PUSL3131 Lab

Writing Snort Rules - Answers

Exercise 1: View a sample snort configuration file (snort.lua)

1. What is the value of the variable HOME_NET and EXTERNAL_NET? You can find the answer in "snort.lua"

```
24
25 HOME_NET = 'any'
26
27 EXTERNAL_NET = 'any'
28
```

2. Can you spot the rules section?

```
ips =
{
    rules = local_rules,
    variables = local_variables,
}
```

The "snort.lua" file also includes the file "`snort_defaults.lua`"

```
23 dofile(dir .. '/snort_defaults.lua')
```

Open "snort_defaults.lua" and you can see the RULE_PATH to other rules files

```
-- Path to your rules files (this can be a relative path)
RULE_PATH = '../rules'
BUILTIN_RULE_PATH = '../builtin_rules'
PLUGIN_RULE_PATH = '../so_rules'
```

Exercise 2: Write a snort rule to catch a CVS Buffer Overflow Exploit

The first step is to create the rule that will pick up the traffic with the required characteristics. A useful reference in writing rules is snort's manual itself: https://docs.snort.org

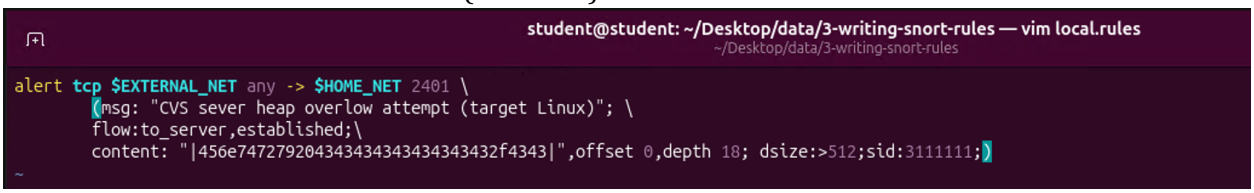
So, your rule should look something like this:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 2401 \
    (msg: “CVS server heap overflow attempt (target Linux)”; \
    flow:to_server,established; \
    content: “|45 6e 74 72 79 20 43 43 43 43 43 43 43 43 2f 43 43|”, offset 0, depth 18; dsize:>512;
    sid:3111111;)
```

Note that before “offset” and “depth” is the colon (,) instead of semicolon.

The second step is to include this rule within snort.lua. Please use the snort.lua file at `'/Desktop/data/3-writing-snort-rules/snort.lua' (that you have copied in exercise 1)` as the basis for this exercise.

Create a rule file named “local.rules” (a text file). Add the rule above into the file and save it.



Create “cvslogs” folder

```
mkdir cvslogs
```

The final step is to run snort to see if your rule picks up the traffic. Open a terminal and navigate to `'/Desktop/data/3-writing-snort-rules'`. Run snort using the following command:

`snort -c snort.lua -r cvs.pcap -l cvslogs -A alert_full --lua "alert_full={file=true}"`

```
snort -c snort.lua -r cvs.pcap -l cvslogs -A alert_full --lua "alert_full={file=true}"
```

After snort finishes executing, you should be able to find 1 alert at the cvslogs folder. On the screen, you will also see this in the result.



The number of alerts is 1 as the figure above.

Please open the alerts file to confirm snort picked up the rule you created.

```
student@student:~/Desktop/data/3-writing-snort-rules$ ls cvslogs
alert_full.txt
```

The content of alert_full.txt should show 1 alert like this.

```
student@student: ~/Desktop/data/3-writing-snort-rules — vim cvslogs/alert_full.txt
~/Desktop/data/3-writing-snort-rules

[**] [1:3111111:0] "CVS sever heap overlow attempt (target Linux)" [**]
[Priority: 0]
05/21-19:08:43.531832 129.170.249.87:45177 -> 129.170.249.118:2401
TCP TTL:64 TOS:0x0 ID:44434 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x9EA8E7AC  Ack: 0x90D127F9  Win: 0x16D0  TcpLen: 20
```

Exercise 3: Write a Snort Rule for a Windows JPEG exploit

Using the snort manual to understand the snort rules syntax, the rule should look something like this:

```
alert tcp $EXTERNAL_NET 80 -> $HOME_NET any \
        (msg: "WINDOWS JPEG buffer overflow attempt" ; \
        flow:to_client, established; \
        content: "Content-Type\: image/jpeg", nocase; \
        content: "|FFED0000|"; sid:3222222;)
```

Add the above rule to local.rules (this file has been created in exercise 2).

```
student@student: ~/Desktop/data/3-writing-snort-rules — vim local.rules
~/Desktop/data/3-writing-snort-rules

alert tcp $EXTERNAL_NET any -> $HOME_NET 2401 \
        (msg: "CVS sever heap overlow attempt (target Linux)"; \
        flow:to_server,established;\
        content: "|456e74727920434343434343434343432f4343|",offset 0,depth 18; dsize:>512;sid:3111111;)

alert tcp $EXTERNAL_NET 80 -> $HOME_NET any \
        (msg: "WINDOWS JPEG buffer overflow attempt"; \
        flow:to_client, established;\
        content: "Content-Type\: image/jpeg", nocase;\
        content: "|FFED0000|"; sid:3222222;)
```

Note that before “nocase” is the colon (,) instead of semicolon.

Because “local.rules” has been added in exercise 2, we can skip this step.

The final step is to run snort to see if your rule picks up the traffic. Open a terminal and navigate to `'/Desktop/data/3-writing-snort-rules'`.

Create a log folder named ‘jpeglogs’

```
mkdir jpeglogs
```

Run snort using the following command:

snort -c snort.conf -r jpeg.pcap -l jpeglogs -A alert_full --lua "alert_full={file=true}"

```
snort -c snort.lua -r jpeg.pcap -l jpeglogs/ -A alert_full --lua "alert_full={file=true}"
```

Check the summary on the screen to see how many alerts are created

```
-------------------------------------------------
detection
                analyzed: 6
              hard_evals: 11
         cooked_searches: 3
            pkt_searches: 3
            pdu_searches: 6
           file_searches: 1
                  alerts: 3
            total_alerts: 3
                  logged: 3
               buf_dumps: 7
-------------------------------------------------
```

Check if any alert file is created in the jpeglogs folder. After snort finishes executing, you should be able to find 1 alert at the jpeglogs folder. Please open the alerts file to confirm snort picked up the rule you created.

```
student@student:~/Desktop/data/3-writing-snort-rules$ ls jpeglogs/
alert_full.txt
```

Exercise 4: Analysing spam traffic

Analysis of 5 unique alerts

DNS SPOOF query response with TTL of 1 min. and no authority

The vast majority of logged traffic (66%) is DNS. By clicking on the Total# of 88 alerts, and then the ID# of the first few entries, we can access the raw packet data. We can see that most of the DNS names relate to smtp mail servers. Therefore, they seem to be DNS replies about mail servers. Snort alerts that the TTL of these replies is less than 1 minute, which is abnormal and could signify the use of fast-flux DNS botnets.

portscan: TCP Portsweep

Portsweep traffic corresponds to 13% of logged activity. By clicking on the Total# of these 18 alerts, and then on the ID# of the first few entries, we can see that the port range is 25/tcp-80/tcp. Again, this is another indication that there is large volume of smtp and web traffic. The scanned IP range does not give many interesting results, but a common factor in all these alerts, is the Source IP, which is our infected host 192.168.0.103.

Snort Alert [1:100000206:0]

By clicking on the Total # of 2 alerts, and then the ID#, we can access the raw packet data. The detected attempts show a POST to a newpost.php, followed by some encoded data. We cannot be sure what this activity is, but given their relatively small volume, it could be communication attempts with botnet handlers. The originator of the activity is our infected host 192.168.0.103 and destinations are 2 external web servers.

ICMP Destination Unreachable Communication with Destination Host is Administratively Prohibited

A large percentage (18%) of traffic corresponds to ICMP error messages. By clicking on the Total# of 24 alerts, we can see that they all originate from external addresses, with our host 192.168.0.103 as destination. One host suddenly receiving error messages from a large number of hosts is another indication of performing illegal activities and causing errors during the process. By accessing this traffic in Wireshark, we can identify the original datagram that caused these errors, but this is for another practical.

Snort Alert [1:100000205:0]

By clicking on the Total # of 2 alerts, and then the ID#, we can access the raw packet data. The detected attempts show a POST to a forums.php, followed by some encoded data. We cannot be sure what this activity is, but given their relatively small volume, it could be communication attempts with botnet handlers. The originator of the activity is our infected host 192.168.0.103 and destinations are 2 external web servers.