

Detecting Residential Proxies at the OPNsense Gateway.

Zurich University of Applied Sciences
School of Engineering

Authors	S. Selviyani
Supervisors	Dr. Peter Heinrich
Study Program	InfoCyberSec
Submitted on	Spring Semester 2026

1 Introduction

A residential proxy (RESIP) is a service offered by third parties that allows users to navigate the Internet through other people's IP addresses. The third party offers a proxy service that utilizes a residential IP allocated by a household device, e.g. a computer, smartphone, smart TV, or home router. In many cases, the proxy user may gain access to the household residential IP address regardless of the owner's knowledge or consent, in spite of several residential proxies service providers claiming that they use the residential IP in an ethical manner and with the owner's approval [1]. Besides, RESIP can be exploited to make Internet traffic appear as if it is originating from and destined for the residential IP address.

The third party offers this service for a cost that ensures safety and includes an anti-detection guarantee for its customers, which we can see in their FAQ [1]. In contrast, there is no assurance provided to the owner of the household devices who lends their IP address for RESIP purposes regarding their safety. For instance, this RESIP can assist cybercriminals in bypassing anti-fraud systems, more precisely targeting businesses, and performing many other tasks. This can bring the blame to the the residential IP address owner when something goes wrong, such as when an attack or fraud happen. Furthermore, there are several purposes for using a RESIP, including [2]:

- Bypassing geo-blocking restrictions
- Data extraction for market monitoring and competitive evaluation
- Data extraction for the development of large language models (LLMs) in Artificial Intelligence
- Violation of Copyright and other terms
- 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) [3] used by Advanced Persistent Threat (APT) organizations

This project addresses the challenge of detecting residential proxy traffic from a network gateway, 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, and it is usually hard to distinguish from legitimate traffic. Therefore, it is challenging to identify whether the suspicious activity is being conducted by the attacker or the victim.

The purpose of this experiment is to develop a detection methods and tools which can be used inside the network that will work together with the existing network gateway (i.e. router or firewall) to record and analyze the packet and flow metadata in order to detect anomalies that indicate RESIP. As a prototype, I will implement a detection machine to receive and process the network traffic report from OPNsense (router and firewall) to detect the suspicious IP address that is indicated as the broker of the residential proxy provider.

The detection methods are based on previous research of residential proxies [4] [5] that has been published in 2022 and 2024. However, the author will continue to improve the adaption

of the existing methods that the previous research has done to be more practical in the application of organization network firewalls. At the end of this report, the author will present simulation results, using a confusion matrix to summarizes the results of several simulation tests with virtual machines and real-world tests.

2 Technical Background

2.1 Proxies

Proxy servers are servers which operate as an extension and gateway between clients and servers on the Internet. From the server perspective, the proxy appears to be the client. Proxy servers are especially popular in the context of HTTP.

There are several benefits to proxies for protecting users' privacy, and for bypassing server-side blocking and regional censorship. But there are also drawbacks, such as enabling bad actors to use a proxy to cover up their identity (e.g. IP address) and perform attacks. As a result, many websites restrict proxy access to them, leading to the creation of blacklists to filter out and ban known proxy servers [6]. In addition, caching performed by a proxy server minimizes the necessity for direct interaction between clients and remote servers, preventing network bottlenecks.

2.2 Residential Proxies (RESIP)

A 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 with known proxy blacklists. As stated in the introduction, a residential proxy service might be a tool to camouflage unwanted or even criminal activities. A large discussion started in 2019 when Mi et al. brought up the residential proxy as a dark service topic in their research paper [7].

Furthermore, RESIP providers claim to collect residential IP addresses under consent, and their entire proxy network is completely opt-in only, as stated in their FAQ [1]. Other 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) [8]. However, the lack of regulation allows for varying definitions of ethical procurement. Of course, these companies of residential proxy providers sound legitimate. They use the terms "market search" or "ad verification" [9] 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 [5].

To illustrate this illegal activity, we can consider the residential proxy victim, who was not aware of the presence of a PUP (potentially unwanted 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 by 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 organization (e.g. school or company) or 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.

As the author discusses in the first part of *Detecting Residential Proxy Operations in Network Traffic* research, there are several methods to detect proxy behavior on the network, according to the article written by Leonidas Gorgo on his Medium site [10], such as:

- IP Address Analysis
- Reverse DNS Lookups
- Latency Assessments (with Ping and Traceroute)
- HTTP Header Analysis
- Device Fingerprinting
- WebRTC Leak Detection
- TLS Fingerprinting
- Machine Learning Models
- Behavioral Honeypots
- Deep Packet Inspection (DPI)
- Behavioral Analysis

2.3 Technology Overview

The author will use several of the same tools for this experiment as in the first part of the *Detecting Residential Proxy Operations in Network Traffic* paper. However, many of the tools that I have used before are not really successful for my case to detect residential proxies, like Suricata and Akvorado. For those reasons, I will use tools that are only for collecting packets and flows such as *tcpdump* and *nfdump*.

2.3.1 Oracle VirtualBox

Oracle VirtualBox is a multi-functional virtualization application that operates across various platforms, allowing a computer to run multiple operating systems at once within virtual machines (VMs). Oracle VirtualBox also has other advantages over virtualization tool such as QEMU (UTM), e.g. the snapshot feature which is very useful for experimental setups. For

my use case, VirtualBox has particularly convenient features for defining virtual networks.

2.3.2 Debian 13

Debian 13 Trixie was released on September 6, 2025. In this experiment, I used the Debian operating system for the host as well as for the VM guests. It is essential to recognize that I require a minimum of five virtual machines to act as the actors (attacker, response providers, detection machine, victim, and unrelated user) in my threat simulation.

2.3.3 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. The firewall is the only VM in my setup which is not running Debian Linux.

2.3.4 tcpdump and nfdump

`tcpdump` is a command-line tool for collecting packets and saving them in a file for later analysis. The resulting PCAP files can be processed with a variety of tools including the popular Wireshark. In my detection, I use the Python Scapy library to analyze the recorded packets.

`nfdump` is a command-line tool included Debian to collect and process NetFlow information from the network. `nfdump` reads the flow data from files, typically created by any nfdump collector such as nfcapd, nfpcapd, and sfcapd. In this experiment, I will use both tools to collect packets and flows for the analysis.

3 Existing Methodologies and Continuing Improvement

A variety of methodologies have been developed in previous research to detect residential proxies. The researchers have reported success in identifying residential proxies through various methodologies, including the application of machine learning and a built-in detection program based on the Python programming language. This chapter will concentrate on two previously published methodologies available on the Internet, which will serve me as the framework for my ongoing research efforts.

3.1 Exisiting Detection Methodology with Machine Learning Overview

A detection method of residential proxies with machine learning is proposed by Mi et al. in their research paper titled “Shining Light into the Tunnel: Understanding and Classifying Network Traffic of Residential Proxies” (2024) [4]. In their research, Mi et al. introduce a comprehensive framework for setting up RESIP nodes and collecting RESIP traffic. In addition, using various machine learning-based RESIP traffic classifiers that provide timely and precise detection of specific traffic flow that classified as RESIP traffic.

In order to collect large-scale and realistic RESIP traffic data for the traffic collection, the researchers mainly focus on three major RESIP bandwidth sharing (residential proxy) providers, which are PacketStream, IPRoyal, and Honeygain. First, the researchers need to

implement multiple RESIP nodes, which are referred to as "bandwidth-sharing" nodes. They established the bandwidth-sharing nodes in two primary locations: United States and China. That implementation comprised a central dispatcher, also referred to as *the controller*, in addition to several nodes running as distributed agents.

To capture residential proxie traffic, the researchers implemented the RESIP node collection framework as shown in figure 3.1.

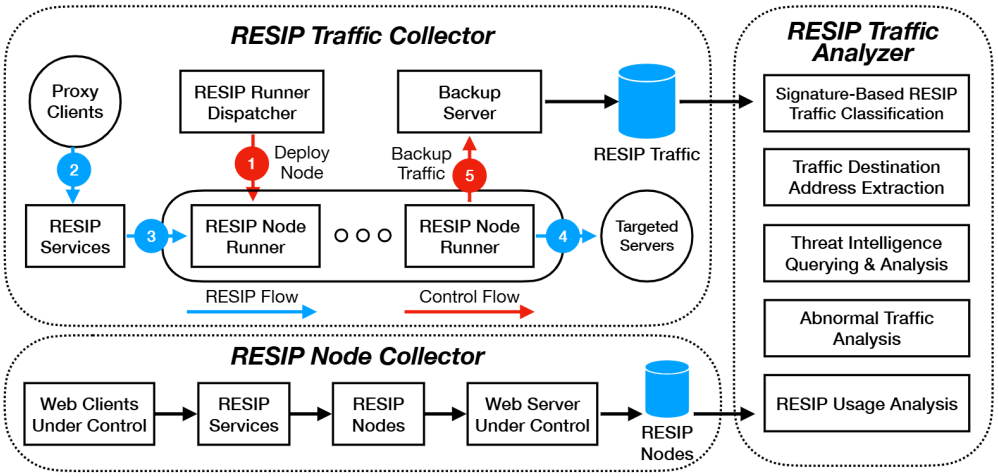


Figure 3.1: Methodologies of RESIP Detection by Mi et al. [4]

As shown in the figure above, the initial step in capturing RESIP nodes involves the use of web clients and web servers. Afterwards, probes will be sent out as straightforward HTTP requests originating from web clients. These requests will be relayed through proxy gateway servers, ending at RESIP nodes, before arriving at the web servers. In this process, the web servers will have the capability to observe the public IP address of each participating RESIP node.

For the RESIP Traffic Analyzer, Mi et al. proposed several steps, which are:

Signature-Based RESIP Traffic Classification. In this step, they categorize captured RESIP traffic flows into three distinct groups: tunnel flows, relayed flows, and control flows. Tunnel flows are created between RESIP nodes and proxy gateway servers, while relayed flows indicate the traffic directed towards various destinations. Additionally, control flows enable communication between RESIP nodes and associated services. A signature-based classifier is employed to identify relayed flows, using service-specific domain names and tailored protocols as signatures.

Traffic destination addresses extraction. In order to set up a comprehensive statistical evaluation of relayed RESIP traffic, it is important to extract destination addresses, which include IP addresses, TCP or UDP ports, fully qualified domain names (FQDNs) and URLs. In this step, the extraction of the IP and transport-layer port from packets provides a limited understanding of the activities occurring within the flow.

Querying and analyzing threat intelligence of traffic destinations. In order to assess the maliciousness of the hierarchical destination addresses, the researchers also conducted

an analysis of threat reports sourced from various representative threat intelligence datasets, including VirusTotal, IBM X-Force, and URLhaus.

Abnormal traffic analysis. To evaluate this, the researchers implement a defined set of criteria aimed at identifying abnormal or suspicious traffic flows along with their corresponding destination addresses.

RESIP usage analysis. In this step, the researchers claim that a RESIP node is deemed to have relayed the infiltration traffic only when one or more pairs of traffic flows from the two datasets are precisely matched regarding the traffic destination, timestamp, and traffic size.

3.1.1 RESIP Machine Learning-Based Classification

The researchers state, in the context of the established ground truth, they explore two distinct machine learning paradigms for both classification tasks. One approach involves fine-tuning a pre-trained transformer-based model, this enabling the automatic encoding of traffic flow into a generic representation. Another approach involves the manual design of features and the application of traditional classification algorithms to develop the classification models.

Classifying RESIP flows through fine-tuning ET-BERT. The classification of RESIP flows through ET-BERT requires a process of fine-tuning especially focused on distinguishing between relayed and tunnel flow classifications. A BERT-like pre-trained model for generic network traffic representation. The researchers state these transformer-based models rely on only the initial five packets for the prompt identification of RESIP traffic.

Classifying RESIP flows with expert-crafted features and classic ML algorithms. Throughout the training and evaluation phases, various traditional classification algorithms have been examined by them, including decision trees, support vector machines, and random forests. The researchers conducted experiments with a range of feature engineering configurations, especially exploring various combinations of values, to determine the optimal packets required for an effective classification model. As a result, the random forest (RF) algorithm has consistently demonstrated outstanding performance and is therefore selected as the default algorithm for classification tasks.

Their research results are shown in figure 3.2.

The performance of various classification models for RESIP flows detection.

Model	Precision	Recall	F1-Score	FPR[1]
TF-BERT[2]	0.9036	0.8891	0.8963	0.0535
RF-BERT[2]	0.9287	0.9304	0.9296	0.0396
TF-RF[2]	0.9571	0.9139	0.9350	0.0223
RF-RF[2]	0.9620	0.8880	0.9235	0.0191

[1]FPR stands for the false positive rate.

[2]TF-BERT is downstream task that fine-tunes the ET-BERT model for tunnel flow classification. RF-BERT is downstream task that fine-tunes the ET-BERT model for relayed flow classification. TF-RF denotes Random Forest classifiers specialized in tunnel flow classification, while RF-RF denotes Random Forest classifiers designed for relayed flow classification.

Figure 3.2: The Result of RESIP Detection by Mi et al. [4]

3.2 Exisiting Detection Methodology with PCAP File Analysis Overview

The detection of residential proxies with PCAP file analysis is proposed by Tosun et al. on their research paper with the title "RESIP Host Detection: Identification of Malicious Residential IP Proxy Flows" (2022). In their research paper, Tosun et al. introduced their detection mechanism that consists of three anomaly detection algorithms. Tosun et al. predicted that the proxy traffic will exhibit a particular sequence among the packets within the network traffic flow. The suspicious flow is likely to be visible on the residential hosts that are acting as a proxy in a proxy network. The flow that generates suspicion can be described as follows:

- The client must transmit to the proxy the details concerning the web that it is trying to get access to.
- As the resource is retrieved, a stream of packets is expected to be generated between the compromised host and the web server.
- The compromised host is required to relay the resource obtained from the web server back to the client.

The methodology for identifying flows that show these suspicious patterns is defined in Algorithm 1. Simply stated, the first algorithm is identifying the two-way connections, which are incoming and outgoing traffic that happens on the observed devices with other IPs, where the response time (in terms of packet steps) to an incoming packet is too long and exceeds the threshold that the program defined.

Along with another expected characteristic of a proxy server network flow is that the volume of data transmitted back to the client must be at least equal to the data received from the target web server. That chapter can imply that the residential proxy providers may alter the data; however, these modifications are likely to be identified by standard proxy detection systems, thus decreasing the stealth of the residential proxies. In algorithm 2, the researchers introduce the methodology for the data check.

After the detection of a suspicious connection by Algorithm 1, which is indicated by the packet distance exceeding a fixed threshold, Algorithm 2 proceeds to analyze potential anomalies in data transmission within a specific time frame. This algorithm specifically evaluates two things; first, the total volume of data transmitted from the device to the identified suspicious destination IP is measured from the initial suspicious packet up to the specified time threshold. Second, the total data received by the device from any IP source, specifically the one with the most, during the period between the last incoming packet from that IP and the outgoing packet that triggers concerns. When the volume of incoming data is less than or equal to that of outgoing data, this scenario can be classified as a data breach, which may suggest the existence of residential proxies.

Algorithm 3 evaluates potentially suspicious connections identified by previous algorithm 1 and 2 through the examination of data volume and DNS queries. It determines whether a connection is normal or abnormal by evaluating the lack of recent DNS lookups, which are usually conducted prior to data transmission. If a significant amount of data is sent to an IP that has not been resolved in a recent period of time, it may indicate potential threats, including hardcoded IPs, established command and control communication, or attempts to

circumvent DNS monitoring, thereby increasing the suspicion surrounding the connection.

3.3 Continuing Improvement from Existing Methodologies

Despite the satisfying percentage results that the previous researchers showed, I initiatively improved the previous methods to make the detection of residential proxies more practical for the real-world case. In this experiment, I tested two data collections from packets by `tcpdump` and flows from `nfdump` to do packet correlation analysis. I feed the data to a Python program to detect the existing proxy behavior. To be highlighted here, I used the previous algorithms that Tosun et al. provide in their research paper [5], and then I improvise from the input and the output as desired.

In the beginning of this experiment, I plan to implement the PCAP file analysis by Tosun et al. [5] and the detection of residential proxies with machine learning analysis by Mi et al. from the research paper [4]. However, unfortunately, Mi et al. are not allowed to share with me the machine learning model that they used. I emailed them three times to ask for the guidelines and tips on how the authors can get residential proxy detection with machine learning with fine-tuning ET-BERT and machine learning algorithms. However, I got no results because they will not release the model that they used. For those reasons, I decided to focus on PCAP files and NetFlow analysis. Due to my limited machine learning experience, I decided to cancel the machine learning methodology.

4 Implementation and Evaluation

4.1 Simulation Setup

In this chapter, I discuss the improved setup of the RESIP simulation. I will add some new actors and remove one actor that existed in my previous setup. To make this simulation more intresting, I will create a story to illustrate this residential proxies simulation.

Alpha (the attacker) wants to use a residential proxy feature that Bravo (the broker) offers as his service. Bravo has successfully injected his malware into one of the computers in an organization network named ABC. The ABC organization has three users in their network: Romeo, Juliett, and Victor (the victim). Once upon a time, the ABC organization decided to put an OPNsense firewall on its network to protect and to monitor the traffic in their network. When the administrator of the ABC organization monitored their network activity on the OPNsense interface, a suspicious IP address popped up in the logs, see in figure 4.1. After the administrator has suspicion of malware in the organization's network, he decided to set up the Delta server as a detection machine to monitor the traffic on OPNsense for suspicious behavior.

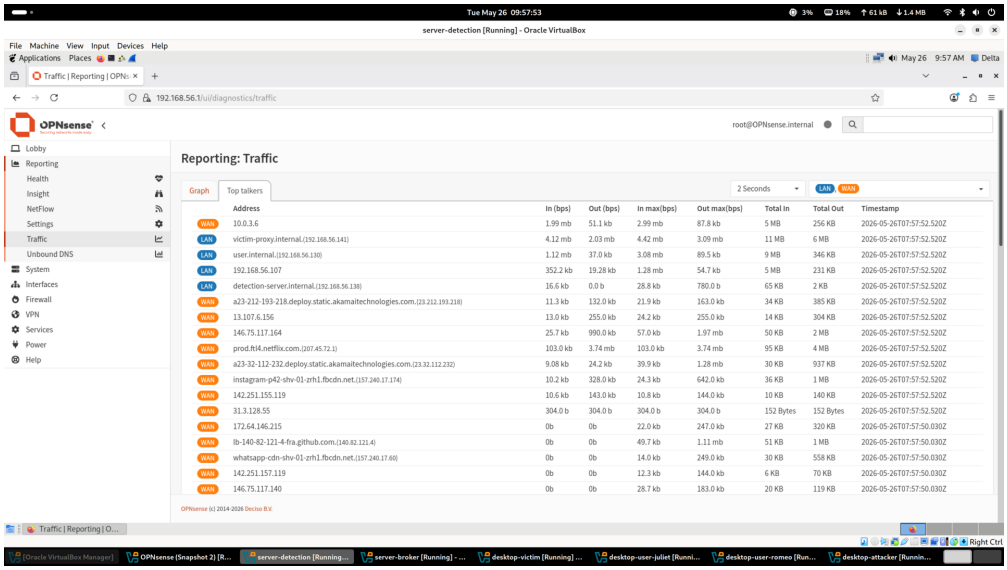


Figure 4.1: Suspicious Top-Talkers

4.1.1 Installing the Actors' Virtual Machines

As I stated in the story above, I need to add some new actors for this simulation scenario. There are four actors in this scenario that are newly added: Broker (Bravo), detection server (Delta), Romeo, and Juliett. The rest remain the same as in the previous research, which are the attacker (Alpha) and the victim (Victor). Besides, the author removed the dedicated target machine as the RESIP can be used to target any website. All VMs I installed for the actors are based on Debian Linux, except the firewall, which uses OPNsense (based on FreeBSD).

I split the actors' details based on the network they connect to: the ABC network or the public network, representing the Internet. The first machine in the ABC organization is the OPNsense firewall.

- Machine Name: OPNsense
- Username: root
- Role: Firewall and gateway
- Operting System: OPNsense (based on FreeBSD)
- IP Address Internal (LAN): 192.168.56.1/24
- IP Address External (WAN): 10.0.3.4/24
- Internal Network Adapter: opnsense-lan
- External Network Adapter: public-net

Next, I installed the detection server which I used to run the detection algorithms on the data which was collected by the OPNsense firewall.

- Username: Delta
- Role: Detection server
- Machine Name: server-detection
- Operting System: Debian 13 Server and Desktop

- IP Address: 192.168.56.138/24
- Network Adapter: opnsense-lan

The next step, I set up three actors to simulate the users in the network of an organization. One actor named Victor is the victim who will have the RESIP malware installed.

- Username: Victor
- Role: Victim
- Machine Name: desktop-victim
- Operting System: Debian 13 Desktop
- IP Address: 192.168.56.141/24
- Network Adapter: opnsense-lan

The other two machines support me to generate traffic in the network, so I have noise in my network traffic to challenge the detection.

- Username: Romeo
- Role: User
- Machine Name: desktop-user-romeo
- Operting System: Debian 13 Desktop
- IP Address: 192.168.56.130/24
- Network Adapter: opnsense-lan

- Username: Juliett
- Role: User
- Machine Name: desktop-user-juliett
- Operting System: Debian 13 Desktop
- IP Address: 192.168.56.107/24
- Network Adapter: opnsense-lan

The second network of my simulation is the “public-net”. This network is outside of my OPNsense firewall and represents the Internet. There are only two actors in this network. Alpha is the attacker who uses the RESIP to hide his traffic.

- Username: Alpha
- Role: Attacker
- Machine Name: desktop-attacker
- Operting System: Debian 13 Desktop
- IP Address: 10.0.3.5/24
- Network Adapter: public-net

The second actor in the public network is Bravo. Bravo is the broker who sells his residential proxy service. The malware in Victors server will connect to it to enable the RESIP. Alpha will connect to it to use Victor’s computer in the ABC network as a proxy.

- Username: Bravo
- Role: Residential Proxy Broker
- Machine Name: server-broker
- Operting System: Debian 13 Server
- IP Address: 10.0.3.6/24

- Network Adapter: public-net

After all the machines are completely installed and running, the Virtual Box manager gives a list of all machines as in figure 4.2

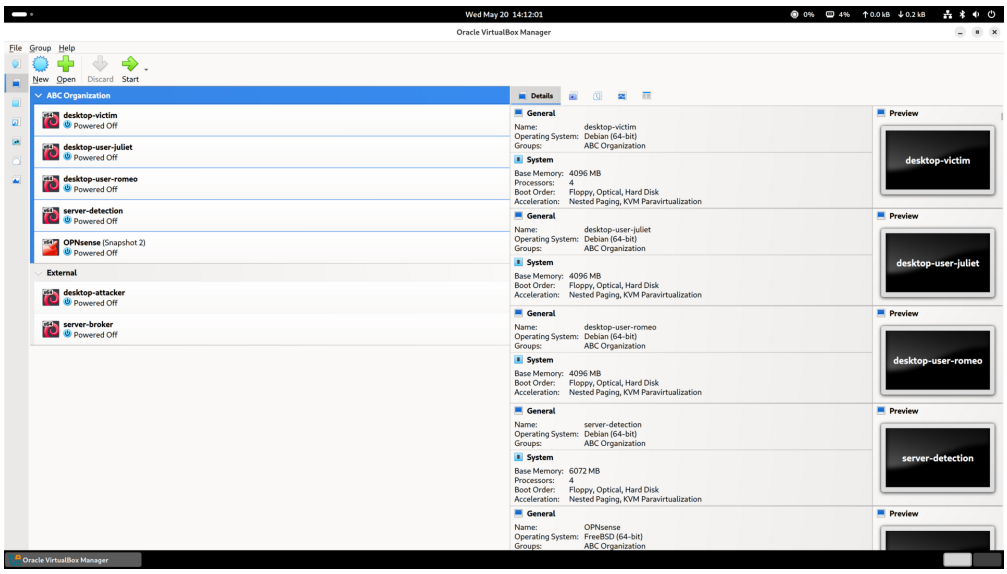


Figure 4.2: Virtual Machines Details in ABC Organization

4.1.2 Network Setup

At this step, I need to set up the LAN network configuration in OPNsense. I used the DHCP service in OPNsense so that all devices in the ABC organization automatically get IP addresses.

For that, I set up the DHCP ranges in OPNsense in: `OPNsense > Service > Dnsmasq DNS & DHCP > DHCP range` as shown in figure 4.3.

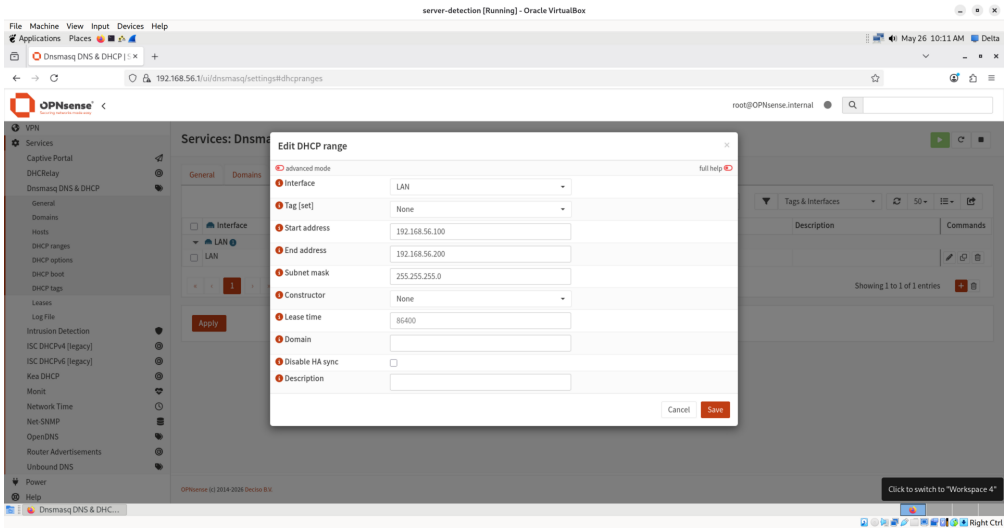


Figure 4.3: Setting Up DHCP Range on OPNsense

Next, I enabled DHCP in `Dnsmasq DNS & DHCP`, as shown in figure 4.4.

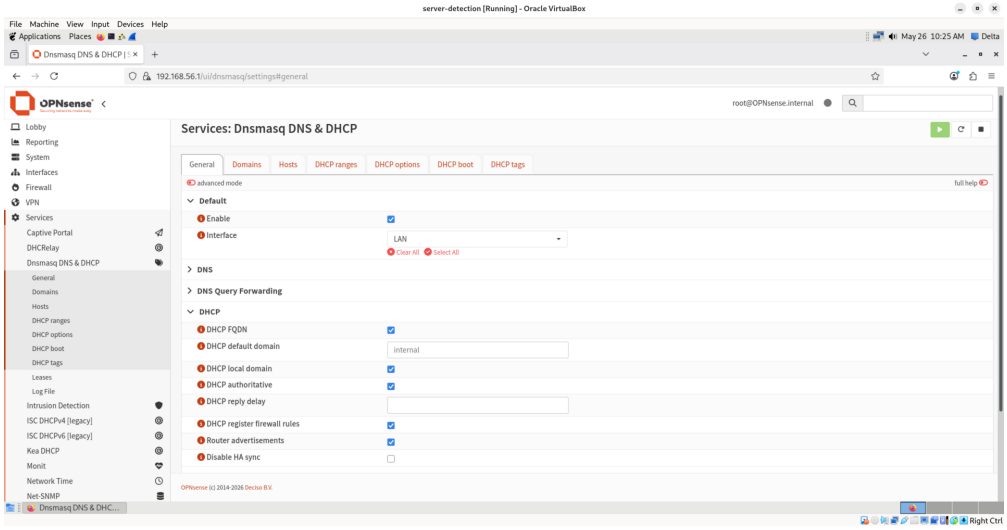


Figure 4.4: Setting Up DHCP in General

As next step, all the machines are connected to the correct virtual network in VirtualBox. As I mention before, I will build two different networks here. First the NAT Network which I name `public-net`. This network has a role as the public network, where Alpha and Bravo are connected, as well as the WAN interface of OPNsense.

The other network is Internal Network which I name `opnsense-lan`. This network has a role as ABC organization network, where all the machines of ABC members are connected as well as then LAN interface of OPNsense.

The configuration for NAT and Internal Network are shown in figures 4.5 and 4.6, respectively.

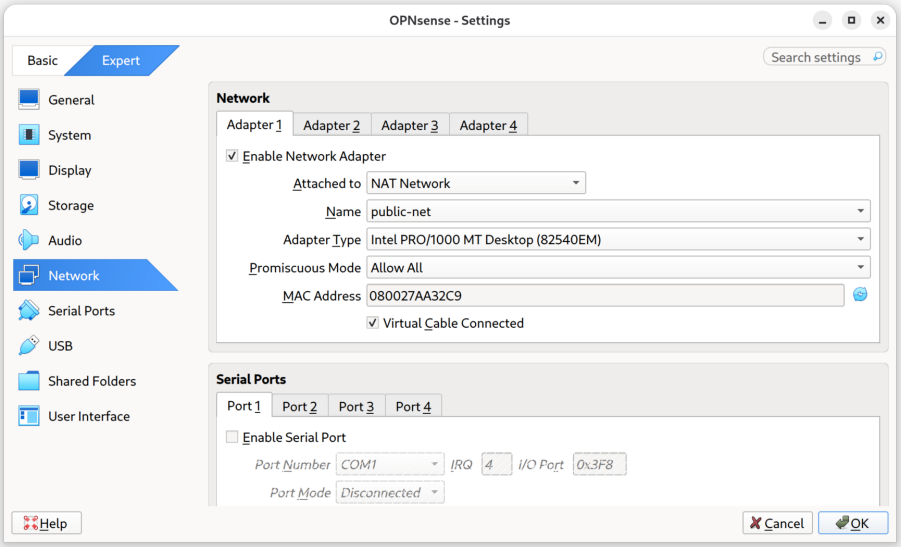


Figure 4.5: Setting Up NAT Network Adapter in Virtual Box

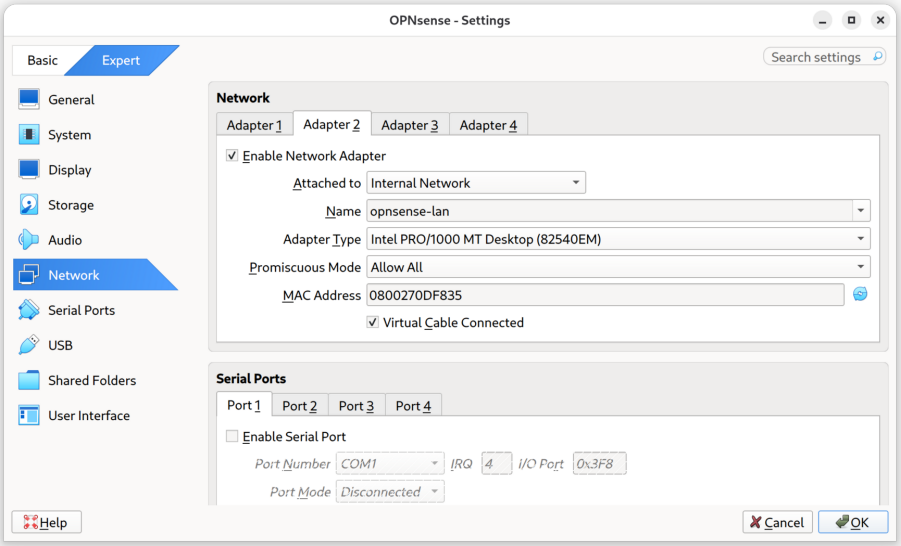


Figure 4.6: Setting Up Internal Network Adapter in Virtual Box

Figure 4.7 gives an illustration of the final simulation setup and its corresponding IP addresses.

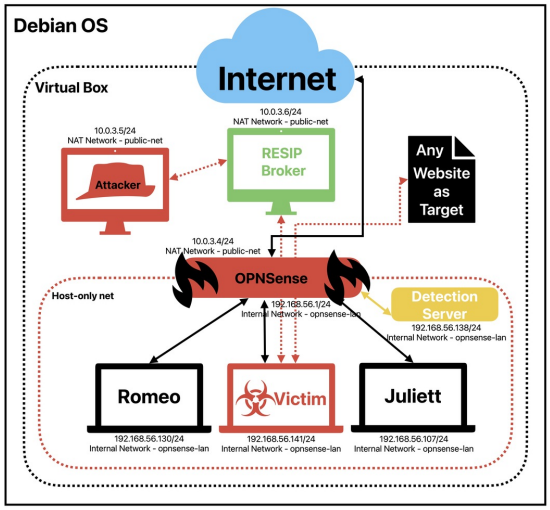


Figure 4.7: Illustration of the Full Setup

In total, I have four desktop virtual machines, two server virtual machines and one firewall which is the gateway router.

4.1.3 Reverse Tunneling Setup

At a high level, I will be using two types of tunnels to connect the malware on the victim machine with the RESIP broker.

The test malware script uses SSH with a reverse tunnel to connect to the broker, enabling the RESIP clients to connect to a local Squid proxy on the victim machine. However, I want to hide all these connections and protocols from the firewall and other network monitoring. To hide it, I establish a TLS connection on TCP port 443 with `socat` and `tunnel SSH` through it. In this way the TCP ports and handshakes will look identical to HTTPS connections.

On the broker, I run `socat` as a server and enable reverse tunneling in the SSH configuration. Figure 4.8 illustrates the layering of the tunnels.

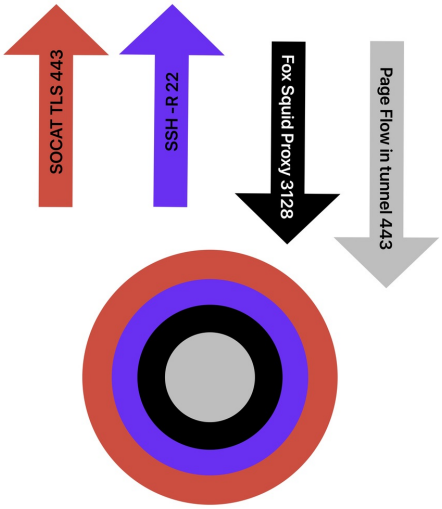


Figure 4.8: Reverse Tunnel

To implement this, I installed `socat` and `openssl` on the broker machine. A little script starts a listener with socat with debug logging (`-dd`). It is executed as root in order to easily access TCP port 443. The fork option allows to start a new process for each connection a client makes. The `cert` option is required for TLS in order to define the certificate. I use the self-signed snakeoil certificate which Debian generated automatically (`ssl-cert-snakeoil`). `ssl-cert-snakeoil.key` and `ssl-cert-snakeoil.pem` key have to be combined into one file, which resulted in `combined.pem`. I set the `ciphers` option to `HIGH` and the `verify` option to 0, which means the server does not need to verify any client certificate. Incoming connections are forwarded to `localhost:22`, where the SSH daemon is listening.

The broker setup can be seen in the following bash script.

```
#!/bin/bash

socat -dd OPENSSL-LISTEN:443,fork,cert=/root/combined.pem,ciphers=HIGH,verify=0 \
TCP:localhost:22
```

Next, I need to set up the test malware in the victim machine which connects to the broker via `socat` and `SSH`.

```
#!/bin/bash

socat -d -d TCP-LISTEN:2222,fork OPENSSL:broker:443,verify=0 &
sleep 1
ssh -p 2222 -R *:3128:localhost:3128 localhost
sleep 1
killall socat
```

The malware on the victim's machine connects to `broker:443` with socat. It uses `verify=0` because of the server's self-signed certificate is cannot be verified. At the end of the command line, `&` is used to make it run in the background while the next command is executed. On localhost, socat will listen to TCP port 2222 and forward it through the tunnel to TCP port 22 on the broker.

Next, SSH connects to `localhost:2222`, which means it is forwarded to port 22 on the broker via the socat tunnel. SSH, creates a reverse tunnel with the option `(-R)` where TCP port `3128` is opened on the broker and connected to the Squid proxy on the victim machine (`localhost:3128`).

As last step, the attacker has to set up `broker:3128` as his HTTP proxy, e.g. in the Firefox web browser or in cURL.

An illustration of this reverse tunnel setup is shown in figure 4.9. The socat tunnel is drawn as a big red arrow. Within it, the SSH connection is shown in purple. Using these tunnels, the connection to the proxy is established, shown in black. Now the attacker can access any website using the RESIP, here shown as a grey arrow.

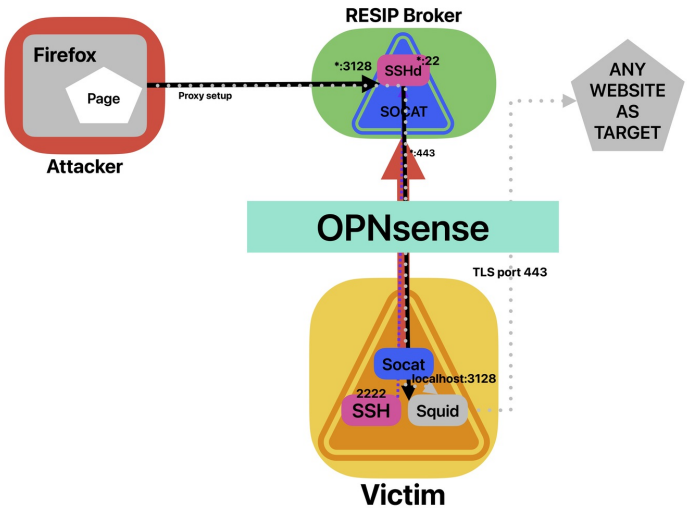


Figure 4.9: Reverse Tunnel Illustration

4.1.4 PCAP and NetFlow Setup

Before I start the simulation generating traffic from all the actors, I need to install and set up the PCAP and NetFlow tools in OPNsense and in the detection machine. The purpose is to record all the traffic coming and going through the firewall and transmit it to my detection machine.

For PCAP, `tcpdump` is already pre-installed on OPNsense, which can be run in a shell in the admin console. The result is shown in figure 4.10.

```
tcpdump -i em1 -w "/root/pcap/lan_%Y%m%d-%H%M%S.pcap" -s 64 -G 60
```

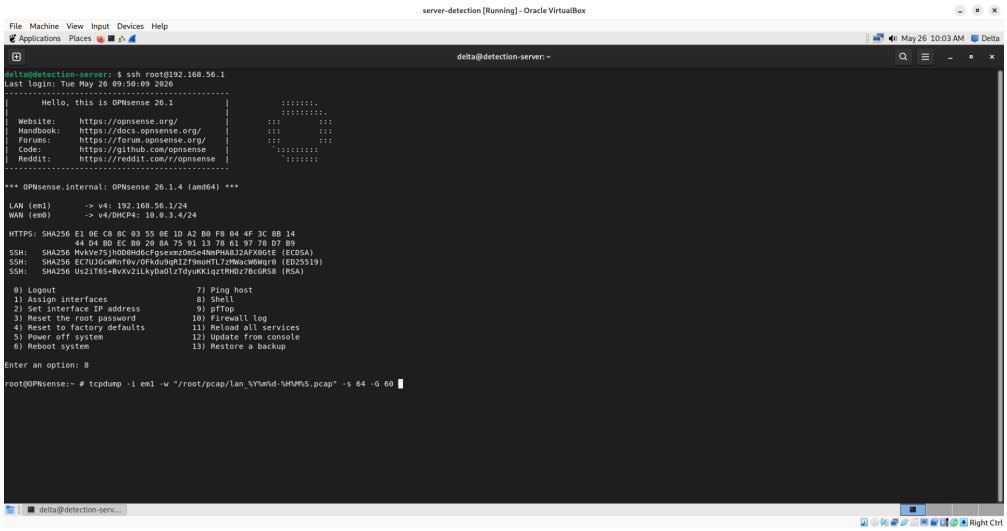


Figure 4.10: TCPdump Command in OPNsense

`em1` is the network interface used for the LAN. The option `-s 64` means that I want to crop each package at 64 bytes. This is useful for me because I want to reduce the file size of PCAP file, and this already provides me with enough information about protocol, source, destination and ports. The payload of the data is not helpfule as it is encrypted with TLS. `-G 60` means that I want to create a new PCAP file every 60 seconds. Instead of writing all traffic into one big PCAP file, I ask tcpdump to write packets for 60 seconds and continue capturing indefinitely with new files and fresh timestamp for every 60 seconds. The `-w` option will write the captured packets into a file instead of displaying them in the console. In this context, I ask tcpdump to write it to the file name that include the current date and time into the filename and place them in the `/root/pcap/` directory as shown in figure 4.11.

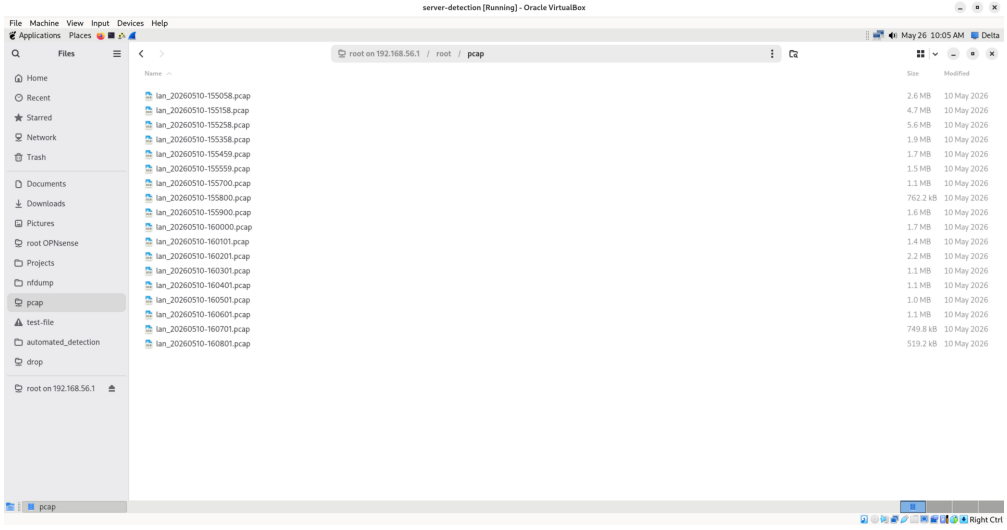


Figure 4.11: TCPdump Recorded Files

For NetFlow, `nfdump` is available in Debian Linux on the detection machine and can be

installed via APT. The Debian package includes the `nfcapd` service which opens UDP port 2055 to collect NetFlow data.

In the OPNsense user interface, I can set up the NetFlow reporting capture destination to the detection machine. This setting is located under: `Reporting > NetFlow > Capture`. I set up the destination address to the detection machine IP address, which is `192.168.56.138:2055` as shown in figure 4.12.

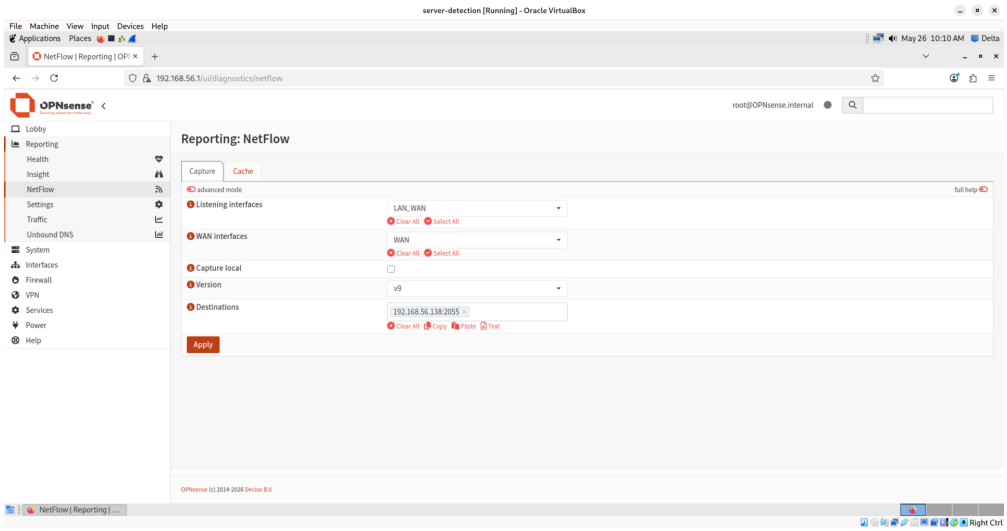


Figure 4.12: NetFlow Setup in OPNsense Interface

`nfcapd` is saving the NetFlow data in one report file for every five minutes, which can be seen in the file names. However, NetFlow has its own binary file format which is not readable by humans or the common tools I want to use. However, I can export it to a more portable format such as CSV. The default file location for nfcapd is in `/var/cache/nfdump/`.

4.1.5 Automated Traffic Generation

To create a scenario filled with active Internet traffic noise, I want to create a simulation in which the users in the ABC organization are browsing the Internet. Victor, Romeo, and Juliett use their devices to browse various websites they like on a daily basis.

To simulate regular user browsing behavior, Firefox has been set up on all user machines. Then, Youtube was opened with autoplay which loads one music video after another. On the attacker machine, the broker was set as proxy in Firefox.

To simulate background connections by apps, I used some Python scripts which connnect to random URLs with random time patterns. In the case of the attacker, the Python script uses the broker as proxy.

In addition, some requests were manually performed with cURL during the simulation. This was particularly useful to simluate low-bandwith attacker traffic.

To minimize the length of this report, I will provide only the traffic generation script by Alpha using his proxy.

```
#!/usr/bin/python3

import requests
import random
import time
import sys

urls = ["https://www.google.com/", "https://www.netflix.com/", "https://www.x.com/", ...]

proxies = {'http': 'http://broker:3128/', 'https': 'http://broker:3128'}

min_delay = 0.01
max_delay = 0.2
n = int(sys.argv[1])

def access_random_url():
    url = random.choice(urls)
    try:
        response = requests.get(url, timeout=1, proxies=proxies)
        print(f"{url} - Status: {response.status_code}")
        return response
    except requests.exceptions.RequestException as e:
        print(f"{url} - Error: {e}")
        return None

def main():
    for i in range(n):
        print(f"Request {i}/{n}");
        access_random_url()
        delay = random.uniform(min_delay, max_delay)
        time.sleep(delay)

if __name__ == "__main__":
    main()
```

For the traffic generation by the victim and the other students (Romeo, Juliett, etc.), I use a similar script without a proxy.

4.1.6 Running the Simulation

After the setup is completed, I can run the simulation to generate the traffic as it shows in fig 4.13 where all the actors are doing their own activities (the automation script is running and YouTube is open with autoplay).

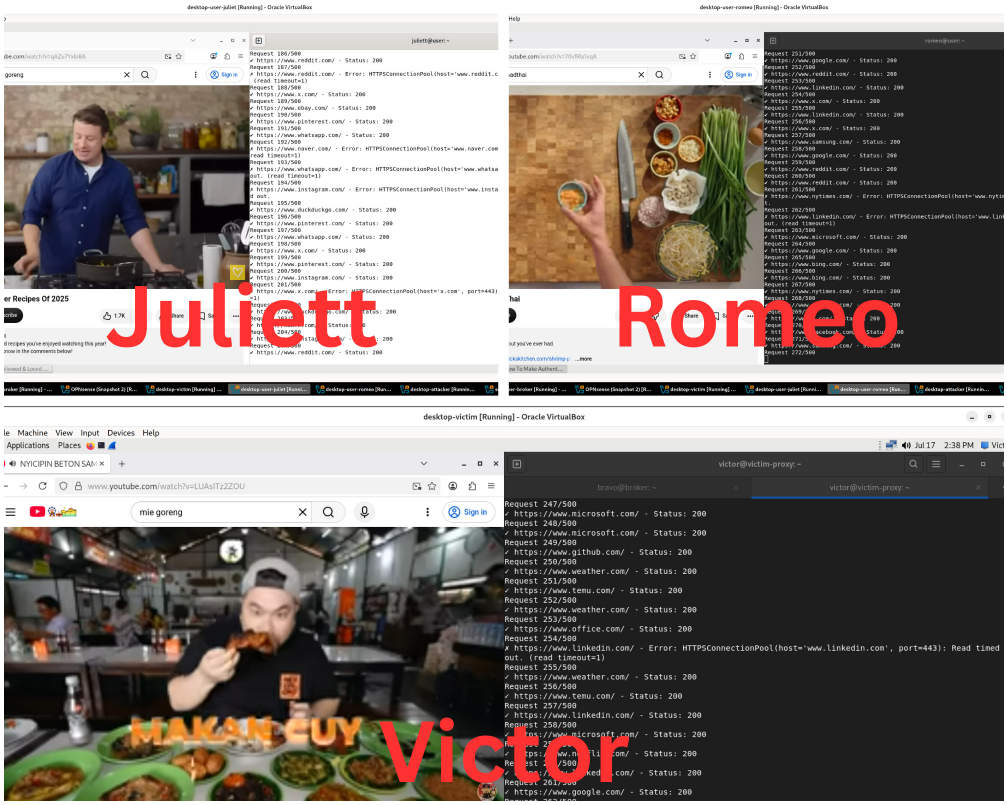


Figure 4.13: Overview of the Actors' Activity

On the residential proxy provider, the broker has activated its listener program to allow the connection from the malware to the broker. Next, Alpha (the attacker) will set up and use proxy service that given from the Broker to run his action as it shown in figures 4.14 and 4.15.

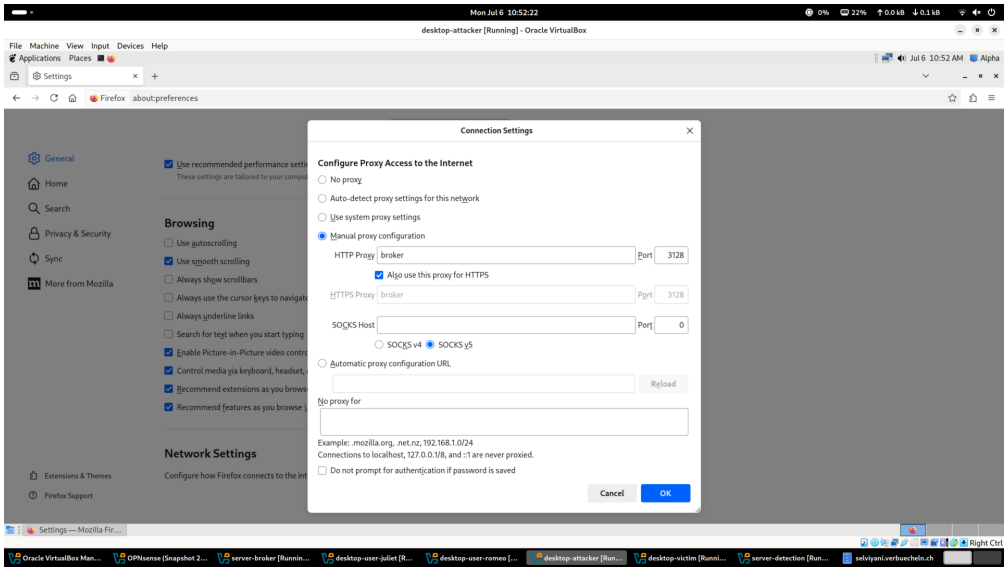


Figure 4.14: Attacker Setup of Residential Proxy

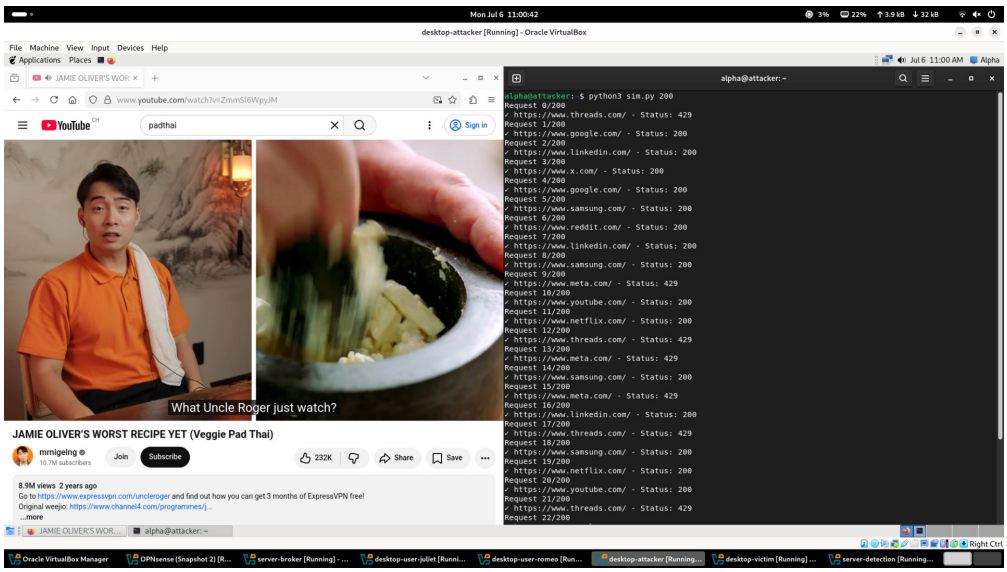


Figure 4.15: Attacker Surfing the Internet with RESIP

4.2 Development of NetFlow-Based Detection

4.2.1 Detection with NetFlow

For the first approach, I created a Python script to analyze the flows traffic from OPNsense which was collected by *nfdump* and *nfcapd*. nfcapd captures all this activity in NetFlow format and stores it to the directory `/var/www/cache/nfdump/` as it shows in the figure 4.16.

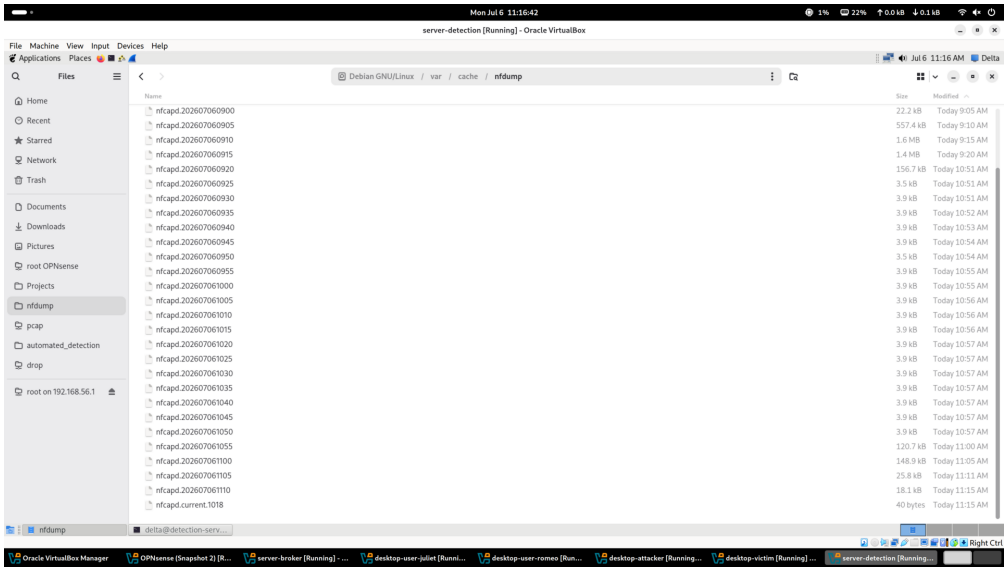


Figure 4.16: NetFlow Files Collection

Since there was no native library available for processing nfcapd files with Python, I converted each NetFlow file to the CSV format. The Python script is using the same logic that the algorithms in Tosun et. al paper [5] describe. However, since the NetFlow data is representing

flows rather than individual packets, there are not a lot of options to correlate flows. This also means that a lot of information about timing and size of individual patterns is lost in the NetFlow representation.

Here is the Python script to detect the RESIP on NetFlow:

```
#!/usr/bin/python3

import sys
import ipaddress

filename = sys.argv[1]
print("Reading CSV from: ", filename)

monitoring_ip = ["192.168.56.107", "192.168.56.130", "192.168.56.141"]

flow = []

with open(filename, "r") as file:
    for line in file:
        fields = line[:-1].split(',')
        if not fields[2] == "6" or float(fields[1]) < 100:
            continue
        if fields[4] != "443" and fields[6] != "443":
            continue
        if fields[3] in monitoring_ip:
            fields.append("out")
        elif fields[5] in monitoring_ip:
            fields.append("in")
        else:
            continue
        flow.append(tuple(fields))

sorted_flow = sorted(flow, key=lambda x: int(x[8]))

toplist = {}

for a in flow:
     for b in flow:
         if a[3] == b[5] and b[3] == a[5]:
             if a[10] == "in" and b[10] == "out":
                 if int(a[8]) < int(b[8]):
                     if a[3] in toplist:
                        toplist[a[3]] += 1
                     else:
                        toplist[a[3]] = 1


toplist = dict(sorted(toplist.items(), key=lambda x: x[1], reverse=True))

for candidate, score in toplist.items():
    print(f"suspicious IP address \033[93m{candidate}\033
        [0m appearing \033[93m{score}\033[0m times")
```

To automate the processing of multiple CSV files, the following Bash script was used.

```
#!/bin/bash

rm -vf netflow-data/*

for day in 20260706
do
  for i in /var/cache/nfdump/nfcapd.$day*
  do
    echo "Converting $i ..."
    nfdump -R $i -o csv | grep "^2026" >> netflow-data/flows-$day.csv
  done
done

for i in netflow-data/*.csv
do
  python3 netflow-detection.py $i
done
```

The figure 4.17 shows the output when I run the detection script based on NetFlow. As it shows, the detection produces many false IP addresses that are suspicious. However, the Broker IP address also shows on the top of the list.

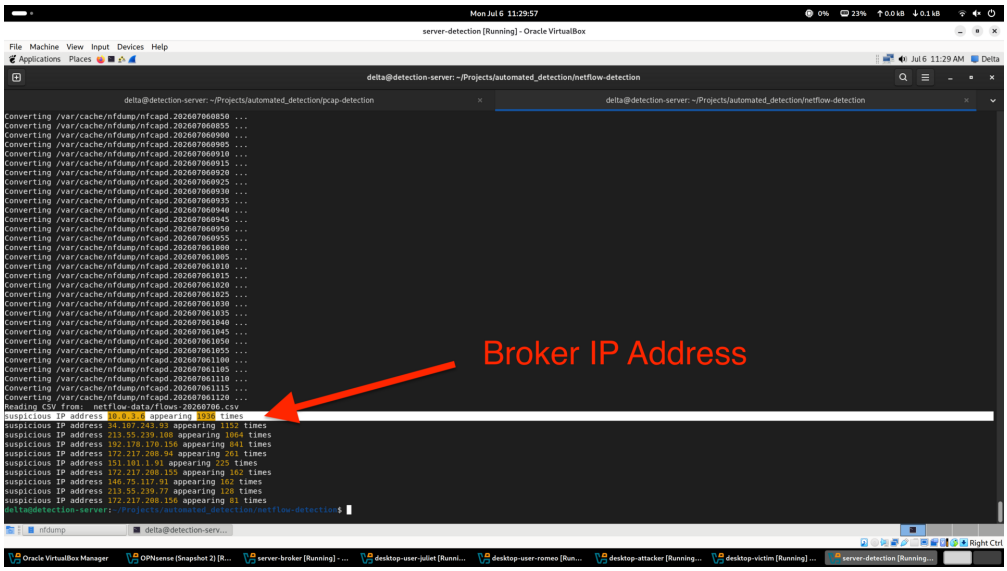


Figure 4.17: NetFlow Detection Result

4.3 Development of PCAP-Based Detection

After the experiment with NetFlow did not work out as hoped, I started to develop the PCAP-basd detection. For this case, I will refer to the algorithms and source code from Tosun et al. on their research paper [5], which I used as basis. For my use case, I adjusted the input and output of the detection program and wrote scripts to automate the integration with OPNsense.

The problem with the first attempt, the original source code from Tosun et al. [5], is that it only analyzes a single PCAP file and outputs the entire PCAP file as a `.txt` file, which only

displays the IP address, position, violations, and packet count. This solution is impractical in my use case where I want to record network traffic over a long period of time (e.g., 2 hours). Therefore, the resulting PCAP file will be large, and will result in a very long processing time since the program only processes a single PCAP file. To solve this, I break down the PCAP files to smaller units by rotating every 60 seconds, and I change the Python program to make suitable output for my case by adding a detection score minimum in order to reduce false positives result, where it only prints the suspected proxy if it reaches the minimum score.

While all the actors are doing the same activities, the detection machine is transferring the PCAP files from OPNsense using `rsync` and analyzing the PCAP files for RESIP activity in the ABC organization. The first step is to record the network traffic on OPNsense using `tcpdump`, as shown in the figure 4.18, and then copy the PCAP files from the OPNsense location at `192.168.56.1:/root/pcap` to the detection machine using `rsync`, as illustrated in the figure 4.19, to analyze and detect the residential proxy's behavior.



Figure 4.18: OPNsense Capture the Traffic with tcpdump

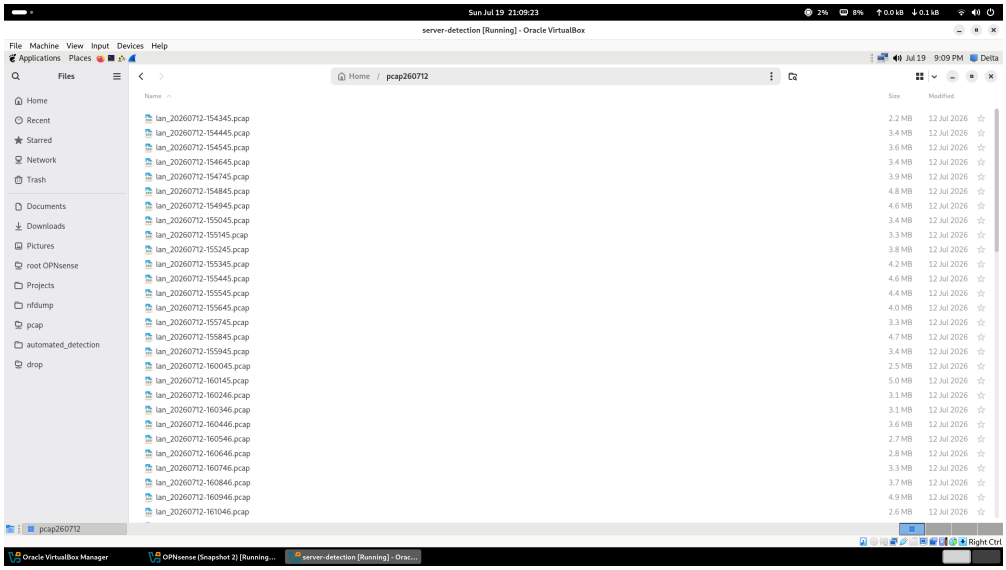


Figure 4.19: PCAP Files Collection

4.3.1 Detection with PCAP

Tosun et al. in their paper [5] define three algorithms to detect existence of residential proxy.

Algorithm 1: Suspicious Flow Check is checking for packets with suspicious timing correlations, based on the assumption that a request to the proxy is always preceding a request from the proxy to the target, and vice versa for responses.

Algorithm 2: Data Check is then used to reduce the false positive rate based on the assumption that packets between attacker and proxy have to be larger than the packets from proxy to target.

Algorithm 3: DNS Check tries to reduce false positives even further by analyzing DNS traffic. However, since DNS involves a lot of caching, it is the least reliable of the three and turned out not as significant in my simulation.

Algorithm 1: Suspicious Flow Check

```
device_ip ← IP address of the device
network_flow ← List of packets extracted from PCAP file
threshold ← Chosen threshold for suspicion
Initialize suspicious_connections as an empty list
Initialize inc_flow_ips as an empty list
Initialize out_flow_ips as an empty list
Initialize prev_inc_pckt as an empty string
Initialize prev_out_pckt as an empty string

for each packet in network_flow do
    if destination ip address of packet = device_ip then
        if source ip address of packet = prev_inc_pckt then
            skip to processing next packet
        else if source ip address of packet not in inc_flow_ips then
            insert source ip address of packet into inc_flow_ips
        prev_inc_pckt ← source ip address of packet
    else if source ip address of packet = device_ip then
        distance ← CalculateDistance(position of packet, destination ip of packet)
        if destination ip address of packet = prev_out_pckt then
            skip processing to next packet
        else if destination ip address of packet not in out_flow_ips then
            insert destination ip address of packet into out flow_ips
        else if destination ip address of packet in inc_flow_ips and
         distance > threshold then
            insert suspicious connection into suspicious_connections
        prev_out_pckt ← destination ip address of packet

procedure CalculateDistance(pos, ip)
    distance ← 0
    for i ← pos to 0 do
        if ip = source ip of network_flow[i] then
            distance ← pos − i
            break
    return distance
```

Algorithm 2: Data Check

```
device_ip ← IP address of the device
suspicious_connections ← List of suspicious connections detected in algorithm 1
network_flow ← List of packets extracted from PCAP file
time_threshold ← Chosen threshold for required time to send response back

for each connection in suspicious_connections do
    total_data_outgoing ←
     TotalDataSent(position of connection, IP address of connection)
    total_data_incoming ←
     TotalDataReceived(position of connection, number of packets flowed during suspicion)
    if total_data_incoming ≤ total_data_outgoing then
        update suspicious_connections by adding violation of data to current connection

procedure TotalDataSent(pos, ip)
    total_data ← 0
    start_time ← time of network_flow[pos]
    for i ← pos to size of network_flow do
        time_diff ← time of network_flow[i] − start_time
        if time_diff < time_threshold and ip = destination ip of network_flow[i]
         then
            total_data ← total_data + size of network_flow[i]
        else if time_diff ≥ time_threshold then
            break
    return total_data

procedure TotalDataReceived(pos, num_of_packets)
    Initialize incoming_data as an empty dictionary
    for i ← pos − num_of_packets to pos do
        if destination ip of network_flow[i] = device_ip then
            key = source ip of network_flow[i]
            if key in keys of incoming_data then
                value of incoming_data[key] ← incoming_data[key]+ size of
                 network_flow[i]
            else
                insert [key, size of network_flow[i]] into incoming_data
    total_data ← select maximum total data sent by single IP from incoming_data
    return total_data
```

Algorithm 3: DNS Check

```
suspicious_connections ← List of suspicious connections updated in algorithm 2
network_flow ← List of packets extracted from PCAP file
time_threshold ← Chosen threshold for required time to make DNS lookup

for each connection in suspicious_connections do
    time_diff ← DNSCheck(position of connection, number of packets flowed during suspicion)
    if time_diff ≥ time_threshold then
        skip to the next iteration of connections
    else
        update suspicious_connections by adding violation of DNS to current connection

procedure DNSCheck(pos, num_of_packets)
    iterator ← pos − num_of_packets
    start_time ← time of network_flow[iterator]
    time_diff ← maximum integer
    while True do
        packet ← network_flow[iterator]
        if iterator = pos then
            break
        if protocol of packet = DNS then
            time_diff = time of packet − start_time
            break
        iterator ← iterator + 1
    return time_diff
```

The authors released corresponding Python programs which implement these algorithms using the `Scapy` library to process PCAP files as input. I modified the programs for my use cases and added shell scripts to implement the automation and integration with OPNsense. Since firewall and router appliances typically do not have a lot of processing power available for extra tasks and usually lack a full Python environment, I execute the detection on the detection server using rsync to fetch new PCAP data from OPNsense as it is recorded.

```
#!/bin/bash

# sync data from OPNsense
rsync -avvi --del root@192.168.56.1:/root/pcap/ pcap-data/
```

I also use `GNU parallel` as a shell tool for automatically executing the jobs for each 60-second sample in parallel using more CPUs if available, increasing processing time faster. The option `-k` tells GNU parallel to preserve the order of outputs as if the jobs were executed in serial order.

```
#!/bin/bash

# delete old result
```

```
rm -f suspicious_connections.txt

echo "sample;address;count;" > result.csv

# run detection script in parallel
parallel -k python3 detection.py $1 {} ::: $2/*.pcap
```

Figure 4.20 shows the result from my residential proxy detection. I successfully detect the correct IP address of the broker, which is `10.0.3.6`, and inform me of the infected victim machine IP address, which is `192.168.56.141`. In this case, it does not show the false positive IP address because I set the threshold pretty high, to 120.

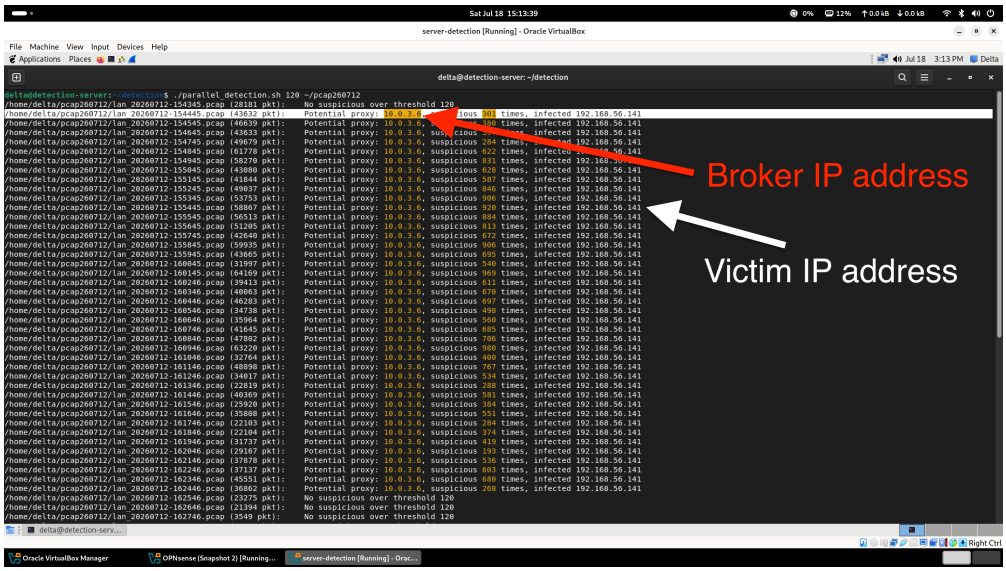


Figure 4.20: PCAP Detection Result

4.4 Simulation in a Real Hardware Network

To investigate the behavior for future plans of practical application, I also run a simulation with real machines in a physical network in addition to the VM-based simulation. The implementation of the test malware and the RESIP with socat and SSH is identical to the virtual setup.

The AMD-based main desktop (hostname: `deep`) was used as the victim machine. An old Intel-based Macbook Pro (hostname: `jeltz`) which is set up as a Debian server was used as the broker. A Lenovo ThinkPad (hostname: `eddie`) was used as the attacker machine, setting up the proxy in Firefox web browser. As a gateway, a Dell laptop (hostname: `dell`) was used which used an Ethernet interface as public network and provided a WLAN hotspot as internal network.

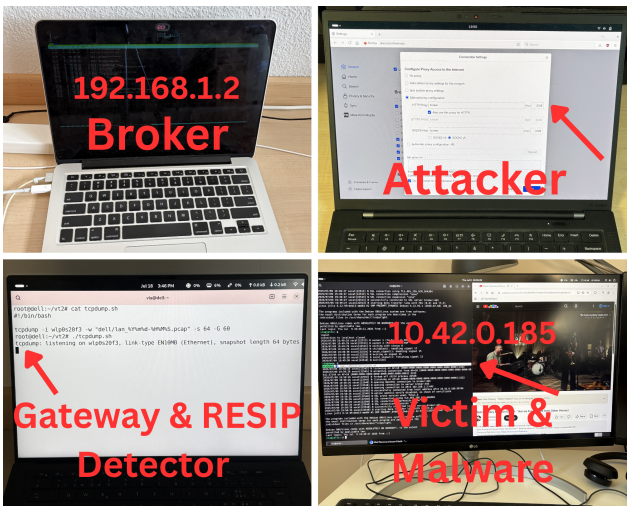


Figure 4.21: Real Test Simulation

The picture 4.21 above illustrates the following: the broker machine is running the `socat` listener, the malware on the victim machine is running, and the victim is surfing the Internet to make noise and generate traffic. Meanwhile, the attacker is connected to the broker's proxy port. The gateway records the traffic with `tcpdump`.

After I collect the PCAP files on the gateway. The next step is to run the detection program to detect RESIP. As the picture 4.22 shows, it shows a mixed-up result of the IP addresses, both correct and false. This possibly happens because I only set up a threshold of 30. However, as I discussed above, the main characteristic of the RESIP (broker) IP address `192.168.1.2` will always appear with high numbers. Furthermore, I can adjust the threshold number as needed.

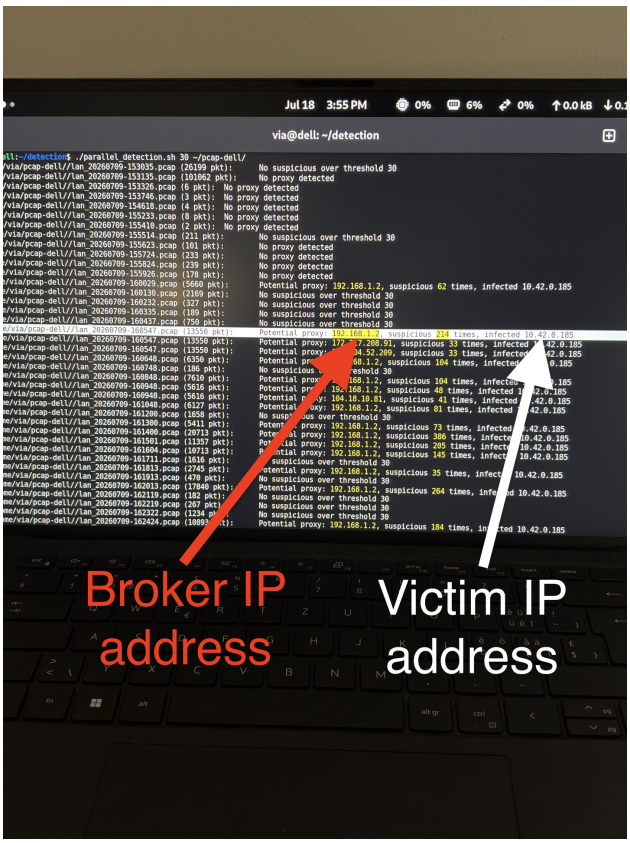


Figure 4.22: Real Test Detection

5 Results and Conclusion

5.1 Results

In this chapter, I will discuss the results of the simulated tests I conducted to detect the existence of RESIP proxies from two different data formats, NetFlow and PCAP.

5.1.1 NetFlow-Based Detection

NetFlow was a promising candidate for RESIP detection because it is a universally accepted protocol supported by many network router devices across vendors. However, it turned out that the NetFlow detection program produces many false positive errors in its result while failing to detect the RESIP. Further tests came to the conclusion that NetFlow alone cannot be relied on to detect the proxy behavior.

There might be various reasons why NetFlow is not suitable for proxy detection:

1. NetFlow does not record individual packets but flows. Since the timing information (in micro seconds) of individual packets is lost, correlation of incoming and outgoing traffic is much less accurate.
2. NetFlow in OPNsense will record the traffic and send it to the nfcapd only every five minutes. This makes it even harder to detect individual traffic patterns.

3. Some flows do not appear in the OPNsense NetFlow records at all. I assume that the reason is that the flow is only recorded if it starts and ends during the monitoring period. Packets which belong to flows started before or ended after recording appear to be ignored. Since proxy connections are usually persistent over a long time, they are often missed by the OPNsense NetFlow plugin.

Because of the unsuccessful tests and the reasons above, I conclude that NetFlow is not suitable to detect RESIP behavior in my use case. However, this conclusion does not rule out the possibility that NetFlow could be useful in the future for detecting RESIP when a newer version is available.

5.1.2 PCAP-Based Detection

The PCAP-based detection algorithms are showing much more promising results. The RESIP in my test setup was consistently detected over multiple runs, regardless whether the attacker, victim and the other users were using web browsers (e.g. YouTube on automatic playback) or automated scripts (Python, cURL) generating HTTP requests.

Table 5.1 shows the result of the algorithms applied to the PCAP files. Note that the original Python program is applied offline to one PCAP file and results in one output file. In contrast, I run the program on each sample which represents one minute of tcpdump recording, as I set a rotation time of 60 seconds as explained in chapter 4. This makes my application of the detection program more suitable for live detection in an organization.

Table 5.1: Result matrix for both simulations

Type	Samples	Not det.	Incorrectly det.	Correctly det.	FP	FN
VirtualBox	107	0	61	42	93.8%	0.0%
Hardware	56	23	10	23	17.9%	58.9%

At first, the result looks mixed because there are a lot of false positives. However, after running the detection algorithm multiple times, I noticed that the false positives show a low number of detection per sample, while the correct proxy showed a significantly higher number, as can be seen in figure 5.1.

```
pcap-sim/lan_20260710-203917.pcap (11659 pkt): Most likely proxy: 213.55.239.110, suspicious 6 times
pcap-sim/lan_20260710-201113.pcap (22133 pkt): Most likely proxy: 213.55.239.108, suspicious 14 times
pcap-sim/lan_20260710-203817.pcap (24870 pkt): Most likely proxy: 74.125.29.91, suspicious 20 times
pcap-sim/lan_20260710-202616.pcap (21343 pkt): Most likely proxy: 10.0.3.6, suspicious 128 times
pcap-sim/lan_20260710-203417.pcap (31837 pkt): Most likely proxy: 173.222.108.160, suspicious 20 times
pcap-sim/lan_20260710-203717.pcap (34256 pkt): Most likely proxy: 192.168.56.1, suspicious 5 times
```

Figure 5.1: Example output of the PCAP detection

In the next step, I set a minimum how often it is detected per sample by the detection program in order to filter the false positives. In my simulation, 30 and 60 were working well. Note that this minimum applies to each rotated PCAP file, i.e. for 60 seconds. I expect that in real organization networks, these parameters have to be calibrated to match the organization's traffic volume. The results of this tweak can be seen in tables 5.2, 5.3, and 5.4.

Table 5.2: Confusion matrix without minimum

	detected	not detected
proxy (broker IP address)	42 (100.0%)	0 (0.0%)
no proxy (wrong IP address)	61 (93.8%)	4 (6.2%)

Table 5.3: Confusion matrix for minimum of 30

	detected	not detected
proxy (broker IP address)	30 (71.4%)	12 (28.6%)
no proxy (wrong IP address)	6 (9.2%)	59 (90.8%)

Table 5.4: Confusion matrix for minimum of 60

	detected	not detected
proxy (broker IP address)	20 (47.6%)	22 (54.2%)
no proxy (wrong IP address)	3 (4.6%)	62 (95.4%)

As one can see, the false positives can be eliminated with this approach. The next question is whether the false negative rate is low enough. At first look, it looks quite high with up to 54.2 percent. However, note that this is the detection rate for each sample representing 60 seconds. When we look at the results over a longer time window, the detection success rate is pretty high.

5.2 Conclusion

From my tests, I arrive at the conclusion that these algorithms as proposed by Tosun et. al [5] worked well in my simulation. It could be interesting to apply these methods to a real organization network and evaluate the results.

Especially the following questions appear when I want to transfer from my simulation to a real organization, e.g. a school with students:

- Can `tcpdump` or a similar tool be run on a gateway router safely and securely in a real organization?
- Does the performance of `tcpdump` hold up?
- Does the performance of the PCAP analysis hold up in case of real organization traffic volumes?
- How could the data be filtered or preprocessed in order to drop unnecessary data collection and improve performance?
- Is it possible to detect the broker's IP address precisely, as we know in the real-world case, that the broker might have multiple IP addresses to run its RESIP services that might change at any time?

In summary, based on my simulation of residential proxy detection, I conclude that this program could assist the network administrators of an organization in identifying the presence

of RESIP. Even though we cannot rely a hundred percent on the detection program (e.g., for automated blocking) due to the false positive rate in the statistics, this program will give the network operators enough information allowing them to manually investigate potentially infected devices in the organization and block the IP address.

Bibliography

[1] "Buy Fast Residential Proxies | HTTP & SOCKS5 Rotating Proxies," Proxying, Accessed: May 28, 2026. [Online]. Available: https://www.proxying.io/

[2] 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)*, Oct. 2024, pp. 1–9. DOI: 10.23919/CNSM62983.2024.10814519 Accessed: May 28, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/10814519

[3] D. Group. "What is an ORB network?" DIGITPOL, Accessed: May 28, 2026. [Online]. Available: https://digitpol.com/what-is-an-orb-network/

[4] R. Huang, D. Zhao, X. Mi, and X. Wang. "Shining Light into the Tunnel: Understanding and Classifying Network Traffic of Residential Proxies." arXiv: 2404.10610 [cs], Accessed: Feb. 27, 2026. [Online]. Available: http://arxiv.org/abs/2404.10610 pre-published.

[5] 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: Mar. 2, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/9427688/

[6] 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: Jun. 23, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/9115039

[7] X. Mi et al., "Resident Evil: Understanding Residential IP Proxy as a Dark Service," in *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019, pp. 1185–1201. DOI: 10.1109/SP.2019.00011 Accessed: Jun. 27, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/8835239

[8] "Become a Peer - Proxyrack," Accessed: Jun. 27, 2026. [Online]. Available: https://www.proxyrack.com/become-a-peer

[9] 911proxy. "Make Money with Proxies - Exploring Proxy-Based Income Opportunities," 911proxy, Accessed: Jun. 27, 2026. [Online]. Available: https://www.911proxy.com/make-money-with-proxies/

[10] "Proxy detection methods. Here are some common proxy detection... | by Leonidas Gorgo | Medium," Accessed: Jun. 27, 2026. [Online]. Available: https://leonidasgorgo.medium.com/proxy-detection-methods-a2b33355a99a