

## Lab Assignment 8

### Intrusion Detection using SNORT & Firewall

#### 1. Objective

##### 1.1 Snort

For the intrusion detection assignment, we will use snort. Snort is a network monitoring program written by Marty Roesch and collaborators. Snort is commonly used as network intrusion prevention system (NIPS) and network intrusion detection system (NIDS). It can perform traffic analysis and logging of IP packets, and it can be used to detect probes and attacks on networks.

There are several third-party tools that interface with snort; you do not have to use them at this time. Snort can run in three different “modes”: sniffer, packet logger, and intrusion detection. For this lab, we’re mainly interested in the intrusion detection part. Common snort deployments would be at the “edge” of a network, integrated with a database. For simplicity’s sake, we will not do these things either.

Not all packets entering and leaving a network are necessarily of interest to us -- this is largely a matter of organizational policies and good security practices. Therefore you must configure snort to handle traffic you are interested in. Snort uses a simple description language to specify the packet types and actions. In this lab you will write snort rules to identify certain packet types.

Snort users’ manual and FAQ are suggested reading for this lab.

<http://manual.snort.org/>

<http://www.snort.org/faq/>

##### 1.2 Firewall

In this lab, we will learn about firewalls. We will use iptables to configure firewall rules which enforce various traffic restriction policies. You will be setting up a host-based firewall on both of your servers. You will need to refer to the relevant documentation to accomplish the lab tasks.

#### 2. Lab Environment

##### 2.1 Snort

Use your server VM for this lab. Use “aptitude” or “apt-get” to install snort package and its dependencies. During installation, you will be asked to configure snort with the address range for the local network. You might have noticed that it is 192.168.122.0/24 for your VM network.

##### 2.2 Firewall

The Netfilter project (<http://www.netfilter.org/>) is the packet filtering framework in Linux kernel. The userspace program iptables is commonly used to control and manipulate this framework. There are many different frontends to manage netfilter rules. However, it’s best to become familiar with the

structure of the actual rules -- they will be fundamental to any system, regardless of the frontend used to configure them.

There are useful tutorials and documentation to learn the syntax of these rules.

- <https://help.ubuntu.com/community/IptablesHowTo>
- <http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html>
- iptables man page: use the command `man iptables`.

## Brief Overview

Iptables groups rules by “chains”. There are three chains by default:

- INPUT: packets destined for your server will follow the rules in this chain
- OUTPUT: packets sourced from your server will follow the rules in this chain
- FORWARD: packets not to nor from your server will follow the rules in this chain

Since you will be setting up a host-based firewall, you will only be working with the INPUT and OUTPUT chains. The FORWARD chain is used if traffic for other hosts is routed through yours and you want to block/filter/modify/log it.

Rules applied to packets on a first-match basis, so the order in which you enter rules is important. For example, if you want to allow http from everywhere except for a few networks, you need to have the deny rule before the allow rules. Otherwise, a request that should be blocked will match the allow rule and be accepted before being checked against the deny rule.

## Clearing Rules

If you want to clear out all of the rules you have set, the easiest thing to do is to restart your server to make sure you don't miss any chains. You can also run the following string of commands to clear them all out:

```
iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
```

Don't configure the rules to be loaded on startup to avoid issues with conflicting rules between tasks, etc. Once the rules are cleared, default configuration will be restored. You can check the iptable rules with `iptables -L`

### 3. Lab Tasks

#### 3.1 Snort

Snort needs a configuration file to function correctly. On Ubuntu Server, the default configuration file is installed in `/etc/snort/snort.conf`. Like other Ubuntu Server services, you can start, stop or restart snort, or check its status, with these commands:

```
"sudo service snort {start|stop|restart|status}"
```

(Use one of the options (of start/stop/restart/status) at a time.)

Or you can choose to run snort as a service, and provide an alternate configuration file on the command line and invoke snort "manually":

```
"sudo snort -c <configuration file>"
```

To display the logs and alerts in fast mode, start snort manually with a conf file.

```
"sudo snort -q -A console -c /etc/snort/snort.conf -i eth0"
```

You can exit from a snort process running in the foreground using "Ctrl-C".

You might need to review log files created by snort, and they can be found under the directory `/var/log/snort`. Alert messages are usually in the file `/var/log/snort/alert`.

##### 3.1.1 Snort ".rules" files

You will need to create different rules (stored in a '.rules' file as described below) for generating alerts on different types of network activity. Here is an example:

```
"alert tcp 1.1.1.1 1024 -> 2.2.2.2 2048 (msg:"Alert Name"; sid:1;)"
```

Here this rule will generate an alert if a TCP packet is detected from source IP 1.1.1.1 and port 1024 to destination IP 2.2.2.2 and port 2048. "Alert Name" is the title of the generated alert. "sid:1" means unique rule identifier for this rule is 1. You will have to give a different identifier for each rule that you create.

If you want to generate an alert for all IP addresses or all ports you can use 'any' instead of a specific IP address or port. For example, following rule will generate alerts for all TCP messages irrespective of IP addresses and ports.

```
"alert tcp any any -> any any (msg:"Alert Name"; sid:2;)"
```

Once you have defined your rule, you need to create a text file with '.rules' extension and copy-paste your rule into that file. The snort package on Ubuntu Server comes bundled with several useful rule files under the directory "/etc/snort/rules". You can use a text editor (nano, vi, vim, emacs) to create and manipulate them. For example, you can create a "<username>.rules" file and have your rules in that file. The names of these files are not important as long as you know which one has which rule. Save the file and copy the file to your server VM's "/etc/snort/rules/" directory. Once you have transferred the rule file you need to change the configuration file to load that rule when snort starts. You can do this by modifying the "/etc/snort/snort.conf" file. Save a copy of this file, so that you can restore it if/when you need to.

Add this line to the configuration file:

```
"include /etc/snort/rules/<your-rule-file>"
```

to the file after this paragraph:

```
#####
```

```
# Step #7: Customize your rule set
```

```
# For more information, see Snort Manual, Writing Snort Rules
```

```
#
```

```
# NOTE: All categories are enabled in this conf file
```

```
#####
```

```
To view the alerts generated, after the paragraph in snort.conf after
```

```
#####
```

```
# Step #6: Configure output plugins
```

```
# For more information, see Snort Manual, Configuring Snort - Output  
modules
```

```
#####
```

add the following line below #syslog,

```
"output alert_fast: alert.fast"
```

This will generate the errors in `"/var/log/snort/alert.fast"`.

Save `"/etc/snort/snort.conf"` and try running snort with the following command to make sure there are no errors in your rules. You might want to stop the snort service running in the background before this.

```
"sudo snort -c /etc/snort/snort.conf"
```

Once you have started snort you can view any generated alerts by using the following command, on another terminal:

```
"sudo tail -f /var/log/snort/alert.fast"
```

If have configured the rules correctly you should see alerts being generated. If you want to disable a particular rule put the `#` character in `"/etc/snort/snort.conf"` before the corresponding `'include'` line. For example, for the rule:

`"include /etc/snort/rules/all_tcp.rules"` in `"/etc/snort/snort.conf"`, you could change the line to `"# include /etc/snort/rules/all_tcp.rules"`. You might want to remove or comment out all lines that include the bundled .rules files. Although those are useful, we disable them for now so that we can deal with less noisy alert messages.

### **3.1.2 Scanning rules**

If you want to generate an alert for port scanning you can add the `"flags:"` options in the rule.

```
"alert tcp 1.1.1.1 1024 -> 2.2.2.2 2048 ( flags: F; msg:"Syn Alert";sid:3; )"
```

This will generate an alert if there is a TCP packet from 1.1.1.1:1024 to 2.2.2.2:2048 with the FIN bit set on the packet. If you want to check for the SYN bit you would use S instead of F in the flags.

Note: Alerts are just one of the possible rules. Snort user manual elaborates other kinds of possible rules that you can use.

### **3.1.3 Configure snort to use the following rules**

Sometimes you don't want to block packets to/from your server using a firewall, but you need a way to know which IP address(es) are attempting to establish certain connections. You want to know when someone accesses the server using SSH, for example. If someone tries to scan for open ports on your machine, that might be a concern for you too. Snort can generate such alerts for suspicious activity on network. Write rules to generate alerts for incoming connections any open or closed ports.

1. One of your employees in your organization leaves and joins a competitor. You want to be sure he/she is not trying to connect to your server from his new workplace. You can generate an alert for incoming SSH connections from some particular IP address. Generate alerts for all incoming SSH packets to your server from your host machine.
2. You work for the government and want to know if someone from an enemy country tries to access your web server. If required you can take further actions based on the alerts. Generate alerts for incoming http packets from only your host machine.
3. Your organization has a rather liberal internet access policy for work computers, but not too liberal such that it could be abused. You don't want to block "facebook.com" entirely; but if you configure an alert for facebook.com connections then you can later find out if someone is taking advantage of the policy. Generate alerts for outgoing http/https packets to facebook.com. (Facebook uses a huge number of servers, check the IP address(s) currently used by Facebook). To get IP address of Facebook, ping [www.facebook.com](http://www.facebook.com)
4. Someone is trying to port scan computers at your company to find out what software and services they are running to eventually find a potential victim machine. To be warned of such activity, generate alerts for any incoming packet with the port ranging from 1000 to 1010 with the SYN bit set (This range is set to only 10 ports so that you don't generate millions of alerts). To scan ports from Client VM; read about hping3
  - a. <http://sickbits.net/firewall-testing-using-hping3/>
  - b. [http://0daysecurity.com/articles/hping3\\_examples.html](http://0daysecurity.com/articles/hping3_examples.html)

Perform the following steps-

1. Install hping3
2. On server VM, listen to packets from client VM. Run the command as root user "tcpdump host <client IP>"
3. To scan ports of server VM from client VM; run the command as root user "hping3 -8 <port-number-range-start>-<port-number-range-end> -S <server IP>"  
The "- 8" parameter scans the ports in the range and "- S" denotes SYN bit set
4. You should receive packets from Client VM with SYN bit set.
5. Shut tcpdump

## 3.2 Firewall

### 3.2.1 Block everything except SSH connections to the server VM

On your server VM, add the appropriate rules to allow anyone on the 192.168.122.0/24 network to establish SSH connections with your server. Set the default policy to drop packets on the INPUT and OUTPUT chains.

Run the following command on command line to add a new rule,

```
"iptables -A INPUT -p tcp -s 192.168.122.0/24 --dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT"
```

Change the default policy to drop,

```
"iptables -P INPUT DROP"
```

Check if the iptables rules are modified with command

```
"iptables -L"
```

Established connections will need to be allowed on outbound traffic for an SSH connection to succeed. You can test by attempting to connect to your server VM via SSH from your test machine. It should succeed. However, if you login to silo.soic.indiana.edu (or another machine not in 192.168.122.0/24), you should not be able to connect to your server VM. To test the loopback rule, you should be able to ping 127.0.0.1 and get responses on the server VM. Since many programs use the internal loopback interface to communicate, allow any traffic to or from the loopback interface (lo). Read about loopback interface and configure a rule for loopback interface.

### **3.2.2 Restrict traffic to your server VM**

On your server VM, create rules that match the following policies:

1. Block all inbound and outbound ICMP traffic.
2. Block all inbound and outbound UDP traffic except DNS requests and responses. You will need an established connections rule to allow DNS responses.
3. Deny outbound HTTP and HTTPS traffic to all hosts except 129.79.78.235 and 129.79.78.234  
When you test it with https, you may want to add the option "--no-check-certificate" to avoid the certificated warning/error.
4. Allow inbound HTTP traffic from your client VM except your host machine.
5. Block all inbound TCP traffic except SSH, HTTP, and HTTPS.

How to test these rules? Below are some quick checks. You might want to add some more tests of your own to make sure that your firewall rules are really bulletproof. Replace server VM with the appropriate domain name or IP address.

1. Pinging your server VM should fail. Pinging any other host from your server VM (for example, "ping indiana.edu", "ping 8.8.8.8") should fail.
2. "dig indiana.edu" should return a DNS lookup result.
3. Running "wget http://soic.indiana.edu" or "wget http://129.79.78.235" should print some raw HTML (and CSS and Javascript) on the terminal. Running "wget http://facebook.com" and "wget http://google.com" (or any other host) should fail.
4. Running "wget http://192.168.122.183" from your host machine should fail. Running "wget http://192.168.122.183" from your client VM should succeed.
5. You should be able to connect to services on ports 22, 80, and 443, and nothing else.

#### **4. Turning in your work**

##### **4.1 Snort**

Submit a compressed file (e.g. zip, tarball) of directory containing all your rules files, configuration file and the alert log files, on Canvas.

##### **4.2 Firewall**

Save output from "iptables -L" and "iptables-save" at the end of each step. Submit a txt file on Canvas.

(For example, iptables-save > /root/iptables.txt)