1 INTRODUCTION

The objective of this lesson is to learn how to use signature-based Network Intrusion Detection Systems (NIDS) to identify and understand malicious activity found on networks. Signature-based NIDS like snort and suricata extract both metadata (IP address, port, etc.) and content (a sequence of bytes found in a packet) from network packets and compare the extracted data against a list of rules. A NIDS can sniff packets off the network directly or analyze packets found in PCAP files.

Snort and suricata are both free and open source NIDS. The first version of snort was written by Marty Roesch in 1998. Snort was developed and maintained by the company SourceFire until that company was acquired by Cisco in 2013. The current version of snort is 2.9.16. The logo for Snort is a pig, resulting in many of the third party open source tools around snort having pig-related names, such as pulledpork, which downloads and manages NIDS rules for snort, and Barnyard, an output manager which helps deliver alerts to databases and many other sources.

Suricata was developed by the Open Information Security Foundation (OISF) and was first released in 2010. Suricata provides better performance than snort through multithreading and GPU acceleration; it also has smart preprocessors for common protocols like HTTP. While suricata has the lead on performance and features, it is much easier to find documentation and tutorials on snort. Snort 3.0 was announced at the end of 2014 with many of the features of suricata, but it has not yet been released as of the time of this writing.

2 LEARNING OUTCOMES

1. Identify malicious activity in full packet capture data using a NIDS.
2. Identify which rule triggered to produce a NIDS alert.
3. Identify which packet and flow caused the rule to trigger.
4. Analyze NIDS alerts to identify what malicious activity occurred, the origin of the activity, and the victims affected by the activity.

3 NIDS RULES

In this lesson, we will use the `snort` NIDS. It is already installed on the **Monitor VM**. The following directories and files are important to learning how `snort` works and identifying any alerts it raises. Most of these files are only accessible by user `root`.

- `/etc/snort/` contains `snort` configuration files.
- `/etc/snort/snort.conf` is the main configuration file.
- `/etc/snort/rules/` contains `snort` attack signatures.
- `/var/log/snort/` contains `snort` alert log files.

The web site http://www.testmyids.com/ exists to test network intrusion detection systems. That page returns a plain text document containing the string `uid=0(root) gid=0(root) groups=0(root)`. This string is the output of the `id` command when run by root. Any HTTP connection to that site should trigger an alert on a NIDS.

We can find which file contains the rule triggered by the test site with the command:

```
# grep -l "id check returned root" /etc/snort/rules/*
rules/attack-responses.rules
```

Here is the snort rule itself:

```
alert ip any any -> any any (msg:"GPL ATTACK_RESPONSE id check returned root";
content:"uid=0|28|root|29|"; fast_pattern:only; classtype:bad-unknown;
sid:498; rev:6;)
```

The string `any any` on the left of the arrow indicates that the rule should trigger for any source IP address or port, while the same string on the right of the arrow indicates that the rule should trigger for any destination IP address or port. The string provided after content: provides the pattern of bytes to be matched, which includes the string "uid=0", the hex byte 28, the string "root", and the hex byte 29.

To translate bytes in the range 00-7f to ASCII (ASCII is a 7-bit encoding), we can use the `xxd` command as follows. Note that in the shell, we must use the string "0x" to indicate that the following digits should be treated as a hexadecimal number. The final `echo` command adds a newline, so the output is printed as a single line. In this case, we see that 0x28 is the hexadecimal representation of the left parenthesis character in ASCII.

```
$ echo '0x28' | xxd -r && echo
(
```

In a snort alert, the string after `msg:` is text to be printed when this rule is triggered. The string `fast_pattern:only` indicates an optimization to make the rule run faster. The string after `classtype:` provides a classification for the type of attack identified by the signature. The strings after `sid:` and `rev:` are the signature identifier and revision number respectively. The snort manual provides details on how to read and write snort rules. Chapter 9 of *Applied Network Security Monitoring* discusses snort rules in extensive detail.

While this section focuses on snort, the suricata rule format is highly similar to that of snort, with minor incompatibilities.

4 Testing the IDS

In this lab, we will use the **Monitor VM** to run our NIDS. After logging into this VM, verify that `snort` is running with the command:

```
# service snort status
```

The output should contain the highlighted string **active (running)**. If that string is not present, you will need to restart `snort`.

To verify that snort is working correctly, we will trigger the NIDS test rule discussed in the previous section. As the http://www.testmyids.com/ site was not working properly when this lesson was written,

we will need to trigger the rule ourselves using the `nc` command. Before starting, verify that the `snort` log file exists but has no current alerts as it is zero bytes long.

```
# ls -l /var/log/snort
```

On the **Kali VM**, open a terminal window and run a `nc` listener on port 8080.

```
$ nc -l -p 8080
```

On the **Router VM**, become `root`, then run the following command to send the string detected by the rule to **Kali VM**.

```
$ id | nc 192.168.1.10 8080
```

The string generated by `id` will print in the **Kali VM** terminal. It will also trigger a `snort` alert and add content to the `snort` log file.

If we attempt to read the `snort` log file with `cat`, we find that it's a binary file. The `output` directive in the configuration file specifies that snort create a file named `snort.log` in the `unified2` binary format.

```
# grep ^output /etc/snort/snort.conf
output unified2: filename snort.log, limit 128, nostamp, mpls_event_types,
    vlan_event_types
```

The log file in this format contains the packet that triggered the alert, along with some binary coded data below that. We can read `unified2` format with the command `u2spewfoo` to see the following output. The first part of the output is a description of the alert, including the signature of the rule that triggered and the source and destination IP addresses and ports.

```
# u2spewfoo /var/log/snort/snort.log | head -20
(Event)
        sensor id: 0    event id: 1     event second: 1541954756        event microsecond: 252637
        sig id: 498     gen id: 1       revision: 6      classification: 3
        priority: 2     ip source: 217.160.0.187        ip destination: 192.168.1.10
        src port: 80    dest port: 44980        protocol: 6     impact_flag: 0  blocked: 0
        mpls label: 0   vland id: 0     policy id: 0

Packet
        sensor id: 0    event id: 1     event second: 1541954756
        packet second: 1541954756       packet microsecond: 252637
        linktype: 1     packet_length: 364
[    0] 00 50 56 B9 F1 FC 00 50 56 B9 1D 84 08 00 45 00  .PV....PV.....E.
[   16] 01 5E B5 0C 40 00 30 06 F8 7F D9 A0 00 BB C0 A8  .^..@.0.........
[   32] 01 0A 00 50 AF B4 C8 A5 25 29 66 8D DC B3 80 18  ...P....%)f.....
[   48] 07 55 D4 F6 00 00 01 01 08 0A 0E 1F 9D 28 25 74  .U...........(%t
[   64] F5 D9 48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F  ..HTTP/1.1 200 O
[   80] 4B 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A  K..Content-Type:
[   96] 20 74 65 78 74 2F 68 74 6D 6C 0D 0A 43 6F 6E 74   text/html..Cont
[  112] 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 33 39 0D 0A  ent-Length: 39..
```

While the `unified2` performance has significant performance benefits, which are important when using `snort` on high bandwidth links, it can easier to read the default format, which creates a humanly readable text file named `alert` and a timestamped `snort.log` file containing a PCAP of the packets that triggered the alerts. We can configure `snort` to use this output format by commenting out the line containing the `output` directive and restarting the `snort` service. Let us do that, then view the new output files.

```
# service snort stop
# rm /var/log/snort/*
# vim /etc/snort/snort.conf
# service snort start
```

Repeat the `nc` connection between **Router VM** and **Kali VM** that we did above to trigger the same test `snort` alert, to create the log file in the new format.

```
# ls -l /var/log/snort
total 8
-rw-r--r-- 1 root  adm 348 Nov 11 15:09 alert
-rw-r----- 1 snort adm 404 Nov 11 15:09 snort.log.1541966951
# file /var/log/snort/*
/var/log/snort/alert:                ASCII text
/var/log/snort/snort.log.1541966951: tcpdump capture file (little-endian) -
    version 2.4 (Ethernet, capture length 1514)
```

5 UNDERSTANDING SNORT ALERTS

Let us examine the text alert. The most important information is listed on the first line of alert text, which is the alert message itself: `id check returned root`. That is exactly what we expected from the site www.testmyids.com, so this verifies that snort is receiving network packets and generating alerts. Next, let us look at the remainding alert output.

```
# cat /var/log/snort/alert
[**] [1:498:6] ATTACK-RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/09-15:01:16.668160 192.168.1.1:47680 -> 192.168.1.10:8080
TCP TTL:64 TOS:0x0 ID:2223 IpLen:20 DgmLen:91 DF
***AP*** Seq: 0xFF3B874D  Ack: 0x2811C3F3  Win: 0x1F6  TcpLen: 32
TCP Options (3) => NOP NOP TS: 3767521640 1246392905
```

The three numbers in square brackets have the following meanings:

1. **1:** the gid (generator ID), indicating which part of snort produced the alert. Generator 1 is the rules engine. Other generators include various preprocessors and the decoder subsystem.
2. **498:** signature ID that provides a unique identifier for the rule. This identifier can be used to find the rule that produced this alert from the list of rules in /etc/snort/rules.
3. **6:** revision number indicates which version of the rule is the one that triggered the alert. This indicates that this alert was produced by the 6th version of rule 498.

The second line explains the classification of the rule: Classification: Potentially Bad Traffic. As the `id` command could be run by a user who had permission to have root access via `su` or `sudo`, this alert does not always mean that malicious activity is occurring on your network. However, this result is likely unusual enough to warrant investigation. The priority of 2 indicates a medium priority alert. Priority values range from 1 to 4.

The third line provides a timestamp followed by netflow information:

```
192.168.1.1:47680 -> 192.168.1.10:8080
```

The sender's IP address should be the address of the **Router VM**, while the destination IP address should be the address of the **Kali VM**. The source port 47680 is a random high number TCP port used by the wget

client. As client TCP ports are randomly chosen from all high port numbers 1025-65535, the port number in your alert is likely different from the one above.

The fourth and fifth lines provide metadata from the IP and TCP headers of the packet that triggered the alert. The most important piece of information for an investigation is the IP ID (2223 above, but it will be different in the alert on your Monitor VM), as the IP ID allows us to identify the packet that triggered the alert.

Since snort on the Monitor VM is configured to log full packet capture data as well as alerts, you can find the network packets in the `/var/log/snort` directory. There will be one or more files named `snort.log.N`, where `N` is a seemingly random number representing a timestamp. You can read this file with `tcpdump` or Wireshark.

```
# tcpdump -r /var/log/snort/snort.log.*
```

6 EXAMINING PCAPS WITH SNORT

In this section, we will run `snort` on the command line with a pcap file as input. Note that must run `snort` as `root`, so that it can access its configuration and rule files. Before running snort, create a directory to store the output files, then unzip the files for this lesson into that same directory. Assuming you downloaded the data archive files into the default Downloads directory, the commands to setup this directory will be as follows.

```
# mkdir snort
# cd snort
# unzip ~/Downloads/nids-files.zip
```

6.1 MS08-067 EXPLOIT

Let us PCAP file containing the MS08-067 exploit. This exploit has been one of the most reliable exploits provided by the Metasploit exploitation framework for many years. The following command will run snort to analyze the PCAP containing this exploit.

```
# snort -l . -q -A full -c /etc/snort/snort.conf -r pcaps/nids/ms08-067-
   exploit.pcapng
```

The options tell snort to log alert files to the current working directory, to be quiet (not print too much output), print full alert texts, and to read network packets from the file specified after the `-r` option.

This command will create a file named `alert` with the full text of the alerts, as well as a file named `tcpdump.log.N` containing the packets that produced the alerts. The alert file will contain a single alert with the text shown below.

```
# cat alert
[**] [1:2465:7] NETBIOS SMB-DS IPC$ share access [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]
10/02-14:42:06.828190 192.168.1.100:40183 -> 192.168.1.108:445
TCP TTL:64 TOS:0x0 ID:7347 IpLen:20 DgmLen:127 DF
***AP*** Seq: 0xAA0A8D9  Ack: 0xC9D89D2F  Win: 0xED  TcpLen: 32
TCP Options (3) => NOP NOP TS: 553779126 2527
```

Answer the following questions based on the alert reports, loading `snort.log.N` in Wireshark, and online research where necessary:

1. What are the network, transport, and application layer protocols used in the packet that caused snort to emit the alert above?
2. Which IP address was the source of the packet that produced the alert?
3. Which IP address was the target of the packet that produced the alert?
4. What protocol is described by the acronym SMB?
5. What types of resources can be accessed via SMB?
6. What is an IPC$ share?
7. Does the snort alert correctly identify the attack?

7 INVESTIGATION

The file `attack-trace.pcap` contains a network attack. There are no other details provided about the scenario. However, the captured traffic includes only 5 TCP flows, so it is possible to investigate each one in detail. Begin the investigation using `snort`.

Write a report that answers the following questions.

1. What alerts does `snort` generate for this capture file? Include the entire first line of the alert.
2. Which systems are involved? Identify by IP address.
3. Which port is targeted by the attack? What service normally runs on that port?
4. Which OS is running on the target? Identify the OS as specifically as possible.
5. Which vulnerability is being exploited by the attacker?
6. Were any files transferred? Identify the filetype and name of any transferred files if possible, along with how the files were transferred.
7. Were any transferred files malware?
8. Did the attacker issue any shell commands? If so, what were they?

At the end of the question answers, provide a general description of the attacker's activities in the form of a timeline.

For each piece of information provided, note the command (or describe the GUI feature used in Wireshark) used to identify that information.

8 FILES

1. `attack-trace.pcap`
2. `backdoor.pcapng`

3. `ms08-067-exploit.pcapng`

9 REFERENCES

1. Microsoft. Microsoft Security Bulletin MS08-067. https://docs.microsoft.com/en-us/security-updates/securitybulletins/2008/ms08-067.
2. Rapid7. Exploitable vulnerabilities #1 (MS08-067). https://blog.rapid7.com/2014/02/03/new-ms08-067/.
3. Snort User's Manual. http://manual-snort-org.s3-website-us-east-1.amazonaws.com/.
4. Chris Sanders and Jason Smith. *Applied Network Security Monitoring*. Chapter 9. Syngress. 2014.