Assignment 7: Network traffic monitoring using the Snort intrusion detection system

Students:

Asterios Agiannis 2022030164

Giorgos Vassalos 2022030052

a) Part 2

We used the python add-on named scapy to produce with code the pcap file that contains all the packets that are asked from us. You can see the code in the generate_packets.py file.

b) Part 3

```
#Student's packet
alert tcp any any -> 192.168.1.1 54321 (msg:"ASSIGNMENT7 Student Packet Detected"; content:"aagiannisgvassalos1"; sid:01; rev:1;)

#Port Scan packets
alert ip any any -> 192.168.1.2 any (msg:"ASSIGNMENT7 Port Scan Packet Detected"; content:"aagiannisgvassalos1"; sid:02; rev:1;)

#Base64 packet
alert tcp any any -> 192.168.1.3 8080 (msg:"ASSIGNMENT7 Malicious Base64 ID Detected"; content:"MjAyMjAzMDA1Mg=="; sid:03; rev:1;)

#Suspicious DNS
alert udp any any -> any 53 (msg:"ASSIGNMENT7 Suspicious DNS Query Detected"; content:"malicious"; content:"example"; sid:04; rev:1;)

#Ping packet
alert icmp any any -> 192.168.1.4 any (msg:"ASSIGNMENT7 Custom Ping Detected"; content:"PingTest-2024"; sid:05; rev:1;)
```

```
##### PCAP.pcap #####
        [1:1:1] ASSIGNMENT7 Student Packet Detected (alerts: 1)
        [1:2:1] ASSIGNMENT7 Port Scan Packet Detected (alerts: 10)
        [1:3:1] ASSIGNMENT7 Malicious Base64 ID Detected (alerts: 5)
        [1:4:1] ASSIGNMENT7 Suspicious DNS Query Detected (alerts: 1)
        [1:5:1] ASSIGNMENT7 Custom Ping Detected (alerts: 1)
#####
```

1. The student's packet is generated with these properties: we know the protocol is tcp, we know the destination IP and the port. Finally we know what the payload contains which is constant. So we have the rule check each of these fields for common spots. We see tcp the ip then the port and finally the content arguments in the first rule.
2. The port scan packets are generated as either udp or tcp protocol according to the service we know the destination ip and the content. So we can create a rule that detects either a tcp or a udp packet and then finding the known ip and content. Which are the arguments of the rule.

3. The Base64 packet is generated as a tcp protocol with a known Destination IP as well as port and finally known contents. Knowing these we can create easily a rule which implement those facts.
4. For the DNS suspicious domain packet we know that it is UDP protocol, the destination port and the contents of the packet (the domain name). So we make a rule that checks for all these.
5. The ping packet is ICMP protocol and we know its destination as well as the payload. Creating a snort rule is simple since we use these facts.

c) Part 4

```
#Slammer detection: dest port 1434 and content starting with 04 and being followed by 01 (nop)
alert udp any any -> any 1434 (msg:"ASSIGNMENT7 SQL Slammer Worm Simple"; content:"|04|"; content:"|01 01 01 01 01|"; sid:06; rev:1;)
```

```
##### slammer.pcap #####
        [1:6:1] ASSIGNMENT7 SQL Slammer Worm Simple (alerts: 1)
#####
```

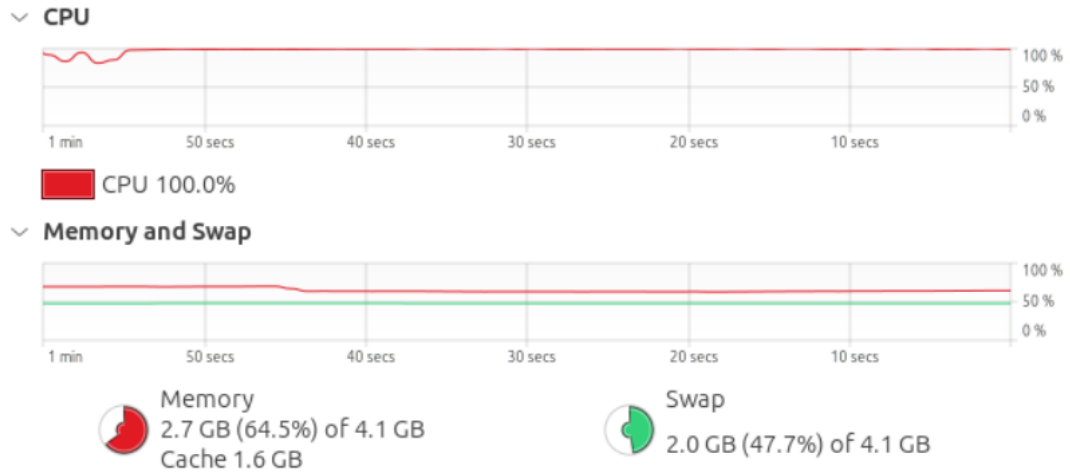
We open the slammer.pcap on wireshark to check on what the SQL slammer's properties are, we notice that the port it sends to is 1434, its protocol is udp and its payload starts with 04 and the continues by having lots of 01s. So noticing this we can create a rule that checks all that that is as it seems above.

d) ***Part 5***

We found the 1GB pcap file at this link. It is the file that's 308MB after we unzip it. Then we run snort over it. We see that we get no messages, however we get a massive impact on our computer. It was running for 10 minutes

```
-------------------------------------------------
Summary Statistics
-------------------------------------------------
timing
                runtime: 00:10:17
                seconds: 617.462114
                packets: 10000000
               pkts/sec: 16207
o")~   Snort exiting
```

And our computer was running at max cpu and high memory usage at all time:



We understood that the high CPU usage was due to the sheer amount of comparisons that the computer had to make to find if any of the packets break a rule.

To optimize the snort program we have we can make a few actions

1. Use BPF which is a filter that filters out packets that are trusted and or low risk. That would make the amount of packets we need to check fewer by a large amount.
2. Fiddle with the settings in snort.conf. There we can change the detection engine search method to one that is less memory intensive as well as comment out some unused preprocessors which are rules we don't need if we aren't doing something specific.

   Those are some optimizations to name a few.

e) ***Part 6***

1. **Describe how Snort uses rules to detect malicious activity.**
   Snort has a certain way of checking packets. It first goes through the header of a packet and sees if the protocol, the source and destination IP and the ports match with one of the rules. Afterwards if it matches those then it checks the payload. If the payload content matches as well then we get an alert.
2. **Mention at least 3 limitations of signature-based intrusion detection systems like Snort with a small description for each one**.
   a. Zero-day-attacks: This kind of detection system can't detect a malicious intent when the attack pattern is unknown. So it can only detect already known exploits.
   b. Encryption: Since snort is reliant on the payload of the packets, if the message is encrypted then it is impossible for snort to find the malicious packet.

c. Since the rules are set in stone if the attacker changes something small in the payload or one of the header properties the attack can go unnoticed.

3. **Pros and cons of using Snort in a real-world scenario**

Pros:

Snort is free to use with a large community behind it which updates and provides more rules to detect even a larger amount of attacks by the day. It also provides the users with an easy way to write custom rules for each system which is really useful in many scenarios.

Cons:

It is a heavy program for a computer to constantly be running especially with the amount of packets we see in our networks daily. So many times it fails to checks packets. The rules need a lot of fine tuning to not be bombing you with alerts while really checking for threats. Finally one needs to constantly search for new rules and even many times configure them to their demands.