COMP4337/9337

Lab 5: Snort Intrusion Detection System

S1, 2019

Use eng.cse.COMP4337@unsw.edu.au for class communications

Students may use the Technical Questions Forum on Moodle to discuss this lab

Introduction

This lab is based on introduction to intrusion detection system, using snort. Snort (www.snort.org) is an open source intrusion detection/prevention system created by Martin "Marty" Roesch, founder of Source fire. It is the most widely used IDS/IPS capable of performing real-time traffic analysis and logging. It can monitor, detect and respond to various attacks by using signature, protocol and anomaly-based inspection techniques.

Objectives

Students will learn about configuring and utilising Snort IDS for protocol analysis and generating real time alerts. They are introduced to the rules language to describe traffic that Snort should analyse.

Deliverables & Marking

Compile a single pdf document by filling in the solution/answer part for all tasks (as directed) along with the screenshots. Submit this pdf file before the deadline.

Submission

Marks will be awarded by submission. We will randomly check for complete submissions. Upto 10 marks will reduced from final marks for the subject if a deliberate attempt to fake submission is found. Submission deadline is **Monday: April-15th,2019-1600 hrs**.

Tools/Software Requirement Snort, Linux based system

1. Configuration file

A new Snort installation requires very few configuration changes. Conveniently, one file has all the configuration settings at /`etc/snort/snort.conf`.

With any new changes you make in the snort.conf file, it is advisable to check if the snort.conf is correct. You can do this using:

```
snort –v -c /etc/snort/snort.conf -T
```

If you made changes in the snort.conf file correctly, it should give you a message

```
Snort successfully validated the configuration!

Snort exiting
```

2. Snort Modes

You can configure Snort to run in three modes. **(1) Inline mode**: acts as an Intrusion Prevention System (IPS) allowing drop rules to trigger (more detail to follow). **(2) Passive mode**: acts as an Intrusion Detection System (IDS). Drop rules are not loaded. **(3) Inline-Test mode**: simulates the inline mode of snort, allowing evaluation of inline behavior without affecting traffic. The drop rules will be loaded and will be triggered as a Wdrop (Would Drop) alert.

3. Snort Capture Modes

Snort can also be configured to run in three basic capture modes:

i. **Sniffer mode**: Snort reads IP packets and displays them on the console.
ii. **Logger mode**: Snort logs IP packets.
iii. **Network Intrusion Detection System (NIDS) mode**: Snort uses rulesets to inspect IP packets. When an IP packet matches the characteristics of a given rule, Snort may take one or more actions

3.1 Sniffer Mode

Run snort in the sniffer mode by typing

```
snort –v –i eth0
```

Note: You need to replace the "i eth0" with the relevant network interface you are using.
Ignore the warning "WARNING: No preprocessors configured for policy 0." It will disappear when we start using our snort.conf rules file.

View different options for snort.

```
snort - - help
```

Task 1: Find the flags that will display data-link headers and the application layer data?

Discover Flags
Give screenshot that shows the command (you use) and the headers and application layer data for a captured packet

3.2 Logger Mode

You can test Snort's logging abilities with the -l (log) switch, by typing:

```
snort -v -l /var/log/snort –i eth0
```

This runs Snort in descriptive verbose mode and logs all its findings to the specified directory.

Task 2: Run a command in snort to capture only ICMP packets. (For testing, you may have to use ping to generate some ICMP packets if your network is not busy)

Capture only ICMP
Provide screenshot that shows the command (you use) and the summary of snort packets captured

3.3 IDS Mode

IDS mode works with the help of rules. You specify the rules file to be used by the switch – c. For example,

```
snort -v –c /etc/snort/snort.conf –l /var/log/snort –i eth0
```

Snort Rules:

For Snort to detect attacks and alert you when attacks occur, Snort needs to know where its rule base is. By default, the rule base is in **`/etc/snort/rules`.** Of specific interest is the local.rules file that you would use to insert your own custom rules for Snort.

Here's the general form of a Snort rule:

```
action proto src_ip src_port direction dst_ip dst_port
(options)
```

A Snort rule can be broken down into two logical parts:

i. **Rule header**: It contains information about

   a. Action to perform
   b. Protocol that the rule applies to
   c. Source and destination addresses and netmasks
   d. Source and destination ports information

ii. **Rule options:** The rule options allow you to create a descriptive message to associate with the rule, as well as check a variety of other packet attributes by making use of Snort's extensive library of plug-ins.

When a packet comes in, its source and destination IP addresses and ports are compared to the rules in the ruleset. If any of them are applicable to the packet, then the options are compared to the packet. If all of these comparisons return a match, then the specified action is taken.

Snort provides several built-in actions that you can use when crafting your rules.

- *log* - log the packet
- *alert* - generate an alert using the selected alert method, and then log the packet
- *pass* - ignore the packet
- *drop* – block and log
- *sdrop* – silently block but do not log
- *reject* - block, log and send response (TCP reset for TCP or ICMP port unreachable for UDP)

Note the last three actions would only work in **inline mode**.

Currently protocols supported are TCP, UDP, ICMP and IP. Direction is specified by ->. The IP address and port number on the left-hand side of the direction operator is considered to be the source host and on right hand side is the destination host. <> is the bidirectional operator.

Rule options are separated by (;) and rule option keywords are separated from their arguments by a (:). *msg* rule option is a simple text string to be printed along with the alert or the log. *sid* keyword is used to identify Snort rules, custom defined rules should have a sid >= 1000000. *rev* is often used with *sid* to mark the revisions numbers.

4. Writing Rules

Your First Rule

Let's get our hands dirty with writing our own rules. Here is the first rule. It may not be a good rule, but it does a very good job of testing if Snort is working well and is able to generate alerts.

```
alert ip any any -> any any (msg: "IP Packet detected"; sid:1000002; rev:0;)
```

Add this rule in local.rules file located at /etc/snort/rules. The rule will generate an alert message for **every** captured **IP** packet. It will soon fill up your disk space if you leave it there! Following is a brief explanation of this rule:

i. The word "alert" shows that this rule will generate an alert message when the criteria are met for a captured packet. The criteria are defined by the words that follow.
ii. The "ip" part shows that this rule will be applied on all *IP* packets.
iii. The first "any" is used for source *IP* address and shows that the rule will be applied to all packets.
iv. The second "any" is used for the port number. Since port numbers are irrelevant at the *IP* layer, the rule will be applied to all packets.
v. The -> sign shows the direction of the packet.
vi. The third "any" is used for destination *IP* address and shows that the rule will be applied to all packets irrespective of destination *IP* address.
vii. The fourth "any" is used for destination port. Again, it is irrelevant because this rule is for *IP* packets and port numbers are irrelevant.
viii. The last part is the rule options and contains a message that will be logged along with the alert. Nothing else is being matched here. We assign this rule a sid of 1000002 and revision number of 0.

Task 3: Now execute Snort with the new rule in effect using the following command.

```
snort -c /etc/snort/snort.conf -l /var/log/snort -K ascii -i eth0
```

Where, -c = Configure file to use (rules file to use), -l = Directory to log, -K = Logging mode [pcap (default), ascii, none]

Examine the alert file in the /var/log/snort folder. Did you find alerts generated by your rule?

Check Alerts
Give screenshot that shows the command (you use) and the output in the alert file (/var/log/snort/alert). Also, provide justification why this rule is a bad rule

4.1 Understanding Standard Alert Output

When Snort generates an alert message, it will usually look like the following:

[**] [116:56:1] (snort_decoder): T/TCP Detected [**]

The first number is the Generator ID, this tells the user what component of Snort generated this alert. The second number is the Snort ID (sometimes referred to as Signature ID). Rule-based SIDs are written directly into the rules with the sid option. In this case, 56 represent a T/TCP event. The third number is the revision ID.

Now, try to understand the alert output generated for your first rule: [**] [1:1000002:0] IP Packet detected [**]

ICMP Rule

The you will write the next rule yourself. It should generate alerts for all captured ICMP packets only. To try out your new rule send ICMP ping packets to your first hop router.

Task 4: Write the new rule that you used.

Alert for only ICMP
Provide screenshot that shows the rule you used and the command. Also, show the alert file output.

Some more simple rules:

Protocol	**Rule**	**Description**
Telnet	**log tcp any any -> 192.168.1.0/24 23**	Logs TCP traffic coming from any IP address and any source port to this network where the destination port is 23 (telnet).
SSH Rule	**log tcp any any -> any 22 (msg: "Someone's trying to use SSH!";)**	Looks for any traffic going to the standard SSH port.
Bi-directional Traffic Rule	**log tcp any any <> any 23**	Logs all TCP traffic where the destination port is 23 (telnet).

Specifying Port Numbers in Rules

Port numbers may be specified in a number of ways:

Method	**Description**	**Example**
"any"	A wildcard value, meaning literally any port	
Static ports	indicated by a single port number	111 for portmapper, 23 for telnet, or 80 for http
Port ranges	indicated with the range operator ":"	**log udp any any -> 192.168.1.0/24 1:1024** Log UDP traffic coming from any port and destination ports ranging from 1 to 1024. **log tcp any any -> 192.168.1.0/24 :6000** Log TCP traffic from any port going to ports less than or equal to 6000. **log tcp any :1024 -> 192.168.1.0/24 500:** Log TCP traffic from privileged ports less than or equal to 1024 going to ports greater than or equal to 500.
Negation operator	"!" can be applied against any of the other rule types	to log everything except specific ports for Xwindows, you could do something like the rule

	(except "any", which would translate to none).	below: **log tcp any any -> 192.168.1.0/24 !6000:6010**

Task 5: Explain what the following rule is doing?
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 !:1024

Understanding rules
Explain what does the rule specify?

5. Rules Options Matching

With the limited number of protocols directly supported in rules (TCP, IP, ICMP, UDP), how can we match L2 or application layer protocols? Note that the options field in a rule can specify additional match criteria. A useful option is **content**, which allows you to search a packet for a sequence of characters or hexadecimal values. If you are searching for a string, you can just put it in quotes. In addition, if you want it to do a case-insensitive search, you can add **nocase**; However, if you are looking for a sequence of hexadecimal digits, you must enclose them in **|** characters. This rule will trigger when it sees the digit 0x90:

alert tcp any any -> any any (msg:"Possible exploit"; content:"|90|";)

This digit is the hexadecimal equivalent of the NOP instruction on the x86 architecture and is often seen in exploit code since it can be used to make buffer overflow exploits easier to write.

The **offset** and **depth** options can be used in conjunction with the **content** option to limit the searched portion of the data payload to a specific range of bytes. If you wanted to limit content matches for NOP instructions to between bytes 40 and 75 of the data portion of a packet, you could modify the previously shown rule to look like this:

alert tcp any any -> any any (msg:"Possible exploit"; content:"|90|"; offset:40;

depth:75";)

You can also match against packets that do not contain the specified sequence by prefixing it with a **!**. In addition, many shell code payloads can be very large compared to the normal amount of data carried in a packet sent to a particular service. You can check the size of a packet's data payload by using the **dsize** option. This option takes a number as an argument. In addition, you can specify an upper bound by using the < operator, or you can choose a lower bound by using the > operator. Upper and lower bounds can be expressed with <>. For example:

This modifies the previous rule to match only if the data payload's size is greater than 6000 bytes, in addition to the other options criteria.

alert tcp any any -> any any (msg:"Possible exploit"; content:"|90|"; offset:40;

depth:75";dsize: >6000;)

TASK 6: Write a rule to alert when a HTTP GET is detected.

Write rule to generate an alert if the HTTP request with "GET" is detected. Your rule should work with both HTTP and HTTPS. You can test by opening webpages on different servers.

Using Content Matching 404 No
Provide the rule, the command used for snort and screenshot of the alert. Also explain how it works

Snort also provides the **flags** option to check the TCP flags of a packet. This option is especially useful for detecting port scans that employ various invalid flag combinations. For example, this rule will detect when the SYN and FIN flags are set at the same time:

alert any any -> any any (flags: SF; msg: "Possible SYN FIN scan";)

Valid flags are S for SYN, F for FIN, R for RST, P for PSH, A for ACK, and U for URG. In addition, Snort lets you check the values of the two reserved flag bits. You can specify these by using either 1 or 2. You can also match packets that have no flags set by using 0. There are also several operators that the **flags** option will accept. You can prepend either a + , *, or ! to the flags, to match on all the flags plus any others, any of the flags, or only if none of the flags are set, respectively.

Task 7: Write a rule that generates an alert if it detects TCP connection attempt using SYN packets.

Alert on TCP Flags
Add the screenshots here, which shows the rule (you use). Also, identify the alert generated as a result

Task 8: Write a rule that detects a telnet session initiation. Once this session is detected, the rule should log the next 10 packets of this session.
Note: You need to telnet to any free telnet server (such as telehack.com) to check whether your rule is working or not.

Generation
Provide screenshots that shows the rule (you use). Also, identify the alert and the logged packets generated as a result of this rule.

If you want to write more complex rules, consult Snort's excellent rule documentation, which contains full descriptions and examples for each of Snort's rule options. The Snort User's Manual is available at http://www.snort.org