Implementation of Open-source IDS (Snort) to Secure docker container

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		
	Page-31		
Word Count:	**Count- 5765**		

Signature: Aniket Singh

Date: 16-08-2021

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):	

Abstract

Technology is moving towards cloud computing. Moreover, to achieve high performance fully functional trustworthy and reliable virtual environment container technology had created a huge dependency. Additionally, for rapid and well-organized large-scale deployment of any application or the services containers are core components nowadays. However, containers-based technology provides such kind of structural process which combines multiple resources for example CPU, memory, storage, disk etc., which than forms a complete independent container along with only required dependencies which makes it a lightweight container which is also considered as lightweight virtual environment also known as Docker containers. Tough, docker is having many of the security concerns and this encouraged me to do my research in such a way that, which can detect malicious activity outside containers to keep the containers reliable and vulnerability free. Hence, introduced signature based IDPS which is after implementation not only successfully able to generate real-time alert but also block unwanted traffic followed by efficient surveillance and protection against unauthorized remote access by abusing the REST-full (Representational State Transfer) API which means compromising the entire system, perhaps this is achieved by writing our own customized rule in **.config** file.

Keywords: Docker containers, IDPS, Virtualization, cloud computing.

Table of Content

1.Introduction

The rapid growth in inventions had evolved the technology. Tough, such development had helped in achieving the plethora's of milestones and cloud in one of such developments, which not only provides the faster processing or computation of the data, but also provides the location-independent computation.(Wan, 2011) Trusting on cloud nowadays is also one of the biggest concerns because there is no complete transparency in the processing of the user's data. However, due to location independent processing it opens many security challenges which need a complete consideration in relation to the cloud services. Moreover, Virtualization which includes "hypervisor" and "Container" based technology. Hypervisor-based technology provides hardware level-based virtualization on the other hand container-based technology involves virtualization which mainly supports OS level virtualization approach. Container based virtualization is most widely used nowadays and also in future because of its light-weight nature and location independent processing.(Singh and Singh, 2016)

The container technology has been adapted by many giant companies for example Azure, IBM, Google and AWS etc., The main advantage of using this technology is easily scalable and lightweight, due to which infrastructure can be established anywhere in the world, also sharing of the infrastructural resources becomes smooth because of the fundamental of sharing kernel of the operating system. However, this fundamental mainly deals with Linux functionality additionally, it uses the **Namespace** and **C-groups** for the isolation as compared to hypervisor in virtual machine perhaps it has many security challenges which requires sincere attention. (Manu et al., 2016)
Contribution in container-based technology consists of Open-VZ, Linux V-Server, LXC (Linux container) but among all of them docker containers are one of the most reliable and practically used container with the proven better result technology.(Bhatia et al., 2018) Like every coin has two phases similarly, docker also has many advantages and disadvantage which involves severity against Denial of services (Dos) which result in stop the services of containers that is breaking down the container services. Moreover, protection against this type of attack is very much necessary, which can be possible via intrusion detection system. An efficient intrusion detection method can not only help in protecting against the Dos attack but also secure it from unwanted breaking down.(Chelladhurai et al., 2016)

To monitor suspicious activity there is a need of some system and that is intrusion detection system which can detect the unwanted dangerous activity so that the running services like any applications, servers, database inside the container can be protected against the malicious impact and tough proper surveillance with IDS can help in overcoming such unauthorized activities. Because IDS can we implemented in such a way that in can detect unwanted traffic and disallow and block the suspicious users to remotely gain the system access. However, implementation of IDS majorly depends on the requirement hence it is categorized based on the different requirements as a Host-based, Network-based signature and anomaly-based detection system. (Lee et al., 2011)

It has been seen that for the security and protection of the cloud environment there are multiple deployments. However, it is also observed that IDS had provided the great contribution in order to protect virtual machine but in case of container technology it is far behind. Hence it is required to think and consider this deployment for the protection of container technology. This is the main

reason behind the selection of this research so that efficient demonstration related to the secure deployment of the docker container with the help of implemented rule based IDPS.(Flora and Antunes, 2019)
In this research I had demonstrated the numbers of attacks which includes Denial of service (Dos) and gaining unwanted remote access via reverse shell using the attacker VM so that such vulnerability can be demonstrated in an efficient way. Moreover, these attacks for example denial of service will impact the services of the containers it will utilize the complete resources so that containers will not work properly and all the services related to it will stop. Additionally, to get the unauthorized access, hacker abuses REST API, the main motive of this attack is to get the remote access that means attacker will dominate and can make unwanted changes which result in complete comptonization of the system. Tough, implementation of rule-based ids will mitigate against such malicious activity and Docker container can work properly and all the services will work efficiently. However, this research focus on the security of the docker containers. The main aim of the implemented IDPS is to stop unwanted traffic and also in case when any unwanted user tries to access, or any request occurs from illegitimate user than blocks it in real time. Additionally, it will protect the containers from such attacks and also guard the host machine on top of which our containers are running.

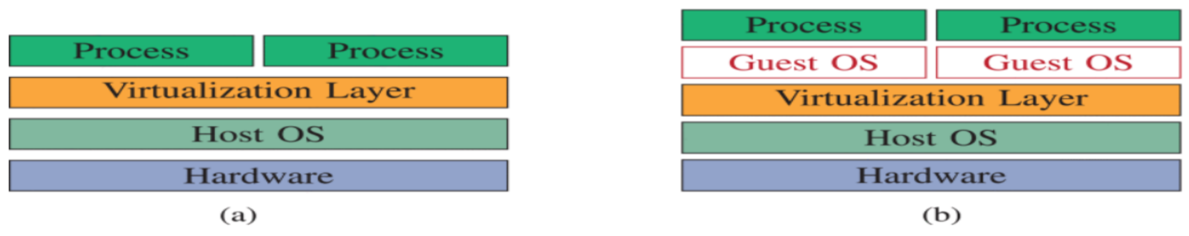
2.Research question

How to secure deployment of docker container from various malicious attacks and threats? Knowing that there is no extra layer of isolation between host machine and container.

3.Literature Review

In this section I am going to describe the past studies and work along with the deep inspection and overview of current hurdles faced and why it is required to protect the docker containers and its deployment from malicious activities along with understanding of associated risks.

3.1) Challenges faced: In a multi-tenancy cloud environment, there are various critical running services which required full proof security otherwise there is a huge chance of social and economic losses via cyber-attack. Moreover, running virtual environment which includes virtual machines consists of hypervisor which establishes an extra layer of protection and this protection acts as a partition in the middle of host machine and the application running inside the VM this feature of virtual machine makes it more secure than docker container technology. In Container technology communication between host machine and application happens with the help of kernel of the host machine but in Virtual machine there is no direct communication between host kernel and the running application in VM. Also, multiple containers can share the same kernel. Tough, it is easy for attacker to target containers and if the hacker is able to get the access of one of the containers, then probably, he will make all the containers malicious or harm the running services also there is chance the entire system get compromised.(Sultan et al., 2019)



The above figure illustrates the difference between the VM and containers. Moreover, it is clearly visible that containers are lightweight hence are used extensively. But in multitenancy cloud environment most of the containers utilizes the same host kernel due to this there is a chance of huge information discloser which may result in cyber-attack. Perhaps, to secure docker container it is mainly relay on few elements as follows, **Proper isolation**, **hardening of the host** and **securing the network**.

3.2) Proper Isolation: Major components of the docker containers are depend on Linux kernel functionality. Linux by default with namespace feature provides the partitioning but in case of c-groups it is different. Moreover, Cgroups are activated based on the requirements of docker container deployment. Containers in the multi-tenant cloud environment share the same host kernel which means utilizing the same network bridge which introduces vulnerability in the isolation process, resulting in ARP poisoning.(Combe et al., 2016)
The security of the docker containers can be narrow down by enabling the few options during the deployment of the container and these options are "-cap-add=<CAP>", "-uts = host", "-net=host", "-ipc =host" etc., However these options are mainly used in establishing the communication between container and host machine which then opens up a considerable vulnerability. For example, if the two containers have different name space that means the running process for both the containers are different from each other. But if option (-net = host) is used at the time of container deployment then both the container will share the same network resources tough increasing the possibility of providing complete access of the host stack resulting in sniffing of the network and privilege escalation. Also, there are options if used during the configuration of container will result in demolishing the Transport layer security (TLS).(Talbot et al., 2020)

3.3) Hardening of the host: Linux uses certain modules so that it can provide some sort of security during the deployment of the container. It basically generates invisible kind of restriction. Perhaps, App-Armor, SE-Linux, Sec-Comp do not come with such restriction. Moreover, in case of APP-Armor all the privileges are open (full-permission) to access any filesystem, network etc., Additionally, using SE-Linux docker container shares the resources in the same domain. However, it has been observed that default behavior while doing the host hardening do not provide the protection to container from other containers. (Combe et al., 2016), (Talbot et al., 2020)

3.4) Securing the network: For remote management and distribution of the image docker daemon utilizes the network resources. Also, to administratively control the docker daemon UNIX sockets are used which are manage by root: docker, placed in /var/run/docker.sock. That means basically if any user has access of such socket, then he can access host and also run any of the containers which are running on top of host in privilege mode. This is one of the drawbacks because there is a chance of gaining the root access with the help of UNIX socket and once the access granted then it can be used in case of TCP socket as well. Hence, to mitigate those it is necessary to take safety measures while downloading the any files or images keep in mind to use trusted network or resources. Moreover, it is very crucial to deeply inspect, verify and download the images in relation to docker containers because once it is downloaded then those files will connect to any registry over TLS.(Wenhao and Zheng, 2020)

3.5) Review of current work: This section will demonstrate different snort-based IDS and related research with various previous applied methods and constraints with its improved methodology including different platforms for example Network or Cloud.

3.5.1) Previous applied Method-1: The author had demonstrated a secure surveillance approach called OUT-VM also known as MNPD (malicious network packet detection) which basically designed to monitor cloud environment which includes VMs at virtualization plus network layers. It is necessary to provide the protection to running servers in cloud network hence to do that behavioral analysis of the incoming and outgoing packets is much needed. So that complete cloud network will get better defense against intrusions. However, traffic monitoring at hypervisor level provides the safety against VM-to-VM attacks. The demonstrated approach worked but the level of the security was not much efficient along with that cost of implementation was also high and system was generating false positive alarm which was consuming CPU(Mishra et al., 2017) . To overcome that author had come up with new approach called HIDCC moreover this approach mainly focused on reducing the false positive alarms but the deployment cost was higher.(Hatef et al., 2018)

3.5.2) Previous applied Method-2: The author had proposed HIDS which includes anomaly detection with misused-based detection. Adding to it this approach was the combination of packet header and network traffic anomaly detection (PHAD+NATAD). Perhaps, this approach utilizes the network traffic but the processing of the and analyzing packets was taking too much time which were making the detection mechanism slower. However, to reduce that author proposed to use classifiers which introduces the budget issues(Aydın et al., 2009).

3.5.3) Previous applied Method -3: The proposed intrusion detection for private cloud uses the snort rules along with multi sensors for the detection of the behavior of the network traffic pattern. Moreover, in this approach scanning of the port along with monitoring the behavior of the OS. However, during the testing of the project the various sensors provides the different set of result for the same set of data. The used dataset was MIT-DARPA 1999. Tough managing the sensors for example modifying the rules for each sensor was bit difficult along with that analysis of each sensors detection pattern was also bit tedious which consumes lots of the time in processing resulting in slower detection.(Sengaphay et al., 2016)

3.5.4) Previous applied Method-4: Basically, the beginning of any attack starts with port scan hence it is required to consider a hypothesis which protects from harmful unseen or secret port scan attack. To solve this problem author has proposed an algorithm which recognizes the host with active malicious content. The proposed algorithm known as Threshold random walk (TRW).(Jaeyeon Jung et al., 2004) However, this approach has some boundaries for example, the identification algorithm behaves slower because of the generation of false positive alarm which resulted in utilizing higher CPU resources. In addition, that to overcome such challenges author comes up with new algorithm called as (EPSDR) efficient port scan detection rule which consists of two methods first is packet capturing and another is preprocessing. Moreover, this approach enhances the performance and analysis along with the detection mechanism. EPSDR approach generated 10% higher improvement then TRW but the major drawback of this it supports only TCP traffic do not have UDP packet capturing capabilities.(Patel and Sonker, 2016)

As observed from proposed method it is clearly identified that there is a need of further research and improvement required to make the network more secure. Knowing that the attack parameters are increasing with the increase in the technology.

3.5.5) Previous applied Method -5: Honeypot based IDS had been proposed by an author in order to get protection against the suspicious activity. In this approach honeypot is basically used to capture the data and the captured data seems to be a malicious with the harmful payloads. Once the data collection done then it is shared with IDS system for identification and protection against suspicious activity by understanding the pattern and behavior or the payload it creates the rules.(Sagala, 2015) But it is observed that as soon as the interaction between honeypot and IDS increases it starts consuming lots of the bandwidth and start utilizing the system resources. Tough, to solve this issue and for further improvement author had suggested an algorithm called Low-interaction honeypot and for the configuration backtrack is used for the smooth communication between IDS and Honeypot. During the demonstration the identification of attack parameter worked well but it faced a major problem in dealing with high volume of the traffic which is an indication for any attacker to plan DOS attack and make the all services unavailable. Not only DOS attack but also the system was responding to attacker's request which opens up vulnerability in the system. Hence, to mitigate such challenges author had proposed improvised method called High-interaction honeypot but this approach was also not able to handle high volume of the traffic.(Xiaoyong and Dongxi, 2005)

3.5.6) Previous applied Method -6: To encounter the previously faced issues and the challenges author suggested and presented the new method to deal with the huge traffic so that no packets were dropped and proper inspection takes place. The author had used the detection method called FPGA for the identification and detection of perilous content and suspicious activity via DPI (deep packet inspection).(Thinh et al., 2012) This DPI includes both static and dynamic analysis. However, this algorithm also faced many challenges and limitations. for example,

- It inspects the specific set of packets and detects the header or payloads inside it.
- The capacity of analyzing the packets were limited up to 32 entries
- Few time it detected a smaller number of entries even if the numbers of matched entries were high.
- Many of the hardware platform dealing with power consumption issue due to hardware compatibility issue.

Additionally, author had introduced new approach to overcome on the above challenges and the proposed algorithm which was known as "NETFPGA-based bloom filter". This approach dealt with the packet drop issues but it was examined that not able to handle DOS attack which reduces the accuracy of the suggested method.(Al-Dalky et al., 2014)

3.5.7) Previous applied Method -7: In order to secure the cloud environment and minimizing the previous faced challenges author introduces Virtual host-based IDS which mainly work on the three parameters

- Event auditor
- Cloud intrusion data sets
- IDS services

However, Ids includes analyzer inside its architecture so that it can surveillance and inspect the packets payloads and entire analysis uses CIDD (cloud intrusion detection datasets), Additionally, this research demonstrated 80% of the attacks in cloud environment and the number of generated false alarm was not greater in numbers. Along with it uses the DARPAA datasets to increase the efficiency level of the detection technique. But the main limitation of the of this approach was latency and the main reason behind this latency was the generated background traffic from DARPAA dataset and if this traffic is more than 2mbps which resulted in instability.(Slominski et al., 2015)

The optimized the above-mentioned challenges back propagation neural network (BPN) algorithm was suggested by the author.(Chiba, 2016) Using this approach many of the attacks were identified for example, DOS, ARP spoofing, DNS poisoning, scanning of the port, gaining complete host control, breaking out of the services in cloud environment. Perhaps, this approach also had many of the limitations.(Zhang et al., 2019)

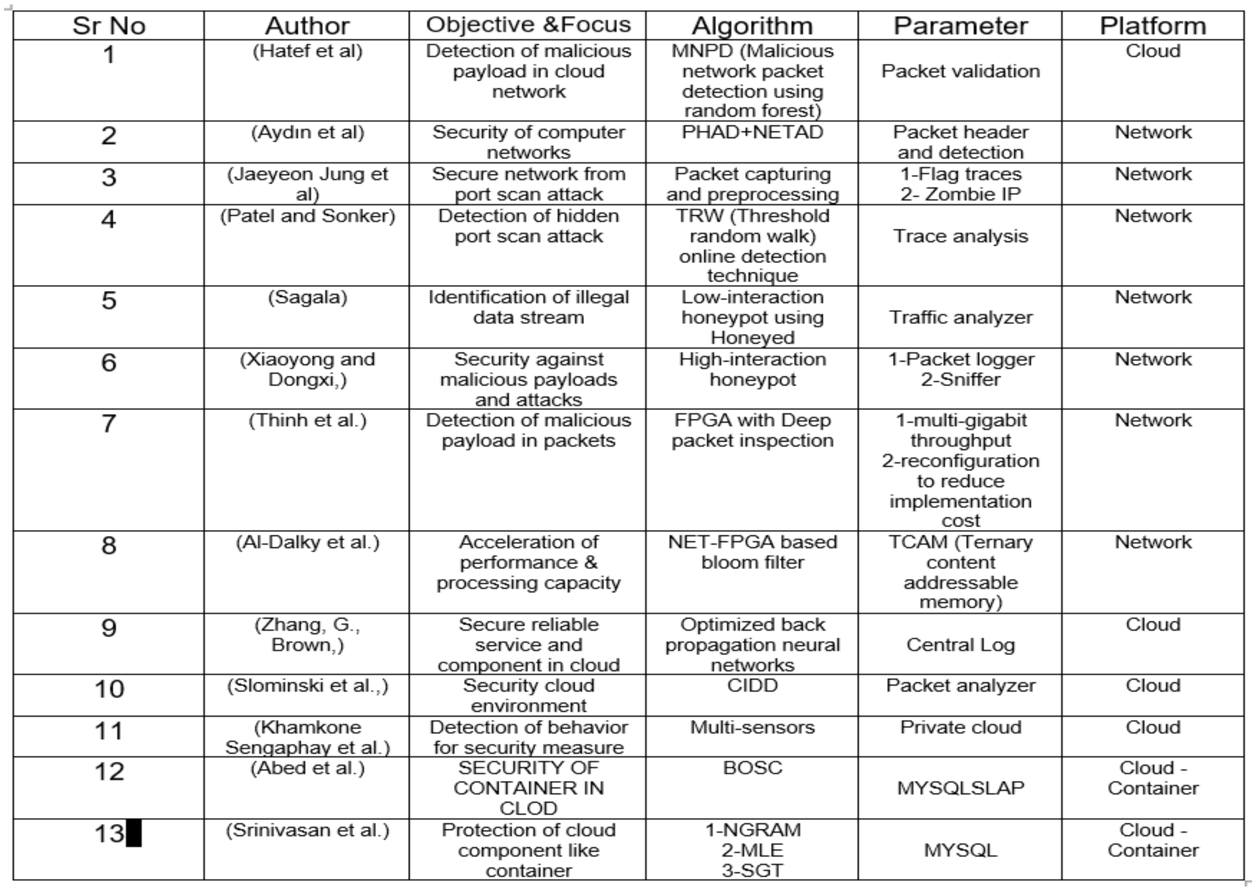
- The level of accuracy was lower
- Slower convergence
- Slower inspection/identification

3.5.8) Previous applied Method -8: Unsecured and open port which are not in use are ignored many times without any security. However, this leads the attacker to take a chance in order to exploit the system tough it is very necessary to protect against such perilous and secret port scan attacks. Hence author had proposed an algorithm to identify such scanning called as Threshold random walk (TRW).(Jaeyeon Jung et al., 2004) Moreover, this approach worked well but as per growing technology this approach looks little outdated. Additionally, to overcome this author had proposed a new approach called EPSDR (efficient port scan detection rule) which includes the collection of packets and processed it. But the main challenge with this approach was not able to detect UDP scan it only capable of detecting TCP scan.(Patel and Sonker, 2016)

Moreover, it is required to research more related to such challenges which includes TCP and UDP both protocol.

3.5.9) Previous applied Method -9: Knowing that traditional networks are now migrating to cloud network the demand of containerization and virtualization is increasing day by day considering this author had proposed Intrusion detection system for container technology and introduced algorithm for the detection and monitoring was BOSC (Bag of system call). However, these algorithms protect against the following attacks (privilege escalation, gaining remote access, DOS, etc.,). Perhaps, author has mentioned that the proposed IDS will generate 100% TPR (true positive rate) and probably possibility of 2% FPR (False positive rate). Moreover, at the time of demonstration it was observed that the designed system was not able to established the interaction between MY-SQL-SLAP and DBMS (data base management system). Also, sometime MY-SQL-SLAP was becoming unstable resulting in performance degradation.(Abed et al., 2019)

Followed by it, Author came with another improved plan which will identify malicious payload in real time placed inside containers and the used algorithm for such detection is called NGRAM probability. Moreover, to detect hidden suspicious activity the used mechanism was SGT (Simple good Turing) along with estimator which was also known as "maximum likelihood technology" This approach was having little similarity as compared with the above approach the main difference it consists of web application running inside containers. However, the main motive of the author behind this design is to monitor the contents inside containers so that another container can't be harmed via it. The investigation showed that false positive rate 0-14% along with recall value in between 78% to 100% also the level of accuracy found in between 85 to 97 percent. Moreover, this methodology of detection also faced the same problem as mentioned above. The used SQLMAP was not generating the anticipated output and the result.(Srinivasan et al., 2019).



Sr No	Author	Objective &Focus	Algorithm	Parameter	Platform
1	(Hatef et al)	Detection of malicious payload in cloud network	MNPD (Malicious network packet detection using random forest)	Packet validation	Cloud
2	(Aydın et al)	Security of computer networks	PHAD+NETAD	Packet header and detection	Network
3	(Jaeyeon Jung et al)	Secure network from port scan attack	Packet capturing and preprocessing	1-Flag traces 2- Zombie IP	Network
4	(Patel and Sonker)	Detection of hidden port scan attack	TRW (Threshold random walk) online detection technique	Trace analysis	Network
5	(Sagala)	Identification of illegal data stream	Low-interaction honeypot using Honeyed	Traffic analyzer	Network
6	(Xiaoyong and Dongxi,)	Security against malicious payloads and attacks	High-interaction honeypot	1-Packet logger 2-Sniffer	Network
7	(Thinh et al.)	Detection of malicious payload in packets	FPGA with Deep packet inspection	1-multi-gigabit throughput 2-reconfiguration to reduce implementation cost	Network
8	(Al-Dalky et al.)	Acceleration of performance & processing capacity	NET-FPGA based bloom filter	TCAM (Ternary content addressable memory)	Network
9	(Zhang, G., Brown,)	Secure reliable service and component in cloud	Optimized back propagation neural networks	Central Log	Cloud
10	(Slominski et al.,)	Security cloud environment	CIDD	Packet analyzer	Cloud
11	(Khamkone Sengaphay et al.)	Detection of behavior for security measure	Multi-sensors	Private cloud	Cloud
12	(Abed et al.)	SECURITY OF CONTAINER IN CLOD	BOSC	MYSQLSLAP	Cloud - Container
13	(Srinivasan et al.)	Protection of cloud component like container	1-NGRAM 2-MLE 3-SGT	MYSQL	Cloud - Container

Fig. Summary Table of Literature Review

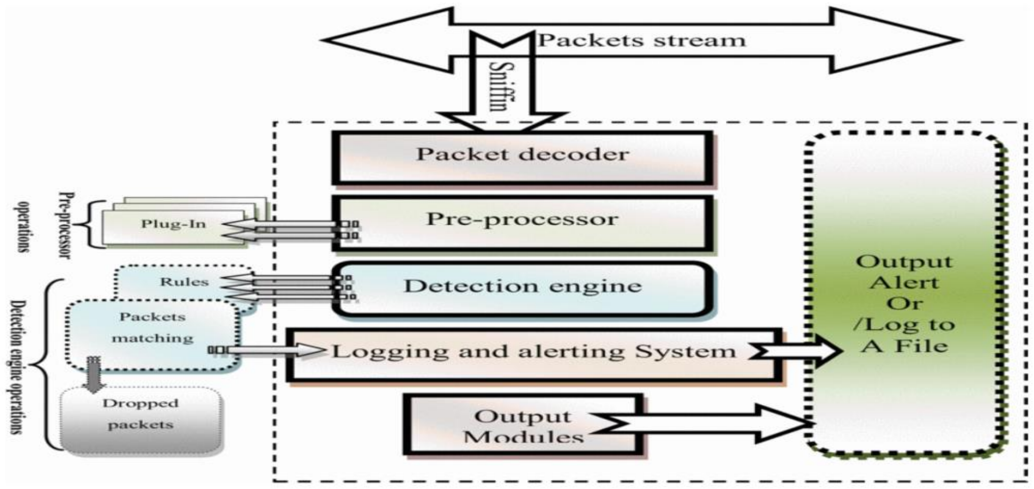
4.Research Methodology & Specification

4.1) Architecture Design: To implement rule-based IDS, the open source and lightweight platform is used called snort. The main benefit of using such platform is, creation of rules because we can customize the rule based on our understanding of network and its security. However, these rules are only for the known attacks.

The rule creation processes support multiple programming language and easy use of text editor. We can create rules and group them in different set of files. Additionally, the main configuration files which includes all the rules and configurations in it is called “Snort.config”. This file is the backbone of the IDS because in consists of all the all the written rules and information about internal and external network connectivity. However, based on it the process of initialization of snort takes place which result in construction of internal data -structure to capture the data.

Now we will be discussed about the architecture of the snort-based intrusion detection system along with its all-core elements. like 1) Packet decoder 2) Preprocessor 3) Detection engine 4) Logging and alerting system 5) Output modules.

4.1.1) Packet Decoder: It is used to collect the traffic from the network interfaces and based on the collection of traffic packets, decoding and pre-processing on that captured packet takes place which is called packet-based analysis. Simply we can say that it is a segregation of numbers of packets which are captured from different set of interfaces. Finally, the analyzed packets were converted into TCPDUMP file. The example of interfaces from where the packet is captured is PPP, Fast-ethernet etc.,



4.1.2) Pre-processor: Pre-processor is one of the core components which is used to process the captured packets like doing some sort of modification for the management and smooth processing before that packet moves towards the next phase called detection engine. Additionally, it carry-out the identification procedure to discover the asymmetry, malformation and deformity in the header of the collected packets so that real time triggered alerts and alarm can be generated. Our implemented IDS is going to collect the data packets analyze it and this analysis is based on the written rules in the detection engine.

There are many techniques with the help of which attacker can manipulate the identification process of IDS. For example, the created rule is going to match the defined signature ("research/ricacadmic") inside the HTTP packet information. However, there is the possibility attacker can manipulate it by small modification in the string which are as fallows.

- "research/./ricacadmic"
- "research/examples/../ricacadmic"
- "research\ricacadmic"
- "research/.\ricacadmic"

The above written strings do not match with the written rules. Hence, using these, attacker can able to bypass the IDS. It is also visible that various methods used by hacker for example it uses web information resource identifier, characters in hexadecimal, Unicode characters which creates illusion that the URL and input data is legitimate. Tough, preprocessor have such intelligence and using those it can able to identify incoming inputs even if they are modified.

The pre-processor has a capability to perform the defragmentation of the packets. Moreover, by default the packets size on internet or on any ethernet interface is approximately 1500 MTU. Which means any packet size more than 1500 MTU is going to be defragmented. Many times, attacker used these capabilities and send the malicious fragmented payloads. Tough, ids won't be able to process over it because it does not match with the written rules. Perhaps, pre-processor had such capability using that identification of such perilous payloads is possible.

4.1.3) Detection engine: The detection of the packets mainly depend on the written rules and the rules are applied once it is a part of detection engine. However, the rules which are part of detection engine also a part of internal data structure and based on the written rules the detection process work. Moreover, the permission related to incoming about outgoing traffic is defined in rules and upon identification of it generates the alerts.

Responding to various traffic packets which is going to take different set of time interval and it depend on few elements:

- Quantity of the written rules
- Load on the network
- Depend on the speed and capability of internal used busses
- Depend on the strength of the machine on which intrusion detection is running.

Below is the Considered factors and protocols along with its characteristics while creation of rule:

- Consideration of every IP header inside the actual traffic
- The transport parameter consists of header for example., TCP, UDP, ICMP etc.
- Taking care of application-level headers for example, FTP, TFTP, DNS, SMTP, SNMP.
- Awareness related to packet payload which includes strings in the packets.

4.1.4) Logging and alerting a system: After detection of perilous payload in the packets by detection engine, the logs related to it is collected in **/var/log/snort** file. Additionally, alerts along with the logs including the time of event are stored in the file called TCPDUMP. We can also change the location of log files, based on any specific requirement or sometimes to make the troubleshooting easy without damaging the actual log files.

4.1.5) Output Modules: The complete control and management of the generated output by logging and producing system alerts. The performance of the output modules is completely based upon the various things as follows:

- Access to /var/log/snort/alerts file
- Suitable of forwarding SNMP traps
- Suitable of forwarding message to syslog servers if configured
- Suitable of accessing the database like MySQL.
- Suitable for creating XML output
- Suitable for editing the configuration on firewall & docker if used.
- Suitable of sending SMB information to Microsoft based machine.

4.2) Flow Diagram: This section we are describing the flow of the implemented IDS along with responsibility of each element of the snort-based IDS plus the flow of the traffic from each element and decision making.

Step-1) The decoding of the packets done by Packet decoder once packets are received.

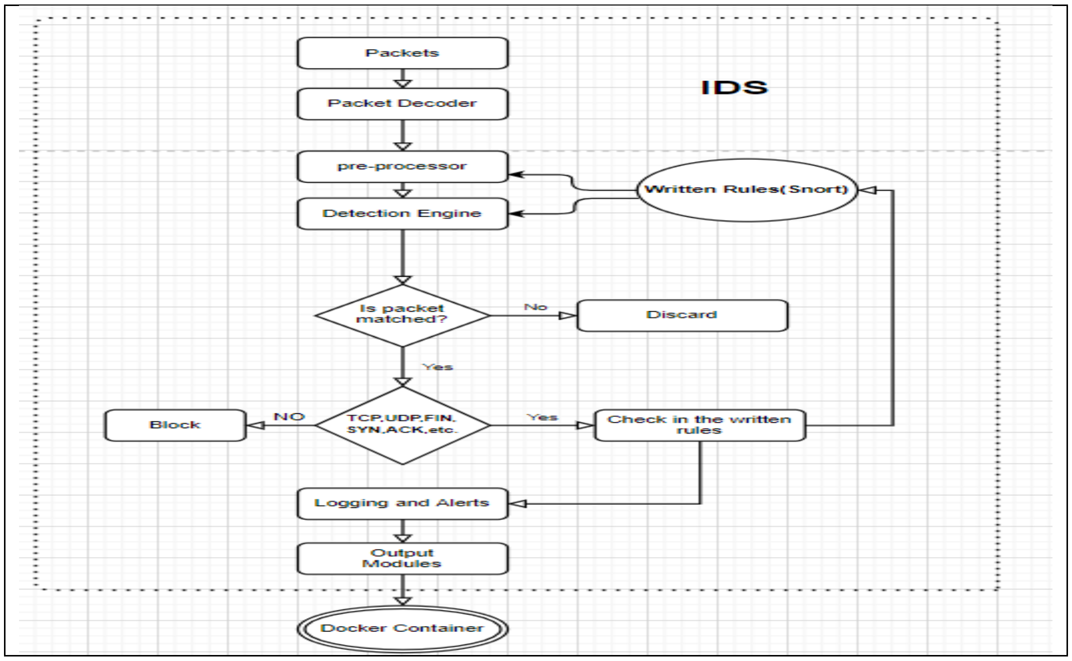


Fig. Flow diagram of Rule based ids to protect Docker Container

Step-2) Once decoding process completed then the packet is forwarded to pre-processor to check and detect the irregularity inside packet payload.

Step-3) Once the anomaly content detected in packet then it is passed to detection engine which reads defined custom rule in snort.config file also known as internal data structure of snort.

Step-4) After that packet are matched as per the written rules and decision tree works accordingly.

Step-5) Packet forwarding and discarding is entirely depending on created rule. That is matched is equal to pass otherwise deny or block

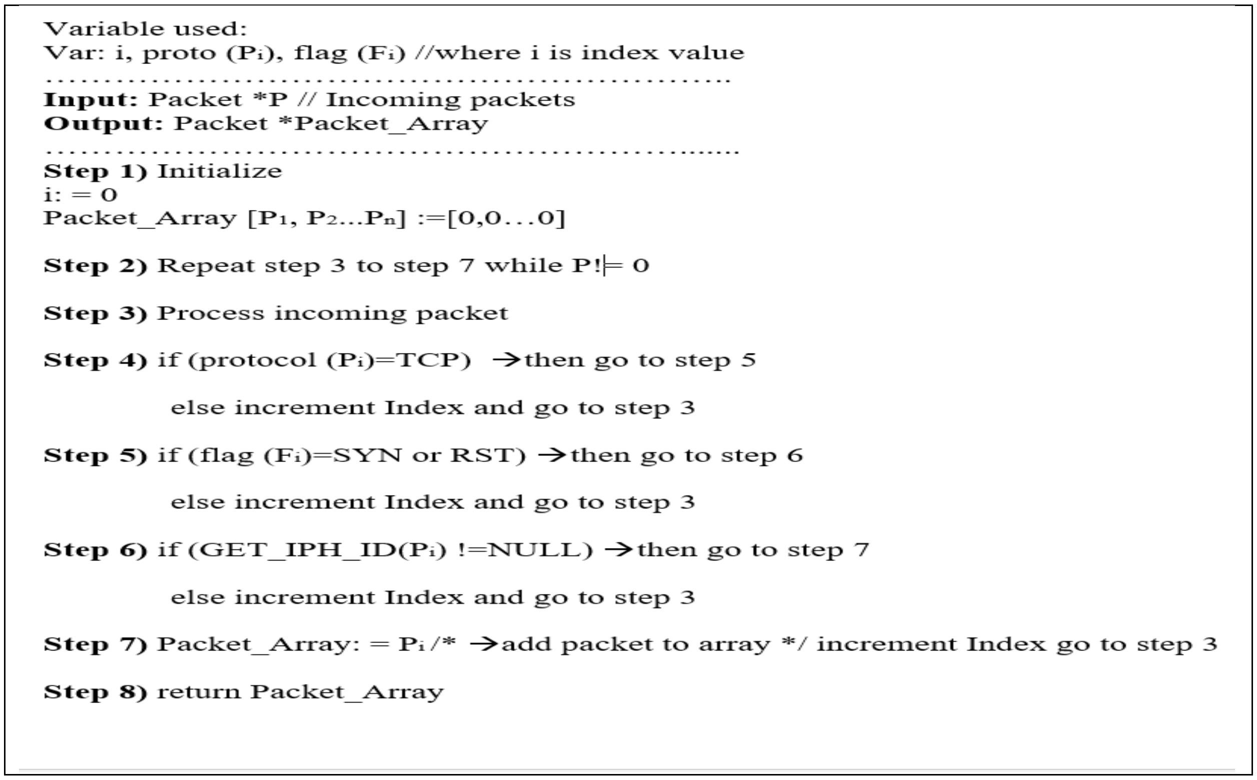
Step-6)If the packets is found to be a legitimate packet with no malicious payload, then its fine otherwise, it is again forwarded to snort.config file for rechecking the content and if any suspicious payload found then the packet is going to block or deny.

Step-7) After that the packets with the perilous payload is tracked by Logging and alerting system.

Step-8) After tracking and analyzing, the final legitimate packet without any anomaly is forwarded towards the docker container.

Step-9) Hence it is proven that when packets reach to the docker container it does not have any malicious content and the container is protected in comparison with other previous approach.

4.3) Pseudo code process for rule-based IDS: In this section we are providing the logic on top of which the detection and identification process of the packets as per the defined rules are going to work.



```
Variable used:
Var: i, proto ($P_i$), flag ($F_i$) //where i is index value
.......................................................
Input: Packet *P // Incoming packets
Output: Packet *Packet_Array
.......................................................
Step 1) Initialize
i: = 0
Packet_Array [$P_1$, $P_2$...$P_n$] :=[0,0...0]

Step 2) Repeat step 3 to step 7 while P!|= 0

Step 3) Process incoming packet

Step 4) if (protocol ($P_i$)=TCP)  →then go to step 5

            else increment Index and go to step 3

Step 5) if (flag ($F_i$)=SYN or RST) →then go to step 6

            else increment Index and go to step 3

Step 6) if (GET_IPH_ID($P_i$) !=NULL) →then go to step 7

            else increment Index and go to step 3

Step 7) Packet_Array: = $P_i$ /* →add packet to array */ increment Index go to step 3

Step 8) return Packet_Array
```

Fig. Pseudo Code

5.Implementation

5.1) Evaluation and planning involved to evaluate the Implementation: The below design illustrates the implemented solution which will protect the deployment of docker container from the malicious activity. Here we are using Kali Linux as an attacking VM with 64-bit Debian derived distributed system along with 50GB of HDD and 4GB RAM.

5.1.1) Selection purpose of Kali Linux: The user interface and its adaptability and suitability to perform penetration testing is the key reason for the selection of Kali Linux.

Moreover, we are using Ubuntu 18.04 as a host machine in which our implemented snort-based ids and containers are running. It is also 64-bit distributed system with 4GB RAM and allocated hard disk drive space of 20GB.

5.1.2) Selection purpose of Ubuntu: The core advantage of using ubuntu is that it provides great environment for Docker to work properly i.e., greater compactivity for docker is provided. Moreover, easy and reliable GUI Graphical user interface.

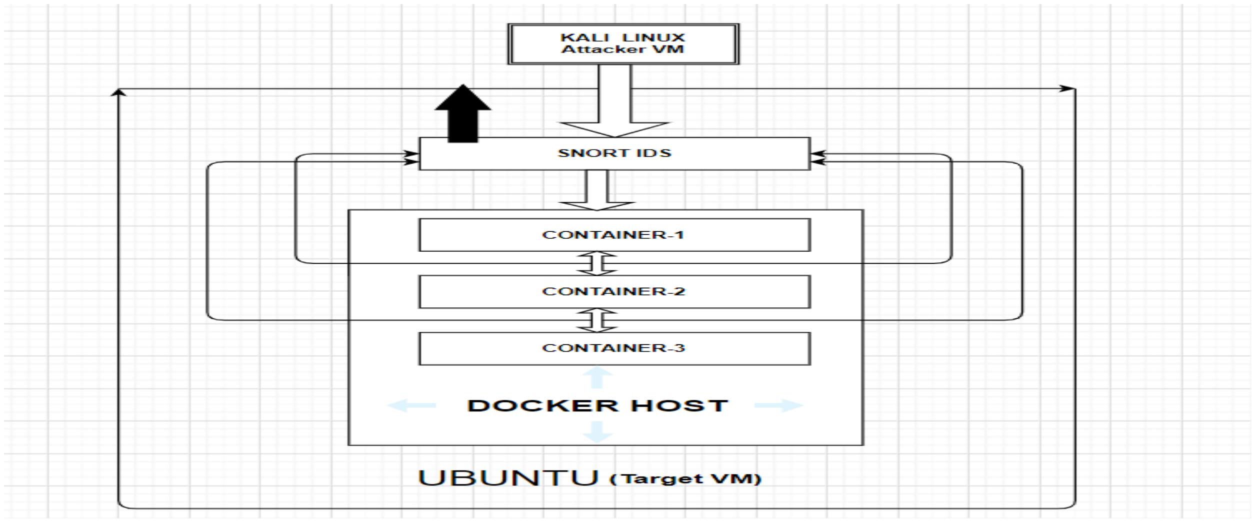


Fig. Implemented solution

6. Functionality that is expected to be completed during the implementation

Case 1: Targeting the docker host using Attacking VM: Using attacking VM we have forwarded the ping request to check the alertness of the implemented IDS. However, IDS is completely able to detect that which means packets are observed and analyzed before it reaches to the container. Doing this we are stating that if any malicious content arrived then Snort will first process it and based on defined rule blocking & accepting processing will take place.

Case 2: Compromising the ubuntu VM: In this case the hacker will attempt to escalate the privileges of ubuntu vm remotely by abusing the REST-API by making use of SSH and Telnet. Moreover, if hacker succeeds then probably, he will be able to access host along with the running containers inside it. Additionally, attacker can forcefully stop and run newly created malicious container inside the host machine also there is a high probability of data theft or discloser of critical confidential information. Tough, to tackle such situation the implemented snort-based ids play an important role and protect against such malicious activity by generating the alerts and blacklisting the specious IPs or the users.

Case 3: One malicious container attacking and affecting other container: In this case the attacker for example able to get the access of container then from there hacker is trying to infect the other containers or run and stop the services running inside the containers. Moreover, in militancy cloud environment multiple containers work on single host kernel. Tough infecting other containers is not that much difficult. Hence, the implemented IDS is monitoring the activities of the running containers and any suspicious activity. However, any packets forwarded to other container will be analyzed by ids and based on the analysis the decision will take place.

Case 4: Host (ubuntu) attacked by infected container: If we assume one of the containers got compromised or infected by malicious user as per the above case then there is a chance, that hacker will try to get access of host machine or escalate the privileges of another container. Moreover, the main motive behind implementation of IDS is to protect the deployment of the docker container. Here we are able to protect the container from another container as well as host machine along with it also protecting from REST-API abuse.

In order to recapitulate we can say that, the implemented snort-based IDS is contributing in the protection of Confidentiality integrity and availability (CIA) of the deployed containers. The implemented

Intrusion detection protection system not only surveillance the traffic but also takes an appropriate action against the harmful payloads and blocks it so that containers and the host machine will work without risk. Moreover, scanning of complete incoming and outgoing traffic to protect against suspicious activity.

6.1) Test case wise summary table: This table illustrates types of testing with used method to get the expected result.

Test cases	Type of testing	Method of testing	Expected results
1	1] Connectivity of the attacker VM and Docker host 2] Alerting and blacklisting ICMP packets	1] Transmitting ICMP packets from Kali VM to Docker host installed in the Ubuntu VM. 2] Writing rules in local.rules files as per connectivity request and running Snort	1] Transmitted packets should receive a response and packets lost should be 0%. 2] Snort should be able to generate
2	Remote access attack launched 1] Gathering information 2]Obtaining the reverse shell 3] Attack the host 4] Remote Login 5]Inter-container connectivity	1] Docker ports scanned using the Nmap Scanner 2] Creating a container remotely and getting a reverse shell by using the Netcat listener. 3] Escaping container and creating a new user on host 4] SSH tunneling for remote access. 5] Transmitting ICMP packets between containers	1] Provide information about open ports, hops, versions, etc. 2] Attacker able to access the Docker system. 3] Provide access to sensitive on the host. 4] Attacker should be able to login to Ubuntu as a legitimate user. 5] Transmitted packets should receive a response and packets lost should be 0%.
3	Alerting and prevented if 1] Reverse Shell requested 2] Inter-container pings 3] Container ping the host 4] Container tries to access the host. 5] If remote login requested (SSH)	Writing the rules in local.rules file and running the Snort on the Ubuntu Host in inline mode before launching the attacks.	We should be able to see that Snort has successfully taken the actions desired by the user. 1] Alerts should be generated 2] Alerts should be generated 3] Alerts should be generated 4] Alerts should be generated 5] The connection should be blocked, and an alert should be generated.
4	To launch a DoS attack (TCP packets) against Docker Host.	Hping3 tool would be used to launch the flood attack from random IP addresses.	The attack should cripple the system by making its resources unavailable to legitimate users.
5	Alerting and blocking the DoS attack TCP packets	Writing rules and re-configuring the Snort to protect the Docker host against flood attack	Requested action should be performed by the Snort. Alerts should be generated, and traffic blacklisted.
6	Launch DoS attack (UDP Packets)	LOIC GUI tool would be used to launch to send UDP packets to the Docker host using port 80	The attack should affect all the containers running in the Docker. It may even shut down the system for a limited time.
7	Alert and block DoS attack (UDP packets)	Writing rules and re-configuring the Snort to protect the Docker host against UDP attack	Snort should be able to take appropriate action by generating alerts and blacklisting the unwanted traffic.

Fig. summary of the test cases

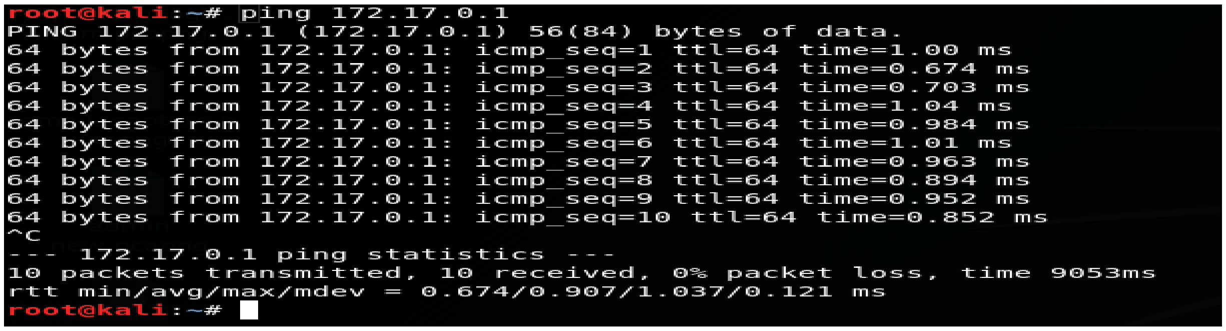
7.Result

7.1) Summary of the result as per the performed test cases.

Type of Test	Detection	Alert
Ping request	YES	YES
Blacklisting of ping request	YES	YES
Reverse shell	YES	YES
Container to container	YES	YES
Host to container access	YES	YES
TCP-based DOS	YES	YES
UDP-based DOS	YES	YES

7.2) Test case wise Result explanation:

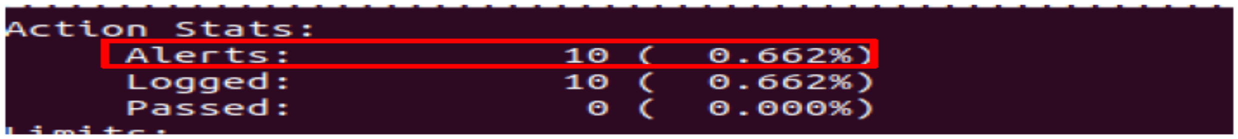
7.2.1) Ping request by attacker to container.



```
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:~#
```

Fig.1

Generated alert by IDS

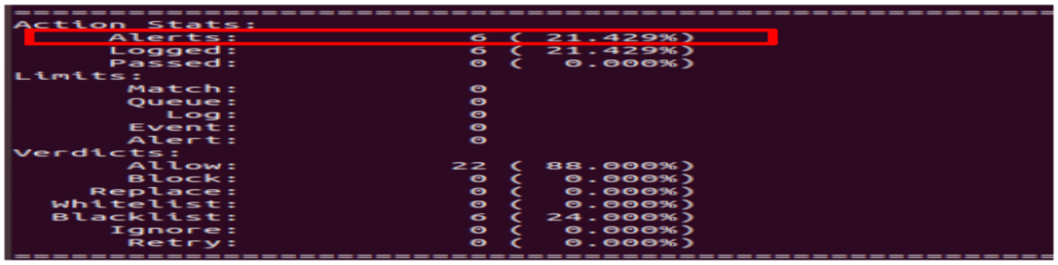


```
Action Stats:
     Alerts:            10 (   0.662%)
     Logged:            10 (   0.662%)
     Passed:             0 (   0.000%)
```

Fig.2

Results: It is clearly visible that the total number of packets send is equal to the number of generated alerts is 10 which indicates that ids is able to detect 100% forwarded packets.

7.2.2) Blacklisting of ping:

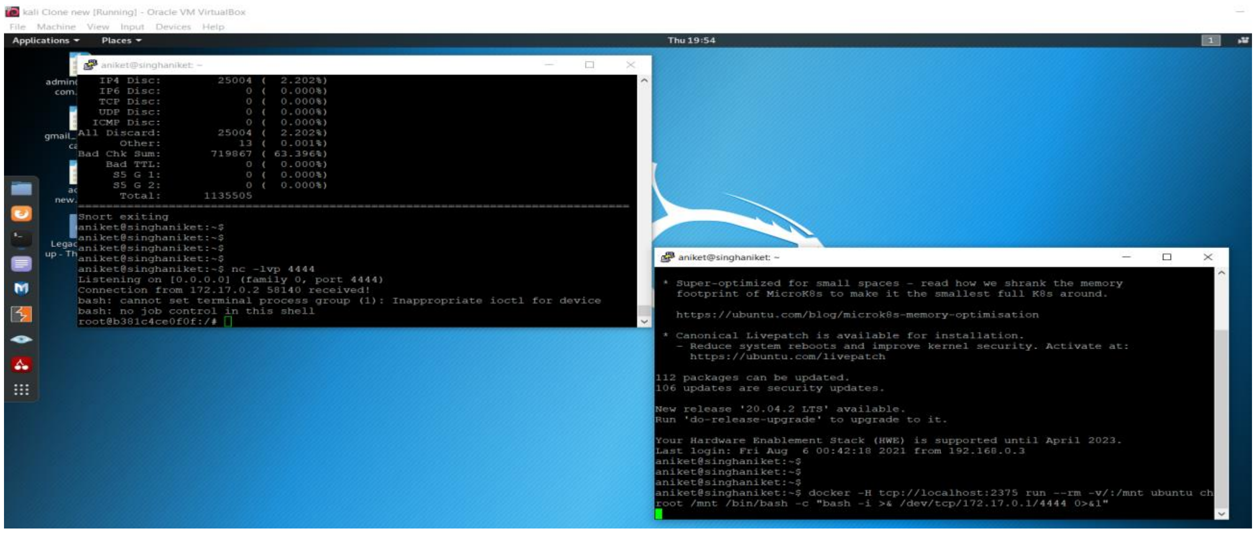


```
===============================================================================
Action Stats:
     Alerts:            6 (  21.429%)
     Logged:            6 (  21.429%)
     Passed:            0 (   0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:            0
      Alert:            0
Verdicts:
      Allow:           22 (  88.000%)
      Block:            0 (   0.000%)
    Replace:            0 (   0.000%)
  Whitelist:            0 (   0.000%)
  Blacklist:            6 (  24.000%)
     Ignore:            0 (   0.000%)
      Retry:            0 (   0.000%)
===============================================================================
```

Fig.3

Results: from the above figure it is clearly visible that Snort IDS is able to black list the ping request. Here attacker is sending 6 packets and all the six packets are blacklisted and blocked by implemented IDS.

7.2.3) Attacker is requested for the reverse shell to get unauthorized remote access.

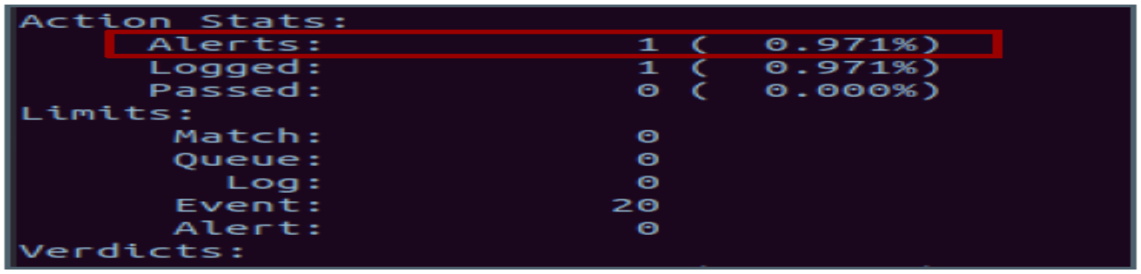

kali Clone new [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications
Places
Thu 19:54
aniket@singhaniket: ~

```
   IP4 Disc:          25004 (  2.202%)
   IP6 Disc:              0 (  0.000%)
   TCP Disc:              0 (  0.000%)
   UDP Disc:              0 (  0.000%)
  ICMP Disc:              0 (  0.000%)
All Discard:          25004 (  2.202%)
      Other:             13 (  0.001%)
Bad Chk Sum:         719867 ( 63.396%)
    Bad TTL:              0 (  0.000%)
     S5 G 1:              0 (  0.000%)
     S5 G 2:              0 (  0.000%)
      Total:        1135505
===============================================================================
Snort exiting
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$ nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 172.17.0.2 58140 received!
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@b381c4ce0f0f:/#
```

```
 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

112 packages can be updated.
106 updates are security updates.

New release '20.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Fri Aug  6 00:42:18 2021 from 192.168.0.3
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$ docker -H tcp://localhost:2375 run --rm -v/:/mnt ubuntu ch
root /mnt /bin/bash -c "bash -i >& /dev/tcp/172.17.0.1/4444 0>&1"
```

Fig.1

Alert generated by snort.

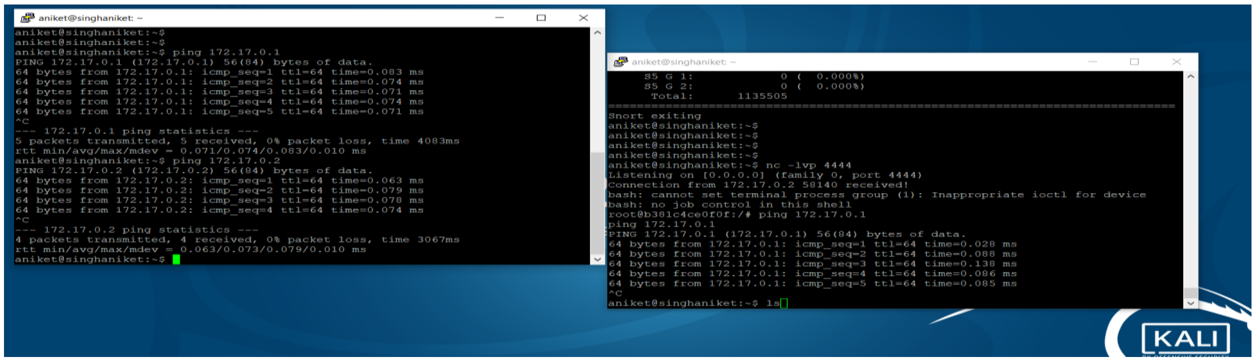


```
Action Stats:
     Alerts:            1 (  0.971%)
     Logged:            1 (  0.971%)
     Passed:            0 (  0.000%)
Limits:
      Match:            0
      Queue:            0
        Log:            0
      Event:           20
      Alert:            0
Verdicts:
```

Fig.2

Result: Here, attacker is able to get the access of container. However, the root access of container is same as the root access of host Machine. But Snort is generating an alert during the reverse shell.

7.2.4) Communication between two containers.

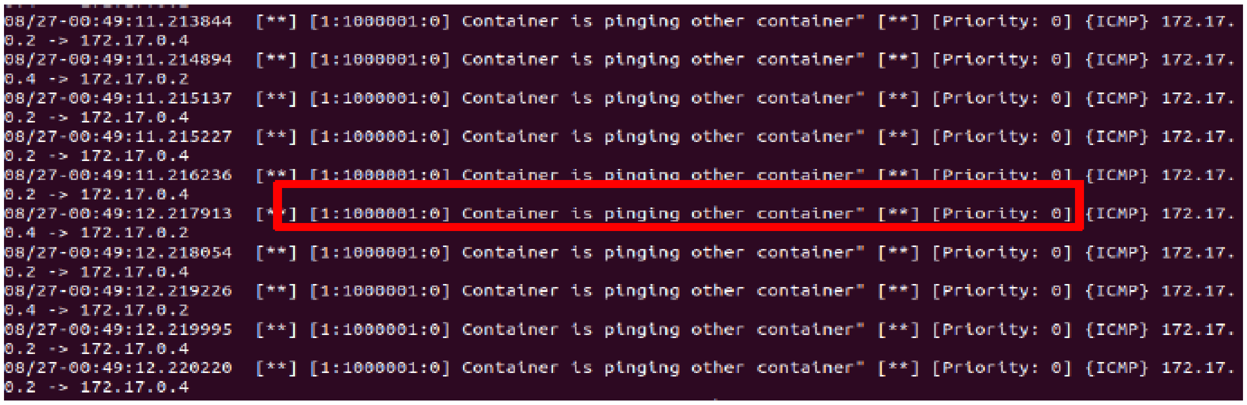


```
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$ ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
64 bytes from 172.17.0.1: icmp_seq=1 ttl=64 time=0.083 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 172.17.0.1: icmp_seq=4 ttl=64 time=0.074 ms
64 bytes from 172.17.0.1: icmp_seq=5 ttl=64 time=0.071 ms
^C
--- 172.17.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4083ms
rtt min/avg/max/mdev = 0.071/0.074/0.083/0.010 ms
aniket@singhaniket:~$ ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.079 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.078 ms
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.074 ms
^C
--- 172.17.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3067ms
rtt min/avg/max/mdev = 0.063/0.073/0.079/0.010 ms
aniket@singhaniket:~$
```

```
     S5 G 1:              0 (  0.000%)
     S5 G 2:              0 (  0.000%)
      Total:        1135505
===============================================================================
Snort exiting
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$ nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 172.17.0.2 58140 received!
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@b381c4ce0f0f:/# ping 172.17.0.1
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=0.028 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=64 time=0.138 ms
64 bytes from 172.17.0.1: icmp_seq=4 ttl=64 time=0.086 ms
64 bytes from 172.17.0.1: icmp_seq=5 ttl=64 time=0.085 ms
^C
aniket@singhaniket:~$ ls
```

Fig.1

Alert generation if one container ping another.

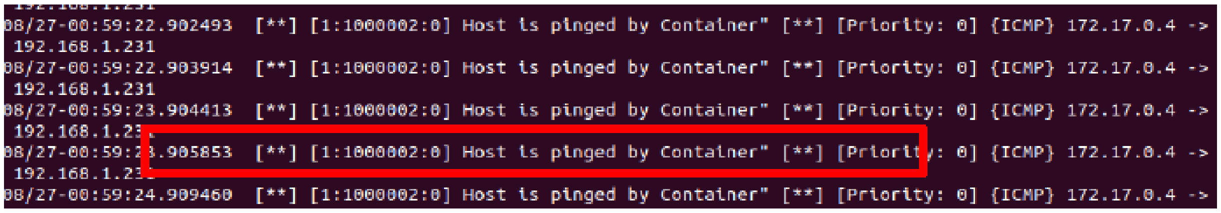


```
08/27-00:49:11.213844  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.2 -> 172.17.0.4
08/27-00:49:11.214894  [**] [1:1000001:0] Container is pinging other container" [**] [Priority: 0] {ICMP} 172.17.
0.4 -> 172.17.0.2
08/27-00:49:11.215137  [**] [1:1000001:0]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

Fig.2

Result: one container can harm another with malicious content hence to avoid that monitoring of such communication done via Snort IDS.

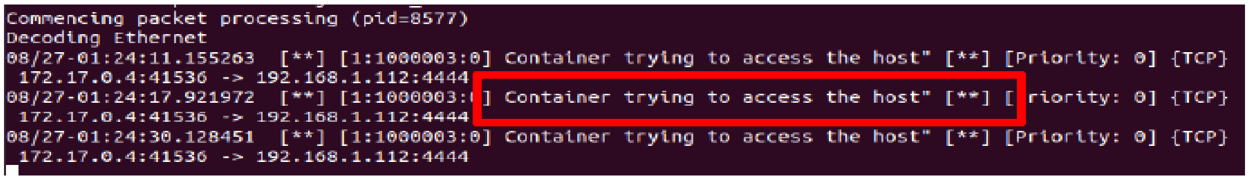
7.2.5) Communication between host and container.



```
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 ->
```

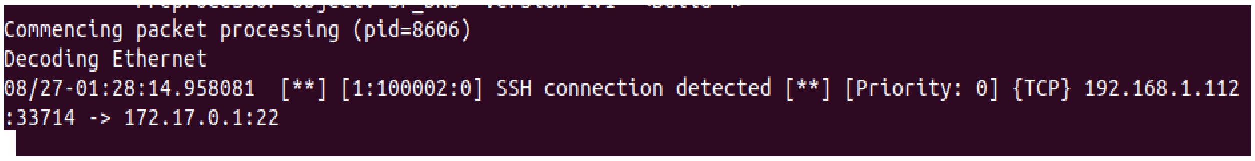
Fig.3

7.2.5.1) Container is trying to access host.



```
Commencing packet processing (pid=8577)
Decoding Ethernet
08/27-01:24:11.155263  [**] [1:1000003:0] Container trying to access the host" [**] [Priority: 0] {TCP}
 172.17.0.4:41536 -> 192.168.1.112:4444
08/27-01:24:17.921972  [**] [1:1000003:0] Container trying to access the host" [**] [Priority: 0] {TCP}
 172.17.0.4:41536 -> 192.168.1.112:4444
08/27-01:24:30.128451  [**] [1:1000003:0] Container trying to access the host" [**] [Priority: 0] {TCP}
 172.17.0.4:41536 -> 192.168.1.112:4444
```

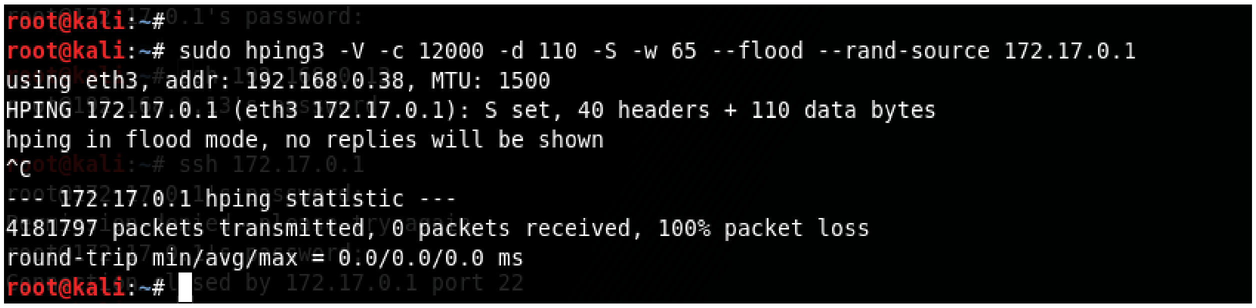
7.2.5.2) Blocking SSH action via Snort IDS



```
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
```

Result: from the above demonstration it is clearly visible that snort is able to protect from unwanted SSH accessing.

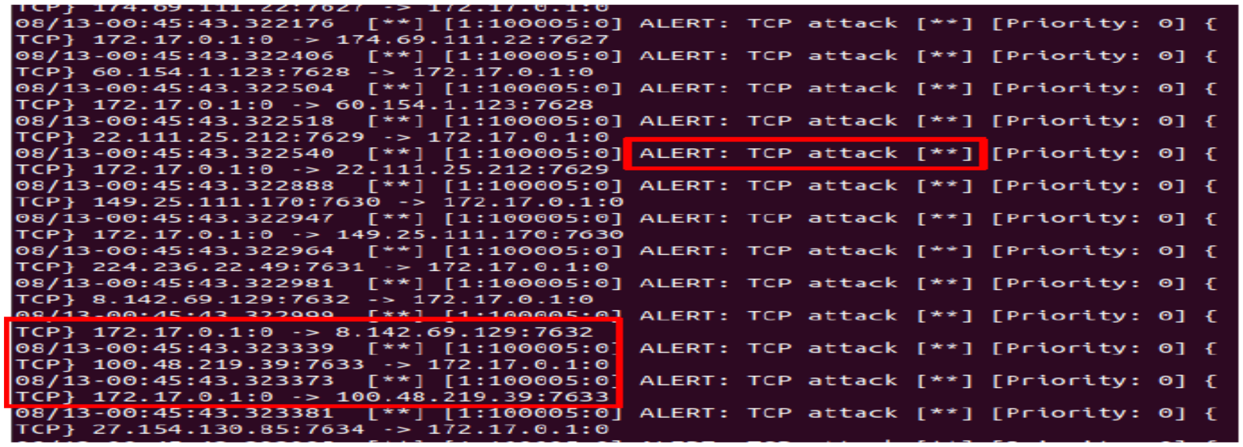
7.2.6) Dos attack and its generated alert (Protocol-TCP)



```
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:~#
```

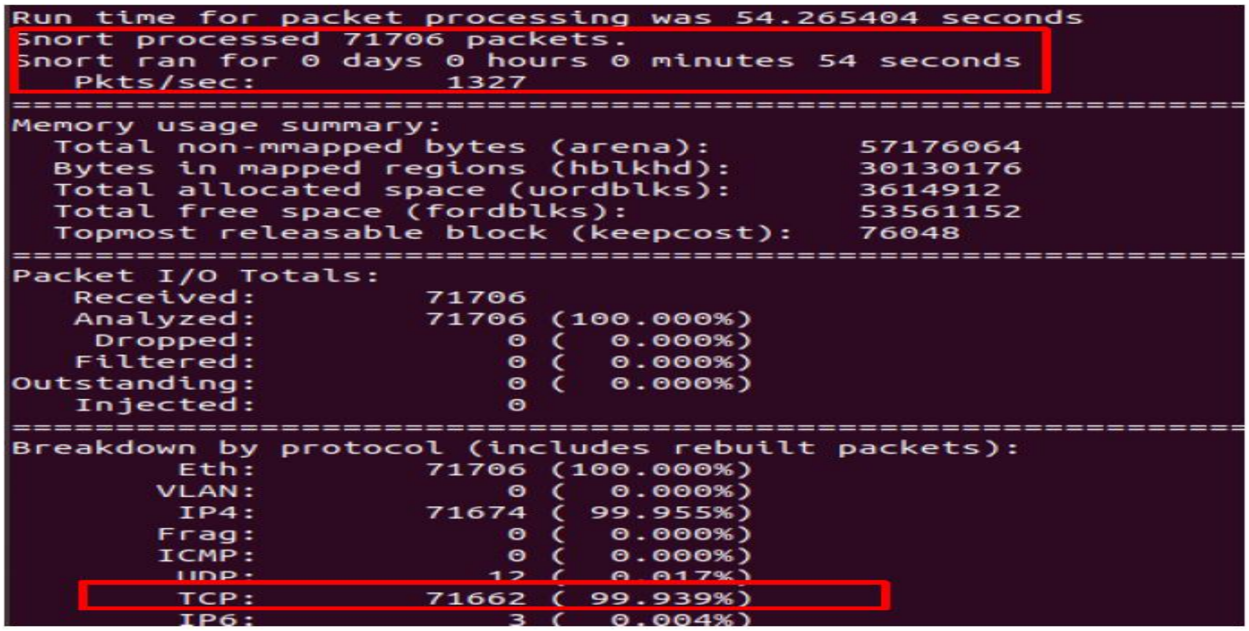
Fig.1

Below is the Generated alert by snort:



```
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.2

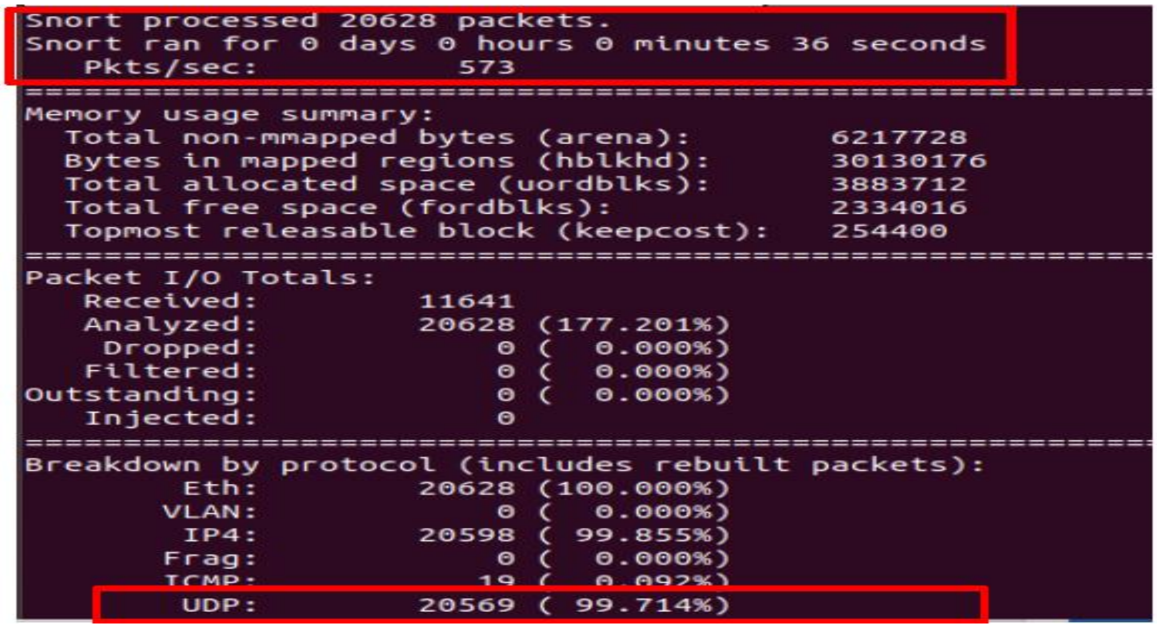


```
Run time for packet processing was 54.265404 seconds
Snort processed 71706 packets.
Snort ran for 0 days 0 hours 0 minutes 54 seconds
   Pkts/sec:             1327
===============================================================================
Memory usage summary:
  Total non-mmapped bytes (arena):       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.3

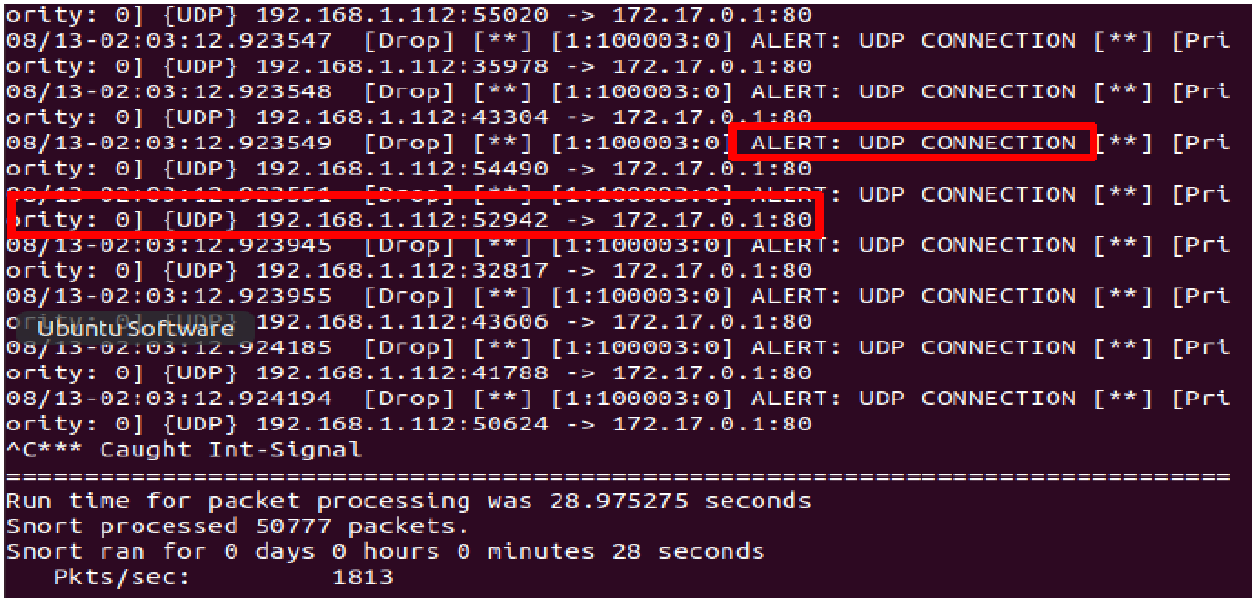
Result: It is clearly reflecting that snort is alerting to unwanted traffic along with information about attacker to victim. Also, the information about the protocol used (TCP) with the port details. Moreover, the accuracy of the analysis of the captured [packets id approximately 99%.

7.2.7) Dos attack and its generated alert (Protocol-UDP)



```
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%)
```

Fig.1



```
ority: 0] {UDP} 192.168.1.112:55020 -> 172.17.0.1:80
08/13-02:03:12.923547  [Drop] [**] [1:100003:0] ALERT: UDP CONNECTION [**] [Pri
ority: 0] {UDP} 192.168.1.112:35978 -> 172.17.0.1:80
08/13-02:03:12.923548  [Drop] [**] [1:100003:0] ALERT: UDP CONNECTION [**] [Pri
ority: 0] {UDP} 192.168.1.112:43304 -> 172.17.0.1:80
08/13-02:03:12.923549  [Drop] [**] [1:100003:0] ALERT: UDP CONNECTION [**] [Pri
ority: 0] {UDP} 192.168.1.112:54490 -> 172.17.0.1:80
08/13-02:03:12.923551  [Drop] [**] [1:100003:0] ALERT: UDP CONNECTION [**] [Pri
ority: 0] {UDP} 192.168.1.112:52942 -> 172.17.0.1:80
08/13-02:03:12.923945  [Drop] [**] [1:100003:0] ALERT: UDP CONNECTION [**] [Pri
ority: 0] {UDP} 192.168.1.112:32817 -> 172.17.0.1:80
08/13-02:03:12.923955  [Drop] [**] [1:100003:0] ALERT: UDP CONNECTION [**] [Pri
ority: 0] {UDP} 192.168.1.112:43606 -> 172.17.0.1:80
08/13-02:03:12.924185  [Drop] [**] [1:100003:0] ALERT: UDP CONNECTION [**] [Pri
ority: 0] {UDP} 192.168.1.112:41788 -> 172.17.0.1:80
08/13-02:03:12.924194  [Drop] [**] [1:100003:0] ALERT: UDP CONNECTION [**] [Pri
ority: 0] {UDP} 192.168.1.112:50624 -> 172.17.0.1:80
^C*** Caught Int-Signal
===============================================================================
Run time for packet processing was 28.975275 seconds
Snort processed 50777 packets.
Snort ran for 0 days 0 hours 0 minutes 28 seconds
   Pkts/sec:           1813
```

Fig.2

Result: It is clearly reflecting that snort is alerting to unwanted traffic along with information about attacker to victim. Also, the information about the protocol used (UDP) with the port details.

8.Conclusion and Discussion

Knowing that scalability of the cloud is mainly depend on the containers and its deployment across the world. However, securing the deployment of the docker container is very important because it is a huge part of any cloud service provider. Additionally, Confidentiality, integrity, availability of any service provider using container technology is critically valuable. Moreover, its protection must be bidirectional that is from inside and outside both. Tough, during our research we have seen many of the protection methodologies for the protection of network and cloud environment perhaps, discovered various limitation for example few approaches do not support various protocols, few of them working with the older technology without even upgradation. Many of the implemented research showed a considerable result but having latency, slowness and sometime not even able to detect the attacks like DOS attack, few approach supported only TCP based filtering and few UDP based.

As we know technology is growing and attack parameters are also growing. So, it is necessary to improve the protection mechanism to reduce the malicious impact. All in one, our contribution states that the implemented Snort-Based IDS is lightweight and scalable and has a capability to monitor and detect the suspicious activity based on the written customized rule and the main advantage it supports many programming languages for the creation of rules. It also able to identify the DOS attack whether it is TCP or UDP based attack.

Focus on Novelty aspect:

The previous approaches illustrated the feasible result. However, the previous demonstrated approach failed to showcase the case where attacker can gain unauthorized access of container in order to execute the malicious activity. Moreover, the researcher had run the container and then injected the malicious payload to evaluate the difference by executing the various technique as mentioned above in the literature review section. Perhaps the older detection technique resulted in high rate of false alarm. That is the detection mechanism was detecting and identifying any activity which were different from the normal regular activity and many times these activities were legitimate as well. However, few of the research detecting the malicious content but not able to prevent from such malicious activities. Moreover, the implemented approach does both detection and prevention which were not completely achieved by previous researcher. Because it is better not only to detect but also provide protection from getting compromised tough it is better to protect the containers.

The novelty aspect of the project also illustrating that as compared to the previous approach the implemented approach is capable of detecting both TCP/UDP based attack along with it capable of DDOS attack detection and protection against REST_API abuse which were not focused earlier.

Comment/suggestion:

Other form of attacks detection: Detection of encrypted attacks along with CSRF (cross site request forgery) would make an impactful enhancement in our approach. Detecting malware if it is running inside container will also be a good achievement.

Improving the implemented approach for other containers technology: The main focus of the research was security of the docker container because of its deployment in real production environment in most of the cloud service provider. Moreover, the similar method of detection can be applied to other Linux container technology because of the similarity in the architecture

Hosting Docker containers on the Cloud platform: Because of resource limitation the research was demonstrated using Virtual machine instead of hosting the containers in Cloud.

Acknowledgment: I would like to thanks my supervisor Dr. Imran Khan for providing me precious information and guidance along with own insights on security concerns and challenges of container, which kept me motivated and encouraged me throughout the journey. I would also like to thanks my family and friends for there limitless support and help.

10.Video Presentation

1) **Video PPT presentation:**

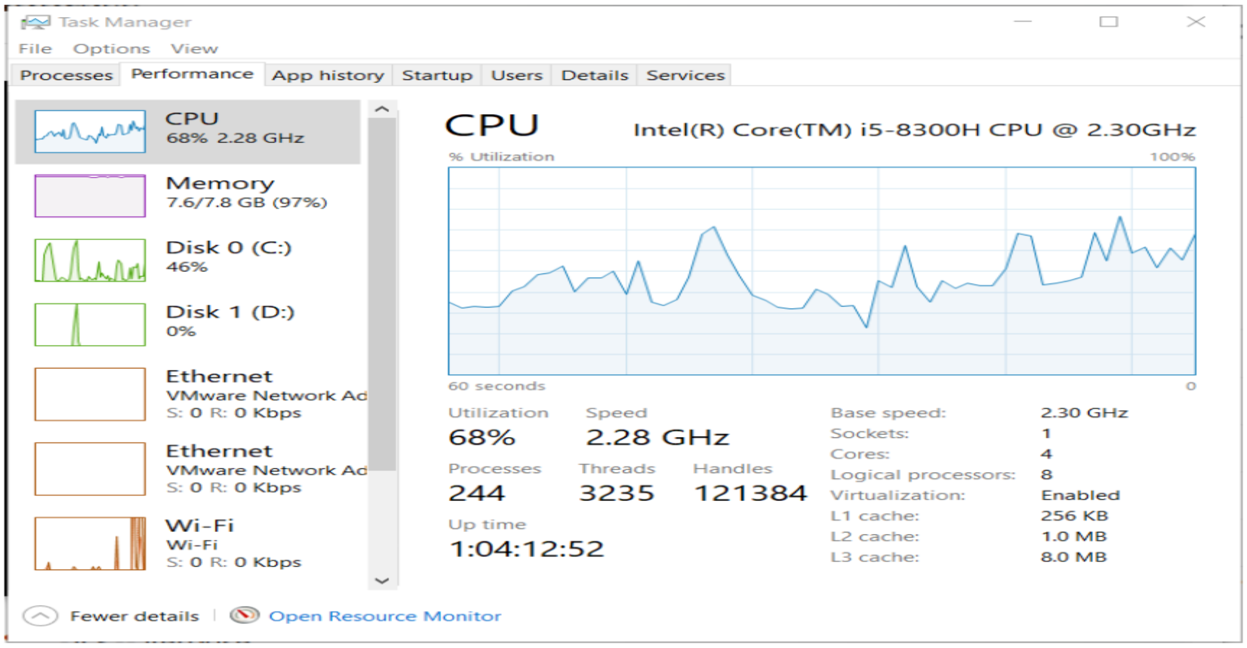
https://drive.google.com/file/d/11Ff0itxtMOFLJO2IhZE93dGZNaJZnJmC/view

2) Demo Video Presentation:

https://drive.google.com/file/d/1rmpLQHxkz9JX3kE64wg6gQiIhKFVhOUj/view

10Appendix:

1) **The below figure illustrates the utilization of resources during launch of DOS attack.**


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
Fewer details
Open Resource Monitor

CPU 68% 2.28 GHz
Memory 7.6/7.8 GB (97%)
Disk 0 (C:) 46%
Disk 1 (D:) 0%
Ethernet VMware Network Ad S: 0 R: 0 Kbps
Ethernet VMware Network Ad S: 0 R: 0 Kbps
Wi-Fi Wi-Fi S: 0 R: 0 Kbps

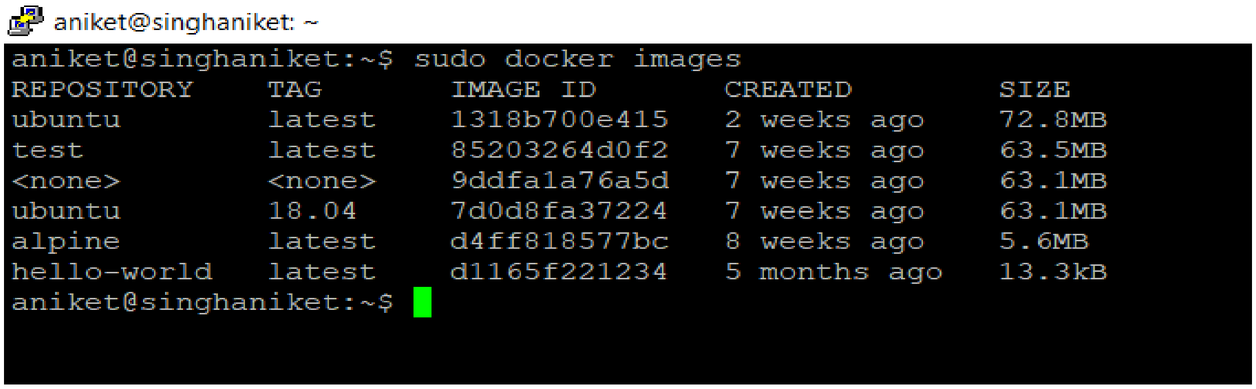
CPU Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
% Utilization 100%
60 seconds 0

Utilization 68%
Speed 2.28 GHz
Processes 244
Threads 3235
Handles 121384
Up time 1:04:12:52

Base speed:	2.30 GHz
Sockets:	1
Cores:	4
Logical processors:	8
Virtualization:	Enabled
L1 cache:	256 KB
L2 cache:	1.0 MB
L3 cache:	8.0 MB

Fig.1

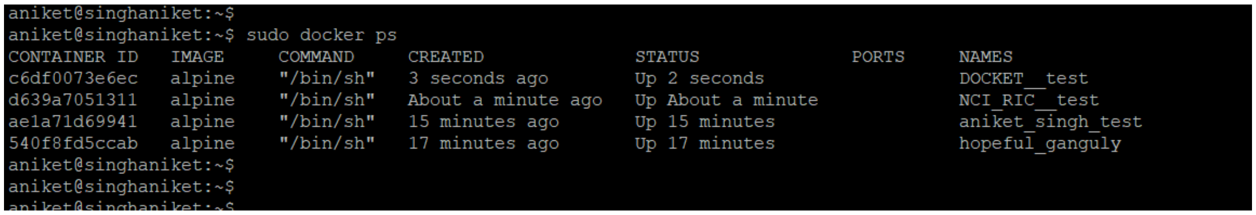
2) Using the below command, we can check the container images.



```
aniket@singhaniket: ~
aniket@singhaniket:~$ sudo docker images
REPOSITORY     TAG       IMAGE ID       CREATED        SIZE
ubuntu         latest    1318b700e415   2 weeks ago    72.8MB
test           latest    85203264d0f2   7 weeks ago    63.5MB
<none>         <none>    9ddfa1a76a5d   7 weeks ago    63.1MB
ubuntu         18.04     7d0d8fa37224   7 weeks ago    63.1MB
alpine         latest    d4ff818577bc   8 weeks ago    5.6MB
hello-world    latest    d1165f221234   5 months ago   13.3kB
aniket@singhaniket:~$
```

Fig.2

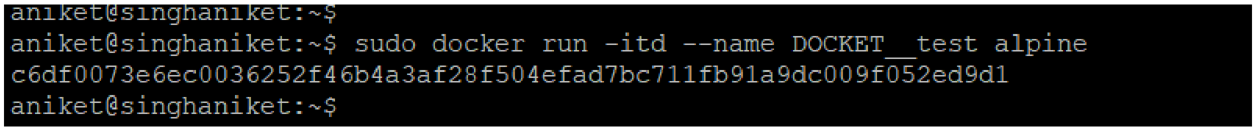
3) The below figure represents running container.



```
aniket@singhaniket:~$
aniket@singhaniket:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED              STATUS              PORTS     NAMES
c6df0073e6ec   alpine    "/bin/sh"   3 seconds ago        Up 2 seconds                  DOCKET__test
d639a7051311   alpine    "/bin/sh"   About a minute ago   Up About a minute             NCI_RIC__test
ae1a71d69941   alpine    "/bin/sh"   15 minutes ago       Up 15 minutes                 aniket_singh_test
540f8fd5ccab   alpine    "/bin/sh"   17 minutes ago       Up 17 minutes                 hopeful_ganguly
aniket@singhaniket:~$
aniket@singhaniket:~$
aniket@singhaniket:~$
```

Fig.3

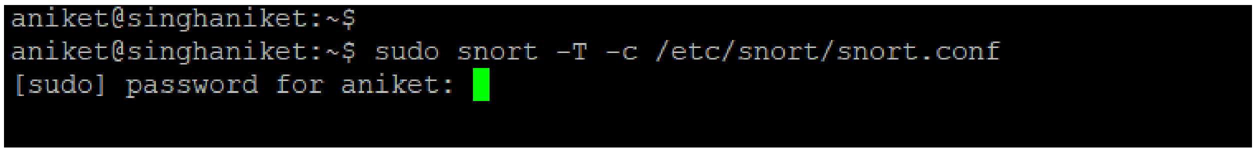
4) The below fig represents starting new container.



```
aniket@singhaniket:~$
aniket@singhaniket:~$ sudo docker run -itd --name DOCKET__test alpine
c6df0073e6ec0036252f46b4a3af28f504efad7bc711fb91a9dc009f052ed9d1
aniket@singhaniket:~$
```

Fig.4

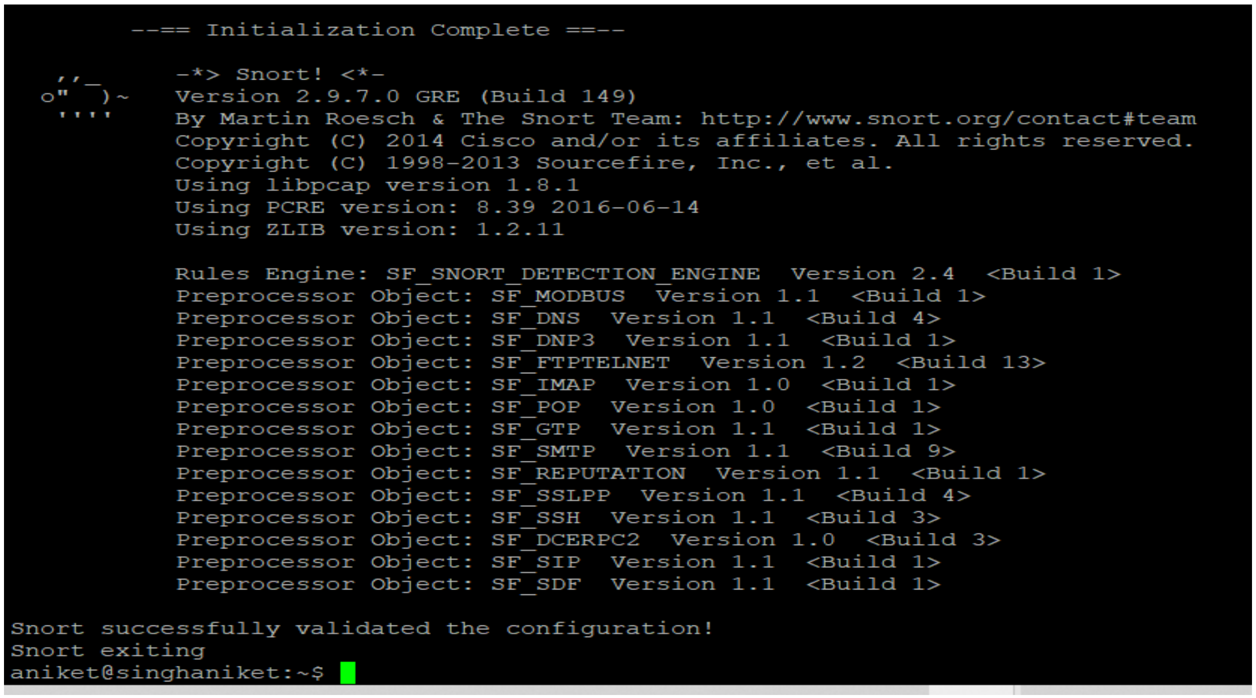
5) Below is the command to test snort rule file.



```
aniket@singhaniket:~$
aniket@singhaniket:~$ sudo snort -T -c /etc/snort/snort.conf
[sudo] password for aniket:
```

Fig.5

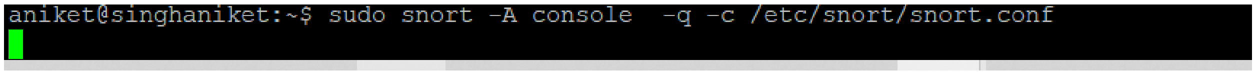
Once the command run successfully, we can get the below output and if the rules are written without any error, then snort validation will take place successfully.



```
        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.7.0 GRE (Build 149)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.8.1
           Using PCRE version: 8.39 2016-06-14
           Using ZLIB version: 1.2.11

           Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 2.4  <Build 1>
           Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
           Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
           Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
           Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
           Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
           Preprocessor Object: SF_POP  Version 1.0  <Build 1>
           Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
           Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
           Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
           Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
           Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
           Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
           Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
           Preprocessor Object: SF_SDF  Version 1.1  <Build 1>

Snort successfully validated the configuration!
Snort exiting
aniket@singhaniket:~$
```

Fig.6

6) Once all the configuration part is ready and validation completed successfully then using the below command, we can make our IDS ready to detect the activity.



```
aniket@singhaniket:~$ sudo snort -A console  -q -c /etc/snort/snort.conf
```

Fig.7

7) Below bar graph illustrates vulnerabilities docker in last five years with its type.

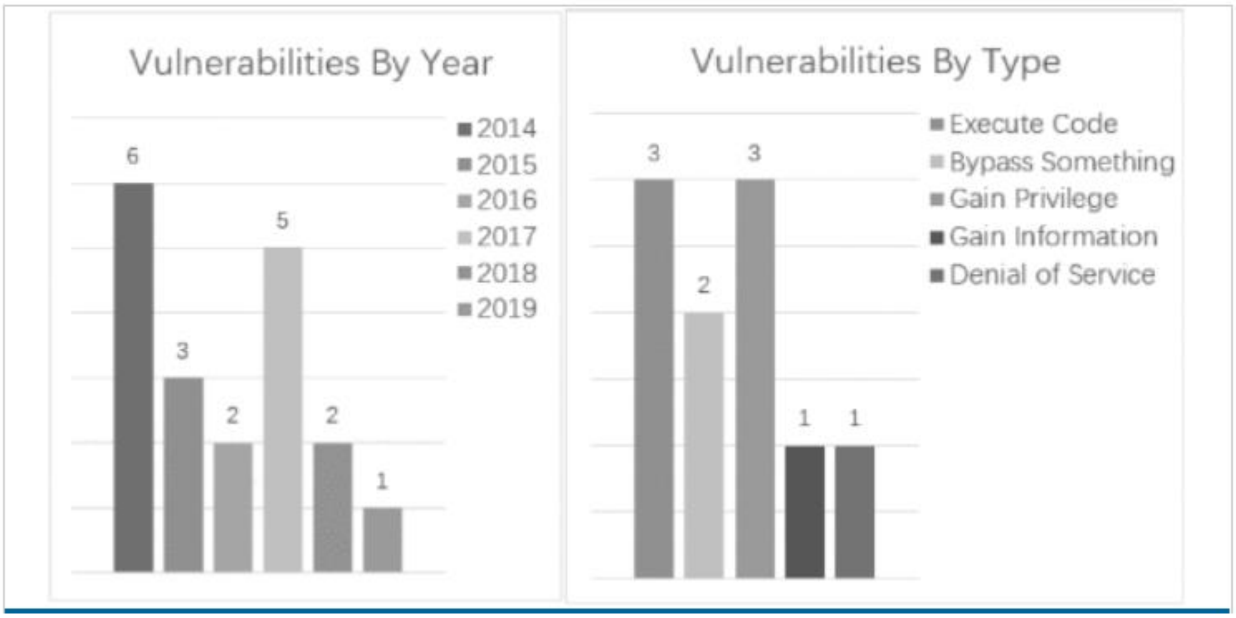


Fig.8(Huang et al., 2019)

11.Reference

1)Abed, A.S., Azab, M., Clancy, C., Kashkoush, M.S., 2019. Resilient intrusion detection system for cloud containers. Int. J. Commun. Netw. Distrib. Syst. 24, 1. https://doi.org/10.1504/IJCNDS.2020.103857

2)Al-Dalky, R., Salah, K., Al-Qutayri, M., Otrok, H., 2014. Framework for a NetFPGA-based Snort NIDS, in: 2014 9th International Symposium on Communication Systems, Networks Digital Sign (CSNDSP). Presented at the 2014 9th International Symposium on Communication Systems, Networks Digital Sign (CSNDSP), pp. 380–383. https://doi.org/10.1109/CSNDSP.2014.6923858

3)Aydın, M.A., Zaim, A.H., Ceylan, K.G., 2009. A hybrid intrusion detection system design for computer network security. Comput. Electr. Eng. 35, 517–526. https://doi.org/10.1016/j.compeleceng.2008.12.005

4)Bhatia, G., Choudhary, A., Dadheech, K., 2018. Behavioral Analysis of Docker Swarm Under DoS/ DDoS Attack, in: 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT). Presented at the 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), pp. 985–991. https://doi.org/10.1109/ICICCT.2018.8472953

5)Chelladhurai, J., Chelliah, P.R., Kumar, S.A., 2016. Securing Docker Containers from Denial of Service (DoS) Attacks, in: 2016 IEEE International Conference on Services Computing

(SCC). Presented at the 2016 IEEE International Conference on Services Computing (SCC), pp. 856–859. https://doi.org/10.1109/SCC.2016.123

6)Chiba, Z., 2016. A Cooperative and Hybrid Network Intrusion Detection Framework in Cloud Computing Based on Snort and Optimized Back Propagation Neural Network. Procedia Comput. Sci. 7.

7)Combe, T., Martin, A., Pietro, R.D., 2016. To Docker or Not to Docker: A Security Perspective. IEEE Cloud Comput. 3, 54–62. https://doi.org/10.1109/MCC.2016.100

8)Flora, J., Antunes, N., 2019. Studying the Applicability of Intrusion Detection to Multi-Tenant Container Environments. 2019 15th Eur. Dependable Comput. Conf. EDCC Dependable Comput. Conf. EDCC 2019 15th Eur. 133–136. https://doi.org/10.1109/EDCC.2019.00033

9)Hatef, M.A., Shaker, V., Jabbarpour, M.R., Jung, J., Zarrabi, H., 2018. HIDCC: A hybrid intrusion detection approach in cloud computing. Concurr. Comput. Pract. Exp. 30, e4171. https://doi.org/10.1002/cpe.4171

10)Huang, D., Cui, H., Wen, S., Huang, C., 2019. Security Analysis and Threats Detection Techniques on Docker Container, in: 2019 IEEE 5th International Conference on Computer and Communications (ICCC). Presented at the 2019 IEEE 5th International Conference on Computer and Communications (ICCC), pp. 1214–1220. https://doi.org/10.1109/ICCC47050.2019.9064441

11)Jaeyeon Jung, Paxson, V., Berger, A.W., Balakrishnan, H., 2004. Fast portscan detection using sequential hypothesis testing, in: IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004. Presented at the IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004, pp. 211–225. https://doi.org/10.1109/SECPRI.2004.1301325

12)Lee, J.-H., Park, M.-W., Eom, J.-H., Chung, T.-M., 2011. Multi-level Intrusion Detection System and log management in Cloud Computing, in: 13th International Conference on Advanced Communication Technology (ICACT2011). Presented at the 13th International Conference on Advanced Communication Technology (ICACT2011), pp. 552–555.

13)Manu, A.R., Patel, J.K., Akhtar, S., Agrawal, V.K., Murthy, K.N.B.S., 2016. Docker container security via heuristics-based multilateral security-conceptual and pragmatic study, in: 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT). Presented at the 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), pp. 1–14. https://doi.org/10.1109/ICCPCT.2016.7530217
14)Mishra, P., Pilli, E.S., Varadharajan, V., Tupakula, U., 2017. Out-VM monitoring for Malicious Network Packet Detection in cloud, in: 2017 ISEA Asia Security and Privacy

(ISEASP). Presented at the 2017 ISEA Asia Security and Privacy (ISEASP), pp. 1–10. https://doi.org/10.1109/ISEASP.2017.7976995

15)Patel, S.K., Sonker, A., 2016. Internet Protocol Identification Number Based Ideal Stealth Port Scan Detection Using Snort, in: 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN). Presented at the 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN), pp. 422–427. https://doi.org/10.1109/CICN.2016.89

16)Sagala, A., 2015. Automatic SNORT IDS rule generation based on honeypot log, in: 2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE). Presented at the 2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE), pp. 576–580. https://doi.org/10.1109/ICITEED.2015.7409013

17)Sengaphay, K., Saiyod, S., Benjamas, N., 2016. Creating Snort-IDS Rules for Detection Behavior Using Multi-sensors in Private Cloud, in: Kim, K.J., Joukov, N. (Eds.), Information Science and Applications (ICISA) 2016, Lecture Notes in Electrical Engineering. Springer, Singapore, pp. 589–601. https://doi.org/10.1007/978-981-10-0557-2_58

18)Singh, S., Singh, N., 2016. Containers amp; Docker: Emerging roles amp; future of Cloud technology, in: 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (ICATccT). Presented at the 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), pp. 804–807. https://doi.org/10.1109/ICATCCT.2016.7912109

19)Slominski, A., Muthusamy, V., Khalaf, R., 2015. Building a Multi-tenant Cloud Service from Legacy Code with Docker Containers, in: 2015 IEEE International Conference on Cloud Engineering. Presented at the 2015 IEEE International Conference on Cloud Engineering, pp. 394–396. https://doi.org/10.1109/IC2E.2015.66

20)Srinivasan, S., Kumar, A., Mahajan, M., Sitaram, D., Gupta, S., 2019. Probabilistic Real-Time Intrusion Detection System for Docker Containers, in: Thampi, S.M., Madria, S., Wang, G., Rawat, D.B., Alcaraz Calero, J.M. (Eds.), Security in Computing and Communications, Communications in Computer and Information Science. Springer, Singapore, pp. 336–347. https://doi.org/10.1007/978-981-13-5826-5_26

21)Sultan, S., Ahmad, I., Dimitriou, T., 2019. Container Security: Issues, Challenges, and the Road Ahead. IEEE Access 7, 52976–52996. https://doi.org/10.1109/ACCESS.2019.2911732

22)Talbot, J., Pikula, P., Sweetmore, C., Rowe, S., Hindy, H., Tachtatzis, C., Atkinson, R., Bellekens, X., 2020. A Security Perspective on Unikernels, in: 2020 International Conference on

Cyber Security and Protection of Digital Services (Cyber Security). Presented at the 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), pp. 1–7. https://doi.org/10.1109/CyberSecurity49315.2020.9138883

23)Thinh, T.N., Hieu, T.T., Van Quoc Dung, Kittitornkun, S., 2012. A FPGA-based deep packet inspection engine for Network Intrusion Detection System, in: 2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology. Presented at the 2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, pp. 1–4. https://doi.org/10.1109/ECTICon.2012.6254301

24)Wan, Z., 2011. A Network Virtualization Approach in Many-core Processor Based Cloud Computing Environment, in: 2011 Third International Conference on Computational Intelligence, Communication Systems and Networks. Presented at the 2011 Third International Conference on Computational Intelligence, Communication Systems and Networks, pp. 304–307. https://doi.org/10.1109/CICSyN.2011.70

25)Wenhao, J., Zheng, L., 2020. Vulnerability Analysis and Security Research of Docker Container, in: 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE). Presented at the 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE), pp. 354–357. https://doi.org/10.1109/ICISCAE51034.2020.9236837

26)Xiaoyong, L., Dongxi, L., 2005. An automatic scheme to construct Snort rules from honeypots data. J. Syst. Eng. Electron. 16, 466–470.

27)Zhang, G., Brown, P., Li, G., Farouk, A., Zhen, D., 2019. Research on personal intelligent scheduling algorithms in cloud computing based on BP neural network. J. Intell. Fuzzy Syst. 37, 3545–3554. https://doi.org/10.3233/JIFS-179158