Network Forensics

- Security analysts are frequently called upon to conduct network forensics on compromised networks by analyzing evidence in the form of captured traffic and flow record logs.

- The objective behind analyzing flow records and packet captures is to identify reconnaissance activity, such as port scans, originating from attacking systems, and then using the data to reconstruct an attacker's processes and activities.

- NMap is a powerful reconnaissance tool that is used by attackers and penetration testers alike for network enumeration, mapping and auditing. Among its most common uses are network mapping, OS identification, firewall auditing, and vulnerability assessments post scanning).

- In this experiment we will generate some nmap scanning activity and then use other tools to analyze the captured traffic, then identify and reconstruct this activity.

- In order for any analysis to be possible it is important that network traffic be captured and preserved since it will contain detailed flow records and evidence of scanning activity.

- We can use a variety of tools that will help us discover the telltale signs of reconnaissance activity, such as Nmap port scans.

- For the purposes of this exercise we will use an open-source tool called "Argus". This is a flow record analysis tool, and it is particularly useful because it operates on flows rather than individual packets.

- Argus will produce reports on TCP flags and connection states. In addition it can use TCP connection states to identify TCP ports that were open during the packet captures, and it can also manipulate the TCP flags for the purposes of Operating System and application fingerprinting.

- Port scanning activity tends to generate frequent and incomplete 3-way handshake connection attempts and a flow oriented tool such as Argus will quickly identify this activity.

- Together with Argus, we can also use tools such as Wireshark to visualize the overall traffic and produce a detailed and compelling body of evidence of reconnaissance and subsequent attacks.

Port Scanning Example

- As an example consider a typical LAN testbed that contains a variety of Windows and Linux machines. The attacking machine's IP address is 192.168.1.200.

- An nmap scan is carried out as follows:

```
# nmap -sS –n -v 192.168.1.0/24
```

- The ***-sS*** SYN scan is the default and most popular scan option since it can be performed quickly, scanning thousands of ports per second on a fast network not hampered by restrictive firewalls.

- It is also relatively stealthy since it never completes TCP connections. This technique is often referred to as half-open scanning, because the 3-way handshake is not completed.

- A SYN packet is sent as a real connection request is initiated and then it waits for a response. A SYN/ACK indicates the port is listening (open), while a RST (Reset) indicates a closed port.

- If no response is received after several retransmissions, the port is marked as filtered. The port is also marked filtered if an ICMP unreachable error (type 3, code 1, 2, 3, 9, 10, or 13) is received.

- At the same time Wireshark was used to capture all of the network activity during the scan. The captured traffic was stored in pcap format.

- The next step is to convert the pcap file to a format require by Argus as follows:

```
argus -r nmap-test.pcap -w nmap-test.ra
```

- We can then start analyzing the ".ra" file with the "**ra**" tool, which is part of Argus (for convenience we can redirect the output to a text file):

```
ra -s stime proto saddr sport daddr dport state -nzr nmap-test.ra > nmap-test.txt
```

- The man page for "ra" provides detailed information on usage. The options and switches above will provide details such as: record start time, protocols, source and destination IPs and ports, transaction state, etc.

- The following is a partial snapshot of the output that is produced as a result of the command above:

```
    StartTime  Proto        SrcAddr  Sport         DstAddr  Dport State
09:27:12.271862   udp fe80::102f:13a4:e*.546          ff02::1:2.547     INT
09:27:12.869491   udp      192.168.1.2.138       192.168.1.255.138     INT
09:27:12.869974   udp      192.168.1.1.138       192.168.1.255.138     INT
09:27:14.767214   udp     192.168.1.141.1024     239.255.255.250.1900    INT
09:27:18.352617   arp     192.168.1.200           192.168.1.22        CON
09:27:18.352858   tcp     192.168.1.200.445       192.168.1.22.49176      E
09:27:18.554500   arp      192.168.1.22          192.168.1.200        CON
09:27:20.272397   udp fe80::102f:13a4:e*.546          ff02::1:2.547     REQ
09:27:22.155627   udp     192.168.1.200.123          66.11.35.2.123     CON
09:27:25.231416   tcp     192.168.1.141.2617      192.168.1.200.139    sSEfF
09:27:25.267126   tcp     192.168.1.141.2618      192.168.1.200.139    sSEfF
09:27:25.320821   tcp     192.168.1.141.2619      192.168.1.200.139    sSEfF
09:27:25.368158   tcp     192.168.1.141.2620      192.168.1.200.139    sSEfF
```

- Now we observe that can see that certain systems have either ongoing traffic flows or are responding to nmap's ARP request ("CON" indicates that there was an ongoing conversation).

- We can isolate all the "CON" states from the data set as follows:

```
ra -s stime proto saddr sport daddr dport state -nzr nmap-test.ra | grep CON
```

- As can be seen below, several systems have responded to the port scanner's ARP requests:

```
09:27:18.352617   arp     192.168.1.200           192.168.1.22        CON
09:27:27.171265   arp     192.168.1.200            192.168.1.1        CON
09:27:37.605630   arp     192.168.1.200            192.168.1.2        CON
09:27:37.605641   arp     192.168.1.200            192.168.1.5        CON
09:27:38.410249   arp     192.168.1.200          192.168.1.104        CON
```

- Once Nmap has confirmed and identified a set of hosts that have responded to its ARPs, it will scan each system using TCP SYN scans.

- Examples of the resulting SYN scans are shown below (notice that nmap is enumerating ports on each live host):

```
09:27:39.063636   tcp   192.168.1.200.56903     192.168.1.22.995     sR
09:27:39.063658   tcp   192.168.1.200.56903      192.168.1.1.995     sR
09:27:39.063682   tcp   192.168.1.200.56903     192.168.1.22.445     sSR
09:27:39.063698   tcp   192.168.1.200.56903      192.168.1.1.445     sSR
09:27:39.063771   tcp   192.168.1.200.56903      192.168.1.5.995     sR
09:27:39.063775   tcp   192.168.1.200.56903      192.168.1.5.445     sSR
09:27:39.063797   tcp   192.168.1.200.56903      192.168.1.2.995     sR
09:27:39.063800   tcp   192.168.1.200.56903      192.168.1.2.445     sSR
09:27:39.065448   tcp   192.168.1.200.56903    192.168.1.104.995     sR
09:27:39.065452   tcp   192.168.1.200.56903    192.168.1.104.445     sSR
09:27:39.067103   tcp   192.168.1.200.56903     192.168.1.22.5900    sR
09:27:39.067119   tcp   192.168.1.200.56903    192.168.1.104.5900    sR
09:27:39.067129   tcp   192.168.1.200.56903      192.168.1.1.5900    sR
09:27:39.067139   tcp   192.168.1.200.56903      192.168.1.2.5900    sR
09:27:39.067150   tcp   192.168.1.200.56903      192.168.1.5.5900    sR
09:27:39.067209   tcp   192.168.1.200.56903     192.168.1.22.199     sR
09:27:39.067237   tcp   192.168.1.200.56903    192.168.1.104.199     sR
09:27:39.067259   tcp   192.168.1.200.56903      192.168.1.1.199     sR
09:27:39.067273   tcp   192.168.1.200.56903      192.168.1.2.199     sR
09:27:39.067287   tcp   192.168.1.200.56903      192.168.1.5.199     sR
09:27:39.067300   tcp   192.168.1.200.56903     192.168.1.22.111     sR
09:27:39.067310   tcp   192.168.1.200.56903    192.168.1.104.111     sR
09:27:39.067320   tcp   192.168.1.200.56903      192.168.1.1.111     sR
09:27:39.067333   tcp   192.168.1.200.56903      192.168.1.2.111     sR
09:27:39.067343   tcp   192.168.1.200.56903      192.168.1.5.111     sSR
```

- The streams are reported in the last field as a set of state variables such as ***s***, ***sR***, ***sSR***, etc. These state variables indicate the TCP connection status.

- A summary of these state changes (see man page) generated by the switches specified is shown below:

```
-z
   Modify status field to represent TCP state changes. The values of the status field when this is
   enabled are:
      's' - Syn Transmitted
      'S' - Syn Acknowledged
      'E' - TCP Established
      'f' - Fin Transmitted  (FIN Wait State 1)
      'F' - Fin Acknowledged (FIN Wait State 2)
      'R' - TCP Reset
```

-Z <s|d|b>
Modify status field to reprsent actual TCP flag values. <'s'rc | 'd'st | 'b'oth>. The characters that can be present in the status field when this is enabled are:

'F' - Fin
'S' - Syn
'R' - Reset
'P' - Push
'A' - Ack
'U' - Urgent Pointer
'7' - Undefined 7th bit set
'8' - Undefined 8th bit set

- With this data we can analyze the activity and draw some conclusions. Notice that none of the TCP flows completed the 3-way handshake (this would be labeled as “E”). This is usually a very good indicator of scanning activity.

- A sequence such as “sR” indicates that a SYN (“s”) was sent but the host at the other end responded with a Reset (“R”), indicating a closed port.

- A sequence such as “sSR” indicated that a SYN was sent and the receiving host responded with a SYN/ACK (“S”), thus indicating an open port. Note that the Reset (“R”) was sent by the receiving host since it had a timeout while waiting for a response to its SYN/ACK.

Fingerprinting the Tools

- We can use the following command to determine the number of TCP probes that were sent by the scanning system to each destination host:

  ra -nnz -s daddr -r nmap-test.ra - 'src host 192.168.1.200 and tcp' | sort -n | uniq –c

  ```
  1       DstAddr
    841     192.168.1.1
    880   192.168.1.104
    867     192.168.1.2
    891    192.168.1.22
    893     192.168.1.5
  ```

- We can also determine the number of ports scanned by the attacker as follows:

```
# ra -nnz -s dport -r nmap-test.ra - 'src host 192.168.1.200 and dst 192.168.1.5 and tcp' | sort -n | uniq -c | grep -v Dport | wc -l
893

# ra -nnz -s dport -r nmap-test.ra - 'src host 192.168.1.200 and dst 192.168.1.22 and tcp' | sort -n | uniq -c | grep -v Dport | wc -l
890
```

- Then we can determine the number of responses (SYN/ACK) returned by the target host as follows:

```
# ra -nnz -S dport -r nmap-test.ra - 'src host 192.168.1.5 and dst 192.168.1.200 and tcp' | sort -n | uniq -c | grep -v Dport | wc -l
43

# ra -nnz -S dport -r nmap-test.ra - 'src host 192.168.1.22 and dst 192.168.1.200 and tcp' | sort -n | uniq -c | grep -v Dport | wc -l
38
```

- NMap usually sends out up to 1000 probes using the -sS default scan. The number of probes sent out to each target is fairly consistent and close to 1000. Given the statistics above we can be reasonably certain the nmap was being used to scan the network.

Examining the Responses From Each Target

- We can now reconstruct the traffic flows between the attacker and target hosts using Argus. Basically what we look for are ports that returned a SYN/ACK (open port), and those that returned a RST (closed port), and which did not return anything at all.

- The Argus command to accomplish this is as follows:

```
ra -nnz -s daddr dport state -r nmap-test.ra - 'src host 192.168.1.200 and syn and synack' | sort -ur
```

- The above command will generate an output similar to:

```
DstAddr  Dport State
     192.168.1.5.80      sSR
     192.168.1.5.445     sSR
     192.168.1.5.443     sSR
     192.168.1.5.25      sSR
     192.168.1.5.22      sSR
     192.168.1.5.143     sSR
     192.168.1.5.139     sSR
     192.168.1.5.111     sSR
     192.168.1.5.110     sSR
     192.168.1.22.49156   sSR
    192.168.1.22.49154   sSR
    192.168.1.22.49153    sS
    192.168.1.22.49152    sS
    192.168.1.22.445     sSR
    192.168.1.22.139     sSR
    192.168.1.22.135     sSR
```

- From the output above we have a clear picture of the specific ports that were open and responded to the SYN probes.

- Now as a further piece of analysis recall that earlier we determined the total number of SYN/ACK's sent back from each host as follows:

```
# ra -nnz -S dport -r nmap-test.ra - 'src host 192.168.1.5 and dst 192.168.1.200 and tcp' | sort -n | uniq -c | grep -v Dport | wc -l
43

# ra -nnz -S dport -r nmap-test.ra - 'src host 192.168.1.22 and dst 192.168.1.200 and tcp' | sort -n | uniq -c | grep -v Dport | wc -l
38
```

- Comparing these totals to the open ports output above, we draw another conclusion that the target systems were retransmitting the SYN/ACK's after timing out waiting for the final ACK.

- We can easily confirm this by examining the number of "R" flows (Reset) from each target:

```
# ra -nnz -R dport -r nmap-test.ra - 'src host 192.168.1.5 and dst 192.168.1.200 and tcp' | sort -n | uniq -c | grep -v Dport | wc -l
43

# ra -nnz -R dport -r nmap-test.ra - 'src host 192.168.1.22 and dst 192.168.1.200 and tcp' | sort -n | uniq -c | grep -v Dport | wc -l
38
```

- As can be observed the results correlate, this confirming our hypothesis that this must be scanning (reconnaissance) activity.

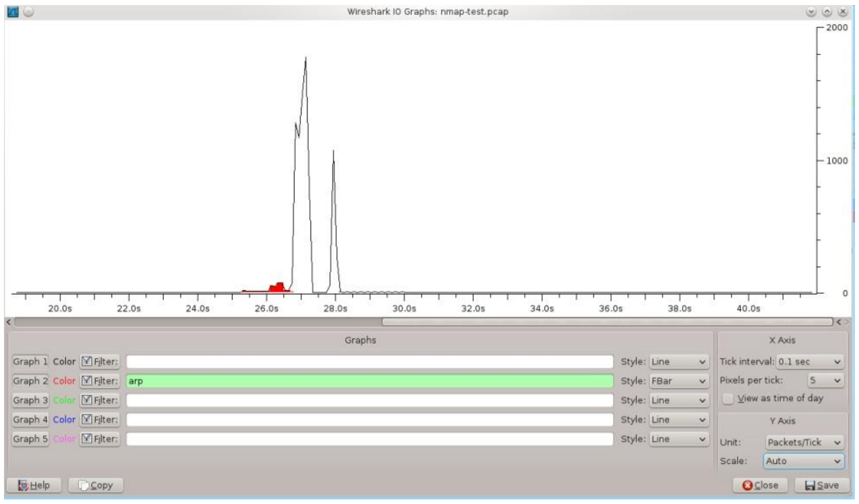
- Note that if there was any additional traffic flow from the open ports (such as would be the case after a compromise) and external IP addresses, we can examine and analyze the traffic flows using similar techniques. In this case we would be looking for established flows (“E”).

- Although we have only analyzed a network scan in our experiment above, the network forensics techniques illustrated above can be used to analyze any traffic flow, including other protocols such as UDP.

Visualizing Network Traffic

- Now let us see if we can visualize the overall data set using Wireshark and draw further conclusions to support (or not) the hypothesis that what we are observing is network scanning activity.

- We will use the “Statistics->IO Graphs” menu item in Wireshark to graph the data set. The following graph shows the overall network traffic (black) together with ARP traffic (red):

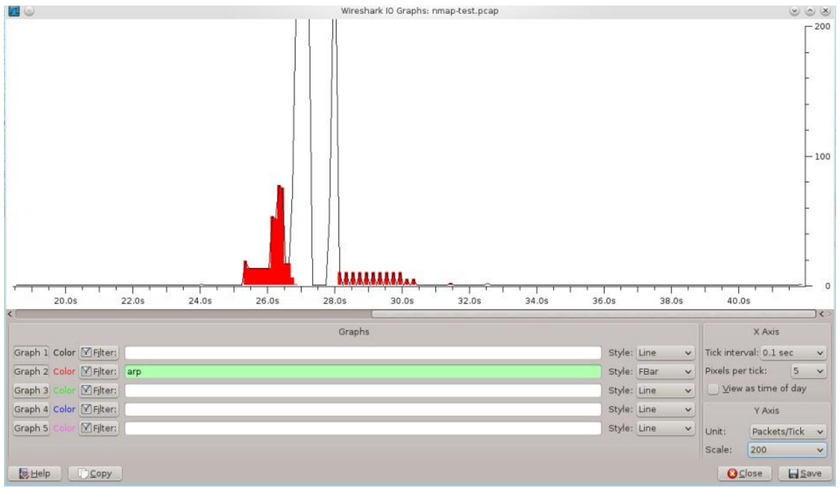


Graph 1 Color Filter: Style: Line
Graph 2 Color Filter: arp Style: FBar
Graph 3 Color Filter: Style: Line
Graph 4 Color Filter: Style: Line
Graph 5 Color Filter: Style: Line

Tick interval: 0.1 sec
Pixels per tick: 5
View as time of day

Unit: Packets/Tick
Scale: Auto

- We can see that there are two main peaks of traffic indicating that there was an initial surge and then a pause, followed by another burst of traffic. To get a better view of detail we will change the vertical scale to better see the smaller peaks:



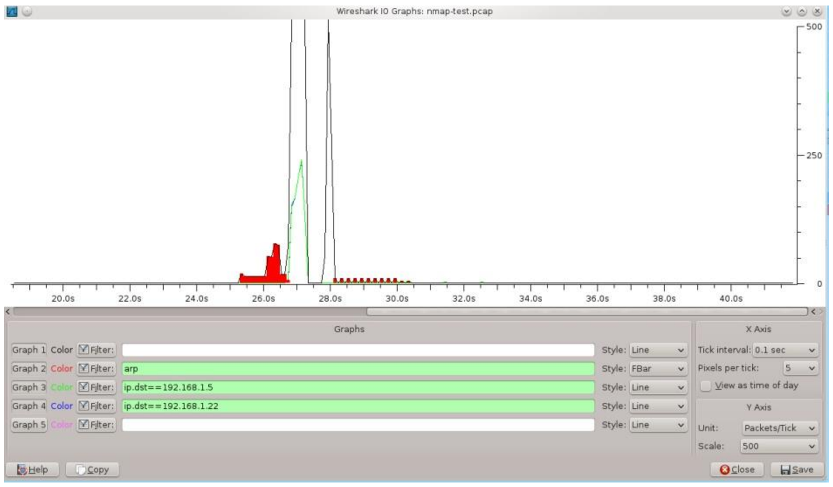
Graph 1 Color Filter: Style: Line
Graph 2 Color Filter: arp Style: FBar
Graph 3 Color Filter: Style: Line
Graph 4 Color Filter: Style: Line
Graph 5 Color Filter: Style: Line

Tick interval: 0.1 sec
Pixels per tick: 5
View as time of day
Unit: Packets/Tick
Scale: 200

- Now we observe that the burst of traffic was preceded by a large burst of ARP traffic, followed by the two main peaks and then smaller peaks of ARP traffic.

- At this point we form a working hypothesis the initial ARP burst was due to the scanning activity of the lower IP addresses, followed by targets responding and then the remainder of the ARP traffic for the higher IP address space.

- In order to confirm this we can start graphic the traffic from some of the target hosts to see how their traffic correlates to what we have seen so far:

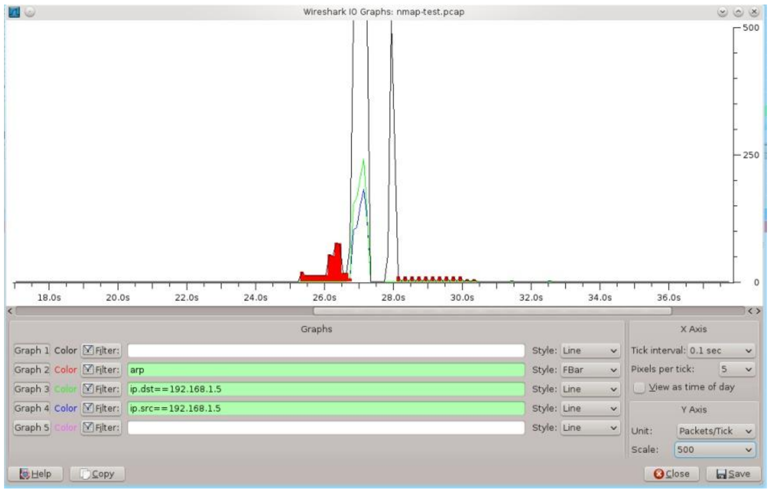


Graph 1 Color Filter: Style: Line
Graph 2 Color Filter: arp Style: FBar
Graph 3 Color Filter: ip.dst==192.168.1.5 Style: Line
Graph 4 Color Filter: ip.dst==192.168.1.22 Style: Line
Graph 5 Color Filter: Style: Line

Tick interval: 0.1 sec
Pixels per tick: 5
View as time of day
Unit: Packets/Tick
Scale: 500

- We now observe that the traffic sent to the two hosts 192.168.1.5 (green) and 192.168.1.22 (blue) almost overlap and are part of the first peak of traffic.

- Next we will plot the traffic to and from host 192.168.1.5 to see where it fits in the graph:

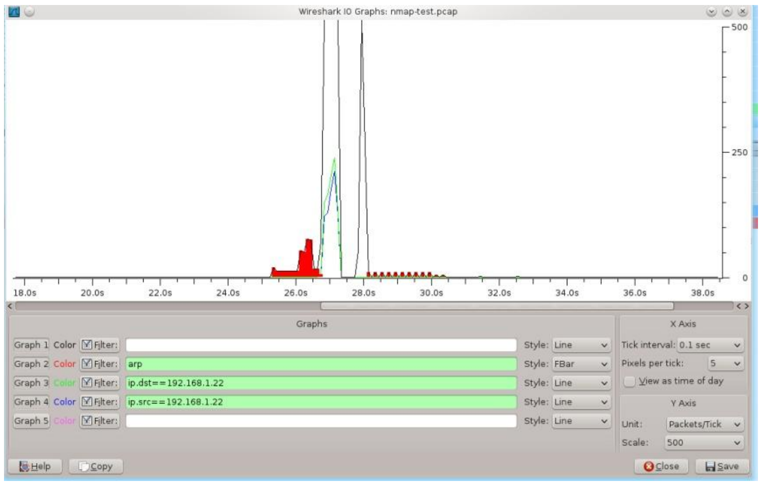


Graph 1 Color Filter: Style: Line
Graph 2 Color Filter: arp Style: FBar
Graph 3 Color Filter: ip.dst==192.168.1.5 Style: Line
Graph 4 Color Filter: ip.src==192.168.1.5 Style: Line
Graph 5 Color Filter: Style: Line

Tick interval: 0.1 sec
Pixels per tick: 5
View as time of day
Unit: Packets/Tick
Scale: 500

- Observe that the host responds (blue) almost immediately to the scanning traffic (green).

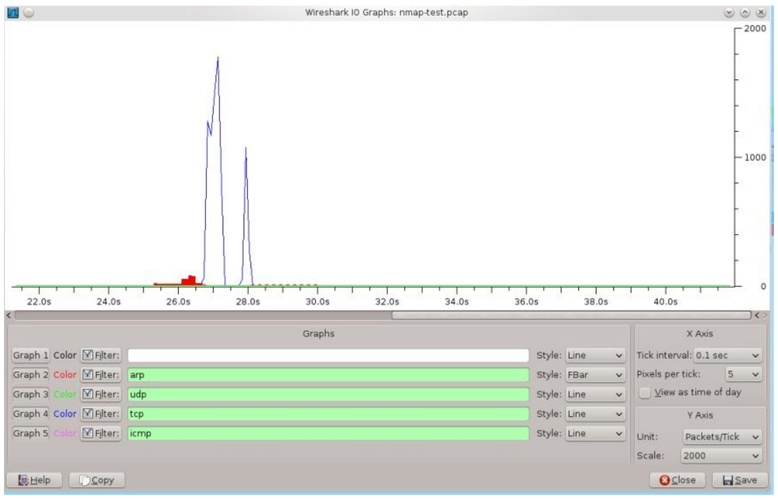
- We make a similar observation for host 192.168.1.22:



Graph 1 Color Filter: Style: Line
Graph 2 Color Filter: arp Style: FBar
Graph 3 Color Filter: ip.dst==192.168.1.22 Style: Line
Graph 4 Color Filter: ip.src==192.168.1.22 Style: Line
Graph 5 Color Filter: Style: Line

Tick interval: 0.1 sec
Pixels per tick: 5
View as time of day
Unit: Packets/Tick
Scale: 500

- Now let us plot the traffic by protocol to see how our observations so far correlate:



Graphs

Graph 1 Color Filter: Style: Line
Graph 2 Color Filter: arp Style: FBar
Graph 3 Color Filter: udp Style: Line
Graph 4 Color Filter: tcp Style: Line
Graph 5 Color Filter: icmp Style: Line

X Axis

Tick interval: 0.1 sec
Pixels per tick: 5
View as time of day

Y Axis

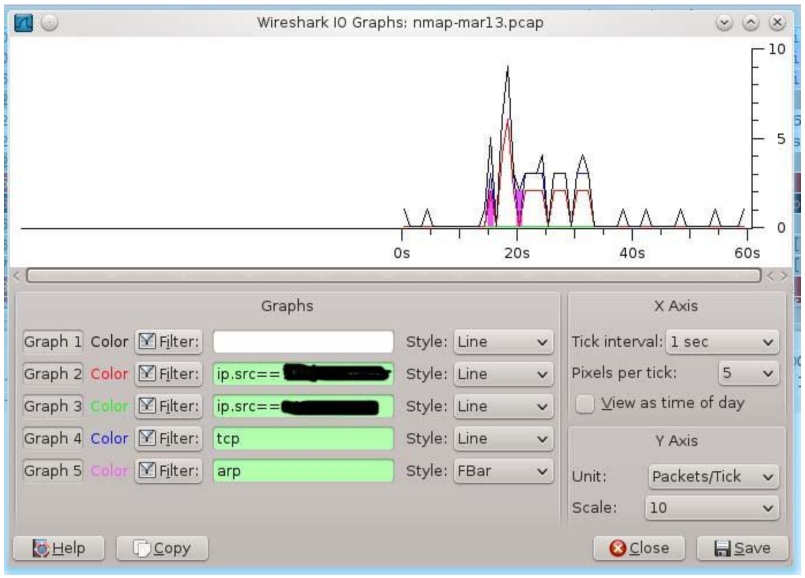
Unit: Packets/Tick
Scale: 2000

- We observe that virtually all of the traffic is TCP traffic. This combined with our earlier observations of individual host TCP traffic confirms our hypothesis that this is indeed reconnaissance activity in the form of port scanning.

- Furthermore, we can conclude that the second peak is a characteristic of nmap, which scans systems in groups, and so after the first lower IP group scans were complete, it carried out a scan of a second group at the higher IP addresses. This also confirms that nmap was the scanning tool.

Example of an external scan

- The following graph shows reconnaissance activity in the form of port scanning from an external attacking host.



Graphs

Graph 1 Color Filter: Style: Line
Graph 2 Color Filter: ip.src==[redacted] Style: Line
Graph 3 Color Filter: ip.src==[redacted] Style: Line
Graph 4 Color Filter: tcp Style: Line
Graph 5 Color Filter: arp Style: FBar

X Axis

Tick interval: 1 sec
Pixels per tick: 5
View as time of day

Y Axis

Unit: Packets/Tick
Scale: 10

- The black trace is the overall traffic captured during the scan.
- The red trace is the inbound traffic coming from an external, attacking system.
- The green trace is outbound traffic from the victim host.
- The blue trace is the overall TCP traffic, and the pink bars are the ARP traffic.