard successfully loaded, showing real-time events and network activity, confirming that packet capture and monitoring were functioning as expected.

Chapter 4

Evaluation Testing of System Performance



Flowchart of System Testing

In the ICMP attack simulation, the simulation starts with pinging across VLANs to evaluate network communication and security visibility. Suricata can see the mirrored traffic and identify any ICMP packets and will log any of them it can discern. Once Suricata finish seeing the traffic, the logs of the detected ICMP packets will be sent to the ELK stack using Filebeat. In the ELK stack there is a Kibana dashboard that will display what Suricata saw. During this process there will be several other conditions to check - this includes VLAN set up, local Suricata logs in Kibana check, pinging vlan from the router to each VPC to see if the pings are physically working. If all conditions were successful, it would entail that VLANs, Suricata, and the ELK

pipeline are all configured and functioning normally. If any of the conditions failed, it would mean there is some sort of setup or visibility problem, and the simulation ends.

4.1 VLAN Traffic Mirroring to Suricata

- When the testing was conducted, the network interfaces were configured for promiscuous mode by running sudo ip link set vnet0_10 promisc on and sudo ip link set vun10_20_0 promisc on. This allows the interfaces to capture all packets on the network, not only packets that are directed to them.
- Afterwards, ingress traffic control was applied on vnet0_10 using the following command: sudo tc qdisc add dev vnet0_10 ingress, followed by the installation of the traffic control filter with the command: sudo tc filter add dev vnet0_10 parent ffff: protocol all u32 match u8 0 0 action mirred egress mirror dev vun10_20_0.

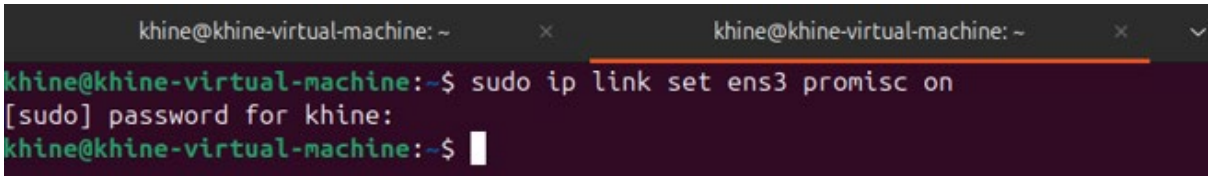
```
Last login: Sun Jun 15 13:37:44 2025 from 192.168.91.1
root@SPOTO-EVE:~# sudo ip link set vnet0_10 promisc on
root@SPOTO-EVE:~# sudo ip link set vunl0_20_0 promisc on
root@SPOTO-EVE:~# sudo tc qdisc add dev vnet0_10 ingress
root@SPOTO-EVE:~# sudo tc filter add dev vnet0_10 parent ffff: \
>     protocol all u32 match u8 0 0 \
>     action mirred egress mirror dev vunl0_20_0
root@SPOTO-EVE:~#
```

This configuration mirrors all incoming packets on vnet0_10 to the interface vun10_20_0, enabling effective monitoring and analysis of network traffic for the test environment.

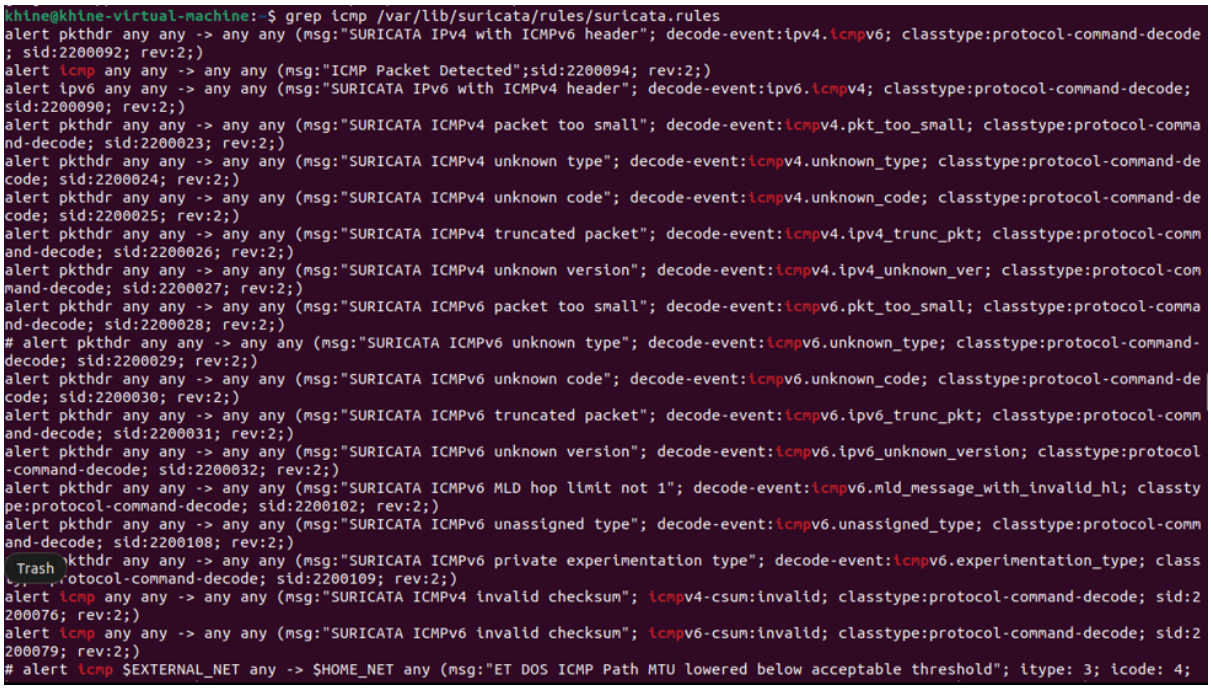
On the Suricata Ubuntu VM, the command sudo modprobe 8021q was used to enabled the module 8021q. This module is necessary to allow the system to identify and decode (802.1Q) VLAN tags in Ethernet frames. An IP address was assigned to the VM in the VLAN 10 subnet and the interface was brought up.

Next, promiscuous mode was enabled on the ens3 interface by running sudo ip link set ens3 promisc on.

- This setting allows the interface to capture all packets on the network segment, not just those addressed to it, which is essential for comprehensive traffic monitoring and intrusion detection.



- To further verify ICMP rules were specifically present and active, the grep utility was used:



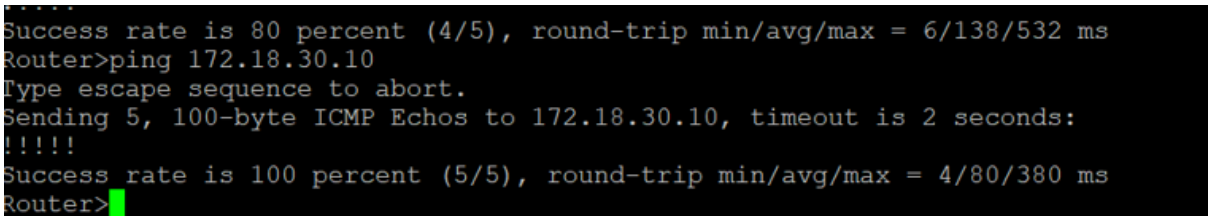
This command filters and lists all lines that include "ICMP", helping ensure that detection signatures for ping-related communications were indeed part of the active rule set.

4.2 Attack Simulation and Traffic Generation

i. ICMP Echo Request Test (Ping Attack Simulation)

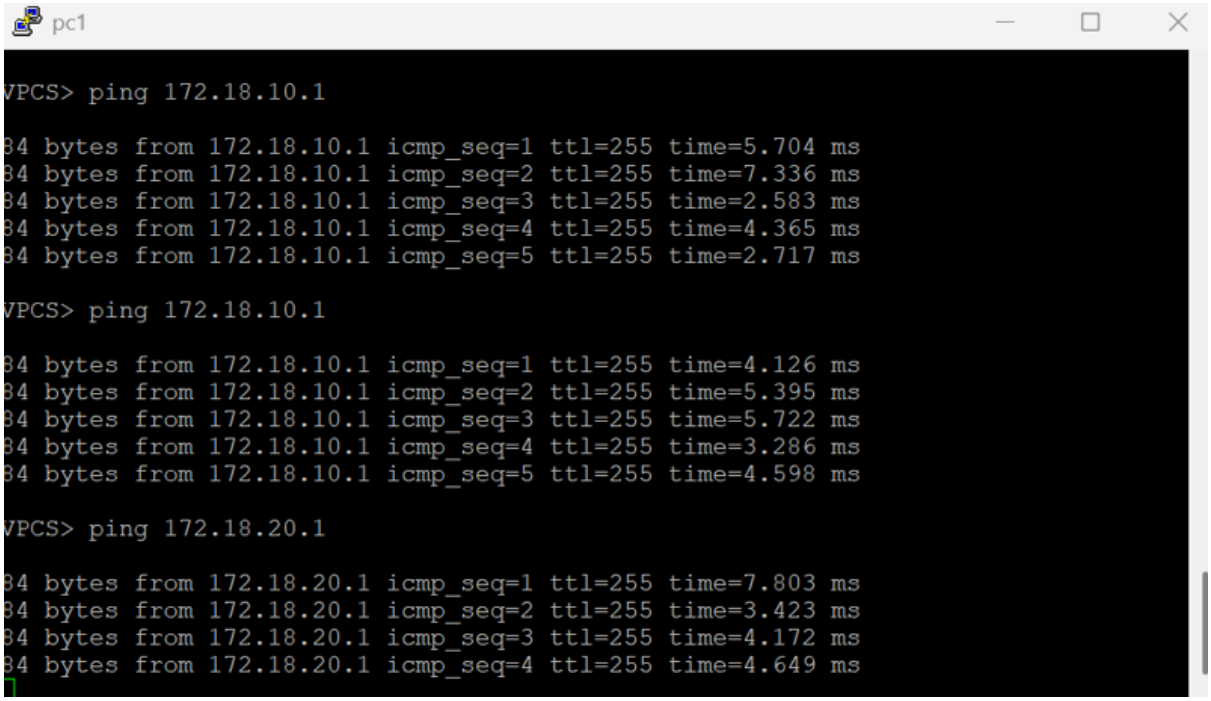
In order to verify that VLAN configurations were accurate and ensure complete bidirectional communication across the campus network, an ICMP Echo Request test (ping-based attack simulation) was utilized. Each test verified specific aspects of VLAN routing, interface configurations, and end-device connectivity.

- The process began with successful pings from the router to VPC1 (172.18.10.10), confirming that VLAN 10 was correctly assigned and that the switch trunk properly forwarded tagged traffic to the router's subinterface Gi0/0.10.



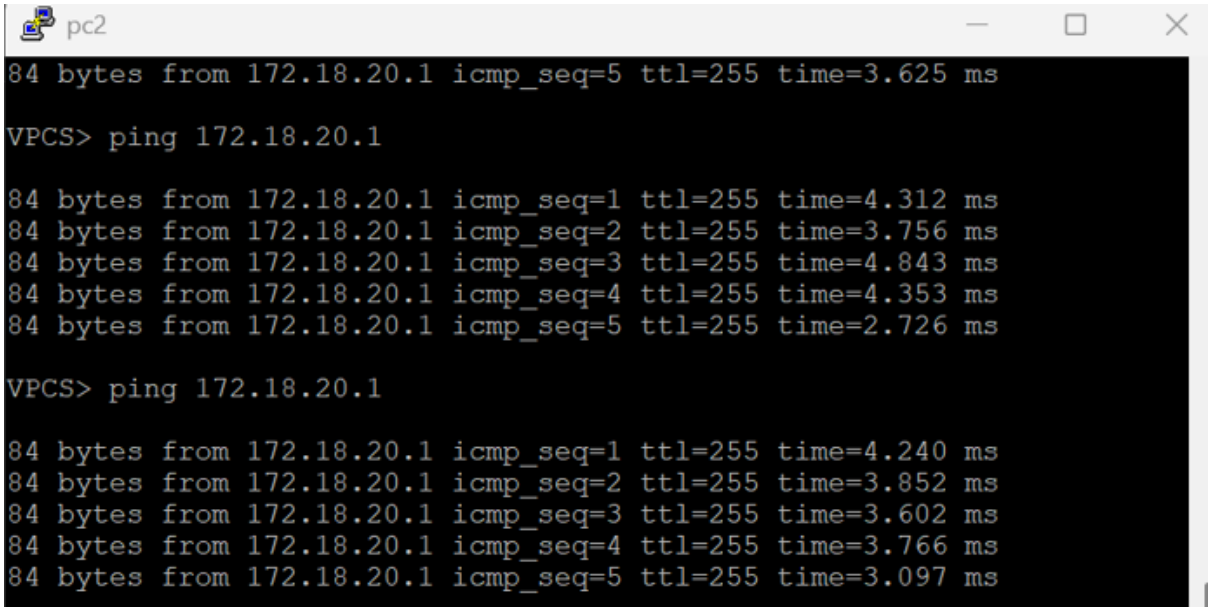
Further tests included pings to and from VPC2 (172.18.20.10) and VPC3 (172.18.30.10) verifying operational configurations on subinterfaces Gi0/0.20 and Gi0/0.30 respectively. All ping responses confirmed that there was legitimate two-way communication between the router and the hosts which verified proper VLAN tagging, trunking, and Layer 3 interface configuration across all devices.

Execute the command: ping 172.18.10.1 -



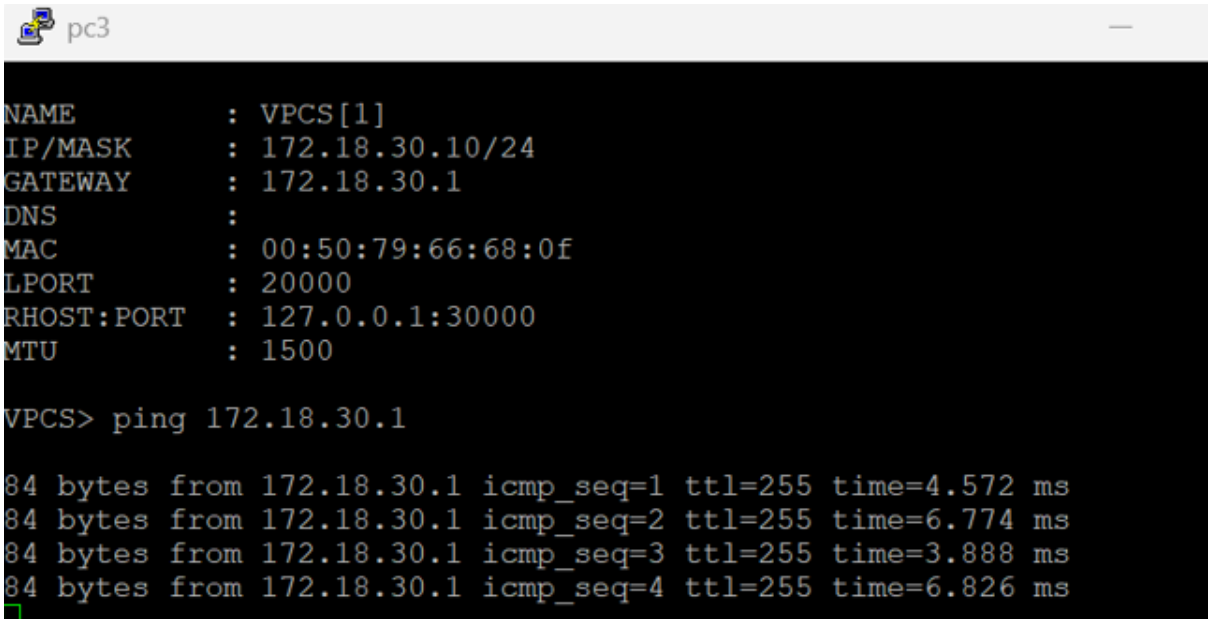
This successful ICMP echo reply confirmed that VLAN 10 was properly assigned to VPC1, the switch trunk was forwarding tagged VLAN 10 traffic to the router, and the subinterface g0/0.10 was up and active on the router.

Ping 172.18.20.1 -



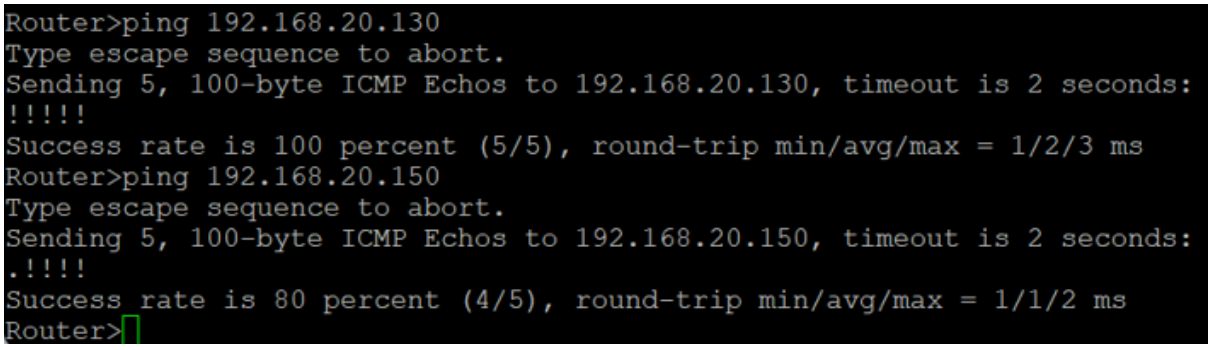
The successful response confirmed that VLAN 20 was operational. VPC2 had the correct IP and subnet mask for its VLAN, and the router's subinterface Gi0/0.20 was configured and reachable.

Issue the command: ping 172.18.30.1 -



The successful response provided verification that VLAN 20 was functional. VPC2 had the proper IP and subnet mask for its VLAN and the router had the sub interface Gi0/0.20 configured and reachable.

- Then test reachability from router R1 to Kali VM :

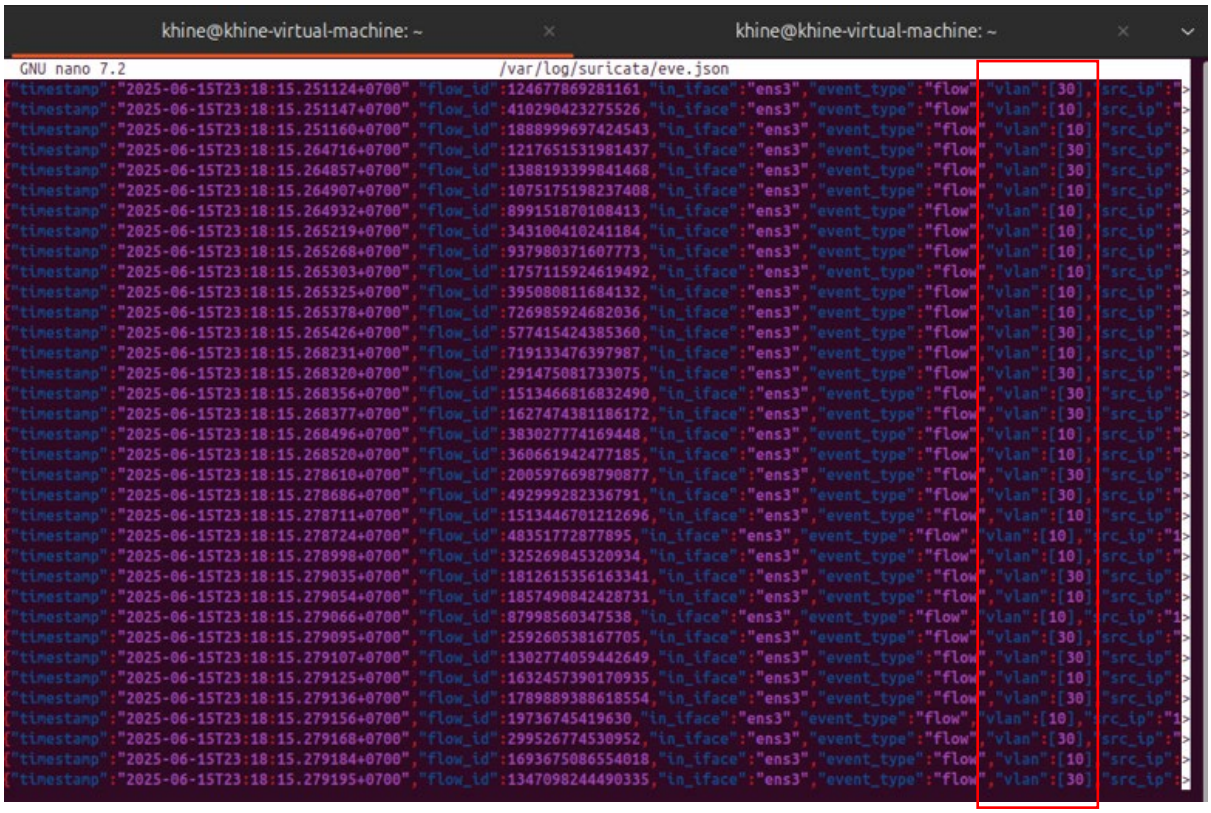


- From Kali VM , ping 172.18.10.1 :

```
┌──(kali㉿kali)-[~]
└─$ ping 172.18.10.1
PING 172.18.10.1 (172.18.10.1) 56(84) bytes of data.
64 bytes from 172.18.10.1: icmp_seq=1 ttl=255 time=19.0 ms
64 bytes from 172.18.10.1: icmp_seq=2 ttl=255 time=9.06 ms
64 bytes from 172.18.10.1: icmp_seq=3 ttl=255 time=10.1 ms
64 bytes from 172.18.10.1: icmp_seq=4 ttl=255 time=12.6 ms
64 bytes from 172.18.10.1: icmp_seq=5 ttl=255 time=6.15 ms
```

These ICMP tests are confirming bidirectional reachability across the segmented network. A successful ping indicates the VLAN is configured correctly, trunking is occurring properly, the interfaces are communicating and that Suricata is placed correctly in the traffic path and receiving packets to inspect.

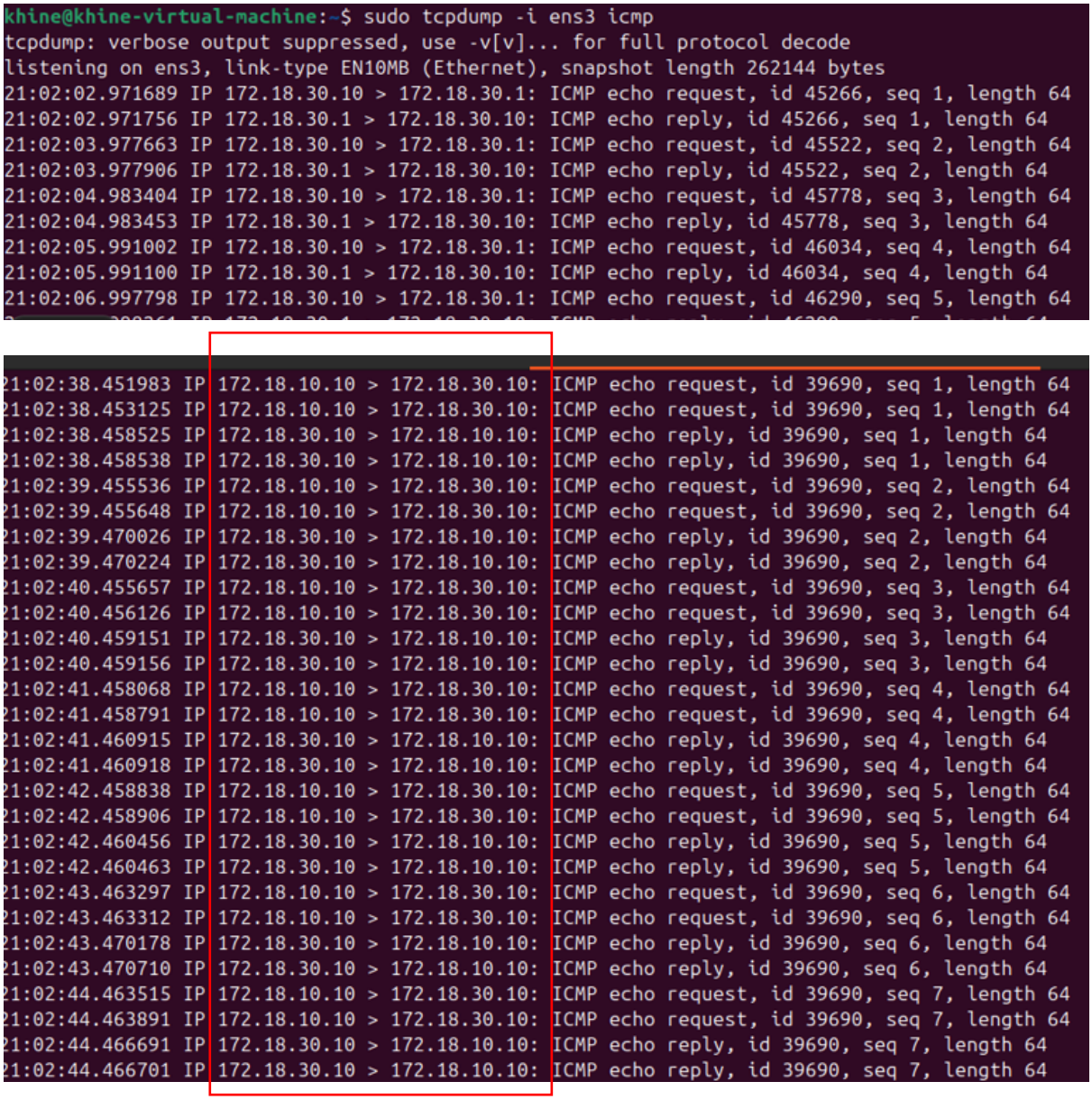
These ICMP tests also served as functional Layer 3 tests validating IP addressing, subnet mask, and router reachability on each VLAN. Each successful ping shows that Suricata, in the mirrored traffic path, is actively receiving ICMP traffic which can be checked with sudo nano /var/log/suricata/eve.json.



To validate Suricata's traffic visibility and inspection capabilities across VLANs, ICMP echo requests (pings) were initiated from one VPC to another situated in

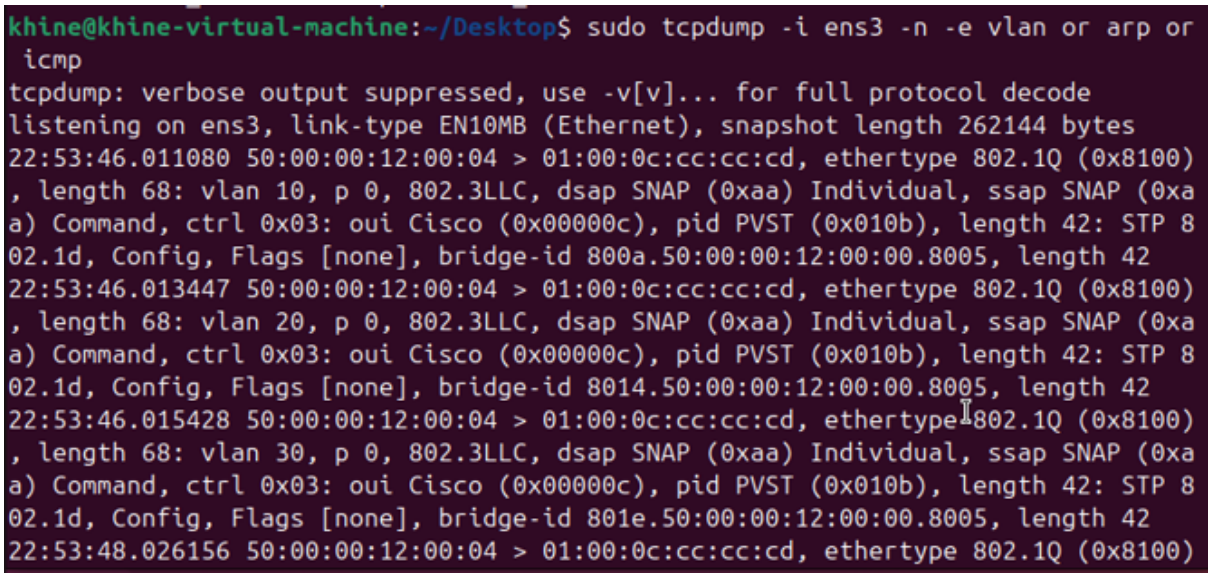
different VLANs. These pings simulate benign network communication, but in the context of intrusion detection, they also serve as reconnaissance attempts or precursors to ICMP flood attacks.

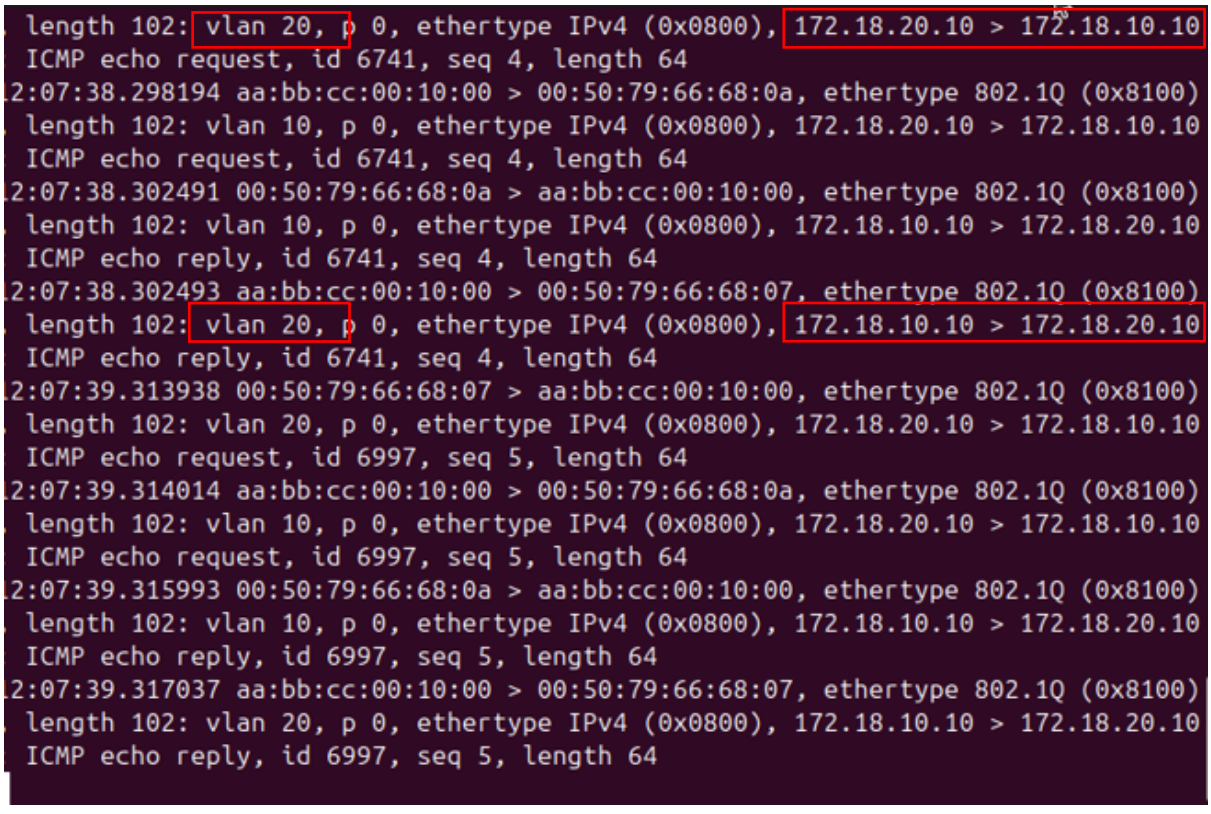
- The first stage involved executing the command sudo tcpdump -i ens3 icmp :



The output has clearly shown incoming ICMP packets containing both types of requests and replies from and to the Campus VLAN subnets. For instance, the packet traces showed 172.18.20.10 > 172.18.20.1: ICMP echo request and 172.18.20.1 > 172.18.20.10: ICMP echo reply and also the same for as seen above 172.18.10.10 > 172.18.10.1 and 172.18.30.10 > 172.18.30.1. Evidently, the port mirroring configuration was operational, and Suricata was seeing the correct mirrored traffic.

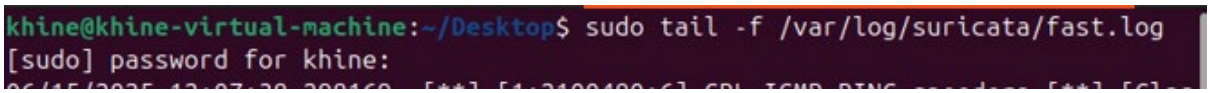
- To narrow the visibility to VLAN-specific traffic and analyse tagged Ethernet frames, the following enhanced tcpdump command was used:

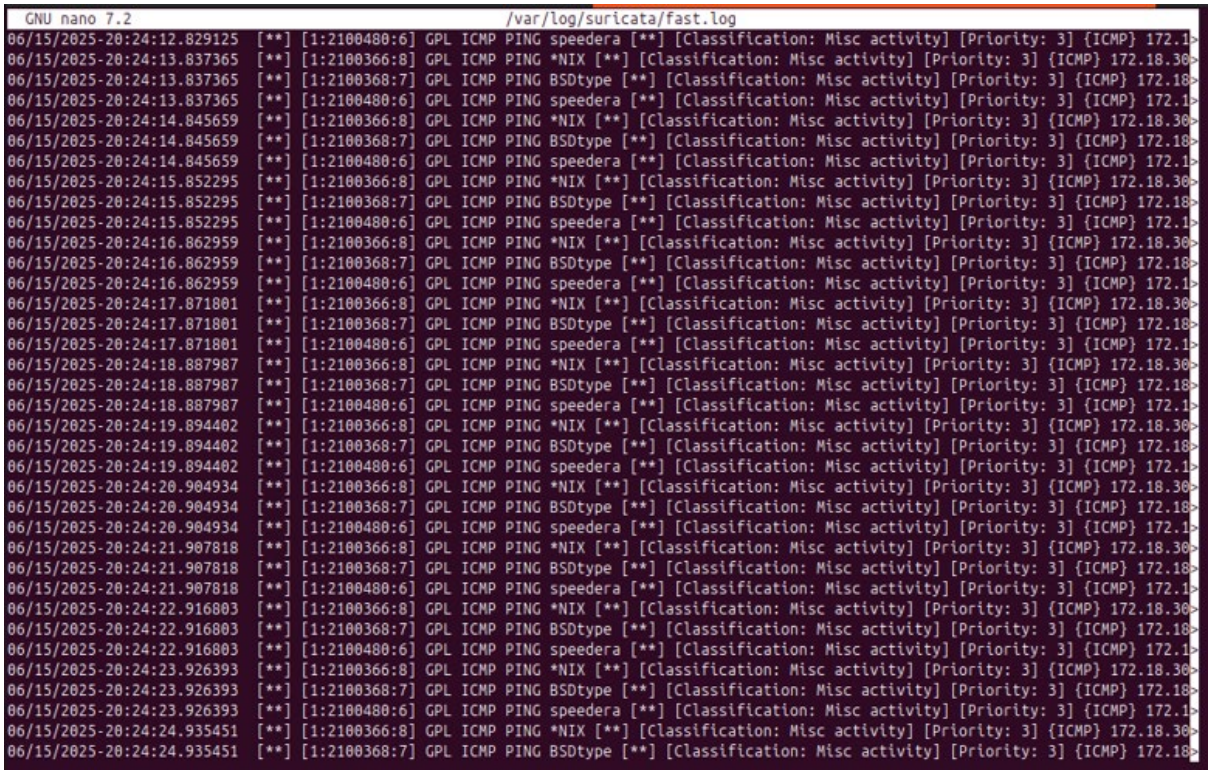




This command displays Ethernet headers (-e), disables DNS resolution (-n), and filters for VLAN-tagged packets. It successfully identified packets tagged with VLAN IDs such as 10, 20, and 30, verifying correct SPAN/mirror port setup and Suricata's ability to analyze data-link-layer tagged traffic.

- After configuring the SPAN/mirroring setup and verifying traffic mirroring with tcpdump, we proceeded to test whether Suricata IDS could accurately inspect, identify, and generate alerts based on ICMP traffic across VLAN segments in our virtual campus network.
- Logs from /var/log/suricata/fast.log showed alerts generated by both custom and GPL rulesets, with specific source and destination IPs confirming proper VLAN-based traffic detection.





After this confirmation, Suricata's capability for real-time detection could be observed with the logging. Running the command sudo tail -f /var/log/suricata/fast.log, the alert logs confirmed that Suricata's rule engine identified the ICMP packets as defined in both custom rules and GPL community rules.

The logs had examples like:

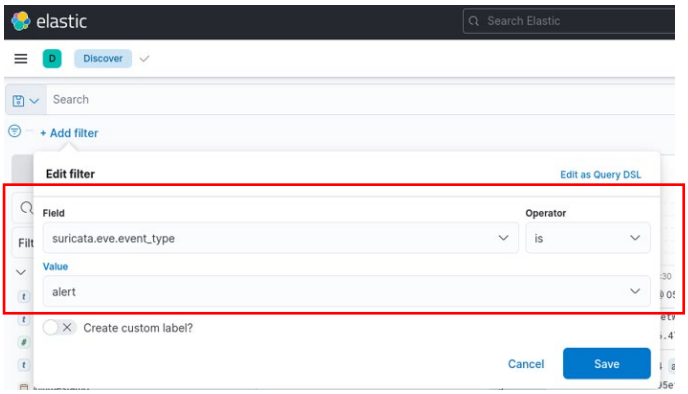
06/15/2025-20:24:13.837365 [**] [1:1000001:1] ICMP Packet detected [**] {ICMP} 172.18.20.1 -> 172.18.20.10

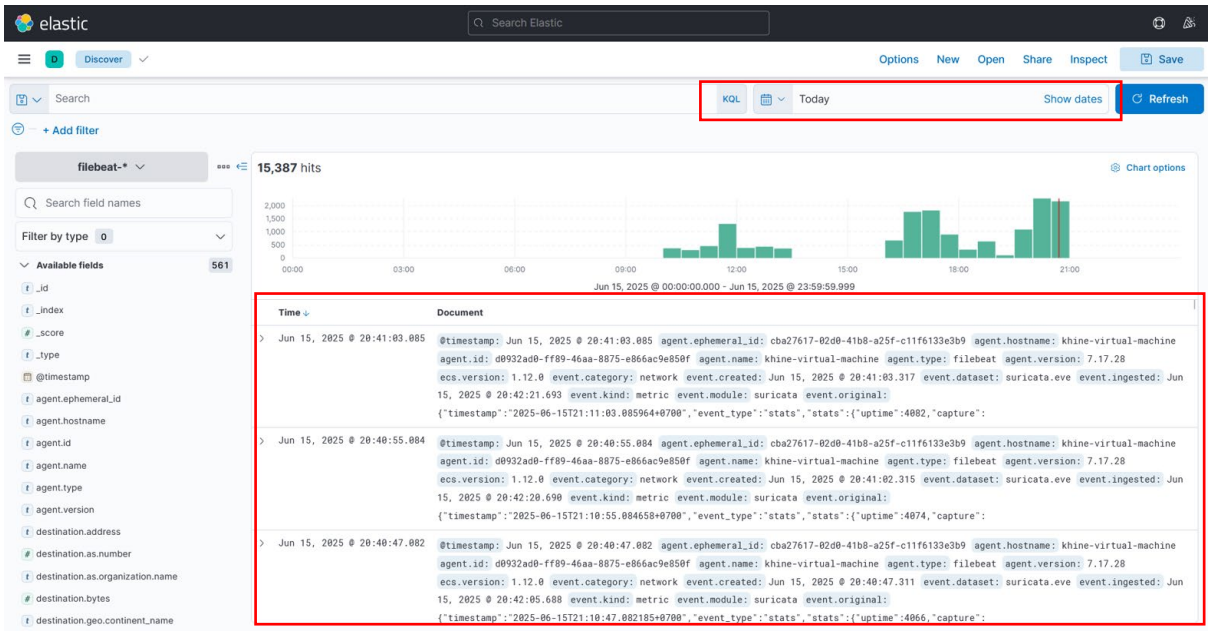
*06/15/2025-20:24:14.845659 [**] [1:2100366:8] GPL ICMP PING *NIX [**] {ICMP} 172.18.10.1 -> 172.18.10.10*

06/15/2025-20:24:15.854926 [**] [1:2100368:7] GPL ICMP PING BSDtype [**] {ICMP} 172.18.30.10 -> 172.18.30.1

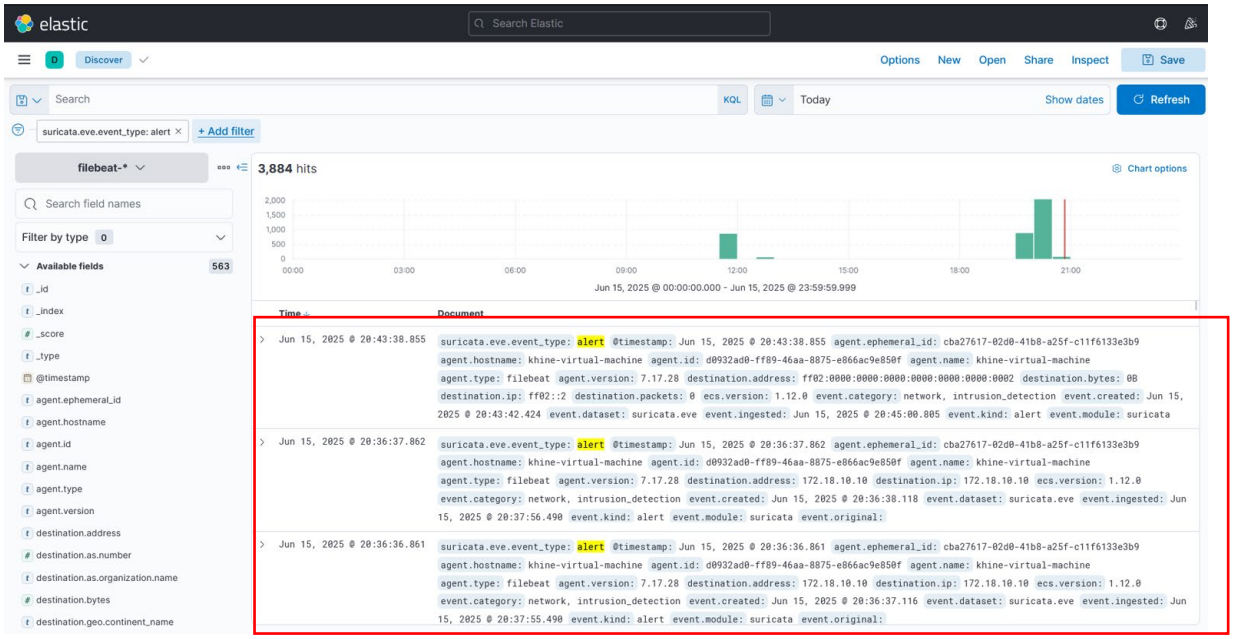
These logs showed that Suricata was accurately classifying ICMP echo requests based on source IP, destination IP, and rule identifiers. The classification fields in the logs went further in illustrating its detected traffic, characterizing them as generic ICMP traffic, or as a specific "*NIX" or "BSDtype" pings, which provided clues with respect to the operating system. The ability to examine this in the packet data is important in determining reconnaissance activity or lateral movement attempts.

- Filters applied in Kibana began with:





In Kibana , log entries ingested via Filebeat from Suricata EVE JSON logs, including metadata like timestamps, agent information, and Suricata event fields filtered by the filebeat-* index pattern.



When filter with suricata.eve.event_type: alert, only Suricata-generated alert logs are displayed , which indicate potential security threats detected in the network traffic.

t agent.name	khine-virtual-machine
t agent.type	filebeat
t agent.version	7.17.28
t destination.address	172.18.10.10
destination.ip	172.18.10.10
t ecs.version	1.12.0
t event.category	network, intrusion_detection
event.created	Jun 15, 2025 @ 20:36:38.118
t event.dataset	suricata.eve

Figure 7 : Metadata from Filebeat agent on ‘khine-virtual-machine’ capturing Suricata intrusion detection event targeting destination IP 172.18.10.10

t service.type	suricata
t source.address	172.18.10.1
IP source.ip	172.18.10.1
t suricata.eve.alert.category	(empty)
# suricata.eve.alert.gid	1
# suricata.eve.alert.rev	1
t suricata.eve.alert.signature	ICMP Packet detected

Figure 8 : Suricata alert for ICMP packet detection originating from source IP 172.18.10.1

IP related.ip	172.18.30.10, 172.18.30.1
t rule.category	Misc activity
t rule.id	2100368
t rule.name	GPL ICMP PING BSDtype
t service.type	suricata
t source.address	172.18.30.10

Figure 9 : Alert triggered by Rule GPL ICMP PING BSD type (NIX variant)

t rule.name	GPL ICMP PING *NIX
t service.type	suricata
t source.address	172.18.30.10
# source.bytes	510B

Figure 10: Alert triggered by Rule GPL ICMP PING *NIX

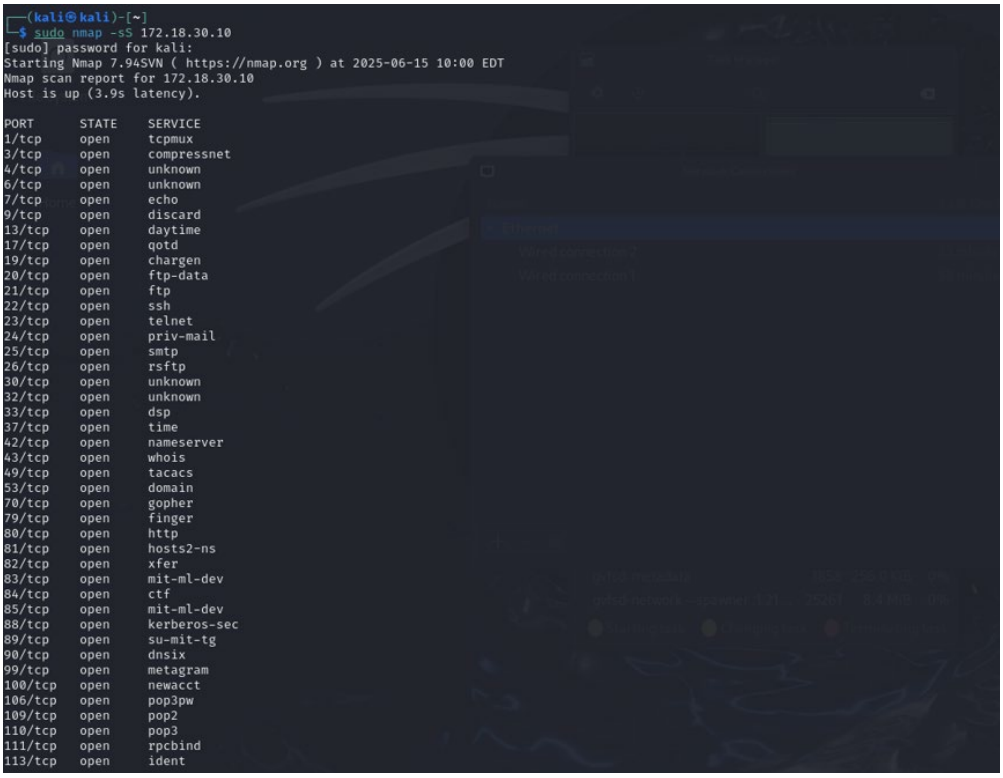
In addition, logs from the eve.json file, with the ELK Stack, recorded further structuring and enhancement of metadata on each alert and, in total, verified the same source-destination pairs for traffic from VLAN's 10, 20, and 30. For instance, the field 'event_type': alert, src_ip, dest_ip and signature, matched our textual alerts outlined in fast.log. The alert concerning 'ICMP Packet detected' recorded it with metadata traffic information from

172.18.10.10 to 172.18.10.1. Suricata identified this activity, demonstrating it could identify and document even a basic network utility like ping, which could be weaponized in more dynamic applications.

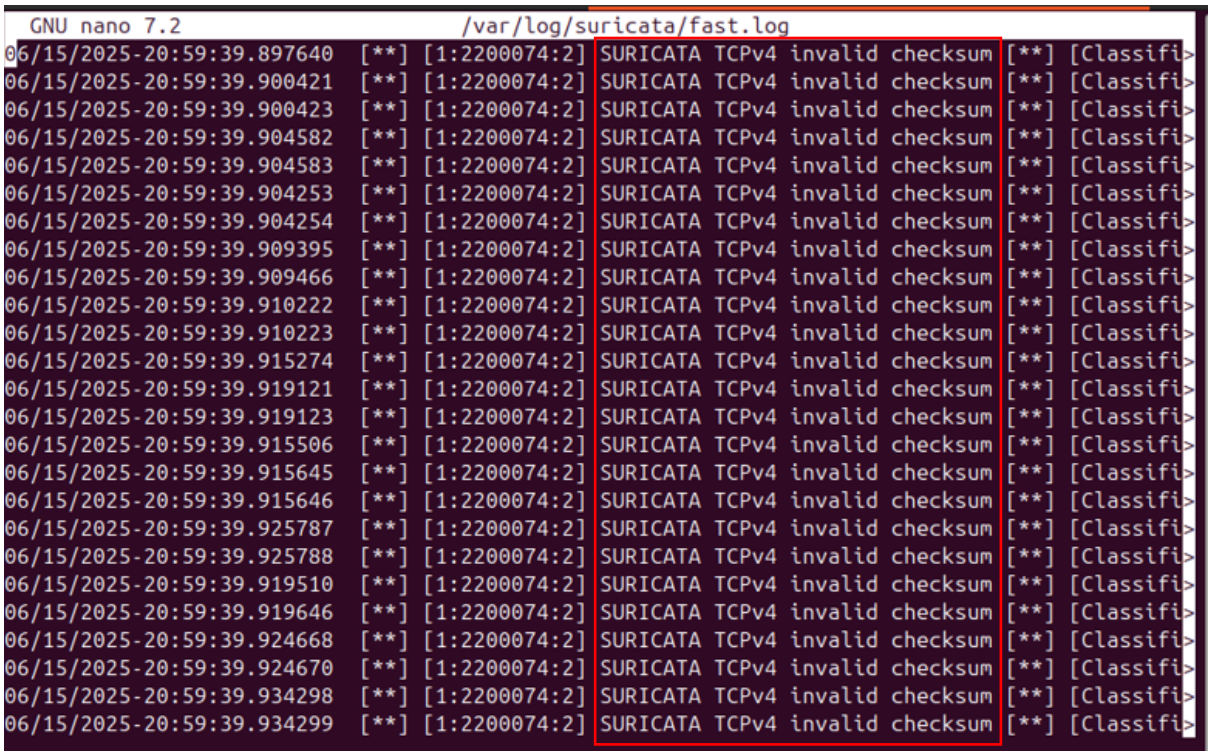
This entire process from generating VLAN-specific ICMP traffic, to seeing the mirrored packets with tcpdump, to validating Suricata detections in both text log and structured log files shows that the IDS system functionally works and is appropriately sited. The SPAN port passed VLAN-tagged traffic to Suricata. Suricata could evaluate the contents of each packet as it passed through its interface, apply its detection rules, and capture actionable alerts with exact source and destination information across all VLANs in the test network.

ii. TCP SYN Scan Simulation using Nmap

To represent active reconnaissance action within the campus network, a stealth TCP SYN scan to a management VLAN host with an IP of 172.18.30.10. The scan was done using Nmap's -sS option which sends a TCP SYN packet and does not complete the handshake process; this option is also used as a means of disguising traffic from firewalls and IDS / IPS. The entirety of this action was successfully captured and analyzed in real-time by the Suricata IDS running on the monitoring interface 'ens3'.



- During the scan, live alerts were observed on Suricata using “sudo tail -f /var/log/suricata/fast.log” command .



Several alerts were instantly triggered and displayed live on the terminal. Among them were signatures such as SURICATA TCPv4 invalid checksum , indicating that Suricata was actively flagging the scan attempts against common service ports. The scan targeted the Suricata Ubuntu VM to test the intrusion detection response for unauthorized port access.

- To further investigate and verify the detection pipeline, the Kibana dashboard (accessed via http://192.168.20.130:5601) was used.

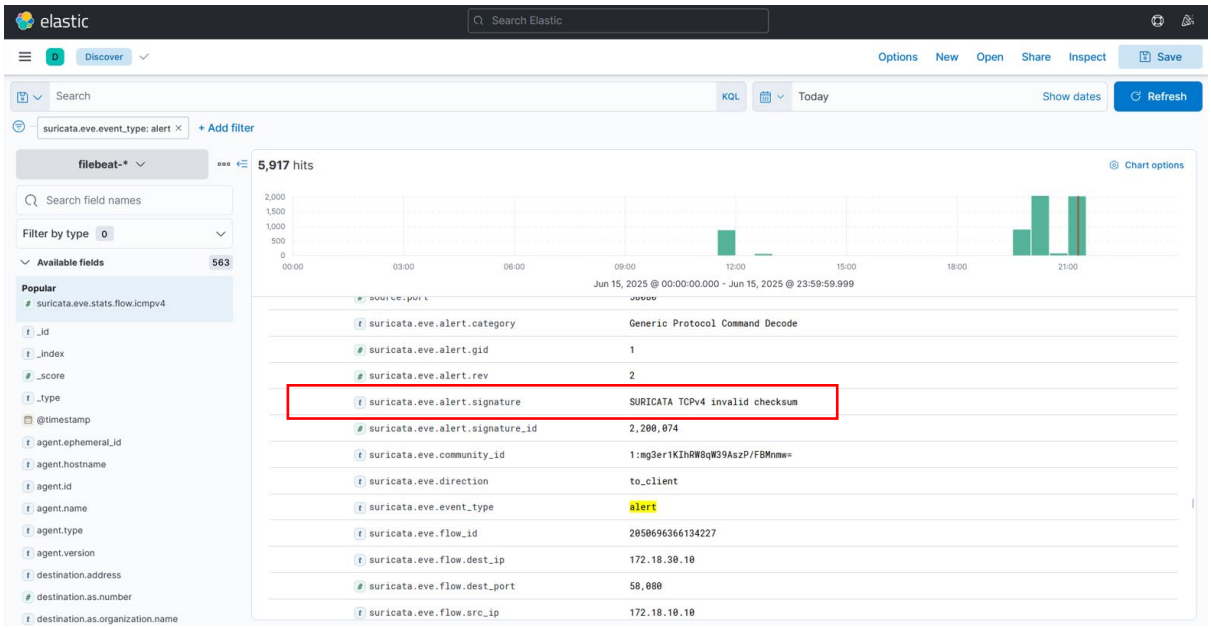


Figure 11 : Results Filtered by event_type:alert

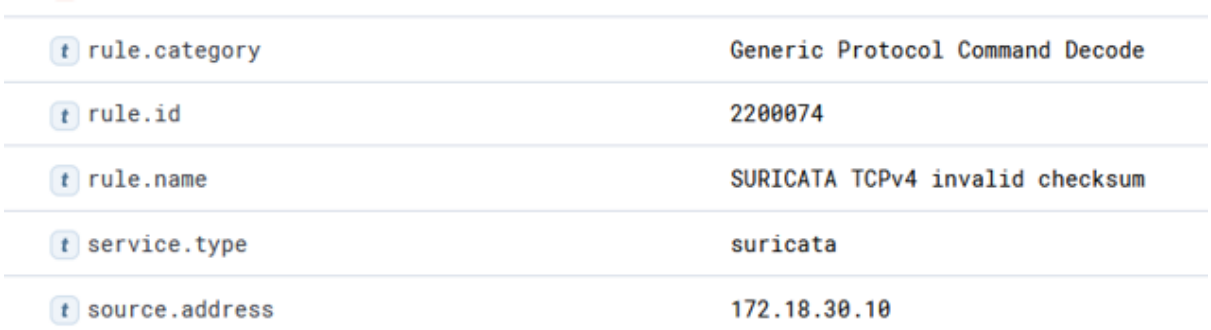


Figure 12 : Suricata alert triggered by Rule SURICATA TCPv4 invalid checksum

In the Kibana, Suricata noticed unusual abnormal packet behaviors NOT related to port scanning. The logs showed entries with the signature SURICATA TCPv4 invalid checksum. The alarm signature ID was repeated numerous times and occurred around the time stamp of 20:59:39. The alerts were classified under the Generic Protocol Command Decode. The instances would typically be packets coming from source IP address 172.18.10.10 directed toward destination IP address 172.18.30.10 using port 58080. The alerts were reporting on TCP checksums which, depending the calculation, were either too small or too large. The alerts indicated possible malformed packets or potential evasion tactic. Once again, these logs validated the effectiveness of Suricata. Because a TCP checksum of this nature can

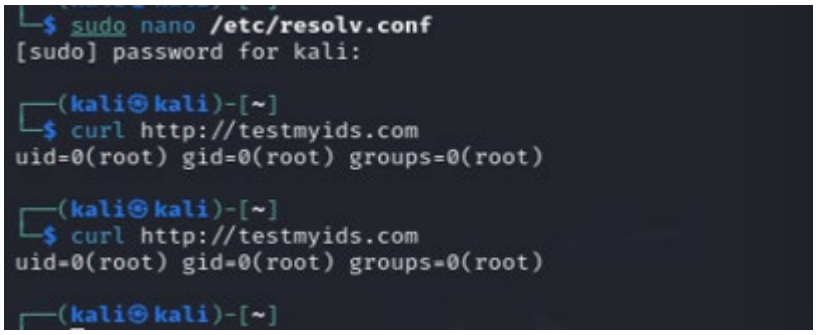
indicate corrupted packets or potential obfuscation or potentially packet crafting - typical indicators used in network probing and exploit attempts.

With this test, it was validated that Suricata could detect multiple known scanning techniques and low-level instances of abnormalities. The ELK stack's integrations mean that these alerts could be logged and visualized, along with Suricata visualizations, to ensure that they could be detected in a proactive method, logged, and subject to forensic analysis. It is very strong finding that the intrusion detection system is working properly and alerting on initial reconnaissance stages in the emulated campus environment similar to a Nmap scan, malformed packets, and unusual protocol behavior, supported by Suricata. This provides reasonable validation in the value of the intrusion detection system to increase network situational awareness and prepare countermeasures to avert superior tactics in the initial reconnaissance stage of advanced cyber attacks.

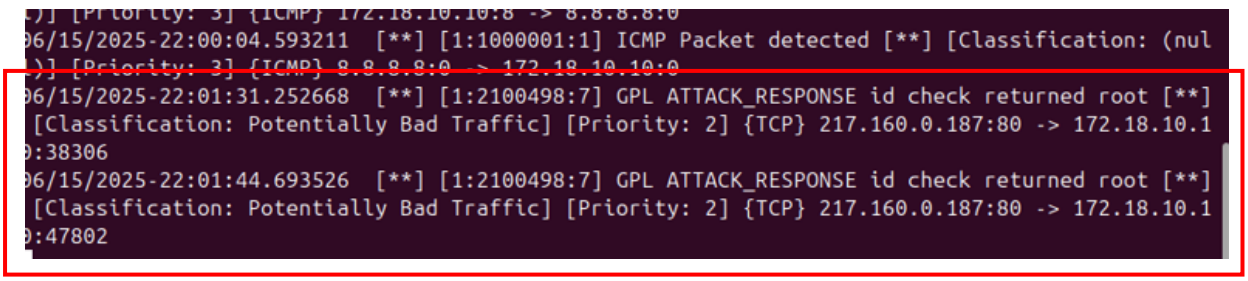
iii. Malicious Domain Access via HTTP

To simulate an outbound HTTP access to a known suspicious domain, a simple curl request was initiated from the Kali attacker machine using the command :

- curl http://testmyids.com



This domain is intentionally designed to trigger intrusion detection systems by returning payloads that mimic malicious or sensitive command-line outputs.



As the request was executed, Suricata immediately flagged the HTTP response as suspicious. Within the fast.log, alerts were generated with the signature GPL ATTACK_RESPONSE id check returned root, having Signature ID 2100498 and classified under the rule category Potentially Bad Traffic . The alert indicated that the external host 217.160.0.187:80 responded to the internal target host 172.18.10.1 (or 172.18.10.10) using TCP port 80 (HTTP).

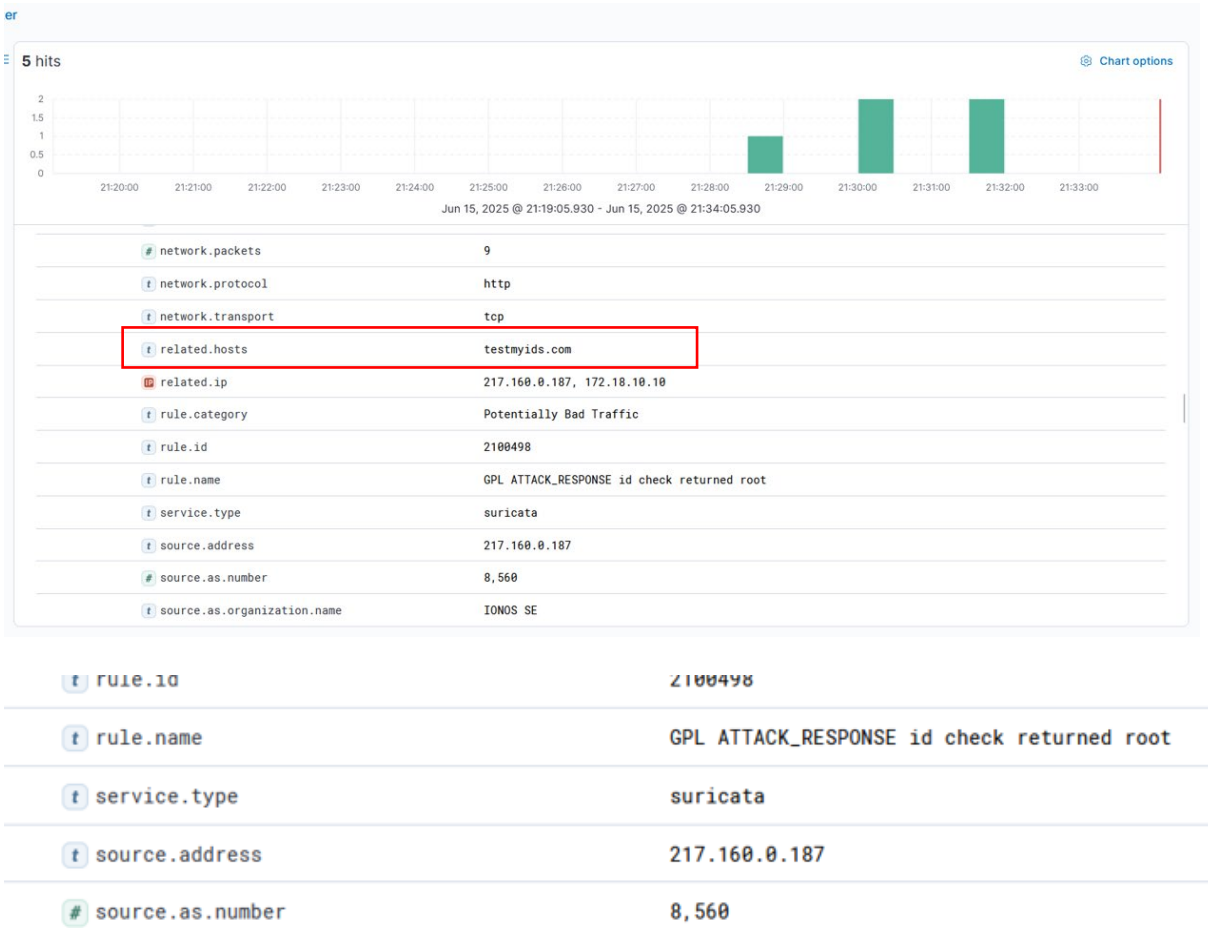


Figure 13 : Alert for Root Privilege Disclosure (Rule: GPL ATTACK_RESPONSE)

Upon further inspection using Kibana, the analyst navigated to the filebeat-* index via the Discover tab and confirmed the presence of this event with detailed metadata. The relevant fields showed that the transport protocol was TCP and the application protocol was HTTP. The related.hosts field identified the destination domain as testmyids.com, while the related.ip field listed both the external source IP 217.160.0.187 and the internal target IP 172.18.10.10. The rule name and ID were consistent with the Suricata fast log output, confirming that the detection pipeline was functioning end-to-end—from packet capture by Suricata, alert generation, log

forwarding via Filebeat, and final visualization in Kibana. The Autonomous System (AS) information further confirmed that the source IP belonged to AS number 8560, owned by the organization "IOMOS SE", matching known threat simulation infrastructure.

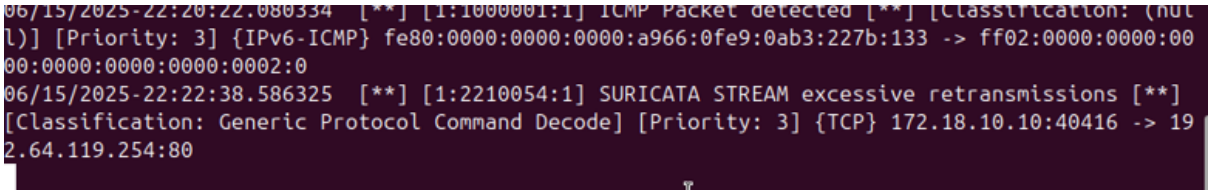
This test validates that Suricata is capable of detecting outbound connections to potentially malicious domains and identifying suspicious HTTP response content even from seemingly benign actions like a curl request. The alert classification and priority suggest that while this may not be a full-fledged exploit, it is a strong indicator of compromise or post-exploitation behavior such as data leakage or reconnaissance, reinforcing the importance of outbound traffic monitoring within the IDS deployment.

iv. Malicious HTTP Access and Stream Anomalies

- To evaluate the system's ability to detect access to potentially harmful domains, a curl command was executed from the internal Kali attacker VM targeting the URL http://malicious.test.malwaredomainlist.com.

This domain is part of a public repository of known malicious URLs often used for IDS testing.

- As the request was sent from the attacker machine with IP 172.18.10.10 to the suspicious external server at 192.64.119.254 over TCP port 80, Suricata generated an alert indicating an anomaly in the traffic stream.



The alert was logged in the fast.log with the signature "SURICATA STREAM excessive retransmissions" (Signature ID 2210054), categorized under "Generic Protocol Command Decode" with a priority of 3. This signature typically indicates that the network session experienced repeated retransmissions, possibly due to dropped

packets, session corruption, or deliberate obfuscation attempts by malicious actors trying to evade detection or degrade IDS effectiveness.

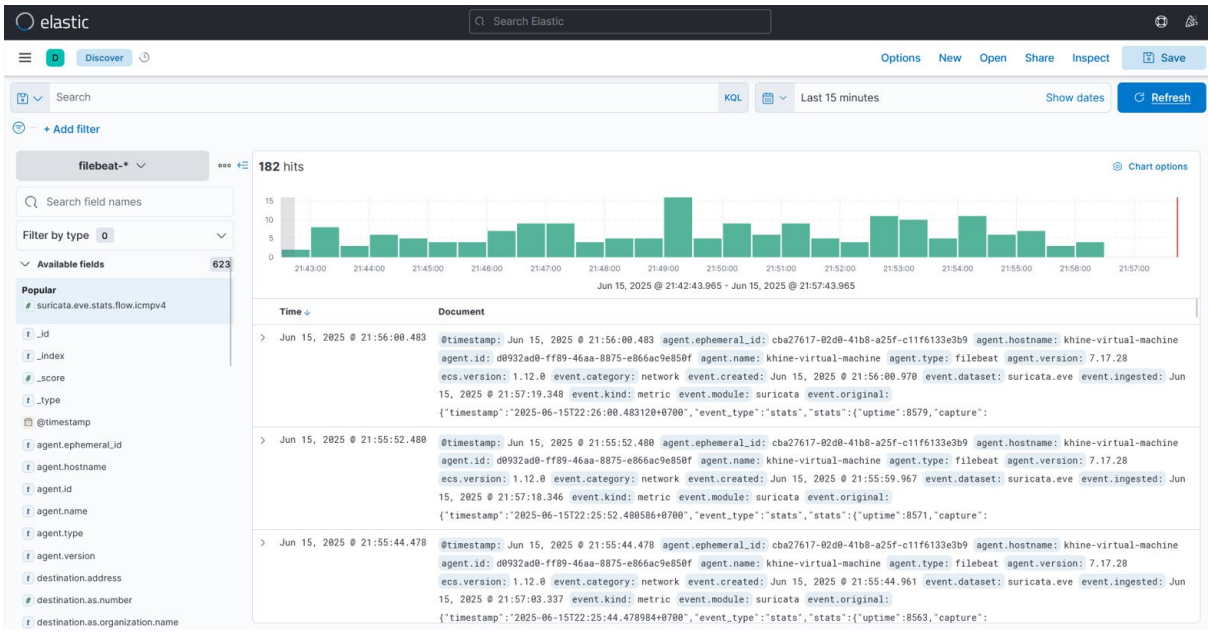


Figure 14 : Kibana dashboard displaying real-time Suricata event logs and traffic statistics captured via Filebeat for network security monitoring

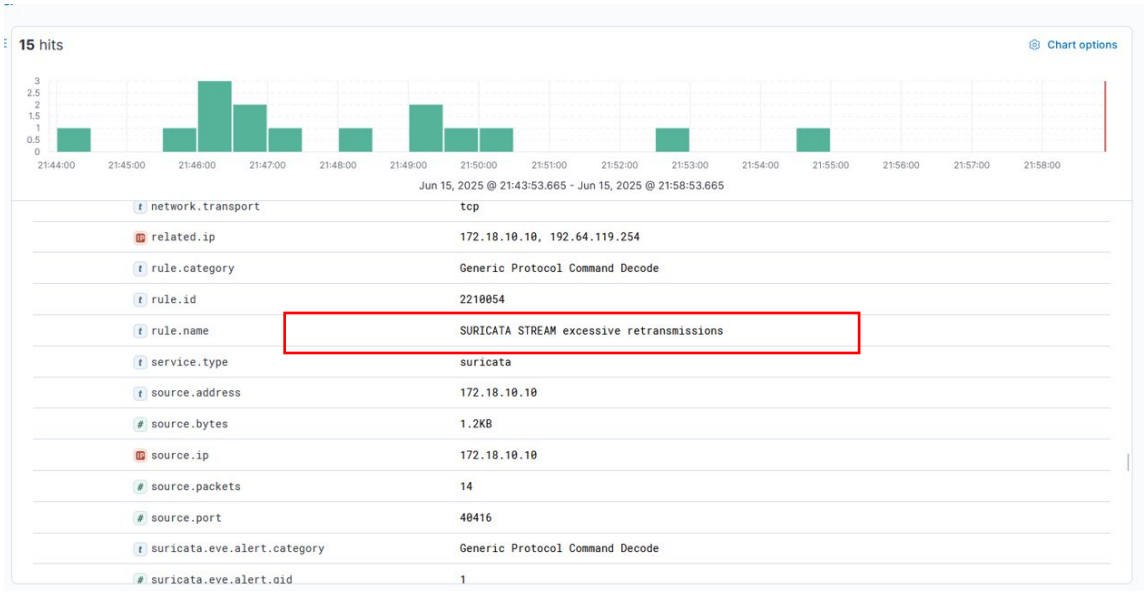


Figure 15 : Suricata alert indicating excessive TCP retransmissions classified under 'Generic Protocol Command Decode'

Within Kibana, viewing the filebeat-* index returned log details matching the same detection. The related.ip field confirmed the internal and external IPs with

corresponding address of 172.18.10.10 (Kali attacker) and 192.64.119.254 (malicious server). The Suricata rule metadata indicated the alert was a Generic Protocol Command Decode, with service.type as suricata, and observed 14 packets of 1.2KB total from the source address. The alert was also categorized as lower priority, however, it is important to note that high levels of retransmissions could imply evasion techniques or some type of misconfigured/malicious intent using session based protocols.

This scenario illustrates that Suricata is able to observe and detect either high confidence attack signatures, to relatively low level stream anomalies that may represent use of covert communications methodology, or possible poor implementations of mutual cooperation implementation of a malicious payload. These logs and detections are an important area of discoverable findings, to identify stealthy attack attempts, or attacks that are not fully communicating or complete, in the monitored network.

4.3 Dashboard Visualization

When Suricata and the ELK Stack were fully integrated, Kibana offered a live dashboard for viewing security alerts as well as traffic trends in real time. The top panel of the dashboard had a "Metrics" visualization that showed a high-level overview of all events generated by Suricata, captured by Filebeat, on the current day. This live view gave analysts the ability to assess live threats and anomalies occurring on the virtual campus network.

One of the interesting visualizations included a dual-panel bar chart that showed the number and type of Suricata events over time along with the hosts that generated the most events. This visualization allowed the administrator to immediately note which systems were currently under active threat or witnessing suspicious activity.

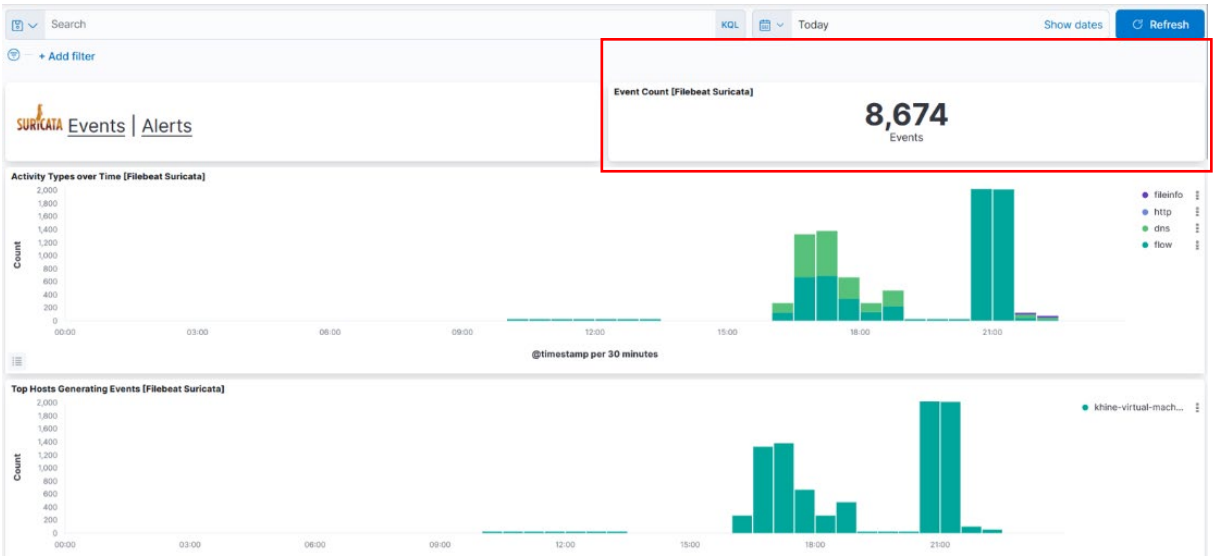


Figure 16 : Network Activity Type Over Time and Top Event-Generating Hosts

Yet another helpful panel provided distribution of events over transport protocols like UDP and TCP which then allowed analysts to quickly identify anomalous, or attack, activity specific to each noted protocol. This concept could be expanded to include a geolocation-based view of the top destination countries based on detected communications. The geolocation view would be especially useful for informing the analyst about suspicious traffic or connection patterns that involved more than a couple of hops, or any international overtures from this network, or any resources overly burdened due to systematic traffic. The last panel covered host-level statistics providing a summary listing of metrics such as total number of alerts produced or associated with each system or IP; event types associated with each host; and communication patterns. This bottom-level view of activity was especially important for providing an analytical snapshot of the most compromised, attacked, or targeted systems in an environment.

Figure 17 : Event Types by Protocol and Top Destination Countries

Additionally, the dashboard included two distinct sections summarizing Suricata signature alerts: one for top-used signatures and another for alert classifications by severity and category. These provided further granularity in understanding which specific detection rules were being triggered most frequently and the nature of the threat landscape within the emulated campus environment.

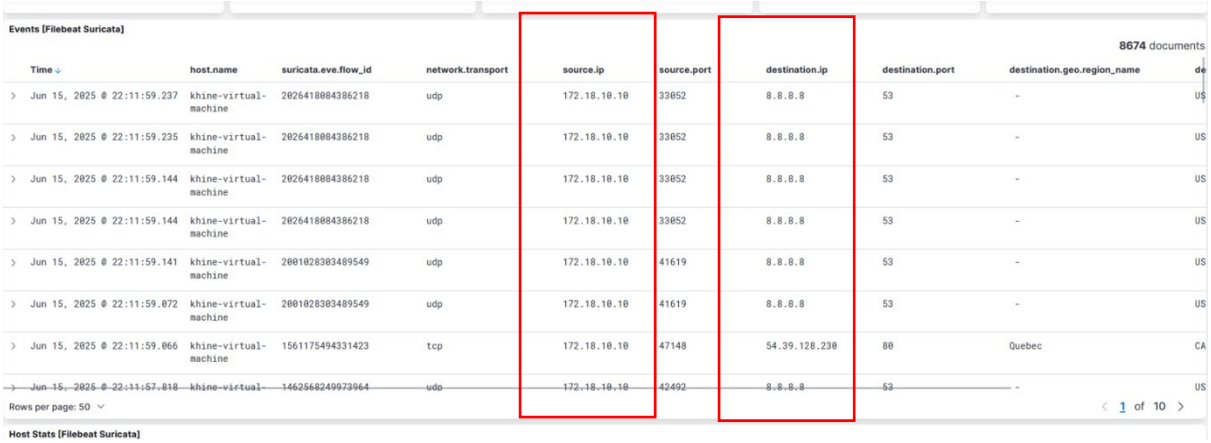


Figure 18 : Alert Signature Status Overview 1

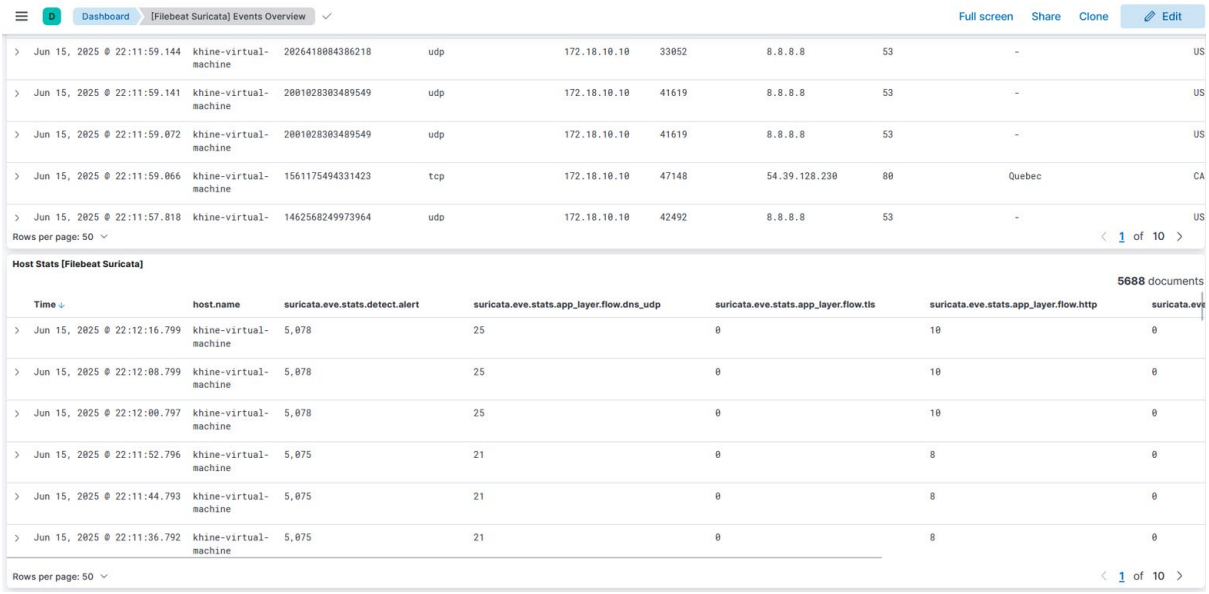


Figure 19 : Alert Signature Status Overview 2

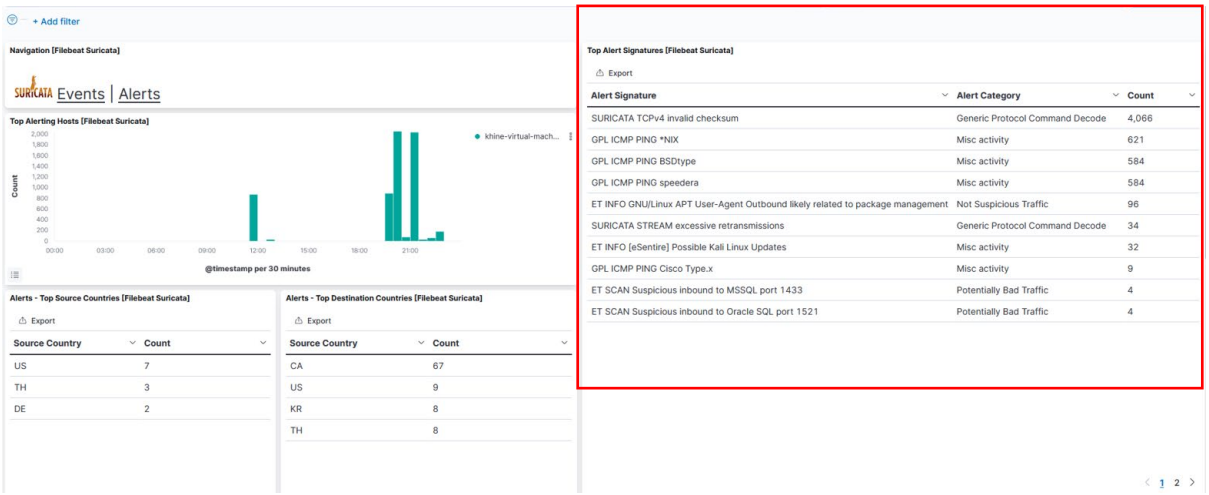


Figure 20 : Visualization of Top Alert Signatures

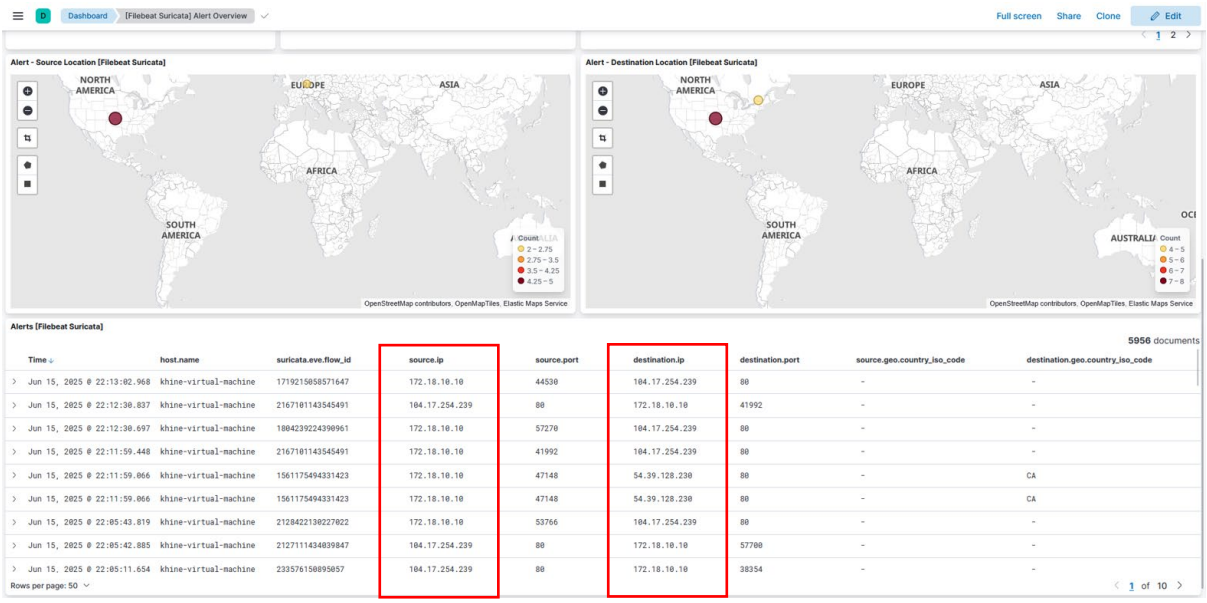


Figure 21 : Geographic Hotspots of Suricata Alerts

Overall Process Explanation

The entire project proceeded in a logical way, beginning with the design and implementation of a simulated campus network within EVE-NG. The topology was designed to emulate a tangible university environment, including logically separated VLANs for the administration, human resources, management, guest, students, and teacher zones. The separate VLAN's were implemented using subinterfaces created through a central router and using 802.1Q trunking and tagging on the trunk link to allow for inter-VLAN routing. The multilayer switches were implemented to designate core and access layers, while end-devices (VPCs) were located within their defined VLAN, to represent endpoints in a departmental simulation.

So, after the network was established, a Suricata IDS was installed on an Ubuntu virtual machine and involved in mirrored traffic of all the VLANs. To allow Suricata the ability to passive inspect actual traffic for intrusions, port mirroring was configured on the core switches to give Suricata passive visibility over the live traffic. Suricata was configured to log to a JSON digest format, while monitoring specific categories of threats like scans, malicious payloads, malformed protocols, etc. Filebeat was also installed on the same Ubuntu VM for forwarding of Suricata's logs, by being configured to monitor Suricata's log directory and send this to a centralized SIEM system.

The SIEM platform utilized the ELK Stack and was hosted on an independent Kali Linux VM. Logstash was configured with parsing filters that enabled it to ingest and format the incoming Suricata logs properly. Each parsed log entry was successfully stored in Elasticsearch making them all searchable in real-time. Kibana, the front-end to this system, was used to build dashboards that visualized live alerts, traffic patterns, and any suspicious activity. Each stage of this log pipeline was tested by generating test traffic (e.g., ICMP, TCP SYN scans, and HTTP requests to known malicious domains). The alerts successfully rendered in Kibana as structured data and communicated the successful end-to-end functionality of the monitoring stack.

Altogether, this complete platform illustrated how network segmentation, a higher performance IDS, and centralized analytics help produce a responsive and intuitive monitoring platform. This project demonstrated not only the success of Suricata to detect intrusion attempts but also how the ELK Stack intuitively promotes rapid data analysis, correlation, and responses. The do-it-yourself approach offered a valuable insight into how operational network security is performed and presented a tangible example of how network security is implemented in academic or business situations.

Chapter 5

5.1 Implementation Challenges and Technical Difficulties

Establishing the resilient campus network security solution—using VLAN segmentation, a Suricata-based IDS/IPS, and centralized log analytics with the ELK Stack—covering both physical and logical, was challenging on several levels, some of which were operational and systemic. Some considerable challenges were uncovered that taught us some vital things about deploying solutions the right way in a real-world technology environment, especially one bounded by virtual environments like EVE-NG.

At the hardware layer, one of the initial obstacles was the available RAM and CPU resources for the VMs. The total RAM was divided among the Suricata node, the ELK Stack (installed on a Kali Linux VM), and several access switches, ultimately creating resource contention. Slowed system behaviors, VM crashes, and substantial lags in boot and service start-up were prominent conditions. In particular, the Elasticsearch component of the ELK Stack had the most difficult challenge with memory during data indexing, and the system demonstrated a great deal of unresponsiveness when under pressure due to memory constraints. In particular, it would not start in terms of a default Java heap of 2 GB.

Furthermore, the host system's inconsistent Internet connectivity wreaked havoc when retrieving packages from upstream repositories; installs (e.g., Filebeat, Suricata, and ELK Stack components) would take a long time and could sometimes result in error messages or failure to install at all.

Within the network layer, VLAN configuration and trunking proved deceptively complex. Although VLANs were logically defined for zones like AUS_ADMIN, TEACHER, STUDENT, and GUEST, their operationalization was hindered by misaligned switch port settings, particularly between the central switches and the router. Several trunk links failed to propagate VLAN tags correctly, resulting in traffic drops and failed ping tests.

One key issue was the position of the initial Suricata VM, which was also hosted in a host-only network on vmnet2 via VMware. Thus, in initial testing, pinging between

PCs within the VLAN towards their router gateway appeared to be successful, but ultimately misleading since the response originated from the VM host IP, bypassing the simulated VLAN path. Thus, the network was not visible and inaccurate testing results were produced. Further, when capturing traffic in Suricata as the VM was running, the only type of traffic observed in the logs from the tcpdump was Spanning Tree Protocol (STP) packets; no ICMP traffic or packets from the VLAN, leading to the obvious conclusion that Suricata was not in a proper monitoring position in the virtual topology. It was overall impossible at that point to check for, verify IDS working or analyze the traffic in said topology.

Additional network-related issues arose from having no native support for SPAN (port mirroring) on the central switch nodes within EVE-NG. This led to unconventional settings to replicate any traffic to the IDS since there was no native mirroring, and attempting to set this up and validate success was not straightforward and often challenging. Hence, it was extremely challenging for Suricata to represent what type of traffic it passively received from the individually defined VLANs resulting in nothing but incomplete packet captures and (no) visibility into inter VLAN activity.

The Suricata installation process was wrought with complications as well. When I first attempted installing via ppa:oisf/suricata-stable, when the repository proved to be deprecated and there were broken dependencies. Following the initial failed installation, when I attempted other versions and installs, the suricata.yaml configuration file regularly pointed to the wrong interface names or rules directories, and the interface, either ens3 or eth0 (I have a dual-homed virtual machine) was not spinning in promiscuous mode, and I captured nothing. Other than the default rules not including ICMP detection, which made my basic demonstration of ping to be invisible to the IDS. Unbeknownst at the time, this led to assumptions of failed system when it was merely documentation that detection rules were missing, or misconfigured.

Filebeat, which is the default method for forwarding files to be analyzed in Elasticsearch was ridden with difficulty as well. I had issues managing GPG keys, outdated repository sources, erroneously configured filebeat.yml files that caused stops to file ingestion. Even though Suricata was generating alerts, Filebeat frequently would not forward the data as anticipated, which led to inconsistencies in what was

visible in Kibana dashboards. And, in conjunction, delivery of filebeat indices to Elasticsearch would at times break the index mapping and make the Kibana dashboard unusable because the versioning was inconsistent between Filebeat, and Elasticsearch.

The deployment of the ELK Stack was particularly delicate. The components installed on the Kali Linux host (Elasticsearch, Logstash and Kibana) often would not go live due to heap issues, version incompatibilities, and unresolved dependencies. Further complicating the troubleshoots were incorrect or missing firewall configurations that blocked access to Kibana (port 5601) and Elasticsearch (port 9200). Under further testing, all Kibana dashboards rendered as blank pages because of non-existent or improperly configured index patterns or simply because log forwarding to the ELK Stack was too delayed to reliably distinguish whether any alerts at all were actioned.

In the final stages of testing, and even with VLANs now configured, inter-VLAN pings frequently failed due to incorrect IP addressing or the presence of untagged native VLAN traffic travelling along trunk links. The traffic destined to be inspected, often didn't even reach Suricata as the interfaces were not set to promiscuous mode or configured to mirror incorrectly within the network's virtual topology. Even after properly configuring with configuration commands, still no alerts of any worth were being generated until much later, evoking confusion about whether it was Suricata or in fact the network failing to operate properly.

5.2 Overcomes and Solutions in Implementation and Testing

Overcoming the challenges faced during the campus network security system implementation required a structured, iterative approach involving resource optimization, topology correction, interface mirroring, and toolchain alignment. A series of refinements transformed the unstable initial deployment into a fully operational virtual lab capable of supporting real-time monitoring and security analysis.

To resolve the hardware constraints, memory and CPU cores allocated to the most demanding virtual machines—Suricata and the ELK Stack—were increased. This reduced system lag, eliminated boot delays, and stabilized services. Elasticsearch and Suricata performed significantly better once system resources were

expanded. To tackle internet reliability issues, a wired host connection was used, and large downloads were scheduled during off-peak hours, minimizing package failures during installations.

During the network reconfiguration phase, all VLAN trunking issues were resolved by manually adjusting switch settings based on Cisco documentation and EVE-NG interface mappings. Trunk ports were explicitly set using:

```
interface GigabitEthernet0/1
switchport mode trunk
switchport trunk encapsulation dot1q
switchport trunk allowed vlan 10,20,30,40,50,60
```

This allowed VLANs to be routed through the central switches to reach the router and have inter-VLAN routing again. At this time, gateway assignments were confirmed for each subnet and ping tests started to be successful.

To solve the issue about Suricata traffic visibility that had become critical, there was a significant redesign of the topology. The previous design had Suricata on a host-only vmnet2 interface and resulted in ping response issues (the ping response was coming from the VM's host IP) and did not route VLAN traffic. Even when using tcpdump only STP packets were visible showing Suricata's placement wasn't effective.

As part of the redesign a central hub switch was placed between the main router and both central switches. Suricata was redeployed and was connected to this hub for great traffic mirroring. The Suricata VM Cloud1 on vmnet2 was also connected to the ELK Stack to facilitate log forwarding.

On EVE-NG, promiscuous mode was manually enabled on all mirrored virtual interfaces to ensure traffic forwarding:

```
root@SPOTO-EVE:~# sudo ip link set vnet0_10 promisc on
root@SPOTO-EVE:~# sudo ip link set vunl0_20_0 promisc on
```

Additionally, traffic control filters (tc) were used to mirror ingress packets from VLAN 10's virtual NIC to Suricata's interface:

```
root@SPOTO-EVE:~# sudo tc qdisc add dev vnet0_10 ingress
root@SPOTO-EVE:~# sudo tc filter add dev vnet0_10 parent ffff: \
>     protocol all u32 match u8 0 0 \
>     action mirred egress mirror dev vunl0_20_0
```

Inside the Suricata VM, its interface (ens3) was explicitly set to promiscuous mode:

```
~$ sudo ip link set ens3 promisc on
```

These steps allowed Suricata to observe full inter-VLAN traffic, resolving earlier detection failures and validating its new position in the network.

Suricata installation was improved by deprecating old PPAs, and instead will be using official OISF packages. When thoroughly testing Suricata, the other deployment method of Docker was functional for patch management. yaml int was utilized to eliminate YAML file errors, and Suricata was properly bound to the NIC with accurate ruleset paths. To improve upon the detection capabilities, the Emerging Threats ruleset was also installed, as this ruleset can now also detect ICMP scans, DNS tunnelling, port sweeps, and malformed packets.

To fix Filebeat-related issues, the correct Elastic GPG key was added:

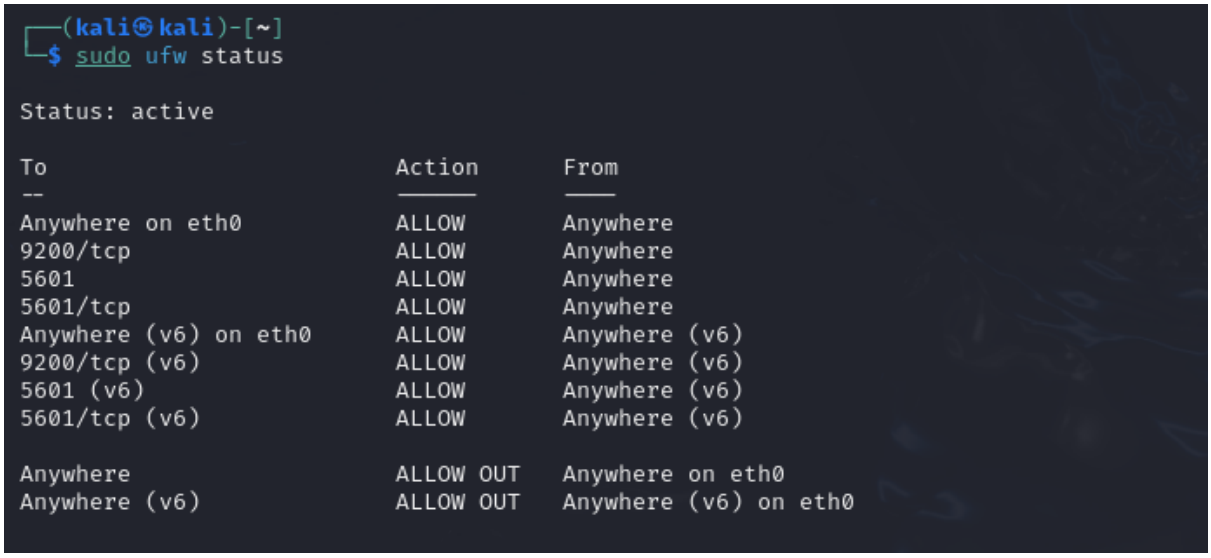
```
wget -qO – https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add –
```

The Elastic APT repo was then appended to /etc/apt/sources.list, and versions were aligned with Elasticsearch. The filebeat.yml file was corrected to point to Suricata's logs at:

- /var/log/suricata/eve.json

This enabled reliable log ingestion from Suricata to ELK.

The ELK Stack was stabilized by aligning all component versions. Firewall rules were also corrected:



Index patterns in the Kibana application were correctly defined to identify Filebeat's log indices which allowed for the real-time analysis of security events. Dashboards were set up to identify Suricata alerts and analytics of network traffic, showing end-to-end functionality to the monitoring system.

Based off the improvements made in the installation and implications of theoretical study and after testing, it was apparent that overall performance improved after remedial actions were made to the monitoring architecture. All hosts within VLANs were reached with success from a ping, which demonstrated that VLAN and trunking were configured correctly. Suricata logs started to show real-time alerts for network traffic anomalies, as well Filebeat addressed delivery and forwarded the logs in a timely fashion to Elasticsearch. Kibana dashboards now showed rich visualizations for better threat detection and situational awareness.

Chapter 6

Conclusion and Future Perspectives

6.1 Comprehensive Summary of Project Outcomes

This the project achieved the establishment, implementation, and evaluation of a complete campus network security monitoring solution, which encompasses VLAN segmentation, Suricata IDS/IPS, and the ELK Stack SIEM. It illustrated that when foundational networking concepts and open-source security tools are utilized, scalable and responsive infrastructure can be successfully integrated into contemporary

organizational contexts. VLAN segmentation was effectively employed to segment departmental traffic in accordance with the IEEE 802.1Q VLAN standard, therefore enhancing security and performance within the institutional infrastructure. One of our largest accomplishments was to overcome early detection failures caused by attempting to place Suricata inside a limited host-only network; after re-connecting Suricata to the central hub embedding and using promiscuous response mode on applicable interfaces, detection capabilities were functioning adequately.

By successfully deploying filters and ip link configurations unique to each VLAN, the traffic across the VLANs was finally copied into Suricata VM for total visibility. The core detection capabilities of Suricata were enhanced with current ruleset files, and the proper interfaces on the Suricata VM are allocated. Having the ELK Stack deployed also allowed central oversight into Suricata alerts and behaviours on the network. Filebeat was lastly, successfully deployed to forward logs into Elasticsearch from Suricata after previous issues with repository compatibility, packages compatibility, and configuration limits. After resolving Kibana index duplication, some blocked firewall settings, and memory allocation limits, we achieved total situational awareness of the events being tracked during network security traffic analysis completing the stack.

Testing confirmed successful alerting of scanning and malformed packets, validating the end-to-end security architecture. The project's iterative troubleshooting—from interface binding errors and VLAN misconfigurations to ELK pipeline tuning—demonstrated the critical importance of systematic validation and cross-layer integration. Ultimately, the project delivered a fully functional, adaptable, and educationally valuable security monitoring system, aligned with real-world cybersecurity principles and best practices.

6. 2 Key Achievements and Contributions

The successful delivery of this project provided many key deliverables that incorporate both technical skills and real-world applicability in network security monitoring. The most significant deliverable was the deployment of VLAN segmentation across the the campus infrastructure to enhance security through logical isolated traffic and improve network performance with limited broadcast domains. By strictly following the IEEE 802.1Q standard this also need an expectations in

interoperability and framework for segregation and future scalability. The project delivered the capability of Suricata as an operational IDS/IPS as a no cost, we will look to implement more in the future. The system had successfully demonstrated capture of real-time threats based on its multi-threaded inspection, through customized rules and external threat intelligence integration Suricata was able to adjust to the quickly shifting patterns of an attack. The successful deployment of the ELK Stack as a situational awareness single pane of glass SIEM and centralized log parsing, storage, and visualization was an additional success.

The integration of Elasticsearch, Logstash, and Kibana provided real-time dashboards and forensic depth allowing security analysts to have full situational awareness. The technical implementations, however, came with challenges typical of any deployment, including memory and CPU restrictions, VLAN misconfiguration, Suricata's unsuccessful capture of the mirrored traffic, and the software version mismatches. Each hurdle was addressed via solution approach in the form of resource retention, configuring Suricata's interface setup and tc filter method to assist with traffic mirroring, and ELK pipeline development and configuration. Furthermore, it highlighted the benefits of open-source resources in executing scalable and resilient security architectures, especially when the true value of the solutions is proved through a process of validation and adaptation. Ultimately, the result was a working monitoring framework demonstrating appropriate levels of integration achievable from a budgetary, security, and pedagogical standpoint that can further meaningful contributions to modern cybersecurity practices and be a working reference model for students or enterprise when pursuing similar objectives.

6.3 Lessons Learned from System Design and Implementation

During this project, a number of useful lessons arose that highlighted the pragmatic challenges of developing and deploying an end-to-end network security monitoring system (NSMS). One of the first, and perhaps most important realizations included understanding the anticipation, planning and the necessity and importance of availability of useable infrastructure—specifically planning for hardware resources. Insufficient RAM and CPU resource allocation delayed installations significantly and caused outages intermittently, which reinforced the importance of doing adequate resource planning and forecasting capacity prior to being deployed. The project also

highlighted the importance of adopting a good network configuration practice. VLAN segmentation, trunking, and port mirroring made it necessary to focus on how switches may be set and protocols for tagging; the slightest miss-configuration in setting would cause failures of communication paths and no access to hosts. Enabling promiscuous mode in the virtual environment for packet capture was also a critical, and overlooked, factor that we utilized when using Suricata, which resulted in baselines that we did not clearly acknowledge until we missed a key indicator that degraded the environment.

From a software perspective, getting open-source tools like Suricata, Filebeat, and the ELK Stack running was not without a number of technical problems. This included repository failures, broken dependencies, and bad configurations, which only buttressed the importance of validating package sources, version consistency, and understanding the first principles of how each tool interfaced with the system. Iterative refinement was also a major lesson. The first rule sets in Suricata were either not adequate or did not detect the behaviors we expected, leading to missed alerts or too much noise.

Using community-driven rulesets and regularly ingested threat intelligence had a huge impact in improving detection reliability. Finally, the exercise highlighted that success is not dependent on simply installing tools, but rather, on technologically embedding these tools into a working pipeline. End-to-end visibility-from data ingestion to visualization- requires functional interoperability between all the tools. Small, seemingly insignificant problems (for example, a Kibana index pattern, firewall restrictions) can sever the chain of visibility altogether.

Overall, the project highlighted that a systems approach to deploying cybersecurity systems isn't simply a technical exercise but requires adaptive problem solving, iterative experimentation, and cross-disciplinary fluency. The lessons learned represent a deepening of our collective practical knowledge, enhancing, not simply tool competence, but also the strategic mindset needed to secure evolving and complicated threat environments.

6.4 Future Work and Recommendations for System Enhancement

Based on the lessons learned and experiences obtained while working on this project there are a few key avenues to consider for future work when thinking of how

to improve scalability, performance, and intelligence in the network security monitoring system. One of the most significant recommendations is to increase hardware resources, specifically increasing RAM and CPU resources which would greatly increase the system's ability to deal with increased network traffic while also providing complex real-time analyses without degrading performance. This need becomes more critical as networks become larger or encounter increased sophisticated cyber threats. Additionally, strategically changing VLAN configurations and looking into new forms of segmenting—dynamic VLAN assignment and Software-Defined Networking (SDN)—could provide improved grouping of policies, network agility, and responsiveness to changing conditions from an architectural perspective.

On the software side, maintaining up-to-date Suricata rule sets and fine-tuning Filebeat log parsing are crucial for accurate detection and streamlined data ingestion. Integrating machine learning–based anomaly detection modules within the ELK Stack, such as Elastic ML, would elevate the system from reactive to predictive, enabling the identification of unknown threats and supporting automated incident response. Additionally, automating the deployment and configuration process using infrastructure-as-code tools like Ansible or Terraform would improve reproducibility, reduce configuration errors, and simplify scaling or re-deployment across different environments.

It is of equal importance to also enhance the monitoring layer. If, in Kibana, we generated more tailored dashboards and alerting that reflect the organizational risk priorities, it would enhance real-time visibility and decision-making. Additionally, integrating the system with an external threat intelligence feed, along with SOAR platforms, would add layers of context and potentially allow more proactive and coordinated defensive action to take place. Finally, carrying out system testing against more realistic and advanced threat models, including APTs and zero-day scenarios, would both stress-test system resilience, and identify any detection blind-spot that may need filling. By also collaborating with open-source security communities to share appropriate rules and intelligence updates, our system can remain relevant and adaptable.

In short, the next foreseeable evolution of this system involves scaling infrastructure capacity, adopting emerging technologies, automating operations, and better integrating threat intelligence. This would build a transitional path toward a more resilient, adaptive and intelligent cybersecurity defense framework aligned with modern enterprise networks.

References

(OISF), O. I. S. F., 2024. *Suricata Documentation.* [Online]
Available at: https://docs.suricata.io/
[Accessed 16 June 2025].

Ahmad, A. M. S. a. P. S., n.d. *Information security strategies: Towards an organizational multi-strategy perspective.* [Online]
Available at: https://doi.org/10.1007/s10845-012-0693-8
[Accessed 18 June 2025].

Alsmadi, I. &. Z. M., 2019. 'A Framework for Secure Campus Networks Based on SDN'. *International Journal of Computer Network and Information Security (IJCNIS),* Volume 11, pp. 21-29.

Bejtlich, R., 2013. The Practice of Network Security Monitoring: Understanding Incident Detection and Response. In: San Francisco, USA: s.n.

Bhuyan, M. B. D. &. K. J., 2014. Network Anomaly Detection: Methods, Systems and Tools. *Computers & Security,* Volume 42, pp. 1-23.

Camisso, J., 2021. *How To Install Suricata on Ubuntu 20.04.* [Online]
Available at: https://www.digitalocean.com/community/tutorials/how-to-install-suricata-on-ubuntu-20-04
[Accessed 17 June 2025].

Corporation, M., 2025. *MITRE.* [Online]
Available at: https://attack.mitre.org
[Accessed 17 June 2025].

Easttom, C., n.d. Computer Security Fundamentals (4th ed.. In: s.l.:Pearson IT Certification.

Foundation, L., 2024. *Understanding Traffic Mirroring for IDS Integration,* San Francisco, CA, USA: The Linux Foundation.

GitHub, 2024. *Filebeat + Suricata ELK Integration Examples.* [Online]
Available at: https://github.com/StamusNetworks/SELKS
[Accessed 17 June 2025].

Gupta, A. M. M. &. S. V., 2020. Evaluation of ELK Stack for Network Intrusion Detection. *International Journal of Computer Applications,* Volume 176, pp. 10-15.

IDS, S., 2024. *Suricata User Guide 7.0.0,* Washington, DC, USA: Open Information Security Foundation (OISF).

Kumar, R. &. T. J., 2021. Cybersecurity for Smart Campuses: Threats, Trends, and Recommendations. *IEEE.*

Labs, E. S., 2025. *Elastic SIEM Detection Rules and Anomaly Detection with ML.* [Online]
Available at: https://www.elastic.co/security-labs
[Accessed 17 June 2025].

Manual, U., 2023. *Uncomplicated Firewall (UFW) on Ubuntu.* [Online]
Available at: https://help.ubuntu.com/community/UFW
[Accessed 18 June 2025].

Networks, P. A., 2024. *Understanding the MITRE ATT&CK Framework.* [Online]
Available at: https://www.paloaltonetworks.com/resources/mitre
[Accessed 18 June 2025].

NIST, 2022. *NIST Special Publication 800-61 Rev. 2,* Gaithersburg, MD, USA: National Institute of Standards and Technology (NIST).

Oltramari, A. &. C. S., 2020. *'Improving Suricata's Accuracy Using Knowledge-Driven Techniques,* Pittsburgh, PA, USA: Software Engineering Institute, Carnegie Mellon University.

Project, N., 2024. *Nmap Reference Guide.* [Online]
Available at: https://nmap.org/book/man.html
[Accessed 18 June 2025].

Scarfone, K. a. M. P., 2025. *Guide to Intrusion Detection and Prevention Systems (IDPS), SP 800-94,* Gaithersburg, Maryland: NIST.

Stallings, W., n.d. Network Security Essentials: Applications and Standards (6th ed.). In: s.l.:Pearson.

Systems, C., 2025. *Configuring VLANs.* [Online]
Available at:
https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst2960/software/release/12-2_55_se/configuration/guide/scg2960/swlan.html
[Accessed 18 June 2025].

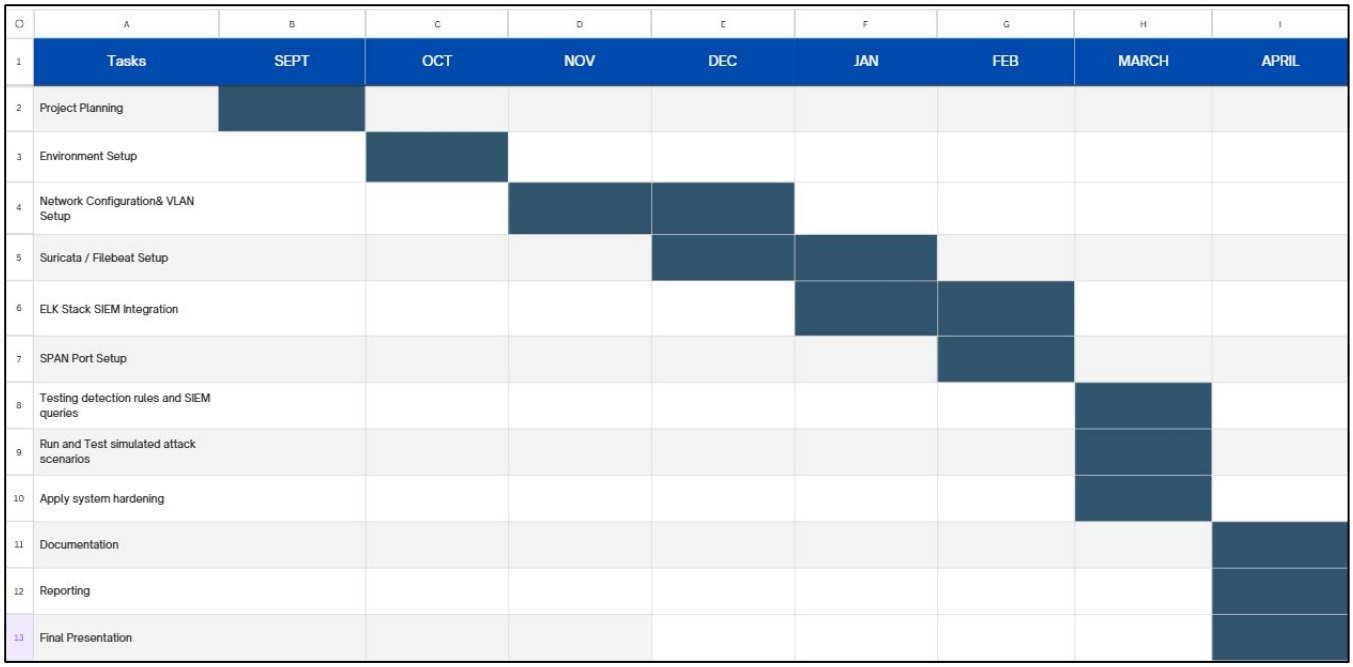
Team, U. D., 2025. *Ubuntu Server Documentation.* [Online]
Available at: https://ubuntu.com/server/docs
[Accessed 16 June 2025].

Wiki, D., 2024. *Promiscuous Mode and Packet Capture on VirtualBox.* [Online]
Available at: https://wiki.debian.org/BridgeNetworkConnections
[Accessed 18 June 2025].

Zuech, R. ,. K. T. ,. W. R., n.d. *Intrusion detection and Big Heterogeneous Data: A Survey.* [Online]
Available at: https://doi.org/10.1186/s40537-015-0013-4
[Accessed 18 June 2025].

APPENDIX

PROJECT GANTT CHART



Tasks	SEPT	OCT	NOV	DEC	JAN	FEB	MARCH	APRIL
Project Planning								
Environment Setup								
Network Configuration& VLAN Setup								
Suricata / Filebeat Setup								
ELK Stack SIEM Integration								
SPAN Port Setup								
Testing detection rules and SIEM queries								
Run and Test simulated attack scenarios								
Apply system hardening								
Documentation								
Reporting								
Final Presentation								

LIVERPOOL JOHN MOORES UNIVERSITY

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting : September

Student: Khaing Zin Thein

Main issues / Points of discussion / Progress made
• Initial project idea refined and approved. • Project scope defined: Campus Network Security System using Suricata IDS and ELK Stack. • Research on current trends in network security and SIEM integration.
Actions for the next month
• Begin environment setup (EVE-NG, VMs, host configuration). • Draft high-level project plan and Gantt chart.
Deliverables for next time
• Completed project plan with milestone timeline. • Tools and software list finalized.
Other comments

Supervisor Signature:

Student Signature: 

LIVERPOOL JOHN MOORES UNIVERSITY

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting : October

Student: Khaing Zin Thein

Main issues / Points of discussion / Progress made
• Successfully set up EVE-NG platform and initial virtual machines (Ubuntu, Kali). • Installed base tools for testing and monitoring. • Researched VLAN setup requirements.
Actions for the next month
• Begin configuring VLANs and trunk ports on Cisco switches. • Establish connectivity between virtual nodes.
Deliverables for next time
• Working base network with VLAN segmentation plan. • Screenshot evidence of working EVE-NG environment.
Other comments

Supervisor Signature :

Student signature: 

LIVERPOOL JOHN MOORES UNIVERSITY

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting : November

Student: Khaing Zin Thein

Main issues / Points of discussion / Progress made
• Configured and tested VLANs and trunk ports. • Resolved EVE-NG interface mapping and default gateway misconfigurations. • Documented VLAN design with subnetting.
Actions for the next month
• Integrate Suricata and Filebeat into the system. • Begin SPAN/mirroring setup for packet capture.
Deliverables for next time
• Functional VLAN environment with inter-VLAN routing. • Suricata partially installed and network capturing tested.
Other comments

Supervisor Signature:

Student Signature: 

LIVERPOOL JOHN MOORES UNIVERSITY

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting : December

Student: Khaing Zin Thein

Main issues / Points of discussion / Progress made
• Completed Suricata installation with correct NIC bindings. • Filebeat setup started; some issues with version compatibility. • Port mirroring configured using workaround due to EVE-NG limitations.
Actions for the next month
• Complete Filebeat and ELK Stack integration. • Begin threat detection testing using test traffic.
Deliverables for next time
• Verified log forwarding from Suricata to Elasticsearch. • Kibana dashboards partially functional.
Other comments

Supervisor Signature:

Student Signature: 

LIVERPOOL JOHN MOORES UNIVERSITY

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting : January

Student: Khaing Zin Thein

Main issues / Points of discussion / Progress made
• ELK Stack fully functional; Filebeat forwarding tested successfully. • Visualizations created on Kibana with basic Suricata alert data. • System resource upgrades implemented to improve performance.
Actions for the next month
• Optimize dashboard views. • Download latest threat intel rulesets. • Start testing against simulated attacks.
Deliverables for next time
• Optimized and filtered dashboard for port scans, DNS anomalies, ICMP alerts. • Detection logs triggered by Nmap or crafted packets.
Other comments

Supervisor Signature :

Student Signature: 

LIVERPOOL JOHN MOORES UNIVERSITY

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting : February

Student: Khaing Zin Thein

Main issues / Points of discussion / Progress made
• SPAN mirroring improved; detection rules working well. • Updated Suricata rules from Emerging Threats. • YAML configs cleaned and automated using lint tools.
Actions for the next month
• Test system resilience through continuous traffic flow. • Introduce some stealthy or complex attack simulations.
Deliverables for next time
• Detailed test logs with timestamps and matching alerts. • Observations on false positives or missed detections.
Other comments

Supervisor Signature:

Student Signature: 

LIVERPOOL JOHN MOORES UNIVERSITY

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting : March

Student: Khaing Zin Thein

Main issues / Points of discussion / Progress made
• Ran multiple simulated attacks and validated detection performance. • Applied system hardening (UFW firewall, services restricted). • Detection alert accuracy improved via rule tuning.
Actions for the next month
• Complete detailed documentation. • Begin preparing presentation material and results summary.
Deliverables for next time
• Complete draft of final report. • Screenshots and summaries of attack detection and system flow.
Other comments

Supervisor Signature:

Student Signature: 

LIVERPOOL JOHN MOORES UNIVERSITY

BSc(Hons) Final Year Project

Monthly Supervision Meeting Record

Month Meeting : April

Student: Khaing Zin Thein

Main issues / Points of discussion / Progress made
• Final documentation and project report completed. • Final presentation prepared with visualization of alerts, dashboards, and architecture. • Received feedback from test audience and incorporated improvements.
Actions for the next month
• Submit final report. • Deliver final presentation and project handover.
Deliverables for next time
• Final printed report. • PowerPoint slides and recorded demo
Other comments

Supervisor Signature:

Student Signature: