Security Operations & Incident Management

Lab: Writing snort rules

Overview

The aim of this practical is to introduce you to Snort rules. It contains three parts:

- Viewing a sample snort.lua file
- Writing a snort rule to catch a CVS Buffer Overflow Exploit
- Writing a snort rule to catch a Windows JPEG Exploit
- Understand an advanced snort rule (byte_test)
- Analysing spam traffic

By the end of the practical, you should be able to:

- Understand some basic attributes of a snort rule
- Write snort rules with basic options, using the snort manual as point of reference.
- Understand advanced snort rule options
- Run snort in IDS mode, and get it to read a pcap file as input
- Search for specific rules within the VM image, using the 'grep' command

In order to run this practical, you need to open the virtual machine '**Ubuntu 26'** with VMWare Workstation. Use the following credentials to login:

Username: student
Password: student

A useful reference to understand the syntax and functionality of Snort rules is the Snort Manual itself: https://docs.snort.org

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

Copy the default snort.lua, snort_defaults.lua and sensitive_data.rules to the folder `'Desktop/data/3-writing-snort-rules'`

```
cp /usr/local/etc/snort/snort.lua .
cp /usr/local/etc/snort/snort_defaults.lua .
cp /usr/local/etc/snort/sensitive_data.rules .
```

```
student@student:~/Desktop/data/3-writing-snort-rules$ cp /usr/local/etc/snort/snort.lua .
```

```
cp /usr/local/etc/snort/snort_defaults.lua .
```

```
cp /usr/local/etc/snort/sensitive_data.rules .
```

Open the file `'snort.lua'` in a Text Editor and view its contents. For more information on snort.lua configuration options, please see:
https://docs.snort.org/start/configuration

Please note that any line starting with '--' is a comment.

1. What is the value of the variable HOME_NET and EXTERNAL_NET?
2. Can you spot the rules section?

There are 2 ways you can add a rule into Snort:

a) Snort rules can be placed directly in one's Lua configuration file(s) via the "ips" module,
b) Snort rules can be placed in distinct .rules files that get "included". For example, say we had a malware.rules file in the same directory as our Lua configuration file. We could "include" that rules file like so:

```
ips = { include = 'malware.rules' }
```

If users want to include multiple .rules files, then they can do so like:

```
ips =
{
    rules = [[
        include /path/to/rulesfile1.rules
        include /path/to/rulesfile2.rules
        …
    ]]
}
```

Open the file `'snort_defaults.lua'` in a Text Editor and view its contents.
Open the file 'sensitive_data.rules' in a text editor and view its content

---

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

---

**********Use the tcpdump binary file **cvs.pcap** as input for this exercise. **********

Open a terminal and type the following command, in order to change directory:

```
cd  Desktop/data/3-writing-snort-rules/
```

The `cvs.pcap` file contains a connection that uses an exploit against a host that has a listening CVS (Concurrent Versions Systems) service and attempts to execute a buffer overflow against the server. CVS provides a mechanism that supplies version control, allowing multiple developers to access and change management on software and files. More information about the exploit can be found at:

http://security.e-matters.de/advisories/072004.html

http://securitytracker.com/alerts/2004/May/1010208.html

Write a rule that will alert upon seeing the following exploit details.

Exploit Details:

The exploit that we want to look for will detect an exploit used by attackers to find and compromise vulnerable CVS servers.

Specifically, the rule needs to find the following (please use gedit to write the rule):

1) Connection from a client on an **external network** destined for a **server** on the **home network**
2) **Destination port 2401** (CVS service listens on this port)
3) CVS is a **connection-oriented** protocol and a true attack will be come after a session has been **established**
4) The packet will contain hex code:
   a. `45 6e 74 72 79 20 43 43 43 43 43 43 43 43 43 2f 43 43`
   b. This hex code will be found at **0-byte offset** of the payload
   c. This hex code will not go beyond **18 bytes** into the payload
5) The **data size** of the payload containing this hex code will be **greater than 512** bytes

- Once you write the rule, you need to include it into the `snort.lua` file.
  - Create a text file named “local.rules” and write the rule in the file
  - Open ‘snort.lua’ and reference the rule file within snort.lua with the ‘`include`’ keyword in “ips” module. Please use gedit to edit the file, or you can also use any text editor (vi, vim,…)

```
ips =
{
    rules = [[
        include local.rules
    ]],
    variables = local_variables,
}
```

- Now you can test whether your rule works. You can run snort in IDS mode, using configuration file `snort.lua`, and reading the file `cvs.pcap` as input, to see if it has triggered the alert. Which command did you use?

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

**********Use the tcpdump binary file **jpeg.pcap** as input for this exercise. **********

The `jpeg.pcap` file contains an exploit against vulnerable Windows systems, where a buffer overflow occurs when a malicious JPEG file is read. The Microsoft bulletin regarding this vulnerability can be found at:
http://www.microsoft.com/technet/security/bulietin/MS04-028.mspx

This particular attack involves the attacker's malicious JPEG file being sent to a client host from an HTTP server. Examine the pcap file and write a Snort rule to detect this particular JPEG exploit. All of the conditions that trigger the exploit will be discussed below.

Exploit Details
The exploit will be detected by examining an HTTP response that contains a JPEG image. HTTP responses have headers that detail many aspects of the content being returned. One header is a Content-Type header that, as you would imagine, indicates the type of content being returned. A Content-Type of image/jpeg identifies that a JPEG image is being sent.
A JPEG image is composed of many sections. Each section is delimited by a marker 0xFF. Each section marker is followed by a 1-byte hex character that identifies the type of section it is. Then, following that is a 2-byte length. It has been discovered that if the length has a value of 0 or 1 and the section marker byte has a value of 0x01 to 0xD7 or 0xDA to 0xFF, a buffer overflow may occur. We will simplify our rule to look for a section marker of 0xED followed by a value of 0.
A default set in Snort's configuration returns the first 300 bytes only of an HTTP request. This is supposed to make Snort more efficient, since attacks are generally aimed at the server, not the client. This can be altered in snort.conf by setting the http_inspect flow_depth keyword to have a larger value. However, depending on the HTTP traffic in the network, this may be a hefty performance penalty. We will do this for this exercise, since the content we need to find for this attack is deeper than 300 bytes into the payload.

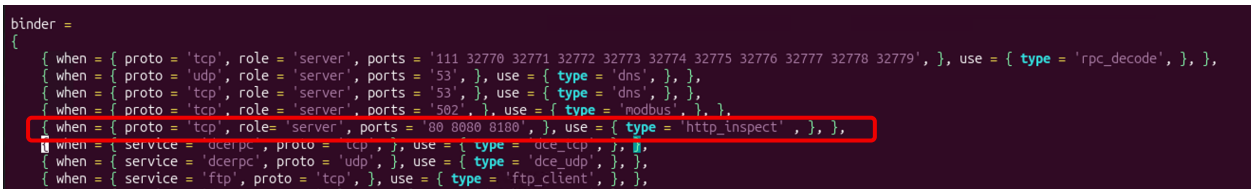
- Edit the `snort.lua` file found in the '`3-writing-snort-rules`' directory with `gedit`.

Find the "binder" and add port 80, 8080, 8180 as follows:

Before



After



- Specifically, the rule needs to find the following:

1. Connection from a server on an external network destined for a client on the home network.
2. Source port 80 (an HTTP response)
3. Connection needs to be established
4. The packet will contain a JPEG image:
    a. The HTTP response header will have Content-Type: image/jpeg
    b. The JPEG content of hex FFED0000

- Once you write the rule, follow similar steps in Exercise 2 to run snort using configuration file `snort.lua`, and reading the file `jpeg.pcap` as input, to see if it has triggered the alert. Which command did you use?

Exercise 4: Download and use a community rule

**********Use the tcpdump binary file **spamsnapshot** as input for this exercise. **********

This traffic was captured in September 2010, and it includes 10 minutes of activity that was generated by infected host 192.168.0.103.

1. Download community rule for Snort 3 from
https://www.snort.org/downloads/#rule-downloads
(you can also use this link
https://www.snort.org/downloads/community/snort3-community-rules.tar.gz)
If the link is not working, you can download snort3-community.rules from DLE.

2. Copy the downloaded file to **Desktop/data/3-writing-snort-rules/**

Upzip the file using tar command line

```
student@student:~/Desktop/data/3-writing-snort-rules$ tar -xzvf snort3-community-rules.tar.gz
```

Copy snort3-community.rules file to Desktop/data/3-writing-snort-rules/

```
cp snort3-community-rules/snort3-community.rules .
```

3. **Add snort3-community.rules to snort.lua**

```
ips =
{
    rules = [[
        include local.rules
        include snort3-community.rules
    ]],
    variables = local_variables,
}
```

4. **Create a new log folder spamlogs**

```
mkdir spamlogs
```

5. **Run snort**

snort –c snort.conf –r spamsnapshot –l `spamlogs` -A alert_full --lua "alert_full={file=true}"

```
snort -c snort.lua -r spamsnapshot -l spamlogs/ -A alert_full --lua "alert_full={file=true}"
```

Note: You can also run snort and output the alerts in json format

- **Run Snort and output the alerts in json format**

snort –c snort.conf –r spamsnapshot –l `spamlogs` **-A alert_json --lua "alert_json={file=true}"**

```
snort -c snort.lua -r spamsnapshot -l spamlogs -A alert_json --lua "alert_json={file=true}"
```

Check if a new alert file is created in spamlogs folder

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

You can read more about different format for alert output at https://docs.snort.org/start/alert_logging

6. View the content of alert_full.txt. Choose 5 unique alerts. What can you tell about these alerts?
7. Choose 3 rules in snort3-community.rules that have generated the alerts, and find out what the rule means.

Hint: You can use the sid each alert (which can be found in the field “alert”) to search for the rule that triggered the alert. Similarly, you can use the sid to search for the snort rule description online.

```
[**] [1:254:17] "PROTOCOL-DNS SPOOF query response with TTL of 1 min. and no authority" [**]
[Priority: 0]
08/23-20:45:55.018510 192.168.0.1:53 -> 192.168.0.103:58983
UDP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:77 DF
Len: 49
```

For example, Snort rule **1:254:17** is a standard text rule (`GID: 1`, `SID: 254`, `Revision: 17`)

Hint: Apart from the alert details, also look for source IP / Destination IP patterns to determine what could have triggered this activity.

Note: Please remember to use the snort alerts as a “torch” that will point your attention towards suspicious activity within the dataset. At this stage, you cannot be certain that snort is right, you need to investigate these alerts and validate them yourself. You can only do this by accessing the raw packets. In the meantime, it is useful to think of the most likely scenario that could have caused these incidents. That way, you can try to find evidence within the dataset to support your theory. It is useful to remember Occam’s razor, e,g, think of the most likely scenario and not the most unbelievably complex and outlandish explanation for what has happened.