

Detecting Residential Proxy Operations in Network Traffic

Zurich University of Applied Sciences
School of Engineering

Authors	S. Selviyani
Supervisors	Dr. Peter Heinrich
Study Program	InfoCyberSec
Submitted on	Autumn Semester 2025

1 Introduction

Residential Proxies (RESIP) have become a common topic on the Internet. These proxies provide access to use someone else's network with or without the user knowing. There are several purposes for using a residential proxy, including the ones listed below [1]:

- Bypassing geo-fencing restriction
- Data extraction for market monitoring and competitive evaluation
- Data extraction for the development of extensive language models in Artificial Intelligence
- Violation of copyrights
- Advertising fraud and click fraud
- Avoiding anti-fraud mechanisms of e-commerce platforms and financial institutions
- Techniques for password and credential spraying
- Management of social media accounts, propagation of disinformation on social media
- Anonymization layers, such as Operational Relay Boxes (ORB) [2] used by Advanced Persistent Threat (APT) organizations

Residential proxies are often installed on the devices of legitimate users, including computers, tablets, smartphones, routers, IoT devices (smart TVs etc.), allowing the utilization of their IP addresses. Internet traffic coming from IP addresses includes both legitimate and proxy traffic, making it difficult to identify malicious activity [3].

This project addresses the challenge of detecting residential proxy traffic from within the network, specifically traffic that comes from an infected host inside the network that is being used as an exit node for external attackers. This is a challenging task due to the fact that the traffic is generated from a legitimate internal IP address, that is hard to distinguish, so it is difficult to determine whether it is the attacker or the victim who is running the suspicious activity.

The purpose of this experiment is to develop packet and flow analysis that evaluates packet and flow metadata from firewall traffic in order to detect anomalies that indicate proxy behavior. As a prototype, we have implemented these detection mechanisms with plugins or rules for the OPNsense firewall using the Suricata IDS/IPS engine and NetFlow.

2 Technical Background

2.1 Proxy vs. VPN

Proxy servers operate as an extension and gateway between clients and servers on the Internet. Proxies have many beneficial applications focused on protecting users' privacy, such as bypassing server-side blocking and regional limitations. Although the are benefits to proxies, adversaries can use them to cover up their identities and launch numerous attacks. As a result, several websites restrict proxy access, leading to the creation of blacklists to filter out and ban known proxy servers [4].

With this intention, proxy servers represent one of the most straightforward methods for users to protect their privacy by hiding their actual Internet Protocol (IP) addresses [4]. Proxy servers act as service providers that facilitate online communication between client and Internet services.

By using a proxy, the client can avoid sending their requests directly to the remote server,

such as a web server. After receiving a request from a client for a specific resource, the proxy initially conducts a search within its internal cache first. If the resource is located in the cache, it is then returned to the client. In case that the requested information is not located in internal cache, the proxy will relay the request to the server to get response, which is subsequently returned to the client [4]. The caching role of proxy minimizes the necessity for direct interaction between clients and remote servers, therefore preventing network bottlenecks.

A Virtual Private Network (VPN) establishes a secure tunnel for the transmission of network packets. Similar to a proxy server, it can hide our IP address in turn to protecting our online identity on the Internet and enables us to use the network securely [5]. VPNs operate at the operating system level, effectively rerouting all traffic, regardless of whether it originates from a browser or an application. Common VPN protocols encrypt the data during transmission between the device and the server via the Internet. This encryption ensures that our Internet Service Provider (ISP) cannot track our online activities, they only see that we connected to a VPN server [5].

The primary distinctions between proxy servers and VPNs are in their features. While VPNs offer encryption for all data transmitted, which improves overall security, especially during sensitive transactions, [5]. On the other hand, proxy servers usually have faster performance compared to VPNs. For the IP address which is visible to the server in the target connection, this does not make a difference.

As we mention before, VPNs work at the operating system level, redirecting all traffic through a VPN server and encrypting all online activities, regardless of whether a browser or an application is in use. This provides a broad reach compared to proxy, which operate at the application level and only reroute the traffic of chosen by the app or browser [5]. Moreover, many VPN providers implement a no-log policy, meaning that they do not monitor or retain web activity. This is not consistently true for proxy servers, thus offering VPNs a more interesting choice for individuals who prioritize their only privacy. Proxies are particularly effective for certain applications, including large-scale web scraping and the management of multiple profiles, as they can perform these function efficiently without requiring advanced encryption methods [5].

2.2 Residential Proxy Threat Model

Residential Proxy (RESIP) is a networking proxy setup where traffic is tunneled through end-user devices and their *residential IP addresses* for the purpose of bypassing server-side blocking [6]. As stated in the introduction, a residential proxy has been reported as a tool to camouflage criminal activities. This discussion started in 2019 when Mi et al. bring up *the residential proxy as dark service* topic as their research paper [6].

RESIP providers claim to procure residential proxies ethically, by the user agreeing the installation of the Potentially Unwanted Program (PUP) as written in their Terms and Conditions. However, the lack of regulation allows for varying definitions of *ethical procurement*. Some providers inform their users to share their IP address with the proxy providers in return for some financial benefits (e.g 0.5 USD/GB traffic [7]). Of course, these companies sound legitimate. They use the terms “market search” or “ad verification” [8] to describe their

intended operations, which are carried out over proxy links. However, this is not necessarily true. Industry reports and academic research have shown that free VPN apps for example can carry a RESIP component to recruit unsuspecting users into the RESIP network. Additionally, mobile phone apps, hacked routers, and IoT devices can serve as payload delivery vehicles [9].

To illustrate this illegal activity, we can consider the residential proxy victim, who was not aware of the presence of a PUP program on their device. This poses significant risks, as it can be exploited by attackers who have access to the victim or infected machine network. The attacker may use the victim's machine as a relay point or gateway for their malicious traffic. Rather than the attacker establishing a direct connection from their own computer to the target server, the attacker routes their connection through the victim-proxy machine to access the outside world. Therefore, the malicious traffic appears to originate from the network of the proxy victim or infected machine, rather than exposing the attacker's real IP address.

When the machine got infected from the attacker injecting a residential proxy, the attacker may exploit the proxy victim's machine to hide his identity during an attack. This means that the attacker can use the proxy victim's Internet connection, hiding their own IP addresses, to execute attacks on other targets. This may involve unauthorized access attempts, spreading spam, hacking websites, or even performing distributed denial-of-service (DDoS) attacks. This situation is unfortunate for the proxy victim, as all the blame is directed towards him, with all evidence pointing to his IP address. This can also bring reputational damage for the proxy victim and their ISP, as they could potentially be blacklisted as a source of malicious activity. Furthermore, the proxy victim might face legal consequences for actions of which he was unknowing. In addition to that, the proxy victim may act as a participant in a botnet, meaning that their computer becomes a foot soldier in the attacker's network (the botnet). It might consume the proxy victim's machine resources such as CPU, memory, bandwidth to serve the attacker's goals.

2.3 Existing Proxy Detection Techniques

According to the article written by Leonidas Gorgo on his Medium site, there are existed various common techniques for detecting proxies [10]. The following detections are included.

2.3.1 IP Address Analysis

This technique involves comparing IP addresses against a list of known proxy server IPs, which may include open proxies, VPNs, or Tor exit nodes. This can be carried through third-party services or by maintaining a private database. Services such as MaxMind [11] and IP2Proxy [12] provide a proxy database that can be searched by websites.

The second is to blacklist. We can do a blacklist of repeated violations or suspicious activity from specific IPs addresses, which can result in their classification as proxy addresses on a blacklist.

The third is geolocation mismatch. So when the geolocation linked to an IP address, as determined by public databases, is not consistent with the location reported by the user, the IP maybe subject to further examination, as this difference might indicate the use of a proxy.

Last, the IP reputation, as we know that some systems evaluate the historical behavior of an IP address. So when there is an IP address associated with illicit activities, such as spamming or cyberattacks, for example, it may be identified as a proxy too [10].

2.3.2 Reverse DNS Lookups

Proxy detection tools often conduct reverse DNS lookups on IP addresses to find out if the host is linked to a recognized proxy or VPN service. The presence of a reverse DNS entry indicating that the IP is associated with a data center rather than a home ISP may suggest the use of a proxy [10].

2.3.3 Latency Assessments (with Ping and Traceroute)

Certain detection tools evaluate the latency between the server and the client. An unusually high response time for the user's stated location can indicate the use of a proxy. As their traffic is being routed through an alternative server, it is causing delays in responses. In addition, through the traceroute tool, it can examinate the hops that a packet passes between servers and check whether the system is able to figure out whether the traffic is being directed through an intermediary proxy server or not [10].

2.3.4 HTTP Header Analysis

The technique of analyzing HTTP headers involves identifying the HTTP headers and finding the end user's IP address through their standard proxy headers. This solution is widely used and has been implemented to mitigate IP spoofing, even in specialized security devices like the CISCO IronPort Web Security Appliance [13]. The most commonly used headers to indicate the IP address of a proxy's end client are:

- X-Forwarded-For: This HTTP header field acts as a typical method for identifying the originating IP address of a client that connects to a web server via a load balancer or HTTP proxy. The originating IP address may be hidden during the server connection phase. Therefore, this approach is only reliable when used with trusted servers [13]. Nonetheless, a proxy server can include particular headers such as X-Forwarded-For, Forwarded, or Via in the HTTP request, which may reveal the original client IP address that is situated behind the proxy [10].
- User-Agent anomalies: Certain proxy or VPN have the ability to modify HTTP headers, including the User-Agent. The modification in these headers, such as unusual or often related proxy signatures, can indicate the use of a proxy [10].
- Request fingerprinting: This involves comprehensive analysis of various components of an HTTP request, including headers and protocols, to identify and compare them with established patterns linked to proxy or anonymization services [10].

2.3.5 Device Fingerprinting

A device fingerprint, or machine fingerprint, refers to the data gathered regarding the software and hardware characteristics of a remote computing device to establish its identity. This information is processed into an easily understood identifier using a fingerprinting algorithm. A common example is the browser fingerprint which depends on information gathered during interactions with the device's web browser [14]. Through the collection of data regarding a user's device, browser, and connection, device fingerprinting method analyze this information against predicted values. Should a user's device show signs of hiding its identity, such as

displaying inconsistent time zones, languages, or system attributes, with this reason may indicate the use of a proxy [10].

2.3.6 WebRTC Leak Detection

WebRTC, also known as Web Real-Time Communication, has the potential to expose a user's actual IP address, even when using a proxy service. Websites have the capability to use JavaScript in order to initiate WebRTC requests, which may reveal the user's actual IP address [10].

2.3.7 TLS Fingerprinting

TLS fingerprinting involves analyzing the TLS/SSL handshake processes initiated by various clients like browsers, proxy services, and VPNs. Each client demonstrates unique patterns during the process of connection establishment, shaped by elements such as the supported cipher suites and the particular extension used. This analysis allows systems to ascertain the origin of the traffic, differentiating between authentic user connections and those originating from proxy services [10].

2.3.8 Machine Learning Models

Advanced proxy detection services leverage machine learning to examine extensive datasets and recognize patterns indicative of proxy usage. These systems have the ability to develop overtime, allowing them to learn from emerging proxy types and innovative anonymization methods. This adaptability is crucial for maintaining effectiveness against advanced evasion techniques [10].

2.3.9 Behavioral Honeypots

A honeypot is a defensive cybersecurity strategy that acts as a counterintelligence asset rather than just a direct approach to mitigating vulnerabilities. This method involves deliberately showing a system, service, or application to replicate genuine targets, thus attracting unauthorized individuals to participate. The primary goal of a honeypot is to collect data, encompassing the observation of attacker behavior, their methods of exploitation, the toolkits they used, and any actions taken after an attack occurs. This method facilitates the analysis of cyber threats while ensuring the integrity of production systems stays unaltered [15].

2.3.10 Deep Packet Inspection (DPI)

Deep Packet Inspection, also referred to as packet sniffing, is an advanced technique for inspecting the content of data packets at a network checkpoint (e.g. firewall). In contrast to conventional stateful packet headers such as destination and source IP addresses along with TCP/UDP port numbers, DPI expands its scope to include a wider range of metadata and payload linked to each packet. This comprehensive evaluation allows DPI to deliver superior network packet filtering capabilities. Furthermore, DPI demonstrates proficiency in identifying hidden threats within data streams, including attempts at data excitation, violations of policy and the presence of malware [16]. This method is applicable for identifying patterns that align with the use of proxies or VPNs, including encrypted patterns that align with the use of proxies or VPNs, including encrypted traffic patterns that suggest the presence of tunneling protocols such as OpenVPN or IPSec.

2.3.11 Behavioral Analysis

With Captcha challenge: In recent years, websites may implement a CAPTCHA challenge to assess whether the incoming traffic originates from a human or a bot operating through a proxy. Automated scripts often run into problems with overcoming these challenges, which helps in distinguishing between legitimate and suspicious users [10].

With traffic patterns: The proxy users might show distinct behaviors in contrast to typical users, such as rapidly accessing numerous pages and using multiple accounts from a single IP address. Reviewing the traffic behavior may reveal the utilization of proxies [10].

We might implement behavioral analysis to our threat simulation. We tried to analyze the behavior of the attacker and victim machine when the residential proxy occur by packet corellation analysis.

2.4 Technology Overview

There are several tools that we used for our experiment. To create a simulation of a residential proxy used by an attacker to access a target, we utilized the following tools.

2.4.1 Oracle VirtualBox

Oracle VirtualBox is a multifunctional virtualization application that operates across various platforms, allowing a computer to run multiple operating systems at once within virtual machines (VMs). Users have the capability to operate multiple operating systems configurations. The main limitation of the VMs are disk space and memory, which facilitate the installation of multiple virtual machines. Virtual machines possess the capability to be effortlessly saved, copied, exported and transferred across hosts [17].

Oracle VirtualBox also has other advantages over another virtualization tool such as QEMU (UTM),the snapshot feature that Oracle VirtualBox offers is so useful for our threat simulation.

2.4.2 Debian 13

Debian 13 was released on September 6th 2025, and is designated with the codename *Debian Trixie*. In this experiment, we used the Debian operating system as host as well as VM guest. It is essential to recognize that we will require a minimum of three virtual machines to act as the actors (attacker, proxy victim and target server) in our threat simulation.

2.4.3 OPNsense

OPNsense is an open-source firewall and router software based on FreeBSD. It is developed by the Dutch company Deciso, which additionally produces hardware and offers support packets for OPNsense. Launched in January 2015, OPNsense represents an improved version of pfSense, which is itself as an extension of the original m0n0wall project. This initiative aimed to create a firewall solution that is compatible with embedded PCs and old hardware, and it is defined by an easy-to-use web interface driven by PHP. Even though m0n0wall mainly focused on Layer 3 and Layer 5 firewalling, there was a growing demand among network and security administrators for advanced features like web proxying and intrusion detection, which were included in commercial firewalls as components of Unified Threat Management (UTM) solutions [18]. We are operating OPNsense as another VM in our VirtualBox setup.

2.4.4 Suricata

Suricata is a free and open-source network intrusion detection system (IDS) and intrusion prevention system (IPS) that provides a high-performance and scalable approach to detecting and blocking various types of malicious network traffic, including malware, exploits, and denial-of-service attacks [18]. Suricata is a plugin that is already provided as a service by OPNsense. No additional installation is required, and we just effortlessly use the service.

Suricata is defined by its exceptional performance and scalability, a broad range of detection capabilities that include malware, exploits, and denial-of-service attacks, flexible configuration options, and compatibility with diverse rule sets. Furthermore, Suricata integrates well with other security tools, such as firewall and Security Information and Event Management (SIEM) systems. However, it is important for users to be aware that Suricata might need substantial resources, thus requiring sufficient hardware for optimal performance [18].

In our experiment, we use custom Suricata rules to implement proxy detection.

2.4.5 Netflow and Akvorado

NetFlow is a monitoring feature invented by Cisco and implemented into the FreeBSD kernel with Netgraph. NetFlow represents the complete capture of packet flows, including both source and destination IP addresses along with their assigned port numbers. OPNsense supports NetFlow as one of its features. OPNsense provides full support for exporting NetFlow data to external collectors, along with an included analyzer for on-device analysis and real-time monitoring. NetFlow can be a useful instrument for the analysis of flow data, including both ingress and egress traffic. When both ingress and egress are enabled, traffic is counted twice because of Network Address Translation.[19].

Akvorado is an open-source program that processes flows from NetFlow, then enriches them with metadata names through the use of Simple Network Management Protocol (SNMP) and geographical information. After that, Akvorado is storing the NetFlow data to ClickHouse, enabling it to be presented in a human-readable format through web interfaces for browsing the collected data [20]. SQL-like queries can be used to analyze the collected data.

In our experiment, we use NetFlow and Akvorado for a stateful approach of proxy detection.

3 Methodology and Threat Simulation

3.1 Proposed Detection Methodology

Numerous methodologies exist for the (attempted) detection of residential proxies in prior research. The researchers primarily analyzed providers offering residential proxy services to determine which countries or regions have the highest availability of residential proxy IP addresses. However, examining the providers that offer residential proxy services will not help us to discover and detect which of our own machines may have been infected with Potentially Unwanted Programs (PUPs) which a residential proxy provider might be using without our knowledge in exploit our machines and IP addresses for residential proxies.

This experiment employs a distinct purpose and methodology compared to others. We attempt a detection method by closely analyzing the network traffic and its corresponding packets which are actively transfered within our network and our machines. By analyzing

the packets and its correlation, we may identify patterns, especially if a residential proxy has been installed in one of our machines. In addition, IoT devices in our network could be exploited as residential proxies.

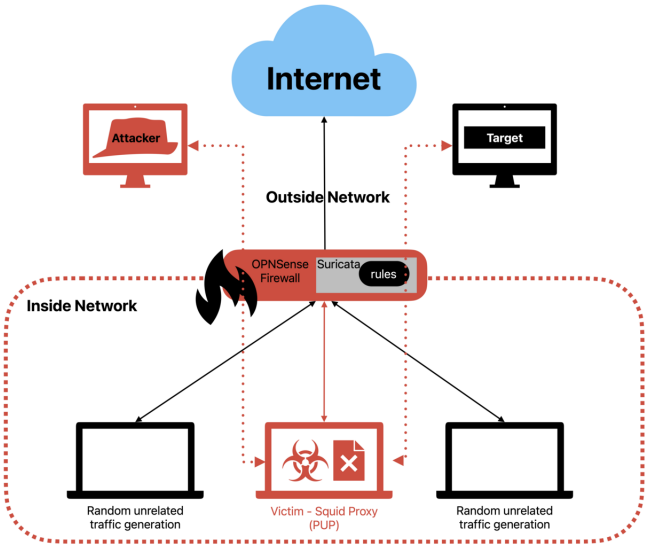


Figure 3.1: Residential Proxy Illustration

For this reason, we attempt to implement our detection with firewall rules. In our lab setup, we will make use of OPNsense. OPNsense allows to implement detection rules with Suricata. Suricata is a popular open source solution for intrusion detection (IDS) and intrusion prevention (IPS). It allows us to implement our custom rules for the detection of proxy communication.

The detection is very challenging to implement for multiple reasons. Firstly, we have to distinguish attacker traffic from legitimate traffic. For example, a legitimate user in our network might access a website with his webbrowser. On the other hand, an attacker could access the same website using a residential proxy in our network. Secondly, most communication over the Internet is encrypted with secure protocols such as TLS and SSH. This means that we have only very limited access to the communication contents.

3.1.1 Static Detection

The simplest strategy to detect unwanted traffic is static detection. We can analyze packets for their TCP port number or protocol metadata. For instance, the popular open source proxy *Squid Proxy* is listening on TCP port 3128 per default. However, an attacker can easily avoid this detection by using a different port.

Another idea is looking at protocol metadata such has HTTP methods and headers. The `CONNECT` method is by HTTP clients to connect to an HTTP proxy. In addition, there are typical HTTP headers used in proxy connections such as `Proxy-Authorization` and `Proxy-Connection`. This deep-packet inspection works because HTTP proxy connections are typically not encrypted, even when the connection to the target website is.

3.1.2 Dynamic Detection

A more sophisticated strategy would be based on packet correlation analysis. In particular, we can examine the specifics of the packets entering and exiting our network. By carefully studying these packets, we can identify correlations between incoming connections to the residential proxy and outgoing connections to the target server.

As before, if the connection is encrypted with TLS (HTTPS) or SSH, it is not feasible to closely examine the details of the payload. However, we can still observe the source and destination, as well as the timing patterns of the packet and the size of the transferred data.

3.2 Proposed Intrusion Detection Rules

As a result of the discussion above, we can categorize our intrusion detection rules into three classes.

3.2.1 TCP/IP Inspection

On the TCP/IP level, we can use source and destination IP addresses and ports. In addition with TCP SYN and ACK messages, this allows us to track connections. In addition, we can use common port numbers to guess used protocols, for instance TCP ports 3128 and 8080 for HTTP proxies.

In a larger company network, we could use additional information to detect unexpected and undesired traffic. Assuming that the company has a well organized network with different subnets for different purposes (e.g. office network, IoT network and server zone), we could sharpen our rules. For example, SSH from the Internet to a Smart TV in a meeting room would be considered unexpected.

3.2.2 Deep Packet Inspection

Even though we expect encrypted connections, *Deep Packet Inspection* might still be useful. For example, it allows us to detect the `HTTP CONNECT` method and suspicious HTTP headers such as `Proxy-Authorization` and `Proxy-Connection`. It is also possible to gain metadata about encrypted connections through their connection establishment, e.g. TLS Server Hello and Client Hello or SSH key exchange.

3.2.3 Traffic Flow Correlation

The biggest limitation of the Suricata approach is that the rules are always stateless. This means that we cannot correlate packets from the attacker to the victim machines with packets from the victim machine to the target server. To implement correlation of incoming and outgoing packets through the OPNsense, we will use NetFlow and Akvorado.

Since we know there will be a lot of packets in the traffic, it is impossible for us to log and track each packet. To solve this, the next step to detect residential proxies could be advanced monitoring. For this approach, we would set up Suricata rules on the OPNsense firewall to detect suspicious connections. In a second step, we will capture the flow using NetFlow and transfer it to a server running Akvorado. Akvorado has the capability to analyze and correlate network flows in order to detect proxy activity from its filter rules.

From Akvorado analysis, blocking rules could be derived, which are enforced by the OPNsense firewall. However, this approach has to consider the possibility of false positives. Since the

content of the traffic is typically not visible to the firewall, we have to make sure that we do not block traffic from legitimate users.

4 Implementation and Evaluation

4.1 Threat Simulation Setup

To make this simulation interesting, we created a little story with the following scenario:

Alpha (the attacker), wants to see the secret recipe that user Tango (the target) is serving as a website on his web server machine. Because Alpha does not want Tango to know that he is accessing his website, Alpha uses a residential proxy as camouflage for his real identity. The residential proxy is installed on the desktop machine owned by Victor (the victim). This means that Alpha will use Victor's identity, namely the IP address of the victim machine, to access Tango's website.

4.1.1 Installing the Actors' Virtual Machines

In this experiment scenario, we implement the simulation using virtual machines on the VirtualBox platform. For this context, we developed four machines: attacker, target, victim and firewall. All three operating systems that we installed as the actors are based on Debian Linux and for the firewall, we will use OPNsense. The author will not write in detail about how the installation process for all the virtual machines we will use, instead, we will draw a picture below to give an overview of the machine and networking setup.

Here are all the details of all our actors in this experiment:

- Username: alpha
- Role: Attacker
- Machine Name: attacker-desktop
- Operting System: Debian 13 Desktop
- IP Address: 10.0.3.5
- Network Adapter: NAT Alphanet Network

Second, we also installed the firewall protection for our simulation internal network. With a specification such as this:

- Username: root
- Role: Firewall
- Machine Name: OPNsense
- Operating System: OPNsense Firewall
- IP Address Internal (LAN): 192.168.56.6/24
- IP Address External (WAN): 10.0.3.4/24
- Internal Network Adapter: Host-only Adapter vboxnet 0
- External Network Adapter: NAT Alphanet Network

Next, we also created the second actor here. This actor acts as the primary victim because the attacker injects a malicious program, which we call the Potentially Unwanted Program (PUP), within in our case, we will use Squid Proxy.

Here is the machine's detail:

- Username: victor
- Role: Victim
- Machine Name: victim-proxy
- Operating System: Debian 13 Desktop
- IP Address: 192.168.56.4/24
- Network Adapter: Host-only Adapter vboxnet 0

Last, we installed the last actor for our experiment. This actor acts as the end victim, which is the end goal of the attacker's purpose to attack the machine using the first victim's identity. For the purpose of attack simulation, we will deploy an Apache web server on the target machine, which can be accessed via its IP address later. Here is the machine's detail:

- Username: tango
- Role: Target
- Machine Name: target-server
- Operating System: Debian 13 Server
- IP Address: 10.0.3.7/24
- Network Adapter: NAT Alphanet Network

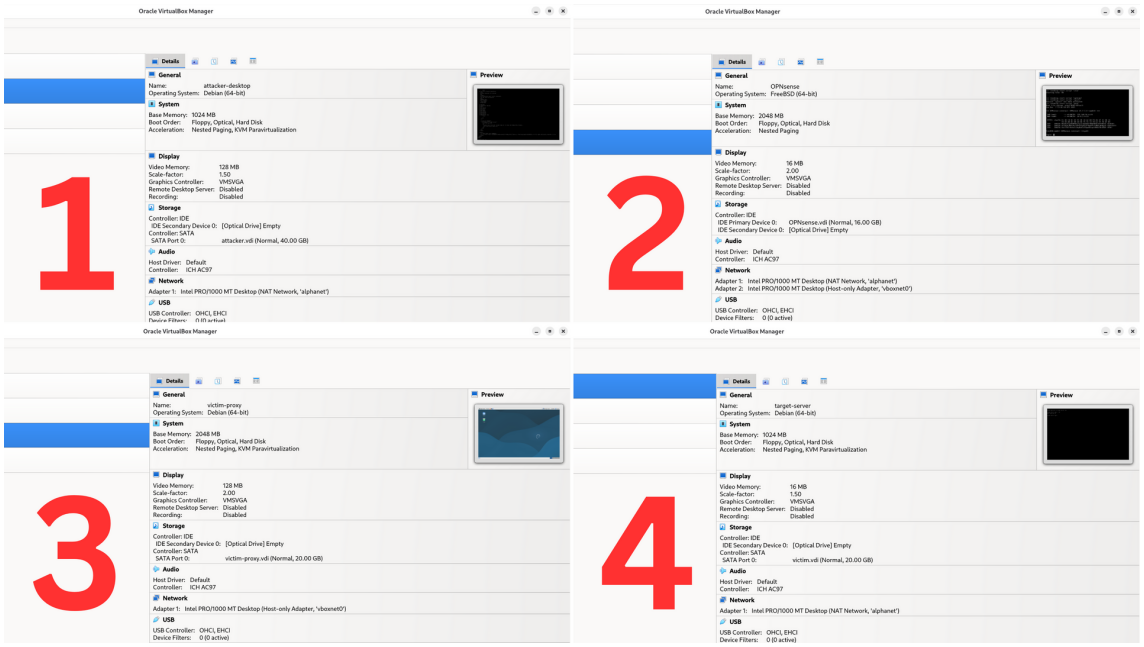


Figure 4.1: Virtual Machines Details

After all the machines are completely installed and running, here is what the final look of our VirtualBox manager with the list of the actors' virtual machines looks like. Next, we will set up the network configuration on the VirtualBox in the next chapter.

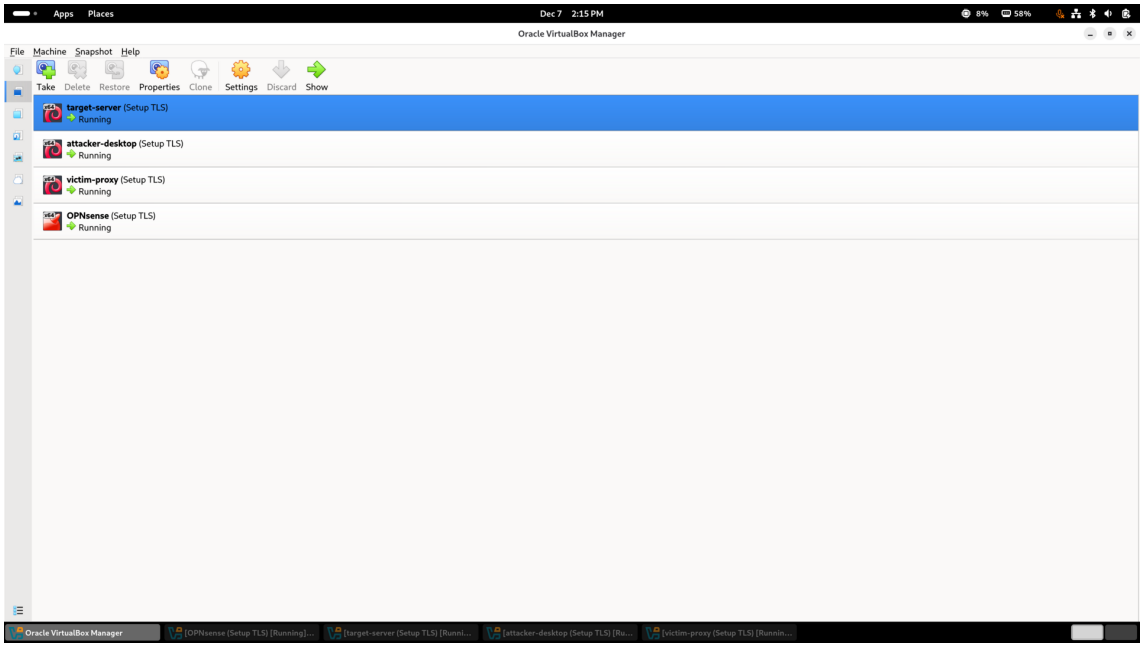


Figure 4.2: List of Virtual Machines

4.1.2 Network Configuration Setup

At this step, we can closely check each machines IP address and make sure that all the machines have distinct IP addresses. We note that the virtual network adapter from VirtualBox is named *enp0s3* here.

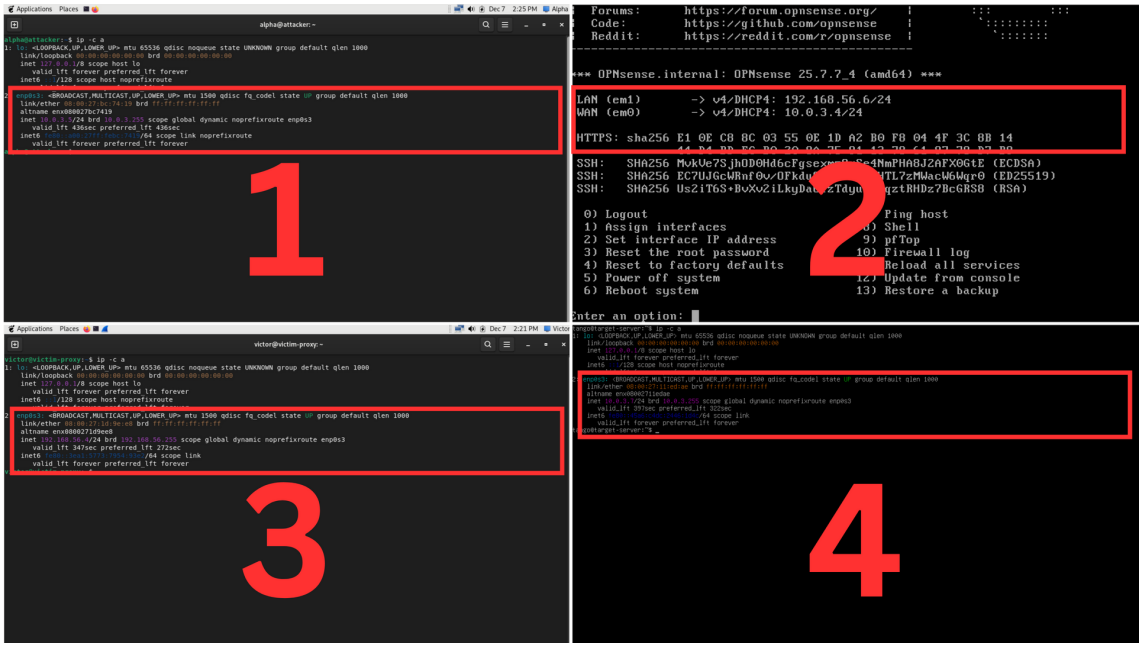


Figure 4.3: IP Addresses of Actors' Virtual Machines

As we discussed before, we want to simulate this threat with two different networks. First,

the internal network with the victim on it is guarded actively by the OPNsense firewall. Second, the external network, which simulates the public Internet. It has both the attacker and target on it. To make it plain, we will isolate the internal network from the Internet. We do this because we want to implement the deep packet inspection on the OPNsense to detect proxy activity on the victim machine.

Here are all the details of our network configuration on the VirtualBox. To break it down, we named the internal network as *Host-only net* and the external network as *NAT-Alphanet*.

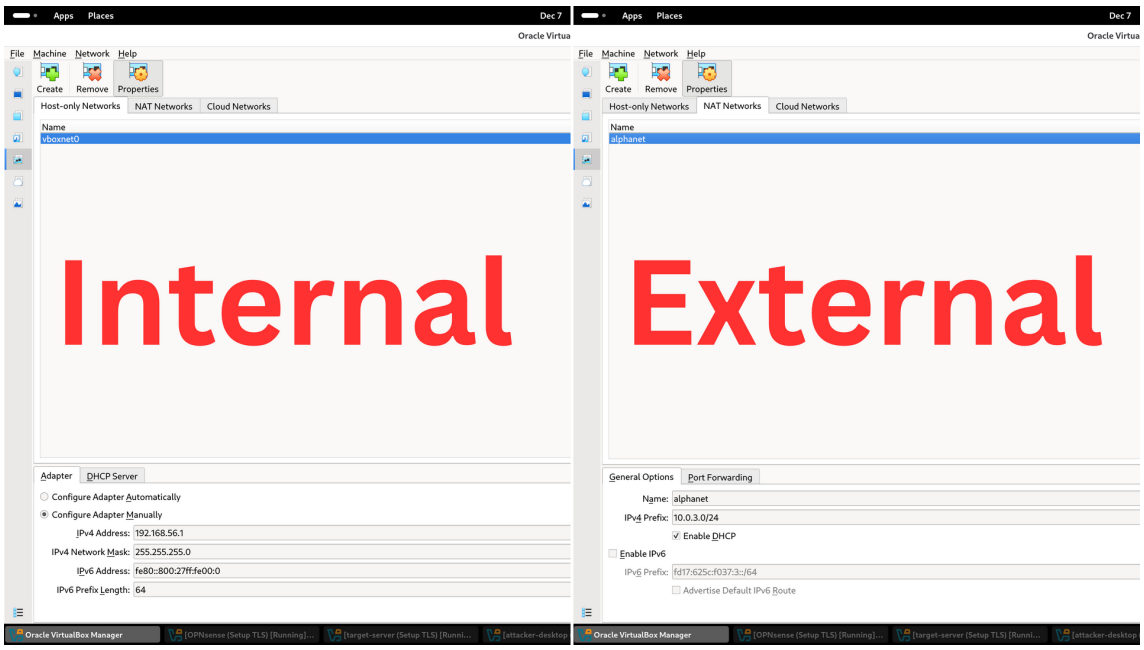


Figure 4.4: Virtual Network Configuration

We can illustrate this network setup with the picture below:

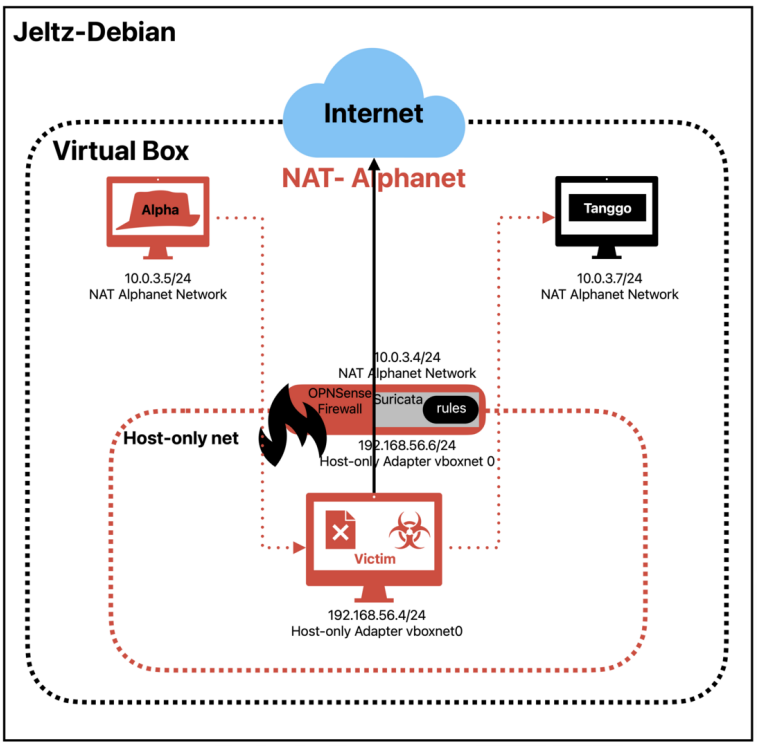


Figure 4.5: Virtual Machine Illustration

4.1.3 Web Server in Target Setup

For our setup, we need a target server, which the attacker accesses using his proxy. For this purpose, we have set up a simple Apache web server on the target machine. The website displays a single page with a secret recipe like we describe from the story simulation above.

4.1.4 HTTP and HTTPS in Target Setup

In order to make the target setup realistic, we have to enable TLS encryption. Since our target is not on the Internet, we cannot get a certificate from a public CA of the WebPKI. We are using a self-signed certificate. Since only the attacker has to accept the certificate, this has no impact on the detection on the firewall and victim machine. We can add simple configuration like the picture below under the `nano /etc/apache2/sites-enabled/000-default.conf` file.

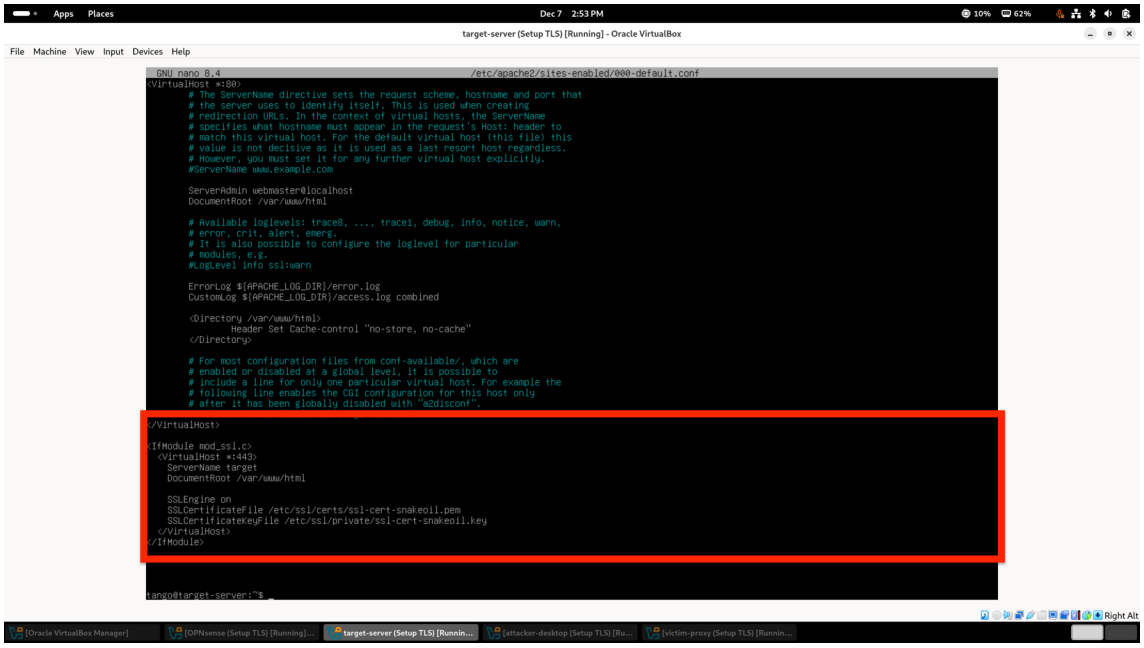


Figure 4.6: HTTPS Web Setup

4.1.5 Installing Squid Proxy in Victim Machine

The next step for us is to install the *Squid Proxy* in the victim machine. As we already discussed, the Squid proxy is acting as the Potential Unwanted Program (PUP) in our story here. Squid is a comprehensive HTTP proxy. Squid provides a comprehensive framework for access control, authorization, and logging, facilitating the development of web proxy and content-serving applications. Squid also provides a wide variety of traffic optimization features, the majority of which are activated by default to facilitate straightforward installation and ensure optimal performance [21].

It is relatively easy to install squid proxy. Here all the step we need to do:

1. As Squid is already included in Debian, we can install Squid using the apt package manager by running this command: `sudo apt install squid`.
2. After finishing the installation, Squid starts automatically. We can verify the Squid status by run this command: `sudo systemctl status squid`.
3. Next, it is essential to add the configuration of Squid. The goal is to establish an Access Control List known as myhost for a specific source's IP address, then allow HTTP access to the proxy server for clients that match with that ACL or IP address. Here, we input Alpha's IP address.
4. The Squid configuration is located under `/etc/squid/squid.conf`. We can edit it with our desired editor, such as nano: `sudo nano /etc/squid/squid.conf` or we can edit it with a text editor such as the picture above.

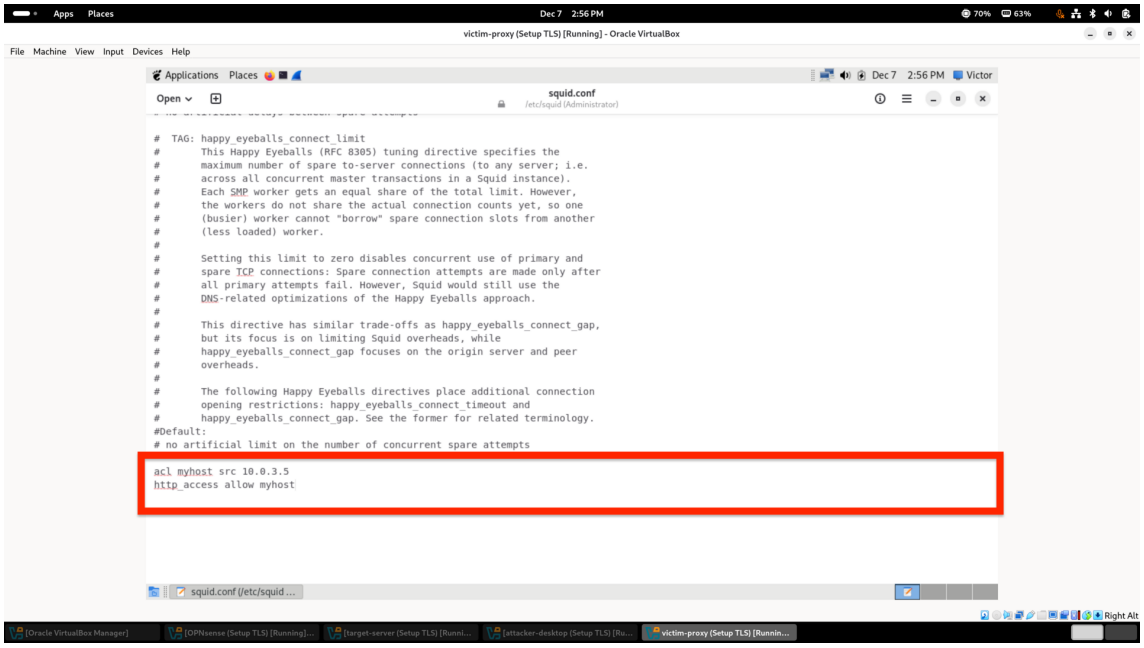


Figure 4.7: Squid Configuration

5. After we add the configuration, we need to save the file and restart the squid by running this command: `sudo systemctl restart squid`. Our squid configuration in the victim machine is ending here. Next, we need to check from the alpha machine if he can already access the proxy from Victor.

Next, we can set up the Squid Proxy from the client side, which is the attacker here, by entering the server's IP address and the port we desire in the attacker's network or browser proxy settings. Squid Proxy is using TCP port 3128 by default, but we can change it as we want.

To be clarified, for the attacker's desktop machine, the author installed it with the two types of interfaces. First, as we start to boot the attacker virtual machine, the command-line interface will appear there, which shows only the server-based interface that requires only the command line. This is used for the purpose of minimizing the memory workload of the real physical machine. Second, the graphic user interface that we can request by running this command: `systemctl enable graphical.target` and next, `systemctl isolate graphical.target`.

We can check the proxy connection from attacker to target through victim by running the curl command: `curl -proxy 10.0.3.4:3128 -insecure https://10.0.3.7/`. If the proxy connection succeeds, the output from the command will be like the picture below.

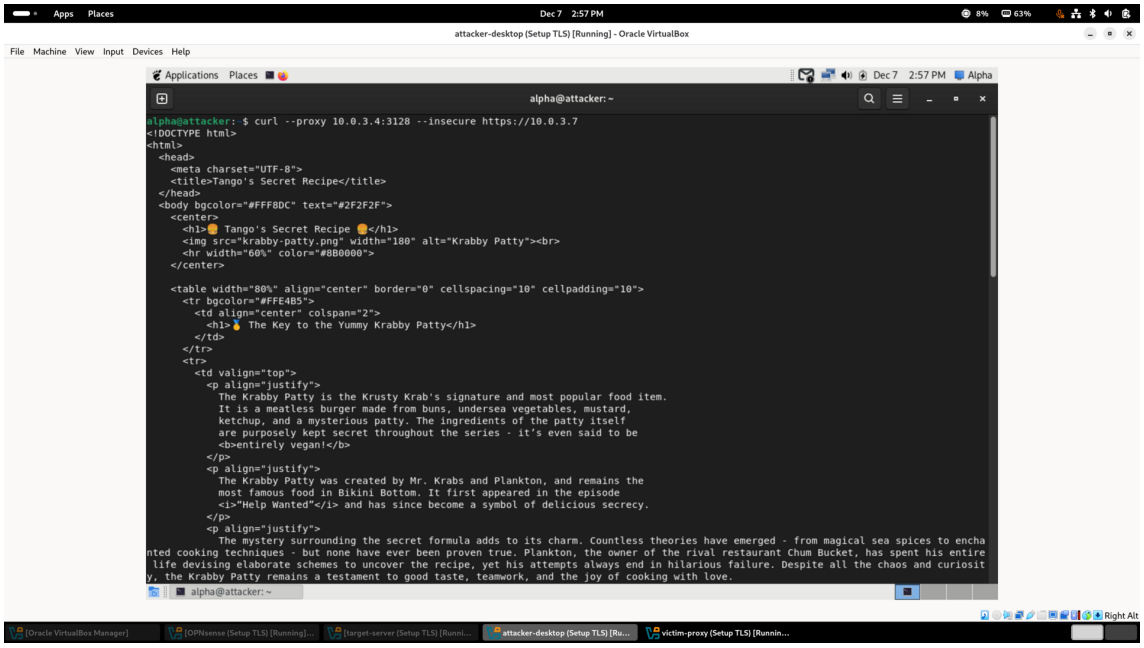


Figure 4.8: Curl Check Connection

We can also configure the proxy setting in the Firefox browser setting. We can go to the tab above and type `about:preferences`, we need to scroll down to the bottom to find the proxy setting and configure it like the picture below.

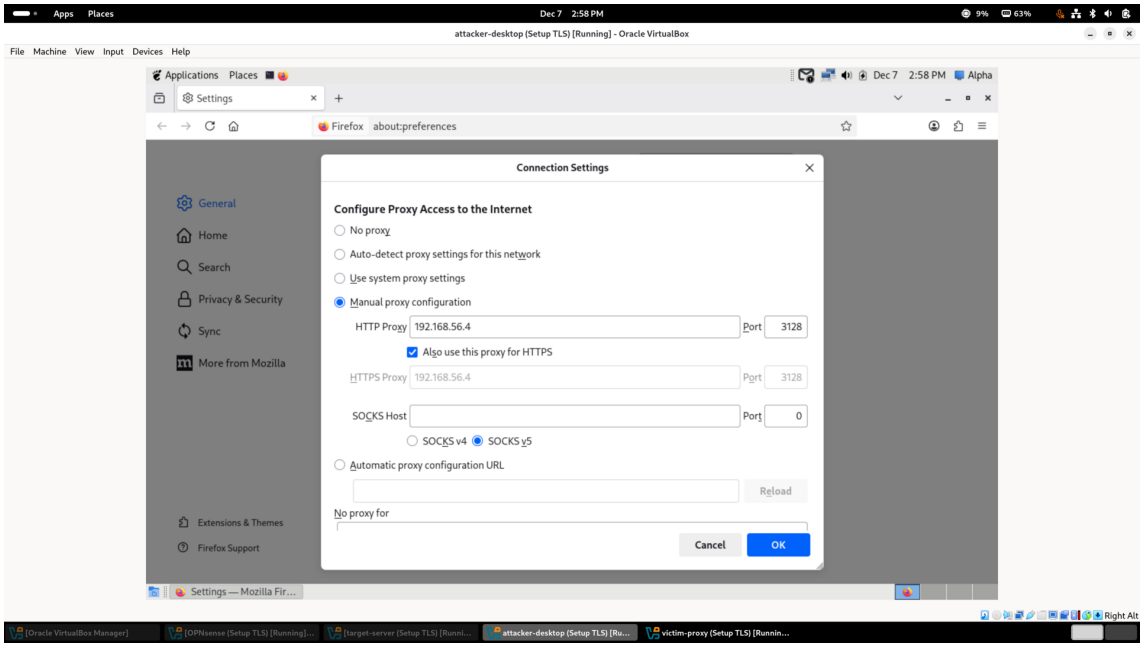


Figure 4.9: Proxy Setup in Firefox

Firefox is showing a security warning because the target server is using a self-signed certificate in our test setup. We can click to accept the risk and continue. Even so, this does not have a negative effect on our proxy detection.

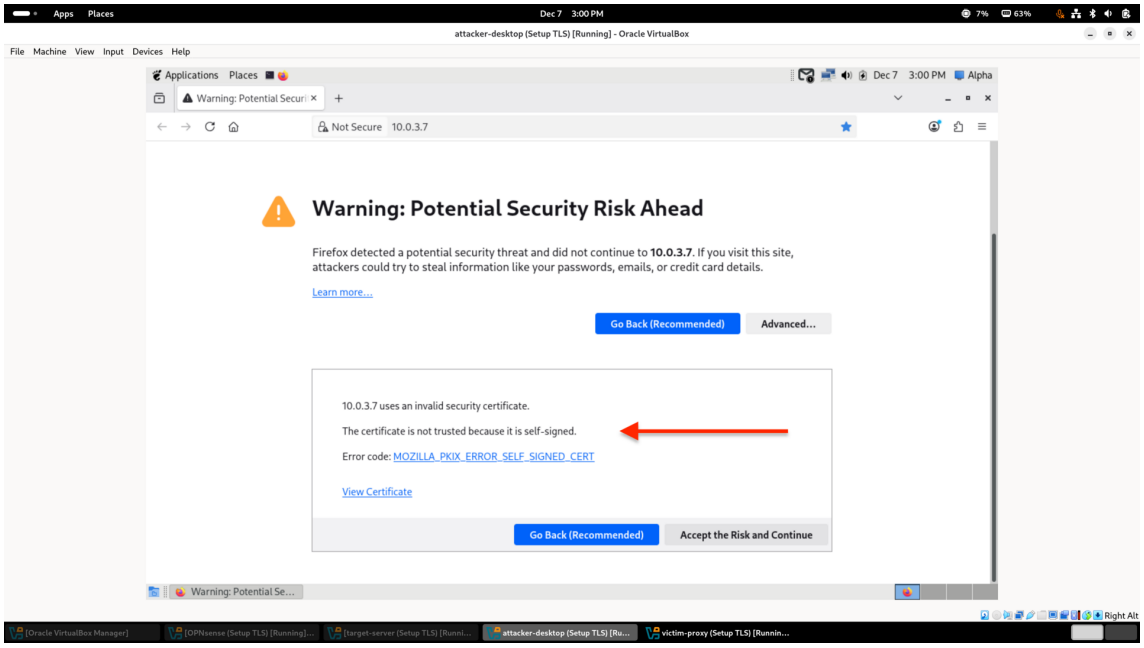


Figure 4.10: Security Risk Pop-up Window

We expect that most connections in real-world scenarios will use HTTPS rather than HTTP. For the proxy setup, this has the effect that the browser will start the connection with the `HTTP CONNECT` method.



Figure 4.11: Tango's Secret Recipe in HTTPS

This is the final look at how the attacker gets access to the target server from the proxy of the victim he used.

4.1.6 Capturing the Packet Traffic with Wireshark

We used Wireshark here as a help tool to identify and inspect the packets we closely looked at. When we run the proxy from the attacker's desktop, we can see the packet traffic using Wireshark. There, we can capture the packet that the attacker sends to the victim.

To use the Wireshark tool, we have to select the correct network adapter that we want to inspect here. As we mentioned earlier, the virtual network adapter we have is *enp0s3*. We also need to add the filter for the packet we want to inspect. The purpose of this is so we do not have so many irrelevant packets (such as DNS), which we do not need and are only distracting us.

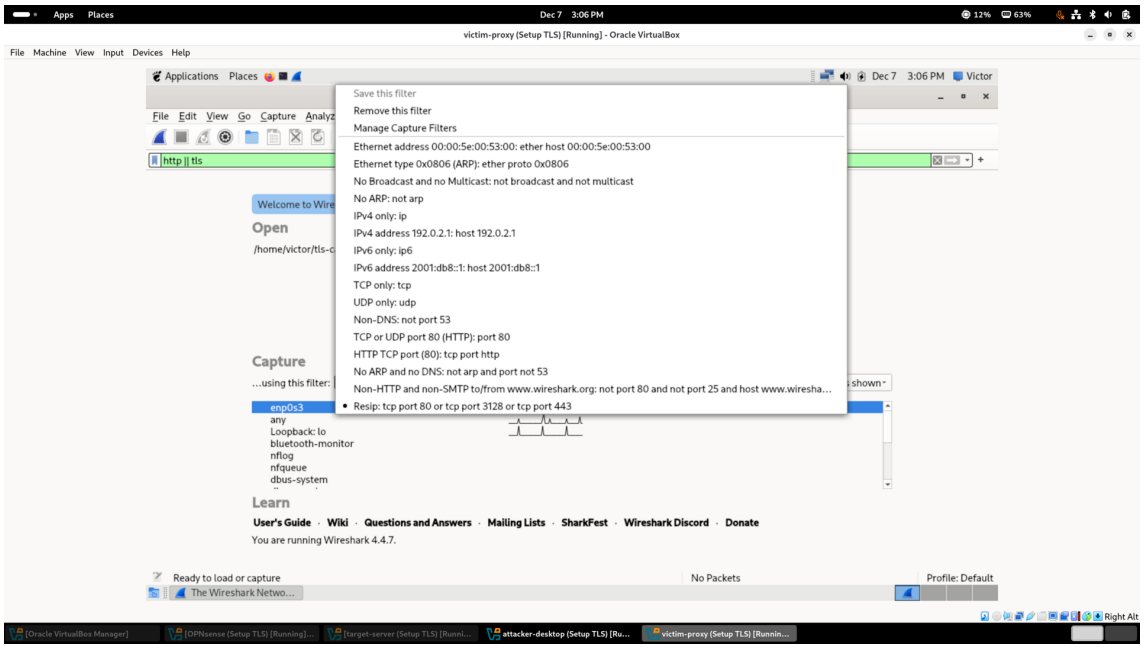


Figure 4.12: Wireshark Filter

We can add the Wireshark filter by adding it under `Capture > Capture Filter ...`. There, we add the filter name and filter expression. As an example here, we named our filter as `Resip` with the filter expression `tcp port 80 tcp port 3128 or tcp port 443`, then we click ok. Next, we select our filter, which we wrote before, and the virtual network adapter and run it. It will show us a lot of traffic there, like the picture below.

As noted here, to make it more readable for us. We can add the name of the host of all the *IP addresses* we include here, such as Alpha, Victor, Tango and OPNsense. Under the root mode in `nano /etc/host`. There, we can add the IP addresses with the name, so we can easily see the packet source and destination on Wireshark.

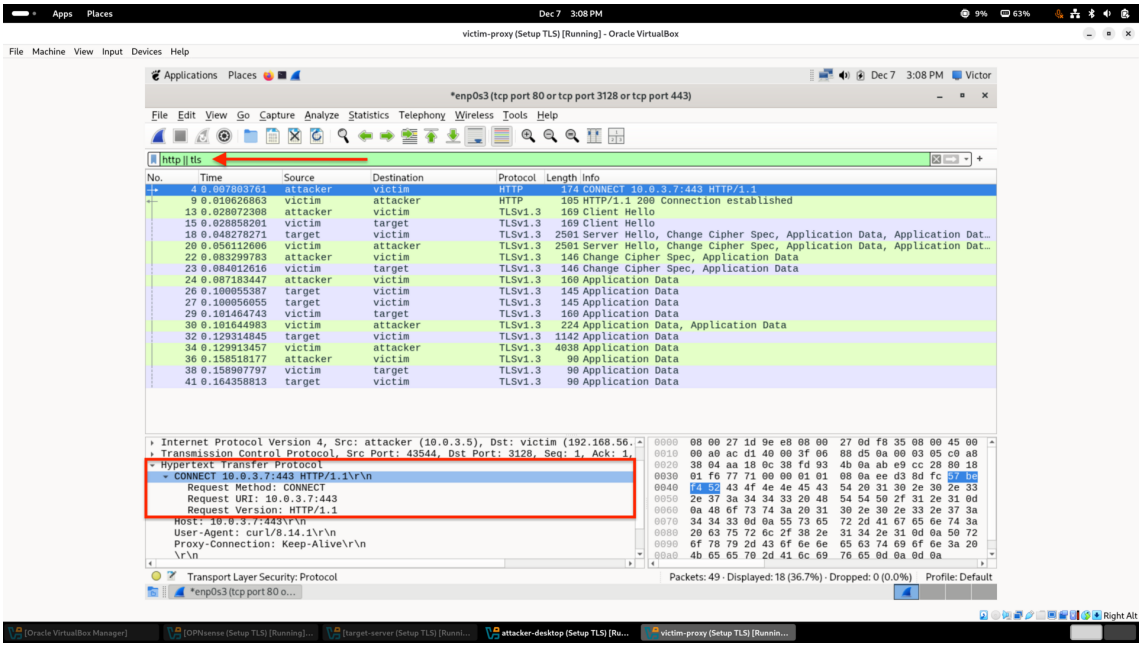


Figure 4.13: Wireshark Inspection

From the picture above, we successfully capture the packets from the sources and destinations we include in our simulation. By filtering the search `http || tls`, we can easily inspect it. However, there is an interesting part here: by closely looking at the proxy connection with HTTP, it always starts the request with `Method:CONNECT`, as we mentioned before in our previous chapter. From there, we can use it as the parameters for our Suricata rules later.

4.2 Suricata Configuration on OPNsense

At this time, after we are able to capture the incoming and outgoing traffic with Wireshark, we can continue to set up our main goal, which is the firewall rules. We can access the OPNsense interface by typing the internal firewall IP address in Firefox on the victim's desktop machine, which in our setup is `http://192.168.56.6/`. It is easier for us to access the OPNsense from the user interface side than from its command-line interface. Here is the OPNsense login page. We can log in to it using the same credentials from our installation.

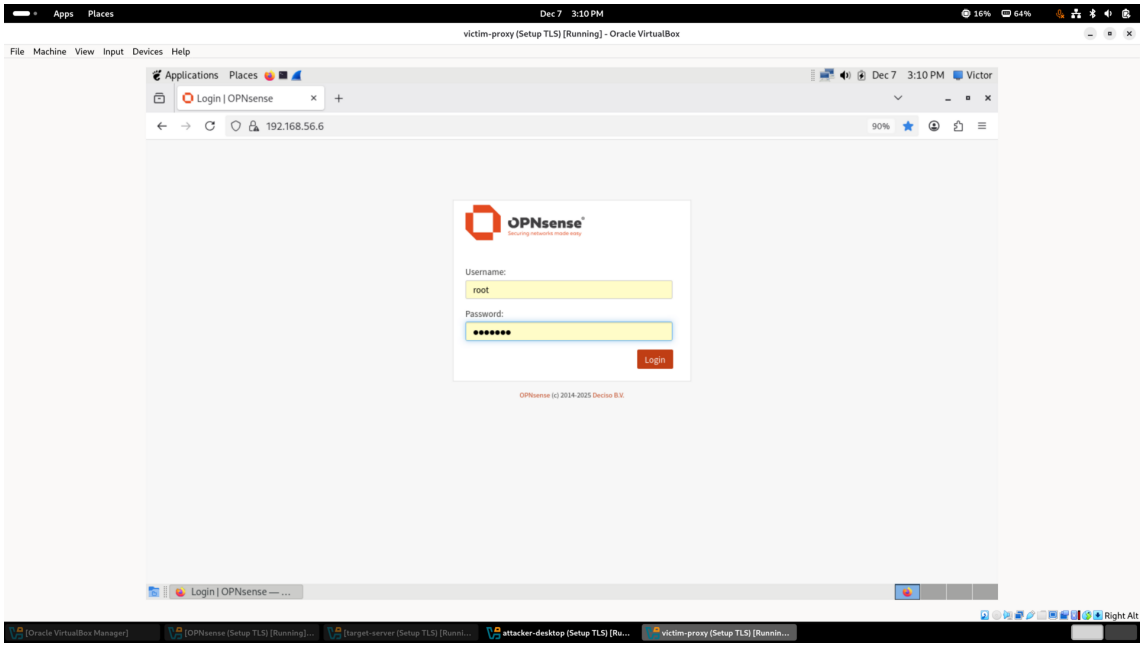


Figure 4.14: OPNsense Login Page

After we successfully log in to the OPNsense interface, we can see the OPNsense dashboard, like the picture below:

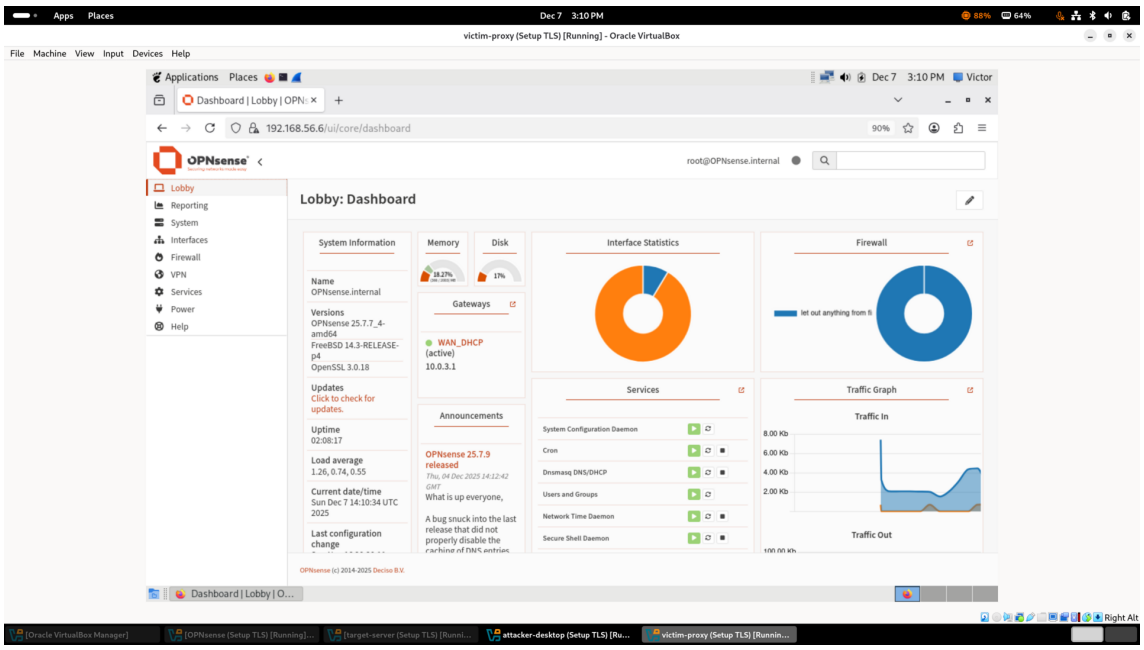


Figure 4.15: OPNsense Dashboard

As we already discussed in the previous section, Suricata is part of the OPNsense features. This means that we do not need to do a manual installation of Suricata. We can access Suricata by going through the OPNsense navigator interface on the left side window, then we

choose `Service > Intrusion Detection > Administration`, like how the picture shows below. As a highlight, we will mostly work in this section of OPNsense because this is where Suricata is.

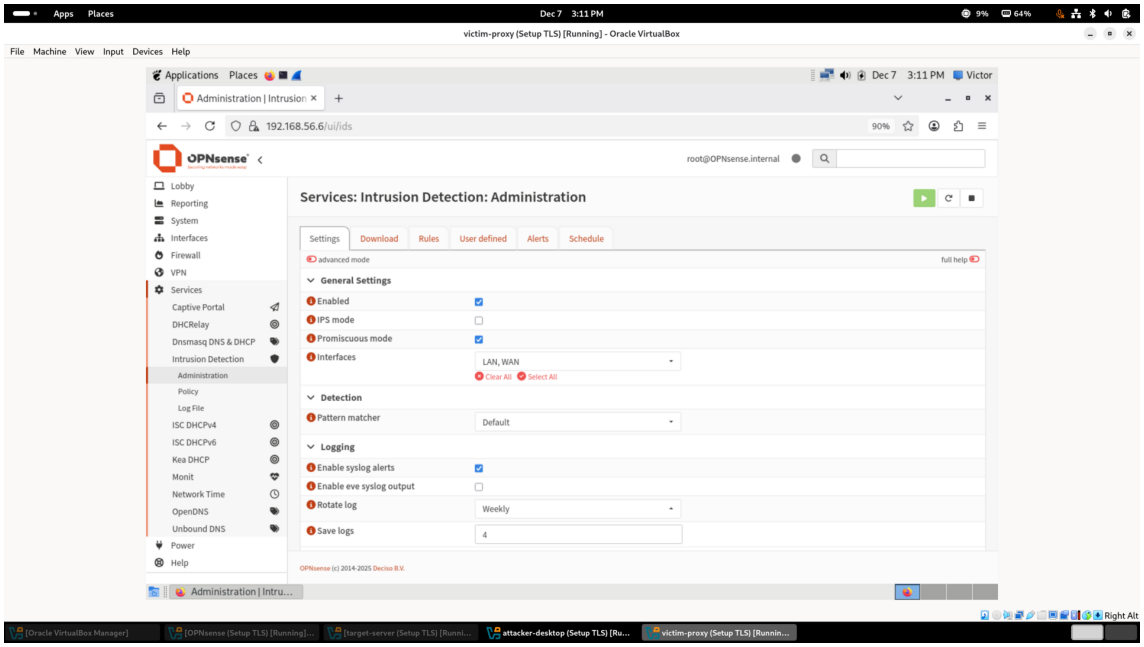


Figure 4.16: Suricata in OPNsense

Suricata in OPNsense has a unique way to be able to use its custom rules. It is different from Snort in pfSense in that we can write the rules directly to its firewall Snort rules box, but in Suricata, it's different. In order to be able to use Suricata custom rules, we need to write down the rules in a file on a web server that we can access. From there, we're simulating the way how OPNsense downloads and updates its rulesets from a URL. We will discuss more about the structure of our custom proxy rules that we will write and execute in the next chapter. However, for this time, we will prepare in Suricata so it can access our custom ruleset.

Before we write our custom rule file that we put on a web server, we need to do several steps. First, we have to create a custom rule file. We can call it `proxy.rules`. As a note, we have to allow `.rules` extensions in request filtering rules in our web server configuration, and add the MIME type as text or plain. Next, we need to locate `proxy.rules` within a website that we can easily get and edit. For this case, the author will use her website as the location to put the ruleset, so we can access it under this URL:
`https://selviyani.verbuecheln.ch/proxy/proxy.rules`.

The website that we used as the location from `proxy.rules` should be accessible from the OPNsense network and display in our text form. Then, we need a way to be able to use our `proxy.rules` from the Suricata interface, which is in `Service > Intrusion Detection > Download`, like figure 4.17 shows. For that, we need to create an XML file and place it in the filesystem of the OPNsense machine. Let us create an XML file, and we named it `proxy.xml`. This `proxy.xml` file must go under this location in OPNsense:

`/usr/local/opnsense/script/suricata/metadata/rules/` and this is how we write our `proxy.xml` file.

```
<?xml version="1.0"?>
<ruleset documentation_url="http://docs.opnsense.org/">
    <location url="https://selviyani.verbuecheln.ch/proxy/"
            prefix="Custom"/>
    <files>
        <file description="proxy rules">proxy.rules</file>
        <file description="proxy rules" url=
            "inline::proxy/proxy.rules">proxy.rules</file>
    </files>
</ruleset>
```

It will look like the picture below:

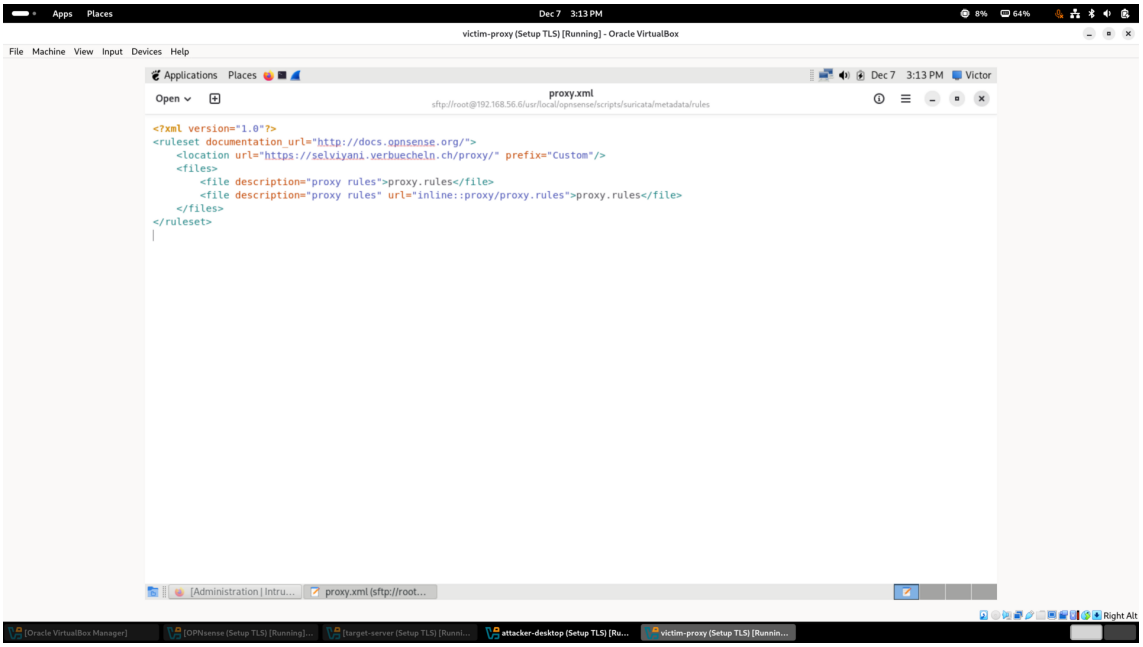


Figure 4.17: XML file in OPNsense

Once all this is done, the ruleset will appear in the download list as `Custom/proxy rules` after a refresh. Then we click `enable selected` on the proxy rules we checkmark and click `Download & Update Rules` as the picture below describes it.

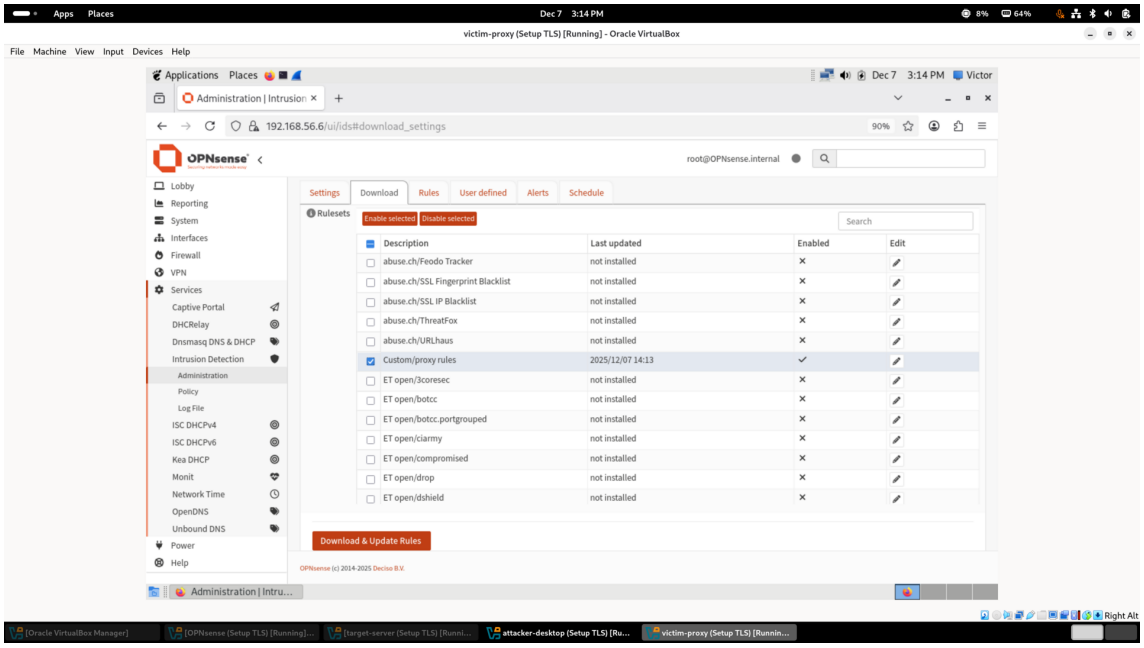


Figure 4.18: Proxy Rules appears in Download List

After we download the rules and enable them, we can check them under `Rules` section to see that our ruleset successfully updated. There we see three different rules we created.

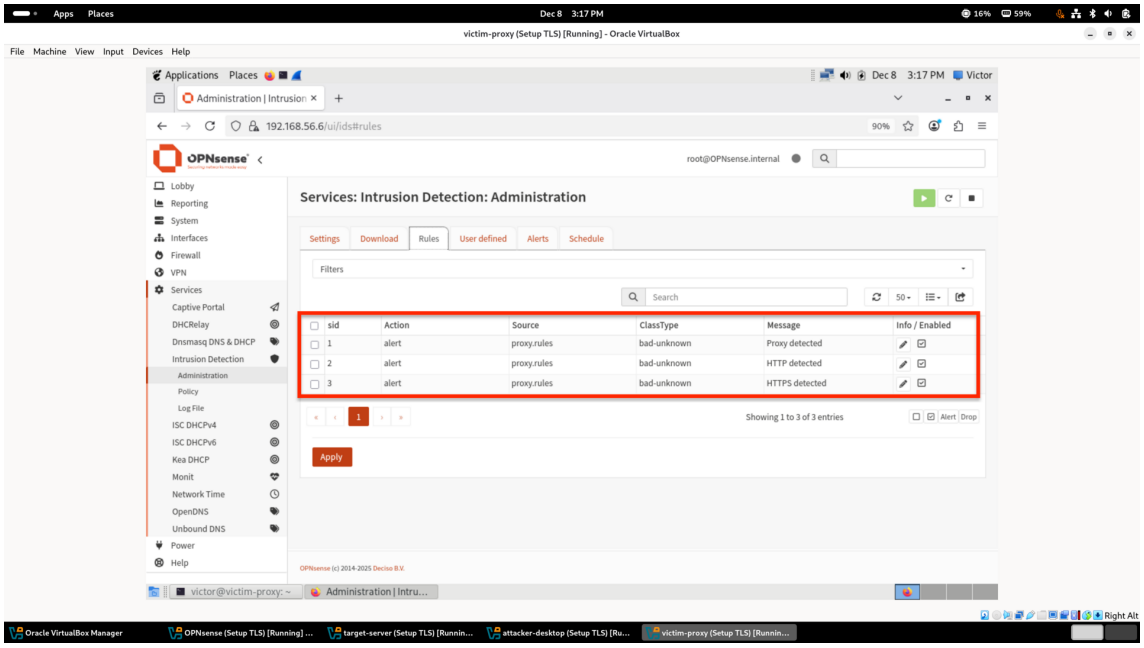


Figure 4.19: Ruleset appears in Rules section

In the next chapter, we will discuss the basics of Suricata rules that we will use and all its protocols, so we can be able to detect the proxy connection in our traffic.

4.3 Development of Detection Rules

This section's explanation will mostly be based on the Suricata rules documentation [22] and the video explaining how to understand the basic rule in Suricata [23]. In here, we try to break down how to write the rules in Suricata. As we already discussed before, we want Suricata to be able to detect the incoming traffic requested from Alpha to use the Victim proxy, and we also want Suricata to be able to detect the outgoing traffic from Victim to Target communication.

Suricata rules have three major components: the action, the header, and the rule options. The action defines what happens when the rule matches. The header defines the major sections of the rules, such as defining the protocol, IP address, ports, and direction of the rule itself. The options provided further refinement of the rules' behavior. For example, in the picture below, the black box is the action, the red box is the headers, and the green box is the options.

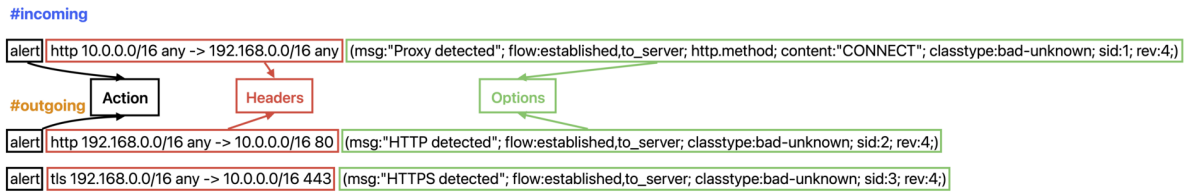


Figure 4.20: Format Rules in Suricata

4.3.1 Action Section

Based on the Suricata rules documentation [22], there are several valid actions:

- alert: Generate an alert and let the packet through
- pass: Stop further inspection and let packet through
- drop: Drop packet, if this happens in IPS mode, and generate alert
- reject: Set the RST/ICMP unreachable message to the sender of the matching packet
- rejectsrc: Same action like *reject*
- rejectdst: Set RST/ICMP error message to receiver of the matching packet
- rejectboth: Set RST/ICMP error message to both sides of the conversation.

The action section of a Suricata rule defines what happens when the rule is matched, and the behavior differs between detection mode and prevention mode. In prevention mode, it allows for more actions, such as stopping packets and sending resets. Meanwhile, in detection mode, the *alert* action generates an alert and lets the packet through.

4.3.2 Header Section

The header section is a major section that contains four subsections, including protocol, source IP addresses, destination IP addresses, ports or port ranges, and traffic direction. Those are essential for defining the behavior of Suricata rules.

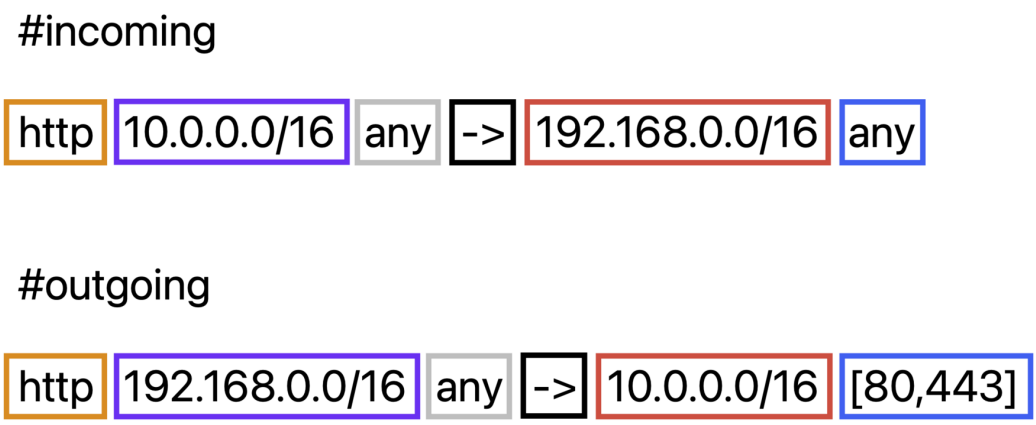


Figure 4.21: Action and Header Section in Suricata Rules

Protocol

Based on Suricata documentation, it outlines a collection of protocol values that may be specified for the application of signatures within a system. Suricata provides a comprehensive overview of options across various layers. It includes:

- Network protocols: IPv4, IPv6, ICMP, TCP, UDP.
- TCP options: tcp-pkt (single packet match) and tcp-stream (reassembled stream match)
- Application layer protocols: HTTP (version 1 and 2), TLS, FTP, DNS, SSH, SMTP and several others

The accessibility of these protocols depends on their activation within the configuration file, *suricata.yaml*. A signature that targets "HTTP" guarantees that it exclusively aligns with HTTP traffic within TCP streams. For that reason here, the protocol we used on the orange box above the figure is *HTTP*.

Source and Destination

In the context of network traffic management, the source and destination indicate the origin and target of data flow, allowing the assignment of IP addresses, which include both IPv4 and IPv6 formats, as well as including IP ranges.

Furthermore, parameters such `$HOME_NET` and `$EXTERNAL_NET` are normally used to improve the flexibility of the configuration. The variables are defined within the *suricata.yaml* configuration file, which specifies the corresponding IP addresses that will replace the variables in the network rules.

For our case here, we specifically define the source and destination address because we already specifically know the source of the attacker and the destination of the target. We can see it in the purple and blue box in the figure 4.21 above.

Ports (from Source and Destination)

Traffic comes and goes via ports. Various protocols have been given different port numbers. For instance, the standard port designated for HTTP is 80, whereas port 443 is commonly

associated with HTTPS. It is important to acknowledge that the port does not determine the protocol utilized in the communication process. Instead, it identifies the specific application that is receiving the data.

The previously mentioned ports are usually considered as the destination ports. Source ports, which refer to the application responsible for sending the packet, are generally assigned a random port by the operating system. When formulating a rule for our HTTP service, one would generally specify `any -> 80`, indicating that any packet originating from any source port directly towards to our HTTP application, which is operating on port 80, and is considered match.

In our case, we used `any` as the defined ports in our rules. As we know that the attacker can use any ports to send the packets, however, for the outgoing rules, we have grouped our ports here, which are only specific to port 80 and port 443.

Direction

The directional arrow represents the process by which the signature will be inspected. In the majority of signatures, an arrow pointing to the right (`->`) is used. This indicates that only packets going in the same direction are able to match. The double arrow (`<->`) maintains the same directionality as the single arrow (`->`), yet it facilitates matching on both directions of transactions, using keywords that correspond to each direction. At some point, it is feasible for a rule to match in either direction (`<>`).

4.3.3 Options Section

The options section is the core of a signature and always contains, in parentheses, with options that can be binary or key-value pairs, and the order of these options matters, as they are processed in sequences.

Each key-value pair in the options section can be either metadata or a matching action, and the keywords used in the key field depend on the selected protocol. For our rules, we used only three types of keywords based on the Suricata Rules documentation, which are: Meta, Flow, HTTP and Payload. This is explained in the boxes we draw in our figure below.

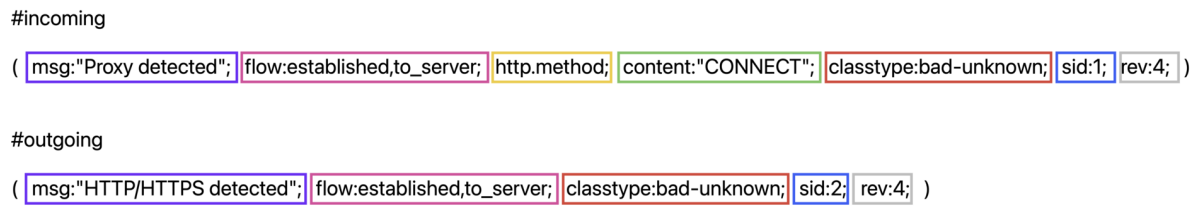


Figure 4.22: Options Section in Suricata Rules

Meta Keyword (`msg`)

The meta keyword `msg` provides contextual information regarding the signature and the potential alert. For our case, the purple box in our figure, we put two different messages here: For the incoming rule, we put the message *Proxy detected*. Meanwhile, for the outgoing rule, we set it as *HTTP/HTTPS detected*.

Flow Keyword (`flow`)

The flow keyword `flow` serves to identify the direction of the flow, whether it is *to* or *from* the client or *to* or *from* the server. It can also determine whether the flow has been established. The flow keyword may also be utilized to indicate that the signature must match only on stream (only_stream) or just one packet (no_stream).

We match the flow keyword with `established`, which means it must establish connections, and `to_server`, which means the packets from client to server.

HTTP Keyword (`http.method`)

For our case, we also used HTTP keywords, which is `http.method` to match on the method-/verb used in an HTTP request.

Payload Keyword (`content`)

The content keyword is very important in the signature. Based on our inspection on Wireshark, we capture that there is a packet that contains the word `CONNECT` when the payload is not encrypted in HTTP. That's why here we set the content keyword between the quotation marks `CONNECT`, so we can inspect it when the incoming packet matches with it.

Meta Keyword (`classtype`)

The classtype keyword provides insights into the categorization of rules and alerts. The elements include a brief designation, an extended title, and a level of priority. It can determine, for instance, whether a rule is just informational or related to a CVE. For example, in our case, we used classtype here as `classtype:bad_unknown`.

Meta Keyword (`sid`)

The keyword sid is an extension of *signature ID*, which means that `sid` assigns a unique identifier to each signature. This ID is specified with a value higher than zero.

Meta Keyword (`rev`)

The sid keyword is often linked with the `rev` keyword. Rev denotes the version iteration of the signature. In the event that a signature is changed, the count of revisions will be increased by those who wrote the signature.

4.3.4 Rules for Resident Proxies

After all the explanations above, we declared, and now we are ready, to set up our rules on the website we assigned as the URL for Suricata to take it as a ruleset.

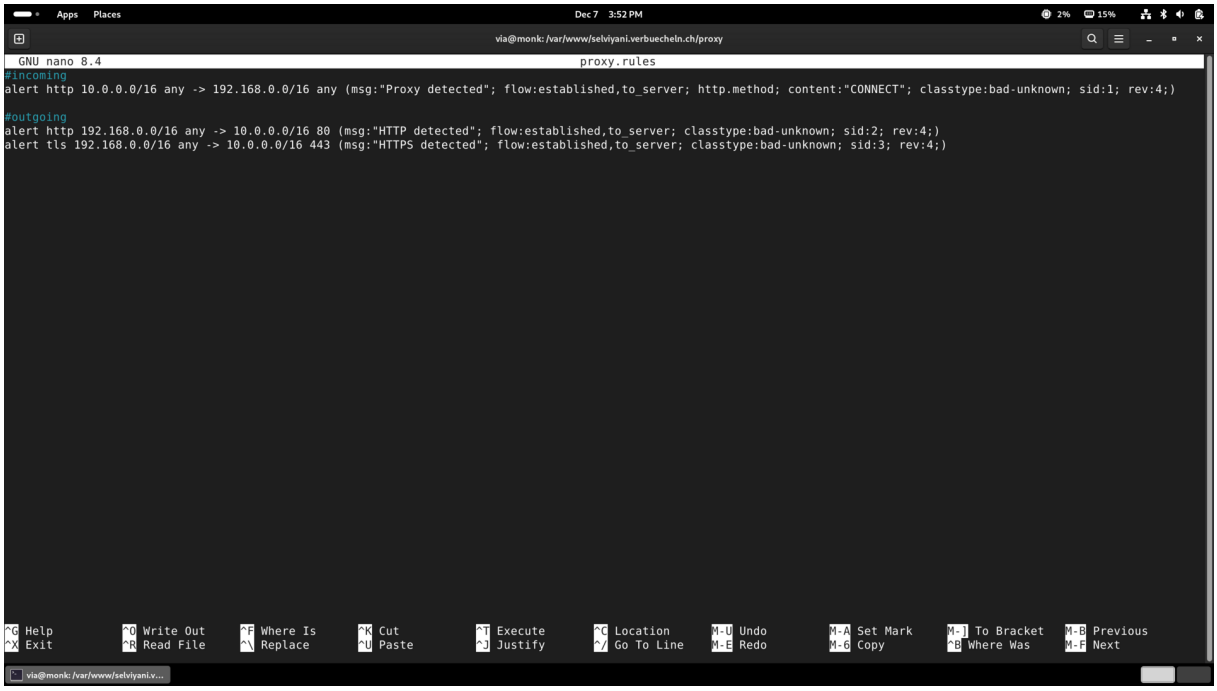


Figure 4.23: Final Ruleset for Detect Residential Proxy

The first rule is designed to detect connections from the outside to a proxy. In this case, we assume that Squid is used, which uses HTTP on port 3128.

The second and third rules are designed to detect the connections that the attacker is attempting with the proxy. In this case, we assume that it is HTTP or HTTPS.

After we download and enable our ruleset in Suricata OPNsense. Then, we can try to run it for access to Tango's Secret Recipe by proxy in Firefox on the Alpha desktop, as we described in the previous chapter.

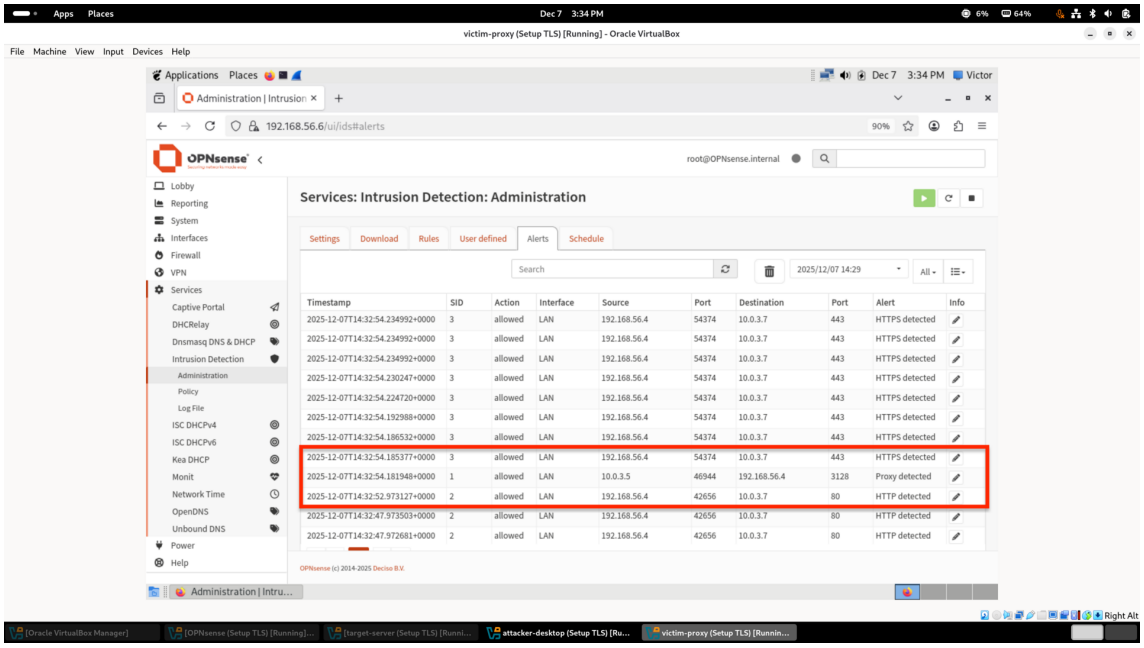


Figure 4.24: Suricata Alerts

As the figure above describes, we can see Suricata triggers under the alert section. It shows our rules work to detect the proxy connection, HTTP, and HTTPS when we run the proxy for access to the target web server.

4.4 Development of Dynamic Detection

As we already discussed in the previous chapter, NetFlow might help us to do dynamic detection. In this chapter, we will explain how we do NetFlow and Akvorado.

4.4.1 NetFlow and Akvorado Setup

For this development setup, we need to install Docker to run Akvorado. `Docker Compose` is available in the Debian operating system. For shortening purposes, we will not explain in detail how we install Docker and Akvorado. The author will reference the Akvorado quick start installation guide GitHub link.

The Akvorado quick start installation ZIP archive contains two main folders, which are `docker` and `config`. We will move our focus to the Docker folder first here. Then, we just run `docker compose ps` to run the Docker and see several containers that are currently running.

As we can see, the Akvorado Console is operating on localhost with port 8080. Keep this in mind, as it will be useful for us to be able to access the Akvorado UI later.

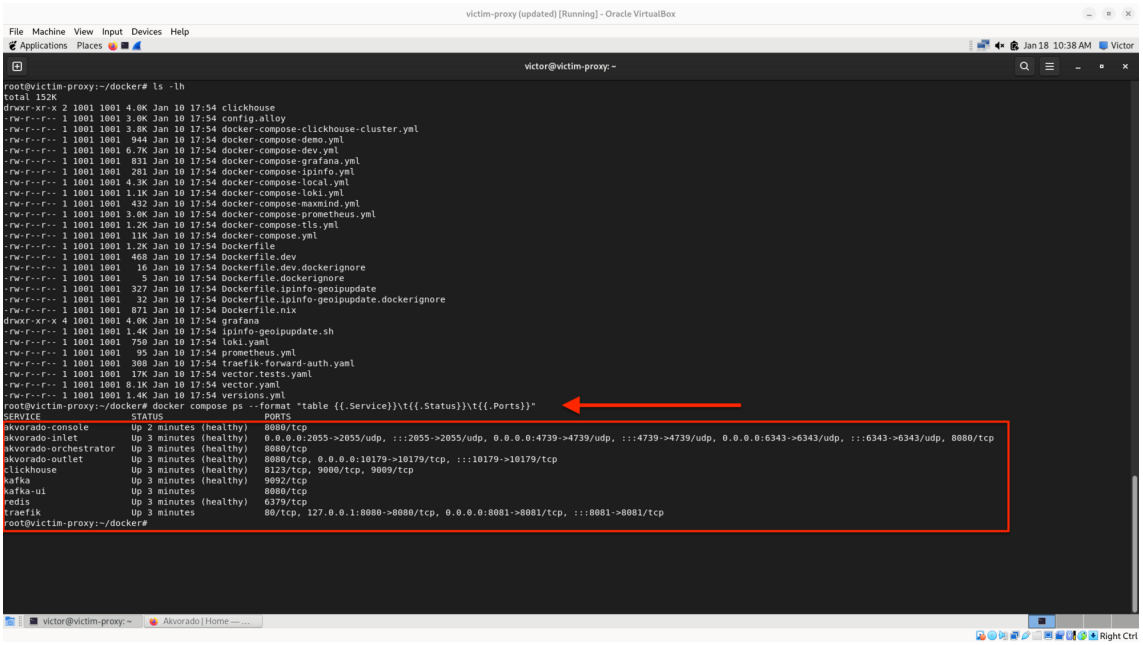


Figure 4.25: Start the Docker

From there, we see several containers, such as Akvorado, ClickHouse, and Kafka, etc., which represent the containers we see in the overview picture [20].

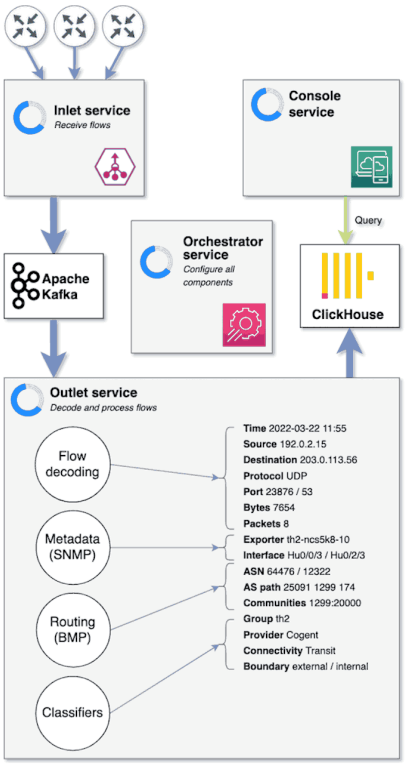


Figure 4.26: Components of Akvorado

For the next step, we need to set up the OPNsense side, for which there are two things we need to do. First, we need to go to the `Reporting > NetFlow` section. There, we need to set up the destination. In this context, we ask OPNsense to send its data to the Inlet in Akvorado. To note here, we don't need to change the port of NetFlow in Akvorado, as it registers automatically in port `2055`, such as in Akvorado's config folder `config > inlet.yaml`, and for the IP address of the destination, we will use the machine's IP address.

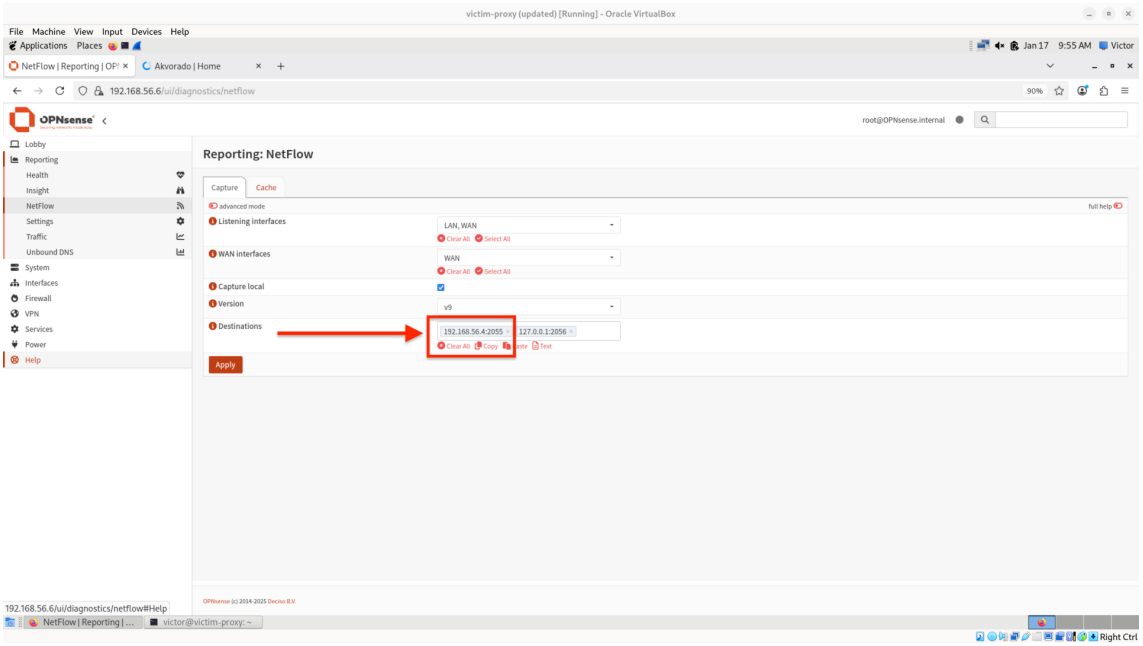


Figure 4.27: NetFlow Destination Setup in OPNsense

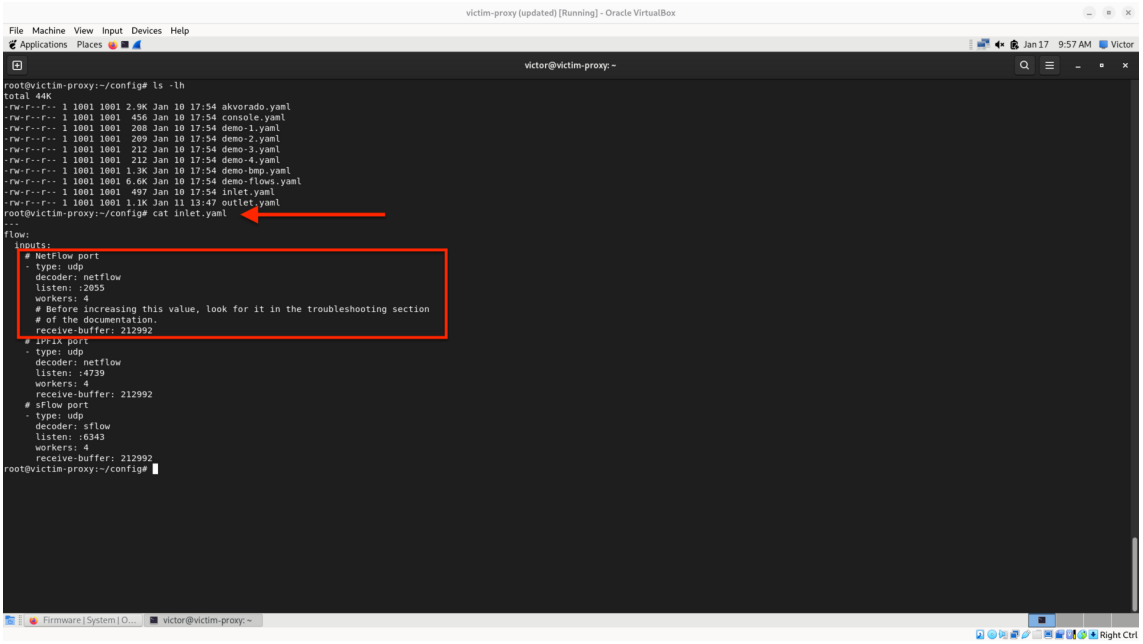


Figure 4.28: NetFlow Port in inlet.yaml from Akvorado Config

Secondly, we need to install the SNMP plugin on the OPNsense side. It is available under the `System > Firmware > Plugins`. There, we need to install `os-net-snmp`. In this context, we add a role for OPNsense as a daemon for the SNMP protocol. After we install the plugin, the SNMP settings appear under `Services > Net-SNMP`. Here, we need to set up the `SNMP Community`, we set it to *public* and `Listen IPs` to the IP address of the firewall. We also need to set up the community to public in the Akvorado config folder, under the `config > outlet.yaml`. To be noted here, we needed to add `default-sampling-rate:1` in the config, like the troubleshooting site suggested, because OPNsense does not report a sampling rate with NetFlow.

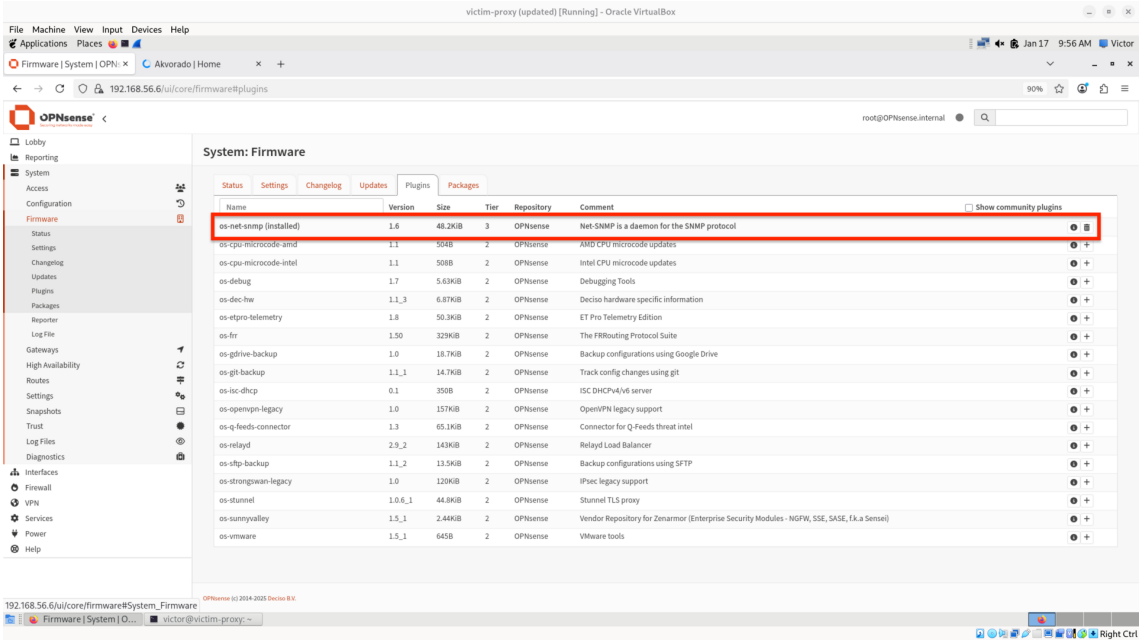


Figure 4.29: Installation SNMP Plugin in OPNsense

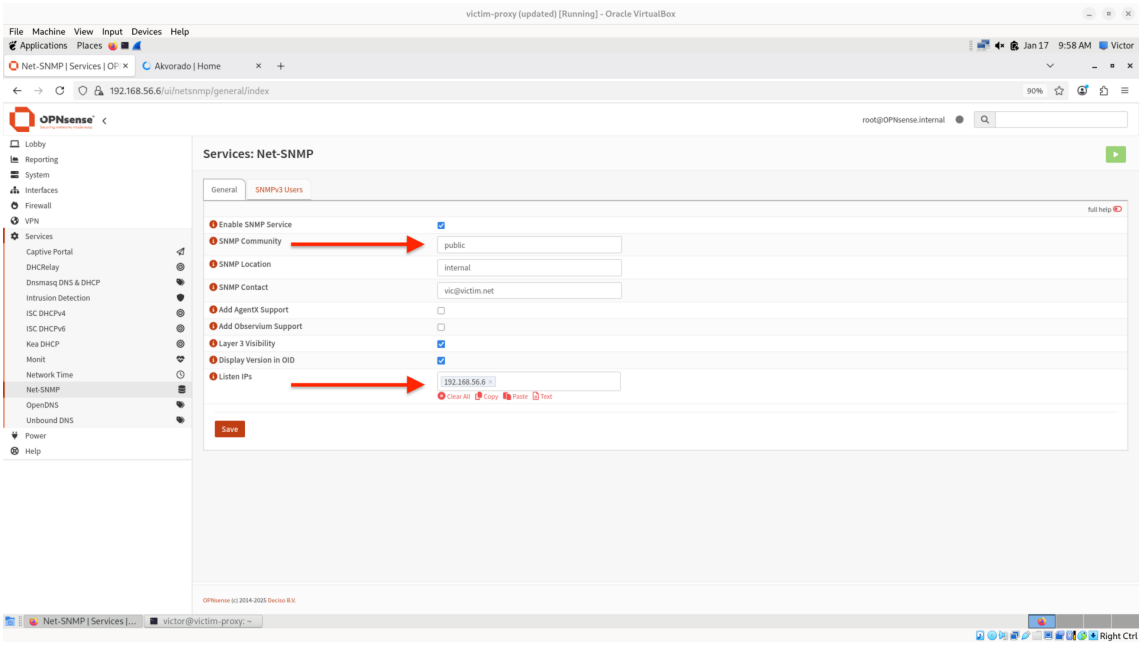


Figure 4.30: SNMP Configuration in OPNsense

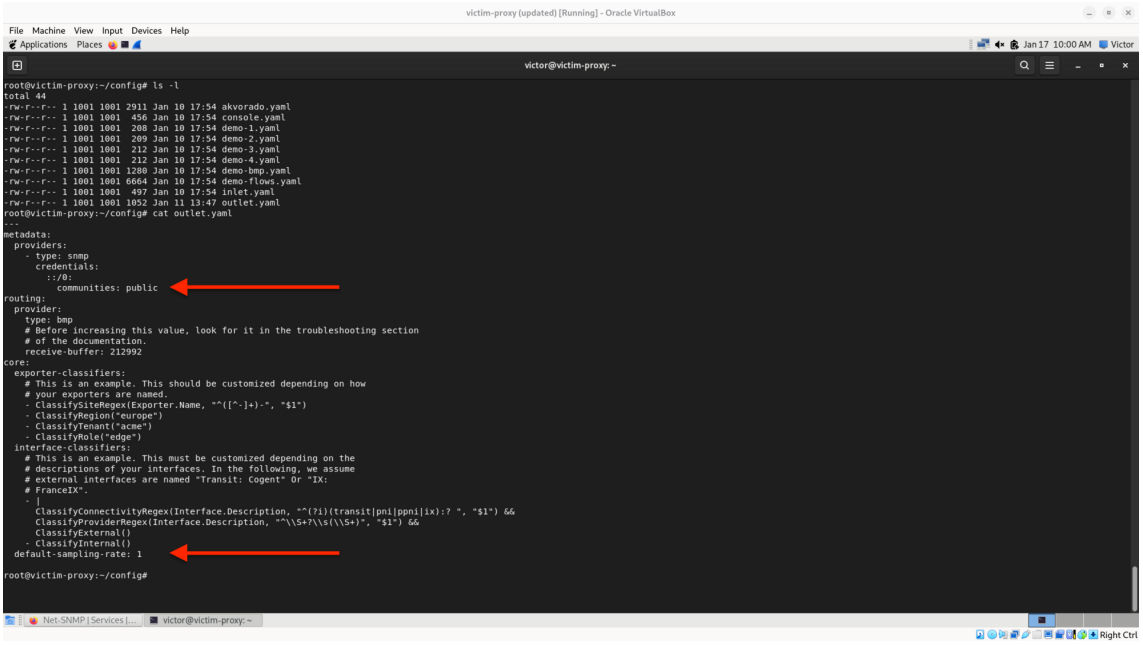


Figure 4.31: Akvorado Config Setup in outlet.yaml

4.4.2 Traffic Simulation

After that, we check that all the containers are running well with Akvorado Docker. We can run our simulation to test how Akvorado displays the flow from the OPNsense traffic. First, we need to access the Akvorado Interface from our browser in `http://localhost:8080`. Second, we will create traffic from the attacker using a proxy of the victim to download ISO images with curl, as shown in Figure 4.32. This traffic will show us a big flow of metadata.

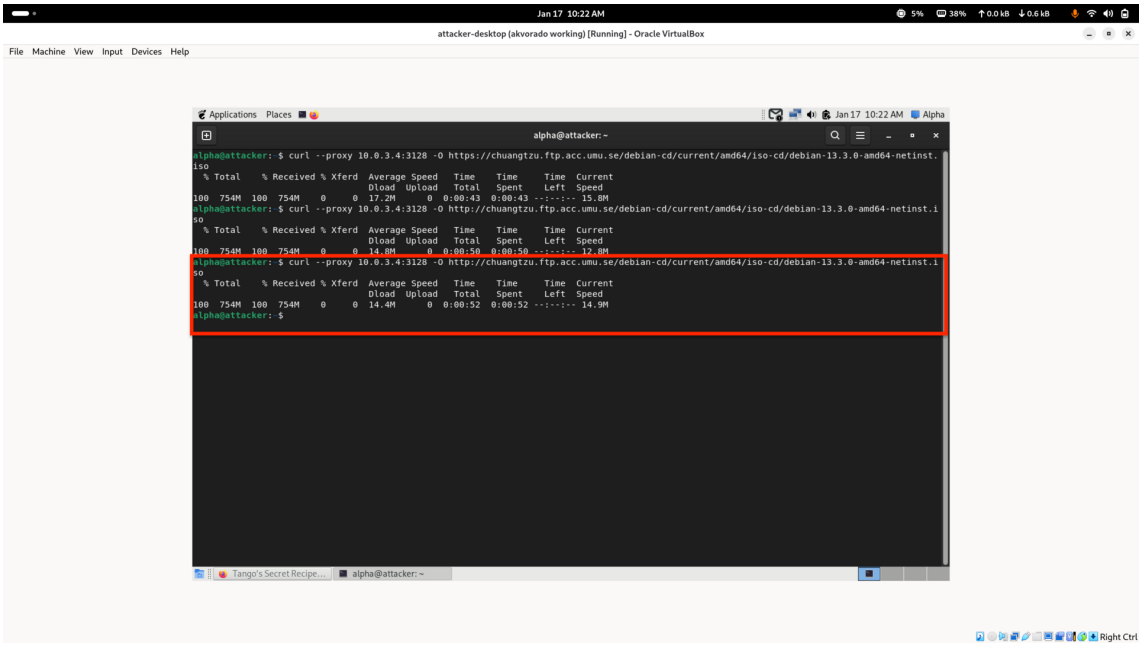


Figure 4.32: Curl Proxy from Attacker to Victim

Then, we check the traffic that we create in the Visualize section of the Akvorado UI, with some dimensions such as `DstAddr`, `SrcAddr`, `DstPort`, `SrcPort`, as the figure below shows.

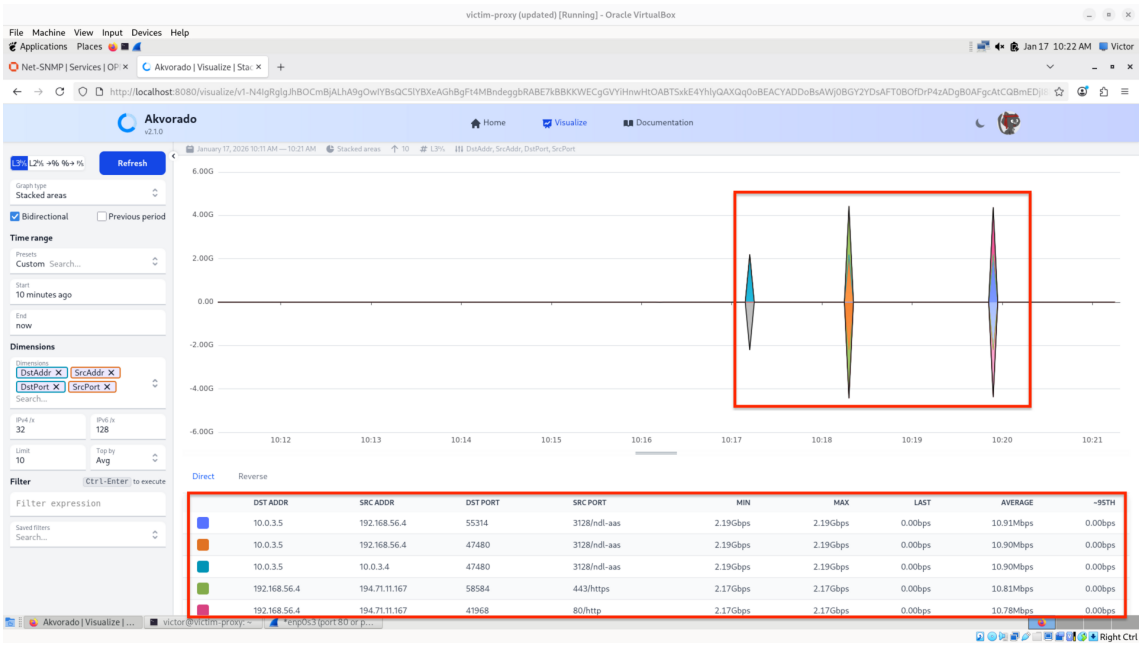


Figure 4.33: Akvorado Shows the Correlation

It appears that we can successfully identify a traffic correlation in the picture above. As it shows, the ingress traffic is around 2.19 GBps, and the egress traffic is around 2.17 GBps. However, even if we successfully capture the correlated traffic between ingress and egress

that the attacker made, we will not stop here. We try to make this simulation more realistic by creating random traffic from the victim's desktop itself to some random websites.

As a first idea, we run a shell script on the victim machine using cURL to generate traffic to many websites. The purpose of this is to simulate traffic from the victim to the Internet, which covers the proxy traffic.

```
#!/bin/bash

WEBSITES=(
"https://www.google.com/"
"https://en.wikipedia.org/wiki"
"https://instagram.com/"
"https://facebook.com/"
"https://netflix.com/"
"https://www.post.ch/"
"https://x.com/"
"https://www.zhaw.ch/"
"https://de.wikipedia.org/"
"https://pinterest.com/"
)

while true; do
    for URL in ${WEBSITES[@]};
    do
        curl -L ${URL}
        sleep 1
    done
    sleep 5
done
```

Second, we will also automatize the generation of traffic from the attacker using his proxy to access the Internet through the victim's network. In order to do so, run a shell script on the attacker's machine.

```
#!/bin/bash

while true
do
    curl --proxy 10.0.3.4:3128 --insecure https://10.0.3.7
    curl --proxy 10.0.3.4:3128 --insecure -O https://10.0.3.7/debian.bin
    rm debian.bin
    sleep 0.5
done
```

We created both programs with non-stop looping because we want to create a simulation of random Internet usage. Now we can inspect the ingress and egress traffic from both the source, attacker, and victim in Akvorado.

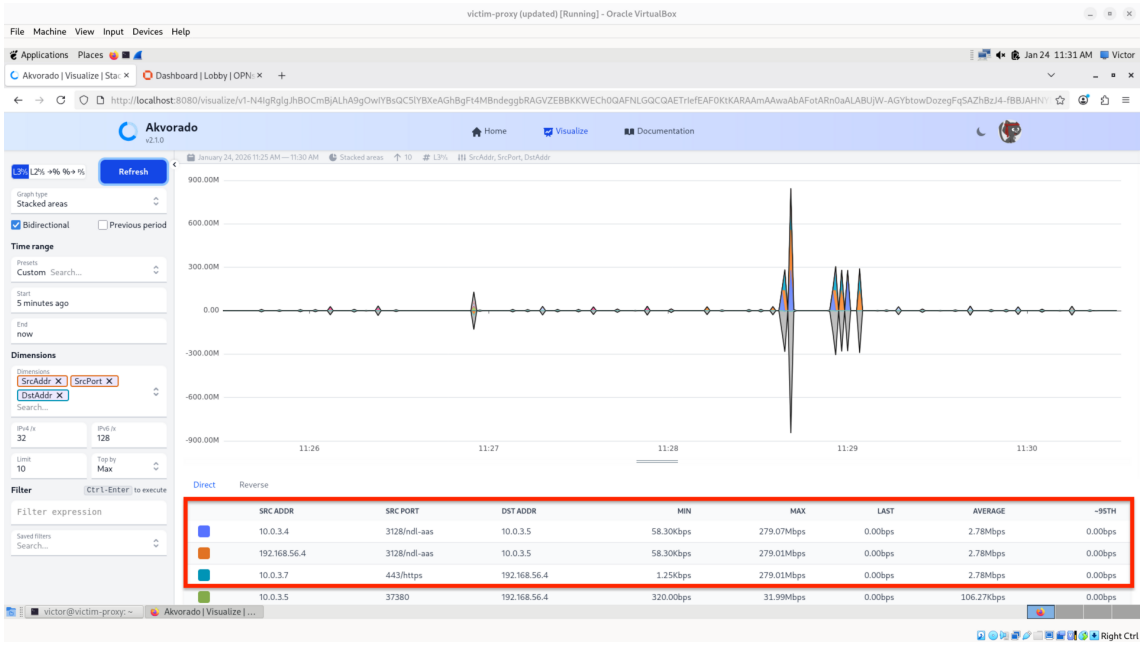


Figure 4.34: Akvorado Displaying Traffic

As we see from the figure above, we can detect that the traffic from the attacker to the proxy and from the proxy to the target correlates. We can use the known attacker IP address and the Squid proxy port (TCP 3128) as an indication to evaluate whether our guess based on the traffic pattern is correct.

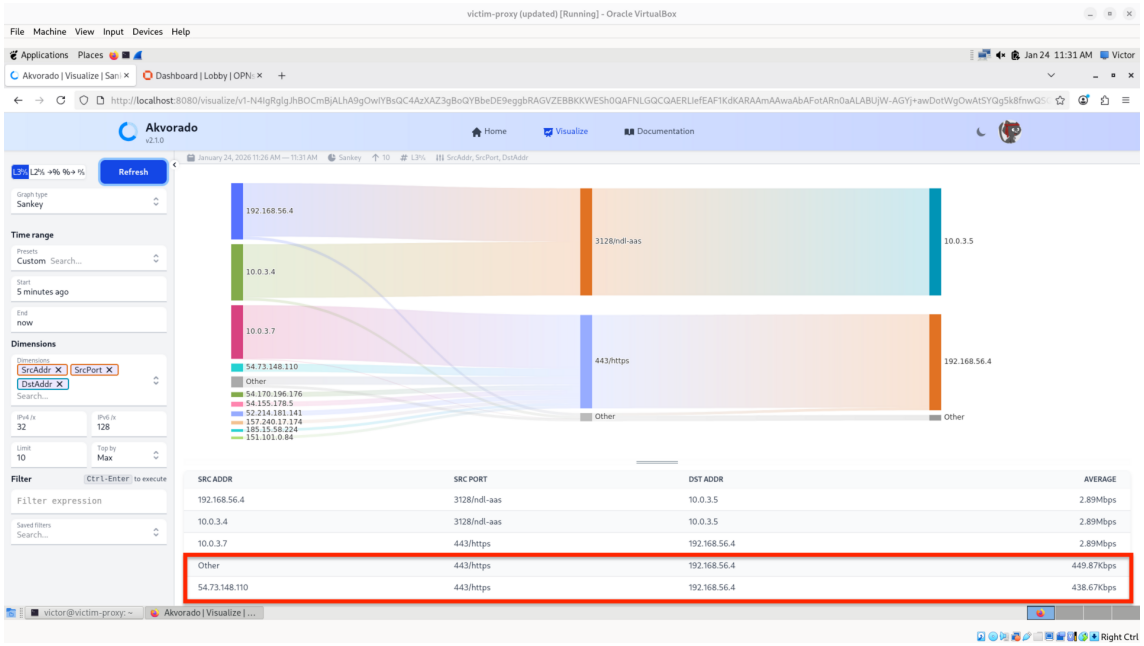


Figure 4.35: Akvorado Displaying Traffic

The figure above also shows that we successfully captured the other traffic that the victim generates as well. However, keep in mind that in the real-life scenario, this could be much

more difficult to distinguish, as we don't know the real attacker's IP addresses, as well as the TCP ports he is using in his potentially unwanted software.

5 Conclusion and Discussion

5.1 Discussion

5.1.1 Limitation of Packet Inspection

In essence, we can successfully detect the proxy connection with Suricata rules, when Suricata give us alert whenever we see known proxy TCP ports. However, there are numerous limitations to this packet inspection. For instance, it cannot detect residential proxies which do not use a known proxy ports. Dynamic detection is a more sophisticated approach which is based on analysis of the correlation of incoming and outgoing traffic. However, we found that Suricata is too limited to perform correlation analysis because Suricata rules are always stateless.

Another idea for this approach is use Lua scripting for detection in Suricata. Lua is a strong, effective, compact, and easy-to-embed programming language. Suricata supports Lua scripting for packet analysis as one of its features. The idea is we can write scripts to detect suspicious connections based on additional properties of the packets. However, the Suricata version included in OPNsense does not include or enable Lua Scripting feature. The Lua scripting also does not change the limitation that Suricata is stateless.

Because of the limitations of stateless detection, we were looking for strategies that can correlate incoming and outgoing traffic over time. One idea was to send the alerts generated by Suricata to a Security Information and Event Management (SIEM) tool such as Wazuh. However, this idea was not investigated because we expect the logs to be too big to handle since every IP packet will generate a log entry.

From that, the idea of using NetFlow came up. NetFlow (and its successor IPFIX) is a protocol to collect aggregate information from firewalls and routers and analyze them in a central server. We extended our setup with an Akvorado instance on the victim side and set up the OPNsense firewall to send its NetFlow data there. Akvorado collects the NetFlow data and enriches it with further information which it gets via SNMP. The result is a database of network flows which we can query in order to identify incoming and outgoing flows in order to detect suspected residential proxies.

However, bear in mind that this correlation has a significant risk of *false positive* detection. Because of this, we did not implement an automatic block of detected proxies. We think that it is better to alert the network admins so they can investigate and take action such as inform the owner of the machine which is suspected to have a unwanted proxy.

5.1.2 Future Work

As future work we propose to follow a combined approach of detection obvious proxies with Suricata rules for TCP ports and HTTP methods and identify suspicious network flows using Akvorado or a similar solution. NetFlow also gives the option to collect network flows from multiple routers and firewalls in a larger organization, which suits our goal well.

5.2 Conclusion

In essence, we successfully detected a residential proxy with Suricata rules in OPNsense and caught the proxy flow with NetFlow in Akvorado. However, more work is required to validate and implement the proxy detection strategies in real-life scenarios. The question of "Can we really detect residential proxies?" is still remaining a challenging task.

Bibliography

[1] E. Khan, E. Chiapponi, M. Verkleij, A. Sperotto, R. Van Rijswijk-Deij, and J. Van Der Ham-De Vos, "A First Look at User-Installed Residential Proxies From a Network Operator's Perspective," in *2024 20th International Conference on Network and Service Management (CNSM)*, Prague, Czech Republic: IEEE, Oct. 28, 2024, pp. 1–9, ISBN: 978-3-903176-66-9. DOI: 10.23919/CNSM62983.2024.10814519. Accessed: Oct. 3, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10814519/.

[2] D. Group. "What is an ORB network?" DIGITPOL, Accessed: Oct. 27, 2025. [Online]. Available: https://digitpol.com/what-is-an-orb-network/.

[3] "The Rise of Residential Proxies as a Cybercrime Enabler | Trend Micro (US)," Accessed: Oct. 27, 2025. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/the-rise-of-residential-proxies-and-its-impact-on-cyber-risk-exposure-management.

[4] J. Choi et al., "Understanding the Proxy Ecosystem: A Comparative Analysis of Residential and Open Proxies on the Internet," *IEEE Access*, vol. 8, pp. 111 368–111 380, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3000959. Accessed: Nov. 5, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/9115039/.

[5] D. Journal. "Proxy vs. VPN: The Ultimate Comparison," Medium, Accessed: Nov. 5, 2025. [Online]. Available: https://medium.com/@datajournal/proxy-vs-vpn-comparison-aeaee29712c3.

[6] X. Mi et al., "Resident Evil: Understanding Residential IP Proxy as a Dark Service," in *2019 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA: IEEE, May 2019, pp. 1185–1201, ISBN: 978-1-5386-6660-9. DOI: 10.1109/SP.2019.00011. Accessed: Oct. 3, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/8835239/.

[7] "Become a Peer - Proxyrack," Accessed: Oct. 6, 2025. [Online]. Available: https://www.proxyrack.com/become-a-peer.

[8] 911proxy. "Make Money with Proxies - Exploring Proxy-Based Income Opportunities," 911proxy, Accessed: Oct. 6, 2025. [Online]. Available: https://www.911proxy.com/make-money-with-proxies/.

[9] A. Tosun, M. De Donno, N. Dragoni, and X. Fafoutis, "RESIP Host Detection: Identification of Malicious Residential IP Proxy Flows," in *2021 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, USA: IEEE, Jan. 10, 2021, pp. 1–6, ISBN: 978-1-7281-9766-1. DOI: 10.1109/ICCE50685.2021.9427688. Accessed: Oct. 3, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/9427688/.

[10] L. Gorgo. "Proxy detection methods," Medium, Accessed: Nov. 3, 2025. [Online]. Available: https://leonidasgorgo.medium.com/proxy-detection-methods-a2b33355a99a.

[11] "Industry leading IP Geolocation and Online Fraud Prevention | MaxMind," Accessed: Nov. 3, 2025. [Online]. Available: https://www.maxmind.com/en/home.

[12] IP2Proxy. "IP2Proxy Proxy VPN Detection - Check Proxy & VPN using IP Address," ip2proxy.com, Accessed: Nov. 3, 2025. [Online]. Available: https://www.ip2proxy.com/.

[13] M. Pannu, B. Gill, R. Bird, K. Yang, and B. Farrel, "Exploring proxy detection methodology," in *2016 IEEE International Conference on Cybercrime and Computer Forensic (ICCCF)*, Jun. 2016, pp. 1–6. DOI: 10.1109/ICCCF.2016.7740438. Accessed: Oct. 28, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/7740438/.

[14] *Device fingerprint*, in *Wikipedia*, Oct. 20, 2025. Accessed: Nov. 3, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Device_fingerprint&oldid=1317780723.

[15] "What Is a Honeypot?" Palo Alto Networks, Accessed: Nov. 3, 2025. [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/honeypots.

[16] "What Is Deep Packet Inspection (DPI)?" Fortinet, Accessed: Nov. 3, 2025. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/dpi-deep-packet-inspection.

[17] "About Oracle VirtualBox," Accessed: Nov. 3, 2025. [Online]. Available: https://www.virtualbox.org/manual/topics/Introduction.html.

[18] M. Balaban, *Beginner's Guide to OPNsense*, 1st Edition. Zenarmor, 2024, 253 pp. [Online]. Available: https://www.zenarmor.com/opnsense-beginner-guide-form?utm_source=website&utm_medium=opnsenseforum&utm_campaign=beginnersguide.

[19] "Netflow Export & Analyses — OPNsense documentation," Accessed: Jan. 16, 2026. [Online]. Available: https://docs.opnsense.org/manual/netflow.html.

[20] "Akvorado | Documentation | Introduction," Accessed: Jan. 16, 2026. [Online]. Available: https://demo.akvorado.net/docs/intro.

[21] "Squid : Optimising Web Delivery," Accessed: Nov. 21, 2025. [Online]. Available: https://www.squid-cache.org/Intro/.

[22] "8. Suricata Rules — Suricata 9.0.0-dev documentation," Accessed: Dec. 2, 2025. [Online]. Available: https://docs.suricata.io/en/latest/rules/index.html#.

[23] Insane Cyber, director, *Writing Suricata Rules: Understanding The Basic Rule Format*, Apr. 26, 2022. Accessed: Nov. 18, 2025. [Online]. Available: https://www.youtube.com/watch?v=d-t-8lJ2PII.