Report 2

Security Onion **2.4**: *A Brief Exploration of Zeek and Suricata*

Devarth Patel
KUID: 3040746

EECS 465
Prof. Alex Bardas

Abstract

This report presents an end-to-end scenario where a standalone Security Onion 2.4 sensor, running inside VirtualBox on a Kali-Linux machine host, monitors incoming traffic and defends a small lab subnet. The sensor employs two flagship engines - Suricata 6, which monitors signature IDS/IPS as well as Zeek 5, which monitors network behaviors. Security Onion is successfully installed, tuned and driven with live traffic which includes ICMP bursts, HTTP browsing, an Nmap Scan and a Telnet probe. Within the demo component of this report, the images presented involve VirtualBox configuration, ISO mounting and installation, Security Onion setup, NIC binding, Tool status, dashboards and Zeek logs. The exploration of these resources shows that Suricata provides high-confidence rule hits while Zeek boasts context enriched forensic analysis which in turn, forms a complementary defense even within smaller systems.

Introduction

Network security monitoring has evolved drastically over the past two decades where it is has transformed from simple packet capturing and basic signature matching to highly complex, multi-layered detection frameworks. The increasingly difficult security challenges of the modern world are a critical motivator to keep adapting security defenses and growing the understanding of attacks. Contemporary challenges could include sophisticated evasion methods, encrypted traffic, high-speed networks, and adaptive attackers. So, among these threats, professionals need complementary perspectives to effectively identify, analyze and respond to any threats. Network defenses require a balance of detecting numerous threats whilst collecting valid contextual

information so that the nature, scope and impacts can be understood. This understand would require two distinct approaches to traffic analysis:

- **Signature Based Detection**
    - This approach matches the traffic that is observed with the known patterns of malicious activity, allowing for easy identification. It is ideal for recognizing established threats with high confidence and low false-positive rates. Some signature engines, such as Suricata, raise flags immediately when detecting patterns that match:
        - Known exploitation attempts
        - Command and control communications
        - Suspicious protocol behavior
        - Network policy violations

      The signature based approach delivers a high specificity but struggles with attacks that have been previously unidentified and increasingly sophisticated evasion techniques.
- **Behaviour Based Detection**
    - An ideal complement to signature detection is behavioural analysis which is what builds protocol-strict models of network communication in order to identify anomalies and give defenders a context-rich forensic breakdown. Behavioral engines, such as Zeek, are great tools for:
        - Tracking the session state across complicated interactions
        - Parsing and validating numerous network protocols
        - Identifying statistical anomalies

    - Constructing timelines of activities

The behavior based approach gives a deeper understanding and context of the activity but can create an overwhelming volume of data without the precision of signature based alerts, making it hard to separate activity.

Security Onion understands the drawbacks of standalone engines and integrates these complementary approaches, all into one unified platform. Instead of treating the signature and behavior detection separately, security onion's architecture connects them through; unified packet captures (which feeds the tools simultaneously), common data format, synchronized timestamps/flow identifiers and a streamlined workflow between the detection and investigation stages. The integration of these makes the traditional security monitoring forensics more intelligent and gives deeper understanding of attacks and through this, can prevent potential security issues.

Lab Context

- The workstation used was a Kali-Linux host machine inside the Cybersecurity Lab (Room 2003, Eaton Hall) which was originally equipped with 16 GB RAM but had to be modified to 24 GB by hand due to Virtual Box and Security Onion requirements. I modified this by taking an 8 GB RAM card from the neighboring Host machine and installing it on my own.
- My Security Onion, 'SecurityOnionDev', was provided with six vCPUs, 23655 MB RAM and over 100 GB SSD in compliance with Security Onion and VirtualBox requirements.
- The ISO image for Security Onion was mounted on top of Oracle Linux 9.

- Two virtual NICs were attached and configured, both of which were placed in promiscuous mode:
    - One bridged interface for management traffic and updates
    - Host-Only interface
- A second Virtual Machine was set up, pointed to the Security Onion setup to test generating traffic. It did not follow any constraints.

Report Objectives

This exploration will detail the deployment, configuration and testing of Security Onion 2.4 in a setup controlled environment. Through rigorous testing and forensic analysis we can understand the implementation of Suricata's signature-based detection and Zeek's behavioral monitoring, which creates a greater security monitoring capability for professionals. I capture and analyze the complete process to give a comprehensive assessment of the effectiveness of the platform in modern defense scenarios. This exploration focuses on:

- Installation - from ISO mounting to booting
- Security Onion status monitoring and NIC binding
- Injecting traffic and correlating it with Suricata and Zeek output

Architecture

Security Onion 2.4

Security Onion has an implementation of a layered architecture that combines multiple open-source security tools onto a singular, cohesive platform that allows users to monitor. Security Onion is built onto a foundation of Oracle Linux 9 which allows for enterprise level scalability while allowing support for containerization. There are several critical layers, seamlessly working together to provide monitoring support, these layers include:

- *Infrastructure*

  Security Onion employs Docker containers which are managed through Salt for deployment consistency and lifecycle management. Using containerization isolates each component, simplifies update processes and allows for flexible scaling for different sensors. It includes the Docker Engine for container management, SaltStack for configuration management, Nginx for web services/proxying and finally Redis for caching and communication.

- *Data Collection*

  At this level, Security Onion captures raw network data and processes it through AF-PACKET for the high-performance capture and distribution of packets, Suricata for a full packet capture and also an interface bonded for unified capture across different interfaces.

- *Detection Tools*

  The primary forensic analytical tools are processed at this level. We look at Suricata for signature based alerts and detection, then at Zeek for protocol analysis and the behavior monitoring process.

- *Data Processing*

Before the captured data is indexed, it flows through Logstash (to be normalized), Elastic Agent (which collects that data and forwards it) and also Filebeat for the log shipping.

- *Storage and Indexing*

  Elasticsearch is the primary resource that Security Onion uses to store data with, it provides time based indices to help with performance optimization, Lifecycle management options for the retention policies and a 'Hot-warm-cold' type of architecture for storage optimization.

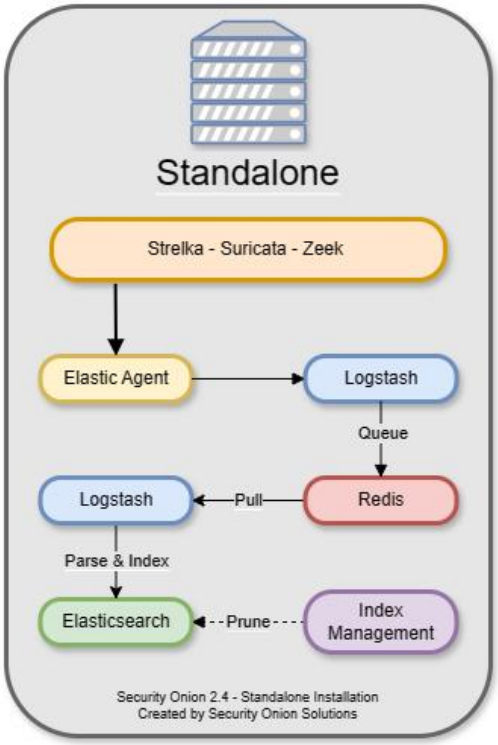
- *User Interface*

  The analytical functionality is highlighted by Security Onion's simplified UI which includes a Security Onion console with an alert tab and dashboards, a hunt interface to find threats, case management and Kibana for searching as well.

Below is a diagram to depict the processes that Standalone Security Onion 2.4 system executes and we can visualize the levels of processing through it:

We can refer to our layer breakdown of Security Onion and compare it to the diagram which gives us a more complete and visual understanding of the architecture of the standalone version of Security Onion 2.4.

(*Figure 1*: Standalone version of Security Onion)

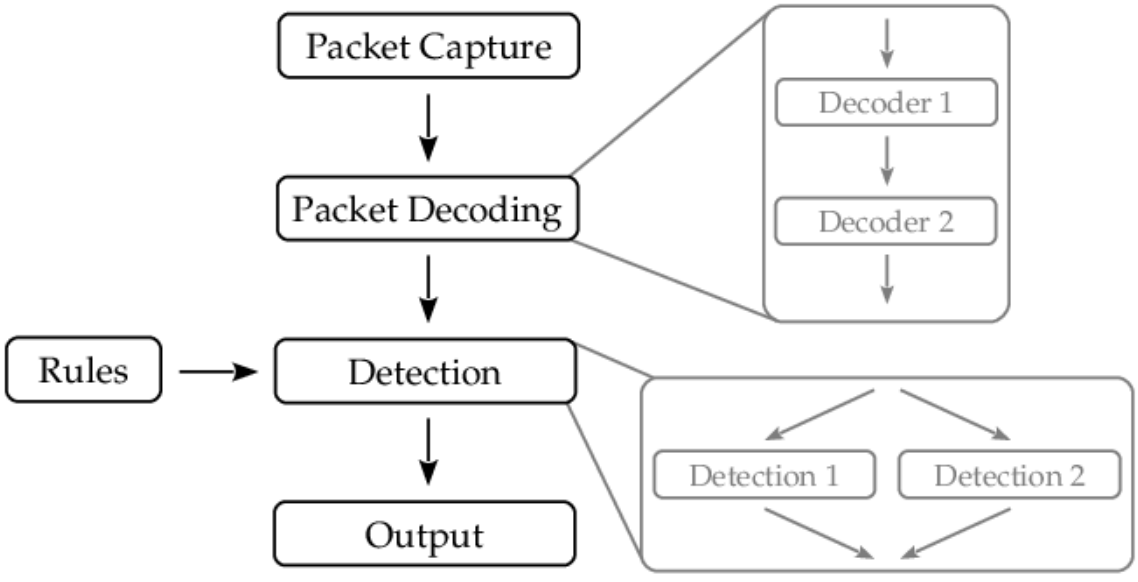


Suricata 6

Suricata is used as Security Onion's main signature-based detection engine, operating inside containers to preserve scalability, isolation, and simplified updates via SaltStack. It employs AF-PACKET for high-performance packet capture directly from network interfaces that are in promiscuous mode, allowing full visibility to see raw traffic.

Once traffic is captured, Suricata performs deep packet inspection (DPI), scanning both the headers and payloads for any known signatures that match threat patterns. When it matches to a threat, Suricata generates structured JSON alerts that are forwarded through Filebeat, processed by Logstash, and indexed into the Elasticsearch.

It uses a multithreaded architecture that allows it to take advantage of multiple CPU cores, supporting high-throughput environments. Beyond its alerting capabilities, it also provides the flow records, file extraction, and protocol classification. However, it lacks contextual depth. This is where it gets complimented. So, it flags threats in real-time and hands off behavioral analysis to Zeek. Together, this pairing makes Security Onion's defense layered and a lot more stronger.

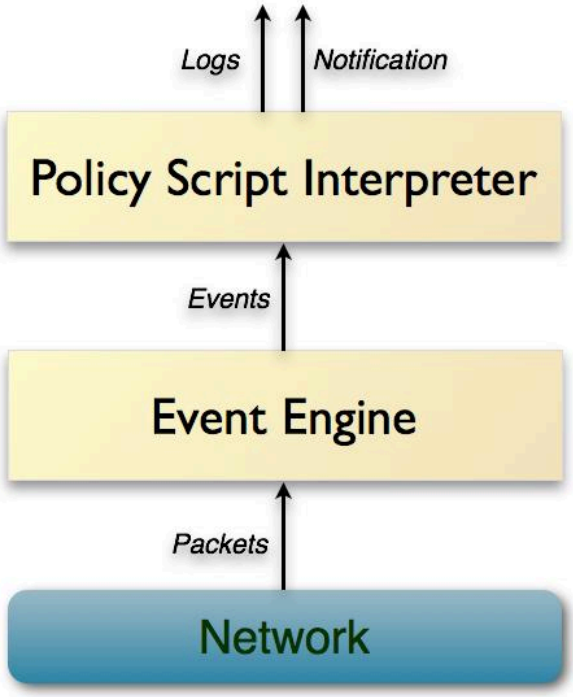


(*Figure 2*: Suricata's architecture describes how it refers to rules that it uses to match raw data to see if an alert needs to be sent for threats.)

Zeek 5

Zeek, formerly known as bro, works as a behavior analysis tool, focused on protocol dissection and deep forensic analysis. Zeek gets traffic through AF-PACKET interfaces (like Suricata), but instead of referencing known patterns, it logs the detailed observations about each session. It uses protocol analyzers to generate logs such as conn.log, http.log, dns.log, and ssl.log, where each of these are capturing different aspects of the network flow.

While Zeek doesn't offer high-confidence alerts like Suricata, it delivers a different function by offering deep context: who was communicating with whom, when, how long, and what was said. It helps explain potential treats as well as post-attack analysis to essentially form an investigative backbone of Security Onion.

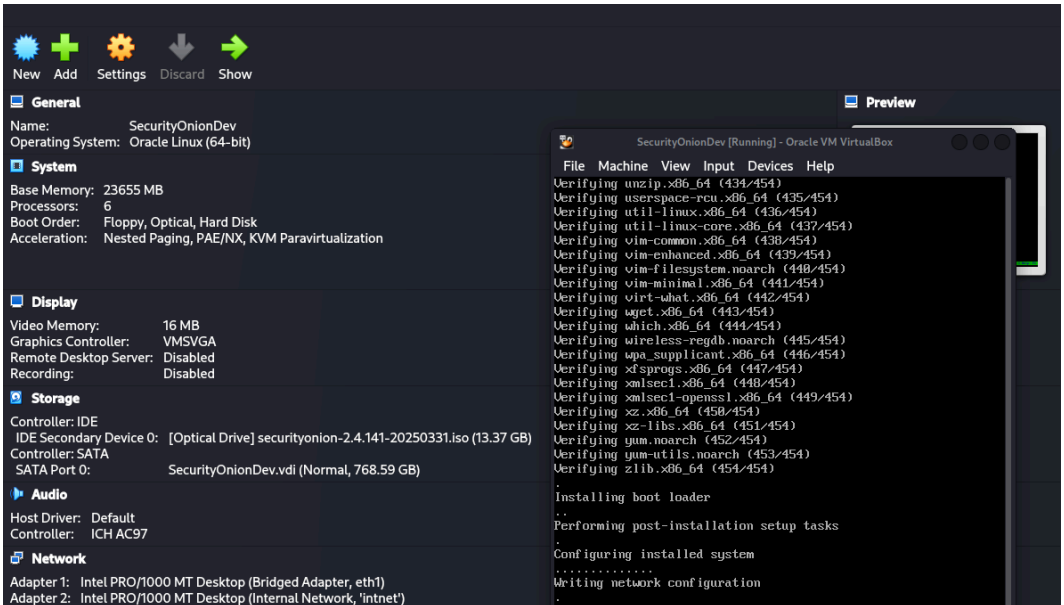


(*Figure 3*: Zeek's architecture is described here as network packets are input through its event engine which then runs the events through its policy script interpreter to generate logs and notifications)

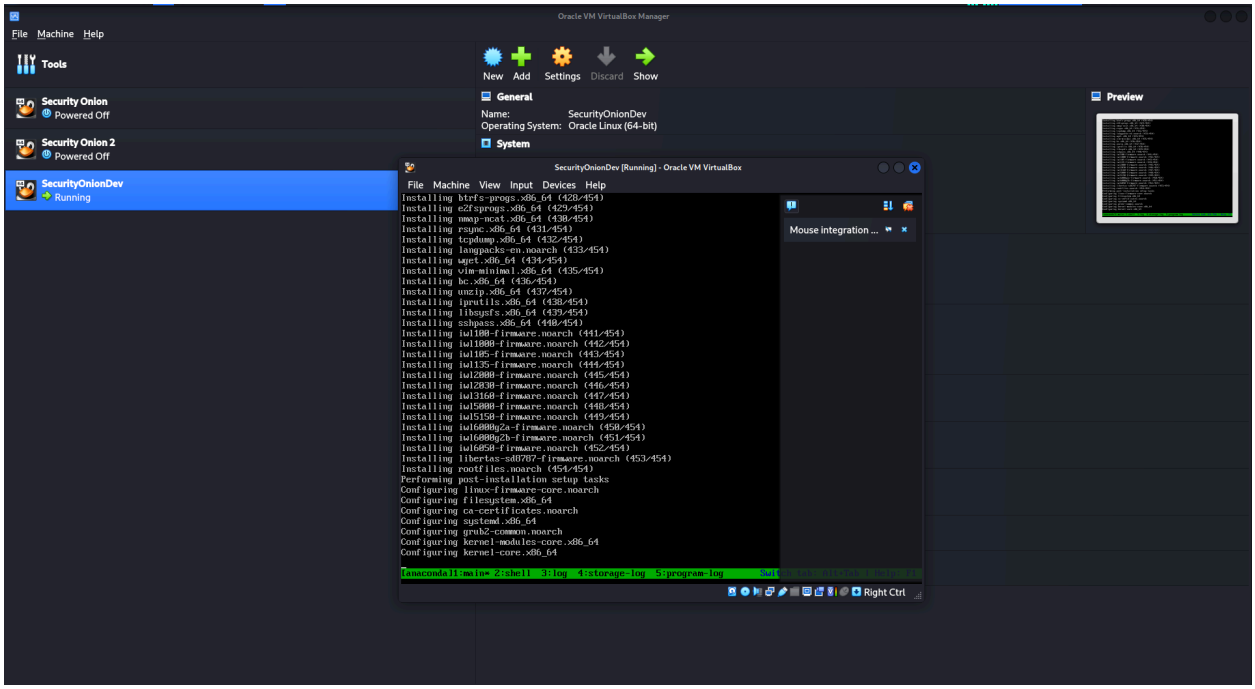
Demo

In this section the report will explore how I implemented Security Onion, used Zeek and Suricata from end-to-end.

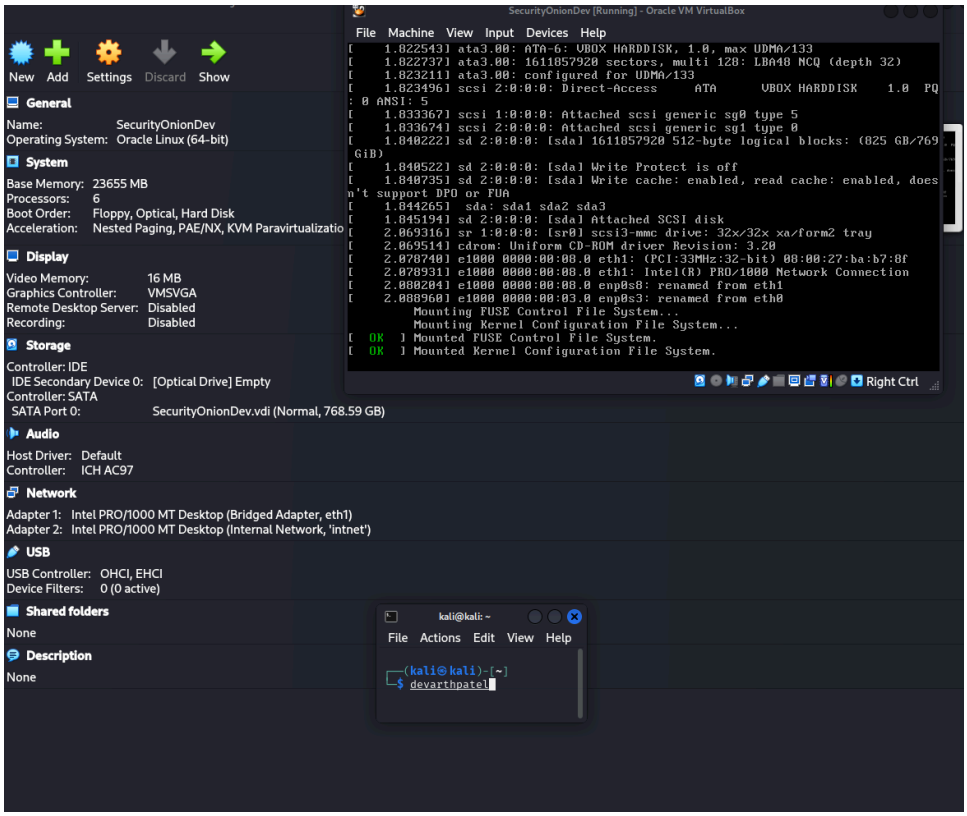
- **[D1]** To install Security Onion 2.4, I created a Virtual Machine in Oracle VirtualBox with 6 processors, 23.6 GB RAM, and over 100 GB of storage. The Security Onion ISO was mounted and booted on this VM. The ISO file securityonion-2.4.141-20250331.iso was mounted as the primary boot source. The installation began with Security Onion starting its base system on Oracle Linux 9. This included installing and configuring packages such as firmware libraries, kernel modules, and networking utilities. I could monitor the painstakingly slow progress through the terminal inside the VirtualBox environment.



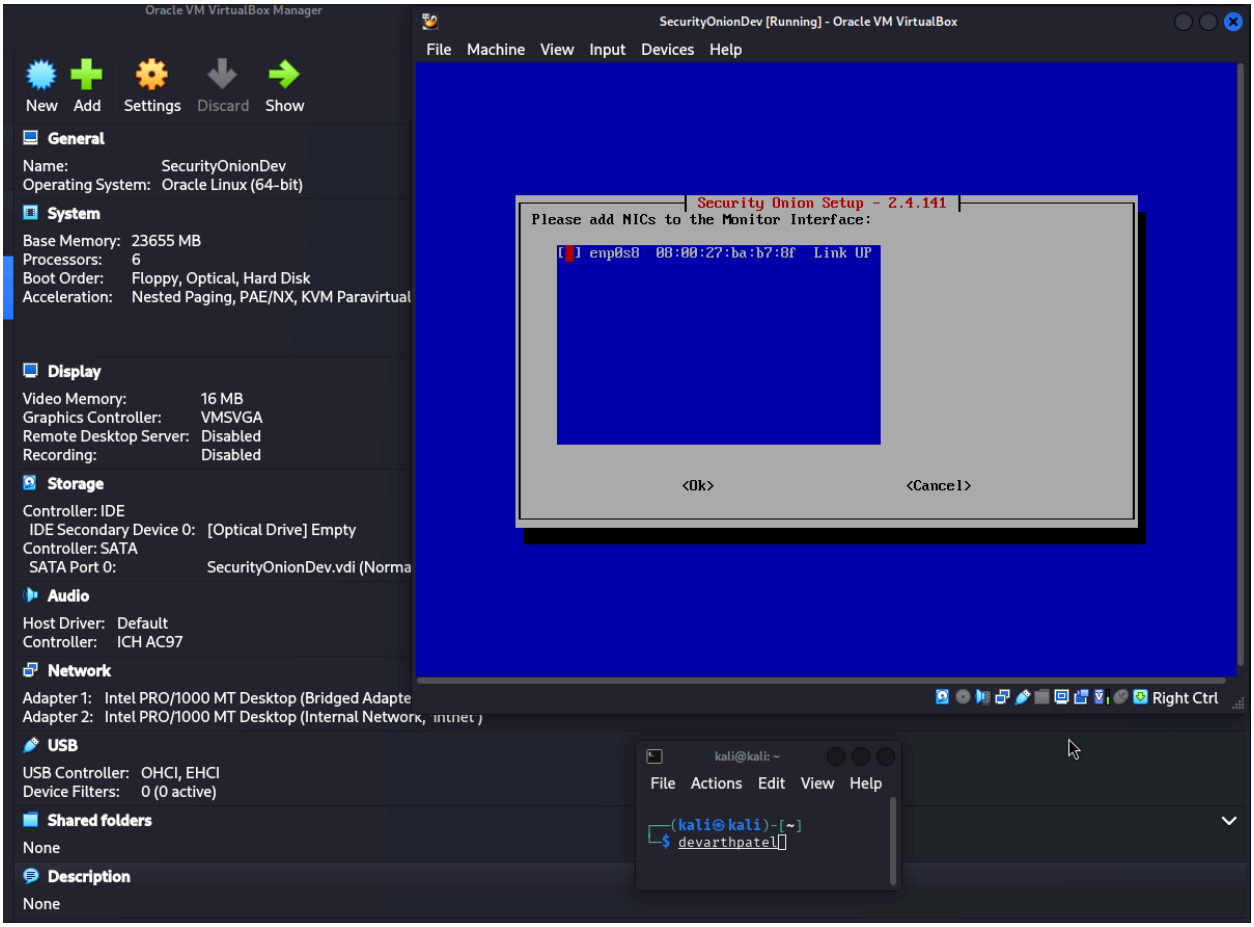
[D1.1]: VM config and Installation Start



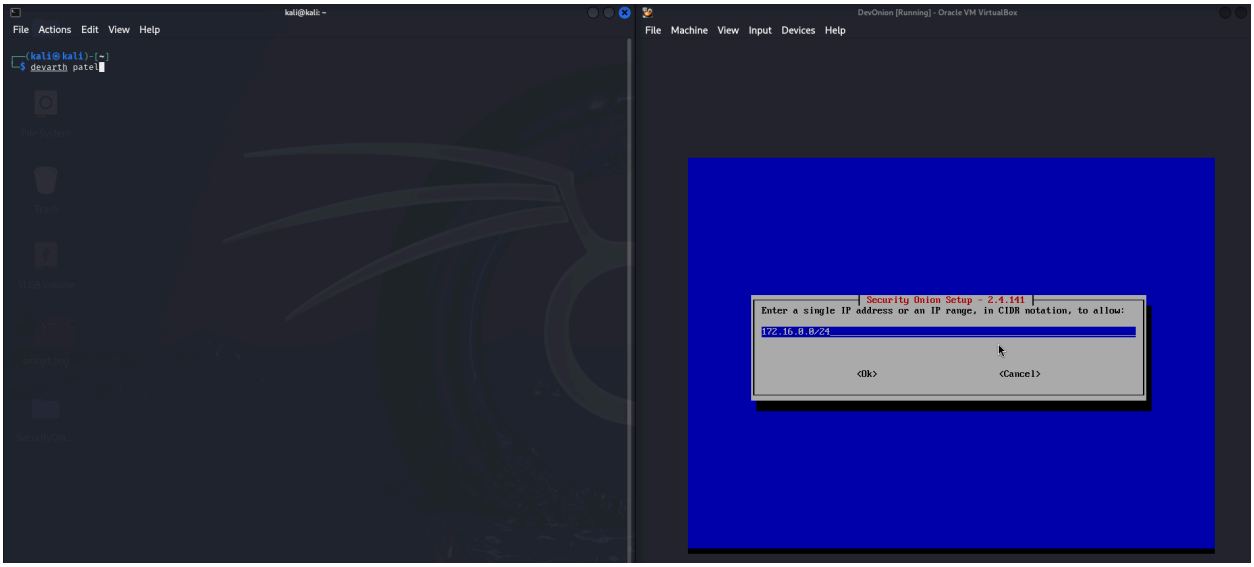
[D1.2]: Installation



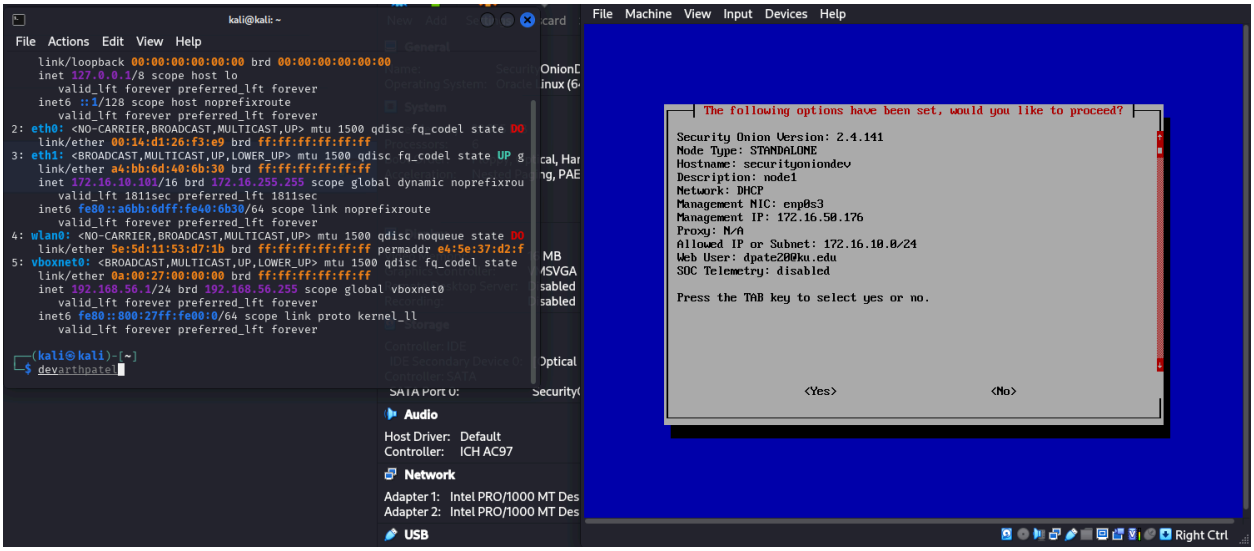
[D1.3]: Still installing and unpackaging



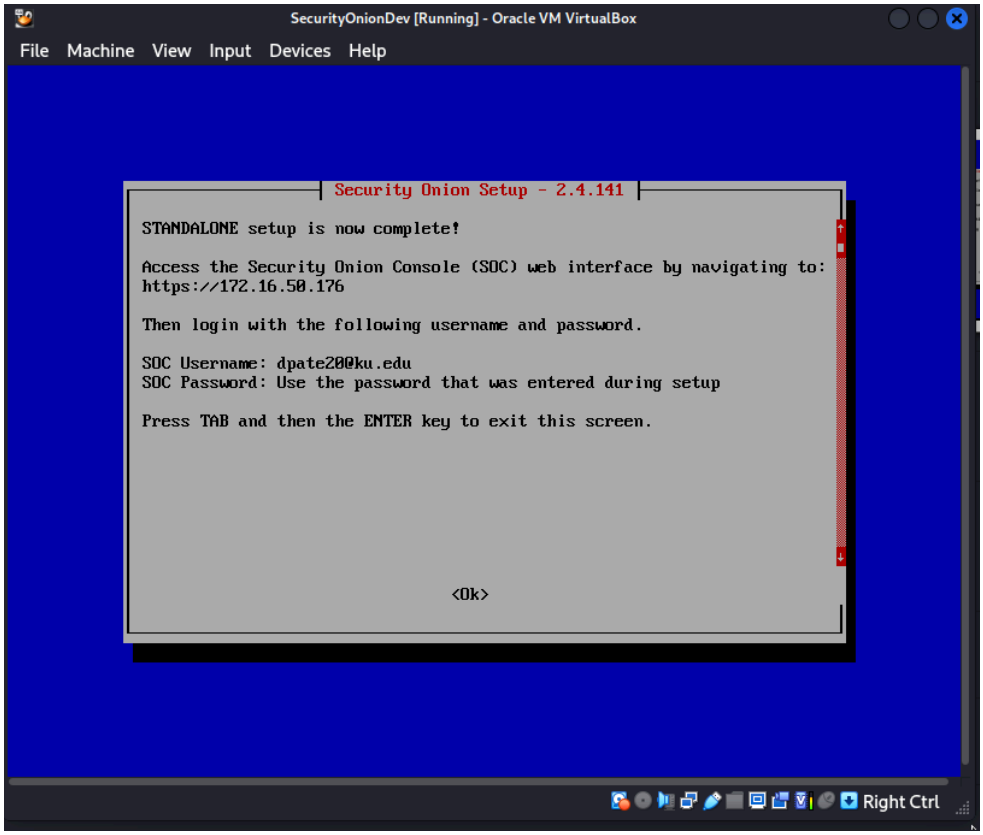
[D1.4]: Adding and setting up the NICs, one as manager and the other as monitor



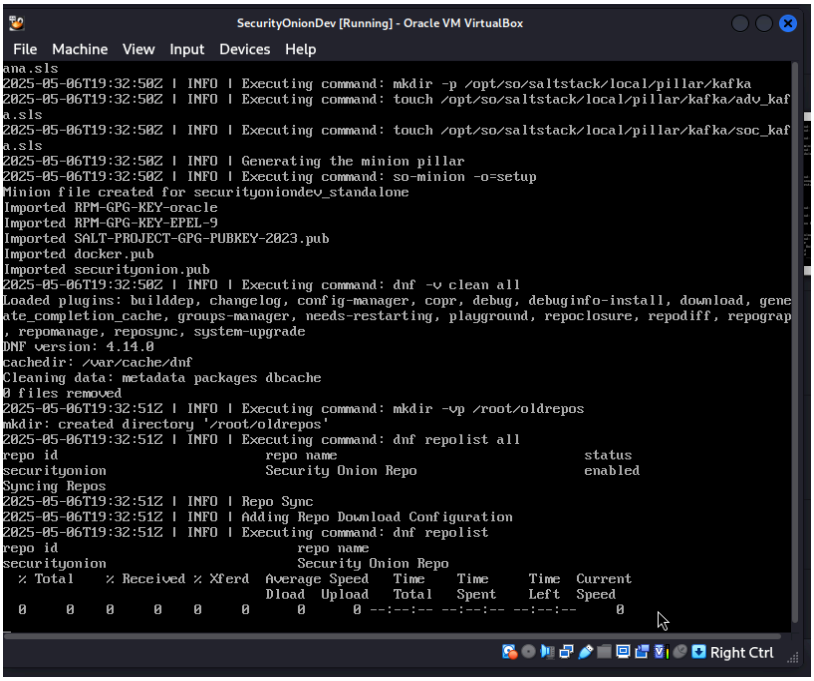
[D1.5]: Setting the IP range



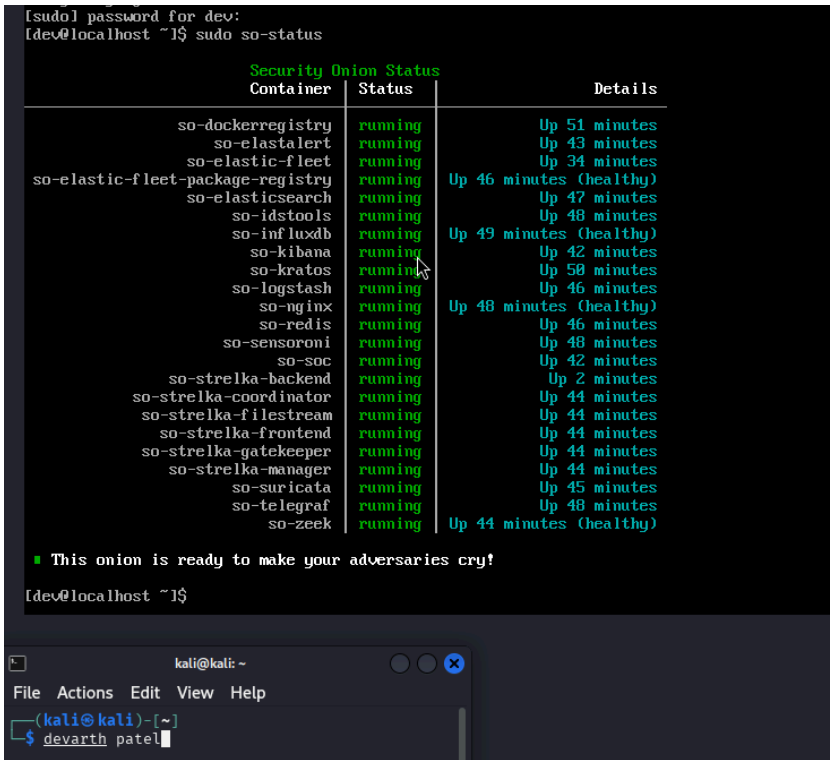
[D1.6]: Setup complete, stats



[D1.7]: Complete setup



[D1.8]: Post setup installation, painfully long



[D1.9]: Status of Security Onion, tools are all running and healthy

- **[D2]** To allow the Security Onion to receive live traffic, I created a bonded interface on my host machine. I did this using the IP command to assign an IP address to the new bonded device. This will collect the traffic from other interfaces to improve visibility and performance. After the bond was assigned, the terminal output confirmed that it was successfully attached and the bond was set to active with the correct IP configuration, ready to receive traffic. After this, I created another Kali VM with a host only network adapter so that it could communicate with the Security Onion host through the bonded network setup. This allowed the traffic from Kali VM to the Security Onion so the tools

```
[dev@localhost ~]$ sudo ip addr add 192.168.56.1/24 dev bond0
```

could interpret the live data and analyze it during testing.

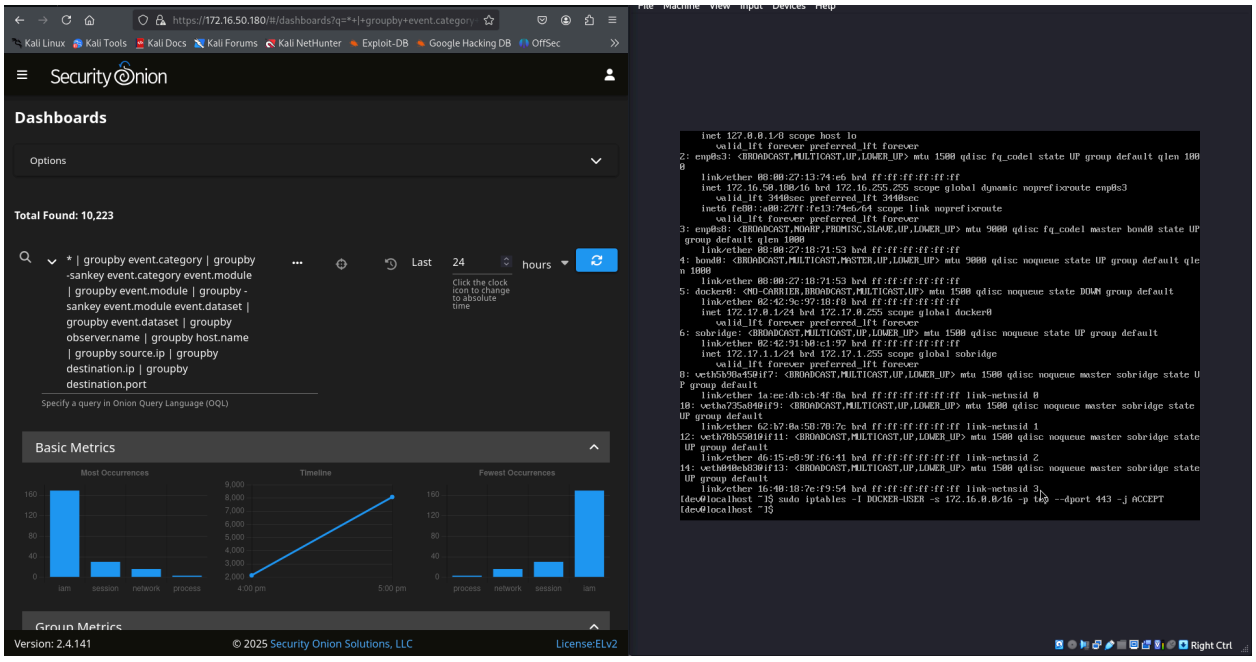
[D2.1]: Setting the bond using IP addr command to assign it

```
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
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 100
0
    link/ether 08:00:27:13:74:e6 brd ff:ff:ff:ff:ff:ff
    inet 172.16.50.180/16 brd 172.16.255.255 scope global dynamic noprefixroute enp0s3
       valid_lft 3149sec preferred_lft 3149sec
    inet6 fe80::a00:27ff:fe13:74e6/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,NOARP,PROMISC,SLAVE,UP,LOWER_UP> mtu 9000 qdisc fq_codel master bond0 state UP
 group default qlen 1000
    link/ether 08:00:27:18:71:53 brd ff:ff:ff:ff:ff:ff
4: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 9000 qdisc noqueue state UP group default qle
n 1000
    link/ether 08:00:27:18:71:53 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.1/24 scope global bond0
       valid_lft forever preferred_lft forever
5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:9c:97:18:f8 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/24 brd 172.17.0.255 scope global docker0
       valid_lft forever preferred_lft forever
6: sobridge: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:91:b0:c1:97 brd ff:ff:ff:ff:ff:ff
    inet 172.17.1.1/24 brd 172.17.1.255 scope global sobridge
       valid_lft forever preferred_lft forever
8: veth5b98a45@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master sobridge state U
P group default
    link/ether 1a:ee:db:cb:4f:8a brd ff:ff:ff:ff:ff:ff link-netnsid 0
10: vetha735a840@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master sobridge state
UP group default
    link/ether 62:b7:0a:58:78:7c brd ff:ff:ff:ff:ff:ff link-netnsid 1
12: veth78b55010@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master sobridge state
 UP group default
    link/ether d6:15:e8:9f:f6:41 brd ff:ff:ff:ff:ff:ff link-netnsid 2
```

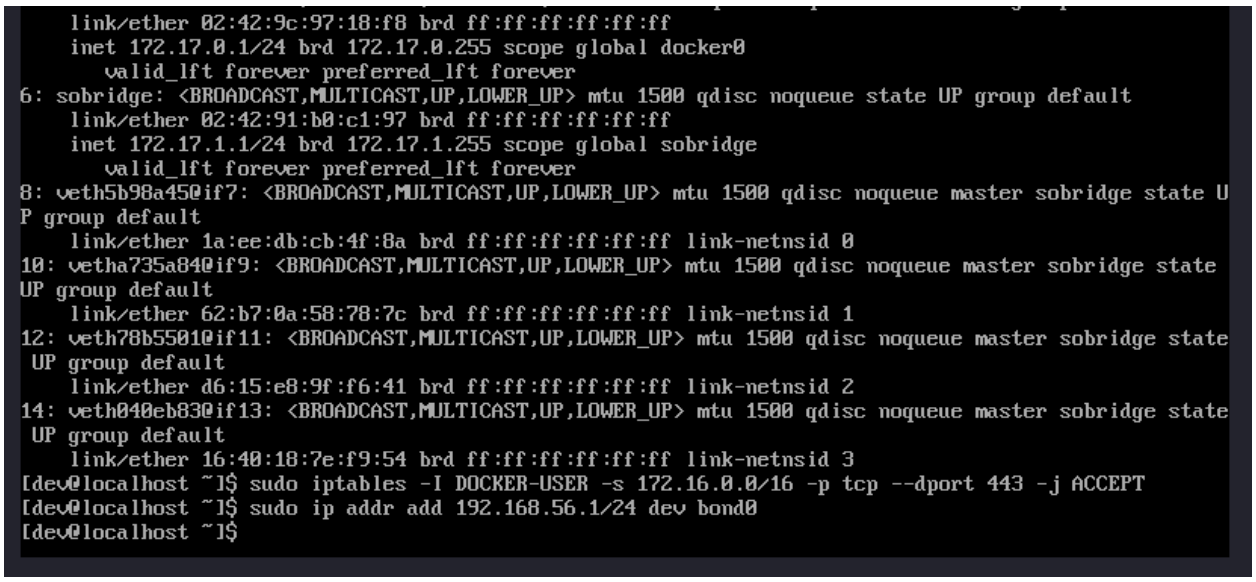
[D2.2]: Confirming that the bond is set and UP and running

[D2.3]: Setting up the Kali VM to send traffic to Security Onion

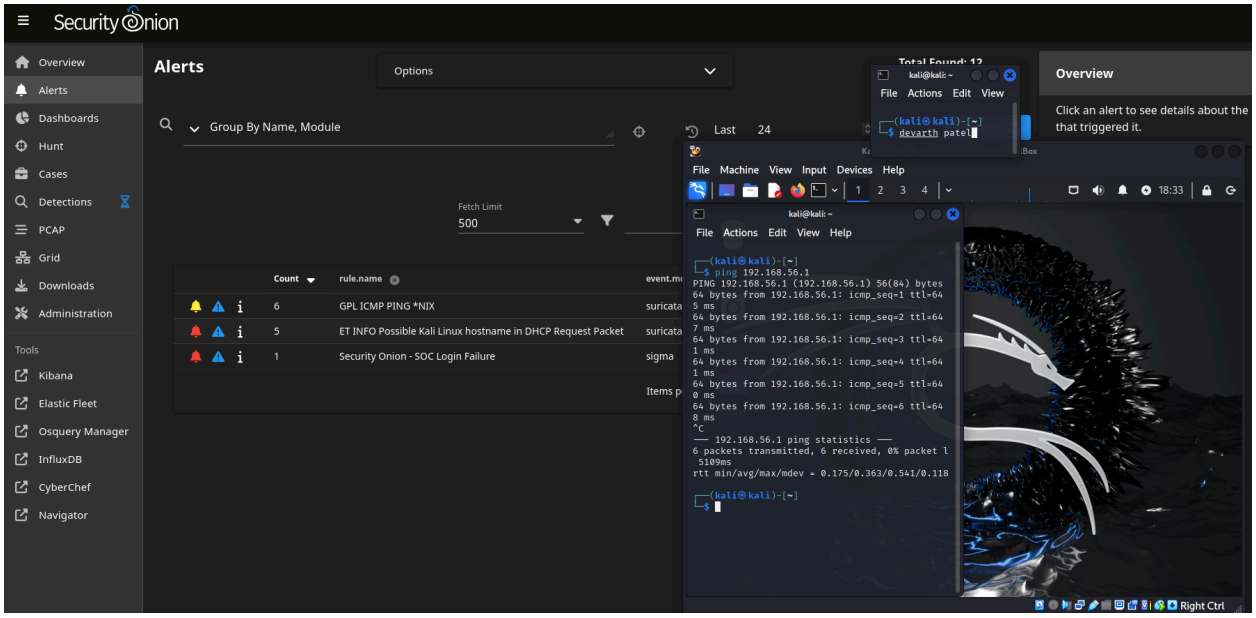
- **[D3]** Once I had everything setup, I wanted to test my Security Onion. I logged into the online user interface by entering the IP set and accessed the home dashboard to see the logs, tools and graphical event data. Sometimes I encountered errors for loading up the interface due to docker level restrictions, so I added iptables rules to the Docker user chain to allow any inbound TCP connections on port 443 from my subnet. After doing this, my Onion dashboard would load smoothly. To test if the tools were capturing and analyzing traffic correctly, I used the secondary Kali VM to simulate activity. I sent ICMP traffic, pinged the Onion, from my Kali VM to the bonded interface. After this, Suricata successfully identified this and flagged it under its protocols. It also picked up my DHCP hostname request and also my first login failure. This was confirmation that the traffic activity was being processed and logged. I then used nmap and hping3 to simulate port scanning and traffic flooding which generated more packet activity.



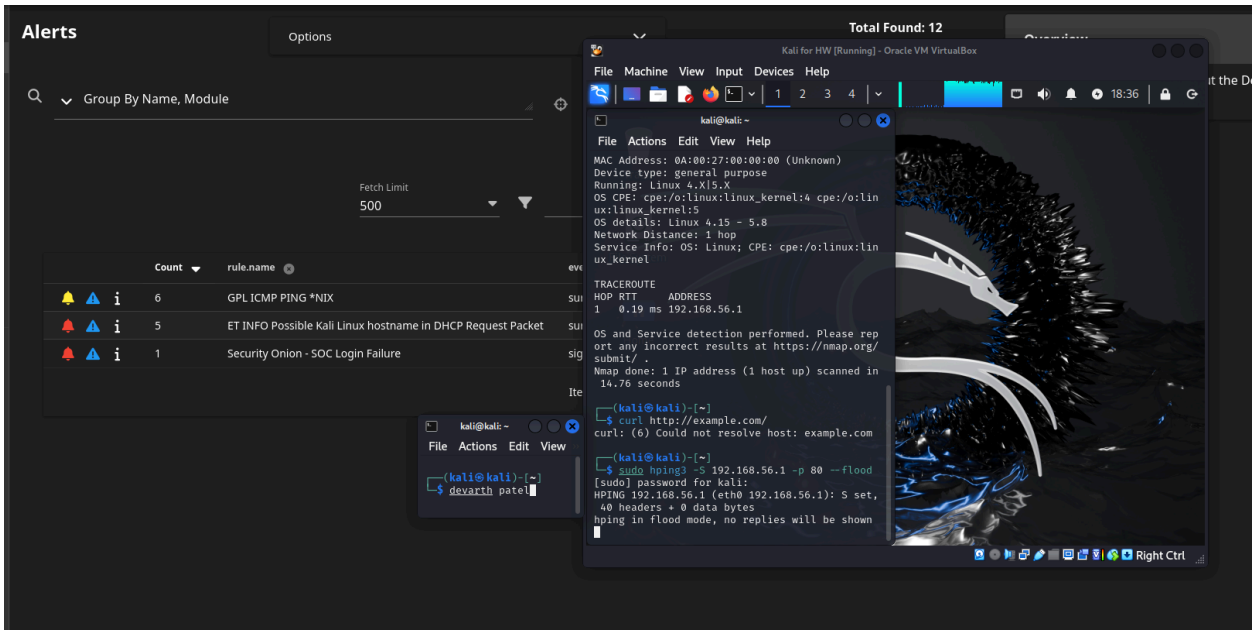
[D3.1]: Accessing the dashboard after using the IP tables rules command and logging into UI



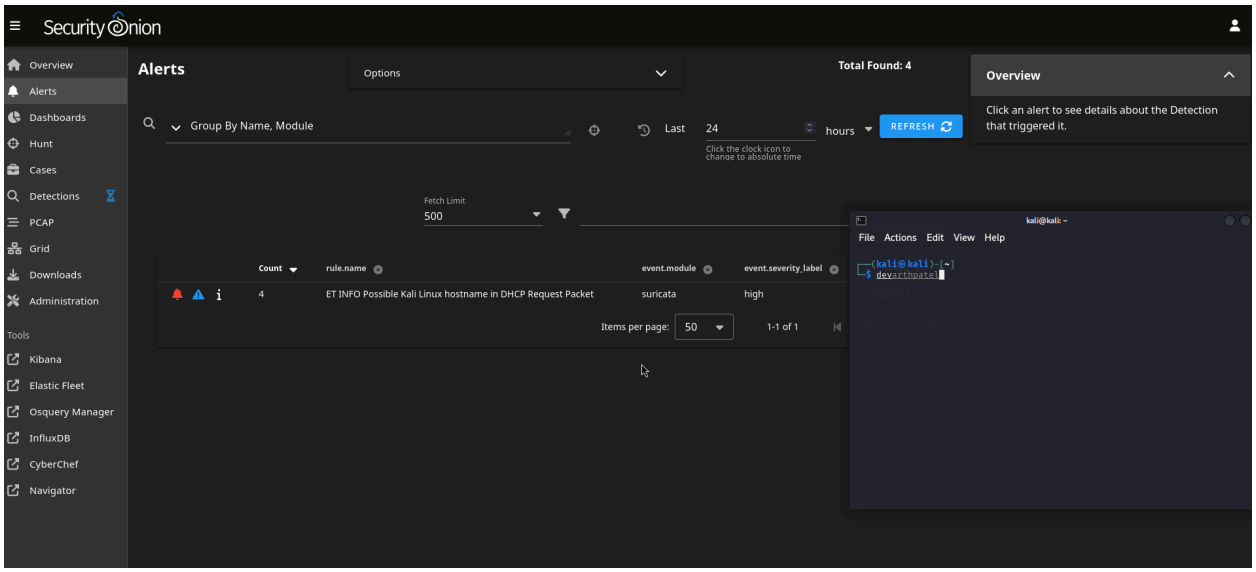
[D3.2]: A closer look at the terminal command to process the IPtables.



[D3.3]: Sending ICMP traffic, 6 pings to the bonded device for detection

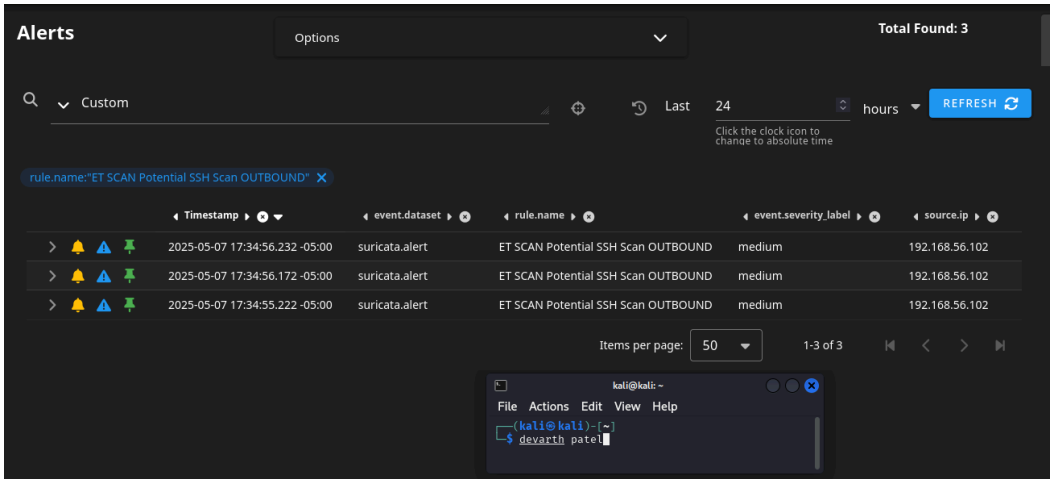


[D3.4]: Nmap and Hping requests sent



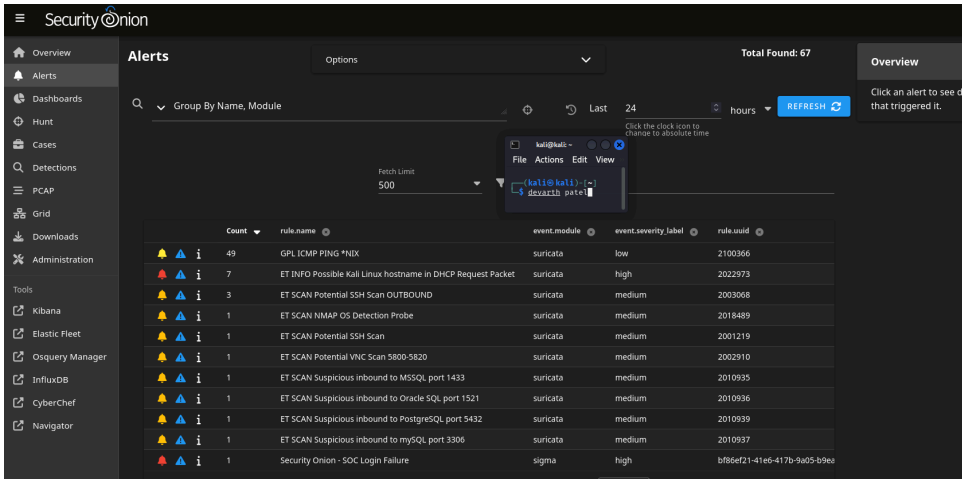
[D3.5]: DHCP request

- **[D4]** Finally I was able to look at the event data for the traffic generated within the Suricata and Zeek tool filters. I could see that Suricata and Zeek worked well together and provided a layered network visibility with signature based alerts and behavioral analysis of the logs. When I generated traffic to the bonded device from my Kali VM, Suricata picked up several alerts including those in **[D3].** While Suricata raised alerts, Zeek logged the data for deeper analysis and saved the connection logs (zeek.conn) where it captured all TCP connection attempts. Together they both showed a powerful

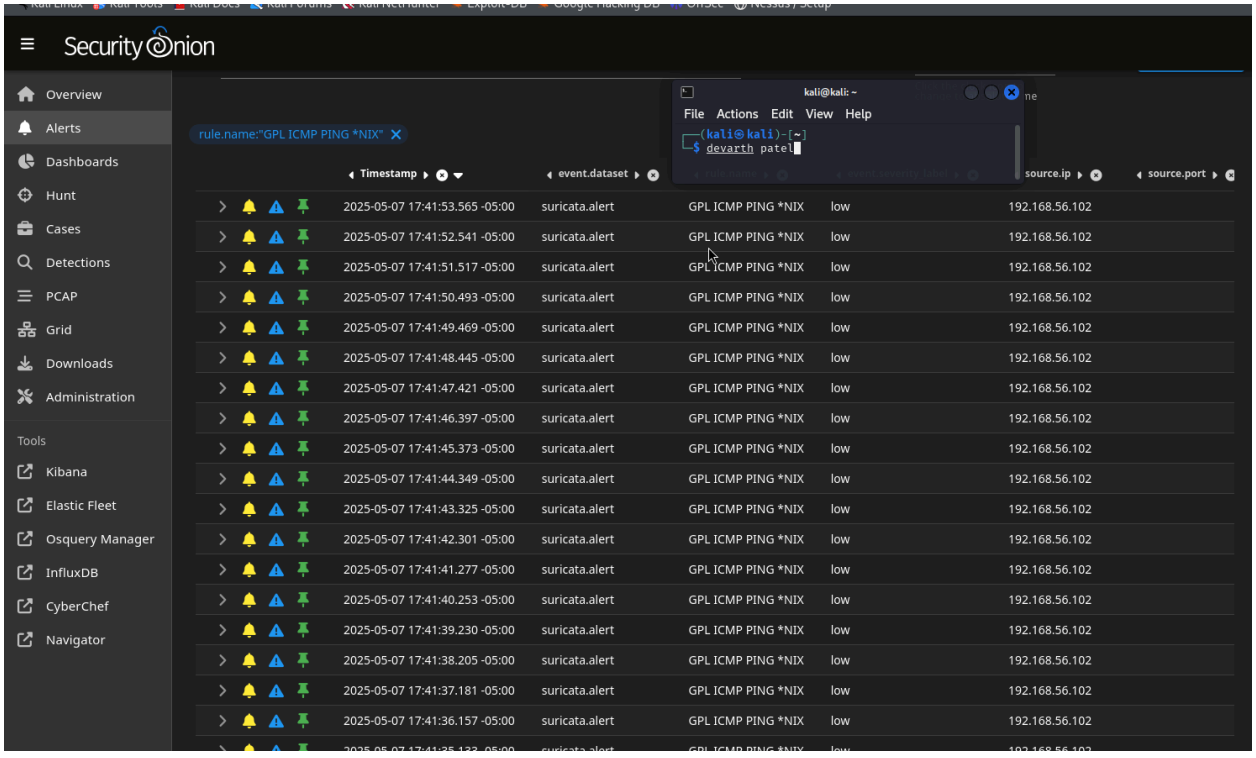


demonstration of Security Onion's capabilities.

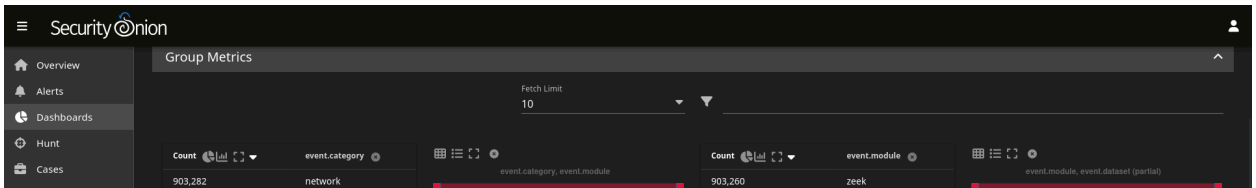
[D4.1]: Port Scan Alerts



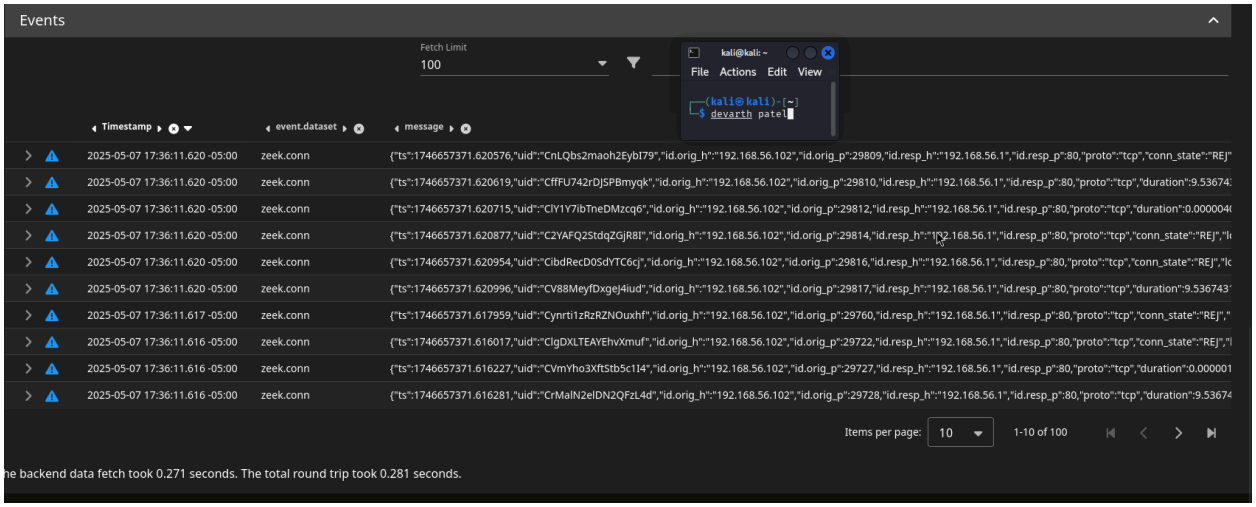
[D4.2]: Scan Alerts



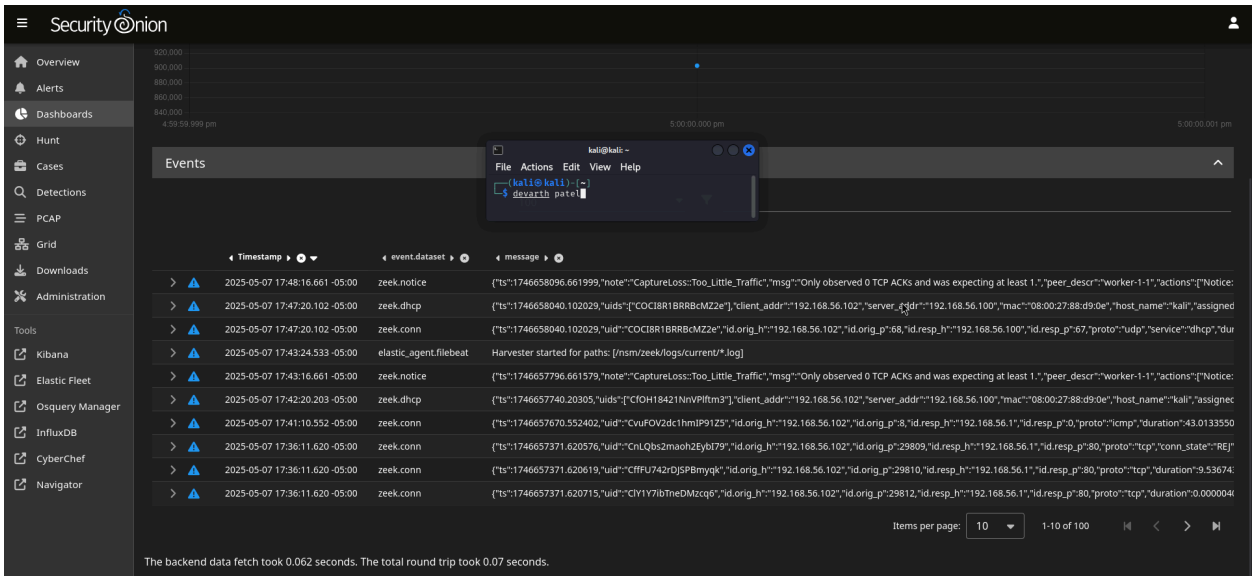
[D4.3]: Ping alert



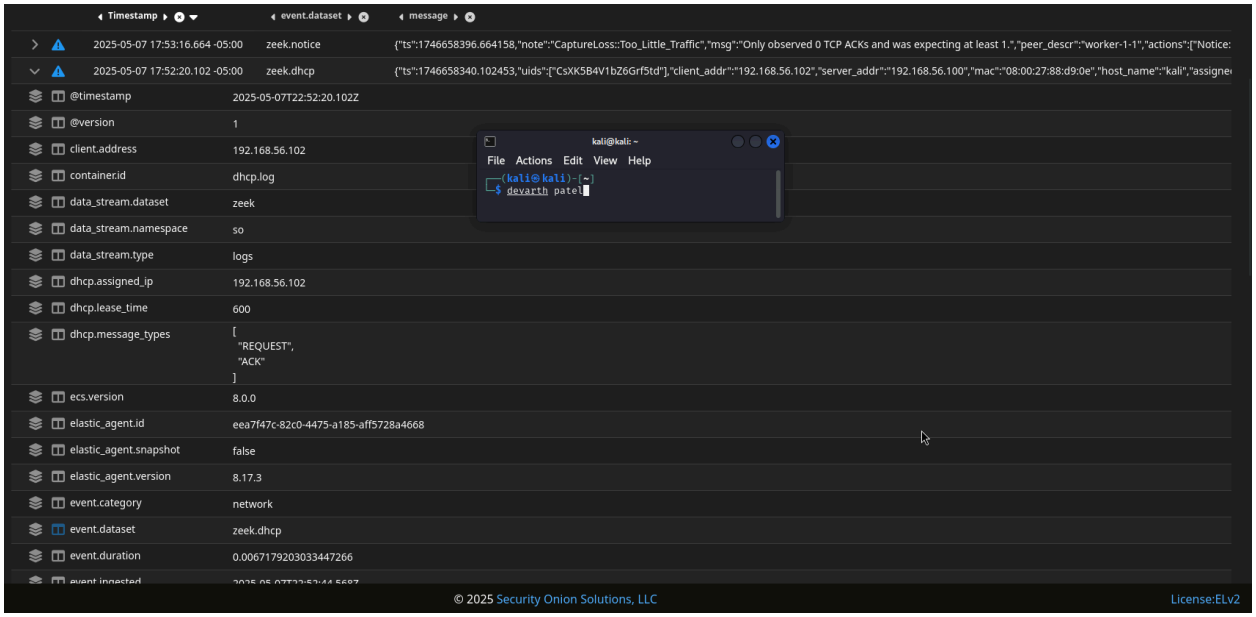
[D4.4]: Zeek dashboard



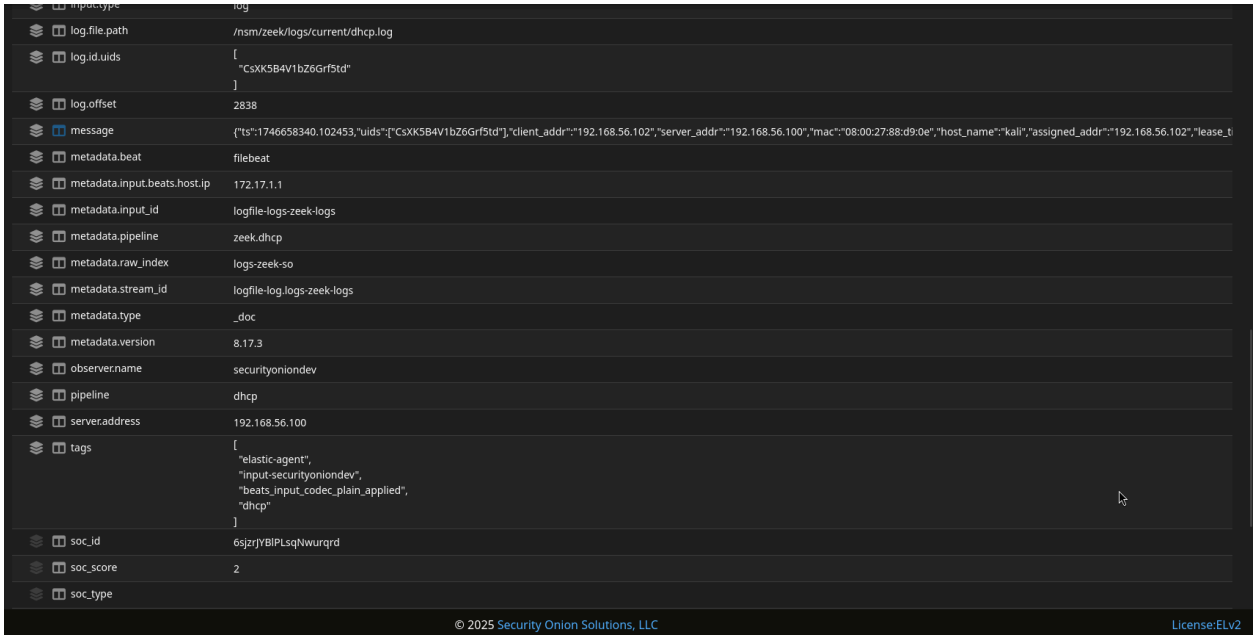
[D4.5]: Zeek logs for (zeek.conn)



[D4.5]: Zeek events



[D4.6.1]: Zeek logs drilldown 1



[D4.6.2]: Zeek event drilldown shows us in depth of everything

Observations

During the testing, traffic was sent from my secondary Kali Linux virtual machine towards the Security Onion machine (installed in standalone mode). The Security Onion VM was monitored through its web interface to make it easier to visualize the data. Traffic included standard pings, flood pings (hping3), and targeted port scans that I sent using Nmap. Once it was running, Security Onion's alert interface was filled with Suricata detections, flagging numerous ICMP pings and DHCP activity with the Kali hostname displayed in the request. Also, outbound SSH scan attempts also triggered the alerts, suggesting that Suricata was active and responsive to both the low-level and high-level scan behavior. While this was processing, Zeek logged the traffic metadata in real time — documenting all rejected TCP connections, active lease information through the DHCP, and generated some notice logs related to the traffic anomalies. All the traffic activities were tied back to the secondary Kali machine and was shown in the logs across both Suricata and Zeek. Throughout testing, the system responsiveness stayed stable and responsive, and the traffic flowed successfully through the bonded network interface.

Analysis

This activity shows a successful multi-tool collaboration between Suricata and Zeek within Security Onion. Suricata's biggest positives lie the rule-based signature detections which was seen when it triggered alerts such as the ping alerts, DHCP request alerts, and multiple scan flags. This indicates that the system was able to detect not only ICMP and DHCP traffic but also

identify the patterns common to the reconnaissance behavior. Each alert included the source IPs, port information, and severity levels of the incoming traffic. To complement that, Zeek acted as an observer, creating deep, explorable logs that captured the full context of each connection that passed through in the traffic. The conn logs delivered multiple rejected TCP attempts to port 80 and other ports, which were also seen in the Suricata alerts. Zeek's logs also recorded the lease times, MAC addresses, and even displayed the hostname as “Kali”. Together, these interpretations gave us layered perspectives: Suricata showing us the what, and Zeek giving us the how and when, allowing us to deeply understand the traffic.

Conclusion

From the testing results, it can be concluded that Security Onion’s integrated design, allowing users to engage Suricata and Zeek in collaboration, allows for detailed forensic abilities. Suricata captured the real-time threats and all the reconnaissance attempts while Zeek gave us detailed logs to support deeper analysis. The test traffic from the secondary Kali machine, the pings, flood scans, and SSH probing was properly captured, logged, and displayed on the Security Onion dashboards. This provides full validation that Security Onion is an ideal solution for monitoring active threats and subtle anomalies, allowing for great forensic analysis. By analyzing both of the Suricata notifications and Zeek logs together, it creates a clear picture of the conditions surrounding traffic.

In an active deployment of Security Onion, this functionality would let professionals move quickly from alerts to investigation, reducing time spent and improving on the incident response time. Overall, the system demonstrated its capacity to detect, log, and contextualize the suspicious traffic with precision.

References

Security Onion Solutions. (2024). Security Onion documentation (Version 2.4). https://docs.securityonion.net/

OpenAI. (2024). ChatGPT (Version 4). https://chat.openai.com/

ResearchGate. (n.d.). ResearchGate. https://www.researchgate.net/