

# **A Practical Guide Wireshark Forensics**



**Alasdair Gilchrist**

## **RG Consulting**

# **A Practical Guide to Wireshark Network Analysis**

Copyright©Alasdair Gilchrist 2015



# Wireshark – Practical Analysis and Forensics

## What is Wireshark?

Wireshark is an open source, network protocol analyzer for Linux and Windows. It has many features as standard such as deep inspection of hundreds of protocols, live capture and offline analysis. Wireshark has an intuitive GUI frontend plus many inbuilt sorting and filtering options making it very simple to use even for beginners. Tshark is the terminal version of Wireshark which is very similar to Tcpdump.

## How does it work?

Wireshark works simply by placing the network card on the machine on which it is running into what is called promiscuous mode. In this mode of operation the network card will accept any network information not just information specifically addressed to itself, which is the normal mode of operation.

In a hub network, which is rare these days, this will be sufficient as all network traffic will be sent out every port on the hub thereby ensuring that the Wireshark network card would receive all traffic traversing the network. Today's modern networks are not hubs though, they are switches, which means only traffic destined for a host station known to be connected on a port is sent out that port. This greatly reduces unnecessary traffic on the network. Unfortunately, this means that Wireshark will not receive all the traffic on the network as it will only see traffic exiting the switch, which is destined for its own directly connected network card.

## Port Mirroring

The solution to the switched network dilemma is to use a technique on the switch to mirror traffic from all the ports on the switch out the port connecting the Wireshark network card. By doing this Wireshark will then see all the network traffic.

## Downloading Wireshark

Wireshark is available for free download here:

<https://www.wireshark.org/download.html>

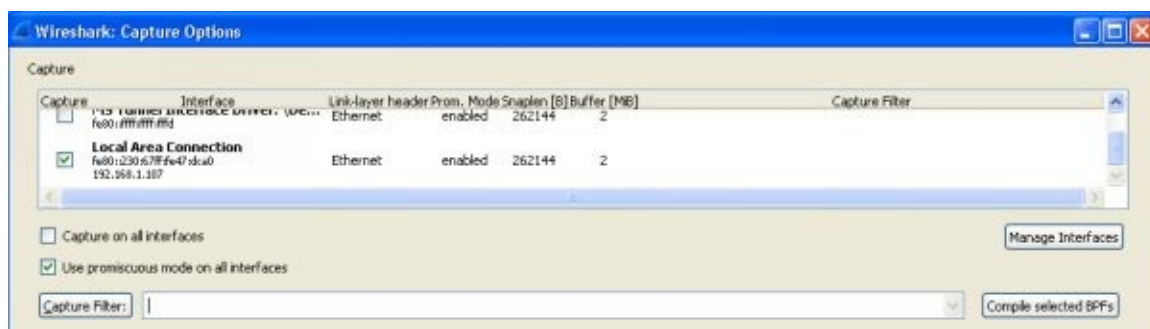
The latest stable version at time of writing (May 2015) is 1.12.4

There is also a video introduction to Wireshark here:

<https://www.wireshark.org/video/wireshark/introduction-to-wireshark/>

## Getting Started

The first thing to do before starting to capture packets on a network, is to make sure you have permission, running Wireshark on your company network without permission is not a good idea. However if you have permission then, first select one or more of the interface cards on the machine running Wireshark to be the capturing interface. This is done from the Wireshark applications front page (Start) or from Capture Options directly below Start. Clicking on the default Capture Options will give you this screen:



Select the interface and a capture filter if required. It is best to enter a capture filter because if you do not you will capture absolutely everything on the wire which is going to be overwhelming. Wireshark is very flexible and there are many inbuilt filters available, by clicking on the Capture Filter button you will see many filters that can be customized to suit your purposes. For example you can select a protocol to be captures such as TCP or UDP, traffic from a given machine's IP address or MAC address, ignore broadcasts, or a combination of all three.

## Capturing Packets

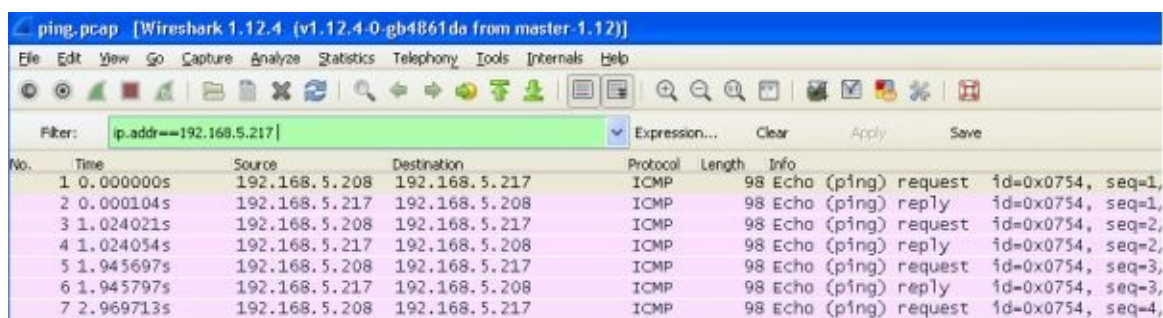
Once the network interface has been configured and optionally a filter applied click start and the packets will soon start to be listed. Click on the stop capture button on the ribbon at the top when you wish to terminate the capture.

## Color Coding

Wireshark color-codes the packets on the screen so that you can easily differentiate between protocols and even different streams. By default green is TCP, light blue UDP, and black is packets with problems such as out of order.

## Filtering Output

Once you have captured your traffic you can refine what is displayed using the filter option at the top of the page. Filtering will limited what is displayed and is very handy for isolating traffic of interest. For example in the image below a filter has been applied to limit the displayed traffic to be only traffic from or to host 192.168.5.217



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000s	192.168.5.208	192.168.5.217	ICMP	98	Echo (ping) request id=0x0754, seq=1,
2	0.000104s	192.168.5.217	192.168.5.208	ICMP	98	Echo (ping) reply id=0x0754, seq=1,
3	1.024021s	192.168.5.208	192.168.5.217	ICMP	98	Echo (ping) request id=0x0754, seq=2,
4	1.024054s	192.168.5.217	192.168.5.208	ICMP	98	Echo (ping) reply id=0x0754, seq=2,
5	1.945697s	192.168.5.208	192.168.5.217	ICMP	98	Echo (ping) request id=0x0754, seq=3,
6	1.945797s	192.168.5.217	192.168.5.208	ICMP	98	Echo (ping) reply id=0x0754, seq=3,
7	2.969713s	192.168.5.208	192.168.5.217	ICMP	98	Echo (ping) request id=0x0754, seq=4,

## Using Wireshark Sample Source Files

If you have no traffic on a network or you do not have permission to capture traffic you can alternatively learn how to use Wireshark using the many sample capture files on the Wireshark Wiki page.

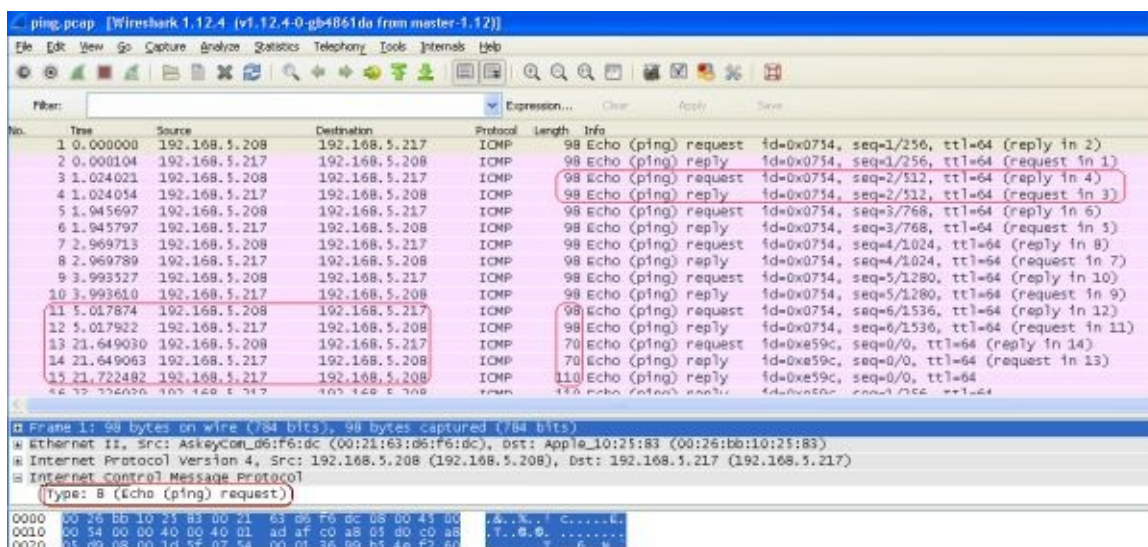
However, in the examples that follow, we are going to show you how to analyze some sample captured files that really show what you can do with Wireshark once you start to apply forensic analysis.

The sample files that we are going to use in the following forensic analysis tutorials you can find here:



## A - Ping.pcap

### 1. What type of ICMP traffic is shown in this capture?



The type of ICMP traffic in the traffic scan is ICMP echo ping requests (type 8) and echo ping replies (type 0) this can be determined by analyzing the ping.pcap with Wireshark or a similar protocol analyser, as shown above.

In the main information panel, we can see that the protocol on view is ICMP (Internet Control Message Protocol) and the information section shows the message types to be echo request and replies between 192.168.5.208 and 192.168.5.217 though they are carrying non-standard ICMP payload.

## 2. What is the number of the first frame that indicates that something funny might be going on?

No.	Time	Source	Destination	Protocol	Length	Info
12	5.017922	192.168.5.217	192.168.5.208	ICMP	98	Echo (ping) reply
13	21.649030	192.168.5.208	192.168.5.217	ICMP	70	Echo (ping) request
14	21.649063	192.168.5.217	192.168.5.208	ICMP	70	Echo (ping) reply
15	21.722482	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
16	23.226939	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
17	24.732304	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
18	26.238410	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
19	27.744492	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
20	29.250005	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
21	30.755704	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
22	32.261811	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
23	33.767892	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
24	35.273062	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
25	36.779217	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
26	38.284646	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply

Frame 13: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0

Ethernet II, Src: Askeycom\_06:f6:dc (00:21:63:d6:f6:dc), Dst: Apple\_10:25:83 (00:26:bb:10:25:83)

Internet Protocol Version 4, Src: 192.168.5.208 (192.168.5.208), Dst: 192.168.5.217 (192.168.5.217)

Internet Control Message Protocol

Type: 8 (Echo (ping) request)

The first indication that something might not be all it seems is at frame number 13. Prior to this point the ping requests, both request and replies were a uniform size of 98 bytes. From Frame 13 onwards the echo request and reply packets no longer followed any uniform pattern, in some cases the replies were much larger than the requests, for example the response to echo request 13, takes 14 – 16 packets to deliver. This was an indication that something more than standard ping test data is being carried within the payload of the ICMP packets. Also from the pcap file, we can see in Wireshark that the identifier and sequence changes in frame 13, which is also indicative of a covert channel being established.

## 3. What is the application layer protocol that is hidden within the ICMP traffic?

No.	Time	Source	Destination	Protocol	Length	Info
12	5.017922	192.168.5.217	192.168.5.208	ICMP	98	Echo (ping) reply
13	21.649030	192.168.5.208	192.168.5.217	ICMP	70	Echo (ping) request
14	21.649063	192.168.5.217	192.168.5.208	ICMP	70	Echo (ping) reply
15	21.722482	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
16	23.226939	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
17	24.732304	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
18	26.238410	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
19	27.744492	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
20	29.250005	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
21	30.755704	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
22	32.261811	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
23	33.767892	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
24	35.273062	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
25	36.779217	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply
26	38.284646	192.168.5.217	192.168.5.208	ICMP	110	Echo (ping) reply

Internet Control Message Protocol

Type: 0 (Echo (ping) reply)

Code: 0

Checksum: 0x190f [correct]

Identifier (BE): 58780 (0xe59c)

Identifier (LE): 40165 (0x9ce5)

Sequence number (BE): 0 (0x0000)

Sequence number (LE): 0 (0x0000)

Data (68 bytes)

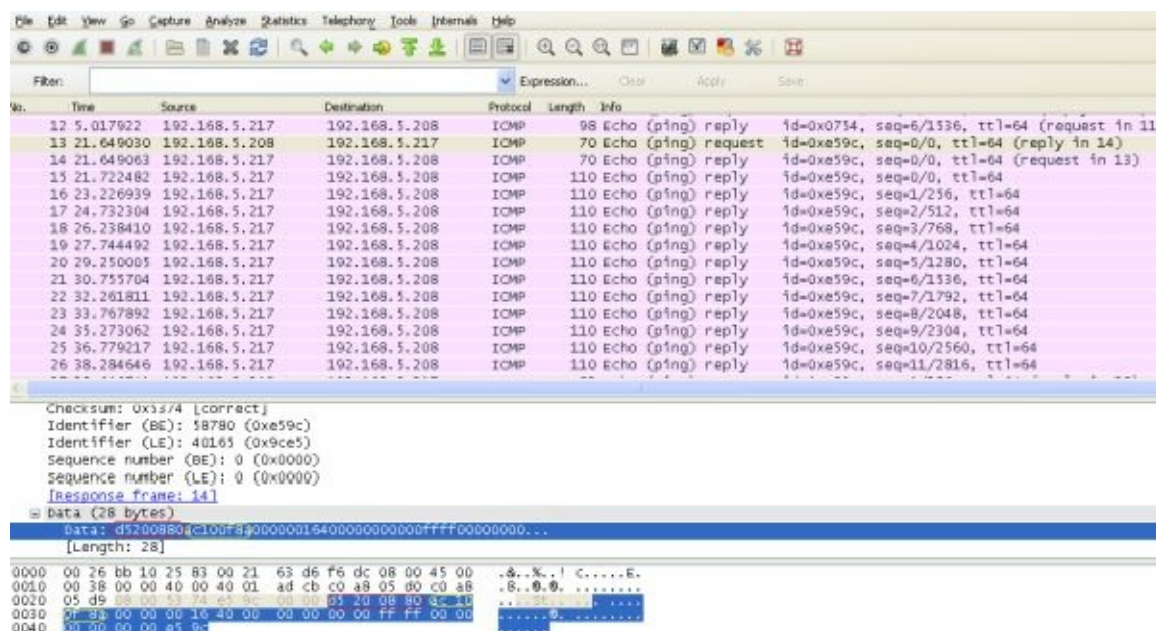
0020 05 00 00 00 00 19 0f e5 9c 00 00 d5 20 08 80 00 00  
0030 00 00 00 00 00 00 80 00 00 02 00 00 00 00 00 00  
0040 00 27 00 00 e5 9c 53 53 48 2d 32 2e 30 2d 4f 70  
0050 63 6e 53 53 48 5f 35 2e 33 70 31 20 44 65 62 69  
0060 61 6e 2d 33 73 62 75 6e 74 75 36 0d 0a fd

.....SS M-2.0-Op  
enSSH\_5.3p1 Debi  
an-3ubun tu9...



The application layer protocol that is hidden within the ICMP payload and being tunneled through ICMP echo request/replies is SSH. This can be seen from frames 15 onwards when the client is negotiating security protocols with the Linux server.

#### 4. What tool most likely generated this ‘malicious’ traffic?



The most likely tool used to generate this type of tunneled traffic over ICMP is Ping Tunnel, which establishes a covert connection between two remote computers (a client and proxy), using ICMP echo requests and reply packets to establish a covert channel between a client, a proxy and a destination machine.

Ptunnel accomplished this by using its own packet format shown below:

Magic	Dest IP	Port	State	Ack	Length	Sequence	Rsv	Data
No								....

The Magic number is used to identify this packet as a pTunnel packet. The reason they have this identifier is to differentiate them from normal ping (ICMP echo request/replies) packets. Therefore, to identify the pTunnel covert channel we need to be able to find this magic number within the payload of our ping trace.

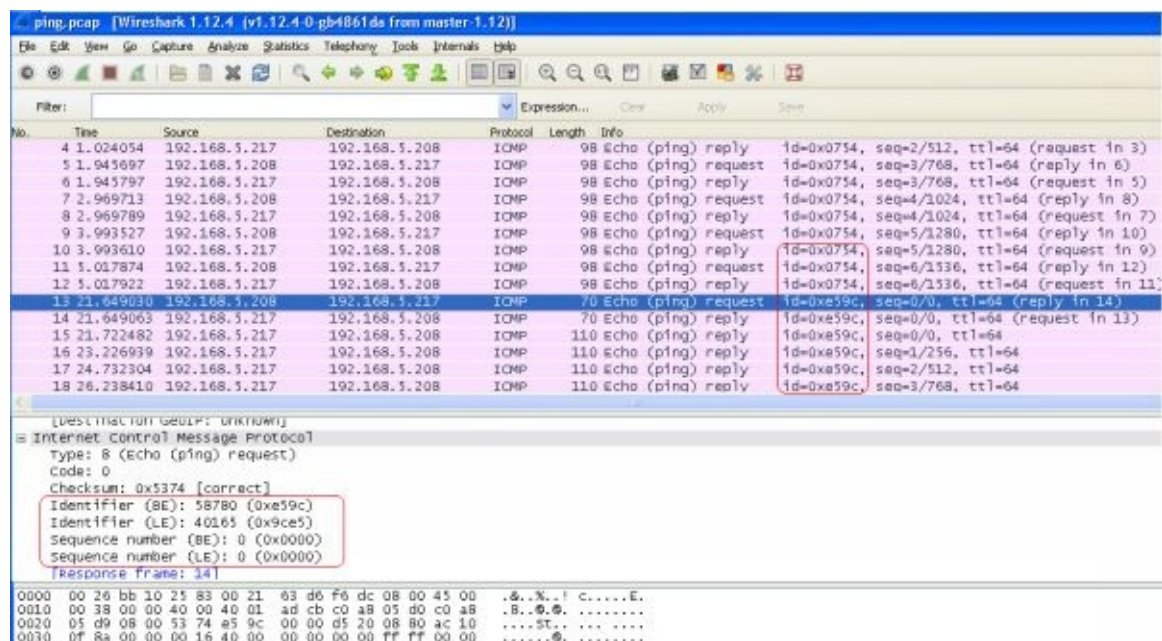
The well-known signature for pTunnel according to Snort IDS and Cisco is Hex d5 20 08 80. By filtering for hex values for Hex values of d5 20 06 80 in Wireshark we can see that frames 13 onwards have this pTunnel signature in the payload indicating that this is indeed a pTunnel channel.



number (magic number d5 20 08 80). Therefore, if we look for a packet with HEX value of d5 20 08 80 the first packet we find is in frame 13 and following the magic number is the IP address of the true destination (the tunnel end-point) which is ac 10 0f 8a. This converts via a hex to decimal convertor to 172.16.15.138

## 6. What is the session identifier for each packet? (answer in hex, 2 bytes)

The session identifier for each packet is dependent on the protocol flows, initially prior to the tunnel being established the session identifier was determined by the BE identifier value in the ICMP packet or it can be seen in the Wireshark frame, as id=0x0754. However at frame 13, as the covert pTunnel channel is established the identifier changes to id=0xe59c, which is shown in the screenshot below.

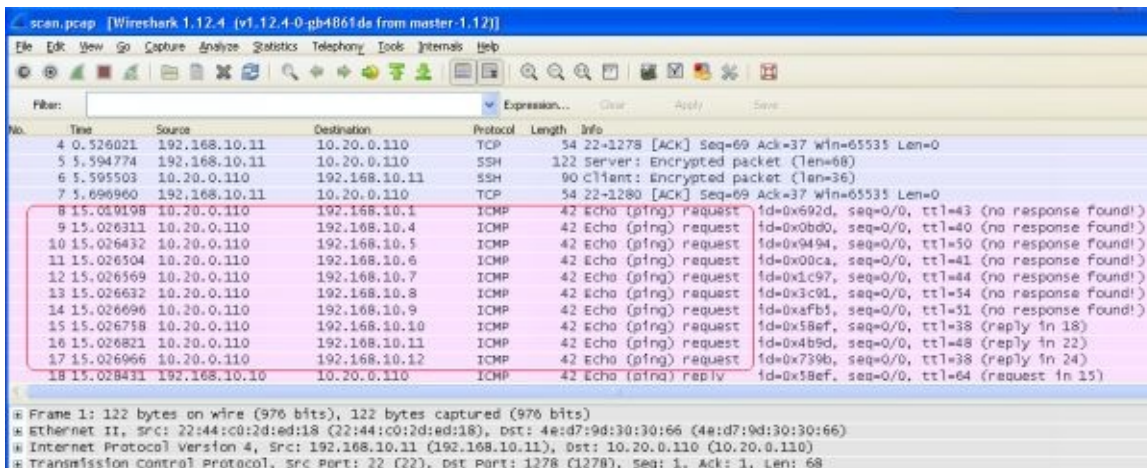


## B - Scan.pcap

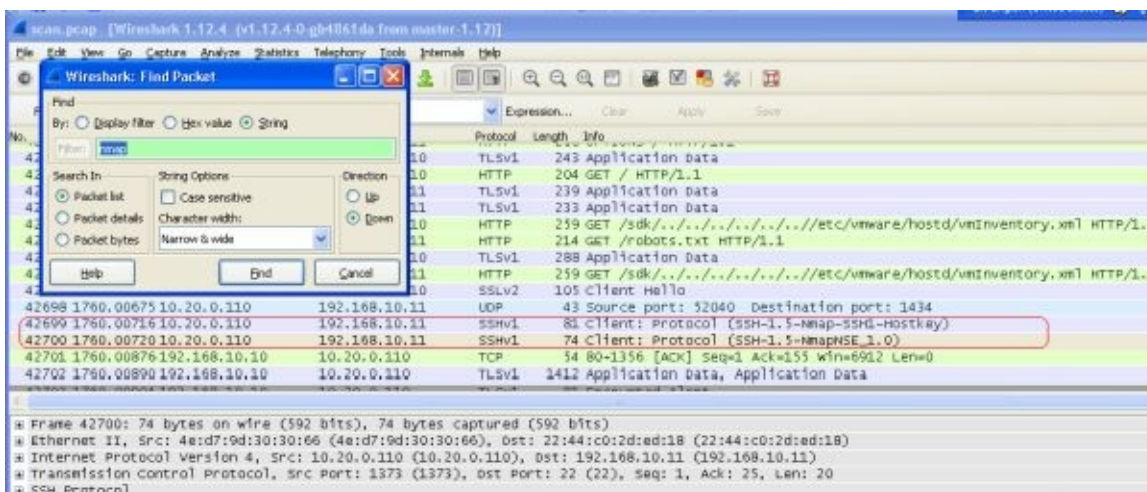
### 1. What tool is generating this traffic?

This is the output from a network discovery scanner, which can step through every address in a network/subnet in sequence to discover hosts within that network range. The tool can discover what hosts are present and on what addresses.





This particular tool appears to be Nmap as it sends reset/ack back to the host to prevent the three-way handshake completing this is a typical network signature for Nmap. To confirm this you can string search for Nmap within Wireshark and Nmap is revealed in frame 42699 and 42700.



## 2. What is the frame that indicates the something strange might be going on?

The first indicate of unusual behaviour is the obvious ping request scans starting from 10.20.0.110 to hosts on the 192.168.10.0 subnet starting at frame 8.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.526021	192.168.10.11	10.20.0.110	TCP	54	22→1278 [ACK] Seq=69 Ack=37 Win=65535 Len=0
5	5.594774	192.168.10.11	10.20.0.110	SSH	122	Server: Encrypted packet (len=68)
6	5.595503	10.20.0.110	192.168.10.11	SSH	90	Client: Encrypted packet (len=36)
7	5.696960	192.168.10.11	10.20.0.110	TCP	54	22→1280 [ACK] Seq=69 Ack=37 Win=65535 Len=0
8	15.059198	10.20.0.110	192.168.10.1	ICMP	42	Echo (ping) request Id=0x692d, seq=0/0, ttl=43 (no response found!)
9	15.026311	10.20.0.110	192.168.10.4	ICMP	42	Echo (ping) request Id=0x0bd0, seq=0/0, ttl=40 (no response found!)
10	15.026432	10.20.0.110	192.168.10.5	ICMP	42	Echo (ping) request Id=0x9494, seq=0/0, ttl=50 (no response found!)
11	15.026504	10.20.0.110	192.168.10.6	ICMP	42	Echo (ping) request Id=0x00ca, seq=0/0, ttl=41 (no response found!)
12	15.026569	10.20.0.110	192.168.10.7	ICMP	42	Echo (ping) request Id=0x1c97, seq=0/0, ttl=44 (no response found!)
13	15.026632	10.20.0.110	192.168.10.8	ICMP	42	Echo (ping) request Id=0x3c01, seq=0/0, ttl=54 (no response found!)
14	15.026696	10.20.0.110	192.168.10.9	ICMP	42	Echo (ping) request Id=0xafb5, seq=0/0, ttl=51 (no response found!)
15	15.026758	10.20.0.110	192.168.10.10	ICMP	42	Echo (ping) request Id=0x58ef, seq=0/0, ttl=38 (reply in 18)
16	15.026821	10.20.0.110	192.168.10.11	ICMP	42	Echo (ping) request Id=0x4b8d, seq=0/0, ttl=48 (reply in 22)
17	15.026966	10.20.0.110	192.168.10.12	ICMP	42	Echo (ping) request Id=0x739b, seq=0/0, ttl=38 (reply in 24)
18	15.028431	192.168.10.10	10.20.0.110	ICMP	42	Echo (ping) reply Id=0x58ef, seq=0/0, ttl=64 (request in 15)

Frame 1: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)  
 Ethernet II, Src: 22:44:c0:2d:ed:18 (22:44:c0:2d:ed:18), Dst: 4e:d7:9d:30:30:66 (4e:d7:9d:30:30:66)  
 Internet Protocol Version 4, Src: 192.168.10.11 (192.168.10.11), Dst: 10.20.0.110 (10.20.0.110)  
 Transmission Control Protocol, Src Port: 22 (22), Dst Port: 1278 (1278), Seq: 1, Ack: 1, Len: 68

### 3. What does this frame constitute the beginning of? (What type of Scan?)

This scanning behaviour, of stepping through a subnet in sequence is the typical signature of a Network discovery scan. Nmap is a network-scanning tool that is used to discover hosts on a network or subnet. Nmap can use many different scanning techniques but in the initial scan shown it is the beginning of a simple ICMP ping request/reply scan to discover what IP addresses have active hosts listening and responding.

**4. The ‘miscreant’ then runs two scans beginning just after six minutes and 24 minutes into the trace, however, these traces weren’t to his/her liking as they were too slow. On the following scans, a switch was removed from the command to improve the speed, what was this switch (just the letters, case-sensitive)?**

We cannot answer this question by just tracing through the scan.pcap file in Wireshark, there is just too much noise – normal network traffic - and simultaneous Nmap scans running for us to isolate and identify individual Nmap traffic.

Therefore, what we have to do is try to filter and isolate individual Nmap scans by applying display filters to Wireshark.

The display filters that we will apply in sequence will conform to specific and commonly used Nmap scan criteria. Unfortunately, there are many Nmap configurations possible but if we start with the most commonly applied scan switches, we should be able to identify which ones were used due to the 6 minute and 24 - minute information that we have. This is our starting point.

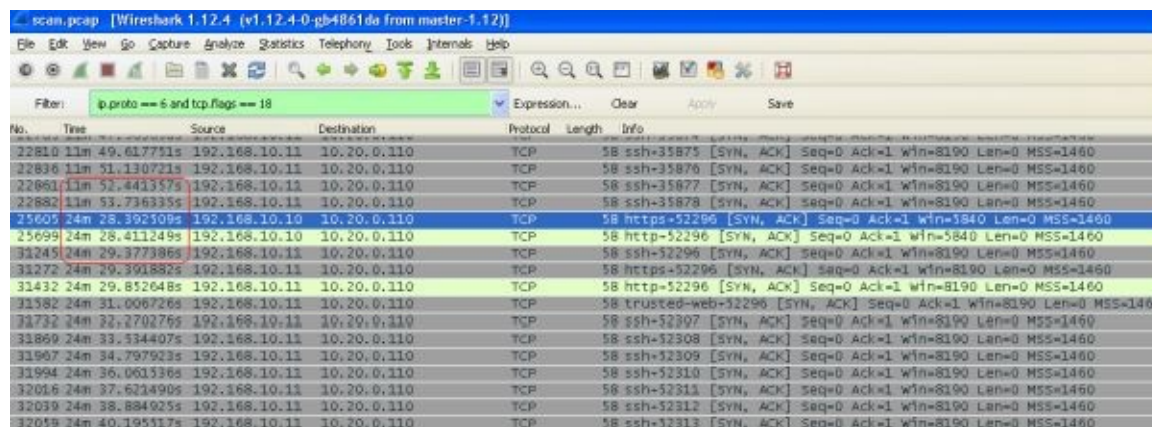
In order to do this however we must understand the common Nmap configurations used



when scanning a network.

Nmap -sT (this is the TCP connect or Vanilla scan, which established a full three-way handshake)

The display filter we use in Wireshark is (ip.dst == 10.0.20.110 and ip.proto == 6 and tcp.flags == 18 )



No.	Time	Source	Destination	Protocol	Length	Info
22810	11m 49.617751s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=35875 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
22836	11m 51.130721s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=35876 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
22861	11m 52.441357s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=35877 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
22882	11m 53.736335s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=35878 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
25605	24m 28.392509s	192.168.10.10	10.0.20.0.110	TCP	58	https=52296 [SYN, ACK] Seq=0 Ack=1 Win=1840 Len=0 MSS=1460
25699	24m 28.411249s	192.168.10.10	10.0.20.0.110	TCP	58	http=52296 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
31245	24m 29.377386s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=52296 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
31272	24m 29.391882s	192.168.10.11	10.0.20.0.110	TCP	58	https=52296 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
31432	24m 29.852648s	192.168.10.11	10.0.20.0.110	TCP	58	http=52296 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
31582	24m 31.006726s	192.168.10.11	10.0.20.0.110	TCP	58	trusted-web=52296 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
31732	24m 32.270276s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=52307 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
31860	24m 33.334407s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=52308 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
31907	24m 34.797923s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=52309 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
31994	24m 36.061536s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=52310 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
32016	24m 37.621490s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=52311 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
32039	24m 38.884925s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=52312 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
32059	24m 40.195517s	192.168.10.11	10.0.20.0.110	TCP	58	ssh=52313 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460

By scanning through the files, till we reach the 24 minute reference, we discover that a Nmap scan forced a SYN, ACK response from the scanned host in response to a SYNC from the Nmap host, which indeed start just after 24 minutes, as can be seen in the timestamps above.

The next Nmap scan to look for is the SYNC stealth scan, -sS, and we can filter the pcap file in Wireshark using this filter:

ip.proto == 6 and tcp.flags == 2

scan.pcap [Wireshark 1.12.4 (v1.12.4-0-gb4861da from master.1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: `ip.proto == 6 and tcp.flags == 2` Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
17516	10m 36.561817s	10.20.0.110	192.168.10.10	TCP	58	42667→mysql [SYN] Seq=0 win=1024 Len=0 MSS=1460
17517	10m 36.561972s	10.20.0.110	192.168.10.54	TCP	58	42655→http [SYN] Seq=0 win=1024 Len=0 MSS=1460
17550	11m 14.574853s	10.20.0.110	192.168.10.11	TCP	58	35843→telnet [SYN] Seq=0 win=1024 Len=0 MSS=1460
17551	11m 15.576962s	10.20.0.110	192.168.10.53	TCP	58	35843→telnet [SYN] Seq=0 win=1024 Len=0 MSS=1460
17553	11m 15.592604s	10.20.0.110	192.168.10.10	TCP	58	35843→telnet [SYN] Seq=0 win=1024 Len=0 MSS=1460
17554	11m 15.592669s	10.20.0.110	192.168.10.53	TCP	58	35843→imap [SYN] Seq=0 win=1024 Len=0 MSS=1460
17555	11m 15.592734s	10.20.0.110	192.168.10.54	TCP	58	35843→telnet [SYN] Seq=0 win=1024 Len=0 MSS=1460
17556	11m 15.593034s	10.20.0.110	192.168.10.10	TCP	58	35843→imap [SYN] Seq=0 win=1024 Len=0 MSS=1460
17557	11m 15.593132s	10.20.0.110	192.168.10.53	TCP	58	35843→ftp [SYN] Seq=0 win=1024 Len=0 MSS=1460
17558	11m 15.593244s	10.20.0.110	192.168.10.54	TCP	58	35843→imap [SYN] Seq=0 win=1024 Len=0 MSS=1460
17559	11m 15.593445s	10.20.0.110	192.168.10.10	TCP	58	35843→ftp [SYN] Seq=0 win=1024 Len=0 MSS=1460
17560	11m 15.593536s	10.20.0.110	192.168.10.53	TCP	58	35843→ssh [SYN] Seq=0 win=1024 Len=0 MSS=1460
17561	11m 15.593619s	10.20.0.110	192.168.10.54	TCP	58	35843→ftp [SYN] Seq=0 win=1024 Len=0 MSS=1460
17571	11m 15.608221s	10.20.0.110	192.168.10.10	TCP	58	35843→ssh [SYN] Seq=0 win=1024 Len=0 MSS=1460
17572	11m 15.608299s	10.20.0.110	192.168.10.53	TCP	58	35843→smux [SYN] Seq=0 win=1024 Len=0 MSS=1460
17573	11m 15.608376s	10.20.0.110	192.168.10.54	TCP	58	35843→ssh [SYN] Seq=0 win=1024 Len=0 MSS=1460
17574	11m 15.608679s	10.20.0.110	192.168.10.10	TCP	58	35843→smux [SYN] Seq=0 win=1024 Len=0 MSS=1460

[Stream index: 5965]  
 [TCP Segment Len: 0]  
 Sequence number: 0 (relative sequence number)  
 Acknowledgment number: 0  
 Header length: 24 bytes  
 ... 0000 0000 0010 = Flags: 0x002 (SYN)  
 Window size value: 1024  
 [calculated window size: 1024]

Another Nmap switch we could try is Nmap -sP, using a Wireshark filter of `Ip.proto == 1`

This is the ICMP echo request scan, which we saw earlier at the very beginning of the pcap trace. However it runs at 14 seconds to 1 minute, before restarting at 4 minutes and running to 11 minutes, so that again is not the scan we are looking for.

Another common Nmap switch config is -sU, which uses UDP as the scanning protocol.

scan.pcap [Wireshark 1.12.4 (v1.12.4-0-gb4861da from master.1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: `ip.proto == 17 and ip.len == 28` Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
17514	10m 35.750125s	10.20.0.110	192.168.10.10	T.38	42	[Malformed Packet]
17520	10m 37.372719s	10.20.0.110	192.168.10.10	T.38	42	[Malformed Packet]
17521	10m 37.373107s	192.168.10.10	10.20.0.110	ICMP	70	Destination unreachable (Port unreachable)
17522	10m 38.183771s	10.20.0.110	192.168.10.10	T.38	42	[Malformed Packet]
17523	10m 38.184639s	192.168.10.10	10.20.0.110	ICMP	70	Destination unreachable (Port unreachable)
17524	10m 38.994935s	10.20.0.110	192.168.10.10	T.38	42	[Malformed Packet]
17525	10m 38.995841s	192.168.10.10	10.20.0.110	ICMP	70	Destination unreachable (Port unreachable)
17526	10m 39.806179s	10.20.0.110	192.168.10.10	T.38	42	[Malformed Packet]
17527	10m 39.807042s	192.168.10.10	10.20.0.110	ICMP	70	Destination unreachable (Port unreachable)
17528	10m 40.617287s	10.20.0.110	192.168.10.10	T.38	42	[Malformed Packet]
17529	10m 40.618108s	192.168.10.10	10.20.0.110	ICMP	70	Destination unreachable (Port unreachable)
17530	10m 41.429242s	10.20.0.110	192.168.10.10	T.38	42	[Malformed Packet]
17531	10m 41.429988s	192.168.10.10	10.20.0.110	ICMP	70	Destination unreachable (Port unreachable)
17532	10m 42.239761s	10.20.0.110	192.168.10.10	T.38	42	[Malformed Packet]
17533	10m 42.240721s	192.168.10.10	10.20.0.110	ICMP	70	Destination unreachable (Port unreachable)
17534	10m 43.050943s	10.20.0.110	192.168.10.10	T.38	42	[Malformed Packet]
17535	10m 43.051808s	192.168.10.10	10.20.0.110	ICMP	70	Destination unreachable (Port unreachable)

Frame 10127: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)  
 Ethernet II, Src: 4e:d7:9d:30:30:66 (4e:d7:9d:30:30:66), Dst: 22:44:c0:2d:ed:18 (22:44:c0:2d:ed:18)  
 Internet Protocol Version 4, Src: 10.20.0.110 (10.20.0.110), Dst: 192.168.10.11 (192.168.10.11)  
 User Datagram Protocol, Src Port: 42627 (42627), Dst Port: 21298 (21298)  
 Source Port: 42627 (42627)  
 Destination Port: 21298 (21298)

What we can see in the filtered -sU traffic is that the scan starts at 4 minutes and runs till it is stopped and the -sU switch removed at 10 min, 43 seconds.

However if we are looking for a switch being removed after the 24 min scan starts to run, and performance is judged to be poor, then we have to look past the 24 min threshold.

What we do find though is that at 27 minutes a new scan starts using the Nmap -sS stealth switch. Prior to the 27 minute the Nmap scan is using a SYNC sent to the hosts and the host port if closed returns a RST, ACK as shown below, should a port be open, then the host returns a SYN, ACK.

### A Closed Port:



The screenshot shows a Wireshark capture of a scan.pcap file. The filter is set to 'tcp.stream eq 11139'. The packet list shows two TCP packets: a SYN packet at 23635.24m and a RST, ACK packet at 23688.24m.

No.	Time	Source	Destination	Protocol	Length	Info
23635	24m 28.402037s	10.20.0.110	192.168.10.10	TCP	58	52296->imap [SYN] Seq=0 Win=1024 Len=0 MSS=1460
23688	24m 28.407201s	192.168.10.10	10.20.0.110	TCP	54	imap->52296 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

### An Open Port:

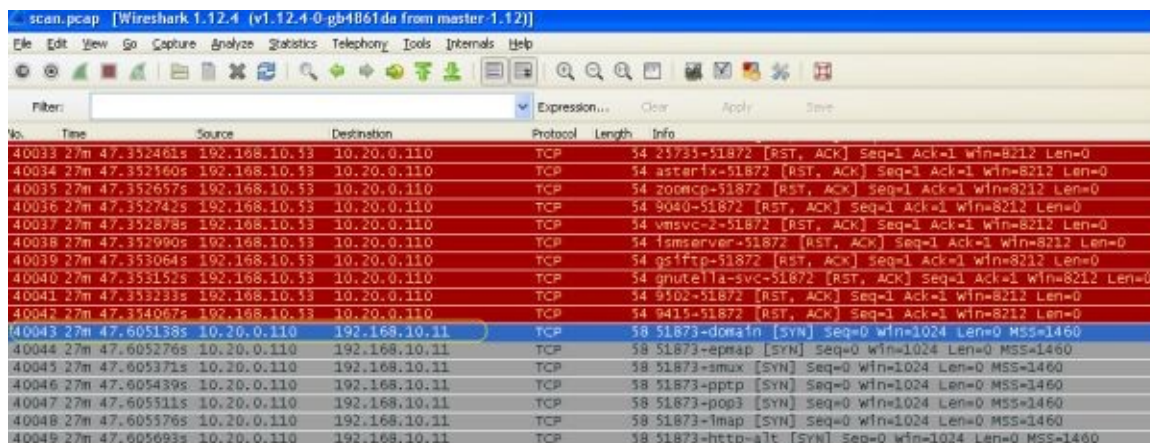


The screenshot shows a Wireshark capture of a scan.pcap file. The filter is set to 'tcp.stream eq 11154'. The packet list shows three TCP packets: a SYN packet at 23656.24m, a SYN, ACK packet at 23699.24m, and a RST packet at 23700.24m.

No.	Time	Source	Destination	Protocol	Length	Info
23656	24m 28.403633s	10.20.0.110	192.168.10.10	TCP	58	52296->http [SYN] Seq=0 Win=1024 Len=0 MSS=1460
23699	24m 28.411249s	192.168.10.10	10.20.0.110	TCP	58	http->52296 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
23700	24m 28.411286s	10.20.0.110	192.168.10.10	TCP	54	52296->http [RST] Seq=1 Win=0 Len=0

This is typical of the Nmap -sT full handshake TCP connection scan, which started at 24 minutes.

However, at 27 minutes we see this;



The screenshot shows a Wireshark capture of a scan.pcap file. The filter is set to 'tcp.stream eq 11154'. The packet list shows a series of RST, ACK packets from various ports to the scanner at 27 minutes, followed by SYN packets from the scanner to the host at 27 minutes.

No.	Time	Source	Destination	Protocol	Length	Info
40033	27m 47.352461s	192.168.10.53	10.20.0.110	TCP	54	51735->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40034	27m 47.352360s	192.168.10.53	10.20.0.110	TCP	54	asterix->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40035	27m 47.352657s	192.168.10.53	10.20.0.110	TCP	54	zooecp->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40036	27m 47.352742s	192.168.10.53	10.20.0.110	TCP	54	9040->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40037	27m 47.352878s	192.168.10.53	10.20.0.110	TCP	54	vmsvc-2->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40038	27m 47.352990s	192.168.10.53	10.20.0.110	TCP	54	lmsrserver->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40039	27m 47.353064s	192.168.10.53	10.20.0.110	TCP	54	gsiftp->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40040	27m 47.353152s	192.168.10.53	10.20.0.110	TCP	54	gnute/la-svc->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40041	27m 47.353233s	192.168.10.53	10.20.0.110	TCP	54	9502->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40042	27m 47.354067s	192.168.10.53	10.20.0.110	TCP	54	9415->51872 [RST, ACK] Seq=1 Ack=1 Win=8212 Len=0
40043	27m 47.605138s	10.20.0.110	192.168.10.11	TCP	58	51873->domain [SYN] Seq=0 Win=1024 Len=0 MSS=1460
40044	27m 47.605276s	10.20.0.110	192.168.10.11	TCP	58	51873->epmap [SYN] Seq=0 Win=1024 Len=0 MSS=1460
40045	27m 47.605371s	10.20.0.110	192.168.10.11	TCP	58	51873->smux [SYN] Seq=0 Win=1024 Len=0 MSS=1460
40046	27m 47.605439s	10.20.0.110	192.168.10.11	TCP	58	51873->pptp [SYN] Seq=0 Win=1024 Len=0 MSS=1460
40047	27m 47.605511s	10.20.0.110	192.168.10.11	TCP	58	51873->pop3 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
40048	27m 47.605576s	10.20.0.110	192.168.10.11	TCP	58	51873->imap [SYN] Seq=0 Win=1024 Len=0 MSS=1460
40049	27m 47.605693s	10.20.0.110	192.168.10.11	TCP	58	51873->http-alt [SYN] Seq=0 Win=1024 Len=0 MSS=1460



The thing to note here is the difference in the timestamp, there is a noticeable gap between packets being sent between frame 40042 and 40043, which could indicate a change of criteria. Furthermore, when we follow the stream we get this:

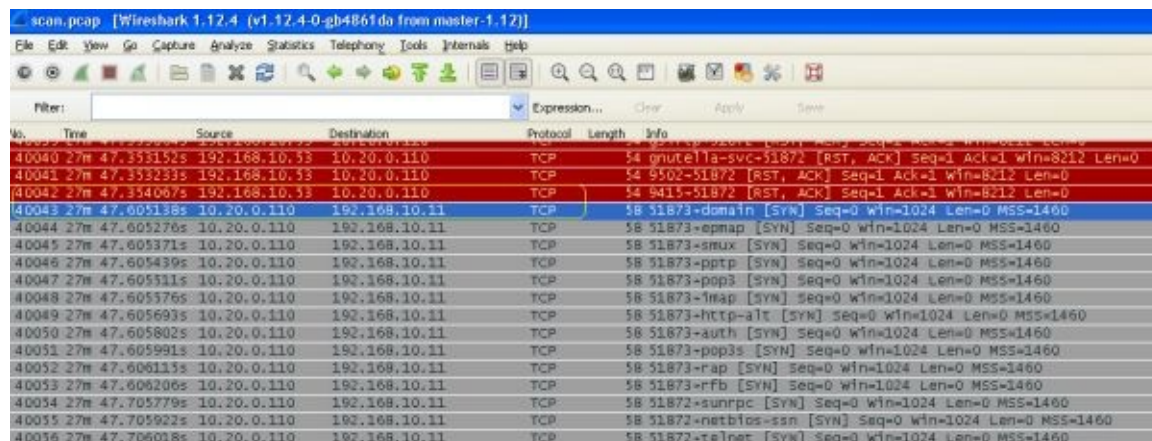


No.	Time	Source	Destination	Protocol	Length	Info
40043	27m 47.605138s	10.20.0.110	192.168.10.11	TCP	58	51873->domain [SYN] Seq=0 win=1024 Len=0 MSS=1460

For example, a SYNC is sent but if the port is closed the packet is dropped, there is no RST, ACK returned and this speed things up considerably.

## 5. What switch was added to the final scan (case-sensitive)?

The final switch applied was Nmap -sS at 27 minutes and 47 seconds, as shown below.



No.	Time	Source	Destination	Protocol	Length	Info
40043	27m 47.605138s	10.20.0.110	192.168.10.11	TCP	58	51873->domain [SYN] Seq=0 win=1024 Len=0 MSS=1460
40044	27m 47.605276s	10.20.0.110	192.168.10.11	TCP	58	51873->epmap [SYN] Seq=0 win=1024 Len=0 MSS=1460
40045	27m 47.605371s	10.20.0.110	192.168.10.11	TCP	58	51873->smux [SYN] Seq=0 win=1024 Len=0 MSS=1460
40046	27m 47.605439s	10.20.0.110	192.168.10.11	TCP	58	51873->pptp [SYN] Seq=0 win=1024 Len=0 MSS=1460
40047	27m 47.605511s	10.20.0.110	192.168.10.11	TCP	58	51873->pop3 [SYN] Seq=0 win=1024 Len=0 MSS=1460
40048	27m 47.605576s	10.20.0.110	192.168.10.11	TCP	58	51873->imap [SYN] Seq=0 win=1024 Len=0 MSS=1460
40049	27m 47.605693s	10.20.0.110	192.168.10.11	TCP	58	51873->http-alt [SYN] Seq=0 win=1024 Len=0 MSS=1460
40050	27m 47.605802s	10.20.0.110	192.168.10.11	TCP	58	51873->auth [SYN] Seq=0 win=1024 Len=0 MSS=1460
40051	27m 47.605991s	10.20.0.110	192.168.10.11	TCP	58	51873->pop3s [SYN] Seq=0 win=1024 Len=0 MSS=1460
40052	27m 47.606113s	10.20.0.110	192.168.10.11	TCP	58	51873->rap [SYN] Seq=0 win=1024 Len=0 MSS=1460
40053	27m 47.606206s	10.20.0.110	192.168.10.11	TCP	58	51873->rfb [SYN] Seq=0 win=1024 Len=0 MSS=1460
40054	27m 47.705779s	10.20.0.110	192.168.10.11	TCP	58	51872->sunrpc [SYN] Seq=0 win=1024 Len=0 MSS=1460
40055	27m 47.705922s	10.20.0.110	192.168.10.11	TCP	58	51872->netbios-ssn [SYN] Seq=0 win=1024 Len=0 MSS=1460
40056	27m 47.706018s	10.20.0.110	192.168.10.11	TCP	58	51872->telnet [SYN] Seq=0 win=1024 Len=0 MSS=1460

Therefore the last scan that is done is to remove the switch for client reply of RST, ACK (-sT), which is part of the Nmap -sT full connection scan at frame 40043 at 27 minutes and 47 seconds.

However, there is also a case for an xmas scan as one is detected starting just as the trace ends as shown below:

The image shows a Wireshark packet capture titled 'scan.pcap [Wireshark 1.12.4 (v1.12.4-0-gb4861da from master 1.12)]'. The filter is 'ip.proto == 6 and tcp.flags == 41'. The table below represents the data shown in the packet list pane.

No.	Time	Source	Destination	Protocol	Length	Info
42467	29s 16.391827s	10.20.0.110	192.168.10.53	TCP	74	52097→tcpmux [FIN, PSH, URG] Seq=1 win=2147450880 urg=0 Len=0 WS=3
42468	29s 16.391915s	10.20.0.110	192.168.10.54	TCP	74	52097→tcpmux [FIN, PSH, URG] Seq=1 win=2147450880 urg=0 Len=0 WS=3
42469	29s 16.347547s	10.20.0.110	192.168.10.10	TCP	74	52097→tcpmux [FIN, PSH, URG] Seq=1 win=2147450880 urg=0 Len=0 WS=3
42489	29s 16.347708s	10.20.0.110	192.168.10.11	TCP	74	52097→30115 [FIN, PSH, URG] Seq=1 win=2147450880 urg=0 Len=0 WS=32
42531	29s 18.995684s	10.20.0.110	192.168.10.53	TCP	74	[TCP Previous segment not captured] 52097→tcpmux [FIN, PSH, URG] Seq=1 win=2147450880 urg=0 Len=0 WS=32
42532	29s 18.995916s	10.20.0.110	192.168.10.54	TCP	74	[TCP Previous segment not captured] 52097→tcpmux [FIN, PSH, URG] Seq=1 win=2147450880 urg=0 Len=0 WS=32
42563	29s 19.152726s	10.20.0.110	192.168.10.11	TCP	74	52097→36345 [FIN, PSH, URG] Seq=1 win=2147450880 urg=0 Len=0 WS=32
47259	30s 55.061611s	10.20.0.110	192.168.10.100	TCP	74	41685→34332 [FIN, PSH, URG] Seq=1 win=2147450880 urg=0 Len=0 WS=32
47299	30s 57.027236s	10.20.0.110	192.168.10.100	TCP	74	41685→30672 [FIN, PSH, URG] Seq=1 win=2147450880 urg=0 Len=0 WS=32

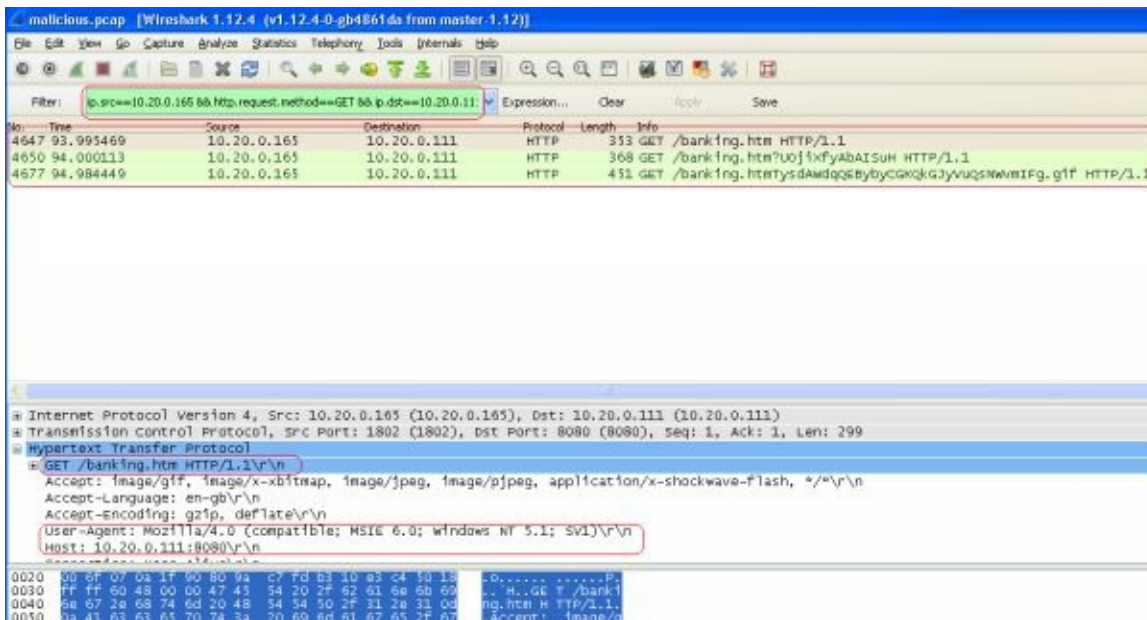
However, there is insufficient data to consider this a genuine scan as the pcap ends at 30 mins just as this scan starts. The evidence for this being an Nmap xmas lies in the fact that the TCP flags were set for FIN, PSH & URG which is a classic fingerprint for the Nmap xmas scan.

## C – Malicious.pcap

### 1. What was the complete URI of the original web request that led to the client being compromised?

The first thing we have to do is get some clue where to start looking here for suspicious activity. We can do this using an IDS that can take a pcap file as an input and analyse the file just as if it was reading the packets from the wire. Or, we can use Wireshark to filter out and display any requested downloads. For example if we run the filter: (ip.src==10.20.0.165) && (http.request.method==GET) && (ip.dst==10.20.0.111) we should be able to see what files have been downloaded.

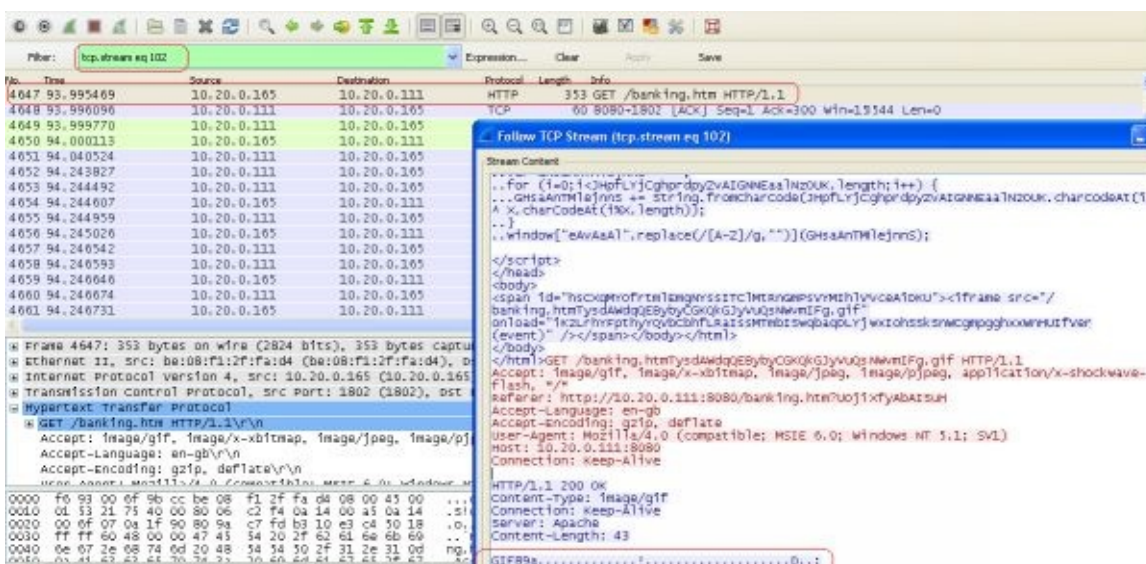




The result is that we have at frame 4647 the URL <http://10.20.0.111:8080/banking.htm> and this gives us a starting point for our analysis.

## 2. What file type was requested in the final web request to the malicious server?

The final request was for a gif file gif89a as can be seen when we follow the TCP sequence stream (tcp.stream eq 102) between 10.20.0.165 and 10.20.0.111 starting at the initial compromise URL at frame 4647.



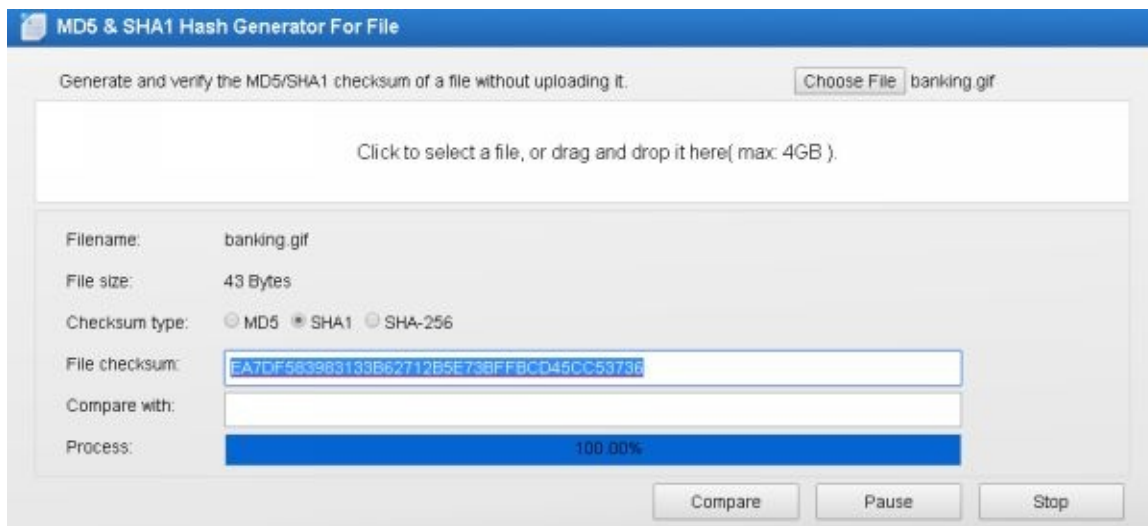
## 3. What is the sha1 hash of the afore-mentioned file?

There is no information on the SHA1 hash for this file – GIF89a – but we can reconstruct it using the data in the pcap.

The sha1 hash of the file gif89a can be calculated by using the data in the packet:

GIF89a.....!.....,.....D.;

then convert it to binary and save it as banking.gif and using an online tools - an online checksum calculator - we can calculate the SHA1 value by using an online SHA1 convertor, such as Online MD5 :



MD5 & SHA1 Hash Generator For File

Generate and verify the MD5/SHA1 checksum of a file without uploading it. Choose File banking.gif

Click to select a file, or drag and drop it here( max: 4GB ).

Filename: banking.gif

File size: 43 Bytes

Checksum type: ☐ MD5 ☒ SHA1 ☐ SHA-256

File checksum: EA7DF583983133B62712B5E738FFBCD45CC53736

Compare with:

Process: 100.00%

Compare Pause Stop

Therefore the SHA1 of the file GIF68a is :

EA7DF583983133B62712B5E738FFBCD45CC53736

#### **4. What is the number of the first frame that indicates that the client has been compromised?**

If we take our starting point as being frame 4647 the time of the compromise, then we can look to see what developed after that point.

The first frame that shows a malicious payload is frame 4722 as 10.20.0.165 (client) it is connecting to destination port 4444 on 10.20.0.111, which is a well-known port for Metasploit and Trojans using an HTTP reverse connection exploit, which will open up a backdoor into 10.20.0.165.



details of the stream content taken from the data payload in frame 4726 that alerts us to the fact that a script is executable:

‘This program cannot be run in DOS mode’.

The next thing we can do now is to export all objects that have been downloaded by going to Wireshark -> file -> export objects -> http. This will provide us with a list of all objects downloaded using HTTP. We can then look at the banking files that have been downloaded and processed and check them against an anti-virus or run sha1 hash checks on them. Alternatively, we can export our Wireshark tcp-stream to a pcap file and then analyse that file.

The steps we need to take are:

- In Wireshark Go to frame 4722 – Analyse – Follow tcp stream
- Then when tcp-stream 104 is displayed, go to File, Export specified packets
- Save the exported packets to a file called analysis.pcap

At this stage, a file analysis.pcap will contain only the packets from the tcp-stream relevant to the malicious file download.

The next stage is to verify if the file is malicious and we can do this by running the file through an online anti-virus software application such as VirusTotal.





Analysis	File detail	Additional information	Comments	Votes
----------	-------------	------------------------	----------	-------

Intrusion Detection System	Result
Snort	1 alert
Suricata	4 alerts

Antivirus	Result	Update
Ad-Aware	Gen:Trojan.Heur.GM.09C4000000	20150316
Avast	Multi:Swroot-A [Trj]	20150317
BitDefender	Gen:Trojan.Heur.GM.09C4000000	20150317
Emsisoft	Gen:Trojan.Heur.GM.09C4000000 (B)	20150317
F-Secure	Gen:Trojan.Heur.GM.09C4000000	20150317
GData	Gen:Trojan.Heur.GM.09C4000000	20150317
MicroWorld-eScan	Gen:Trojan.Heur.GM.09C4000000	20150317
Qihoo-360	Trojan.Generic	20150317
Symantec	Meterpreter	20150317

However, using the directly exported files from the objects download from Wireshark, the file banking.htm%3fUOjiXfyAbAISuH gave us similar results.

SHA256:	150f7eab03c0484d184b4abb6ac45472ee0e59c1f848ae7213be77c2b3ba22f
File name:	banking.text
Detection ratio:	11 / 55
Analysis date:	2014-12-07 22:08:37 UTC ( 3 months, 1 week ago )

Analysis	Additional information	Comments	Votes
----------	------------------------	----------	-------

Antivirus	Result	Update
Avast	JS:CVE-2010-0247-S [Exploit]	20141207
Avira	EXP/CVE-2010-0247_A.3	20141207
Comodo	UnclassifiedMalware	20141207
ESET-NOD32	JS/Kryptik.AFR	20141207
GData	HTML.Trojan.Agent.Y1HMUF	20141207
Ikarus	Trojan-Downloader.JS.Agent	20141207

The follow further details were available through the details tab:

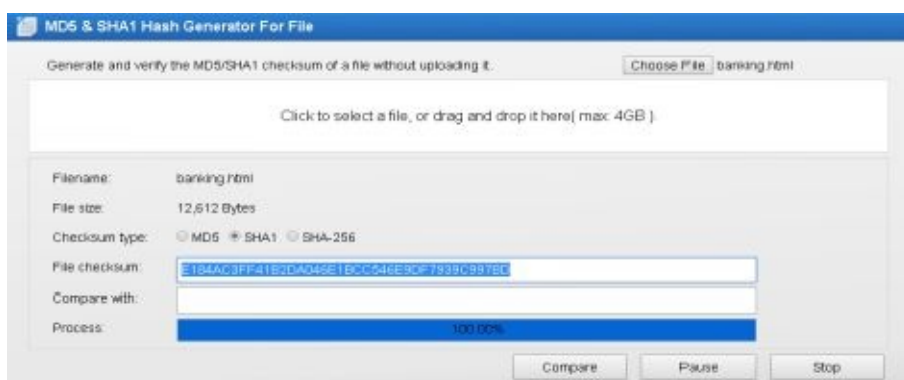




Therefore, after our analysis we can confidently say that the type of file downloaded was a windows executable or DLL file by the name of Banking.htm, which is known malware.

## 6. What is the sha1 hash of the malicious file?

The sha1 hash for the malicious file banking.htm which we exported earlier from Wireshark is:



E184AC3FF41B2DA046E1BCC546E9DF7939C997BD

If we now do an internet search for files with this SHA1 hash we are directed to:

<https://malwr.com/analysis/OGIxMzhiODRjNjk5NDhkNThkNjNiNzU0NTUwNTUxNDM>

The Malware site has this information:

FILE NAME	banking.htm.8B5A693B.html
FILE SIZE	12612 bytes
FILE TYPE	HTML document, ASCII text, with very long lines

MD5	a576776feb36992086fe9a7d2662836
SHA1	e184ac3ff41b2da046e1bcc546e9df7939c997bd

The Malware site also identifies as having these signatures:

- Identified as malware by at least one Anti-virus on Virus Total as being malicious
- Steal private information from local internet browsers
- Installs itself for autorun at Windows startup

## 7. What vulnerable software is exploited?

The vulnerable software exploited is the agent in the browser Mozilla 4.0 which we can see in frame 4650 with the data showing user-agent mozilla 4.0 in Internet Explorer 6.0.

```

* Internet Protocol Version 4, Src: 10.20.0.165 (10.20.0.165), Dst: 10.20.0.111 (10.20.0.111)
* Transmission Control Protocol, Src Port: 1802 (1802), Dst Port: 8080 (8080), Seq: 1, Ack: 1, Len: 299
* Hypertext Transfer Protocol
  * GET /banking.htm HTTP/1.1\r\n
    Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*\r\n
    Accept-Language: en-gb\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1; SV1)\r\n
    Host: 10.20.0.111:8080\r\n
    Connection: close\r\n
0020  50 6f 07 0a 1f 90 80 9a c7 fd b3 10 43 c4 50 10  10.....P
0030  ff ff 60 48 00 00 47 45 54 20 2f 62 81 6a 6b 59  ..M..GE T /bank
0040  6e 67 2e 68 74 6d 20 48 54 54 50 2f 31 2e 31 0d  ng.htm H TTP/1.1
0050  00 41 63 63 65 70 74 3a 20 69 6d 61 67 65 7f 00  accept: image/g

```

As we saw with the earlier Anti-Virus and Malware analysis the vulnerabilities lie with SHELLCODE and POLICY PE EXE or DLL Windows file downloads.

**Snort alerts**
Sourcetime VRT ruleset

INDICATOR-SHELLCODE x86 inc ebx NOOP (Executable Code was Detected)

**Suricata alerts**
Emerging Threats ETPro ruleset

ET SHELLCODE Possible Call with No Offset TCP Shellcode (Executable Code was Detected)

GPL SHELLCODE x86 inc ebx NOOP (Executable Code was Detected)

ET SHELLCODE Possible Call with No Offset TCP Shellcode (Executable Code was Detected)

ET POLICY PE EXE or DLL Windows file download (Potential Corporate Privacy Violation)

## 8. Can you give the corresponding CVE security bulletin that covers the vulnerability here that was exploited (answer in form of CVE-\$year-\$number).

The CVE security bulletin for this exploit was a well know Internet Explorer 5 exploit known as CVE-2010-0249 and is also known as EXP/CVE-2010-0247.A.3, JS/Kryptik.AFR Trojan amongst other names by online virus scanner VirusTotal. Other

anti-virus products also consider it to be based on the CVE-2010-0247 exploit.

Avast	JS:CVE-2010-0247-S [Exploit]	20141207
Avira	EXP/CVE-2010-0247.A.3	20141207

**9. From the capture, it is clear that the attacker gets a certain form of access (i.e. the interface), what (type of) access does the attacker “get” on the client?**

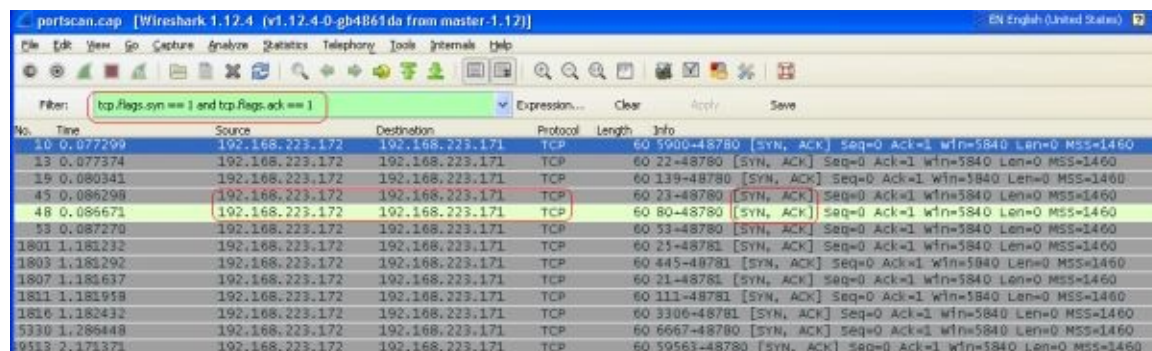
JS/Exploit.CVE-2010-0249.F is viewed as a malicious Trojan that can change the settings of the important files and registry on the host computer. Users may encounter several unfamiliar things when they start up their system. The trojan will run automatically in the background taking advantage of its auto-start feature and It will eat up the system resources and slow down the PC performance. The Exploit.CVE-2010-0249.F will help hackers to record the web searching history and collect privacy data such as private information, account details, and even your email or account password. Those records will be used for further marketing purpose by third party.

Additionally the CVE-2010-0249 exploit can create backdoors and reverse connections to allow remote control of the computer for instance by using VNC and other remote client software.

## **A-portscan.cap**

**For this first challenge, You have been provided with a PCAP file that contains transactions from two hosts. One of the hosts is performing a port scan against another. Your goal is to determine the open ports on the server. (You should list each port in numerical order and indicate the default protocol that usually uses that port, if any)**

By using Wireshark we can configure the display filter `tcp.flags.syn == 1` and `tcp.flags.ack == 1` against the pcap file to filter only those host ports that returned an open indication by sending a sync/ack return as part of the TCP three way handshake. This would eliminate all the closed ports and list all the open responsive ports:



The screenshot shows the Wireshark interface with the display filter `tcp.flags.syn == 1 and tcp.flags.ack == 1` applied. The packet list shows several TCP packets from 192.168.223.172 to 192.168.223.171, all with the SYN-ACK flag set. The first packet is at time 0.077299, and the last is at 2.173371. The packet details pane shows the TCP header with Seq=0, Ack=1, Win=5840, Len=0, and MSS=1460.

No.	Time	Source	Destination	Protocol	Length	Info
10	0.077299	192.168.223.172	192.168.223.171	TCP	60	5900->48780 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
13	0.077374	192.168.223.172	192.168.223.171	TCP	60	22->48780 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
15	0.080341	192.168.223.172	192.168.223.171	TCP	60	139->48780 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
45	0.086298	192.168.223.172	192.168.223.171	TCP	60	23->48780 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
48	0.086671	192.168.223.172	192.168.223.171	TCP	60	80->48780 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
53	0.087270	192.168.223.172	192.168.223.171	TCP	60	53->48780 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
1801	1.181232	192.168.223.172	192.168.223.171	TCP	60	25->48781 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
1803	1.181292	192.168.223.172	192.168.223.171	TCP	60	445->48781 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
1807	1.181637	192.168.223.172	192.168.223.171	TCP	60	21->48781 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
1811	1.181958	192.168.223.172	192.168.223.171	TCP	60	111->48781 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
1816	1.182432	192.168.223.172	192.168.223.171	TCP	60	3306->48781 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
5330	1.286448	192.168.223.172	192.168.223.171	TCP	60	6667->48780 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
9513	2.173371	192.168.223.172	192.168.223.171	TCP	60	59563->48780 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460

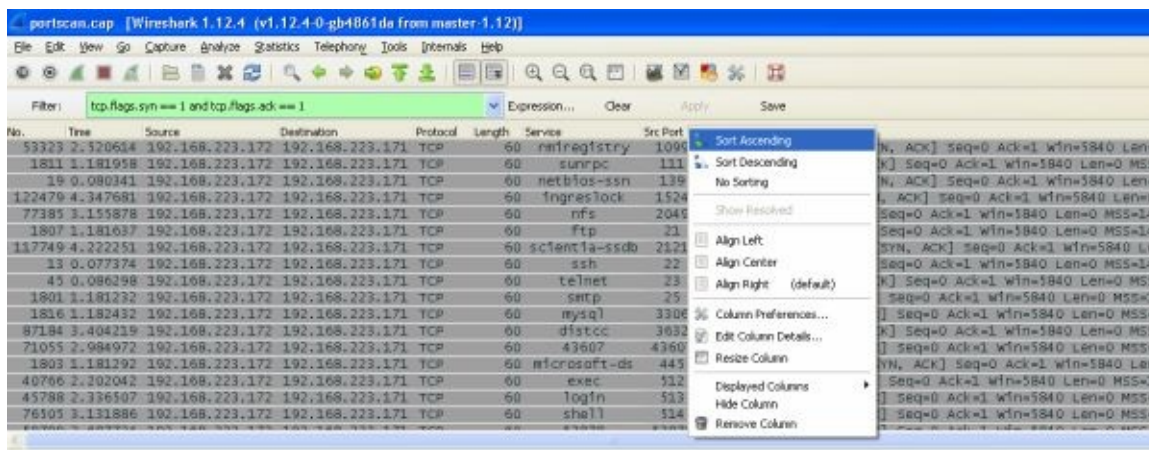
In the Wireshark display above we can see the replies from the scanned machine 192.168.223.172 to the machine doing the scanning 192.168.223.171, however as we have our display filter set to

`Tcp.flags.sync == 1` and `tcp.flags.ack == 1`, we are only seeing those packets that are returning SYNC, ACK as part of the three way handshake that indicates a port is open, listening and available to complete the connection.

However to display the port numbers in their own column and resolve the service we need to add two new columns to Wireshark default setup. We do this by going to Edit -> Preferences -> Column.

Then add new column, and add Field types; src-port (unresolved) for the port number and src-port (resolved) to obtain and display the service.

Consequently, we can determine all the open ports on 192.168.223.172 by listing the open port numbers and services that replied with a SYN/ACK. Additionally we can sort the columns in ascending or descending order by right clicking on the column header, and choosing options from a drop up box – as shown below.



However to arrange and export the data from the columns for reporting etc, we can export the Wireshark data into Excel via a CSV file.

To do this we have to do the following:

- Go to File -> Export Packet Dissections -> Save as CSV
- Name the file and save as .CSV extension
- Open Excel and select -> Open File
- Open the CSV file and Excel will prompt you to continue as it is not in Excel format, select, Ok
- Excel will pop up the Text Import Wizard, Select defaults, and press next
- In Text Import Wizard step 2 – select comma as the delimiter, you will see a screen like this:





- Press Finish
- The formatted data will appear in an Excel spreadsheet which you can then manipulate to create a table like the one below.

The full list of open ports and their corresponding services are listed below, not all ports have a dedicated service.

Service	Src Port
ftp	21
ssh	22
telnet	23
smtp	25
domain	53
http	80
sunrpc	111
netbios-ssn	139
microsoft-ds	445
exec	512
login	513
shell	514
rmiregistry	1099
ingreslock	1524
nfs	2049
scientia-	

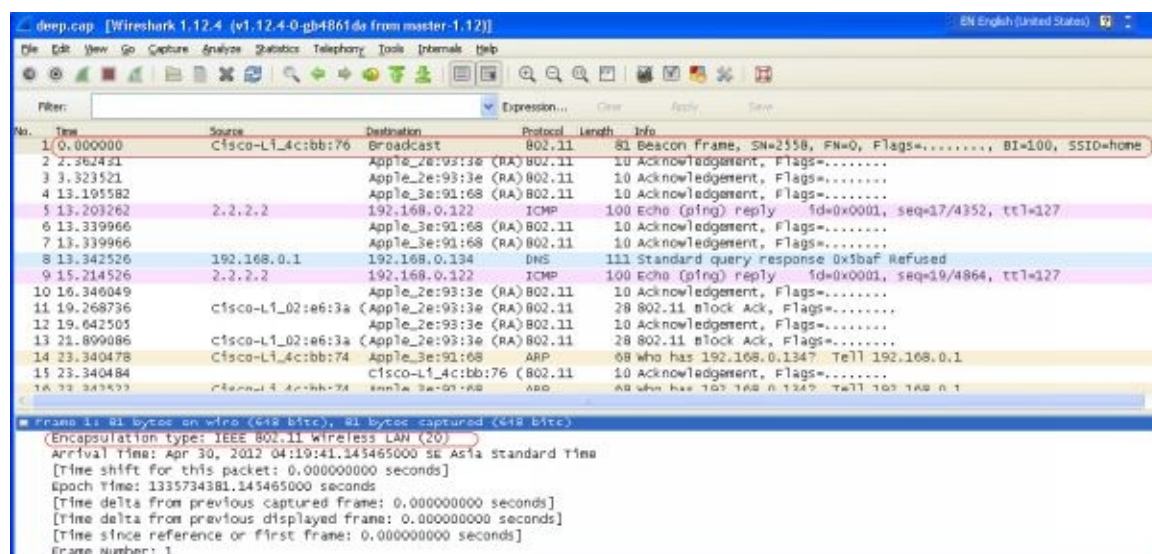
ssdb	2121
mysql	3306
distcc	3632
postgresql	5432
rfb	5900
x11	6000
ircu	6667
ircs-u	6697
8009	8009
8180	8180
msgsrvr	8787
43607	43607
52878	52878
56166	56166
59563	59563

## A-deep.cap

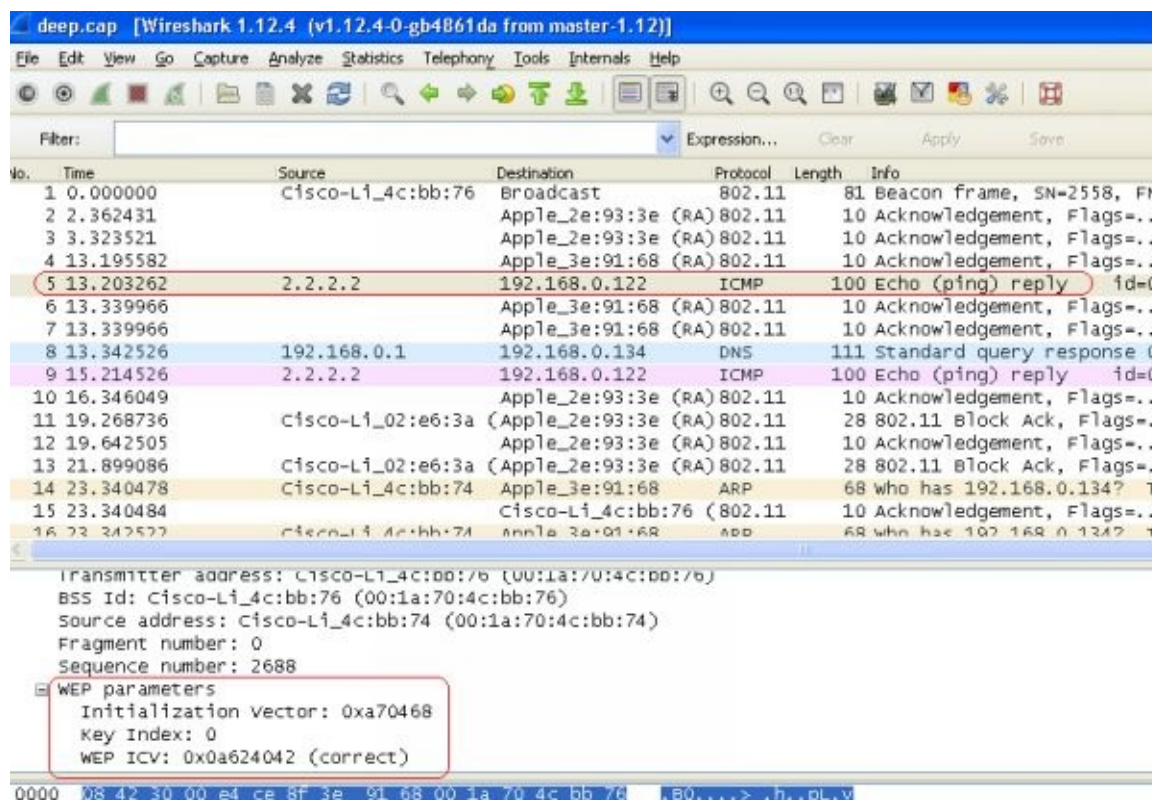
**For your final challenge, you've been giving a capture file and you need to find a password hidden inside a text file somewhere within the capture. (you might**

have to perform a few more steps in between, to get the text file ?)

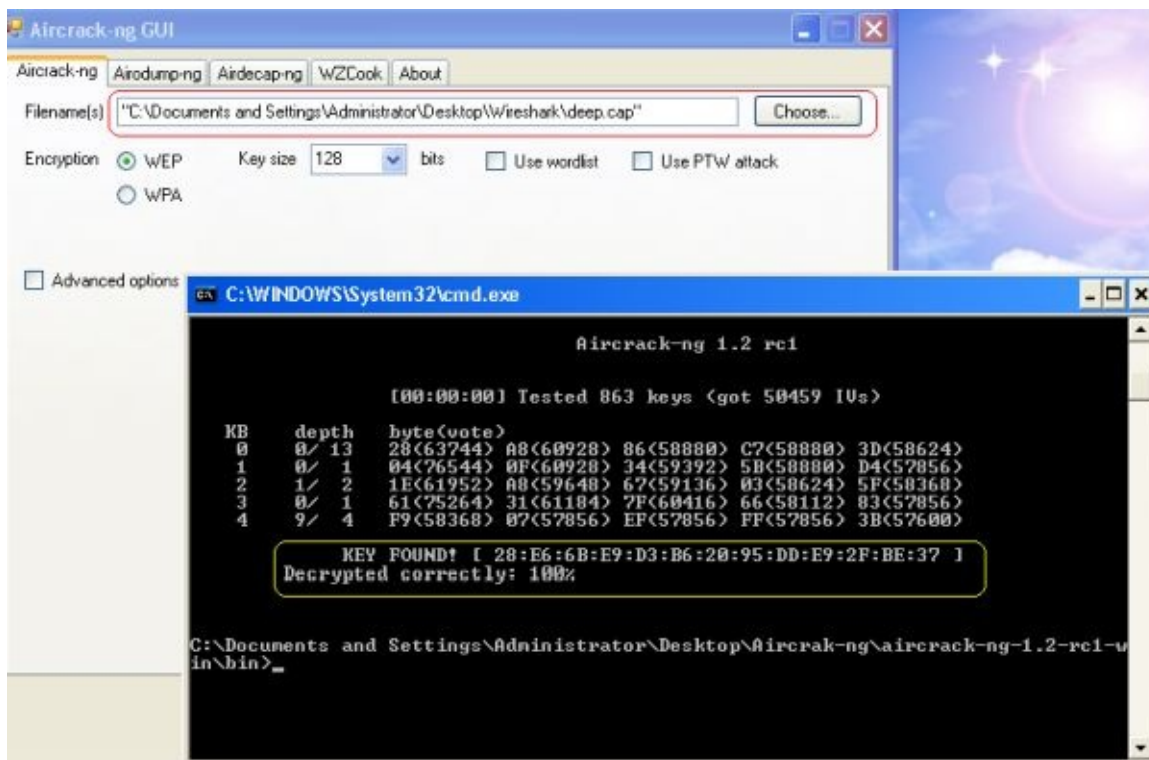
In this pcap trace, we can see straight away that this is an 802.11 protocol (wireless network) traffic scan.



Furthermore, within the packet details we can see that it is using WEP encryption and in order to decipher the payload details we will need to crack the WEP key.

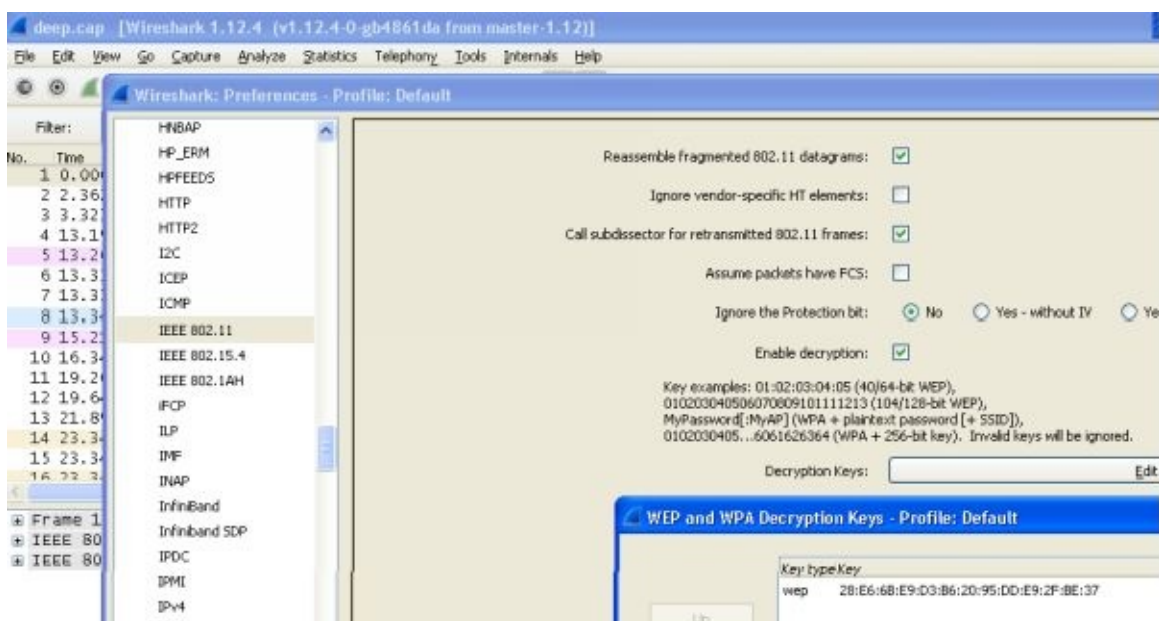


This fortunately is relatively straightforward as WEP is a weak form of encryption that can be relatively easily cracked. Therefore, by using Aircrack-ng we can open up Aircrack-ng gui and use the deep.cap file as the input. This results with Aircrack-ng quickly returning the WEP key as being:



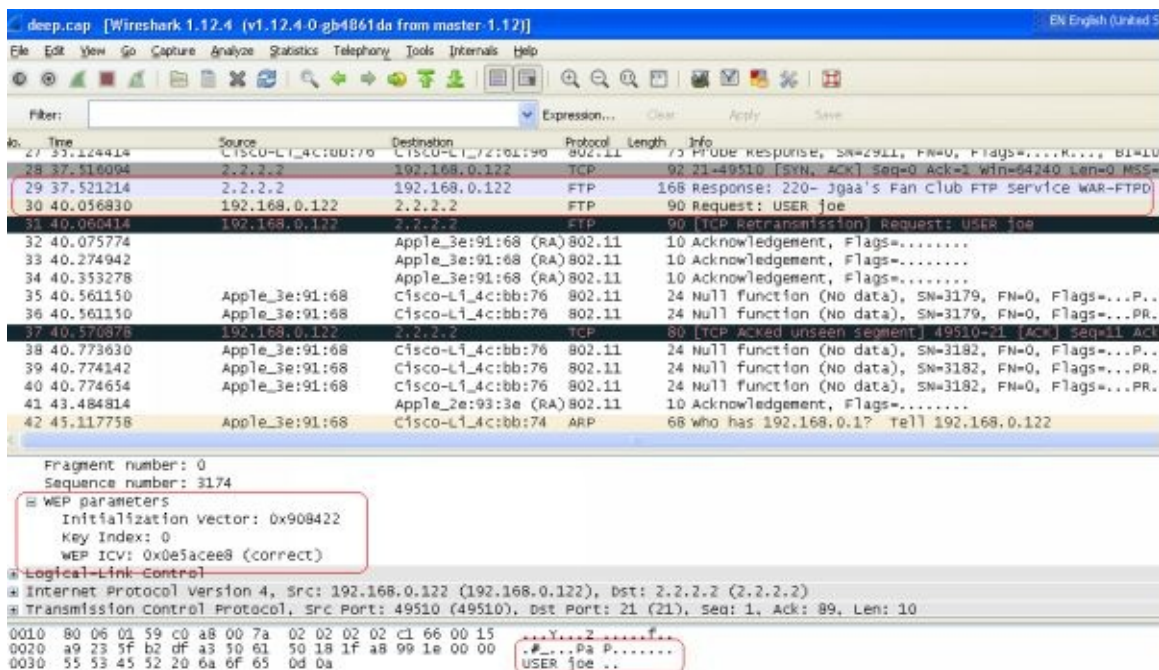
28:E6:6B:E9:D3:B6:20:95:DD:E9:2F:BE:37

We can now use this WEP key in Wireshark to decrypt the deep.cap 802.11 packets payload by entering this WEP key in the 802.11 preferences – profile – 802.11 section and this will allow Wireshark to decrypt and display the payload in clear text.

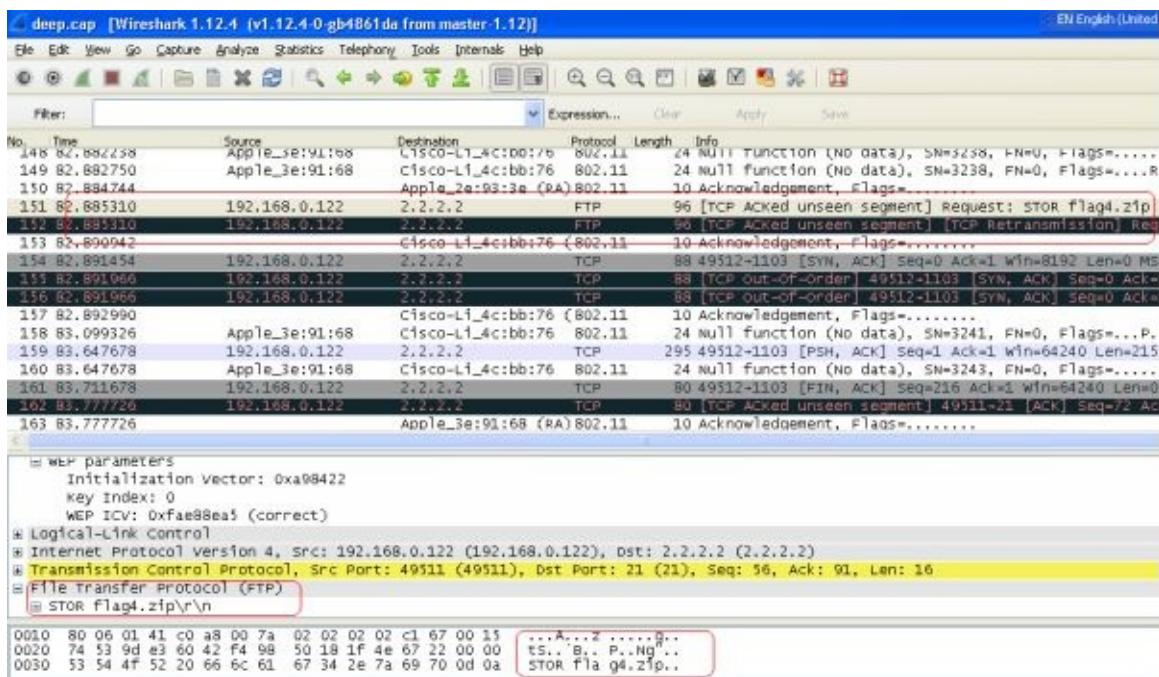


Once we have decrypted the packet payloads within the deep.cap trace we see information displayed such as User Joe and Jgaa's Fan Club FTP Service in frames 31 and 29 respectively, and this shows us that the WEP key is now working and decrypting the packets.

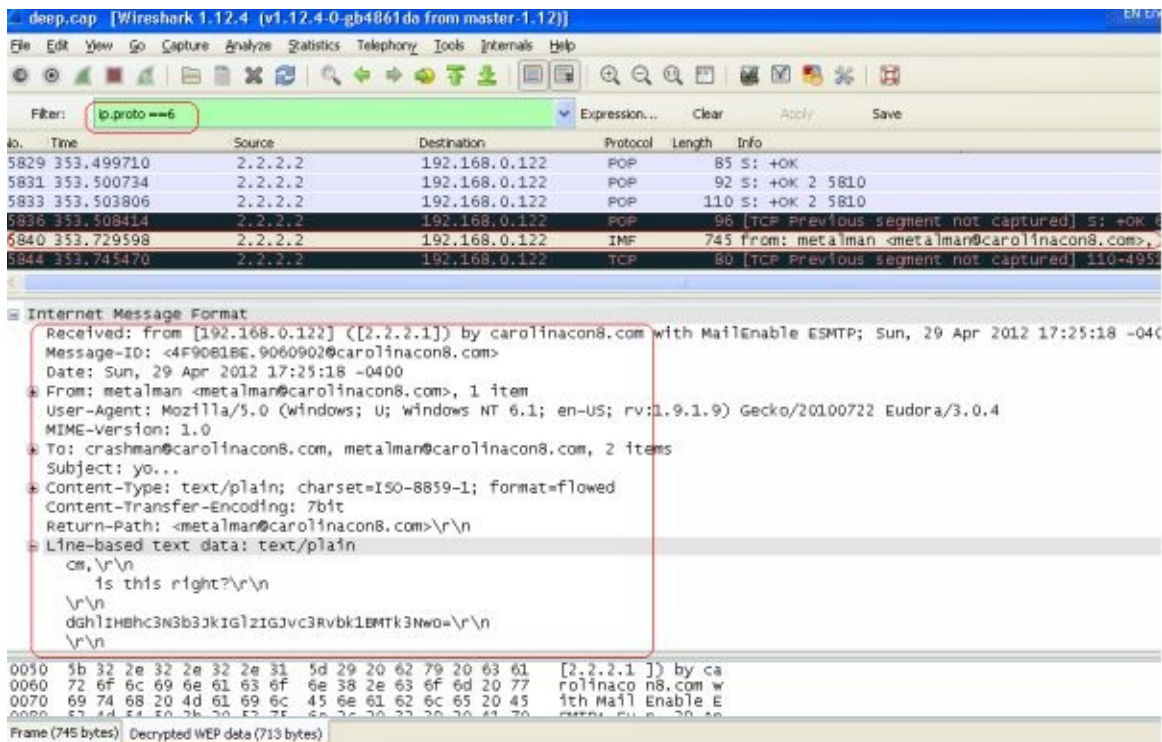




Now that we have successfully cracked the WEP encryption, the next stage is to find the password file, and we can do this by scanning through the trace until we reach frame 152 where we see a file name Flag4.zip.



In order to clear away a lot of the noise we can adjust Wireshark to only display TCP streams and then we can clearly see our FTP conversation as well as some email especially one IMF message at frame 105840.



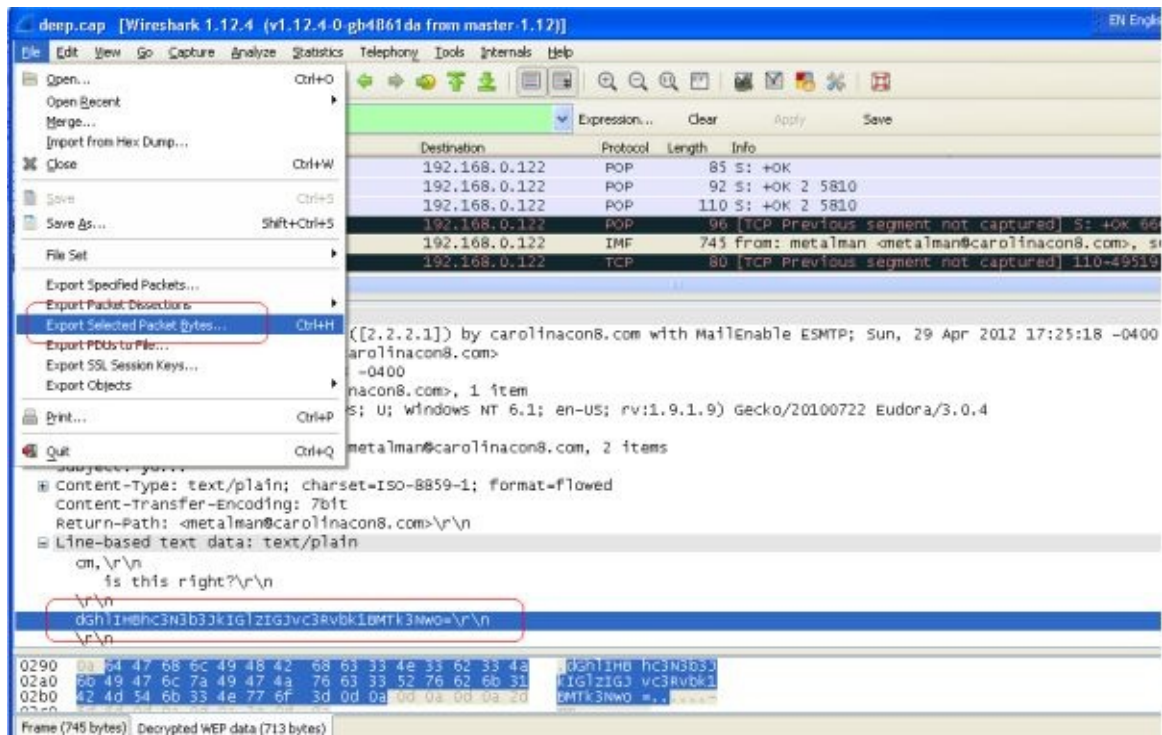
This message is from [Metalman@carolinacon8.com](mailto:Metalman@carolinacon8.com). We can decrypt the IMF message by clicking on the Wireshark decrypt WEP tab at the bottom of the data display. Moreover, what we see is:

An email from metalman to crashman, subject Yo ...,

Is this right?

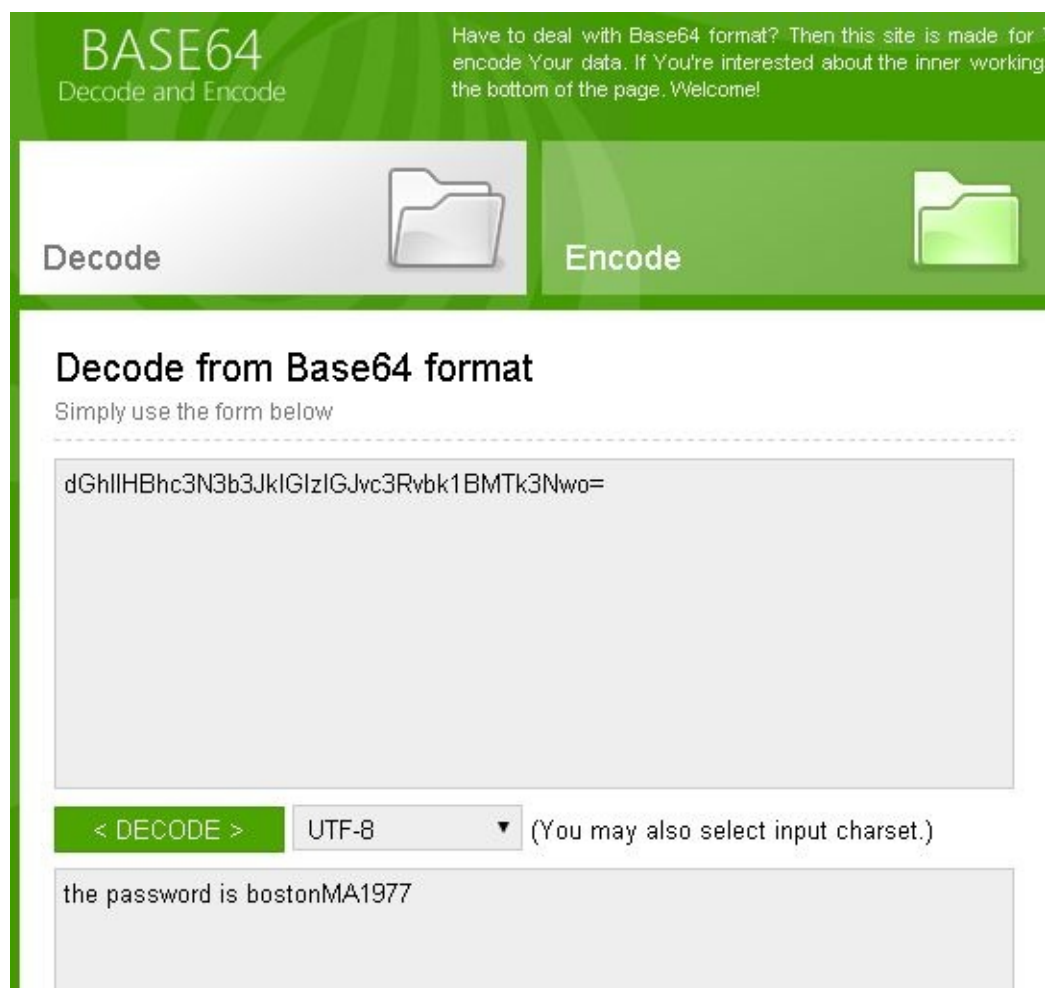
dGhlIHBhc3N3b3JkIGlzIGJvc3RvbklBMTk3Nwo=

We now want to examine that string of characters so we highlight the string in Wireshark.



Then the packet can be exported using File -> 'export selected packet bytes' and we save it as the default raw data, calling it 'password'.

If we then open the file password, we will have the data string available to manipulate. And the first thing we want to do is to convert it from Base64 to Ascii in a decoder and when we do we get returned:



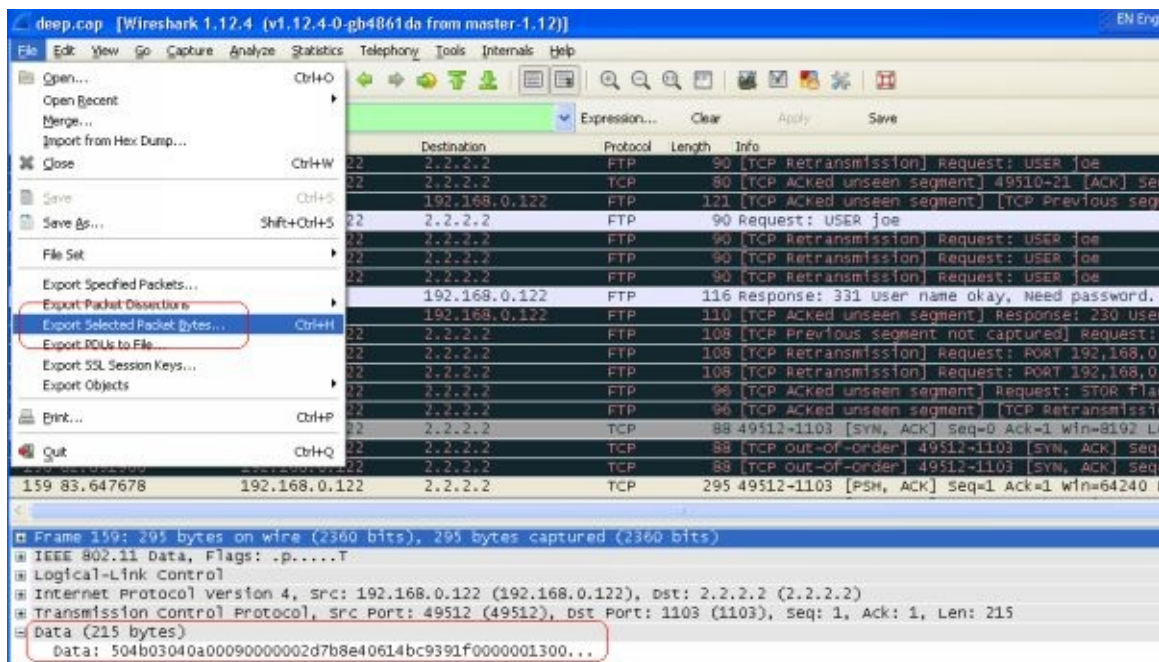
The screenshot shows a web application titled "BASE64 Decode and Encode". It has a green header with a welcome message: "Have to deal with Base64 format? Then this site is made for you. encode Your data. If You're interested about the inner workings, look at the bottom of the page. Welcome!". Below the header are two buttons: "Decode" with a folder icon and "Encode" with a folder icon. The "Decode" button is active. Below the buttons, the text "Decode from Base64 format" is displayed, followed by the instruction "Simply use the form below". A large text input field contains the Base64 string "dGhlIHBC3N3b3JkIGlzlGJvc3RvbK1BMTk3Nwo=". Below the input field is a green button labeled "< DECODE >" and a dropdown menu set to "UTF-8" with the text "(You may also select input charset.)". Below these, a text output field displays the decoded result: "the password is bostonMA1977".

“the password is bostonMA1977”

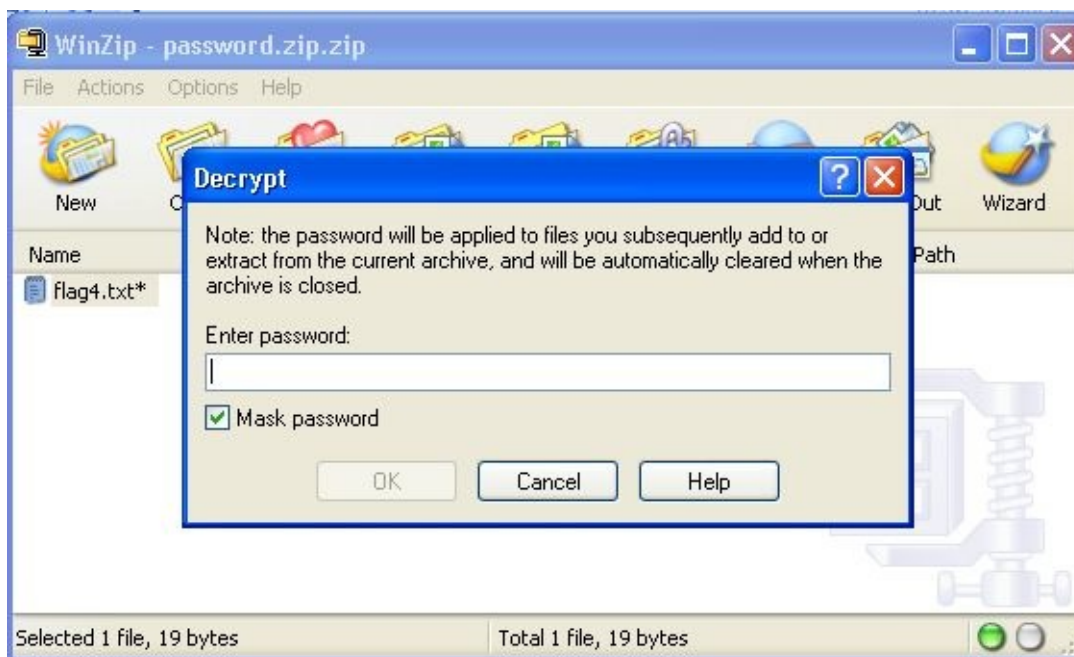
So we now know the password for the Flag4.zip file and that reveals a file Flag4.txt  
And within Flag4.txt will be the password.

To do that we have to find the actual zip file flag4.zip and extract it from the pcap file. We do this by looking first for the flag4.zip file and we can see from our decrypted WEP Wireshark file that it is found in frame 159.

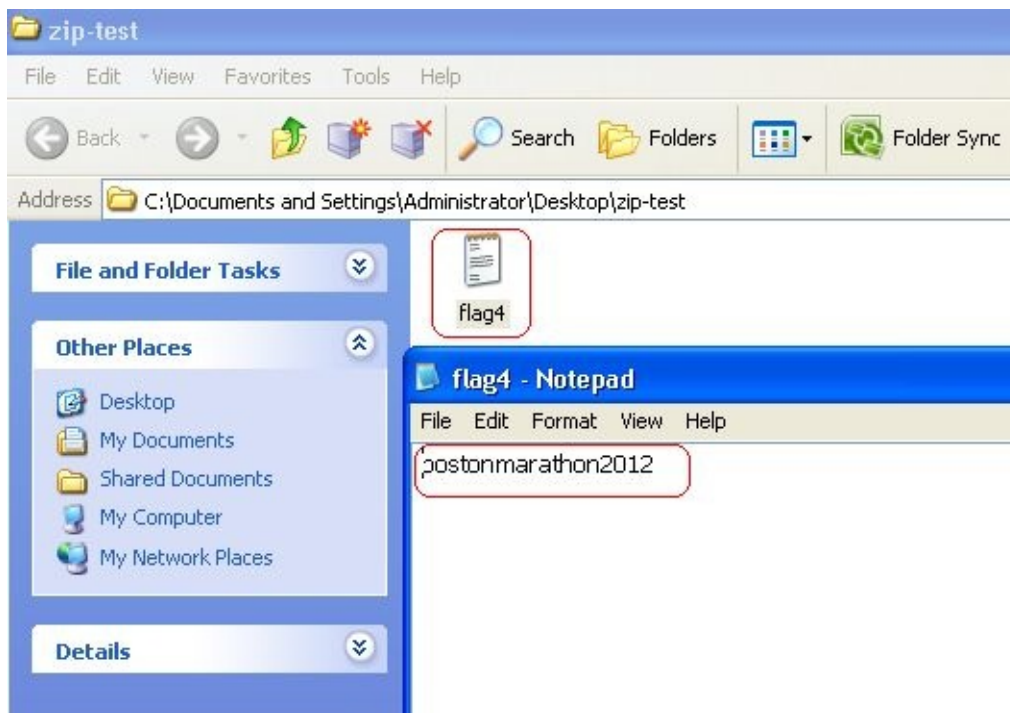




We scroll down and select the data from the TCP packet, as shown above and then go to file -> export specific packets Bytes. Then we save the raw output to a file named something like password.zip on the desktop. We then have a captured flag4.zip file called password.zip, which we can now open using Winzip, and we are prompted for a password.







We enter the previous discovered password ' bostonMA1977 ', and we find the flag.txt file, opening it we discover the real password:

**bostonmarathon2012**