Practical Assignment 2

Information Tecnology Security

Alexy de Almeida	2019192123	adenis@student.dei.uc.pt
Rodrigo Ferreira	2019220060	rferreira@student.dei.uc.pt

PL3

April 2023

Contents

1 Introduction 2
2 Experimental scenario 2
3 Firewall Configurations 3
3.1 Configuration to protect the router 3
3.2 Configuration to authorize direct communications (without NAT) 4
3.3 Configuration for connections to the external IP address of the firewall (using NAT) . 5
3.4 Configuration for communications from the Internal network to the Internet (using NAT) 5
4 Intrusion detection and prevention (IDS/IPS) 7
4.1 Snort installation 7
4.2 Testing connections 8
4.3 Snort configuration 9
5 Conclusion 11

1 Introduction

The aim of this project is to configure a network firewall using IPTables/Netfilter and configure Snort as an IDS/IPS system, so that it can function as an intrusion detection and prevention/reaction tool. To do so, we'll first start by defining our experimental scenario, configure the rules for our firewall to allow and forbid specific packets traffic and then we shall configure Snort to be able to detect two types of SQL injections, two types of DoS attacks and OS fingerprinting attempts.

2 Experimental scenario

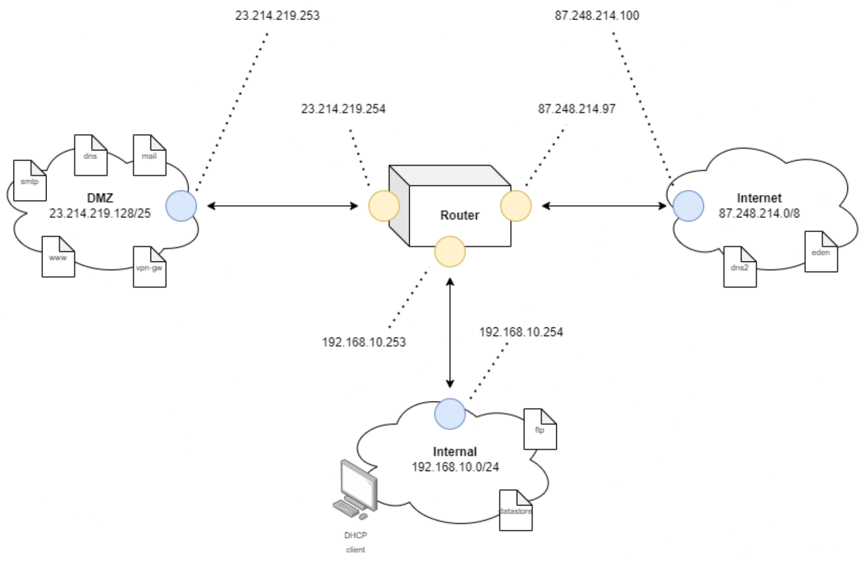


Figure 1: Experimental Scenario

For this assignment, there are 3 networks that will be used:

- [1] **DMZ**: 23.214.219.128/25
- [2] **Internal**: 192.168.10.0/24
- [3] **Internet**: 87.0.0.0/8

In order to be able to test our scenario, 4 CentOS7 virtual machines were used in VMWare along with the 3 networks:

- VM1: **"STI_DMZ"**, which contains the network [1] **DMZ** and is where most of the public services are placed. The services are as follows:
  - **dns**
  - **mail**
  - **vpn-gw**
  - **www**
  - **smtp**

  The interface was configured as follows:

  - 23.214.219.253 — 255.255.255.128 — 23.214.219.254 (**ens36**)

- VM2: **"STI_Internal"**, which contains the network [2] **Internal** and is the one that provides connectivity to clients with dynamic IP addresses, while also supporting servers with specific purposes. The services are as follows:
  - **ftp**
  - **datastore**

  The interface was configured as follows:
  - 192.168.10.253 — 255.255.255.0 - 192.168.10.254 (**ens36**)
- VM3: **"STI_Router"**, which interconnects the various networks, while also supporting all the security functionalities. The interfaces were configured as follows:
  - 23.214.219.254 — 255.255.255.128 (**ens36**)
  - 192.168.10.254 — 255.255.255.0 (**ens37**)
  - 87.248.214.97 — 255.0.0.0 (**ens38**)
- VM4: **"STI_Internet"**, which contains the network [3] **Internet**. The services are as follows:
  - **dns2**
  - **eden**

  The interface was configured as follows:
  - 87.248.214.100 — 255.0.0.0 — 87.248.214.97 (**ens36**)

For simplicity purposes, none of the services have an IP address and we will only use the only network interface of each VM to test each connectivity.

3 Firewall Configurations

In order to test everything accordingly, the firewall configurations were created inside a bash script to make testing and running easier on **"STI_Router"**.

```
touch /root/iptables-config.sh
chmod +x /root/iptables-config.sh
nano /root/iptables-config.sh
```

For every VM machine, run the following commands:

```
systemctl stop firewalld
systemctl disable firewalld
iptables -F
```

3.1 Configuration to protect the router

The first thing the script does is disabling the default firewall, enabling packet forwarding on the router, flushing all iptables rules and drop all communications entering the router.

```
systemctl stop firewalld
systemctl disable firewalld
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -F
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```

The commands "-P INPUT DROP", "-P FORWARD DROP" and "-P OUTPUT ACCEPT" set the default policies for INPUT, FORWARD AND OUTPUT chains to drop, so that any packet that comes or goes through the router that doesn't match any rule in the INPUT, FORWARD or OUTPUT chains will be dropped. With that, the firewall will deny all connections but not the ones allowed by other rules.

The next step is optional, as it allows us to use SSH connections to the VMs from the Windows Terminal:

```
iptables -A INPUT -s 192.168.6.0/24 -j ACCEPT
```

Another necessary command to add for later on is ”modprobe nf_conntrack_ftp”, which provides connection tracking support for the FTP protocol due to the fact that FTP uses multiple connections for control and data transfers.

```
modprobe nf_conntrack_ftp
```

With this command, we’re enabling the Linux kernel to provide the proper assistance for NAT and firewall rules with FTP connections in the two modes (active and passive), which will be addressed later on.

We now want to enable communications for specific services and those can be identified in ”/etc/services”. First we want to allow the router to do dns requests via TCP and UDP, as well as allow connections to the router via SSH when they are from the [2] **Internal** network or the **vpn-gw** machine:

- DNS name resolution requests sent to outside servers:

  ```
  # TCP
  iptables -A OUTPUT -p tcp --dport domain -j ACCEPT
  iptables -A INPUT -p tcp --dport domain -j ACCEPT

  # UDP
  iptables -A OUTPUT -p udp --dport domain -j ACCEPT
  iptables -A INPUT -p udp --dport domain -j ACCEPT
  ```

  The first two commands allow DNS traffic from the router to other DNS servers and the last two ensure that incoming DNS packets are allowed. We can notice that we make use of ”–dport” to match the destination port of DNS traffic. The tag ”domain” corresponds to port 53, which is the standard port for DNS traffic. From now on, we shall use the name tags instead of the port numbers for simplicity of reading.

- SSH connections to the router system, if originated at the [2] **Internal** network or at the VPN gateway (vpn-gw), i.e, allow connections to the router via SSH when originated at the [2] **Internal** network or at **vpn-gw** machine.

  ```
  # Internal network
  iptables -A INPUT -s 192.168.10.0/24 -p tcp --dport ssh -j ACCEPT

  # vpn-gw
  iptables -A INPUT -s 23.214.219.253 -p tcp --dport ssh -j ACCEPT
  ```

3.2 Configuration to authorize direct communications (without NAT)

Here, the goal of the firewall configuration it to drop all communications between networks, while also allowing forwarding between networks without NAT for the following services:

- DNS

  ```
  # Domain name resolutions using the dns server
  iptables -A FORWARD -p udp -d 23.214.219.253 --dport domain -j ACCEPT
  iptables -A FORWARD -p udp -s 23.214.219.253 --dport domain -j ACCEPT

  # The dns server should be able to resolve names using DNS servers on the Internet (dns2 and
      also others).
  iptables -A FORWARD -p tcp -d 23.214.219.253 --dport domain -j ACCEPT
  iptables -A FORWARD -p tcp -s 23.214.219.253 --dport domain -j ACCEPT
  ```

  The **dns** and **dns2** servers should be able to syncrhonize the contents of DNS zones. All of that is done in the code above by using TCP ports, because contrary to UDP ports, TCP allows to synchronize information between DNS servers.

- SMTP connections to the **smtp** server

  ```
  iptables -A FORWARD -p tcp -d 23.214.219.253 --dport smtp -j ACCEPT
  ```

- POP and IMAP connections to the **mail** server

```
iptables -A FORWARD -p tcp -d 23.214.219.253 --dport pop3 -j ACCEPT
iptables -A FORWARD -p tcp -d 23.214.219.253 --dport imap -j ACCEPT
```

- HTTP and HTTPS connections to the **www** server

```
iptables -A FORWARD -p tcp -d 23.214.219.253 --dport http -j NFQUEUE --queue-num 0
iptables -A FORWARD -p tcp -d 23.214.219.253 --dport https -j NFQUEUE --queue-num 0
```

  We shall see later on that there needs to be additional security provided by Snort to some services, hence the need of using "NFQUEUE –queue-num0". These two rules forward TCP packets destined to the **www** server and ports 80 (http) or 443 (https) to a specific queue (number 0), which enables custom processing or filtering. By that, we mean to detect attacks coming from the outside network that go through the router.

- OpenVPN connections to the **vpn-gw** server

```
iptables -A FORWARD -p tcp -d 23.214.219.253 --dport openvpn -j ACCEPT
```

- VPN clients connected to the gateway should be able to connect to all services in the [2] **Internal** network

```
iptables -A FORWARD -s 23.214.219.253 -d 192.168.10.0/24 -j ACCEPT
iptables -A FORWARD -s 192.168.10.0/24 -d 23.214.219.253 -j ACCEPT
```

3.3 Configuration for connections to the external IP address of the firewall (using NAT)

The connections originated on the outside ([3] **Internet**) and destined to the external IP address of the **"STI_Router"** firewall should be authorized and treated according to the following requirements:

- FTP connections (in passive and active modes) to the **ftp** server:

```
iptables -t nat -A PREROUTING -p tcp -d 87.248.214.97 --dport ftp -j DNAT --to-destination
    192.168.10.253
iptables -t nat -A PREROUTING -p tcp -d 87.248.214.97 --dport ftp-data -j DNAT --to-destination
     192.168.10.253
iptables -A FORWARD -p tcp -d 192.168.10.253 --dport ftp -j ACCEPT
iptables -A FORWARD -p tcp -d 192.168.10.253 --dport ftp-data -j ACCEPT
```

Since the goal is to use NAT, we need to use "-t nat" to specify the nat table. "-A PREROUTING" is used to process packets before routing.

- SSH connections to the **datastore** server, but only if originated at the **eden** or **dns2** servers:

```
iptables -t nat -A PREROUTING -d 87.248.214.97 -s 87.248.214.100 -p tcp --dport ssh -j DNAT --
    to-destination 192.168.10.253
iptables -A FORWARD -d 192.168.10.253 -s 87.248.214.100 -p tcp --dport ssh -j ACCEPT
```

With this, machines on the [3] **Internet** can access the FTP service of the **ftp** server. Additionally, the **eden** and **dns2** servers can access the SSH service of the **datastore** server. Both the **ftp** and **datastore** servers belong to the [2] **Internal** network, hence why we set **destination port** "-d" to 87.248.214.97, **destination ip** "–to-destination" to 192.168.10.1 (**ftp**) and 192.168.10.2 (**datastore**, and **source IP addresses** "-s" to 87.248.214.98 (**dns2**) and 87.248.214.99 (**eden**).

3.4 Configuration for communications from the Internal network to the Internet (using NAT)

The goal is to allow three types of communication from the [2] **Internal** network to the [3] **Internet** using NAT. The communications are as follows:

- Domain name resolutions using DNS

```
    iptables -t nat -A POSTROUTING -p udp -s 192.168.10.0/24 --dport domain -j SNAT --to-source
        87.248.214.100
    iptables -t nat -A POSTROUTING -p tcp -s 192.168.10.0/24 --dport domain -j SNAT --to-source
        87.248.214.100
    iptables -A FORWARD -p udp -s 192.168.10.0/24 --dport domain -j ACCEPT
    iptables -A FORWARD -p tcp -s 192.168.10.0/24 --dport domain -j ACCEPT
```

With this, computers in the [2] **Internal** network to translate domain names into IP addresses that computers on the [3] **Internet** can understand.

- HTTP, HTTPS and SSH connections

```
    # HTTP
    iptables -t nat -A POSTROUTING -p tcp -s 192.168.10.0/24 --dport http -j SNAT --to-source
        87.248.214.100
    iptables -A FORWARD -p tcp -s 192.168.10.0/24 --dport http -j ACCEPT

    # HTTPS
    iptables -t nat -A POSTROUTING -p tcp -s 192.168.10.0/24 --dport https -j SNAT --to-source
        87.248.214.100
    iptables -A FORWARD -s 192.168.10.0/24 -p tcp --dport https -j ACCEPT

    # SSH
    iptables -t nat -A POSTROUTING -p tcp -s 192.168.10.0/24 --dport ssh -j SNAT --to-source
        87.248.214.100
    iptables -A FORWARD -s 192.168.10.0/24 -p tcp --dport ssh -j ACCEPT
```

With this, computers in the [2] **Internal** can access websites and connect in a safe way to other computers on the [3] **Internet**.

- FTP connections (in passive and active modes) to external FTP servers

```
    iptables -t nat -A POSTROUTING -p tcp -s 192.168.10.0/24 --dport ftp -j SNAT --to-source
        87.248.214.100
    iptables -t nat -A POSTROUTING -p tcp -s 192.168.10.0/24 --dport ftp-data -j SNAT --to-source
        87.248.214.100
    iptables -A FORWARD -p tcp -s 192.168.10.0/24 --dport ftp -j ACCEPT
    iptables -A FORWARD -p tcp -s 192.168.10.0/24 --dport ftp-data -j ACCEPT
```

With this, computers in the [2] **Internal** network can transfer files to FTP servers on the [3] **Internet**, as well as receive files from there. But since some connections can be established in different ports, which is the case for passive FTP, since it uses

```
    iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
    iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
    iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

We can observe that we make use of the tag ”-m state”, which is used for iptables to check the state of the network connection, as well as the tag ”–state ESTABLISHED,RELATED” to tell that we want connections that are already established or that are related to an established connection. We need to do that for incoming (INPUT), outgoing (OUTPUT) and forwarded (FORWARD) packets.

One final thing we can add to the script is to print all the rules we established until now:

```
iptables -L
```

Finally, we can run the script:

```
/root/iptables-config.sh
```

This is the obtained result of the policies we set:

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  192.168.6.0/24       anywhere
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:domain
ACCEPT     udp  --  anywhere             anywhere             udp dpt:domain
ACCEPT     tcp  --  192.168.10.0/24      anywhere             tcp dpt:ssh
ACCEPT     tcp  --  gateway              anywhere             tcp dpt:ssh
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     udp  --  anywhere             gateway              udp dpt:domain
ACCEPT     udp  --  gateway              anywhere             udp dpt:domain
ACCEPT     tcp  --  anywhere             gateway              tcp dpt:domain
ACCEPT     tcp  --  gateway              anywhere             tcp dpt:domain
ACCEPT     tcp  --  anywhere             gateway              tcp dpt:smtp
ACCEPT     tcp  --  anywhere             gateway              tcp dpt:pop3
ACCEPT     tcp  --  anywhere             gateway              tcp dpt:imap
NFQUEUE    tcp  --  anywhere             gateway              tcp dpt:http NFQUEUE num 0
NFQUEUE    tcp  --  anywhere             gateway              tcp dpt:https NFQUEUE num 0
ACCEPT     tcp  --  anywhere             gateway              tcp dpt:openvpn
ACCEPT     all  --  gateway              192.168.10.0/24
ACCEPT     all  --  192.168.10.0/24      gateway
ACCEPT     tcp  --  anywhere             gateway              tcp dpt:ftp
ACCEPT     tcp  --  anywhere             gateway              tcp dpt:ftp-data
ACCEPT     tcp  --  gateway              gateway              tcp dpt:ssh
ACCEPT     udp  --  192.168.10.0/24      anywhere             udp dpt:domain
ACCEPT     tcp  --  192.168.10.0/24      anywhere             tcp dpt:domain
ACCEPT     tcp  --  192.168.10.0/24      anywhere             tcp dpt:http
ACCEPT     tcp  --  192.168.10.0/24      anywhere             tcp dpt:https
ACCEPT     tcp  --  192.168.10.0/24      anywhere             tcp dpt:ssh
ACCEPT     tcp  --  192.168.10.0/24      anywhere             tcp dpt:ftp
ACCEPT     tcp  --  192.168.10.0/24      anywhere             tcp dpt:ftp-data
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:domain
ACCEPT     udp  --  anywhere             anywhere             udp dpt:domain
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
```

4 Intrusion detection and prevention (IDS/IPS)

In the last goal of the assignment we were asked to enable, in the firewall system, the capability to detect and react to security attacks. Those attacks may be originated on the [3] **Internet** and when an attack is successfully detected the **"STI_Router"** firewall should block it. Intrusion detection and prevention was implemented in the firewall considering the following requirements:

- The firewall should be able to detect and block attacks, using Snort and IPtables;
- Detect and block (at least) the following attacks/scans:
  - Two types of SQL injection;
  - Two types of DoS (Denial of Service) attacks;
  - OS fingerprinting attempts.

4.1 Snort installation

The very first step is to install Snort on **"STI_Router"**, since this VM will be the one to work as a firewall. The installation process may be different depending on the CentOS version, but since we're using CentOS7 those are the steps to follow:

```
# update is important to get most recent packages
yum update
yum -y install epel-release

# Install required packages
yum install libpcap-devel pcre-devel libdnet-devel zlib-devel libnetfilter_queue libnetfilter_queue-devel
     gcc make perl luajit-devel openssl openssl-devel libnghttp2-devel bison flex

# Download required sources for libdaq and Snort 2
cd /usr/local/src
wget https://www.snort.org/downloads/snort/daq-2.0.7.tar.gz
wget https://www.snort.org/downloads/snort/snort-2.9.20.tar.gz

# Compile and install libdaq with support for the nfq DAQ
tar zxvf daq-2.0.7.tar.gz
cd daq-2.0.7

# As the result of the following command we should make sure that the nfq module is enabled for
    compilation
```

```
./configure --enable-nfq

# Compile and install the DAQ
make
make install

# Enable the usage of the compiled modules
nano /etc/ld.so.conf
        # add this line
        include /usr/local/lib/daq

# Update the shared libraries
ldconfig

# Configure Snort for compilation
cd /usr/local/src
tar zxvf snort-2.9.20.tar.gz
cd snort-2.9.20
./configure --with-daq-includes=/usr/local/lib --with-daq-libraries=/usr/local/lib --prefix=/usr/local/
    snort --disable-open-appid

# Compile and install Snort
make
make install
ln -s /usr/local/snort/bin/snort /usr/sbin/snort
```

4.2 Testing connections

By looking at the list of services on /etc/services, we can use each service number to test every connectivity. This list of services to test is as follows:

- FTP: 20 and 21
- SSH: 22
- SMTP: 25
- DNS: 53
- HTTP: 80
- POP: 110
- IMAP: 143
- HTTPS: 443
- OpenVPN: 1149

To be able to test the attacks, we must install some packages on the [2] **Internal** network, we must do the following on **STI_Internal**:

```
yum install vsftpd
yum install hping3
yum install nmap
systemctl enable vsftpd
systemctl start vsftpd
```

Now, every test must be made as follows:

```
# To test DNS, SMTP, POP, IMAP, OpenVPN

[destination machine] nc -l -p <dest_port> -v
[source machine] echo "test" | nc <dest_IP> <dest_port>

# To test HTTP and HTTPS
[destination machine] nc -l -u -p <dest_port> -v
[source machine] echo "test" | nc -u <dest_IP> <dest_port>

# To test SSH

[source machine] ssh root@<dest_IP>
```

```
# To test FTP

[destination machine] nc -l -p <dest_port>
[source machine] ftp <dest_IP>
```

The following tests were then performed:

Service	**Port**	NAT	**Source**	**Destination**	**Expected**	**Result**
DNS	53	no	Internal	Internet	Internet receives a packet	Success
DNS	53	no	DMZ	Internet	Internet receives a packet	Success
SSH	22	no	Internet	Router	Internet host connects to Router	Success
SSH	22	no	DMZ	Router	DMZ host connects to Router	Success
SMTP	25	no	Internal	DMZ	DMZ receives a packet	Success
POP	110	no	Internal	DMZ	DMZ receives a packet	Success
IMAP	143	no	Internal	DMZ	DMZ receives a packet	Success
HTTP	80	no	Internal	DMZ	DMZ receives a packet	Success
HTTPS	443	no	Internal	DMZ	DMZ receives a packet	Success
OpenVPN	1194	no	Internal	DMZ	DMZ receives a packet	Success
VPN client	1194	no	Internal	DMZ	DMZ receives a packet	Success
FTP	20/21	yes	DMZ	Internal	Internal receives a packet	Success
SSH	20/21	yes	Internet	Internal	Internet host connects to Internal	Success
DNS	53	yes	Internal	DMZ	DMZ receives a packet	Success
HTTP	80	yes	Internal	Internet	Internet receives a packet	Success
HTTPS	443	yes	Internal	Internet	Internet receives a packet	Success
SSH	22	yes	Internal	Internet	Internal host connects to Internet	Success
FTP	20/21	yes	Internal	Internet	Internet receives a packet	Success

4.3 Snort configuration

The next step is to configure Snort as an IPS, by doing the following:

```
# Create a Snort configuration directory to make things simpler
mkdir /etc/snort
cp /usr/local/src/snort-2.9.20/etc/* /etc/snort

# Configure Snort for inline mode
nano /etc/snort/snort.conf
        config daq: nfq
        config daq_mode: inline
        config daq_var: queue=0
        var RULE_PATH /etc/snort/rules
        var BLACK_LIST_PATH /etc/snort/rules
          var WHITE_LIST_PATH /etc/snort/rules

        dynamicpreprocessor directory /usr/local/src/snort-2.9.20/src/dynamic-preprocessors/build/usr/
            local/snort/lib/snort_dynamicpreprocessor
        dynamicengine directory /usr/local/snort/lib/snort_dynamicengine/libsf_engine.so

        # comment the following lines:
        # dynamicdetection directory
        # all the rules

mkdir /var/log/snort
touch /etc/snort/rules/white_list.rules
touch /etc/snort/rules/black_list.rules
```

This allows us to use Snort inline mode. Next, we download the rules from the Snort community. On **"STI_Router"**, we do:

```
cd /etc/snort
wget https://www.snort.org/downloads/community/community-rules.tar.gz
mkdir /etc/snort/rules
tar -xzf community-rules.tar.gz -C rules
```

The idea here is to protect both the networks [1] **DMZ** and [2] **Internal** from possible attacks coming from anywhere.

Since we want two rules for SQL injection, two rules for DoS rules and one rule for OS fingerprinting, we can create three different files. Every file follows a similar structure at the beginning:

```
alert tcp <external network addresses attacks> <source port to monitor> -> <addresses to protect> <
    destination port to monitor>
```

As we want to protect the [1] **DMZ** and [2] **Internal** networks from any attack coming from the exterior of those networks, "external network addresses attacks", "source port to monitor" and "destination port to monitor" will be set to "any", and the "ip addresses to protect" will be set to [23.214.219.128/25, 192.168.10.0/24].

```
cd /etc/snort/rules

nano sql.rules

        drop tcp any any -> [23.214.219.128/25,192.168.10.0/24] any (msg:"SQL Injection - Simple UNION";
            content:"union"; nocase; classtype:web-application-attack; sid:1000001; rev:1; tag:session,
            1, packets;)

        drop tcp any any -> [23.214.219.128/25,192.168.10.0/24] any (msg:"SQL Injection - DROP TABLE
            Attempt"; flow:established,to_server; content:"DROP"; nocase; content:"TABLE"; nocase;
            classtype:web-application-attack; sid:1000002; rev:1; tag:session, 1, packets;)


nano dos.rules

        alert tcp any any -> [23.214.219.128/25,192.168.10.0/24] any (msg:"DoS - SYN flood detected";
            flags:S; flow:stateless; threshold:type both, track by_src, count 100, seconds 1; classtype:
            attempted-dos; sid:1000003; rev:1;)

        alert udp any any -> [23.214.219.128/25,192.168.10.0/24] any (msg:"DoS - UDP flood detected";
            flow:stateless; threshold:type both, track by_src, count 100, seconds 1; classtype:attempted
            -dos; sid:1000004; rev:1;)

nano os.rules

        alert tcp any any -> [23.214.219.128/25,192.168.10.0/24] any (msg:"Nmap Xmas scan detected";
            flags:FPU; reference:url,nmap.org; classtype:attempted-recon; sid:1000005; rev:1;)
```

Let's explain some parts of the rules:

- **alert** allows the rule to alert when its conditions are met;
- **tcp** is the protocol to be used to detect the attacks, as such, if we use send an attack with udp, it will not be detected;
- **content:"union"** is what we're trying to detect in the attacks, so for example, if the attack contains the message "union" it will be detected by Snort as a menace;
- **flag:S** indicates that we're looking for TCP packets with SYN flag;
- **flow:stateless** allows Snort to no track the state of the connection and, as such, it can detect SYN floods, which are composed of multiple connection attempts;
- **count 100** is the maximum number of packets coming from the same IP address before Snort considers it an attack;
- other parts such as **classtype:**, **sid:**, **rev:** and **tag:** are options for categorization and logging.

We now want to include those rules in the snort.conf file:

```
nano /etc/snort/snort.conf
        include $RULE_PATH/sql.rules
        include $RULE_PATH/dos.rules
        include $RULE_PATH/os.rules
```

On **"STI_Router"**, we can try to detect those attacks by using Snort:

```
snort -Q --daq nfq --daq-var queue=0 -c /etc/snort/snort.conf -A console -l /var/log/snort/
```

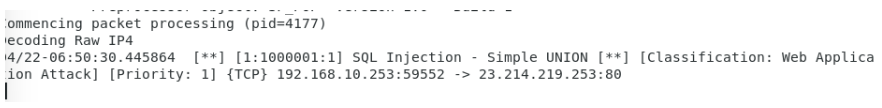
We can send SQL injections from the **"STI_Internal"** to the [1] **DMZ** network by doing the following::

```
[STI_Internal] echo "union" | nc -t 23.214.219.253 443
[STI_DMZ] nc -l -t -p 443 -v
```

Let's revisit these two commands:

```
iptables -A FORWARD -p tcp -d 23.214.219.253 --dport http -j NFQUEUE --queue-num 0
iptables -A FORWARD -p tcp -d 23.214.219.253 --dport https -j NFQUEUE --queue-num 0
```

Here we allow detection of potential attacks to networks on ports 80 and 443 by redirecting packets to a queue numbered 0. Snort will detect and alert those potential attacks, but only for the ports 80 and 443:



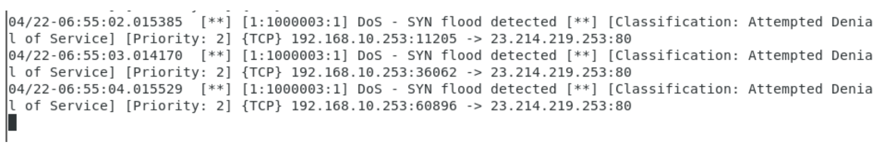
```
ommencing packet processing (pid=4177)
ecoding Raw IP4
4/22-06:50:30.445864  [**] [1:1000001:1] SQL Injection - Simple UNION [**] [Classification: Web Applica
ion Attack] [Priority: 1] {TCP} 192.168.10.253:59552 -> 23.214.219.253:80
```

Figure 2: SQL injection detected by Snort

We can send DoS attacks from the **"STI_Internal"** to the [1] **DMZ** network by doing the following and the same happens for the same ports:

```
# infinite
hping3 -S --flood -p 80 23.214.219.253

# only 5 packets, but Snort won't detect since it's below 100
hping3 -S -p 80 23.214.219.253 -c 5
```



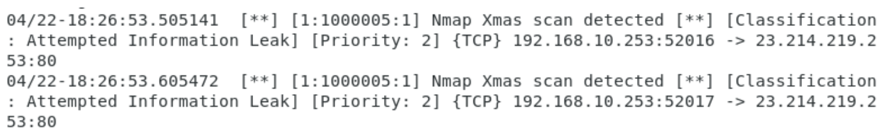
```
04/22-06:55:02.015385  [**] [1:1000003:1] DoS - SYN flood detected [**] [Classification: Attempted Denia
l of Service] [Priority: 2] {TCP} 192.168.10.253:11205 -> 23.214.219.253:80
04/22-06:55:03.014170  [**] [1:1000003:1] DoS - SYN flood detected [**] [Classification: Attempted Denia
l of Service] [Priority: 2] {TCP} 192.168.10.253:36062 -> 23.214.219.253:80
04/22-06:55:04.015529  [**] [1:1000003:1] DoS - SYN flood detected [**] [Classification: Attempted Denia
l of Service] [Priority: 2] {TCP} 192.168.10.253:60896 -> 23.214.219.253:80
```

Figure 3: DoS attack detected by Snort

We can send OS Fingerprinting from the **"STI_Internal"** to the [1] **DMZ** network by doing the following:

```
nmap -sX --system-dns -p 80 23.214.219.253
```

In a **Xmas scan** we can send TCP packets with the FIN, PSH, and URG flags set and we can see below the result:



```
04/22-18:26:53.505141  [**] [1:1000005:1] Nmap Xmas scan detected [**] [Classification
: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.253:52016 -> 23.214.219.2
53:80
04/22-18:26:53.605472  [**] [1:1000005:1] Nmap Xmas scan detected [**] [Classification
: Attempted Information Leak] [Priority: 2] {TCP} 192.168.10.253:52017 -> 23.214.219.2
53:80
```

Figure 4: OS Fingerprint attempt

5 Conclusion

We successfully configured the rules of our firewall, as we were able to test the right connections between each network and we were able to use Snort to detect simple attacks from one network to the other in specific ports.