Patrick Austin

CS 445 Homework 3

26 March, 2018

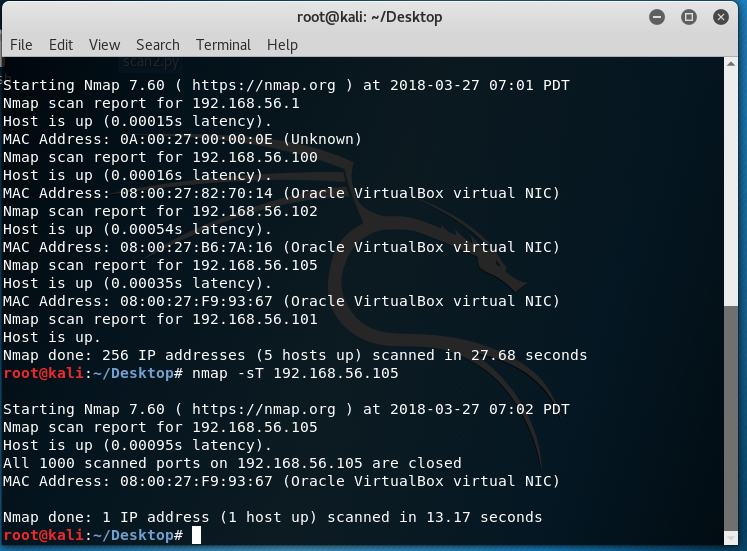
1. Hosts were found with a ping sweep on the network, using the command:

nmap -sn 192.168.56.0/24

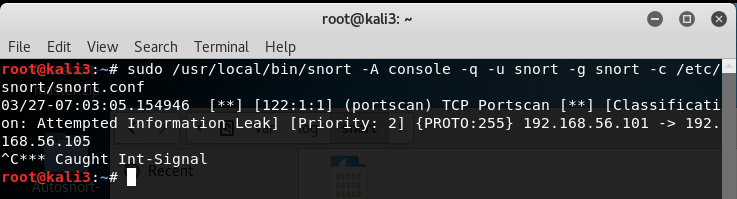
Then a full connect port scan was launched on Kali3 using the command:

nmap -sT 192.168.56.105

These scan results are shown below.



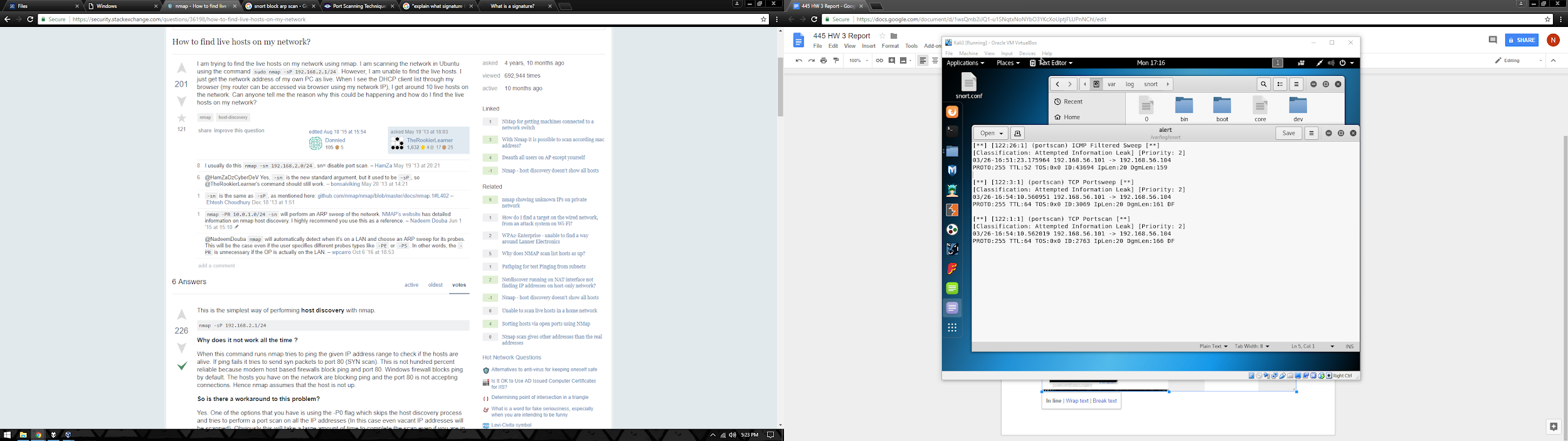
2. Snort was able to detect the port scan, but not the ping sweep. After conducting the sweep and scan commands, only the following signature was triggered:



My theory to explain the fact that the ping sweep was not detected comes down to a peculiarity of nmap: when the -sn command is used by a privileged user to scan a local ethernet network, only ARP requests are used for the sweep. ARP packets are not examined by Snort, and so do not trigger an alert in the sfportscan preprocessor. When I forced nmap to do the full ping sweep featuring TCP and ICMP packets via the command:

nmap -sn --send-ip 192.168.56.104

(this was on a different network configuration, where Kali3 was at 192.168.56.104) and then did the port sweep on Kali3, the following signatures were triggered:

****

The --send-ip command forces the ping sweep to use ICMP echo requests and TCP SYN scanning, so these signatures now being triggered makes sense.

In order to explain the physical significance of the signatures being triggered, consider the structure of the ARP, ICMP, and TCP packets being sent to Kali3 in the course of the ping sweep and port scan. The ARP packets do not trigger any signature and so go undetected. But the ICMP packets contain an echo request and the TCP packets have a raised SYN flag. The sfportscan preprocessor has been designed to detect the presence of these values in the appropriate fields in TCP and ICMP packets and raise an alert upon their arrival at Kali3, so long as snort.conf has been appropriately configured to enable the preprocessor.

See signature documentation at <https://www.snort.org/rule_docs/122-26>, <https://www.snort.org/rule_docs/122-3>, and <https://www.snort.org/rule_docs/122-1>.

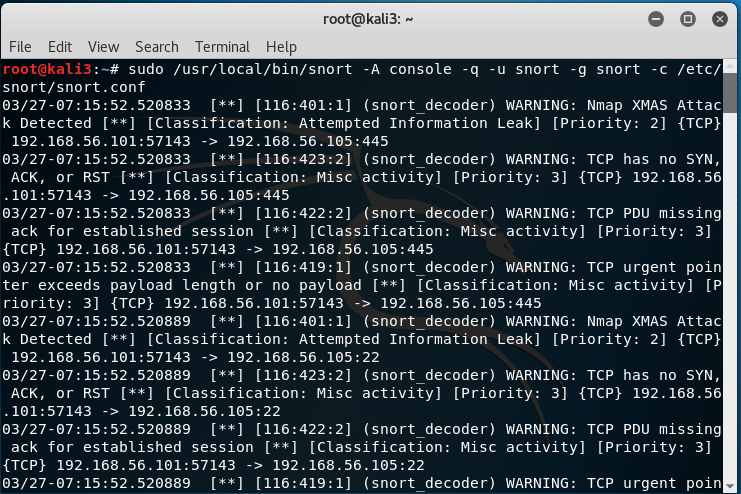
I would like to note an ambiguity in this question relating to the term signature. I take <https://www.snort.org/faq/what-is-a-signature> on its word when it claims that “a signature is any detection method that relies on distinctive marks or characteristics being present in explotis”; therefore I hope this explanation of the distinctive marks or characteristics in the nmap scans will be sufficient; since this detection was done in the sfportscan preprocessor there does not appear to be a simple Snort rule I can provide as illustration.

3. I chose to answer this question using the XMAS scan. When attempting to XMAS scan Kali3 with command:

nmap -sX 192.168.56.105

As the output below shows, Snort was able to detect the nmap XMAS scan in progress via the decoder. Due to innumerable difficulties configuring Snort I was unable to make these warnings into actual alerts without writing my own rules or modifying existing ones; I beg mercy, I tried for hours and hours.

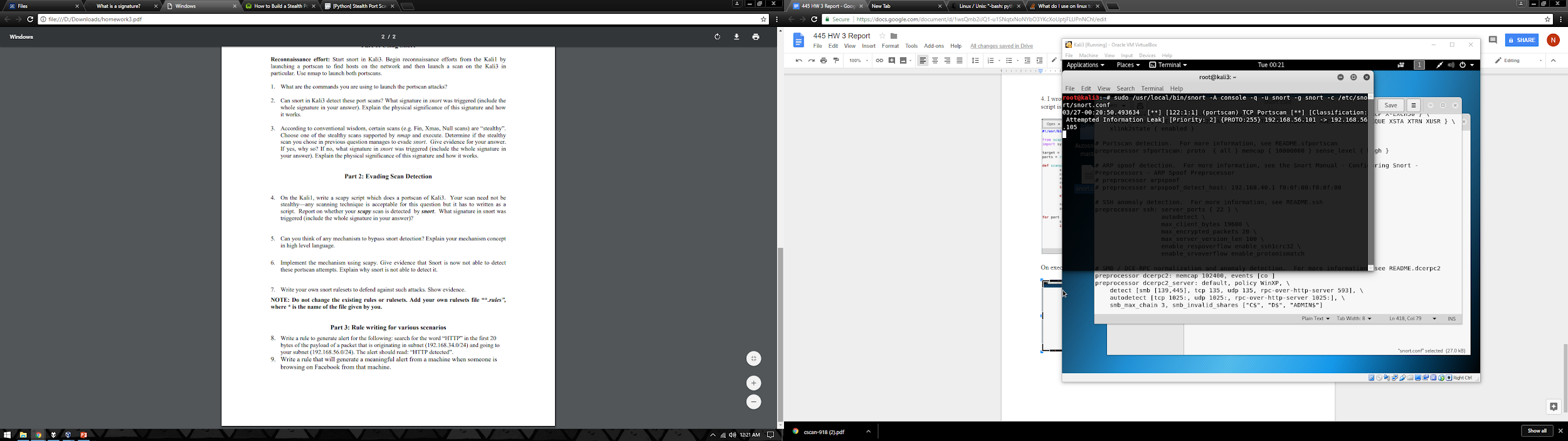
In any case, clearly this stealthy scan does not evade snort. The signature triggered, documented at <https://www.snort.org/rule_docs/116-401>, relies on the FIN, URG, and PSH flags of the incoming TCP packet being raised.



4. I wrote a scapy script to do a half-connect stealth scan, otherwise known as a SYN scan. The script is shown below:



On execution, the scapy script was detected by Snort, as shown below:



The signature triggered was the same one described in question 1, <https://www.snort.org/rule_docs/122-1>, i.e. the sfportscan preprocessor signature for conspicuous behavior of port scanning in a short time frame. When many ports receive SYN packets in a short time frame, as my script executes, the preprocessor can detect and trigger this alert.

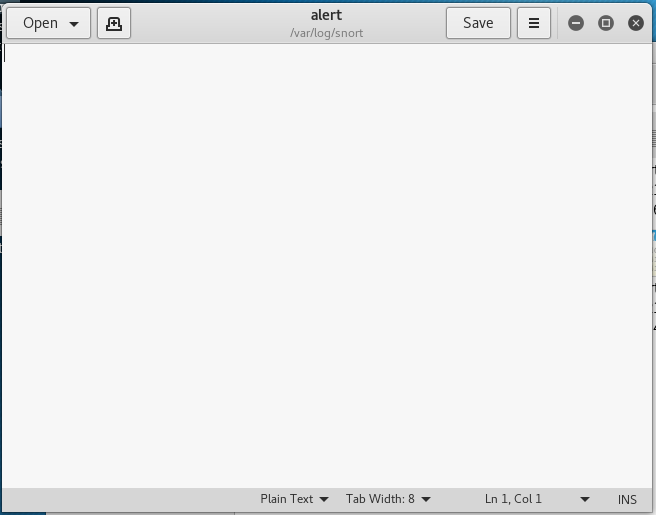
5. Bypassing snort detection will depend on how Snort has been configured by the defender, the presence of any custom rules that have been written, etc. There are probably several approaches that could be successful or fail, all depending on different situations, configurations, and canniness of the IDS designer.

In general though the best practice seems to be to scan in a “low and slow” fashion over as long a period as possible, with many decoys, in order to avoid tripping an alert which might be triggered by a high frequency of packets, or a high frequency of any individual IP trying ports.

6. To implement this “low and slow” approach in scapy, I made some simple modifications to my original script:



I cleared my alerts file and allowed this script to run overnight, and it completed without generating any alerts on the Kali3 snort, as shown by the empty alerts file that greeted me:

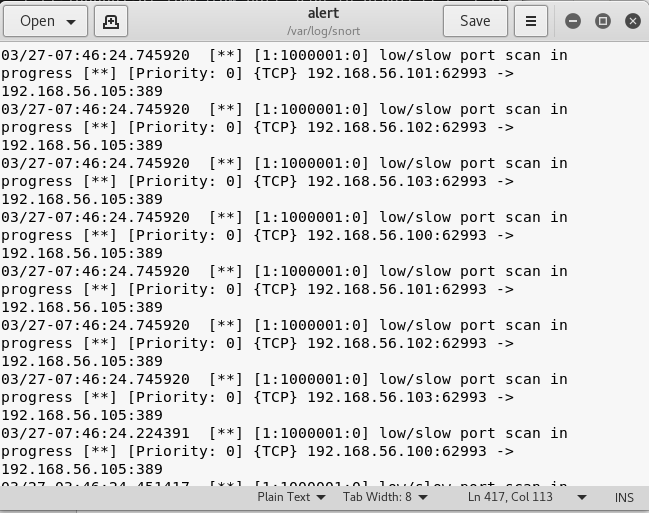


Snort is unable to detect this scan strategy because the default, uncustomized signatures used to detect port scanning rely on a high frequency of scans occurring in a short timespan. Even with sfportscan set to high, it was still unwilling to detect this scan. Otherwise, it is difficult to differentiate someone doing a port scan from a normal user. My use of decoys may also contribute to the muddying effect. The false positive rate would likely be unacceptably high were rules against this type of scan implemented on a network receiving massive traffic from the outside world.

7. Still, this scan strategy can be detected via the addition of a simple custom Snort rule. First I added the following rule to local.rules:

alert tcp $EXTERNAL\_NET any -> $HOME\_NET any (msg:”low/slow port scan in progress (threshold)”; threshold: type both, track\_by\_dst, count 10, seconds 300; flags: S; sid:1000001;)

This rule logs at most one event every 5 minutes if at least 10 ping scans were conducted. Due to the scan frequency this means the log printed one alert every 5 minutes, like so:



This alert struck a balance between verbosity and detail about the scan; it could be tweaked in either direction as need required, but it clearly shows that this type of scan can be detected fairly easily with the addition of custom rules.

8. alert tcp 192.168.34.0/24 any -> 192.168.56.0/24 any (msg: “HTTP detected”; offset:0; depth:20; content: “HTTP”; sid: 10000001; rev:1;)

9. alert tcp $HOME\_NET any -> $EXTERNAL\_NET $HTTP\_PORTS (msg:"facebook use detected by src"; flow:to\_server,established; content:"facebook.com"; nocase;  
sid:1000001; rev:1;)