AGH

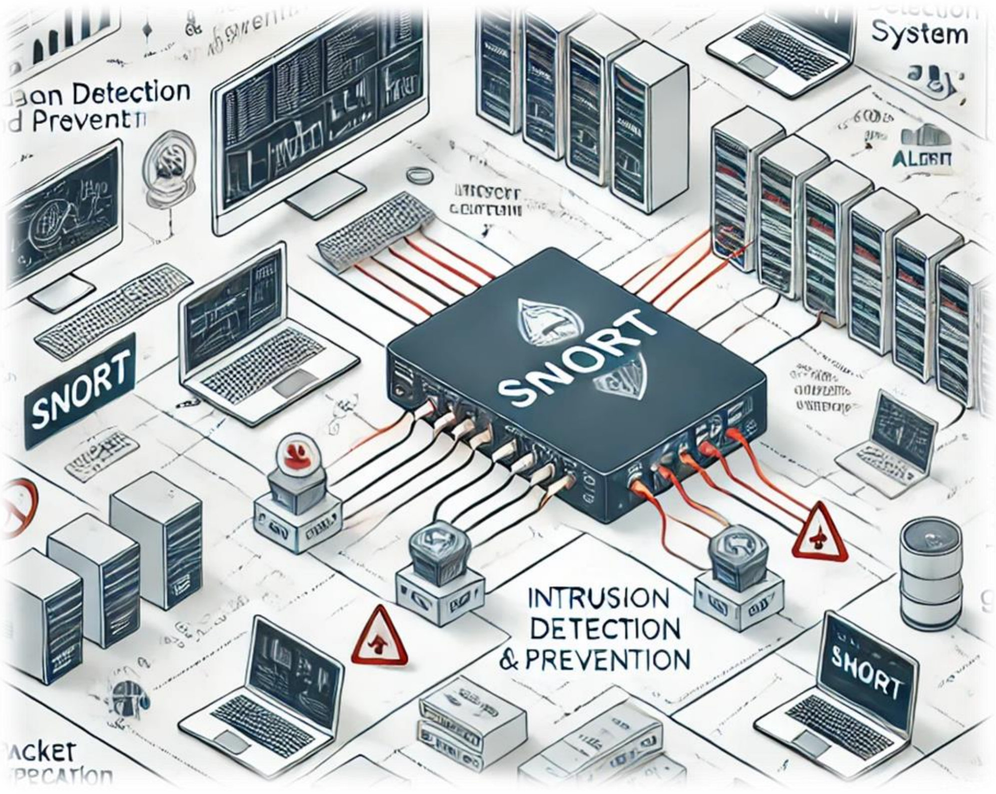
FACULTY OF COMPUTER SCIENCE OF AGH
UNIVERSITY OF SCIENCE AND TECHNOLOGY

ERASMUS EXCHANGE

TEACHER	TOMASZ SZUMLAK
DATE	30/01/2025

PYTHON IN THE ENTERPRISE

INTRUSION DETECTION SYSTEM WITH SNORT



NDJAMBA BATOMEN GABRIELLA - gndjamba@student.agh.edu.pl

Table des matières

1.Introduction

1.1. Motivation for the project

The rising number and complexity of cyberattacks have made it essential to use effective tools for detecting and preventing intrusions. This project focuses on setting up a network intrusion detection system using Snort, a well-known open-source tool. My goal is to gain an understanding of how Snort works and how it can identify and address threats in a controlled setting and also to show how useful and relevant it can be for enterprise as an IDS.

1.2. Choice of Snort for Network Intrusion Detection

Snort was selected for this project because of its versatility, cost-effectiveness, and strong reputation as a leading open-source intrusion detection and prevention system (IDPS). As a free tool, Snort offers a powerful platform for monitoring network traffic.

One of its key strengths is flexibility, allowing users to create and customize detection rules, which was crucial for simulating specific attacks in this project. Additionally, snort functions as both an intrusion detection system (IDS) and an intrusion prevention system (IPS), enabling experimentation with both passive monitoring and active threat mitigation.

Snort supports various protocols, making it capable of detecting different types of attacks, such as ICMP and SYN floods. Overall, using Snort provided a valuable learning experience by connecting theoretical cybersecurity knowledge with practical application. It was an ideal tool for exploring concepts like traffic analysis and rule creation in a real-world context.

2. Project Scope and objectives

2.1. Project scope

This project focuses on implementing Snort in an isolated virtualized environment to monitor network traffic. It involves configuring Snort as an IDS, simulating different types of attacks, and analyzing the resulting logs for insights. The version of snort used in this project is snort3 which is the latest version of snort.

Goals of the Project

- Successfully install and configure Snort.
- Create and test custom rules.
- Simulate various network attacks.
- Analyze Snort-generated alerts and visualize findings.

3.Installation and Setup

3.1. Environment preparation

The project involved two virtual machines (VMs) within a controlled, isolated network: an attacker VM (Kali Linux, 192.168.0.28) and a Snort sensor VM (192.168.0.29). This isolated environment prevents any unintended impact on external networks and allows for safe and ethical testing of attack scenarios. Both VMs were configured with bridged networking in VirtualBox, placing them on the same subnet as the host machine and allowing them to communicate as if they were physical devices on the same network. This configuration simulates a more realistic network environment for the experiment.

Installation Steps

To install Snort, the following steps were followed meticulously:

1.Cloning the Snort Repository

The Snort source code was downloaded directly from its official GitHub repository. The repository link used was Snort3 GitHub Repository.

Commands executed to clone the repository:

```
git clone https://github.com/snort3/snort3.git

cd snort3
```

2.Installing Dependencies

Before compiling Snort, essential dependencies were verified and installed. These included:

- libpcap
- DAQ (Data Acquisition library)
- Development tools such as gcc, cmake, and make.

Command to install dependencies on Kali Linux:

```
sudo apt update && sudo apt install -y build-essential libpcap-dev libpcre3-dev zlib1g-dev
cmake git
```

3.Compiling and Installing Snort

The following commands were executed to build and install Snort:

```
./configure_cmake.sh --prefix=/usr/local/snort

cd build

make

sudo make install
```

4.Verifying the Installation

To ensure Snort was installed correctly, the version was checked using the command:

```
snort –version
```

(PICTURE) This output confirmed that Snort 3 was successfully installed and operational.

By following these detailed steps, Snort was set up and ready for further configuration and testing.

4. Configuration of snort

Snort needs a comprehensive configuration file that defines network interfaces, specifies rule sets, establishes logging parameters, and sets up detection mechanisms. This configuration file serves as the blueprint for Snort's operational framework, dictating how it will monitor, interpret, and respond to network activities.

Additionally, Snort requires predefined rule files that contain specific patterns and conditions for identifying potential security threats, network anomalies, and malicious activities. These rules act as the system's intelligent filters, enabling it to distinguish between normal network traffic and potential security breaches.

4.1. Understanding snort’s configuration file

The primary configuration file for Snort, **snort.lua**, defines variables, rules, and operational modes.

Here is a breakdown of snort’s configuration file:

Section 1: Configure Defaults

```
HOME_NET = 'any'

EXTERNAL_NET = 'any'
```

This is like setting up the boundaries of your network neighborhood. HOME_NET defines the network you're protecting (in this case, it's set to 'any', which means it'll cover all possible network addresses), and EXTERNAL_NET defines networks outside your protected zone.

Section 2: Configure Inspection

This section is telling Snort what types of protocol to pay attention to. It's full of various protocol inspectors:

Stream modules: These track the state of network connections

Protocol-specific inspectors like :

- **dns** : Watches DNS traffic
- **http_inspect:** Monitors HTTP connections
- **ssl:** Checks SSL/TLS traffic
- **ssh** : Monitors SSH connections

It's basically Snort's toolbox of traffic analyzers, each specialized for different types of network communication.

Section 3: Configure Bindings

The **binder** section is like a traffic director. It tells Snort which inspector to use for specific network services and ports. For example:

- Traffic on port 53 (DNS) gets the DNS inspector
- Traffic on port 502 (Modbus) gets the Modbus inspector

It's similar to having different security checkpoints for different types of vehicles.

Section 4: Configure Performance

This section (currently commented out) allows you to:

- Monitor packet processing time
- Track rule performance
- Capture performance data for later analysis

It’s like putting speedometers and performance trackers on your network security system.

Section 5: Configure Detection

Here, you set up how Snort will detect potential security threats:

- **references**: Provides context for security rules
- **classifications**: Categorizes different types of network threats
- **ips**: Configures intrusion prevention system rules

Section 6: Configure Filters

This section lets you customize how Snort handles detected events:

- **suppress**: Ignore specific types of alerts
- **event_filter**: Limit the number of alerts for certain rules
- **rate_filter**: Create custom actions based on connection rates

It's like having fine-tuned noise-canceling for your network security alerts.

Section 7: Configure Outputs

This section determines how Snort logs and reports events:

- Various alert formats (CSV, fast, full)
- Packet logging options
- Additional logging mechanisms

Section 8: Configure Tweaks

The last section allows for including additional configuration files, providing flexibility for custom setups.

4.2. Modifying snort configuration file for custom settings

4.2.1. Network Definition Changes

Default:

```
HOME_NET = 'any'

EXTERNAL_NET = 'any'
```

Modified:

```
HOME_NET = '192.168.0.29/24'

EXTERNAL_NET = '!$HOME_NET'
```

Why the Change?

The modified version is more secure because:

- It specifically defines our protected network (192.168.0.29/24)
- Uses '!$HOME_NET' to define external networks as "everything except our home network"
- This makes the rules more focused and reduces false positives

4.2.2.Detection Settings Changes

Default:

```
ips = {

-- enable_builtin_rules = true,

variables = default_variables

}
```

Modified:

```
ips = {
enable_builtin_rules = true,
include = "/usr/local/etc/rules/pulledpork.rules",
variables = default_variables
}
```

Why the Change?

- Enables built-in rules for better detection
- Adds external rules from PulledPork (a rule management tool)
- Provides more comprehensive threat detection

4.2.3.Output Configuration Changes

Default:

```
--alert_csv = { }
--alert_full = { }
--log_pcap = { }
```

Modified:

```
alert_csv = { file = true }
alert_full = { file = true }
log_pcap = { }
```

Why the Change?

- Enables CSV alerts for easy log analysis
- Enables full alerts for detailed incident investigation
- Keeps PCAP logging for packet-level analysis
- Better logging helps in incident response and forensics

5. Snort Rules with PulledPork

5.1. PulledPork for rule management

Managing and maintaining an effective Snort ruleset is a complex and time-consuming task. Manually downloading, updating, and configuring hundreds or even thousands of rules is prone to errors and can quickly become unmanageable. PulledPork is a popular open-source Perl script specifically designed to automate this process. It simplifies the acquisition, customization, and deployment of Snort rules, making it significantly easier to keep your Snort installation up-to-date and effective.

PulledPork offers several key advantages for Snort rule management:

- **Automated Downloads:** PulledPork can automatically download rules from various sources, including the official Snort and Talos rulesets, as well as community-maintained lists and custom rules. This eliminates the need for manual downloads and ensures you have access to the latest rules.
- **Rule Updates:** PulledPork can check for rule updates on a scheduled basis and automatically download and apply them. This ensures the snort installation is always using the most current rules to detect the latest threats.
- **Customization:** PulledPork allows us to customize the ruleset by enabling, disabling, or modifying rules based on our specific needs and network environment. This gives us fine-grained control over the alerts Snort generates.
- **Policy Management:** PulledPork supports policy-based rule management, allowing us to define different rule sets for different purposes or network segments.
- **Simplified Configuration:** PulledPork streamlines the rule update process, eliminating complex manual configuration steps and reducing the risk of errors.

5.2. PulledPork Installation and Configuration

In this project, PulledPork was used to manage Snort rules, ensuring an updated and effective ruleset throughout the experiment. PulledPork was installed on the Snort VM by cloning the official GitHub repository and then running the install.pl script:

Cloning the Repository:

```
git clone https://github.com/shirkdog/pulledpork.git
cd pulledpork
```

Running the Installer:

```
sudo ./install.pl
```

This script handles the installation process, including copying the necessary files and setting up the default configuration.

After installation, the PulledPork configuration file, pulledpork.conf, located at /etc/snort/pulledpork.conf, was customized to match the project's requirements. The needed part of the configuration used in this project is shown below (what is written in red is what has been modified):

```
# Which Snort/Talos rulesets do you want to download (recomended: choose only one)
community_ruleset = false
registered_ruleset = true
LightSPD_ruleset = false


# Your Snort oinkcode is required for snort/talos Subscription, Light_SPD, and Registered rulesets
oinkcode = xxxxxxxxxxxxxx


# where to save our single combined rule file (This is required, and needs to be an absolute path):
rule_path = /usr/local/etc/rules/pulledpork.rules


# Enable / Disable rules based on the level of functionality/security you want.
# must be one of: connectivity, balanced, security, max-detect, none
# default is connectivity. Will not work with community ruleset.
# https://www.snort.org/faq/why-are-rules-commented-out-by-default
ips_policy = balanced


# Rule Output mode:
# simple (just a single .rules file, with any rule that's not commented out enabled)
# policy (rules file with all rules enabled, using a policy file to enable/disable rules)
rule_mode = simple


# where should so rules be saved
# so rules will only be processed if this is uncommented
sorule_path = /usr/local/etc/so_rules/
```

The key settings within pulledpork.conf are:

- **registered_ruleset = true**: This setting enables downloading the registered Snort/Talos ruleset, providing access to a comprehensive set of rules maintained by Cisco Talos.
- **oinkcode = <my_oinkcode>**: The Snort Oinkcode, is a unique code that grants access to the registered ruleset. It is essential to keep this code confidential.
- **rule_mode = simple**: This configures PulledPork to generate a single, combined .rules file containing all enabled rules. This simplified approach is suitable for this project and makes rule management easier.
- **rule_path = /usr/local/etc/rules/pulledpork.rules**: This sets the absolute path where PulledPork saves the combined rule file. It's important to use an absolute path here. This combined rule file will be included within Snort's snort.lua configuration file so that Snort loads all the downloaded and customized rules.

- **sorule_path = /usr/local/etc/so_rules/**: This specifies the directory where shared object (.so) rules are saved. These precompiled rules, downloaded via PulledPork, help to boost Snort's performance.
- **ips_policy = balanced**: This sets the intrusion prevention system (IPS) policy to "balanced," which provides a moderate level of security by enabling a subset of rules considered appropriate for general-purpose intrusion detection. Other options include "connectivity," "security," "max-detect," and "none," each offering different levels of detection sensitivity and coverage. The "balanced" policy provides a reasonable trade-off between security and performance.

After configuring pulledpork.conf, PulledPork was executed to download and compile the specified rules. The resulting pulledpork.rules file, containing the updated and customized ruleset, was then included in Snort's configuration file (snort.lua) using the include directive, ensuring that Snort utilized the latest rules during the attack simulations. This process automated the rule management, enabling consistent and reliable intrusion detection throughout the project.

6.Simulation of Network Attacks

6.1. Overview of the Simulated Attacks with python

To assess Snort’s detection capabilities, we simulate multiple network attacks using Python and Scapy. Each attack targets the Snort-monitored machine and is designed to trigger specific alerts. The attacks include ICMP flooding, UDP flooding, and DNS amplification, all executed from an attacker VM. A controlled attack intensity (att_int) is set to adjust the frequency of packets.

6.2. Attack 1: ICMP Flood (Ping Flood)

The ICMP flood attack overwhelms the target with a continuous stream of ICMP Echo Request (ping) packets. This is commonly used in denial-of-service (DoS) scenarios to exhaust network bandwidth. In the simulation, a Python script sends ICMP packets from the attacker's IP (192.168.0.28) to the target (192.168.0.29) in an infinite loop with minimal delay between packets. Snort should detect an unusual volume of ICMP traffic and trigger an alert.

6.3. Attack 2: UDP Flood

The UDP flood attack involves sending large UDP packets to the target, consuming bandwidth and resources. The attack script randomly selects packet sizes (1024, 2048, or 4096 bytes) and continuously sends them to port 53 (commonly used for DNS). This simulation aims to observe Snort’s ability to flag high volumes of UDP traffic, which may indicate a potential flooding attempt.

6.4. Attack 3: DNS Amplification Attack

DNS amplification is a reflection-based attack where an attacker spoofs the victim’s IP and sends DNS queries to a public resolver (e.g., 8.8.8.8). The DNS server responds with a much larger reply, effectively amplifying the traffic volume toward the victim. In this simulation, the attack script sends DNS queries with a spoofed source IP (192.168.0.28) to the public DNS resolver. Snort should detect an unusual number of outbound DNS requests and potential IP spoofing attempts.

6.5. Overview of the python script

To conduct the attack simulations, a Python script leveraging the Scapy library was developed. This script automates the execution of multiple network attacks, allowing real-time traffic generation targeting the Snort-monitored system. The attacks are executed in parallel using multithreading, ensuring a continuous and realistic traffic flow.

Key Components of the Script:

1.Target and Attacker Configuration

- target_ip: The IP address of the machine running Snort (192.168.0.29).
- source_ip: The attacker's IP address (192.168.0.28).
- att_int: The time interval between packets, set to 0.001 seconds to increase attack intensity.

2.Attack Implementations

- Ping Flood (ICMP Echo Request): Repeatedly sends ICMP Echo Requests to the target, attempting to overload the network.
- UDP Flood: Generates high-volume UDP packets of random sizes (1024, 2048, or 4096 bytes) targeting port 53 (commonly used for DNS).
- DNS Amplification Attack: Sends spoofed DNS queries to an external resolver (8.8.8.8), potentially causing a traffic surge toward the victim.

3.Multithreading for Parallel Execution

Each attack runs in a separate thread to ensure simultaneous execution. Threads are set as daemon processes, meaning they will terminate when the main program stops.

4.Attack Control Mechanism

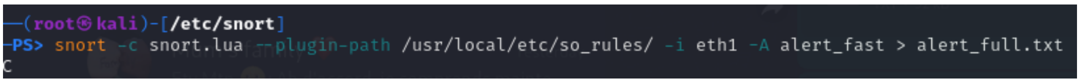
- The script continuously runs until manually stopped via CTRL+C.
- A stop_attacks() function safely terminates all attack threads when an interruption is detected.

This Python script effectively simulates network attacks, enabling Snort to analyze and respond to various malicious traffic patterns. By monitoring the Snort logs and alerts, we can assess its detection accuracy.

7.Analyzing Snort Alerts

7.1. Overview of Snort Alert Logs

The alert logs contain crucial information, including timestamps, alert types, priority levels, and source/destination IP addresses. Analyzing these logs helps detect attack patterns and potential security threats. Here is the code that helps us generate those logs:



```
─(root㉿kali)-[/etc/snort]
─PS> snort -c snort.lua --plugin-path /usr/local/etc/so_rules/ -i eth1 -A alert_fast > alert_full.txt
C
```

1. **snort** → Executes Snort (Snort++/Snort 3).

2. **-c snort.lua** → Specifies the configuration file (snort.lua).
   - In Snort 3, configuration is written in **Lua** instead of the older Snort 2 .conf format.

3. **--plugin-path /usr/local/etc/so_rules/** → Defines the path where Snort should look for **shared object (SO) rules**.
   - SO rules are dynamically loaded detection rules, often used for advanced threat detection.

4. **-i eth1** → Sets the network interface to monitor (eth1).
   - If you have multiple interfaces (eth0, wlan0), you must choose the one where network traffic is captured.

5. **-A alert_fast** → Chooses the **alert output format** (alert_fast).
   - Alert_fast provides a compact, single-line format for alerts.

6. **> alert_full.txt** → Redirects all output (alerts) to a file (alert_full.txt).
   - This ensures that Snort logs alerts into the file instead of printing them in the terminal.

7.2. Downloading and Parsing snort logs

The Snort-generated log file (alert_full.txt) is processed using a Python script that extracts key fields using regular expressions. The extracted data is then stored in a structured Pandas DataFrame for further analysis.

7.3. Identifying Key Information in Alerts

Each alert in the Snort logs contains:

- Timestamp: When the alert was triggered.
- GID, SID, Rev: Identifiers for Snort rules.
- Message: Description of the detected event.
- Protocol: The type of traffic (ICMP, UDP, etc.).
- Source and Destination IPs: The origin and target of the suspicious activity.
- Source and Destination Ports: Relevant for TCP/UDP traffic.

Sample extracted alert:

```
01/30-12:09:24.621409 [**] [116:434:1] "(icmp4) ICMP ping Nmap" [**] [Priority: 3]
{ICMP} 192.168.0.28 -> 192.168.0.29
```

1. Timestamp

- **01/30-12:09:24.621409** → Date and time of the alert (**MM/DD-HH:MM:SS** format).

- Snort captured this alert at **12:09:24** on **January 30th**.

2. Snort Rule Metadata

- **[116:434:1]** → Snort **Generator ID (GID), Signature ID (SID), and Revision Number**:
  - **116** → Generator ID (GID): Identifies the rule category.
  - **434** → Signature ID (SID): Identifies the specific rule triggered.
  - **1** → Revision number of the rule.

3. Alert Message

- **"(icmp4) ICMP ping Nmap"** → The message from the Snort rule:
  - **icmp4** → Indicates it's an **IPv4 ICMP packet**.
  - **ICMP ping Nmap** → This suggests that Snort detected an **Nmap ICMP ping scan**.
  - Nmap's **ICMP scan (-sn)** sends ICMP Echo Requests (ping) to discover live hosts on the network.

4. Priority Level

- **[Priority: 3]** → This rule has a **priority level of 3**.
  - **Priority 1** → Critical (e.g., exploit attempts).
  - **Priority 2** → Important but less severe.
  - **Priority 3** → Lower priority, likely **network scanning**.

5. Protocol

- **{ICMP}** → The detected packet is an **ICMP (ping) message**.

6. Source & Destination

- **192.168.0.28 -> 192.168.0.29**
  - **192.168.0.28** → The **source IP** (the attacker machine).
  - **192.168.0.29** → The **destination IP** (the target of the ICMP scan).

8. Data from Snort Alerts

8.1. Tools and Libraries for Data Visualization

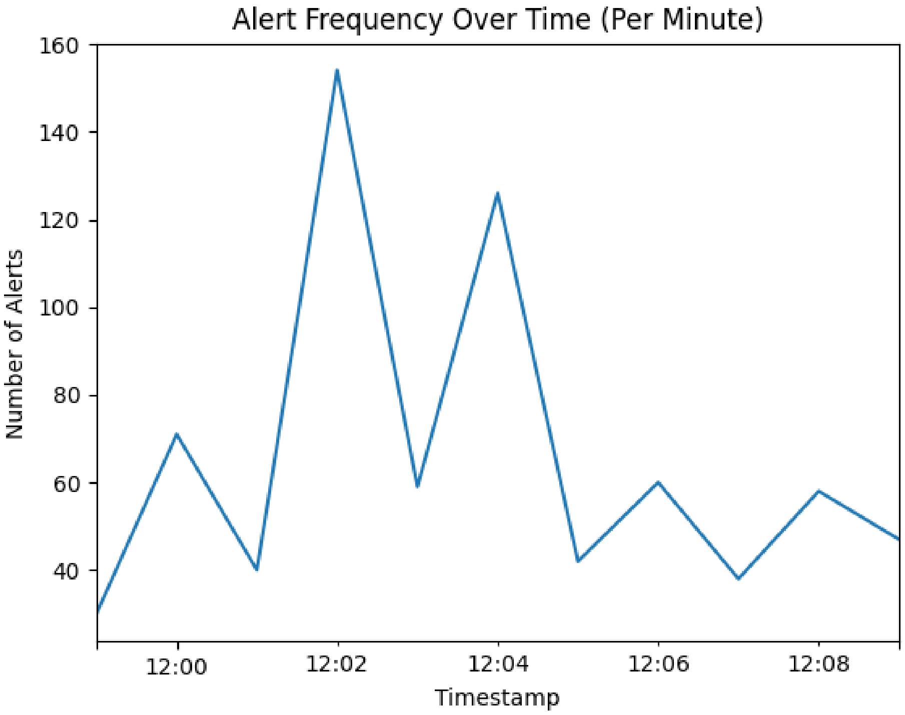
The following Python libraries were used for visualization:

- Matplotlib: Line plots, bar charts, and pie charts.
- Pandas: Data manipulation and analysis.

8.2. Visuals from snort alerts

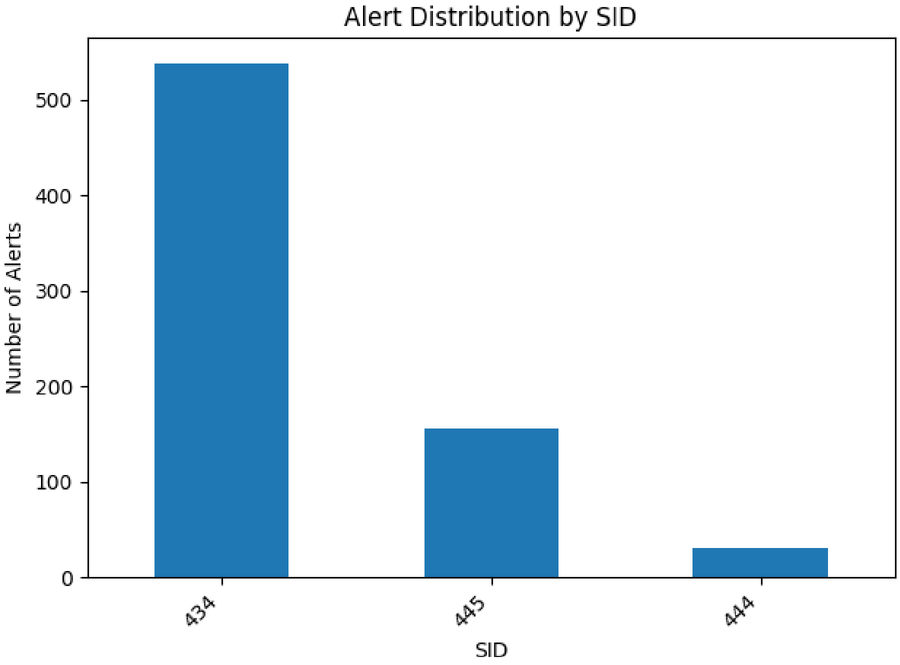
Several visualizations were generated:

1. Alert Frequency Over Time (Per Minute)



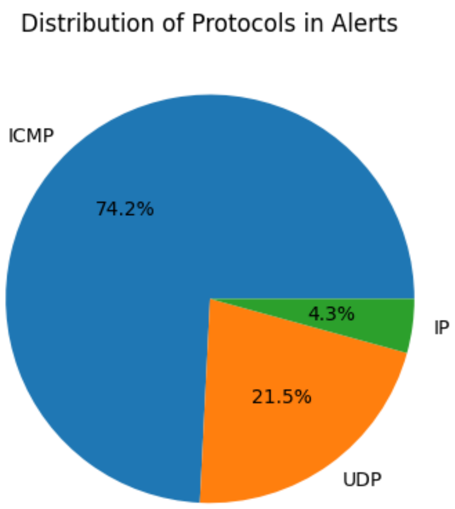
Shows spikes in attack activity over time.

2. Alert Distribution by SID



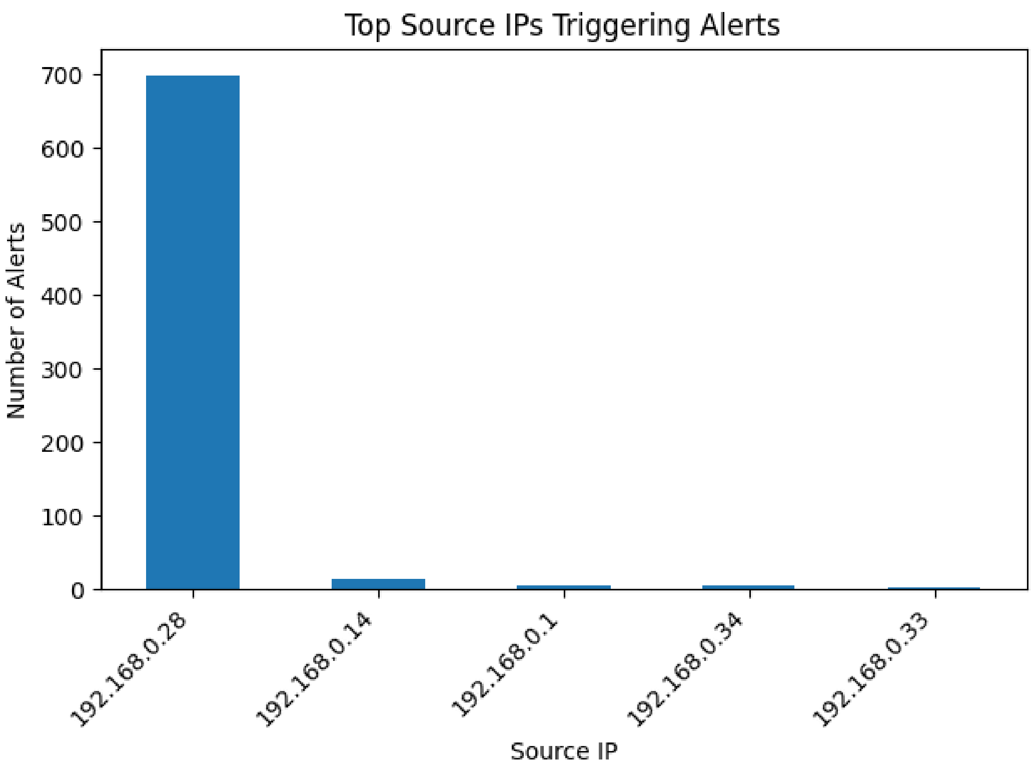
Highlights which Snort rules were triggered most frequently

3. Protocol Distribution



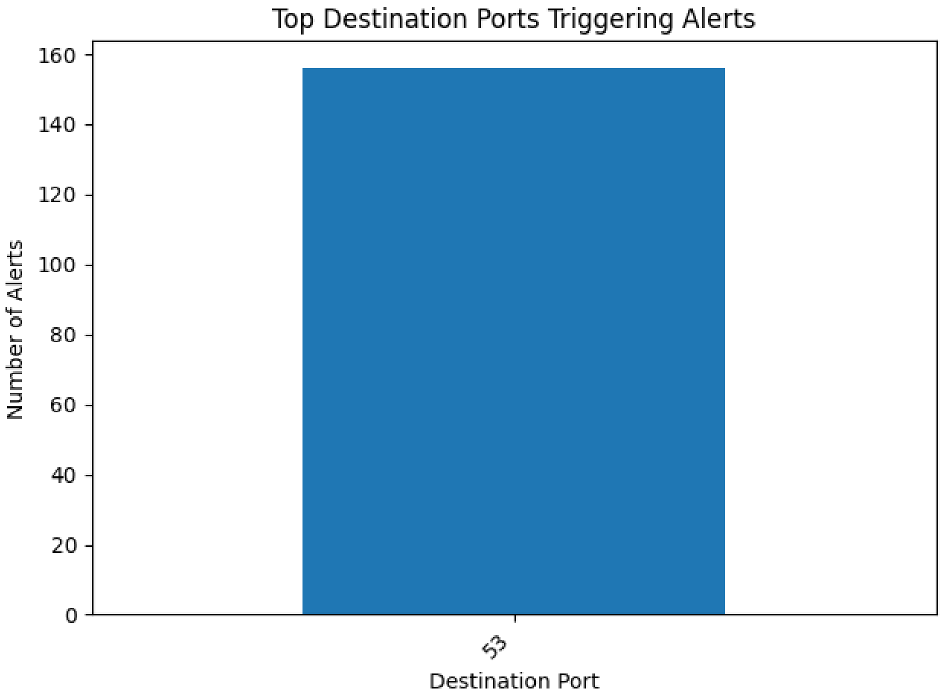
Displays the proportion of ICMP, UDP, and IP-based alerts.

4. Top Source IPs Triggering Alerts



Identify the most active attack sources.

5. Top Destination Ports Triggering Alerts



Reveals which ports were most frequently targeted.

8.3. Analyzing the results

The analysis of `alert_full.txt` revealed the following:

Alert Frequency Over Time (Per Minute):

Timestamp	Number of alerts
2024-01-30 11:59:00	30
2025-01-30 12:00:00	71
2025-01-30 12:01:00	40
2025-01-30 12:02:00	154
2025-01-30 12:03:00	59
2025-01-30 12:04:00	126
2025-01-30 12:05:00	42
2025-01-30 12:06:00	60
2025-01-30 12:07:00	38
2025-01-30 12:08:00	58
2025-01-30 12:09:00	47
Freq: min	

Alert Distribution by SID:

SID	Number of alerts
434	538

445	156
444	31

Distribution of Protocols in Alerts:

Protocol	Number of alerts
ICMP	538
UDP	156
IP	31

Top Source IPs Triggering Alerts:

Source IP	Number of alerts
192.168.0.28	699
192.168.0.14	14
192.168.0.1	5
192.168.0.34	5
192.168.0.33	2

Top 10 Destination Ports Triggering Alerts:

Destination Port	Number of alerts
53	156

- ICMP attacks (Nmap pings) dominated the logs, accounting for 538 alerts.
- UDP-based attacks targeted port 53 (DNS), with 156 occurrences.
- The most aggressive attacker IP was 192.168.0.28, generating 699 alerts.
- The highest alert frequency occurred at 12:02 PM, with 154 alerts in one minute.

1. **Alert Frequency Over Time:** Over the 10-minute simulation period, Snort generated a total of 725 alerts. The alert frequency varied over time, with notable spikes at 12:02 and 12:04. While the precise cause of these fluctuations was not investigated in detail, factors like operating system thread scheduling, network conditions, or target system behavior could contribute to such variations. A more granular analysis, resource monitoring, or packet-level inspection with Wireshark could be used for future, more in-depth studies.
2. **Alert Type Distribution:** Rule descriptions for each SID:
   - SID 434: ICMP ping Nmap –This detects ICMP ping sweeps or floods.
   - SID 445: UDP flood – This detects excessive UDP traffic, triggered by your UDP flood attack.
   - SID 444: IP – This represents generic IP traffic or events (like the IPv4 option set mentioned earlier).

3. **Protocol Distribution:** High number of ICMP and UDP alerts corresponds to the ping flood and UDP flood attacks.
4. **Source/Destination Analysis:** The dominant source IP is my attacker VM and that port 53 (DNS) is the primary destination port, in line with my DNS amplification attack.

9. Conclusion

9.1. Summary

This project successfully demonstrated Snort's ability to detect ICMP flood, UDP flood, and DNS amplification attacks. The analysis of the Snort logs provided valuable insights into the frequency, types, and sources of these attacks. Further, it is necessary to experiment with different attack types and explore Snort's intrusion prevention system (IPS) mode where rules can also block attacks and not just alert on them. Implementing and testing the efficacy of other IDS such as Suricata is also envisioned. But primarily this projects also shows the use of snort as an IDS for companies.

9.2. Key Takeaways from the Project

- Snort logs are a valuable source for security monitoring and threat detection.
- Automating log analysis with Python improves efficiency in identifying attack trends.
- Visualizing data helps in quickly identifying and interpreting security threats.

Sources

Snort Rule Downloads: https://www.snort.org/downloads/#rule-downloads

Snort YouTube Tutorial: https://www.youtube.com/watch?v=j7Wapw3Gxvg

Snort Rule Actions - Promiscuous Mode: https://docs.snort.org/rules/headers/actions?search=promis

Snort3 Lua Script: https://github.com/snort3/snort3/blob/master/lua/snort.lua

PulledPork GitHub Repository: https://github.com/shirkdog/pulledpork