SNORT v3

Network Security Laboratory Project

Group 4: Edoardo Mich, Davide Moletta, Tefera Addis Sisay

University of Trento

June 4, 2023

Contents

1 Setup

1.1 Snort++

An Intrusion Detection System (IDS) and an Intrusion Prevention System (IPS) are two critical security technologies used in today's modern network infrastructure. An IDS monitors network traffic for suspicious behavior such as unauthorized access or malicious activities, while an IPS takes this a step further and can actively block or prevent these types of attacks. Snort is one of the most widely used open-source (formerly maintained by Cisco) IDS/IPS available, offering advanced capabilities like protocol analysis, content searching, and packet logging in addition to the classical packet sniffer. Snort can be deployed also inline to stop these packets as well, gaining IPS capabilities.

In this laboratory will be used the new version of Snort called Snort++ (or simply Snort3). This new version is a complete rewrite of Snort using C++ to allow multi-threaded processing and more modular platform. This modular approach allow a bigger number of plugins written by the community specifically to track particulars packets and flows like OpenAppID, or to implement hardware acceleration for low level jobs like Hyperscan for the pattern matching[2]. Also the syntax and the requisites for the rules are changed[1] allow more robust and fast analysis.

This new version is released in 2014 but still today is not as widespread as the major release Snort 2.9 released in 2010. Because of that we decide to use the new version to explain directly the new advantages and direct towards the use of this innovative version that sooner or later it will become the new standard one for Snort.

The big disadvantage of Snort++ (from now on called Snort3 or simply Snort) is that there are no packet available fro any common Linux Distro and the only way to install is compile from source[1]. Because of that, after choosing Alpine 3.17 as base Linux image for this lab infrastructure we create an APKBUILD file (for version 3.1.61) as show below that is used to create packet for the Alpine VMs, without the need to installing all the build dependency and source code on the final machines. The resulting packets after performing the necessary stability checks they will be made public available by uploading them to the Alpine package repository.

Snort has a strict dependency called `libdaq` that must be install on the system before installing the snort packet itself. LibDAQ: The Data AcQuisition Library[2] is the pluggable abstraction layer used for interacting with the network interface or network data plane. Using its specific API, Snort is able to intreact with all the pluggable DAQ modules as `afpacket`, `pcap`, `nfq` etc. Below to be thorough is reported also the APKBUILD file for version 3.0.11 of `libdaq`.

Listing 1: APKBUILD for `libdaq-3.0.11`

```
pkgname=daq
pkgver=3.0.11
pkgrel=0
pkgdesc="LibDAQ is a pluggable abstraction layer for interacting with a data source"
url="https://github.com/snort3/libdaq"
arch="all"
```

[1] The repository is available at `https://github.com/snort3/snort3`
[2] The repository is available at `https://github.com/snort3/libdaq`

```
license="GPL V2"
makedepends="build-base autoconf automake libtool libpcap-dev"
source="$pkgname-$pkgver.tar.gz::
        https://github.com/snort3/libdaq/archive/refs/tags/v$pkgver.tar.gz"
builddir="$srcdir/lib$pkgname-$pkgver"
options="!fhs"

build() {
    ./bootstrap
    ./configure
    make
}

package() {
    make DESTDIR="$pkgdir" install
}

sha512sums="b794da0a1297e74f29940eeb956a294ed1b3c77695659d9fa540a... daq-3.0.11.tar.gz"
```

Listing 2: APKBUILD for `snort-3.1.61`

```
pkgname=snort3
pkgver=3.1.61.0
pkgrel=0
pkgdesc="Snort 3 is the next generation Snort IPS (Intrusion Prevention System)"
url="https://github.com/snort3/snort3"
arch="all"
license="GPL V2"
depends="daq"
makedepends="cmake build-base autoconf automake cpputest flex-dev libuuid libtool
        hwloc-dev libnet-dev libdnet-dev libpcap-dev libpcre32 libtirpc-dev luajit-dev
        openssl-dev libssl3 libnetfilter_queue-dev libmnl-dev libunwind-dev zlib-dev pcre-dev
        xz-dev gperftools-dev"
source="$pkgname-$pkgver.tar.gz::
        https://github.com/snort3/snort3/archive/refs/tags/$pkgver.tar.gz"
builddir="$srcdir/$pkgname-$pkgver"
options="!fhs"

build() {
    ./configure_cmake.sh --prefix=/usr/local --enable-tcmalloc
}

package() {
    cd ./build
    make
    make DESTDIR="$pkgdir" install
}

sha512sums="7ffd783354b4f41971e000f169e3f80e59158b3afc381564e3c4b... snort3-3.1.61.0.tar.gz"
```

1.2 Network Configuration

Before entering into the network infrastructure of the lab, must be presented the operation and differences between the DAQ modules.

In this lab are been used three DAQ modules:

- `pcap`: is the default DAQ modules and could be used to read from pcap file or attaching directly to a device (passive mode).
- `afpacket`: similar to the memory mapped pcap DAQ but no external library is required to work. Could be also used in inline mode (creating a bridge between two promiscuous interfaces. One interface cannot be connected to multiple interfaces, the connection ratio is 1:1) specifying the pair like `eth0:eth1`
- `nfq`: is the new and improved way to process iptables packets. Snort is attaching to NetFilterQueue at the iptables level.

A benefit of AFPACKET is that you can install an inline Snort machine without any changes to IP addressing, routing, or networking changes. Infact AFPACKET is simpler to setup but only lets bridge sets of paired interfaces. This means the interfaces eth0 and eth1 can be bridged (pass traffic between them), and also for eth2 with eth3, but you can not pass traffic between eth0 and eth4.

Taking in consideration this aspects the initial the network configuration is setted up as described in figure 1. The Snort machine is configured with the an interface in promiscuous mode (both in Virtualbox and Linux settings) and the two other interfaces that connects the two subnets. This infrastructure allow to run Snort in IDS mode in every interface, but restrict the IPS mode only between eth1 and eth2, so no prevention action can be performed on the subnet 77 (for example between client 101 and client 102).

Figure 1: Network configuration for AFPacket

Because of that, the network infrastructure is changed as describe figure 2. The Snort machine and the router are now decoupled. All the client in the subnet 77 could communicate thanks to the Linux Bridge created on the Snort machine between the interfaces (in promiscuous mode) that represent the singles connection (P2P) of each host. In this case

without powering on the Snort machine the client could not communicate, but it's part of the pro and cons game of using Snort an intersubnet IPS.

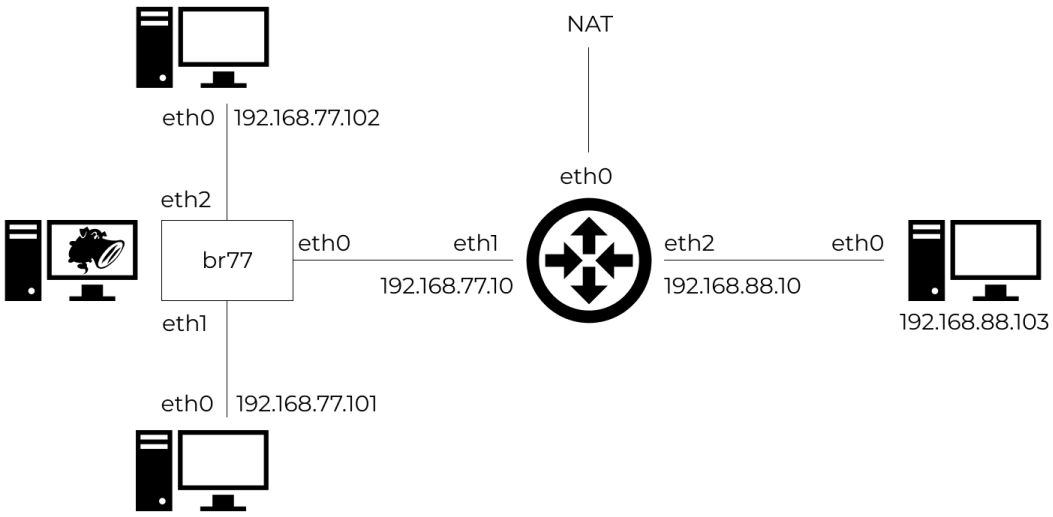

Figure 2: Current network configuration

The VMs described in the figure 2 are the following (using the name defined in the OVA)

- Alpine_NS: no IP
- Alpine_NS_client_101: 192.168.77.101
- Alpine_NS_client_102: 192.168.77.102
- Alpine_NS_client_103: 192.168.88.103
- Alpine_NS_router: 192.168.77.10 [eth1] + 192.168.88.10 [eth2]

During the exercise of IDS the AFPACKET DAQ mode is used directly on the bridge `br77`, but to allow the IPS exercise to work, a change in the Linux network stack must be made. To overcome the problem if the list of interfaces pairs used by AFPACKET, the IPS part is performed using the NFQ module.

NFQ in respect to AFPACKET lets you leverage the power of iptables to make routing decisions. It's more complicated to setup, but is more powerful. You can also use iptables to block traffic before Snort even sees it, being executed on an upper level of the Linux network stack.

To allow the netfilter queue to intercept also the bridge traffic (that for the hierarchy of Linux network stack is managed in a lower level) we must enable the flag `bridge-nf-call-iptables` of the system as can be seen from the configuration belo.

Listing 3: `/etc/network/interfaces` of the Alpine_NS (Snort) machine

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet manual
    up ip link set eth0 promisc on

auto eth1
iface eth1 inet manual
    up ip link set eth1 promisc on

auto eth2
iface eth2 inet manual
    up ip link set eth2 promisc on

auto eth3
iface eth3 inet dhcp

auto br77
iface br77 inet static
    bridge-ports eth0 eth1 eth2
    bridge-stp 0
    pre-up echo 1 > /proc/sys/net/bridge/bridge-nf-call-iptables
```

In addition to that, also a iptables rules must be provided to create the queue and choose what traffic forward through it. The command following command is executed when the user run the script `snort/enableNFQ.sh` (the parameter `--queue-bypass` permit the traffic to flow even if no user-level process is attached to the queue)

```
iptables -I FORWARD -j NFQUEUE --queue-num=0 --queue-bypass
```

2 Rules syntax

Snort defines a rule syntax used to write rules that will be matched against the specified traffic to find correct matches.

A generic rule is of the type

```
action protocol IPaddress port# -> IPaddress port# (options)
```

We will now go deeper into each word meaning and how to tune your rules.

2.1 Snort Actions

```
action protocol IPaddress port -> IPaddress port (options)
```

Snort actions are used to specify what Snort should do when the rule fires, they can be divided into two subcategories.

The basic actions are:

1. Alert: generate an alert on the single packet
2. Block: block the current packet and all the subsequent packets in this flow
3. Drop: drop the current packet
4. Log: log the current packet
5. Pass: mark the current packet as passed

The second group is the one of active responses that perform some action in response to the packet being detected, they are:

1. React: send response to client and terminate session.
2. Reject: terminate session with TCP reset or ICMP unreachable
3. Rewrite: enables overwrite packet contents based on a "replace" option in the rules

2.2 Protocol

```
action protocol IPaddress port -> IPaddress port (options)
```

The protocol field tells Snort what type of protocols a given rule should look at, the currently supported ones are:

1. IP
2. ICMP
3. TCP
4. UDP

A rule can have only one protocol set at a time so, if we want to filter UDP and TCP traffic we need to write two different rules. Snort also support services such as:

1. HTTP
2. SSL
3. SMTP
4. POP3
5. and many more

2.3 IP addresses

```
action protocol IPaddress port -> IPaddress port (options)
```

IP addresses are used to tell Snort what source and destination IP addresses a given rule should apply to. A rule will only match if the source and destination IP addresses of a given packet match the IP addresses set in that rule.

IP addresses can be set in four different ways:

1. Numeric IP address: 192.168.77.101, 192.168.77.0/24
2. Variable (using $) defined in the Snort config: $HOME_NET
3. The keyword any, meaning any IP address
4. List of IP addresses or subnets: [192.168.77.0/24,10.1.1.0/24]

When writing the rule its important to remember that the first IP address is the source one while the second is the destination one.

2.4 Port numbers

```
action protocol IPaddress port# -> IPaddress port# (options)
```

Port numbers are used to tell Snort to apply a given rule to traffic sent from or sent to the specified source and destination ports.

Ports can be declared in five different ways:

1. The keyword any, meaning any port
2. Static port: 8080
3. Variable defined in the Snort config: $HTTP_PORTS
4. Port ranges indicated with the range operator: 1:1024
5. List of static ports: [1:1024,4444,5555]

hen writing the rule its important to remember that the first port number is the source one while the second is the destination one.

2.5 Direction operators

`action protocol IPaddress port` **`->`** `IPaddress port (options)`

The direction operator of a header indicates the direction of the traffic that the rule should apply to. There are two valid direction operators: There are two valid direction operators that indicates the direction of the traffic that the rule should apply to:

1. $->$: The first IP: port# is the source and the second is the destination
2. $<>$: Both addresses: port# are source and destination

2.6 Snort options

`action protocol IPaddress port -> IPaddress port` **`(options)`**

Rule options are used to specify further controls or actions that Snort has to check or execute when the rule fires.

Each rule option has its own set of option-specific criteria, but they all follow the same general structure. Each rule option is declared with its name followed by a *:* character and any option-specific criteria. Then, each rule option is terminated with a *;* character. Lastly, each option is separated by a *,* character.

There are four major categories of rule options:

1. General options: provide additional context for a given rule such as:
   (a) msg: sets the message to be printed out when a rule matches
   (b) sid: identifies the unique signature number assigned to a given Snort rule
   (c) priority: sets a severity level for appropriate event prioritizing
2. Payload options: set payload-specific criteria such as:
   (a) content: is used to perform basic string and/or hexadecimal pattern matching
   (b) fast_pattern: is a content modifier that tells Snort to use that particular match to determine if further rule processing should continue against the traffic
   (c) nocase: is a content modifier that tells Snort to ignore case when looking for a specified pattern
   (d) pcre: is used to create perl compatible regular expressions
3. Non-payload options: set non-payload specific criteria such as:
   (a) flags: checks the TCP header for specific TCP flag bits
   (b) id: looks for specific IP header ID values
   (c) itype: looks for specific ICMP type values
4. Post-detection options: set actions to take on a given packet after the rule has fired, they are:
   (a) detection_filter: sets the rate in which the rule must hit before an event gets generated
   (b) replace: is used to match and then overwrite payload data
   (c) tag: is used to log additional packets after a rule event

3 IDS Exercises

Please refer to the Snort cheat sheet prepared by us and to the official Snort guide on rules syntax [3] for additional help during the exercises.

3.1 Exercise 1

For the exercise 1 the goal is to write a Snort rule to detect and alert ping from any ip address to any ip address.

Before we write the rule we can see that with the command

```
ping 192.168.77.101 // 192.168.77.102 // 192.168.88.103
```

we can ping each client from each client.

We can now write the following rule

```
alert icmp any any -> any any
(
   sid:1
)
```

inside the exercise01.rules file.

We can now run Snort with the command

```
snort3 -i br77 -R snort/rules/exercise01.rules
```

Now if we try to run the command

```
ping 192.168.77.101 // 192.168.77.102 // 192.168.88.103
```

we can see that every ECHO request and ECHO reply are logged by Snort as seen below:

```
[**] [1:1:0] "no msg in rule" [**]
[Priority: 0]
06/04-14:50:08.675044 192.168.77.101 -> 192.168.77.102
ICMP TTL:64 TOS:0x0 ID:48370 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:1858  Seq:3 ECHO

[**] [1:1:0] "no msg in rule" [**]
[Priority: 0]
06/04-14:50:08.676622 192.168.77.102 -> 192.168.77.101
ICMP TTL:64 TOS:0x0 ID:49172 IpLen:20 DgmLen:84
Type:0 Code:0 ID:1858 Seq:3 ECHO REPLY
```

3.2 Exercise 2

For the exercise 2 the goal is to write a Snort rule to detect, alert and write a message for every ping from client 101 (192.168.77.101) to client 102 (192.168.77.102).

Before we write the rule we can see that we can ping the two clients in both directions.

To write the required rule we need to specify the Snort option *msg:“”* which allow us to write a specific message when the corresponding rule fires.

We can now write the following rule

```
alert icmp 192.168.77.101 any -> 192.168.77.102 any
(
   sid:2;
   msg:"ping from client 101 to client 102"
)
```

inside the exercise02.rules file.

We can now run Snort with the command

```
snort3 -i br77 -R snort/rules/exercise02.rules
```

Now if we try to run the command

```
ping 192.168.77.101 // 192.168.77.102
```

we can see that ECHO request and ECHO reply sent by client 101 are logged by Snort and the message is displayed.

3.3 Exercise 3

For the exercise 3 the goal is to write a Snort rule to detect, alert and write a message for every ping reply from client 101 (192.168.77.101) to client 102 (192.168.77.102).

Before we write the rule we can see that with the command

```
ping 192.168.77.101 // 192.168.77.102
```

we can ping each client from each client.

To write the required rule we need to specify the Snort option *itype:* which allow us to match the rule only to specific types of ICMP messages, in this case we can run Wireshark to inspect the communications between the two clients. We can clearly see that ECHO request messages have are of type 8 while ECHO replies are of type 0.

We can now write the following rule

```
alert icmp 192.168.77.101 any -> 192.168.77.102 any
(
   itype:0;
   sid:3;
   msg:"ping reply from client 101 to client 102"
)
```

inside the exercise03.rules file.

We can now run Snort with the command

```
snort3 -i br77 -R snort/rules/exercise03.rules
```

Now if we try to run the command

```
ping 192.168.77.101
```

we can see that ECHO replies sent by client 101 are logged by Snort and the message is displayed while if client 101 pings client 102 we see no alerts since client 101 is sending only ECHO request (of type 8).

3.4 Exercise 4

For the exercise 4 the goal is to write a Snort rule to detect, alert and write a message for every TCP packet sent from client 101 (192.168.77.101) to client 103 (192.168.88.103).

We have prepared a python script with Scapy that is able to craft and send a TCP packet from the client on which it is launched to any other client of the network. The code of the packet is the one below.

```
#! /usr/bin/env python
import scapy.all as scapy
import sys
import random as rand

sourceIP = scapy.get_if_addr("eth0")

destIP = str( input("insert the destination IP address: "))
while destIP != "192.168.77.102" and destIP != "192.168.88.103" :
    destIP = str( input("invalid destination IP address, insert a new
        one: "))

sourcePort = rand.randrange(2000, 2500)
destPort = rand.randrange(2000, 2500)
sequence = rand.randrange(1, 9999)
ttl = 64
tcpFlags = "S"
payload = "hello world"
```

```
#packet build
ip = scapy.IP(
    src = sourceIP,
    dst = destIP,
    ttl=ttl
    )
tcp = scapy.TCP(
    sport = sourcePort,
    dport = destPort,
    flags = tcpFlags,
    seq = sequence)

packet = ip / tcp / scapy.Raw(payload)
scapy.send(packet)
```

Before we write the rule we can see that with the command

```
python3 packet.py
#give as input 192.168.77.102 or 192.168.88.103
```

we can send the packet to the specified client.

We can now write the following rule

```
alert tcp 192.168.77.101 any -> 192.168.88.103 any
(
   sid:4;
   msg:"A TCP packet from client 101 to client 103"
)
```

inside the exercise04.rules file.

We can now run Snort with the command

```
snort3 -i br77 -R snort/rules/exercise04.rules
```

Now if we try to run the command

```
python3 packet.py
#give as input 192.168.88.103
```

we can see that TCP packets sent from client 101 to client 103 are logged by Snort and the message is displayed, while if client 102 or client 103 send the packet to client 101 we see no alerts.

3.5 Exercise 5

For the exercise 5 the goal is to write a Snort rule to detect, alert and write a message for TCP packets with ACK flags sent from client 101 (192.168.77.101) to client 103 (192.168.88.103).

Before we write the rule we can see that with the command

```
nc -lp 666
```

we can start listening from client 103.

And with the command

```
nc 192.168.88.103 666
```

we can connect to client 103 from client 101 and send messages to it.

To write the required rule we need to specify the Snort option *flags:* which allow us to match the rule only to specific types of TCP packets, in this case we can use *flags:A* to filter only packets with the ACK flag enabled.

We can now write the following rule

```
alert tcp 192.168.77.101 any -> 192.168.88.103 any
(
   flags:A;
   sid:5;
   msg:"ACK tcp packet from client 101 to client 103"
)
```

inside the exercise05.rules file.

We can now run Snort with the command

```
snort3 -i br77 -R snort/rules/exercise05.rules
```

Now if we try to run the command

```
nc -lp 666
```

on client 103.

And the command

```
nc 192.168.88.103 666
```

on client 101 we can see that TCP packets sent by client 101 are logged by Snort and the message is displayed, however the rule we wrote logs only the TCP packets with every flag disabled and the ACK flag enabled. To write a more complete rule we need to add a * before the A flag, in this way Snort will log every TCP packet that has at least the ACK flag enabled without caring about other flags.

The updated rule is

```
alert tcp 192.168.77.101 any -> 192.168.88.103 any
(
   flags:*A;
   sid:5;
   msg:"ACK tcp packet from client 101 to client 103"
)
```

Now if we try again and we can see that also executing the pythons script of the exercise 4, where the destination reply with an ACK/RST packet, Snort generate an alert specifying the two flag of the packet.

3.6 Exercise 6

For the exercise 6 the goal is to write a Snort rule to detect, alert and write a message for DNS request from the home network to any DNS server but the company one. In our example the home network is the .77 subnet while the company DNS server is client 103 (192.168.88.103).

Before we write the rule we can see that with the command

```
nslookup google.com 1.1.1.1
nslookup google.com 192.168.88.103
```

some DNS requests are forwarded respectively to the DNS server 1.1.1.1 and 192.168.88.103.

To write the required rule we need to use Snort variables instead of static ip addresses. As we can see in the configuration file snort_ids.lua the variables are set as follows

```
HOME_NET = '192.168.77.0/24'

DNS_SERVER = [[192.168.88.103]]

EXTERNAL_NET = 'any'
```

We can now write the following rule

```
alert udp $HOME_NET any -> !$DNS_SERVER 53
(
   sid:6;
   msg:"DNS request to a not standard DNS server"
)
```

inside the exercise06.rules file.

We can now run Snort with the command

```
snort3 -i br77 -R snort/rules/exercise06.rules -c
    snort/confs/snort_ids.lua
```

Now if we try to run the command

```
nslookup google.com 1.1.1.1
nslookup google.com 192.168.88.103
```

on client 101 we can see that the request sent to 1.1.1.1 is logged by Snort since that ip address is not the one of the company's DNS server while the request to the correct company's DNS server is not logged.

3.7 Exercise 7

For the exercise 7 the goal is to write a Snort rule to detect, alert and write a message for HTTP requests with unallowed User-Agents from the home network to any client 103. In our example the only allowed User-Agent is *Wget*.

Before we write the rule we can see that with the command

```
python3 -m http.server 666
```

we can start a basic server on client 103.

And with the below commands we can connect to such server from client 101 or client 102.

```
wget 192.168.88.103:666
curl 192.168.88.103:666
```

To write the required rule we need to use the Snort options *http_header: field header_field* and *content: “”*. The first allow Snort to match the rule against only one part of the HTTP header already parsed to ASCII while the second is used to check the content of the specified field.

We can now write the following rule

```
alert http $HOME_NET any -> any any
(
   http_header:field user-agent;
   content:!"wget", nocase;
   sid:7;
   msg:"Unallowed user-agent in HTTP request from HOME_NET"
)
```

inside the exercise07.rules file.

We can now run Snort with the command

```
snort3 -i br77 -R snort/rules/exercise07.rules -c
    snort/confs/snort_ids.lua
```

Now if we try to run the command

```
python3 -m http.server 666
```

on client 103.

And executing the commands

```
wget 192.168.88.103:666
curl 192.168.88.103:666
```

on client 101 or client 102 we can see that the request with *curl* is logged by Snort since the User-Agent is not of the allowed type while the request with *wget* is not logged since its user-agent is allowed.

3.8 Exercise 8

For the exercise 8 the goal is to write a Snort rule to detect, alert and write a message for SQL injections. This exercise is especially made to show the differences between Snort and Zeek (explained by group 5). This exercises will be demonstrated through a .pcap file created with Wireshark that contains the communication with a server created by group5 that is vulnerable to SQL injections.

Before we write the rule we can examine the .pcap file with Wireshark double clicking on the GUI or executing the cli command

```
wireshark -r snort/dumps/sqlinj.pcap
```

To write the required rule we need to use the Snort options *http_uri:*, *content: “”* and *pcre: “”*. The first allow Snort to match the rule against only one part of the HTTP uri, the second is used to check the content of the specified field and the third is used to write a regex that the content need to match for the rule to fire.

We can now write the following rule

```
alert http any any -> any 8080
(
   http_uri:query;
   content:"search=",nocase;
   pcre:"/(.*[\"\']\;)+.*/";
   sid:8;
   msg:"SQLi command in search parameter"
)
```

inside the exercise08.rules file.

We can now run Snort with the command

```
snort3_pcap -c snort/confs/snort_ids.lua -r snort/dumps/sqlinj.pcap -R
    snort/rules/exercise08.rules
```

we can see that the Snort logs all the requests that match the regex as SQL injections.

3.9 Exercise 9

The exercise 9 is made to demonstrate the port scan module of Snort. This module allows to check for port scans and write alerts in such cases.

To start Snort in its port scan mode we can write the command

```
snort3_portscan
```

To test the module, we can perform various scans with nmap from client 101 to 102 as shown below

```
# TCP SYN port scan
nmap -nsS 192.168.77.102
# TCP connect port scan
nmap -nsT 192.168.77.102
# UDP port scan
nmap -nsU 192.168.77.102
```

3.10 Exercise 10

The exercise 10 is made to demonstrate a real life scenario. We've prepared a .pcap file of the infamous QakBot malware. This exercise aims at showing the true potential of Snort in a complex environment where detecting new malicious activity can be straightforward like writing some easy Snort rules instead of changing the configuration of multiple firewalls.

First we need to unzip the QakBot directory double clicking on the archive from the GUI interface or with the cli command

```
unzip snort/dumps/Qakbot.zip -d snort/dumps/Qakbot
```

and the password "**infected**".

First of all we can examine the `2023-05-24-obama264-Qakbot-notes.txt` where is store all the IoC detected until 2023-05-24 and after we can have a look directly to the `2023-05-24-obama264-Qakbot-infection.pcap` file with Wireshark.

Since the rules for such a malware may be very complex, we've prepared some of them inside the *secret* folder `/opt/snort_rules` where all the correct rules are stored as follows: (notice the use of *sid* starting from one million to exclude conflicts with built-in and community rules)

```
alert file
(
   file_type: ZIP;
   flow:to_client,established;
   sid: 1000001;
   msg: "Alert on ZIP file"
)

alert file
(
   file_type: MSEXE;
   flow:to_client,established;
   sid: 1000002;
   msg: "Alert on DAT file"
)

alert udp any any -> any 53
(
   content:"|0d 61 64 75 62 75 69 6c 64 65 72 73 63 6f 03 63 6f 6d 00|";
   sid: 1000003;
   msg:"DNS request for 'adubuildersco.com' domain"
)

alert tcp any any -> 142.118.221.248 2222
(
   flow:to_server,established;
   sid: 1000004;
   msg:"Connection to Qakbot C2"
)

alert tcp any any -> 201.130.154.90 443
(
   flow:to_server,established;
   sid: 1000005;
   msg:"Connection to secondary Qakbot C2"
)

alert http any any -> 160.153.53.37 80
(
   flow:to_server,established;
   http_method; content:"GET",fast_pattern;
   http_uri; content:"/ewukhyqpjz.zip",fast_pattern,nocase;
   sid: 1000006;
   msg:"Download of malicious ZIP GET /ewukhyqpjz/ewukhyqpjz.zip"
)

alert http any any -> 45.76.58.72 80
(
   flow:to_server,established;
   http_method; content:"GET",fast_pattern;
   http_uri;content:"/aKUVYL8o0uv.dat",fast_pattern,nocase;
```

```
   sid: 1000007;
   msg:"Download of malicious DAT GET /aKUVYL8o0uv.dat"
)
```

Copy and paste the interesting rules that you want to test in the exercise10.rules file. We can now run Snort with the command

```
snort3_qakbot -r
    snort/dumps/Qakbot/2023-05-24-obama264-Qakbot-infection.pcap -R
    snort/rules/exercise10.rules
```

We can see that Snort logs the malicious traffic filtering the information present in the rules file in a file called `alert_fast.txt` create in the same directory where is executed the previous command.

4 IPS Exercises

Before going into the Intrusion Prevention System exercises we need to run the command

```
snort/enableNFQ.sh
```

to enable the IPS capabilities for Snort.

4.1 Exercise 11

For the exercise 11 the goal is to write a Snort rule to detect and block ping from any ip address to any ip address.

Before we write the rule run the command

```
tcpdump -ni eth0
```

on client 103 to check the communications.

Now if we run the command

```
ping 192.168.88.103
```

on client 101 or client 102 we can see that the ping requests and replies go back and forth.

We can now write the following rule

```
block icmp any any -> any any
(
   sid:11
)
```

inside the exercise11.rules file.

We can now run Snort with the command

```
snort3_ips -R snort/rules/exercise11.rules
```

Now we can restart to listen with the command

```
tcpdump -ni eth0
```

on client 103.

Now if we run the command

```
ping 192.168.88.103
```

on client 101 or client 102 we can see that Snort logs the ECHO requests and ECHO replies and client 103 receives no packet since Snort is blocking them.

4.2 Exercise 12

For the exercise 12 the goal is to write a Snort rule to detect and block TCP packets with ACK flag enabled sent from client 101 to client 103.

Before we write the rule run the command

```
nc -lp 666
```

on client 103 to listen to communications.

Now if we run the command

```
nc 192.168.88.103 666
```

on client 101 we can see that the connection is successful and if we send messages client 103 receives and displays them.

We can now write the following rule

```
block tcp 192.168.77.101 any -> 192.168.88.103 any
(
   flags:*A;
   sid:12;
   msg:"Blocked an ACK packet from client 101 to client 103"
)
```

inside the exercise12.rules file.

We can now run Snort with the command

```
snort3_ips -R snort/rules/exercise12.rules
```

Now we can restart to listen with the command

```
nc -lp 666
```

on client 103.

Now if we run the command

```
nc 192.168.88.103 666
```

on client 101 we can see that Snort logs the requests and client 103 receives no packet since Snort is blocking them.

4.3 Exercise 13

For the exercise 13 the goal is to write a Snort rule to detect and drop DNS requests coming from the home network for *unitn.it* domains to any DNS server but the company one.

Before we write the rule run the commands

```
nslookup google.com 192.168.88.103
nslookup google.com 1.1.1.1
nslookup unitn.it 192.168.88.103
nslookup unitn.it 1.1.1.1
```

on client 101 or client 102. We can see that no command is blocked and that every request is accepted.

We can now write the following rule

```
drop udp $HOME_NET any -> !$DNS_SERVER 53
(
   content:"|05 75 6e 69 74 6e 02 69 74 00|";
   sid:13;
   msg:"Blocked DNS request to a not standard DNS server"
)
```

inside the exercise13.rules file.

We can now run Snort with the command

```
snort3_ips -R snort/rules/exercise13.rules
```

Now if we run the commands

```
nslookup google.com 192.168.88.103
nslookup google.com 1.1.1.1
nslookup unitn.it 192.168.88.103
nslookup unitn.it 1.1.1.1
```

on client 101 or client 102 we can see that only the last one is logged and dropped by Snort since we are trying to resolve unitn.it at a DNS server that is not allowed.

4.4 Exercise 14

For the exercise 14 the goal is to write a Snort rule to detect and block SQL injections that contains the keywords *DROP* or *INSERT*. For this exercise we will use the same .pcap file that we've used in the exercise 8.

Before we write the rule we can inspect the .pcap file with the commands

```
cat snort/dumps/sqlinj.pcap
```

We can now write the following rule

```
block http any any -> any 8080
(
   http_uri:query;
   content:"search=",nocase;
   pcre:"/(.*[\"\']\;)+.*(DROP|INSERT)+/";
   sid:14;
   msg:"Blocked SQL command in search parameter"
)
```

inside the exercise14.rules file.

We can now run Snort with the command

```
snort3_pcap -c snort/confs/snort_ids.lua -r snort/dumps/sqlinj.pcap -R
    snort/rules/exercise14.rules
```

we can see that Snort logs the requests that match the regex as SQL injections.

References

[1] *How rules are improving in Snort 3*. Snort.org. Aug. 12, 2020.
URL: https://blog.snort.org/2020/08/how-rules-are-improving-in-snort-3.html (visited on 06/04/2023) (cit. on p. 3).

[2] *Improve Snort 3 performance with Hyperscan*. Snort.org. Sept. 21, 2020.
URL: https://blog.snort.org/2020/09/snort-3-hyperscan-.html (visited on 06/04/2023) (cit. on p. 3).

[3] *Snort 3 Rule Writing Guide*. Snort.org.
URL: https://docs.snort.org (visited on 06/04/2023) (cit. on p. 11).