Saint Joseph University of Beirut

National Institute of Telecommunications and Informatics


**Master Research Project**

# Assessing the Potential of Suricata Under Multi-Layer Denial-of-Service Attacks

Submitted by

Michaela El Rif
michaela.rif@net.usj.edu.lb

Andrew Zgheib
andrew.zgheib@net.usj.edu.lb

**Submission Date:** June 1, 2026
**Supervisor:** Prof. Gerard Chalhoub

**Abstract**

Denial of Service (DoS) attacks are a significant threat to network security, targeting the availability of services by overwhelming them with traffic. One of the main tools used to detect and mitigate such attacks are Intrusion Detection Systems (IDS). Suricata is an open-source IDS that is designed to handle high volumes of traffic thanks to its multi-threaded architecture, and provide real-time detection of DoS attacks. However, IDS solutions like Suricata have their limitations in terms of detecting complex patterms and combinations of DoS attacks. This paper studies the performance of Suricata under multi-layer DoS attacks, while proposing remediations on top of Suricata.

**Keywords:** Suricata, Intrusion Detection Systems, Denial-of-Service Attacks, Network Security, Performance Evaluation

# Contents


**1 Introduction** **1**
- 1.1 Background . . . 1
- 1.2 Objectives . . . 1
- 1.3 Scope and Constraints . . . 1
- 1.4 Expected Results . . . 2

**2 Literature Review** **3**
- 2.1 Types of Denial-of-Service Attacks and Attack Tools . . . 3
- 2.2 Intrusion Detection Systems Overview . . . 3
- 2.3 Effectiveness of Suricata . . . 4
- 2.4 Detection of UDP Flood Attacks . . . 4
- 2.5 Correlation and Advanced Detection Approaches . . . 4

**3 Technical Background** **6**
- 3.1 Denial-of-Service Attacks . . . 6
  - 3.1.1 Volumetric Attacks . . . 6
  - 3.1.2 Protocol Attacks . . . 6
  - 3.1.3 Application-Layer Attacks . . . 6
- 3.2 Suricata . . . 6
  - 3.2.1 Processing Pipeline . . . 7
  - 3.2.2 Rule Syntax . . . 7

**4 Methodology** **9**
- 4.1 Functionalities . . . 9
- 4.2 Testbed Architecture . . . 9
- 4.3 Tools and Technologies . . . 10
- 4.4 Virtual Machine Specifications . . . 10
- 4.5 Suricata Configuration . . . 10
  - 4.5.1 Individual Denial-of-Service Attacks . . . 11
  - 4.5.2 Combined Distinct Denial-of-Service Attacks . . . 15
- 4.6 Attack Scenarios . . . 15
  - 4.6.1 Scenario 1: Individual Denial-of-Service Attacks . . . 16
  - 4.6.2 Scenario 2: Combined Distinct Denial-of-Service Attacks . . . 17

**5 Results** **19**
- 5.1 Attack Scenarios Results . . . 19
  - 5.1.1 Scenario 1 Results . . . 20
  - 5.1.2 Scenario 2 Results . . . 21
- 5.2 Suricata Results . . . 22

**6 Recommendations** **23**
- 6.1 Security Information and Event Management . . . 23
- 6.2 Anomaly-Based Intrusion Detection Systems . . . 23

**7 Conclusion** **24**

# Chapter 1

# Introduction

## 1.1 Background

Denial-of-Service (DoS) attacks are a notable threat to network infrastructure. The primary goal of such attacks is to target the availability of a service in order to make it inaccessible to legitimate users by overwhelming the target with traffic. Attackers can even combine multiple types of DoS attacks by similarity, in order to evade detection [1]. The impact of these attacks does not stop here, as they can also lead to financial losses, damage to reputation, and even legal consequences. This pushes organizations to explore and put in place security solutions that can mitigate or even block DoS attacks.

Intrusion Detection Systems (IDS) are one of the main security solutions that organizations can use to detect and mitigate DoS attacks. IDS solutions provide real-time monitoring and detection of network traffic in order to identify and alert administrators about suspicious activities based on predefined rules. Suricata is an open-source IDS developed by the Open Information Security Foundation (OISF) that is capable of handling high-volume traffic thanks to its multi-threaded architecture and native support for modern hardware [2].

## 1.2 Objectives

The main objective of this project is to evaluate the performance of Suricata under various DoS attack scenarios. These scenarios will mainly revolve around individual DoS attacks, as well as combinations of distinct flood-based DoS attacks. In order to achieve this, a testbed should be set up that will help simulate the various DoS attack scenarios, while also allowing us to analyze Suricata's performance in detecting these attacks. This will identify any limitations that Suricata may have during the detection of DoS attacks, specifically in its custom ruleset.

## 1.3 Scope and Constraints

The scope of this project is limited to evaluating the detection performance of Suricata under flood-based DoS attack scenarios. The various types of DoS attacks will be discussed

in Chapter 3. Even though Suricata has prevention capabilities, only its detection performance will be evaluated. Additionally, since this project is carried out in a virtualized environment, normal network traffic is negligible compared to a real-world environment, since our testbed is isolated with little to no external connectivity. Therefore, the thresholds set in Suricata's custom ruleset will be adjusted to fit the traffic of the DoS attack scenarios, not the normal traffic.

## 1.4 Expected Results

Suricata is expected to detect flood-based DoS attacks when the configured thresholds in the custom ruleset are met. It is also expected for Suricata to not label normal traffic as a DoS attack, since this would be considered as a false positive. Moreover, combinations of distinct and low-traffic flood-based DoS attacks that individually remain below Suricata's configured thresholds but collectively generate high traffic have no expectations, both in terms of detection and non-detection, since this is one of the main points of the project scope.

# Chapter 2

# Literature Review

This paper discusses the steps implemented to be able to study the effectiveness of Suricata in detecting different types of DoS attacks and their combinations using tools to simulate the attacks. Many studies focus on the evaluation of Suricata as an IDS for DoS attacks. These works generally examine how effectively Suricata can detect different flooding techniques, as well as its limitations when attacks become increasingly complex or when multiple attack types are combined.

The existing literature shows that Suricata can be effective in identifying flood-based attacks, but it also highlights the need for more advanced correlation mechanisms when the attack behavior is distributed across several vectors.

## 2.1 Types of Denial-of-Service Attacks and Attack Tools

Since DoS attacks threaten the availability of a service, it is important to take the right measures to protect the security infrastructure from becoming vulnerable, as mentioned [3].

There are many types of DoS attacks, but they can fall into three main categories based on exploitation type. The paper states that when an attack saturates the network bandwidth, it is considered a volumetric attack, such as UDP and ICMP flood attacks. In addition, there are protocol attacks that take advantage of vulnerabilities present in the protocol, such as SYN floods. And finally, application-layer attacks that target software weaknesses, such as web applications.

Several tools could be used to generate DoS attacks known to have a potential for high impact, such as GoldenEye, LOIC (Low Orbit Ion Cannon), HOIC (High Orbit Ion Cannon), and hping3.

## 2.2 Intrusion Detection Systems Overview

According to [3], an IDS is used to detect suspicious or unauthorized traffic entering a network that may pose a threat to the network or system.

An IDS has two main categories: Network Intrusion Detection Systems (NIDS) that monitor the network flow, and Host-based Intrusion Detection Systems (HIDS) that are specific to each host. The most well-known and open-source NIDS are Suricata and Snort.

## 2.3 Effectiveness of Suricata

[4] studies the performance and detection success of Suricata for a Small-to-Medium Enterprise (SME) when facing several types of attacks, including UDP, ICMP, and SYN floods, as well as on the comparison with other NIDS.

To evaluate the performance of Suricata under heavy flow from DoS attacks, CPU consumption and memory were monitored. The results show an increment in CPU consumption but relatively low memory usage. In addition, Suricata achieved a higher true positive rate and a lower false negative rate than other NIDS, thus outperforming them.

These results show that Suricata possessed a high ability to monitor DoS attacks under heavy load, with successful detection of UDP, ICMP, and SYN floods.

## 2.4 Detection of UDP Flood Attacks

[5] studies the effectiveness of Suricata to specifically detect UDP floods targeting a web server. Their paper shows a custom detection rule in Suricata, leading to a more personalized and accurate detection of a UDP flood activity than the default rules that come with Suricata:

```
alert udp any any -> $HOME_NET any (msg: "Multiple UDP Packets from
Random Source IPs Detected"; sid: 1000003; threshold: type limit,
track by_src, seconds 60, count 50)
```

This rule shows the detection of a UDP flood DoS attack if the arriving number of packets per minute is equal or greater than 50 for each different IP, hence the use of `by_src`.

The tests show a 100% accuracy for Suricata to detect excessive volume of UDP packets to a targeted web server.

## 2.5 Correlation and Advanced Detection Approaches

The functionalities of Suricata are limited to tracking only the traffic that is happening at a specific monitoring point, meaning it does not have visibility into what is happening in the whole enterprise infrastructure. This will make it harder to differentiate between suspicious and legitimate traffic.

The solution to this problem would be a security system that collects logs and events from all security devices in order to aggregate them and correlate them. This can be done by a Security Information and Event Management (SIEM) solution that would be added alongside an IDS and other security solutions, as highlighted by [3].

Suricata is a signature-based IDS, meaning it detects alerts based on a predefined set of rules. This creates a limitation to it if a DoS attack is happening, but it has a new

way of conducting it that could be new or sophisticated. This is a risk in a continuously evolving world of cyberattacks. For this reason, [6] mentions the importance of using Machine Learning-based IDS that helps analyze behavior rather than solely depending on signatures.

# Chapter 3

# Technical Background

## 3.1 Denial-of-Service Attacks

DoS attacks are categorized based on the attack method used and the targeted resources. A distinction can be made between volumetric attacks, protocol attacks, and application-layer attacks.

### 3.1.1 Volumetric Attacks

Volumetric attacks aim to send large volumes of traffic to the target system in an attempt to saturate all the available bandwidth between that target and its users. Examples include UDP floods and ICMP floods.

### 3.1.2 Protocol Attacks

Protocol attacks have as their primary objective the exploitation of specific weaknesses in network protocols present on the target system, consuming resources such as CPU, memory, or connection tables. Examples include SYN floods, SSL/TLS floods, and Ping of Death attacks.

### 3.1.3 Application-Layer Attacks

Application-layer attacks target web applications or other software applications, in order to exploit vulnerabilities when no mitigation strategies are in place. Examples include HTTP floods, Slowloris attacks, and SIP floods [7].

## 3.2 Suricata

Suricata's workflow can be divided into three main stages: packet processing, detection, and output. These three stages are implemented through a series of modules that work together to capture, analyze, and generate alerts based on the network traffic.

### 3.2.1 Processing Pipeline

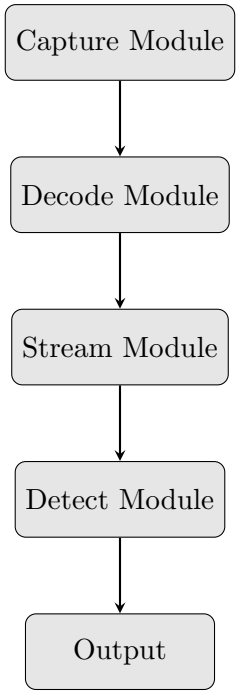


Figure 1: Simplified Suricata packet processing pipeline

As shown in Figure 1, packets flow from the capture module to the decode module, a stream module, the multi-threaded detection module, and finally to the output module.

The capture module is responsible for detecting and acquiring packets arriving to the network interface, while the decode module is responsible for parsing the packets and extracting relevant information such as protocol headers and payloads. Moreover, the stream module is responsible for packet reassembly and flow tracking, before passing the packets to the detection module. The latter initializes the detection plugins and runs the packets through the corresponding rulesets, and parses the rules to generate the relevant alerts, before sending these alerts to the output module that writes alerts to output files, specifically `eve.json` and `fast.log` [8].

### 3.2.2 Rule Syntax

Suricata relies on configured rulesets to detect malicious traffic. Each ruleset is made up of individual rules that define the conditions for matching specific patterns in the network traffic. Suricata uses the following syntax for all of its rules:

```
action proto src_ip src_port direction dst_ip dst_port (options;)
```

Listing 3.1: Rule syntax for Suricata

The `action` keyword is important in order to specify what should happen when a rule is matched. Some common actions include `alert`, which generates an alert, `drop`, which drops the packet and alerts, `pass`, which allows the packet to pass and stops any additional packet inspection, and `reject`, which drops the packet and sends an RST/ICMP

unreachable message.

After deciding on an action, a protocol should be defined. Several protocols are available such as ip, tcp, udp, and icmp.

The protocol is followed by source and destination IP addresses and ports, where the IP address can be specified as a single IP address, a list of IP addresses, a CIDR block, or a predefined variable, and the ports can be specified as a single port, a list or a range of ports, or a predefined variable. The `!` operator can be used to negate the IP address or port, meaning that the rule will match any traffic that does not match the specified IP address or port.

Between the source and destination fields, the direction operator should be set to define the direction of traffic that should be matched by the rule. Three direction operators are currently available: `->` for unidirectional traffic from source to destination, `=>` for unidirectional traffic from destination to source like `->` but with optional matching on bidirectional traffic, and `<>` for bidirectional traffic between source and destination.

Finally, the options field is used to specify the conditions that should be met for the rule to be matched, or to generate feedback in the outputs [9].

It should be noted that variables and ruleset paths should be defined in the Suricata configuration file(s), ending in `.yaml`.

# Chapter 4

# Methodology

## 4.1 Functionalities

In this project, the main functionalities that will be implemented are as follows:

- Installing and setting up Suricata on the victim Virtual Machine (VM) as the IDS monitoring the incoming traffic to that VM
- Launching the attacks from an attacker VM
- Using tools or scripts to launch the attacks rather than feeding Suricata preexisting Packet Capture (PCAP) files
- Simulating individual and combined distinct flood-based DoS attacks
- Evaluating the performance of Suricata under these attacks by analyzing the generated alerts and the resource consumption of the victim VM

## 4.2 Testbed Architecture

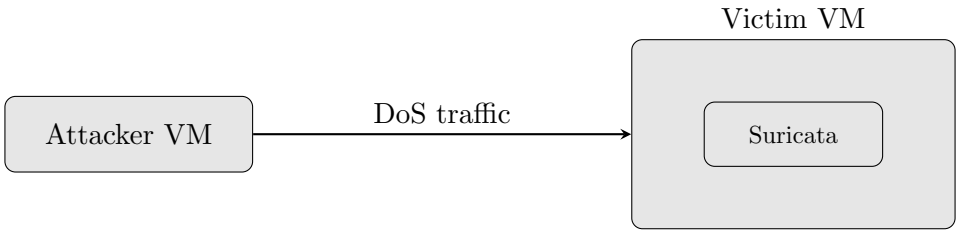


Figure 2: Logical architecture of the virtualized testbed

As illustrated in Figure 2, the architecture consists of two VMs. The first VM acts as the attacker, enabling the generation and simulation of DoS attack scenarios. The second VM acts as the victim and hosts the Suricata IDS. This VM receives the DoS traffic from the attacker VM, before passing it to Suricata for possible detection and alerting.

## 4.3 Tools and Technologies

The testbed was built using VMware Workstation, facilitating the creation and management of the two VMs.

The attacker VM runs Kali Linux, which is a Debian-based Linux distribution that is widely used for penetration testing and security research. Kali Linux was picked as the Operating System (OS) of the attacker VM since it comes pre-installed with a wide range of tools for various security tasks, including DoS attack generation [10].

The victim VM runs Ubuntu Desktop, which is a popular Linux distribution that is widely used for both personal and professional purposes. Ubuntu Desktop was picked as the OS of the victim VM since it provides a user-friendly interface for installing and configuring Suricata, as well as for analyzing the generated alerts [11].

Suricata, the focus of this project, was configured and installed on the victim VM, while `hping3` was used on the attacker VM to generate the various DoS attack scenarios.

## 4.4 Virtual Machine Specifications

| **Specification** | **Attacker VM** | **Victim VM** |
|---|---|---|
| Operating System | Kali Linux 2026.1 | Ubuntu 26.04 LTS Desktop |
| CPU Cores | 8 vCPUs | 2 vCPUs |
| Memory | 8 GB | 4 GB |
| Disk | 80 GB | 20 GB |

Table 4.1: Specifications for the Attacker and Victim VMs

As shown in Table 4.1, the attacker VM is allocated more hardware resources than the victim VM. It has four times the number of CPU cores, double the amount of memory, and four times the disk space compared to the victim VM. The hardware resources of the victim VM are the lowest possible values while still allowing Suricata to run and process the incoming traffic. This was done on purpose to observe a strong reaction from the victim VM when it is under DoS attack, since the testbed has its own limitations in terms of traffic generation and processing, and the victim VM is expected to be overwhelmed by the generated DoS attacks. Allocating more resources to the victim VM would have made it more resilient to DoS attacks, which would have made it harder to analyze the results and draw conclusions about Suricata's performance. This setup will be useful for the analysis of the results in Chapter 5.

## 4.5 Suricata Configuration

After installing and setting up Suricata on the victim VM, the `$HOME_NET` variable was set to be equal to the subnet of the victim VM, and a custom ruleset was created to detect the various DoS attack scenarios that will be generated from the attacker VM.

```
HOME_NET: "[192.168.190.0/24]"
```

Listing 4.1: Suricata configuration for the `$HOME_NET` variable

The ruleset is made up of three rules, each targeting a specific type of flood-based DoS attack.

### 4.5.1 Individual Denial-of-Service Attacks

Three flood-based DoS attacks were chosen: UDP floods, ICMP floods, and SYN floods, and each of the three rules is designed to detect one of these attacks.

```
alert udp any any -> $HOME_NET any (msg:"Possible UDP Flood";
threshold: type threshold, track by_src, count 6000, seconds 1; sid
:1000001; rev:1;)

alert icmp any any -> $HOME_NET any (msg:"Possible ICMP Flood";
threshold: type threshold, track by_src, count 6000, seconds 1; sid
:1000002; rev:1;)

alert tcp any any -> $HOME_NET any (msg:"Possible SYN Flood"; flags:
S; threshold: type threshold, track by_src, count 6000, seconds 1;
sid:1000003; rev:1;)
```

Listing 4.2: Custom Suricata ruleset

As shown in Listing 4.2, the action in all of the rules is `alert`. This is done to ensure that the rules only generate alerts when they are matched, without taking any preventive actions such as dropping or rejecting the traffic, in order to respect the scope of the project, as mentioned in Section 1.3.

Additionally, the protocol specified in each rule matches the type of DoS attack that the rule is designed to detect.

Following the protocol, the source and destination IP addresses are set to `any` and `$HOME_NET`, respectively, with a unidirectional flow. This was defined in order to match any traffic that is directed towards the victim VM, regardless of the source IP address [9]. No specific ports were specified since the generated DoS attacks will originate from random ports, and will target random ports. Moreover, the generated attacks will not target a specific service, but rather the availability of the victim VM as a whole.

The common parameters of the rules are defined as follows:

- `msg`: This parameter specifies the message that will be included in the alert.
- `threshold`: This parameter specifies the threshold for matching the rule and the tracking method used to count events. In this case, it is set to track events per source IP address (`by_src`) and trigger an alert if 6000 packets matching the rule are seen within a 1-second interval.
- `sid`: This parameter specifies a unique identifier for the rule.

- `rev`: This parameter specifies the revision number of the rule. Since this is the first version of the rules, the revision number is set to 1 [12].

In the third rule, the `flags` parameter is set to `S` to match TCP SYN packets.

The `by_src` tracking method used in the `threshold` parameter, which counts events per source IP address, will not trigger an alert if the traffic is generated from multiple sources, even if the total traffic exceeds the defined threshold. Therefore, a little refinement was made to the ruleset to account for this point, by changing the tracking method to `by_rule`, which counts events globally for each rule, regardless of the source IP address, and will trigger an alert even if they are generated from multiple sources [13] as long as the total volume of packets matching that specific rule exceeds the defined threshold.

```
alert udp any any -> $HOME_NET any (msg:"Possible UDP Flood"; flow:
stateless; threshold: type threshold, track by_rule, count 6000,
seconds 1; sid:1000001; rev:2;)

alert icmp any any -> $HOME_NET any (msg:"Possible ICMP Flood";
itype:8; threshold: type threshold, track by_rule, count 6000,
seconds 1; sid:1000002; rev:2;)

alert tcp any any -> $HOME_NET any (msg:"Possible SYN Flood"; flags:
S; flow:stateless; threshold: type threshold, track by_rule, count
6000, seconds 1; sid:1000003; rev:2;)
```

Listing 4.3: Custom Suricata ruleset

Additionally, as shown in Listing 4.3, two new parameters were added to the rules in order to make them more granular and specific to the generated DoS attack scenarios, avoiding false positives from normal traffic. First, the `flow` parameter was added to the first and third rules, and set to `stateless`, in order to match any traffic that meets the other conditions, regardless of its state in a connection, since the generated DoS attacks will not follow a normal connection flow, and will consist of random packets that do not belong to any established connection. Moreover, the `itype` parameter was added to the second rule to specify that only ICMP Echo Request packets (type 8) [14, 15] should be matched by the rule, since these are the packets used in ICMP flood attacks.

The `count` and `seconds` values were set to 6000 and 1, respectively, based on the volume of traffic, measured in pps, that would exhaust the victim's resources. Due to the limitations of the virtualized environment, the CPU was the only exhausted resource.

In real-world environments, an abnormal usage of CPU differs from one system to another [16]. Therefore, normal traffic was considered to be any traffic that is causing no more than 35% CPU consumption, while DoS attack traffic was considered to be any traffic that causes a higher consumption. This was measured by generating each type of attack individually and running the corresponding script on the victim VM.

```
iptables -I INPUT -p udp -d <victim_ip>

while true; do
  a=$(sudo iptables -L INPUT -v -x -n --line-numbers | awk '$1==1 {
print $2}')
  sleep 1
  b=$(sudo iptables -L INPUT -v -x -n --line-numbers | awk '$1==1 {
print $2}')
  echo "$((b - a)) UDP pps"
done
```

Listing 4.4: Script for measuring the traffic of UDP floods

```
iptables -I INPUT -p icmp --icmp-type echo-request -d <victim_ip>

while true; do
  a=$(sudo iptables -L INPUT -v -x -n --line-numbers | awk '$1==1 {
print $2}')
  sleep 1
  b=$(sudo iptables -L INPUT -v -x -n --line-numbers | awk '$1==1 {
print $2}')
  echo "$((b - a)) ICMP Echo-Request pps"
done
```

Listing 4.5: Script for measuring the traffic of ICMP floods

```
iptables -I INPUT -p tcp --tcp-flags SYN SYN -d <victim_ip>

while true; do
  a=$(sudo iptables -L INPUT -v -x -n --line-numbers | awk '$1==1 {
print $2}')
  sleep 1
  b=$(sudo iptables -L INPUT -v -x -n --line-numbers | awk '$1==1 {
print $2}')
  echo "$((b - a)) TCP SYN pps"
done
```

Listing 4.6: Script for measuring the traffic of TCP SYN floods

Each of the above scripts uses `iptables` to create a rule that matches the specific type of traffic generated by the corresponding DoS attack. It then continuously measures the number of pps matching that rule by checking the packet counters before and after a 1-second interval. This can give a rough estimation of the traffic generated by each type of attack since little to no legitimate traffic is present.

Each script was ran for thirty seconds over ten iterations, with a thirty-second cooldown between each iteration. The results were recorded to calculate the minimum, maximum, mean, and standard deviation of the measured traffic for each type of attack.

The CPU consumption measurements were taken using the `/proc/stat` file to evaluate the

impact of the generated DoS attacks on the victim VM. These measurements were taken while Suricata was running in order to account for the processing overhead introduced by Suricata on the victim VM.

First, each attack was generated from a single terminal in order to observe its individual impact in terms of CPU consumption.

| **Attack** | **Min (pps)** | **Max (pps)** | **Mean (pps)** | **Std Dev** |
|---|---|---|---|---|
| UDP Flood | 3404 | 3644 | 3518 | 79 |
| ICMP Flood | 2937 | 3932 | 3489 | 364 |
| SYN Flood | 4004 | 4390 | 4213 | 110 |

Table 4.2: Measured traffic for each attack from a single terminal

As shown in Table 4.2, the measured traffic varied across attack types, with mean values of 3518 pps for UDP floods, 3489 pps for ICMP floods, and 4213 pps for SYN floods.

| **Attack** | **Min (%)** | **Max (%)** | **Mean (%)** | **Std Dev** |
|---|---|---|---|---|
| UDP Flood | 10.20 | 19.72 | 12.17 | 3.28 |
| ICMP Flood | 17.53 | 29.09 | 23.04 | 3.25 |
| SYN Flood | 19.71 | 34.67 | 28.85 | 5.12 |

Table 4.3: Measured CPU consumption for each attack from a single terminal

As shown in Table 4.3, the CPU consumption of the victim VM varied across attack types, with mean values of 12.17% for UDP floods, 23.04% for ICMP floods, and 28.85% for SYN floods. Since the CPU consumption of the victim VM under each type of attack was underwhelming, the same attack was generated from two terminals at the same time, in order to observe the increase in CPU consumption.

| **Attack** | **Min (pps)** | **Max (pps)** | **Mean (pps)** | **Std Dev** |
|---|---|---|---|---|
| UDP Flood | 8334 | 9849 | 9117 | 577 |
| ICMP Flood | 7181 | 8538 | 8070 | 399 |
| SYN Flood | 8435 | 9676 | 9022 | 357 |

Table 4.4: Measured traffic for each attack from two terminals

As shown in Table 4.4, the measured traffic varied across attack types, with mean values of 9117 pps for UDP floods, 8070 pps for ICMP floods, and 9022 pps for SYN floods.

| **Attack** | **Min (%)** | **Max (%)** | **Mean (%)** | **Std Dev** |
|---|---|---|---|---|
| UDP Flood | 37.45 | 52.18 | 44.81 | 4.03 |
| ICMP Flood | 50.42 | 59.16 | 53.83 | 2.40 |
| SYN Flood | 58.02 | 71.12 | 64.57 | 3.84 |

Table 4.5: Measured CPU consumption for each attack from two terminals

As shown in Table 4.5, the CPU consumption of the victim VM varied across attack types, with peak values of 52.18% for UDP floods, 59.16% for ICMP floods, and 71.12% for SYN floods, which are considered abnormal. The same approach was applied to the generation of attacks from three terminals at the same time, but the results were not significantly different from the previous scenario due to the testbed limitations. This implies that a convergence is observed in the CPU consumption of the victim VM when generating DoS attacks from more than two terminals.

Based on the above results, a DoS attack can be considered as a threat to the virtualized environment if it is generated from two terminals, sending more than 7000 pps for each attack. However, a DoS attack generated from one terminal and sending around 3500 pps for each attack will not be considered as a threat, since it is causing an acceptable CPU consumption on the victim VM. Therefore, we can set the threshold for DoS detection for each attack between the two values, so it will be 6000 pps, and can be later on fine-tuned if the expected results are not matched, if needed. It is worth noting that the threshold for each attack is set to the same value, since they are close to each other in terms of pps and abnormality in CPU consumption.

### 4.5.2 Combined Distinct Denial-of-Service Attacks

The above assumptions in Subsection 4.5.1 for normal and abnormal traffic can be applied to combined distinct DoS attacks when each attack exceeds its predefined threshold, which will be detected by Suricata. However, if an attack does not exceed its predefined threshold, it will not be detected by Suricata, but it can still have an impact on the CPU consumption of the victim VM. This is because the generated traffic from that attack can still be considered as abnormal, since the combination of multiple attacks can lead to a higher CPU consumption than each attack individually, and can potentially exhaust the victim's resources. Consequently, the possibility of Suricata generating alerts based on correlation between the different types of DoS floods in Suricata was explored.

First, the idea of setting a global rule that matches any IP traffic was considered, by setting the `proto` option to `ip`. However, this would lead to a lot of false positives since the rule would not be granular enough to differentiate between normal traffic and abnormal traffic. Additionally, the possibility of using Lua scripting for detection was explored, but Lua is primarily used for packet and flow inspection, and is not designed for correlation of multiple events coming from different types of floods [17]. Finally, the idea of using the `xbits` keyword, a custom boolean flag that can be set and checked in the rules, was explored, but it requires predefined conditions and thresholds, which makes it difficult to set correlation rules for combined distinct DoS attacks, since a DoS attack is not fixed, and its volume varies for each flood attack [18].

For these reasons, it was concluded that there is no straightforward way to set correlation rules in Suricata for combined distinct DoS attacks.

## 4.6 Attack Scenarios

Each of the three attacks mentioned in the ruleset will be generated individually, as well as in combination with each other, to evaluate Suricata's performance under the two different scenarios, as stated in Section 1.2.

`hping3` will be used to generate the attacks, with the following commands:

```
hping3 --udp --flood -a 172.16.0.1 <victim_ip>
```

Listing 4.7: Command for generating UDP floods

```
hping3 --icmp --flood -a 172.16.0.1 <victim_ip>
```

Listing 4.8: Command for generating ICMP floods

```
hping3 -S --flood -a  172.16.0.1 <victim_ip>
```

Listing 4.9: Command for generating SYN floods

Each command consists of the following flags:

- `--udp`, `--icmp`, and `-S` to specify the type of attack being generated, which are UDP floods, ICMP floods, and SYN floods, respectively.
- `--flood` to send packets as fast as possible, without waiting for replies, which is essential for generating flood-based DoS attacks.
- `-a` to set the source IP address to a random IP address (in this case, `172.16.0.1`). This is done to simulate a more realistic attack scenario, where the attacker is not on the same subnet as the target.

### 4.6.1 Scenario 1: Individual Denial-of-Service Attacks

Two sub-scenarios will be considered for each type of attack in a scenario with individual DoS attacks:

1. Generating a single attack from a single terminal, sending around 3500 pps as shown in Table 4.2, which is below the predefined threshold of 6000 pps, as illustrated in Figure 3.
2. Generating the same attack twice, with each instance running simultaneously from a separate terminal, sending more than 7000 pps as shown in Table 4.4, which is above the predefined threshold of 6000 pps, as illustrated in Figure 4.

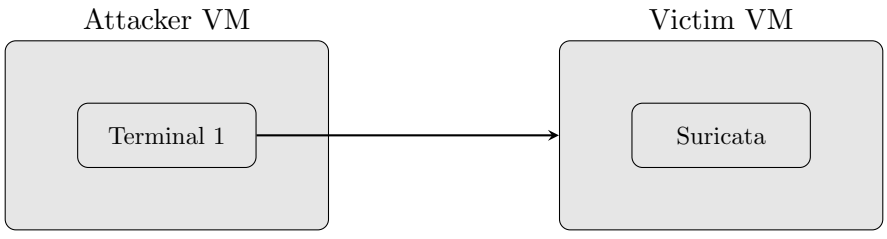


Figure 3: Sub-scenario 1.1: single attack instance

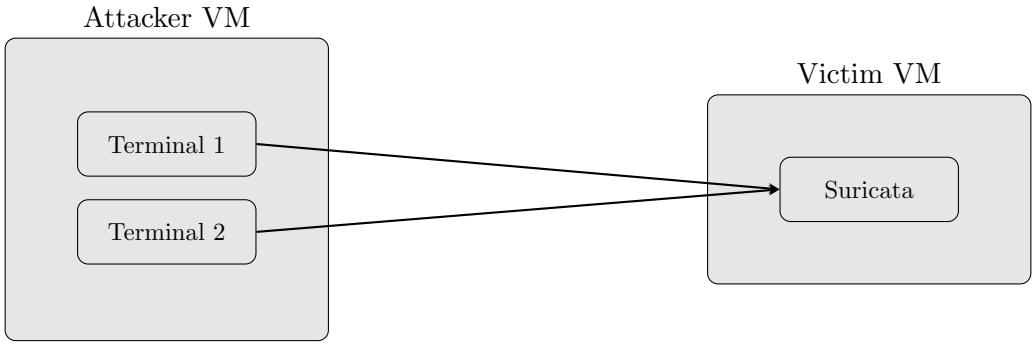


Figure 4: Sub-scenario 1.2: two attack instances

### 4.6.2 Scenario 2: Combined Distinct Denial-of-Service Attacks

Similarly to the first scenario of individual DoS attacks, two sub-scenarios will be considered for the scenario of combined distinct DoS attacks:

1. Generating each attack twice, with each instance running from a separate terminal and all attacks running simultaneously, which is above the predefined threshold of 6000 pps, as illustrated in Figure 5.
2. Generating the combination of distinct attacks, each running in a separate terminal, which is below the predefined threshold of 6000 pps, as illustrated in Figure 6.

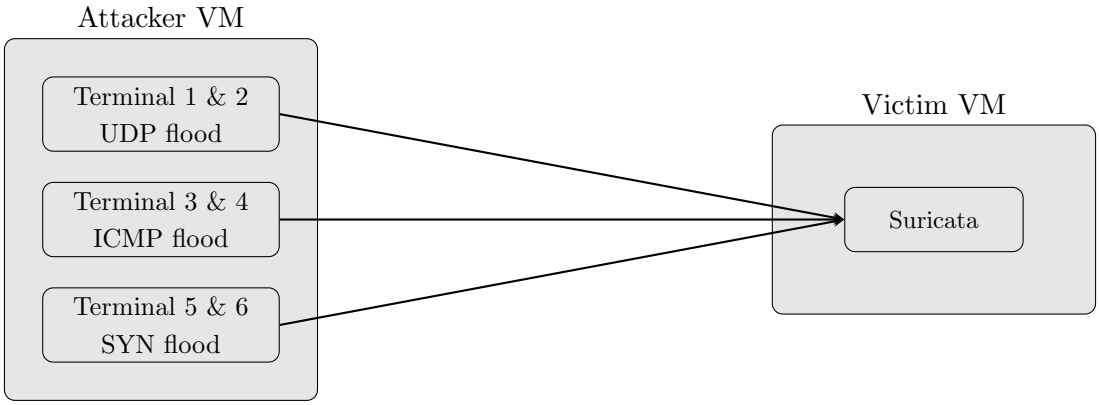


Figure 5: Sub-scenario 2.1: two instances per attack

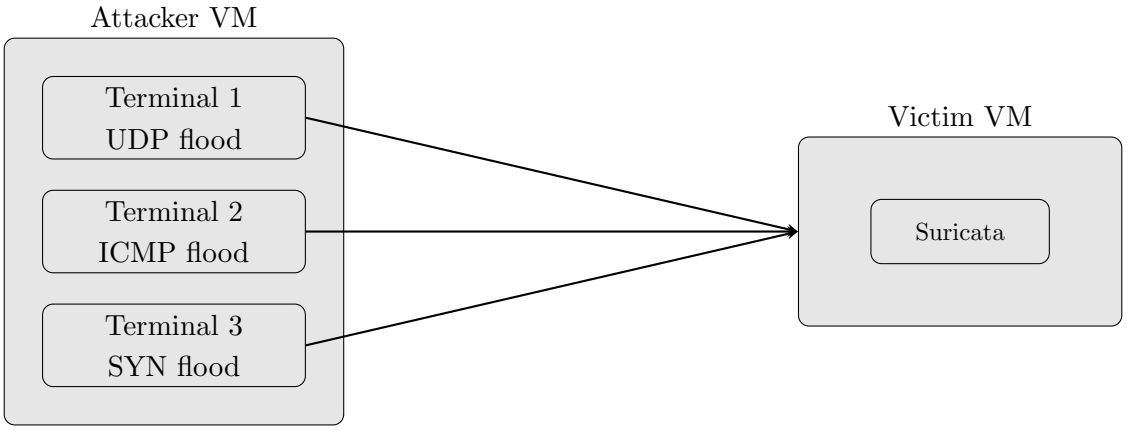


Figure 6: Sub-scenario 2.2: single instance per attack

# Chapter 5

# Results

## 5.1 Attack Scenarios Results

Each of the attack scenarios defined in the previous chapter, Section 4.6, was executed, and the results were analyzed to evaluate Suricata's performance in detecting DoS attacks in the virtualized environment. The CPU consumption of the victim VM was the main metric used to evaluate the impact of the generated DoS attacks, and to determine whether they are considered as threats or not. Additionally, the alerts generated by Suricata were analyzed to evaluate its detection capabilities for each attack scenario.

Before running any attack scenario, the mean CPU consumption was measured on the victim VM while it was idle with Suricata running in the background, with a value of 4.44%. Next, each attack scenario was executed, and the CPU consumption of the victim VM was measured during the execution of each scenario while recording whether Suricata generated an alert or not.

### 5.1.1 Scenario 1 Results

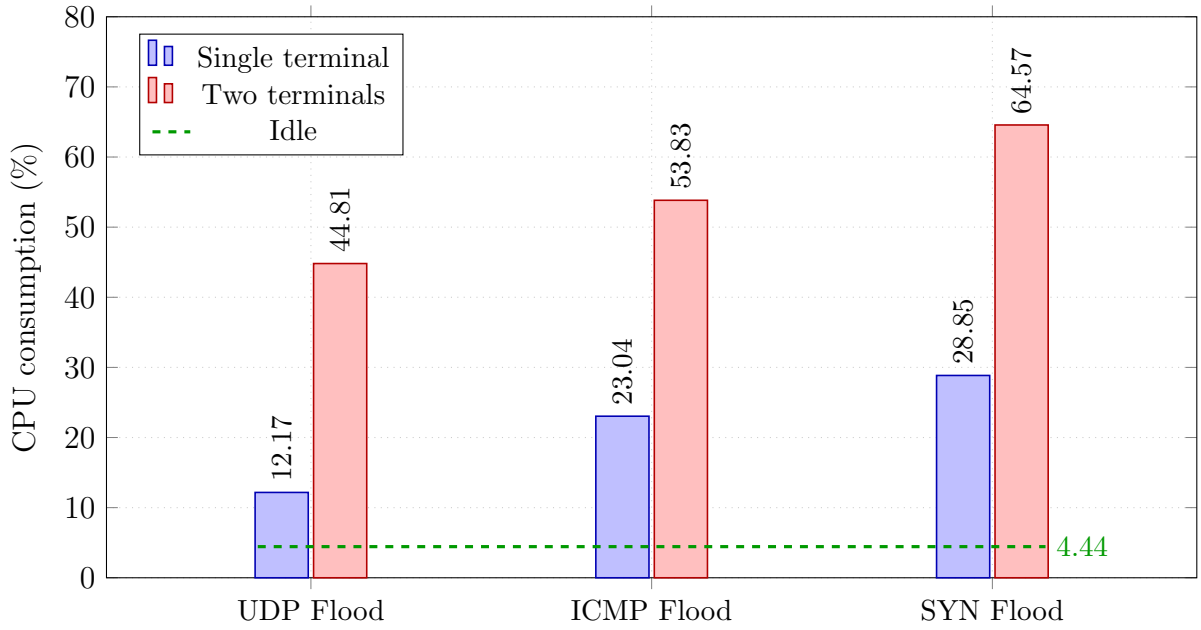


Figure 7: Mean CPU consumption on the victim VM for scenario 1

As shown in Figure 7, the CPU consumption of the victim VM varied across attack types and number of terminals. When the attack is generated from a single terminal, the CPU consumption of the victim VM is noticeably higher than the idle state of 4.44%, with a value of 12.17% for UDP floods, 23.04% for ICMP floods, and 28.85% for SYN floods, the three of them being lower than 35%, which is considered normal in this testbed, as stated in Subsection 4.5.1. However, when the attack is generated simultaneously from two terminals, the CPU consumption significantly increases and is considered abnormal, reaching 44.81% for UDP floods, 53.83% for ICMP floods, and 64.57% for SYN floods, indicating that the victim VM is under a DoS attack.

Suricata has correctly generated alerts for sub-scenario 1.2, since the traffic generated in this scenario was above the predefined threshold of 6000 pps, and the CPU consumption was considered abnormal. However, Suricata did not generate any alert for the idle scenario and sub-scenario 1.1, since the traffic generated in this case was below the predefined threshold of 6000 pps and the CPU consumption was normal, being lower than 35%. This indicates that Suricata is able to detect individual DoS attacks when it is impacting the victim VM by consuming its resources, while not generating false positives for normal traffic.

### 5.1.2 Scenario 2 Results

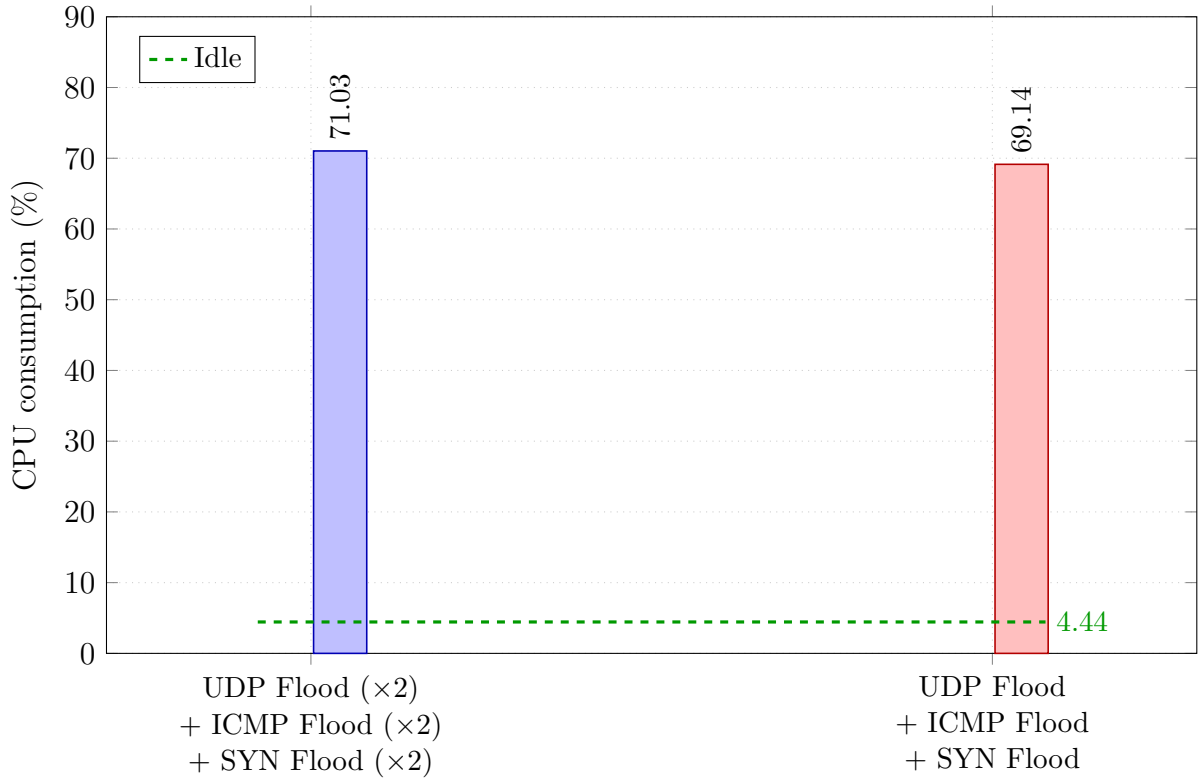


Figure 8: Mean CPU consumption on the victim VM for scenario 2

Figure 8 shows the CPU consumption of the victim VM when combinations of different DoS attacks were generated. The first combination is generated from six terminals, each generating an attack, and each attack generated twice, while the second combination is generated from a single terminal, one for each attack. In the sub-scenario 2.1, Suricata correctly generated alerts for all three attacks, since the traffic generated in this case was above the predefined threshold of 6000 pps for each attack, and the CPU consumption was considered abnormal. However, in the sub-scenario 2.2, Suricata did not generate any alert for any attack, since the traffic generated in this case was below the predefined threshold of 6000 pps per attack, even though the CPU consumption was considered abnormal and indicated that the victim VM was under a DoS attack. This indicates that Suricata can be evaded and is not always able to detect DoS attacks, specifically the combinations of distinct and low-traffic flood-based DoS attacks that collectively generate high traffic.

## 5.2 Suricata Results

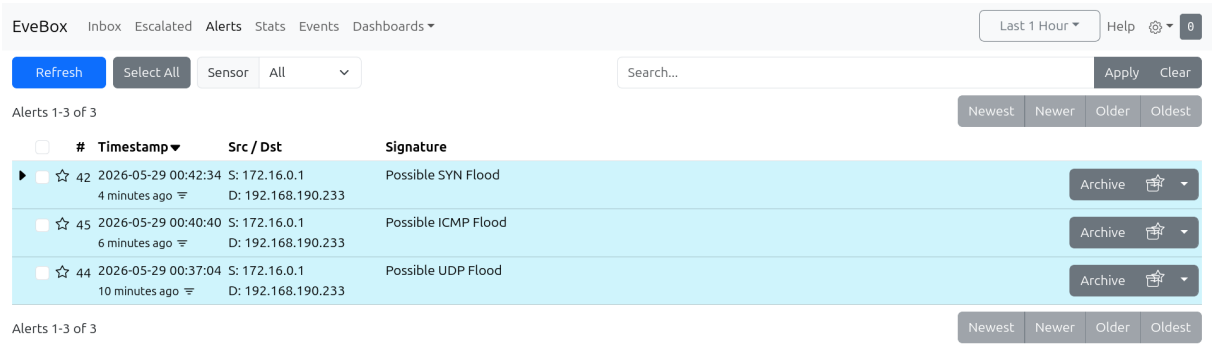


Figure 9: EVE box alerts for scenarios 1.2 and 2.1

As shown in Figure 9, Suricata generated alerts for scenario 1.2, and scenario 2.1. It did not generate any alert for the idle scenario, scenario 1.1 or scenario 2.2.

# Chapter 6

# Recommendations

## 6.1 Security Information and Event Management

To address the evasion of IDS rules when individual attacks remain below predefined thresholds but collectively have a significant impact on target resources, the use of a Security Information and Event Management (SIEM) as a centralized security solution [19] is effective thanks to the possibility of correlating alerts and EVE structured logs generated by Suricata over time [20] and analyzing their combined impact.

## 6.2 Anomaly-Based Intrusion Detection Systems

In addition to using a SIEM, the use of anomaly-based IDS instead of signature-based IDS like Suricata [2] is effective for these types of attacks. Even though each individual attack seems legitimate, their overall combination exhausts the CPU. Anomaly-based IDS use Machine Learning to aggregate and analyze the behavior between multiple low-rate events happening instead of relying only on predefined signatures or thresholds set [21].

# Chapter 7

# Conclusion

In conclusion, using a VMware Workstation testbed consisting of Kali Linux as the attacker VM and Ubuntu Desktop as the victim VM, three types of DoS attacks were generated, which were UDP floods, ICMP floods, and SYN floods, both individually and in combination with each other, to evaluate the performance of Suricata in detecting such attacks.

A noticeable increase in CPU consumption was observed on the victim VM when generating DoS attacks, with a significant increase when generating more attacks simultaneously. Suricata was able to correctly generate alerts for individual DoS attacks when the traffic generated by the attack exceeded the predefined threshold of 6000 pps, while not generating false positives for normal traffic. However, Suricata was not able to generate alerts for combined distinct DoS attacks when the traffic generated by each attack was below the predefined threshold of 6000 pps, even though the overall impact of the combination of attacks was abnormal and exhausted the victim's resources.

This study demonstrates that Suricata, being a signature-based IDS, is effective at detecting DoS attacks when the traffic generated by the attack matches a predefined ruleset. However, it is not effective at detecting correlated attacks related to the same event, highlighting that it does have its limitations, posing a risk to the security posture of victims.

# AI Usage Declaration

This LaTeX report was fully written by the authors.

No AI tool was used for grammar improvements, spelling corrections, language verification, or translation.

Additionally, no AI tool contributed to brainstorming ideas, rephrasing and summarizing texts, or explaining concepts.

Finally, no AI tool contributed to the content of the report, meaning the text and figures were entirely written by the authors.

**Level of AI use:** 0.

# References


[1] Cloudflare. What is a denial-of-service (dos) attack? https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/. [Accessed: 28-05-2026].

[2] Suricata. Suricata features. https://suricata.io/features/. [Accessed: 28-05-2026].

[3] Zeyad Safaa Younus and Mafaz Alanezi. Detect and mitigate cyberattacks using siem. In 2023 16th International Conference on Developments in eSystems Engineering (DeSE), pages 510–515, 2023.

[4] Moez Krichen, Ghadi A., and Nizar Alsharif. Assessing the effectiveness of open-source network intrusion detection systems for small-to-medium-sized enterprises. Journal of Information Security and Cybercrimes Research, 8:147, 12 2025.

[5] Faizal Wahyu Romadhon, Muhammad Azza Ulin Nuha, Yusuf Adiprawira, and Riri Fitri Sari. Comparative analysis of haproxy and nginx load balancers in mitigating user datagram protocol (udp) flood attacks. In 2024 12th International Conference on Information and Communication Technology (ICoICT), pages 354–359, 2024.

[6] Christine Hutabarat and Yudistira Asnar. Development of machine learning module in intrusion detection system for unknown threat detection. In 2024 IEEE International Conference on Data and Software Engineering (ICoDSE), pages 114–119, 2024.

[7] Fortinet. What is ddos attack? https://www.fortinet.com/resources/cyberglossary/ddos-attack. [Accessed: 28-05-2026].

[8] OISF. Suricata packet processing pipeline. https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Packet_Pipeline. [Accessed: 28-05-2026].

[9] Suricata Documentation. Rules format. https://docs.suricata.io/en/latest/rules/intro.html. [Accessed: 28-05-2026].

[10] Kali Linux Documentation. What is kali linux? https://www.kali.org/docs/introduction/what-is-kali-linux/. [Accessed: 28-05-2026].

[11] Ubuntu Desktop Documentation. About ubuntu desktop. https://documentation.ubuntu.com/desktop/en/latest/explanation/about-ubuntu-desktop/. [Accessed: 28-05-2026].

[12] Suricata Documentation. Meta keywords. https://docs.suricata.io/en/latest/rules/meta.html. [Accessed: 28-05-2026].

[13] Suricata Documentation. Thresholding keywords. https://docs.suricata.io/en/latest/rules/thresholding.html. [Accessed: 28-05-2026].

[14] Suricata Documentation. Icmp keywords, itype. https://docs.suricata.io/en/latest/rules/header-keywords.html#itype. [Accessed: 28-05-2026].

[15] IETF. Internet control message protocol (icmp) specification. https://datatracker.ietf.org/doc/html/rfc792, 1981.

[16] Mengqi Zhan, Yang Li, Huiran Yang, Guangxi Yu, Bo Li, and Weiping Wang. Coda: Runtime detection of application-layer cpu-exhaustion dos attacks in containers. IEEE Transactions on Services Computing, 16(3):1686–1697, 2023.

[17] Suricata Documentation. Lua scripting for detection. https://docs.suricata.io/en/latest/rules/lua-detection.html. [Accessed: 28-05-2026].

[18] Suricata Documentation. Xbits keyword. https://docs.suricata.io/en/suricata-8.0.1/rules/xbits.html. [Accessed: 28-05-2026].

[19] IBM. What is security information and event management (siem)? https://www.ibm.com/think/topics/siem. [Accessed: 28-05-2026].

[20] Suricata Documentation. Eve json output. https://docs.suricata.io/en/latest/output/eve/eve-json-output.html. [Accessed: 28-05-2026].

[21] IBM. What is an intrusion detection system (ids)? https://www.ibm.com/think/topics/intrusion-detection-system. [Accessed: 28-05-2026].