

# 1

## Section 1: Obtaining the Evidence

This section focuses on the basics of network forensics while covering essential concepts, tools, and techniques involved in executing a network forensic investigation.

The following chapters will be covered in this section:

- Chapter 1, *Introducing Network Forensics*
- Chapter 2, *Technical Concepts and Acquiring Evidence*

# COMPUTER AND NETWORK FORENSICS

# 1

# Introduction to Network Forensics

**Network forensics** is one of the sub-branches of digital forensics where the data being analyzed is the network traffic going to and from the system under observation. The purposes of this type of observation are collecting information, obtaining legal evidence, establishing a root-cause analysis of an event, analyzing malware behavior, and so on.

Professionals familiar with **digital forensics and incident response (DFIR)** know that even the most careful suspects leave traces and artifacts behind. But forensics generally also includes imaging the systems for memory and hard drives, which can be analyzed later. So, how do network forensics come into the picture? Why do we need to perform network forensics at all? Well, the answer to this question is relatively simple.

Let's consider a scenario where you are hunting for some unknown attackers in a massive corporate infrastructure containing thousands of systems. In such a case, it would be practically impossible to image and analyze every system. The following two scenarios would also be problematic:

- Instances where the disk drives may not be available
- Cases where the attack is in progress, and you may not want to tip off the attackers

Whenever an intrusion or a digital crime happens over the wire, whether it was successful or not, the artifacts left behind can help us understand and recreate not only the intent of the attack, but also the actions performed by the attackers.

If the attack was successful, what activities were conducted by the attackers on the system? What happened next? Generally, most severe attacks, such as **Advanced Package Tool (APT)**, **ransomware**, **espionage**, and others, start from a single instance of an unauthorized entry into a network and then evolve into a long-term project for the attackers until the day their goals are met; however, throughout this period the information flowing in and out of the network goes through many different devices, such as routers, firewalls, hubs, switches, web proxies, and others. Our goal is to identify and analyze all these different artifacts. Throughout this chapter, we will discuss the following:

- Network forensics methodology
- Sources of evidence
- A few necessary case studies demonstrating hands-on network forensics

## Technical requirements

To perform the exercises covered in this chapter, you will require the following:

- A laptop/desktop computer with an i5/i7 processor or any other equivalent AMD processor with at least 8 GB RAM and around 100 GB of free space.
- VMware Player/VirtualBox installation with Kali OS installed. You can download it from <https://www.offensive-security.com/kali-linux-vm-virtualbox-image-download/>.
- Installing Wireshark on Windows: [https://www.wireshark.org/docs/wsug\\_html\\_chunked/ChBuildInstallWinInstall.html](https://www.wireshark.org/docs/wsug_html_chunked/ChBuildInstallWinInstall.html).
- Netcat From Kali Linux (already installed).
- Download NetworkMiner from <https://www.netresec.com/?page=Networkminer>.

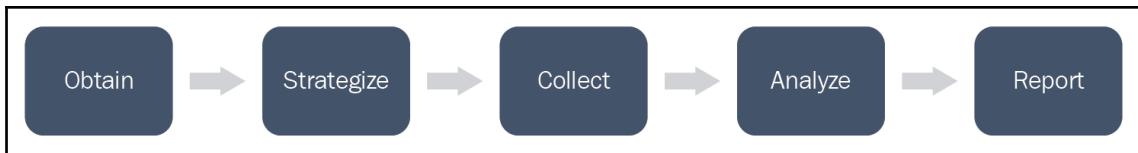
Every investigation requires a precise methodology. We will discuss the popular network forensics methodology used widely across the industry in the next section.



To install Wireshark on Windows, go to [https://www.wireshark.org/docs/wsug\\_html\\_chunked/ChBuildInsta11WinInstall.html](https://www.wireshark.org/docs/wsug_html_chunked/ChBuildInsta11WinInstall.html).

# Network forensics investigation methodology

To assure accurate and meaningful results at the end of a network forensic exercise, you, as a forensic investigator, must follow a rigid path through a methodological framework. This path is shown in the following diagram:



**Obtain, Strategize, Collect, Analyze, and Report (OSCAR)** is one such framework that ensures appropriate and constant results. Let's look at each phase from a network forensics point of view:

- **Obtain information:** Obtaining information about the incident and the environment is one of the first things to do in a network forensics exercise. The goal of this phase is to familiarize a forensic investigator with the type of incident. The timestamps and timeline of the event, the people, systems, and endpoints involved in the incident—all of these facts are crucial in building up a detailed picture of the event.
- **Strategize:** Planning the investigation is one of the critical phases in a network forensics scenario, since logs from various devices can differ in their nature; for example, the volatility of log entries from a firewall compared with that of details such as the ARP of a system would be very different. A good strategy would impact the overall outcome of the investigation. Therefore, you should keep the following points in mind while strategizing the entire forensics investigation process:
  - Define clear goals and timelines
  - Find the sources of evidence
  - Analyze the cost and value of the sources
  - Prioritize acquisition
  - Plan timely updates for the client

- **Collect:** In the previous phase, we saw how we need to strategize and plan the acquisition of evidence. In the collect phase, we will go ahead and acquire the evidence as per the plan; however, collecting the evidence itself requires you to document all the systems that are accessed and used, capturing and saving the data streams to the hard drive and collecting logs from servers and firewalls. Best practices for evidence collection include the following:
  - Make copies of the evidence and generate cryptographic hashes for verifiability
  - Never work on the original evidence; use copies of the data instead
  - Use industry-standard tools
  - Document all your actions
- **Analyze:** The analysis phase is the core phase where you start working on the data and try your hands at the riddle. In this phase, you will make use of multiple automated and manual techniques using a variety of tools to correlate data from various sources, establishing a timeline of events, eliminating false positives, and creating working theories to support evidence. We will spend most of the time in this book discussing the analysis of data.
- **Report:** The report that you produce must be in layman's terms—that is, it should be understood by non-techie people, such as legal teams, lawyers, juries, insurance teams, and so on. The report should contain executive summaries backed by the technical evidence. This phase is considered one of the essential stages, since the last four steps need to be explained in this one.



For more on OSCAR methodology, you can visit [https://www.researchgate.net/figure/OSCAR-methodology\\_fig2\\_325465892](https://www.researchgate.net/figure/OSCAR-methodology_fig2_325465892).

## Source of network evidence

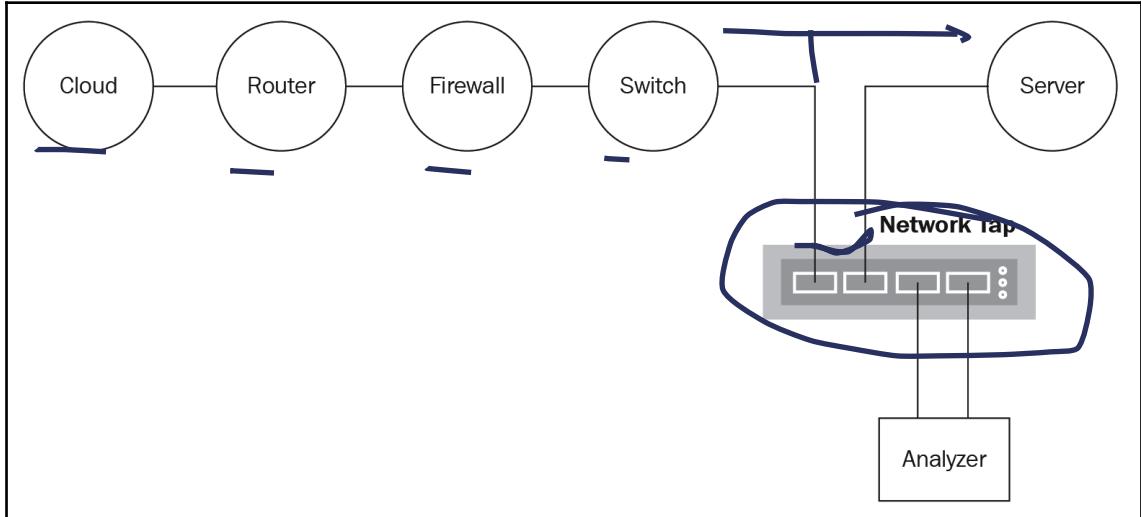
Network evidence can be collected from a variety of sources and we will discuss these sources in the next section. The sources that we will be discussing are:

- Tapping the wire and the air
  - CAM table on a network switch
  - Routing tables on routers
  - Dynamic Host Configuration Protocol logs
  - DNS server logs
  - Domain controller/ authentication servers/ system logs
  - IDS/IPS logs
  - Firewall logs
  - Proxy Server logs
- 
- 

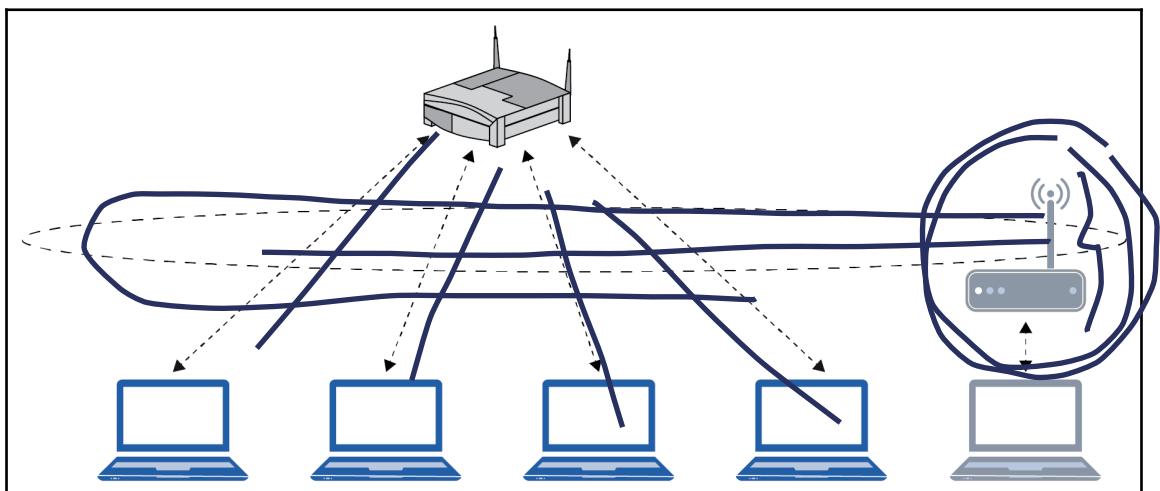
### Tapping the wire and the air

One of the purest and most raw forms of information capture is to put taps on network and optical fiber cables to snoop on traffic.

Many commercial vendors provide network taps and SPAN ports on their devices for snooping where they will forward all traffic seen on the particular port to the analyzer system. The technique is shown in the following diagram:



In the case of WLAN or Wi-Fi, the captures can be performed by putting an external wireless receptor into promiscuous mode and recording all the traffic for a particular wireless access point on a particular channel. This technique is shown in the following diagram:



## CAM table on a network switch

Network switches contain content-addressable memory tables that store the mapping between a system's MAC address and the physical ports. In a large setup, this table becomes extremely handy, as it can pinpoint a MAC address on the network to a wall-jacked system, since mappings are available to the physical ports. Switches also provide network-mirroring capabilities, which will allow the investigators to see all the data from other VLANs and systems.

## Routing tables on routers

Routing tables in a router maps ports on the router to the networks that they connect. The following table is a routing table. These tables allow us to investigate the path that the network traffic takes while traveling through various devices:

Routing Table					
Destination	Gateway	Genmask	Metric	Interface	Type
122.176.127.70	0.0.0.0	255.255.255.255	0	Internet WAN	Dynamic
192.168.1.0	0.0.0.0	255.255.255.0	0	LAN	Dynamic
0.0.0.0	122.176.127.70	0.0.0.0	0	Internet WAN	Dynamic

Most of the routers have inbuilt packet filters and firewall capabilities as well. This means that they can be configured to log denied or certain types of traffic traveling to and from the network.

## Dynamic Host Configuration Protocol logs

**Dynamic Host Configuration Protocol (DHCP)** servers generally log entries when a specific IP address is assigned to a particular MAC address, when a lease was renewed on the network, the timestamp it renewed, and so on, thus having significant value in network forensics. The following screenshot of the router's DHCP table presents a list of dynamically allocated hosts:

DHCP Clients Table			
Host Name	IP Address	MAC Address	Remaining Lease Time (in seconds)
android-73355629bd9b62e5	192.168.1.2	34:bc:00:21:0f:00	26518
iPad	192.168.1.3	54:99:63:82:64:f5	24818
iPhone	192.168.1.4	70:f0:87:bf:17:ab	22451
XboxOne	192.168.1.6	30:59:b7:e5:f9:89	27815
Apex	192.168.1.7	2c:33:61:77:23:ef	26599
Lucideuss-MBP	192.168.1.8	8c:85:90:74:fe:ee	25825
Chromecast	192.168.1.9	54:60:09:84:3f:24	19346
DESKTOP-PESQ21S	192.168.1.10	b0:10:41:c8:46:df	25062
<button>Refresh</button>	<button>Close</button>		

## DNS servers logs

Name server query logs can help understand IP-to-hostname resolution at specific times. Consider a scenario where, as soon as a system got infected with malware on the network, it tried to connect back to a certain domain for command and control. Let's see an example as follows:

467 0.00257700192.168.1.10	192.168.1.1	DNS	59506 53	Standard query 0x193a A malwaresamples.com
468 0.00832700192.168.1.1	192.168.1.10	DNS	53 59506	Standard query response 0x193a A 50.63.202.24
469 0.00142200192.168.1.10	192.168.1.1	DNS	54504 53	Standard query 0x9cd1 AAAA malwaresamples.com
473 0.06258100192.168.1.10	192.168.1.1	DNS	54504 53	Standard query 0x9cd1 AAAA malwaresamples.com
486 0.19158900192.168.1.1	192.168.1.10	DNS	53 54504	Standard query response 0x9cd1
738 35.2107440192.168.1.7	224.0.0.251	MDNS	5353 5353	Standard query 0x0000 PTR _homekit._tcp.local,
792 10.7856550192.168.1.10	192.168.1.1	DNS	51618 53	Standard query 0x00be A support.mozilla.org
793 0.00907100192.168.1.1	192.168.1.10	DNS	53 51618	Standard query response 0x00be CNAME prod.sumo.
794 0.00080100192.168.1.10	192.168.1.1	DNS	58122 53	Standard query 0x6fc1 A prod-tb.sumo.mozilla.org

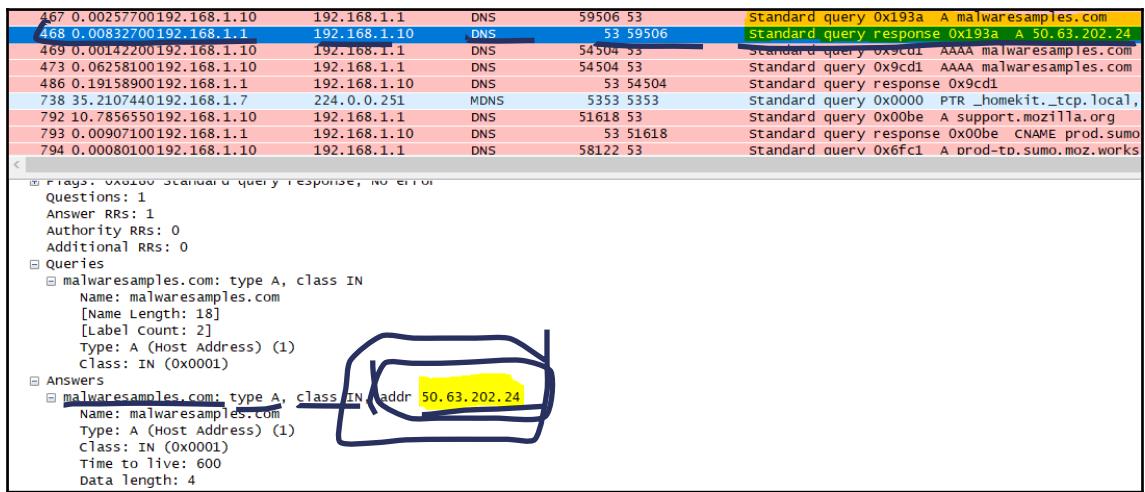
**Flags:** 0x0000 Standard query response, no error  
**Questions:** 1  
**Answer RRs:** 1  
**Authority RRs:** 0  
**Additional RRs:** 0

**Queries**

- malwaresamples.com: type A, class IN
  - Name: malwaresamples.com  
 [Name Length: 18]  
 [Label Count: 2]  
 Type: A (Host Address) (1)  
 Class: IN (0x0001)

**Answers**

- malwaresamples.com: type A, class IN, addr 50.63.202.24
  - Name: malwaresamples.com  
 Type: A (Host Address) (1)  
 Class: IN (0x0001)  
 Time to live: 600  
 Data length: 4



We can see in the preceding screenshot that a DNS request was resolved for malwaresamples.com website and the resolved IP address was returned.

Having access to the DNS query packets can reveal **Indicators of Compromise** for a particular malware on the network while quickly revealing the IP address of the system making the query, and can be dealt with ease.

## Domain controller/authentication servers/ system logs

Authentication servers can allow an investigator to view login attempts, the time of the login, and various other login-related activities throughout the network. Consider a scenario where a group of attackers tries to use a compromised host to log into the database server by using the compromised machine as a launchpad (pivoting). In such cases, authentication logs will quickly reveal not only the infected system, but also the number of failed/passed attempts from the system to the database server.

## IDS/IPS logs

From a forensic standpoint, intrusion detection/prevention system logs are the most helpful. IDS/IDPS logs provide not only the IP address, but also the matched signatures, ongoing attacks, malware presence, command-and-control servers, the IP and port for the source and destination systems, a timeline, and much more. We will cover IDS/IPS scenarios in the latter half of this book.

### Firewall logs

Firewall logs provide a detailed view of activities on the network. Not only do firewall solutions protect a server or a network from unwanted connections, they also help to identify the type of traffic, provide a trust score to the outbound endpoint, block unwanted ports and connection attempts, and much more. We will look at firewalls in more detail in the upcoming chapters.

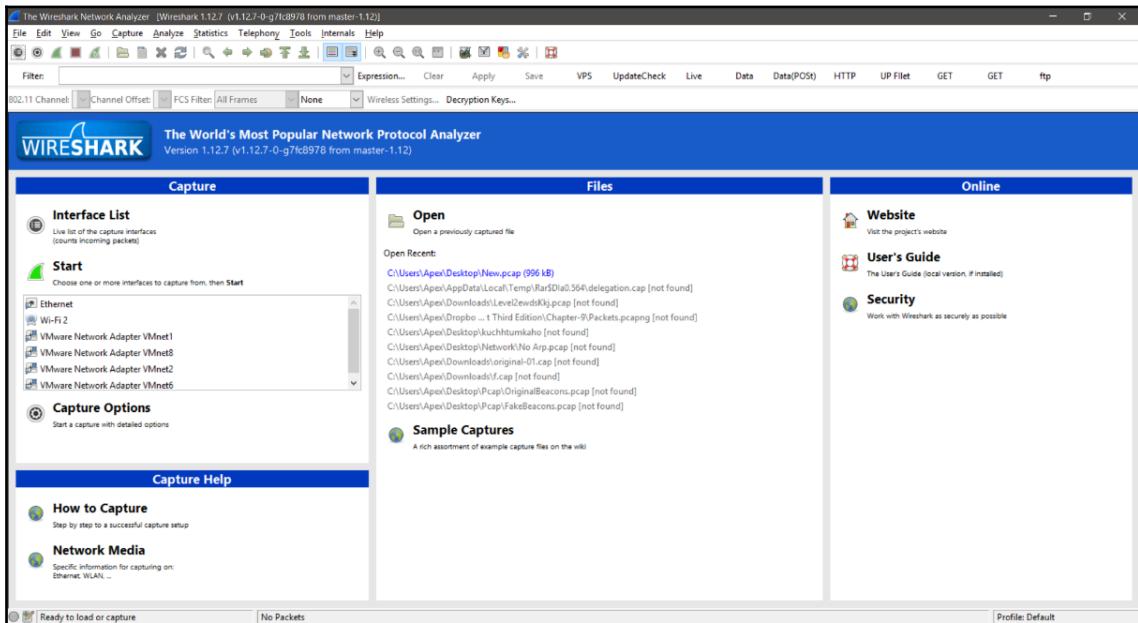
### Proxy server logs

Web proxies are also one of the most useful features for a forensic investigator. Web proxy logs help uncover internal threats while providing explicit detail on events such as surfing habits, the source of web-based malware, the user's behavior on the network, and so on.

Since we now have an idea about the various types of logs we can consider for analysis, let us quickly familiarize ourselves on the basics of Wireshark.

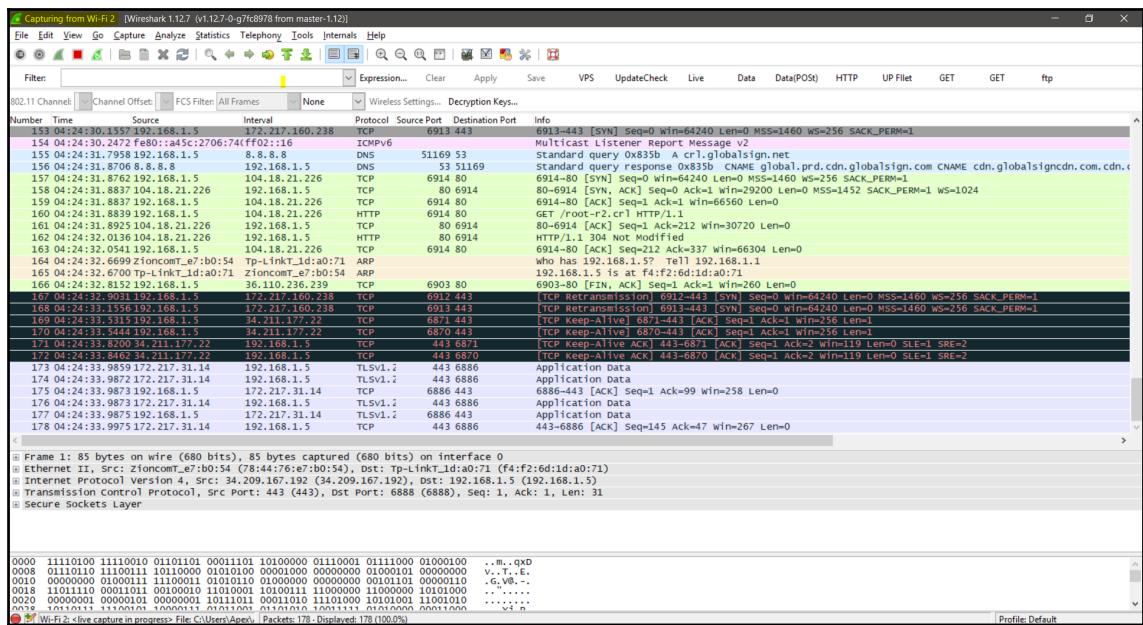
## Wireshark essentials

Readers who are familiar with the basics of Wireshark can skip this section and proceed with the case studies; however, readers who are unfamiliar with the basics or who need to brush up on Wireshark essentials, can feel free to continue through this section. Let's look at some of the most basic features of Wireshark. Look at the following screenshot:



Wireshark

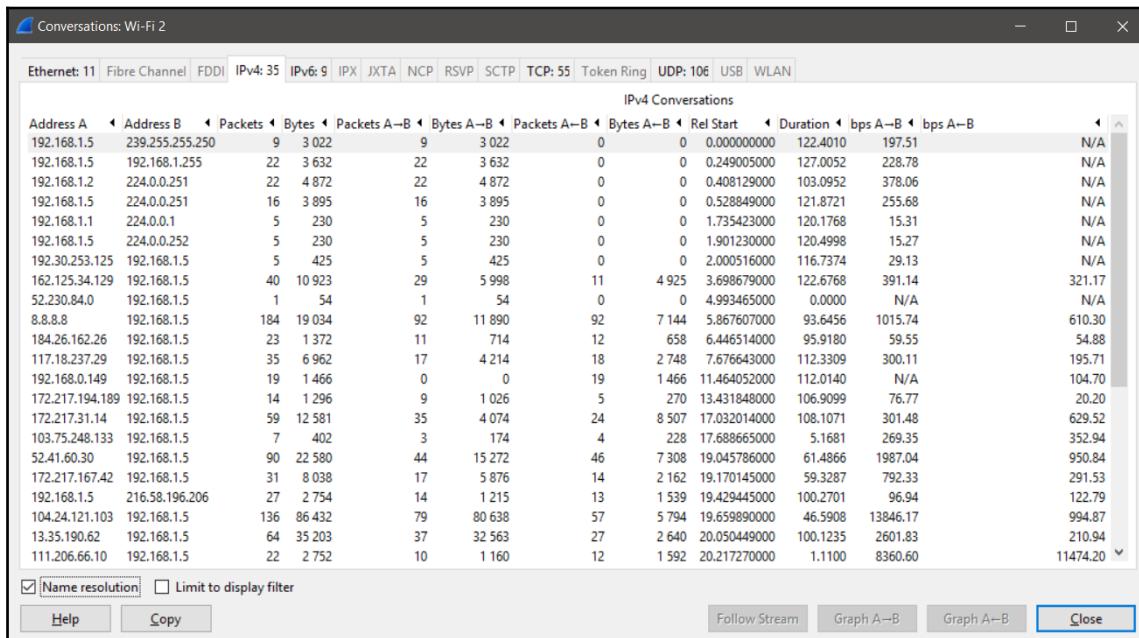
Once we execute Wireshark, we are presented with a screen similar to the preceding picture. On the left-hand side, we have a list of the available interfaces to capture packets from. In the middle, we have recent packet capture files and on the right-hand side, we have online help and user guides. To start a new packet-capture, you can select an interface, such as Ethernet, if you are connected over the wire, or Wi-Fi, if you are connected on a wireless network. Similarly, if you need to open a packet-capture file, you can press the **Open** button, browse to the capture file, and load it in the Wireshark tool. Let's capture packets from the wireless interface by selecting **Wi-Fi** and pressing the **Start** button, as shown in the following screenshot:



We can see from the preceding screenshot that we have various types of packets flowing on the network. Let's understand TCP conversations, endpoints, and basic Wireshark filters in the upcoming sections.

## Identifying conversations and endpoints

You may want to view the list of IP endpoints that your system is communicating with. To achieve this, you can navigate to the **Statistics** tab and select **Conversations**, as shown in the following screenshot:



The screenshot shows the NetworkMiner tool interface with the 'Conversations' tab selected. The main window displays a table titled 'IPv4 Conversations' with various columns representing network statistics. The columns include Address A, Address B, Packets, Bytes, Packets A→B, Bytes A→B, Packets A←B, Bytes A←B, Rel Start, Duration, bps A→B, and bps A←B. The table lists numerous entries, mostly between 192.168.1.15 and other IP addresses like 192.168.1.2, 192.168.1.5, etc. At the bottom of the window, there are checkboxes for 'Name resolution' and 'Limit to display filter', along with buttons for 'Help', 'Copy', 'Follow Stream', 'Graph A→B', 'Graph A←B', and 'Close'.

We can see that we have a variety of endpoints that are having conversations, the number of bytes transferred between the endpoints, and the duration of their data exchange. These options become extremely handy when you want to investigate malicious traffic and identify the key endpoints that are being contracted. Additionally, we can see that most of the conversations in the preceding screenshot involves 192.168.1.15 but we may not recognize the IP addresses its talking to.

We can also make use of the **Endpoints** option from the **Statistics** tab, as shown in the following screenshot:

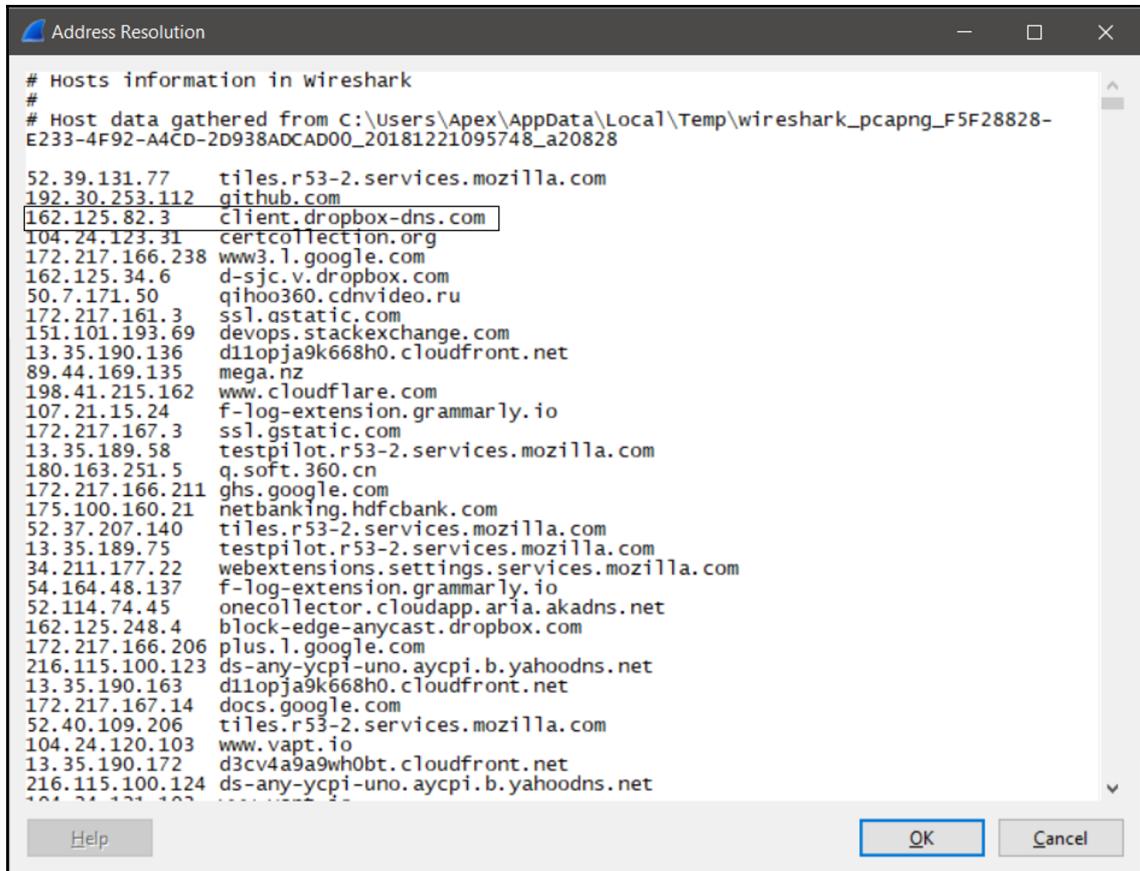
The screenshot shows the NetworkMiner interface with the 'Endpoints' tab selected. The title bar says 'Endpoints: Wi-Fi 2'. Below the title bar is a navigation bar with tabs: Ethernet: 11, Fibre Channel, FDDI, IPv4: 45 (selected), IPv6: 10, IPX, JXTA, NCP, RSVP, SCTP, TCP: 123, Token Ring, UDP: 139, USB, WLAN. The main area is titled 'IPv4 Endpoints' and contains a table of network endpoint statistics. The table has columns: Address, Packets, Bytes, Tx Packets, Tx Bytes, Rx Packets, Rx Bytes, Latitude, and Longitude. The table lists 30 endpoints. At the bottom of the table are two checkboxes: 'Name resolution' (checked) and 'Limit to display filter' (unchecked). Below the table are three buttons: Help, Copy, and Map.

Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Latitude	Longitude
192.168.1.5	2 379	1 032 989	1 281	703 553	1 098	329 436	-	-
162.125.82.4	526	558 733	151	12 360	375	546 373	-	-
8.8.8.8	220	23 139	110	14 388	110	8 751	-	-
192.168.1.2	149	27 880	99	16 699	50	11 181	-	-
172.217.31.14	138	27 352	87	9 578	51	17 774	-	-
104.24.121.103	136	86 432	79	80 638	57	5 794	-	-
172.217.161.14	94	22 998	53	13 561	41	9 437	-	-
52.41.60.30	90	22 580	44	15 272	46	7 308	-	-
23.0.137.239	85	40 579	46	36 220	39	4 359	-	-
162.125.34.129	79	22 453	52	9 063	27	13 390	-	-
104.192.108.133	72	15 724	36	4 362	36	11 362	-	-
13.35.190.62	71	35 625	40	32 737	31	2 888	-	-
52.35.21.65	69	22 594	34	16 878	35	5 716	-	-
162.125.82.3	67	25 570	37	17 653	30	7 917	-	-
216.58.221.35	62	9 935	34	6 764	28	3 171	-	-
224.0.0.251	60	12 033	0	0	60	12 033	-	-
34.195.227.26	55	16 984	26	13 083	29	3 901	-	-
192.168.0.149	51	4 002	0	0	51	4 002	-	-
34.209.167.192	48	22 855	23	13 378	25	9 477	-	-
239.255.255.250	42	14 006	0	0	42	14 006	-	-
117.18.237.29	40	7 245	19	4 334	21	2 911	-	-
172.217.167.42	40	8 693	20	6 084	20	2 609	-	-

From the preceding screenshot, we can see all the endpoints, and sorting them using the number of packets will give us a clear understanding of the endpoints that are transmitting the highest number of packets, which is again quite handy when it comes to analyzing anomalous network behavior.

## Identifying the IP endpoints

Domain names were invented to make it more easy to remember sites with common phrases. Having a list of IP addresses in the previous section would make no sense to us, but having a list that shows the resolution of the IPs into domain names can help us a lot. On clicking the **Show address resolution / Resolved Addresses** option, we will be presented with the following:



The screenshot shows a Windows-style dialog box titled "Address Resolution". The content area displays a list of IP addresses and their corresponding resolved hostnames. The list is as follows:

```
# Hosts information in wireshark
#
# Host data gathered from C:\Users\Apex\AppData\Local\Temp\wireshark_pcapan_F5F28828-E233-4F92-A4CD-2D938ADCAD00_20181221095748_a20828

52.39.131.77 tiles.r53-2.services.mozilla.com
192.30.253.112 github.com
162.125.82.3 client.dropbox-dns.com
104.24.123.31 certcollection.org
172.217.166.238 www3.l.google.com
162.125.34.6 d-sjc.v.dropbox.com
50.7.171.50 qihoo360.cdnvideo.ru
172.217.161.3 ssl.gstatic.com
151.101.193.69 devops.stackexchange.com
13.35.190.136 dllopja9k668h0.cloudfront.net
89.44.169.135 mega.nz
198.41.215.162 www.cloudflare.com
107.21.15.24 f-log-extension.grammarly.io
172.217.167.3 ssl.gstatic.com
13.35.189.58 testpilot.r53-2.services.mozilla.com
180.163.251.5 q.soft.360.cn
172.217.166.211 ghs.google.com
175.100.160.21 netbanking.hdfcbank.com
52.37.207.140 tiles.r53-2.services.mozilla.com
13.35.189.75 testpilot.r53-2.services.mozilla.com
34.211.177.22 webextensions.settings.services.mozilla.com
54.164.48.137 f-log-extension.grammarly.io
52.114.74.45 onecollector.cloudapp.aria.akadns.net
162.125.248.4 block-edge-anycast.dropbox.com
172.217.166.206 plus.l.google.com
216.115.100.123 ds-any-ycpi-uno.aycpib.yahoodns.net
13.35.190.163 dllopja9k668h0.cloudfront.net
172.217.167.14 docs.google.com
52.40.109.206 tiles.r53-2.services.mozilla.com
104.24.120.103 www.vapt.io
13.35.190.172 d3cv4a9a9wh0bt.cloudfront.net
216.115.100.124 ds-any-ycpi-uno.aycpib.yahoodns.net
```

At the bottom of the dialog box, there are three buttons: "Help", "OK", and "Cancel".

Well, this now makes proper sense, as we have a list of IP addresses with their domain resolutions that can help us eliminate the false positives. We saw in the previous endpoint section that the second-highest number of packets in the endpoints originated from 162.125.34.6. Since we don't have an idea of what IP address this could be, we can easily refer to the address resolutions and figure out that this is dropbox-dns.com, which looks suspicious. Let's search for it on Google using the string client.dropbox-dns.com, and browsing the first result from the search, we have the following result:

The screenshot shows a web browser window with the URL <https://www.dropbox.com/help/security/official-domains>. The page content includes a list of legitimate Dropbox domains and a section for other domains used for networking, where 'dropbox-dns.com' is listed.

All Dropbox content should originate from one of the following domains:

- db.tt
- dropbox.com
- dropboxapi.com
- dropboxbusiness.com
- dropboxcaptcha.com
- dropboxinsiders.com
- dropboxmail.com
- dropboxpartners.com
- dropboxstatic.com
- dropbox.zendesk.com
- getdropbox.com
- instructorledlearning.dropboxbusiness.com
- paper.dropbox.com

**Other domains used for networking**

- dropbox-dns.com

We can see from the preceding search result (the official Dropbox website, <https://www.dropbox.com/>) that the domain is a legitimate Dropbox domain and the traffic originating to and from it is safe (assuming that Dropbox is permitted on the network or if allowed for a select group of users that the traffic is associated with those users only). This resolution not only helps us identify domains, but also speaks a lot about the software running on the target as well. We already identified Dropbox as running on the system. We also identified the following domains from the **Resolved Addresses** pane in Wireshark:

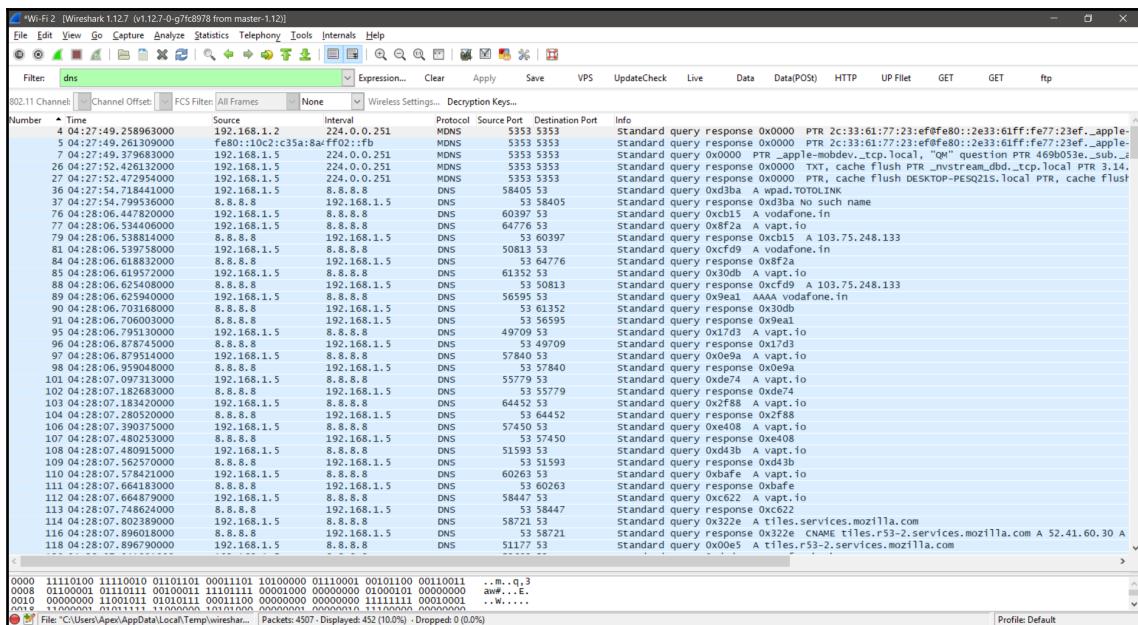
- A Gmail account being accessed
- A Qihoo 360 antivirus

- An HDFC bank account
- The Grammarly plugin
- The Firefox browser

## Basic filters

Network forensics requires you to pinpoint a variety of packets to establish a clear vision for the investigation. Let's explore how we can do this by going through the following steps:

Set up some basic display filters in Wireshark to only view packets of interest, as shown in the following screenshot:



We can see that simply typing in dns as the filter will display DNS packets only; however, we can see that MDNS protocol packets are also displayed.

Considering that we only require DNS packets and not MDNS protocol packets, we can set the filter as dns && !mdns, where ! denotes a NOT operation, as shown in the following screenshot:

Filter: dns && !mdns							Expression...	Clear	Apply	Save	VPS
Number	Time	Source	Interval	Protocol	Source Port	Destination Port					
4	04:27:49.258963000	192.168.1.2	224.0.0.251	MDNS	5353	5353					
5	04:27:49.261309000	fe80::10c2:c35a:8a <sup>ff02::fb</sup>		MDNS	5353	5353					
7	04:27:49.379683000	192.168.1.5	224.0.0.251	MDNS	5353	5353					
26	04:27:49.400000000	192.168.1.5	224.0.0.251	MDNS	5353	5353					
27	04:27:49.400000000	192.168.1.5	224.0.0.251	MDNS	5353	5353					
36	04:27:49.400000000	192.168.1.5	224.0.0.251	MDNS	5353	5353					
37	04:27:49.400000000	192.168.1.5	224.0.0.251	MDNS	5353	5353					
76	04:27:49.400000000	192.168.1.5	224.0.0.251	MDNS	5353	5353					
77	04:27:49.400000000	192.168.1.5	224.0.0.251	MDNS	5353	5353					
79	04:27:49.400000000	192.168.1.5	224.0.0.251	MDNS	5353	5353					
81	04:28:06.625940000	192.168.1.5	8.8.8.8	DNS	56595	53					
84	04:28:06.625940000	192.168.1.5	8.8.8.8	DNS	56595	53					
85	04:28:06.625940000	192.168.1.5	8.8.8.8	DNS	61352	53					
88	04:28:06.625940000	192.168.1.5	8.8.8.8	DNS	53	50813					
89	04:28:06.625940000	192.168.1.5	8.8.8.8	DNS	56595	53					
90	04:28:06.703168000	8.8.8.8	192.168.1.5	DNS	53	61352					
91	04:28:06.706003000	8.8.8.8	192.168.1.5	DNS	53	56595					
95	04:28:06.795130000	192.168.1.5	8.8.8.8	DNS	49709	53					
96	04:28:06.878745000	192.168.1.5	8.8.8.8	DNS	53	49709					
97	04:28:06.879514000	192.168.1.5	8.8.8.8	DNS	57840	53					
98	04:28:06.959048000	192.168.1.5	8.8.8.8	DNS	53	57840					
101	04:28:07.097313000	192.168.1.5	8.8.8.8	DNS	55779	53					
102	04:28:07.182683000	192.168.1.5	8.8.8.8	DNS	53	55779					
103	04:28:07.183420000	192.168.1.5	8.8.8.8	DNS	64452	53					
104	04:28:07.280520000	192.168.1.5	8.8.8.8	DNS	53	64452					
106	04:28:07.390375000	192.168.1.5	8.8.8.8	DNS	57450	53					
107	04:28:07.480253000	192.168.1.5	8.8.8.8	DNS	53	57450					
108	04:28:07.480915000	192.168.1.5	8.8.8.8	DNS	51593	53					
109	04:28:07.562570000	192.168.1.5	8.8.8.8	DNS	53	51593					
110	04:28:07.578212000	192.168.1.5	8.8.8.8	DNS	60263	53					
111	04:28:07.664183000	192.168.1.5	8.8.8.8	DNS	53	60263					
112	04:28:07.664879000	192.168.1.5	8.8.8.8	DNS	58447	53					
113	04:28:07.748624000	192.168.1.5	8.8.8.8	DNS	53	58447					
114	04:28:07.802389000	192.168.1.5	8.8.8.8	DNS	58721	53					

**"dns && !mdns" isn't a valid display filter: "mdns" is neither a field nor a protocol name.**

See the help for a description of the display filter syntax.

OK

We can see from this that we don't have an exact filter for MDNS. So, how do we filter the MDNS packets out? We can see that the MDNS protocol communicates over port 5353. Let's filter that out instead of using an !mdns filter, as shown in the following screenshot:

Filter: dns and (udp.port eq 5353)							Expression...	Clear	Save	VPS	UpdateCheck	Live	Data	Data(Post)	HTTP	UP Filet
Number	Time	Source	Interval	Protocol	Source Port	Destination Port										
36	04:27:54.718441000	192.168.1.5	8.8.8.8	DNS	58405	53	Info	Standard query 0xd3ba A wpad.TOTOLINK								
37	04:27:54.799536000	8.8.8.8	192.168.1.5	DNS	53	58405	Standard query response 0xd3ba No such name									
76	04:28:06.447820000	192.168.1.5	8.8.8.8	DNS	60397	53	Standard query 0xcb15 A vodafone.in									
77	04:28:06.534406000	192.168.1.5	8.8.8.8	DNS	64776	53	Standard query 0x8f2a A vapt.io									
79	04:28:06.538814000	8.8.8.8	192.168.1.5	DNS	53	60397	Standard query response 0xcb15 A 103.75.248.133									
81	04:28:06.539758000	192.168.1.5	8.8.8.8	DNS	50813	53	Standard query 0xcfdf9 A vodafone.in									
84	04:28:06.618832000	8.8.8.8	192.168.1.5	DNS	53	64776	Standard query response 0x8f2a									
85	04:28:06.619572000	192.168.1.5	8.8.8.8	DNS	61352	53	Standard query 0x30db A vapt.io									
88	04:28:06.625408000	8.8.8.8	192.168.1.5	DNS	53	50813	Standard query response 0xcfdf9 A 103.75.248.133									
89	04:28:06.625940000	192.168.1.5	8.8.8.8	DNS	56595	53	Standard query 0x9eal AAAA vodafone.in									
90	04:28:06.703168000	8.8.8.8	192.168.1.5	DNS	53	61352	Standard query response 0x30db									
91	04:28:06.706003000	8.8.8.8	192.168.1.5	DNS	53	56595	Standard query response 0x9eal									
95	04:28:06.795130000	192.168.1.5	8.8.8.8	DNS	49709	53	Standard query 0x17d3 A vapt.io									
96	04:28:06.878745000	192.168.1.5	8.8.8.8	DNS	53	49709	Standard query response 0x17d3									
97	04:28:06.879514000	192.168.1.5	8.8.8.8	DNS	57840	53	Standard query 0x0e9a A vapt.io									
98	04:28:06.959048000	192.168.1.5	8.8.8.8	DNS	53	57840	Standard query response 0x0e9a									
101	04:28:07.097313000	192.168.1.5	8.8.8.8	DNS	55779	53	Standard query 0xde74 A vapt.io									
102	04:28:07.182683000	192.168.1.5	8.8.8.8	DNS	53	55779	Standard query response 0xde74									
103	04:28:07.183420000	192.168.1.5	8.8.8.8	DNS	64452	53	Standard query 0x2f88 A vapt.io									
104	04:28:07.280520000	192.168.1.5	8.8.8.8	DNS	53	64452	Standard query response 0x2f88									
106	04:28:07.390375000	192.168.1.5	8.8.8.8	DNS	57450	53	Standard query 0xe408 A vapt.io									
107	04:28:07.480253000	192.168.1.5	8.8.8.8	DNS	53	57450	Standard query response 0xe408									
108	04:28:07.480915000	192.168.1.5	8.8.8.8	DNS	51593	53	Standard query 0xd43b A vapt.io									
109	04:28:07.562570000	192.168.1.5	8.8.8.8	DNS	53	51593	Standard query response 0xd43b									
110	04:28:07.578212000	192.168.1.5	8.8.8.8	DNS	60263	53	Standard query 0xbafe A vapt.io									
111	04:28:07.664183000	192.168.1.5	8.8.8.8	DNS	53	60263	Standard query response 0xbafe									
112	04:28:07.664879000	192.168.1.5	8.8.8.8	DNS	58447	53	Standard query 0xc622 A vapt.io									
113	04:28:07.748624000	192.168.1.5	8.8.8.8	DNS	53	58447	Standard query response 0xc622									
114	04:28:07.802389000	192.168.1.5	8.8.8.8	DNS	58721	53	Standard query 0x322e A tiles.services.mozilla.com									

We can see that providing the filter `dns` and `!(udp.port eq 5353)` presents us with only the DNS packets. Here, `eq` means equal, the `!` means NOT, and `udp.port` means the UDP port. This means that, in layman's terms, we are asking Wireshark to filter DNS packets while removing all the packets that communicate over UDP port 5353.



In the latest version of Wireshark `mdns` is a valid protocol and display filter such as `dns && !mdns` works fine.

Similarly, for HTTP, we can type in `http` as the filter, as shown in the following screenshot:

Number	Time	Source	Destination	Protocol	Source Port	Destination Port	Info
281	04/28/08, 7:55:56.300000	117.18.237.29	192.168.1.5	OCSP	80	6956	Response
295	04/28/08, 7:58:12.500000	117.18.237.29	192.168.1.5	OCSP	80	6956	Response
326	04/28/08, 8:03:36.040000	104.24.121.103	192.168.1.5	HTTP	80	6989	HTTP/1.1 200 OK (text/css)
338	04/28/08, 8:06:47.206000	104.24.121.103	192.168.1.5	HTTP	80	6987	HTTP/1.1 200 OK (application/javascript)
340	04/28/08, 8:07:42.900000	104.24.121.103	192.168.1.5	HTTP	80	6988	HTTP/1.1 200 OK (application/javascript)
446	04/28/08, 8:44:00.000000	104.24.121.103	192.168.1.5	HTTP	80	6989	HTTP/1.1 200 OK (application/font-woff)
451	04/28/08, 8:44:07.900000	104.24.121.103	192.168.1.5	HTTP	80	6988	HTTP/1.1 200 OK (application/font-woff)
454	04/28/08, 0:04:06:58.000000	104.24.121.103	192.168.1.5	HTTP	80	6987	HTTP/1.1 200 OK (application/font-woff)
473	04/28/08, 0:09:14.038:60000	104.24.121.103	192.168.1.5	HTTP	80	6988	HTTP/1.1 530 (text/html)
522	04/28/08, 0:09:17:55:43:1000	111.206.66.10	192.168.1.5	HTTP	80	6992	[TCP Out-of-Order] HTTP/1.1 200 OK (application/octet-stream)
528	04/28/08, 0:09:17:57:18:6000	111.206.66.10	192.168.1.5	HTTP	80	6991	[TCP Out-of-Order] HTTP/1.1 200 OK (application/octet-stream)
706	04/28/29, 9:02:00.000000	104.192.108.133	192.168.1.5	HTTP	80	6997	HTTP/1.1 200 OK (application/octet-stream)
1222	04/28/29, 9:02:00.000000	104.192.108.133	192.168.1.5	HTTP	80	6996	HTTP/1.1 200 OK (application/octet-stream)
1330	04/28/29, 9:02:00.000000	104.192.108.133	192.168.1.5	HTTP	80	7001	HTTP/1.1 200 OK (application/octet-stream)
1413	04/28/45, 4:49:04:500000	104.192.108.133	192.168.1.5	HTTP	80	7008	HTTP/1.1 200 OK (application/octet-stream)
2669	04/28/45, 4:49:04:500000	104.192.108.133	192.168.1.5	HTTP	80	7051	HTTP/1.1 200 OK (application/octet-stream)
2737	04/33:06, 24:02:08:000000	104.192.108.133	192.168.1.5	HTTP	80	7056	HTTP/1.1 200 OK (application/octet-stream)
4029	04/37:05, 9:59:06:000000	104.192.108.107	192.168.1.5	HTTP	80	7082	HTTP/1.1 200 OK (application/octet-stream)
4131	04/37:14, 5:02:51:000000	104.192.108.163	192.168.1.5	HTTP	80	7089	HTTP/1.1 200 OK (application/octet-stream)
4154	04/37:44, 5:02:51:000000	104.192.108.163	192.168.1.5	HTTP	80	7090	HTTP/1.1 200 OK (application/octet-stream)
4384	04/38:52, 6:07:88:200000	36.110.236.239	192.168.1.5	HTTP/20	80	7096	HTTP/1.1 200 OK [Malformed Packet]
174	04/28:08, 0:28:56:700000	192.168.1.5	216.58.196.206	OCSP	6985	80	Request
1743	04/29:48, 2:48:46:500000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
1744	04/29:48, 2:48:49:440000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
1745	04/29:48, 2:48:50:000000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
1746	04/29:48, 2:48:51:59:700000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
1747	04/29:48, 2:48:56:29:800000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
1748	04/29:48, 2:48:56:48:600000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
1749	04/29:48, 2:48:56:49:300000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
1750	04/29:48, 2:48:56:50:200000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
2567	04/32:50, 7:09:27:000000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
2568	04/32:50, 7:09:27:00:000000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
2569	04/32:50, 8:02:44:500000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
2700	04/32:50, 8:02:44:500000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
2701	04/32:50, 8:02:44:500000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
2702	04/32:50, 8:02:44:500000	192.168.1.1	239.255.255.250	SSDP	43141	1900	NOTIFY * HTTP/1.1
0000	11110100 11100100 01101100 00011101 01010000 00011001 01110100 01000100 ..m..,0x0						
0008	01110110 11100111 10110000 01010100 00001000 00000000 01000100 00000000 v.,T,E.						
0010	00000001 00100000 00000000 00100000 00000000 00000000 00010001 00010001 ..,..,						
0011	00000000 00000000 00000000 00000000 00000000 00000000 00010001 00010001 ..,..,						
File "C:\Users\Apex\AppData\Local\Temp\wireshar...	Packets: 4507 - Displayed: 90 (2.0%) - Dropped: 0 (0.0%)						Profile: Default

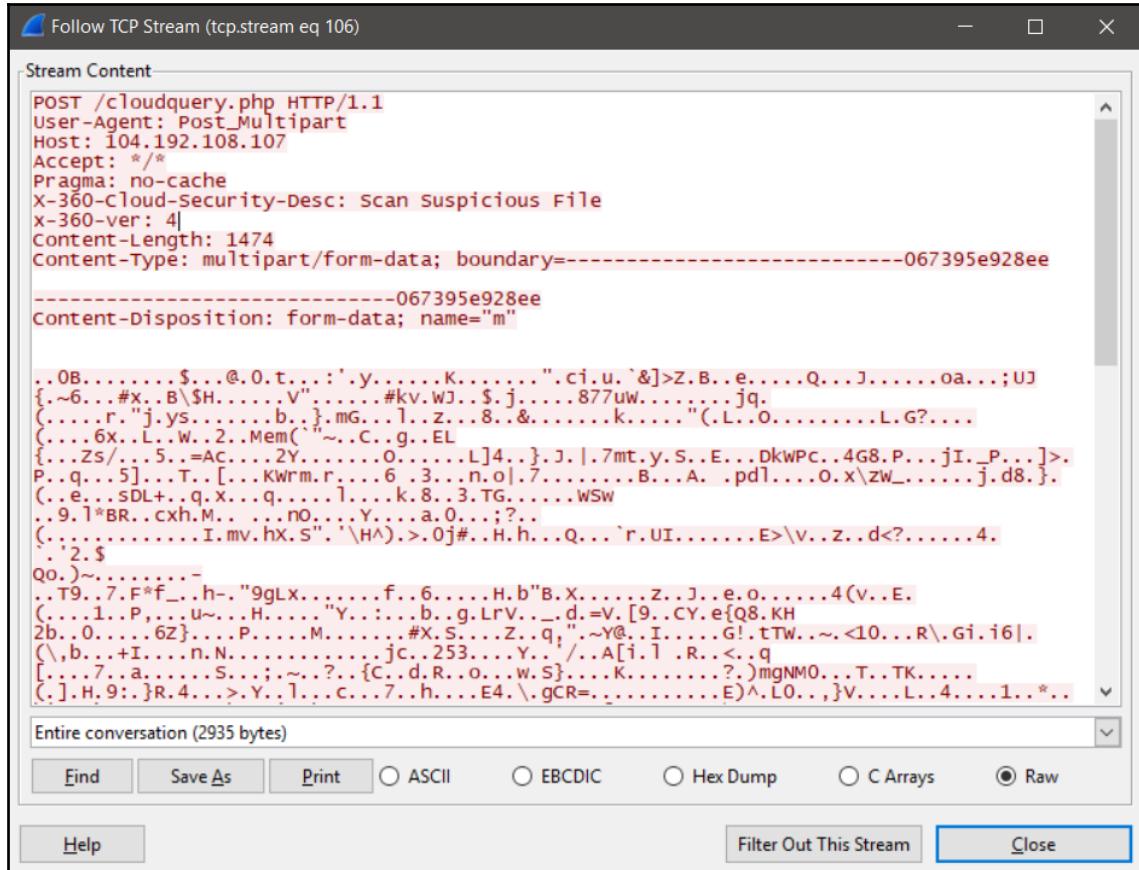
However, we also have OCSP and **Simple Service Discovery Protocol (SSDP)** protocol data alongside the data that is filtered from the stream. To filter out the OCSP and SSDP protocol data, we can type in `http && !ocsp`, and since SSDP poses a similar problem to MDNS, we can type `!udp.port==1900`. This means that the entire filter becomes `http && !ocsp && !udp.port==1900`, as shown in the following screenshot:

Number	Time	Source	Interval	Protocol	Source Port	Destination Port	Info
102.11 Channel	Filter: http && locsp && !udp.port=1900	192.168.1.5	104.192.108.107	HTTP	6997	80	POST /cloudquery.php HTTP/1.1
702	04:28:19.709801000	192.168.1.5	104.192.108.133	HTTP	6997	80	POST /qexquery HTTP/1.1
1280	04:28:35.802854000	192.168.1.5	104.192.108.133	HTTP	7000	80	POST /qexquery HTTP/1.1
1320	04:28:37.299392000	192.168.1.5	104.192.108.133	HTTP	7001	80	POST /qexquery HTTP/1.1
1411	04:28:45.211083000	192.168.1.5	104.192.108.133	HTTP	7008	80	POST /qexquery HTTP/1.1
2659	04:33:03.238067000	192.168.1.5	104.192.108.133	HTTP	7051	80	POST /qexquery HTTP/1.1
2774	04:33:05.880358000	192.168.1.5	104.192.108.133	HTTP	7056	80	POST /qexquery HTTP/1.1
249	04:28:08.590895000	192.168.1.5	104.24.121.103	HTTP	6986	80	GET / HTTP/1.1
299	04:28:08.782846000	192.168.1.5	104.24.121.103	HTTP	6989	80	GET /cdn-cgi/styles/cf.errors.css HTTP/1.1
306	04:28:08.785656000	192.168.1.5	104.24.121.103	HTTP	6987	80	GET /cdn-cgi/scripts/zepto.min.js HTTP/1.1
309	04:28:08.787393000	192.168.1.5	104.24.121.103	HTTP	6988	80	GET /cdn-cgi/scripts/cf.common.js HTTP/1.1
394	04:28:08.956761000	192.168.1.5	104.24.121.103	HTTP	6989	80	GET /cdn-cgi/styles/fonts/opensans-400.woff HTTP/1.1
395	04:28:08.957001000	192.168.1.5	104.24.121.103	HTTP	6989	80	GET /cdn-cgi/styles/fonts/opensans-300.woff HTTP/1.1
396	04:28:08.960113000	192.168.1.5	104.24.121.103	HTTP	6988	80	GET /cdn-cgi/styles/fonts/opensans-600.woff HTTP/1.1
456	04:28:09.051431000	192.168.1.5	104.24.121.103	HTTP	6988	80	GET /favicon.ico HTTP/1.1
498	04:28:09.399384000	192.168.1.5	111.206.66.10	HTTP	6992	80	POST /wdinfo.php HTTP/1.1 (application/octet-stream)
502	04:28:09.417811000	192.168.1.5	111.206.66.10	HTTP	6991	80	POST /wdinfo.php HTTP/1.1 (application/octet-stream)
4122	04:37:41.470246000	192.168.1.5	13.35.190.163	HTTP	7089	80	GET /v3/pc/360safe/isafeup_1ib.cab?mid=8230303bb8689b7d94af75c3eb8cbe0&ver=10.2.0.117
282	04:28:09.683361000	104.24.121.103	192.168.1.5	HTTP	80	8866	HTTP/1.1 200 (text/css)
326	04:28:08.856040000	104.24.121.103	192.168.1.5	HTTP	80	6989	HTTP/1.1 200 (text/css)
338	04:28:08.867206000	104.24.121.103	192.168.1.5	HTTP	80	6987	HTTP/1.1 200 OK (application/javascript)
340	04:28:08.874429000	104.24.121.103	192.168.1.5	HTTP	80	6988	HTTP/1.1 200 OK (application/javascript)
446	04:28:09.044053000	104.24.121.103	192.168.1.5	HTTP	80	6989	HTTP/1.1 200 OK (application/font-woff)
451	04:28:09.048079000	104.24.121.103	192.168.1.5	HTTP	80	6988	HTTP/1.1 200 OK (application/font-woff)
454	04:28:09.048080000	104.24.121.103	192.168.1.5	HTTP	80	6987	HTTP/1.1 200 OK (application/font-woff)
471	04:28:09.140386000	104.24.121.103	192.168.1.5	HTTP	80	6986	HTTP/1.1 200 OK (application/font-woff)
522	04:28:09.755416000	111.206.66.10	192.168.1.5	HTTP	80	6902	[TCP OUT-OF-ORDER] HTTP/1.1 200 OK (application/octet-stream)
528	04:28:09.797186000	111.206.66.10	192.168.1.5	HTTP	80	6991	[TCP OUT-OF-ORDER] HTTP/1.1 200 OK (application/octet-stream)
706	04:28:29.992001000	104.192.108.133	192.168.1.5	HTTP	80	6997	HTTP/1.1 200 OK (application/octet-stream)
1290	04:28:36.206662000	104.192.108.133	192.168.1.5	HTTP	80	7000	HTTP/1.1 200 OK (application/octet-stream)
1370	04:28:37.388126000	104.192.108.133	192.168.1.5	HTTP	80	7000	HTTP/1.1 200 OK (application/octet-stream)
1413	04:28:37.400550000	104.192.108.133	192.168.1.5	HTTP	80	7009	HTTP/1.1 200 OK (application/octet-stream)
2669	04:33:03.582669000	104.192.108.133	192.168.1.5	HTTP	80	7051	HTTP/1.1 200 OK (application/octet-stream)
2737	04:33:06.242088000	104.192.108.133	192.168.1.5	HTTP	80	7056	HTTP/1.1 200 OK (application/octet-stream)
4029	04:37:05.959866000	104.192.108.107	192.168.1.5	HTTP	80	7082	HTTP/1.1 200 OK (application/octet-stream)
4136	04:37:41.495140000	13.35.190.163	192.168.1.5	HTTP	80	7089	HTTP/1.1 200 OK (application/octet-stream)

We can see from this that we have successfully filtered HTTP packets. But can we search through them and filter only HTTP POST packets? Yes, we can, using the expression `http contains POST && !ocsp` as shown in the following screenshot.

Time	Source	Interval	Protocol	Source Port	Destination Port	Info
04:37:05.666390000	192.168.1.5	104.192.108.107	HTTP	7082	80	POST /cloudquery.php HTTP/1.1
04:28:29.709801000	192.168.1.5	104.192.108.133	HTTP	6997	80	POST /qexquery HTTP/1.1
04:28:35.802854000	192.168.1.5	104.192.108.133	HTTP	7000	80	POST /qexquery HTTP/1.1
04:28:37.299392000	192.168.1.5	104.192.108.133	HTTP	7001	80	POST /qexquery HTTP/1.1
04:28:45.211083000	192.168.1.5	104.192.108.133	HTTP	7008	80	POST /qexquery HTTP/1.1
04:33:03.238067000	192.168.1.5	104.192.108.133	HTTP	7051	80	POST /qexquery HTTP/1.1
04:33:05.880358000	192.168.1.5	104.192.108.133	HTTP	7056	80	POST /qexquery HTTP/1.1
04:28:09.399384000	192.168.1.5	111.206.66.10	HTTP	6992	80	POST /wdinfo.php HTTP/1.1 (application/octet-stream)
04:28:09.417811000	192.168.1.5	111.206.66.10	HTTP	6991	80	POST /wdinfo.php HTTP/1.1 (application/octet-stream)

We can see that providing the `HTTP` contains `POST` filter filters out all the non-`HTTP` `POST` requests. Let's analyze the request by right-clicking and selecting the option to follow the `HTTP` stream, as shown in the following screenshot:



The screenshot shows a NetworkMiner window titled "Follow TCP Stream (tcp.stream eq 106)". The main pane displays the "Stream Content" of a POST request. The request includes the following headers:

```
POST /cloudquery.php HTTP/1.1
User-Agent: Post_Multipart
Host: 104.192.108.107
Accept: /*
Pragma: no-cache
X-360-Cloud-Security-Desc: Scan Suspicious File
x-360-ver: 4
Content-Length: 1474
Content-Type: multipart/form-data; boundary=-----067395e928ee
```

The body of the POST request starts with a boundary line: `-----067395e928ee`. It then contains a `Content-Disposition: form-data; name="m"` section, followed by a large amount of encoded file data.

At the bottom of the window, there is a toolbar with buttons for "Find", "Save As", "Print", and "Raw". The "Raw" button is selected. Below the toolbar, there are buttons for "ASCII", "EBCDIC", "Hex Dump", "C Arrays", and "Raw".

We can see that this looks like a file that has been sent out somewhere, but since it has headers such as `x-360-cloud-security-desc`, it looks as though it's the cloud antivirus that is scanning a suspicious file found on the network.

Let's take note of the IP address and match it with the address resolutions, as shown in the following screenshot:

```
# Address resolution IPv4 Hash table
#
# with 120 entries
#
Key:0x4d832734 IP: 52.39.131.77, Name: tiles.r53-2.services.mozilla.com
Key:0x70fd1ec0 IP: 192.30.253.112, Name: github.com
Key:0x3527da2 IP: 162.125.82.3, Name: client.dropbox-dns.com
Key:0x1f7b1868 IP: 104.24.123.31, Name: certcollection.org
Key:0x3ad9ac IP: 172.217.161.3, Name: ssl.gstatic.com
Key:0x32ab0732 IP: 50.7.171.50, Name: qihoo360.cdnvideo.ru
Key:0xcea6d9ac IP: 172.217.166.206, Name: plus.l.google.com
Key:0x6b6cc068 IP: 104.192.108.107, Name: 104.192.108.107
Key:0x16b1d322 IP: 34.211.177.22, Name: webextensions.settings.services.mozilla.com
```

Well, the address resolutions have failed us this time. Let's search the IP on <https://whois/is/>, as shown in the following screenshot:

104.192.108.107 address profile

Whois      Diagnostics

IP Whois

CHINA TELECOM (AMERICAS) CORPORATION CHINANET-LAX-IDC-2014 (NET-104-192-108-0-1) 104.192.108.0 - 104.192.111.255  
Qihoo 360 Inc. CTA-104-192-108-0-23 (NET-104-192-108-0-2) 104.192.108.0 - 104.192.109.255

Yes, it belongs to the QiHU 360 antivirus.

We can also select HTTP packets based on the response codes, as shown in the following screenshot:

Time	Source	Interval	Protocol	Source Port	Destination Port	Info
04:28:08.863604000	104.24.121.103	192.168.1.5	HTTP	80	6989	HTTP/1.1 200 OK (text/css)
04:28:08.867206000	104.24.121.103	192.168.1.5	HTTP	80	6987	HTTP/1.1 200 OK (application/javascript)
04:28:08.874429000	104.24.121.103	192.168.1.5	HTTP	80	6988	HTTP/1.1 200 OK (application/javascript)
04:28:09.044053000	104.24.121.103	192.168.1.5	HTTP	80	6989	HTTP/1.1 200 OK (application/font-woff)
04:28:09.048079000	104.24.121.103	192.168.1.5	HTTP	80	6988	HTTP/1.1 200 OK (application/font-woff)
04:28:09.049658000	104.24.121.103	192.168.1.5	HTTP	80	6987	HTTP/1.1 200 OK (application/font-woff)
04:28:09.755431000	111.206.66.10	192.168.1.5	HTTP	80	6992	[TCP out-of-order] HTTP/1.1 200 OK (application/octet-stream)
04:28:09.797186000	111.206.66.10	192.168.1.5	HTTP	80	6991	[TCP out-of-order] HTTP/1.1 200 OK (application/octet-stream)
04:28:29.992001000	104.192.108.133	192.168.1.5	HTTP	80	6997	HTTP/1.1 200 OK (application/octet-stream)
04:28:36.206662000	104.192.108.133	192.168.1.5	HTTP	80	7000	HTTP/1.1 200 OK (application/octet-stream)
04:28:37.588120000	104.192.108.133	192.168.1.5	HTTP	80	7001	HTTP/1.1 200 OK (application/octet-stream)
04:28:45.499945000	104.192.108.133	192.168.1.5	HTTP	80	7008	HTTP/1.1 200 OK (application/octet-stream)
04:33:03.582669000	104.192.108.133	192.168.1.5	HTTP	80	7051	HTTP/1.1 200 OK (application/octet-stream)
04:33:06.242080000	104.192.108.133	192.168.1.5	HTTP	80	7056	HTTP/1.1 200 OK (application/octet-stream)
04:37:05.959866000	104.192.108.107	192.168.1.5	HTTP	80	7082	HTTP/1.1 200 OK (application/octet-stream)
04:37:41.495514000	13.35.190.163	192.168.1.5	HTTP	80	7089	HTTP/1.1 200 OK (application/octet-stream)
04:37:44.590761000	50.7.171.50	192.168.1.5	HTTP	80	7090	HTTP/1.1 200 OK (application/octet-stream)
04:38:52.687882000	36.110.236.239	192.168.1.5	HTTP/x0	80	7096	HTTP/1.1 200 OK [Malformed Packet]

We can see that we have filtered the packets using `http.response.code==200`, where 200 denotes a status OK response. This is handy when investigating packet captures from compromised servers, as it gives us a clear picture of the files that have been accessed and shows us how the server responded to particular requests.

It also allows us to figure out whether the implemented protections are working well, because upon receiving a malicious request, in most cases, the protection firewall issues a **404 (NOT FOUND)** or a **403 (Forbidden)** response code instead of 200 (OK).

Let's now jump into some case studies and make use of the basics that we just learned.

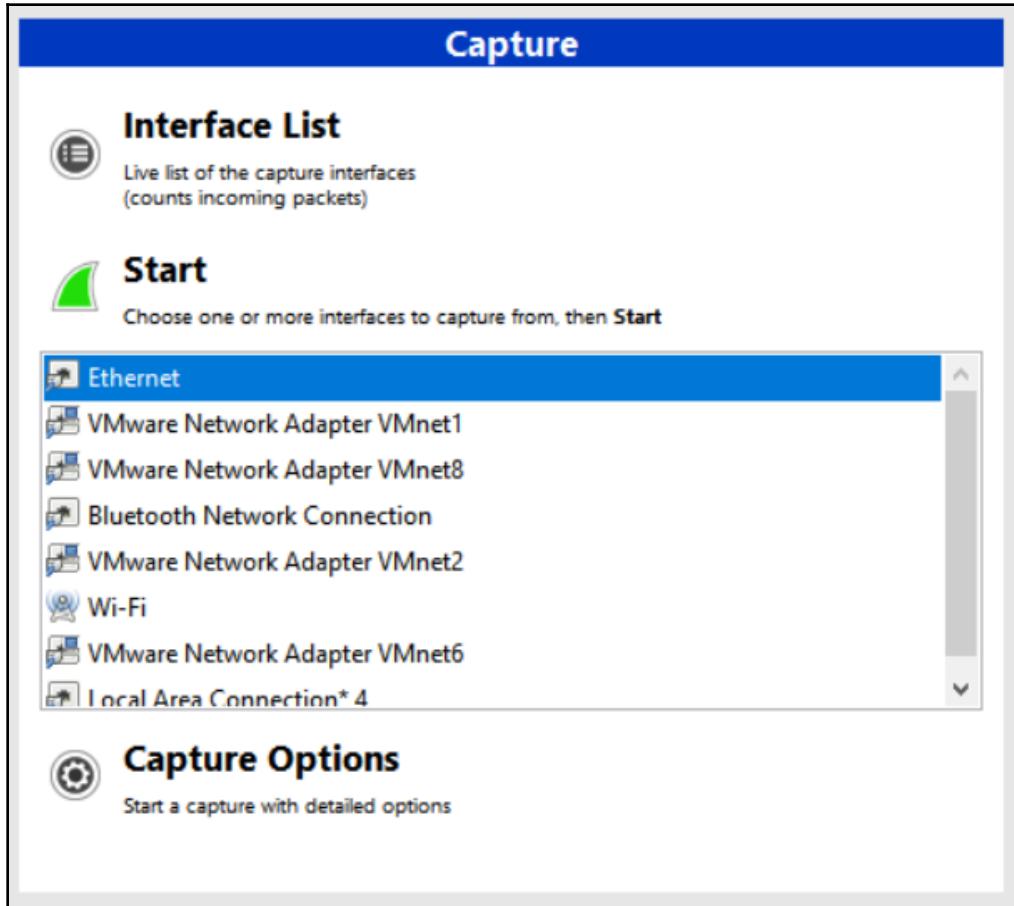
## Exercise 1 – a noob's keylogger

Consider a scenario where an attacker has planted a keylogger on one of the systems in the network. Your job as an investigator is to find the following pieces of information:

- Find the infected system
- Trace the data to the server
- Find the frequency of the data that is being sent
- Find what other information is carried besides the keystrokes
- Try to uncover the attacker
- Extract and reconstruct the files that have been sent to the attacker

Additionally, in this exercise, you need to assume that the **packet capture (PCAP)** file is not available and that you have to do the sniffing-out part as well. Let's say that you are connected to a mirror port on the network where you can see all the data traveling to and from the network.

We can begin our process as follows. We already know that we are connected via a mirror port. Let's sniff around on the interface of choice. If connected to the mirror port, choose the default interface and proceed with collecting packets, as shown in the following screenshot:



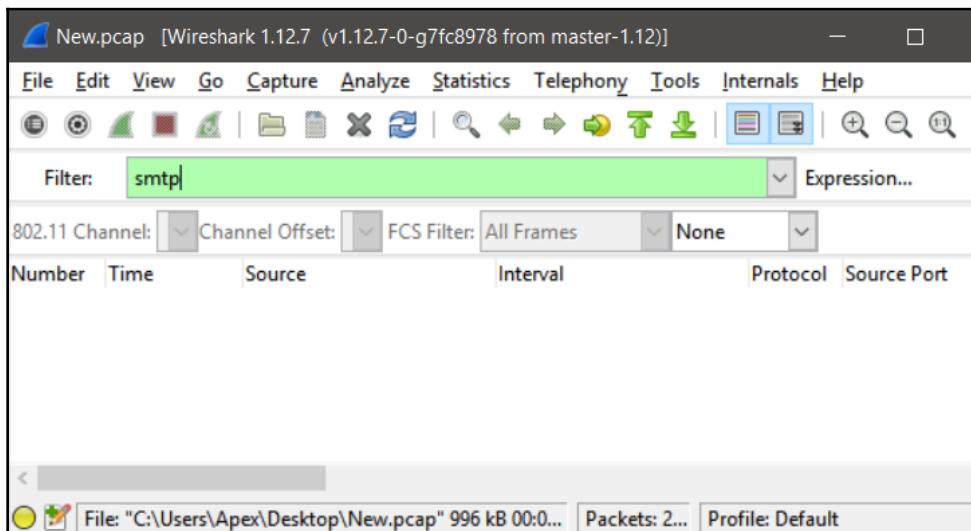
Most keyloggers work on the web (HTTP), FTP, and email for delivering the keystrokes back to the attacker. We will try all of these to check whether there's anything unusual with packets from these protocols.

Let's try HTTP first by setting the `http` filter, as shown in the following screenshot:

Number	Time	Source	Interval	Protocol	Source Port	Destination Port	Info
138	0.000000	192.168.76.131	239, 255, 255, 250	SSDP	49541	1900	M-SEARCH * HTTP/1.1
140	0.025473	192.168.76.131	239, 255, 255, 250	SSDP	49541	1900	M-SEARCH * HTTP/1.1
141	0.061373	192.168.76.131	239, 255, 255, 250	SSDP	49541	1900	M-SEARCH * HTTP/1.1
180	0.001562	117.18.237.29	HTTP	51708	80	80	GET /msdownload/update/v3/static/trustedr/en/disallowcertst1.cab?b7c9442bd2afef18 HTTP/1.1
182	0.001561	f800::98c:a52c:6ffff02::c	SSDP	49539	1900	M-SEARCH * HTTP/1.1	
183	0.000229	192.168.76.131	239, 255, 255, 250	SSDP	49541	1900	M-SEARCH * HTTP/1.1
184	0.014907	117.18.237.29	192.168.76.131	OCSP	80	51652	Response
218	0.728099	192.168.76.131	239, 255, 255, 250	SSDP	49541	1900	M-SEARCH * HTTP/1.1
363	2.266679	f800::98c:a52c:6ffff02::c	SSDP	49539	1900	M-SEARCH * HTTP/1.1	
364	2.266679	f800::98c:a52c:6ffff02::c	SSDP	49541	1900	M-SEARCH * HTTP/1.1	
534	3.029123	f800::98c:a52c:6ffff02::c	SSDP	49539	1900	M-SEARCH * HTTP/1.1	
535	0.000428	192.168.76.131	239, 255, 255, 250	SSDP	49541	1900	M-SEARCH * HTTP/1.1
839	24.159910	f800::98c:a52c:6ffff02::c	SSDP	49539	1900	M-SEARCH * HTTP/1.1	
840	0.000274	192.168.76.131	239, 255, 255, 250	SSDP	49541	1900	M-SEARCH * HTTP/1.1
905	3.014574	f800::98c:a52c:6ffff02::c	SSDP	49539	1900	M-SEARCH * HTTP/1.1	
906	3.002217	192.168.76.131	239, 255, 255, 250	SSDP	49541	1900	M-SEARCH * HTTP/1.1
1065	0.000256	192.168.76.131	239, 255, 255, 250	SSDP	49541	1900	M-SEARCH * HTTP/1.1
1858	24.467684	192.168.76.131	8.253.181.235	HTTP	51702	80	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.cab?c349620299e55c14 HTTP/1.1
1864	0.321851	8.253.181.235	192.168.76.131	HTTP	80	51702	HTTP/1.1 304 Not Modified
1868	0.034518	192.168.76.131	8.253.181.235	HTTP	51702	80	GET /msdownload/update/v3/static/trustedr/en/disallowcertst1.cab?b7c9442bd2afef18 HTTP/1.1
1874	0.171313	8.253.181.235	192.168.76.131	HTTP	80	51702	HTTP/1.1 200 OK (application/vnd.ms-cab-compressed)
1973	0.000217	192.168.76.131	172.217.21.14	HTTP	51708	80	GET /G7TAGC3/MEKwZBMPMWTQJBgurDgkCguABT27b8jYjk8mjx2jXwgnq)KEapsrqqud8k4UjpndnaxLCKG0Igfqz%2Buk5
1982	0.010345	172.217.21.14	192.168.76.131	OCSP	80	51703	Response
2023	1.652017	192.168.76.131	8.253.181.235	HTTP	51705	80	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.cab?36ad90b4d45724d HTTP/1.1
2032	0.239709	8.253.181.235	192.168.76.131	HTTP	80	51705	HTTP/1.1 304 Not Modified
2033	0.035317	192.168.76.131	8.253.181.235	HTTP	51705	80	GET /msdownload/update/v3/static/trustedr/en/disallowcertst1.cab?b7c9442bd2afef18 HTTP/1.1
2039	0.180777	8.253.181.235	192.168.76.131	HTTP	80	51705	HTTP/1.1 200 OK (application/vnd.ms-cab-compressed)
2050	0.024114	117.18.237.29	192.168.76.131	HTTP	51708	80	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.cab?b7c9442bd2afef18 HTTP/1.1
2095	0.024119	117.18.237.29	192.168.76.131	OCSP	80	51706	Response
2177	8.065840	192.168.76.131	8.253.224.254	HTTP	51708	80	GET /msdownload/update/v3/static/trustedr/en/pinrulesstl.cab?b7c9442bd2afef18 HTTP/1.1
2185	0.255139	8.253.224.254	192.168.76.131	HTTP	80	51708	HTTP/1.1 200 OK (application/vnd.ms-cab-compressed)
2187	0.013674	192.168.76.131	8.253.224.254	HTTP	51708	80	GET /msdownload/update/v3/static/trustedr/en/disallowcertst1.cab?b7c9442bd2afef18 HTTP/1.1
2193	0.264187	8.253.224.254	192.168.76.131	HTTP	80	51708	HTTP/1.1 200 OK (application/vnd.ms-cab-compressed)

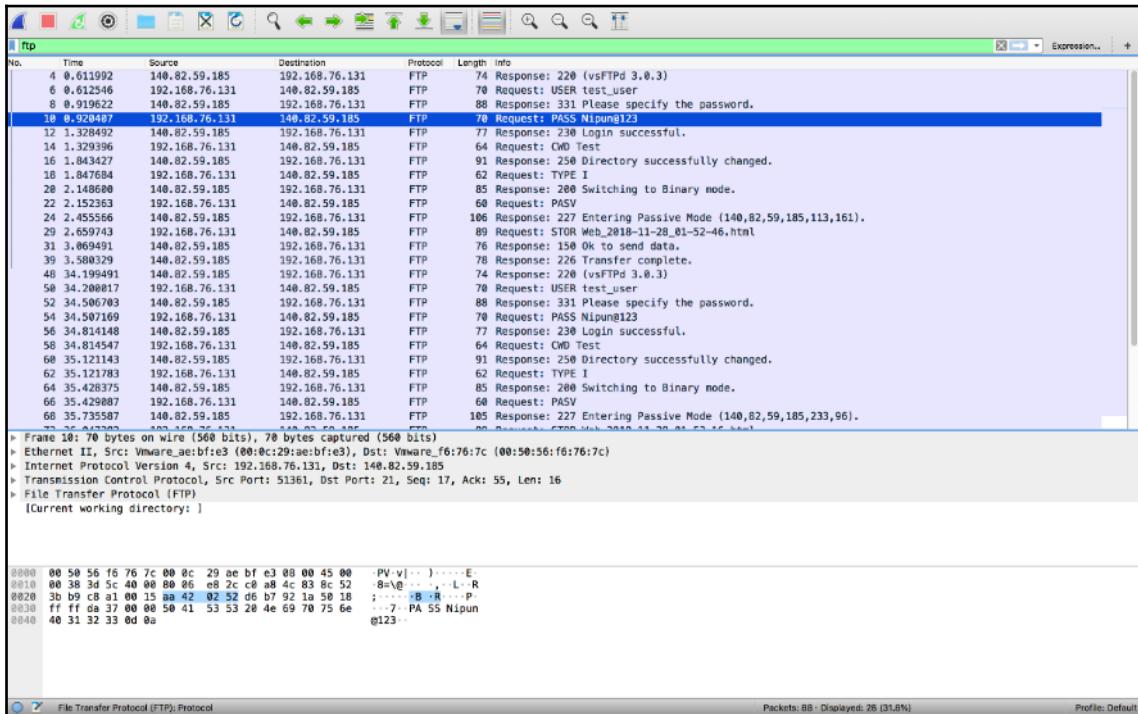
There is HTTP data, but everything seems fine.

Let's try a couple of protocols, SMTP and POP, to check for anything unusual with the email protocol, as shown in the following screenshot:



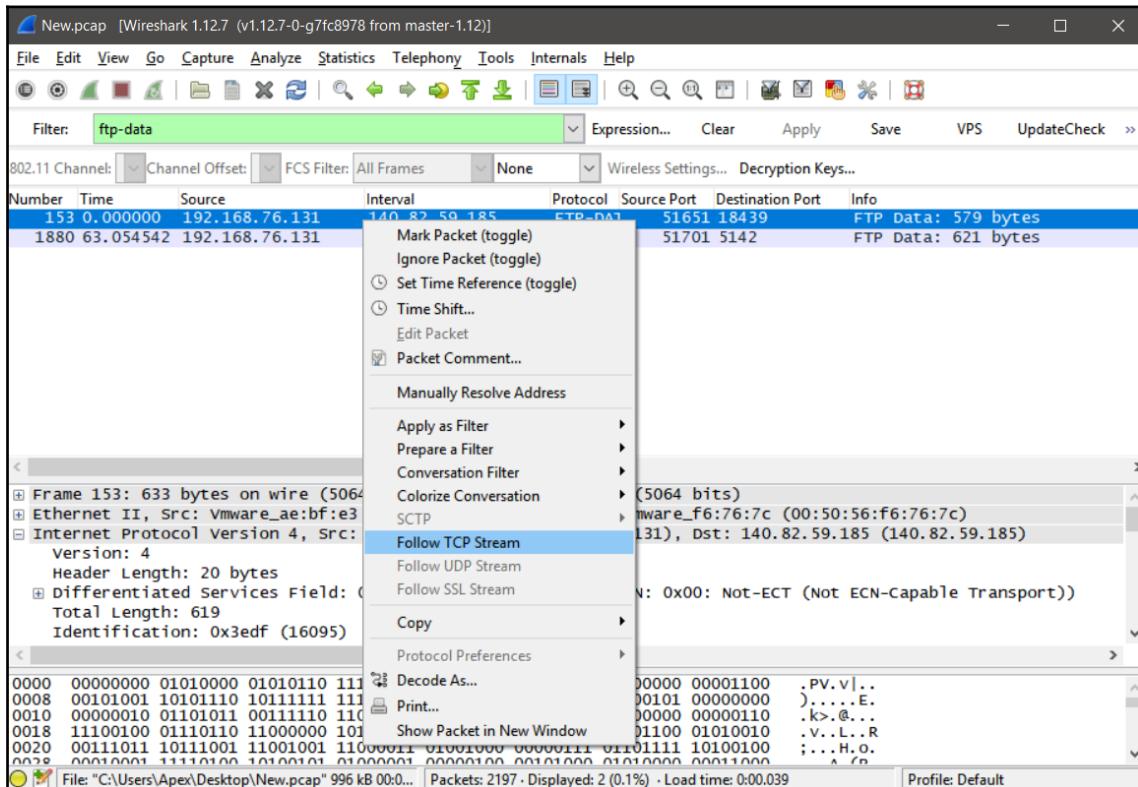
Everything seems fine here as well.

Let's try FTP as well, as shown in the following screenshot:



Well, we have plenty of activity on the FTP! We can see that the FTP packets contain the `USER` and `PASS` commands in the capture, which denotes a login activity to the server. Of course, this can be either the keylogger or a legitimate login from any user on the network. Additionally, we can see a `STOR` command that is used to store files on the FTP server. However, let's note down the credentials and filenames of the uploaded files for our reference and investigate further. Since, we know that the `STOR` command is used to store data on the server.

Let's view these data packets by changing filter to `ftp-data`, as shown in the following screenshot:



## Changing filter to ftp-data



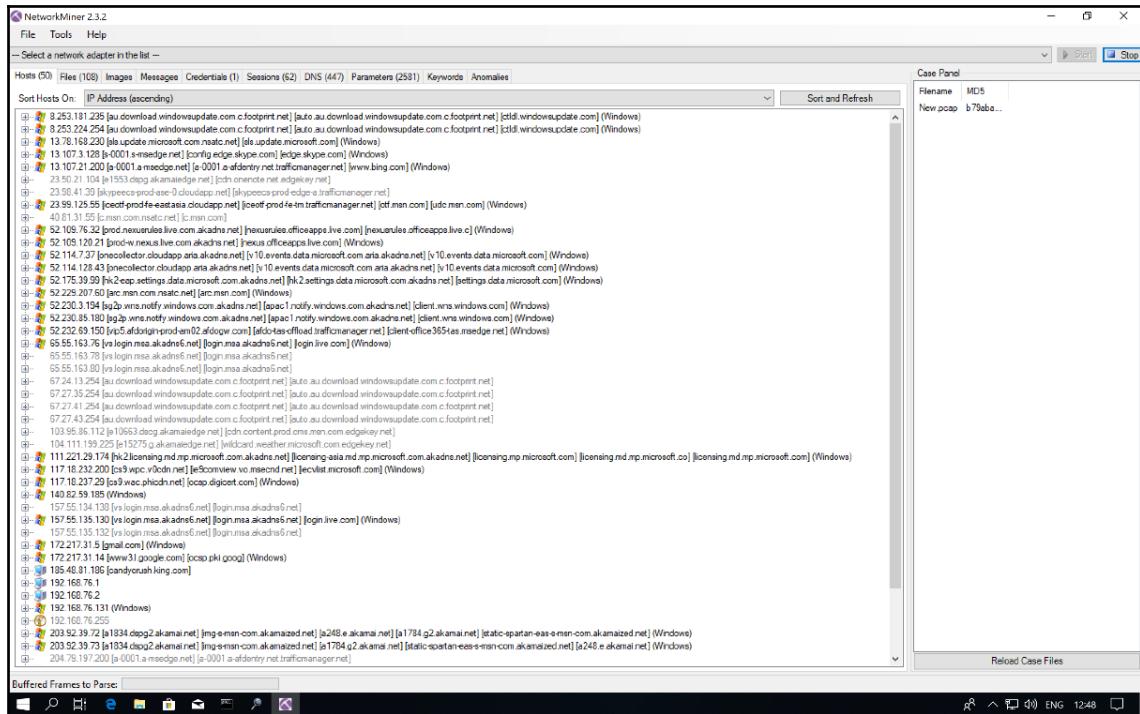
ftp-data will only contain mostly the files and data transferred rather than all the other FTP commands

Let's see what we get when we follow the TCP stream of the packet, we can see that we have the following data being posted to the server:

```
<HTML><HEAD><STYLE>BODY{ BACKGROUND-COLOR: #FFFFFF; FONT-SIZE: 12pt; COLOR: black; FONT-FAMILY:Courier New; }H1{ FONT-FAMILY:Arial; FONT-SIZE: 10pt; FONT-WEIGHT: normal; MARGIN-BOTTOM: 11px; BORDER-STYLE: solid; BORDER-COLOR: #DFDFE5; BORDER-WIDTH: 2px; BACKGROUND-COLOR: #DFDFE5; }H2 { COLOR: black; BACKGROUND-COLOR: #FFFFFF; FONT-SIZE: 12pt; FONT-WEIGHT: normal; MARGIN-BOTTOM: 0px; MARGIN-TOP: 10px; }</STYLE></HEAD><META http-equiv=Content-Type content="text/html; charset=utf-8"><BODY><H1>28 November 2018 [16:04] explorer.exe: Pictures</H1>Ardamax_FTP_Delivery</BODY></HTML>
```

We can see that the data being transmitted contains the word Ardamax, which is the name of a common piece of keylogger software that records keystrokes from the system it has infected and sends it back to the attacker. Let's save the packet capture in PCAP format by selecting **File | Save As** and choosing the .pcap format. We will be using the .pcap format only since the free version of NetworkMiner support only PCAP files and not the pcapng format.

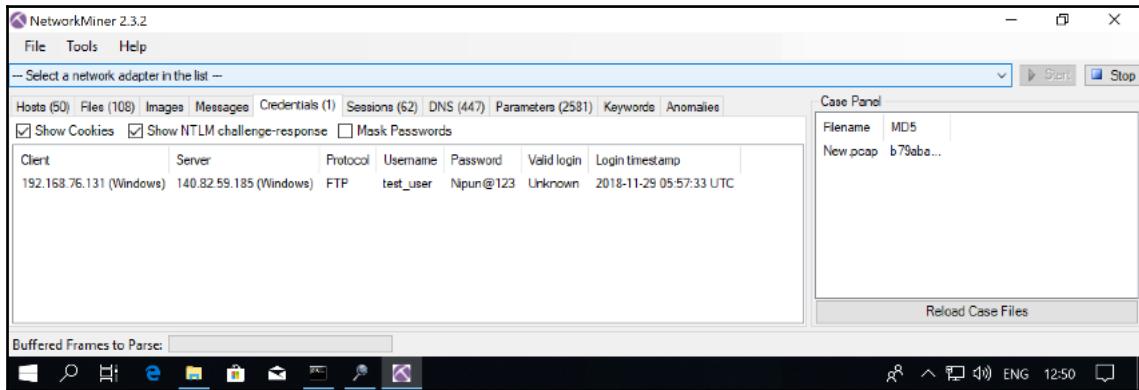
Let's open the saved file using NetworkMiner as shown in the following screenshot:



Opening the saved file using network miner

We can see we have a number of hosts present in the network capture.

Let's navigate to the **Credentials** tab, as shown in the following screenshot:



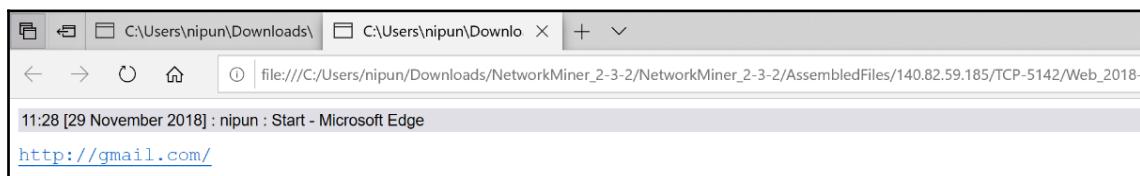
We can see that we have the username and password captured in the PCAP file displayed under **Credentials** tab in NetworkMiner. We previously saw the STOR command, which is commonly used in uploading files to an FTP from the Wireshark dump.

Let's browse to the **Files** tab and see the files that we are interested in:

File list										
Frame nr.	Filename	Extension	Size	Source host	S. port	Destination host	D. port	Protocol	Timestamp	^
1101	Microsoft IT TLS CA 5[4].cer	cer	1 464 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51680	TlsCertificate	2018-11-29 05:58:13 U	
1107	Microsoft IT TLS CA 5[5].cer	cer	1 464 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51683	TlsCertificate	2018-11-29 05:58:13 U	
1112	Microsoft IT TLS CA 5[6].cer	cer	1 464 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51685	TlsCertificate	2018-11-29 05:58:13 U	
1133	Microsoft IT TLS CA 9[1].cer	cer	1 464 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51682	TlsCertificate	2018-11-29 05:58:13 U	
1155	Microsoft IT TLS CA 8[0].cer	cer	1 464 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51687	TlsCertificate	2018-11-29 05:58:13 U	
1226	Microsoft IT TLS CA 5[9].cer	cer	1 464 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51688	TlsCertificate	2018-11-29 05:58:13 U	
319	Microsoft Secure Server CA 2.cer	cer	1 756 B	52.114.128.43 [onecollector.cloudapp.aria.akadns.net] [v1...	TCP 443	192.168.76.131 (Windows)	TCP 51655	TlsCertificate	2018-11-29 05:57:39 U	
824	Microsoft Secure Server CA 2.cer	cer	1 756 B	52.114.128.43 [onecollector.cloudapp.aria.akadns.net] [v1...	TCP 443	192.168.76.131 (Windows)	TCP 51671	TlsCertificate	2018-11-29 05:58:05 U	
368	Microsoft Secure Server CA 2[1].cer	cer	1 756 B	52.114.128.43 [onecollector.cloudapp.aria.akadns.net] [v1...	TCP 443	192.168.76.131 (Windows)	TCP 51658	TlsCertificate	2018-11-29 05:57:40 U	
396	Microsoft Secure Server CA 2[2].cer	cer	1 756 B	52.114.128.43 [onecollector.cloudapp.aria.akadns.net] [v1...	TCP 443	192.168.76.131 (Windows)	TCP 51659	TlsCertificate	2018-11-29 05:57:41 U	
781	Microsoft Update Secure Serv.cer	cer	1 796 B	13.78.168.230 [cls.update.microsoft.com.msact.net] [cls.up...	TCP 443	192.168.76.131 (Windows)	TCP 51670	TlsCertificate	2018-11-29 05:57:58 U	
276	mseguide.net.cer	cer	2 011 B	52.232.69.150 [wp5.adodinprod-am02.afdgw.com] [afdg...	TCP 443	192.168.76.131 (Windows)	TCP 51653	TlsCertificate	2018-11-29 05:57:38 U	
929	msn.com.cer	cer	1 734 B	13.107.21.200 [a-0001.a-msedge.net] [www-msn.com] [a-0...	TCP 443	192.168.76.131 (Windows)	TCP 51674	TlsCertificate	2018-11-29 05:58:13 U	
2117	msn.com[1].cer	cer	1 734 B	204.79.197.203 [a-0003.a-msedge.net] [www-msn.com] [a-0...	TCP 443	192.168.76.131 (Windows)	TCP 51707	TlsCertificate	2018-11-29 05:58:50 U	
341	nexus.officeapps.live.com.cer	cer	1 892 B	52.103.120.21 [prod-w.nexus.live.com.akadns.net] [nexus...	TCP 443	192.168.76.131 (Windows)	TCP 51656	TlsCertificate	2018-11-29 05:57:39 U	
681	nexusx.officeapps.live.c.cer	cer	1 859 B	52.103.76.32 [prod-nexusx.live.com.akadns.net] [nexus...	TCP 443	192.168.76.131 (Windows)	TCP 51665	TlsCertificate	2018-11-29 05:57:48 U	
2177	pinnaclesl.cab	cab	7 796 B	8.253.224.254 [au.download.windowsupdate.com.cofoot...]	TCP 80	192.168.76.131 (Windows)	TCP 51708	HttpGetNormal	2018-11-29 05:58:54 U	
1693	settings.data.microsoft.com.cer	cer	2 288 B	52.175.39.99 [pk2eap.settings.data.microsoft.com.akad...	TCP 443	192.168.76.131 (Windows)	TCP 51695	TlsCertificate	2018-11-29 05:58:29 U	
781	sle.update.microsoft.com.cer	cer	1 695 B	13.78.168.230 [cls.update.microsoft.com.msact.net] [cls.up...	TCP 443	192.168.76.131 (Windows)	TCP 51670	TlsCertificate	2018-11-29 05:57:58 U	
1400	udc.msn.com.cer	cer	1 823 B	23.99.125.55 [ce0ff-prod-fe-eastasia.cloudapp.net] [ce0ff...	TCP 443	192.168.76.131 (Windows)	TCP 51690	TlsCertificate	2018-11-29 05:58:13 U	
1570	vo.msconfig.net.cer	cer	4 235 B	117.18.232.200 [ca9.wpc.vfdcn.net] [je3comview.vo.mse...	TCP 443	192.168.76.131 (Windows)	TCP 51691	TlsCertificate	2018-11-29 05:58:24 U	
1866	Web_2018-11-29_11-28-13.html	html	621 B	192.168.76.131 (Windows)	TCP 5170	140.82.59.185 (Windows)	TCP 5142	FTP	2018-11-29 05:58:38 U	
41	wms.windows.com.cer	cer	1 720 B	52.230.85.180 [sg2p.wms.notify.windows.com.akadns.net] ...	TCP 443	192.168.76.131 (Windows)	TCP 51649	TlsCertificate	2018-11-29 05:57:33 U	
881	wms.windows.com.cer	cer	1 720 B	52.230.3.194 [sg2p.wms.notify.windows.com.akadns.net] [...]	TCP 443	192.168.76.131 (Windows)	TCP 51672	TlsCertificate	2018-11-29 05:58:09 U	
93	wms.windows.com[1].cer	cer	1 720 B	52.230.85.180 [sg2p.wms.notify.windows.com.akadns.net] ...	TCP 443	192.168.76.131 (Windows)	TCP 51650	TlsCertificate	2018-11-29 05:57:33 U	
1072	www.bing.com.cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51676	TlsCertificate	2018-11-29 05:58:13 U	
1077	www.bing.com[1].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51678	TlsCertificate	2018-11-29 05:58:13 U	
1236	www.bing.com[10].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51684	TlsCertificate	2018-11-29 05:58:13 U	
1084	www.bing.com[2].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51679	TlsCertificate	2018-11-29 05:58:13 U	
1090	www.bing.com[3].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51681	TlsCertificate	2018-11-29 05:58:13 U	
1101	www.bing.com[4].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51680	TlsCertificate	2018-11-29 05:58:13 U	
1107	www.bing.com[5].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51683	TlsCertificate	2018-11-29 05:58:13 U	
1112	www.bing.com[6].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51685	TlsCertificate	2018-11-29 05:58:13 U	
1133	www.bing.com[7].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51682	TlsCertificate	2018-11-29 05:58:13 U	
1155	www.bing.com[8].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51687	TlsCertificate	2018-11-29 05:58:13 U	
1226	www.bing.com[9].cer	cer	3 078 B	13.107.21.200 [a-0001.a-msedge.net] [a-0001.a-fdentity.n...	TCP 443	192.168.76.131 (Windows)	TCP 51688	TlsCertificate	2018-11-29 05:58:13 U	

### Files tab

We can see plenty of files. Let's open the files that we found using the STOR command in the browser, as shown in the following screenshot:



The attacker was not only keylogging, but was also fetching details such as the active window title along with the key logs. So, to sum this up, we have the following answers to the questions that we asked at the beginning of the exercise:

- **Find the infected system:** 192.168.76.131
- **Trace the data to the server:** 140.82.59.185
- **Find the frequency of the data that is being sent:** The difference between two consecutive STOR commands for a similar file type is 15 seconds
- **Find what other information is carried alongside the keystrokes:** Active window titles
- **Try to uncover the attacker:** Not yet found
- **Extract and reconstruct the files sent to the attacker:**  
Keys\_2018-11-28\_16-04-42.html

We have plenty of information regarding the hacker. At this point, we can provide the details we found in our analysis in the report, or we can go one step further and try to uncover the identity of the attacker. If you chose to do so, then let's get started in finding out how to uncover this information.



Logging into a computer that you're not authorized to access can result in criminal penalties (fines, imprisonment, or both).

We already found their credentials in the server. Let's try logging into the FTP server and try to find something of interest, as shown in the following screenshot:

```
root@kali:~/fbctf# nc 140.82.59.185 21
220 (vsFTPd 3.0.3)
help
530 Please login with USER and PASS.
USER test_user
331 Please specify the password.
PASS Nipun@123
230 Login successful.
```

We can see that we are easily able to log into the server. Let's use an FTP client, such as Royal TSX in Mac (FileZilla for Windows), to view the files that reside on the server, as shown in the following screenshot:

The screenshot shows a file transfer interface with tabs for 'Overview' and 'File Transfer'. The 'File Transfer' tab is active, showing a directory structure under '/ > Test'. The contents of the 'John' directory are listed in a table:

	Name	Size	Date
..	..		
7	Jo	4 KB	27-Nov-2018 at 8:02:...
ap	John	4 KB	20-Dec-2018 at 8:18:...
D:	App_2018-11-28_01-40-34.html	1 KB	27-Nov-2018 at 8:22:...
fa	App_2018-11-28_14-48-20.html	647 bytes	28-Nov-2018 at 9:18:...

Wow! So much information has been logged; however, we can see two directories named John and Jo. The directory Jo is empty but we may have something in the directory named John.

Let's view the contents of John, as shown in the following screenshot:

The screenshot shows the contents of the 'John' directory under '/ > Test > John'. The table lists one file:

	Name
	John_Langley_Resume_Updated.pdf

It looks as though the attacker is applying for jobs and keeps their updated resume on their server. The case-study analysis proves that the keylogger is a newbie. In answering the last question regarding the identity of the attacker, we have successfully conducted our first network forensic analysis exercise. The resume we found might have been stolen from someone else as well. However, this is just the tip of the iceberg. In the upcoming chapters, we will look at a variety of complex scenarios; this was an easy one.

In the next example, we will look at TCP packets and try figuring out what were the event causing such network traffic.

## Exercise 2 – two too many

Let's analyze the given capture file that we currently don't know any details about and try reconstructing the chain of events.

We will open the PCAP in Wireshark, as follows:

9	0.001913	172.16.0.8	64.13.134.52	TCP	58	36050 53	36050 → 53 [SYN] Seq=0 Win=3072 Len=0 MSS=1460
10	0.001965	172.16.0.8	64.13.134.52	TCP	58	36050 5900	36050 → 5900 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
11	0.0063797	64.13.134.52	172.16.0.8	TCP	60	53 36050	53 → 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1380
12	0.005271	172.16.0.8	64.13.134.52	TCP	58	36050 21	36050 → 21 [SYN] Seq=0 Win=4096 Len=0 MSS=1460
13	0.005341	172.16.0.8	64.13.134.52	TCP	58	36050 113	36050 → 113 [SYN] Seq=0 Win=4096 Len=0 MSS=1460
14	0.126832	64.13.134.52	172.16.0.8	TCP	60	113 36050	113 → 36050 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	0.129000	172.16.0.8	64.13.134.52	TCP	58	36050 80	36050 → 80 [SYN] Seq=0 Win=3072 Len=0 MSS=1460
16	0.129075	172.16.0.8	64.13.134.52	TCP	58	36050 139	36050 → 139 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
17	0.189975	64.13.134.52	172.16.0.8	TCP	60	80 36050	80 → 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1380
18	0.191518	172.16.0.8	64.13.134.52	TCP	58	36050 3389	36050 → 3389 [SYN] Seq=0 Win=3072 Len=0 MSS=1460

From the preceding screenshot, we can see that numerous SYN packets are being sent out to the 64.13.134.52 IP address. However, looking closely, we can see that most of the packets are being sent every so often from a single port, which is 36050 and 36051, to almost every port on 64.13.134.52. Yes, you guessed right: this looks like a port scan. Initially the SYN packet is sent out, and on receiving a SYN/ACK, the port is considered open.

We know that the originating IP address, 172.16.0.8, is an internal one and the server being contracted is 64.13.134.52. Can you figure out the following?

- Scan type
- Open ports

Answering the first question requires a more in-depth understanding of a TCP-oriented communication and its establishment, TCP works on a three-way handshake, which means that on receiving a **synchronize (SYN)** packet from the source IP address, the destination IP address sends out a **synchronize/ acknowledgment (SYN/ACK)** packet that is followed by a final **acknowledgment (ACK)** packet from the source IP address to complete the three-way handshake. However, as we can see from the preceding screenshot, only a SYN/ACK is sent back from port 80, and there hasn't been an ACK packet sent out by the source IP address.

This phenomenon means that the ACK packet was never sent to the destination by the source, which means that only the first two steps of the three-way handshake were completed. This two step half open mechanism causes the destination to use up resources as the port will be help open for a period of time. Meanwhile, this is a popular technique leveraged by a scan type called **SYN scan** or **half-open scan**, or sometimes the **stealth scan**. Tools such as Nmap make use of such techniques to lower the number of network packets on the wire. Therefore, we can conclude that the type of scan we are dealing with is a SYN scan.



Nmap uses RST packet in half open scan periodically to prevent resource exhaustion at the destination.

No.	A	Time	Source	Destination	Protocol	Length	Source Port	Port	New Column
11	0.063797		64.13.134.52	172.16.0.8	TCP	60	53	36050	53 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
14	0.126832		64.13.134.52	172.16.0.8	TCP	60	113	36050	113 - 36050 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	0.180975		64.13.134.52	172.16.0.8	TCP	60	80	36050	80 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
46	1.465651		64.13.134.52	172.16.0.8	TCP	60	22	36050	22 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
47	1.465099		64.13.134.52	172.16.0.8	TCP	60	25	36050	25 - 36050 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
118	1.818597		64.13.134.52	172.16.0.8	TCP	60	31337	36050	31337 - 36050 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
529	3.063375		64.13.134.52	172.16.0.8	TCP	60	53	36050	[TCP Retransmission] 53 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
571	3.132131		64.13.134.52	172.16.0.8	TCP	60	113	36061	113 - 36061 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
632	3.187263		64.13.134.52	172.16.0.8	TCP	60	80	36050	[TCP Retransmission] 80 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
1233	4.077986		64.13.134.52	172.16.0.8	TCP	60	79	36050	79 - 36050 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1963	5.063418		64.13.134.52	172.16.0.8	TCP	60	23	36050	[TCP Retransmission] 23 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
2006	9.071690		64.13.134.52	172.16.0.8	TCP	60	53	36050	[TCP Retransmission] 53 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
2007	9.387931		64.13.134.52	172.16.0.8	TCP	60	80	36050	[TCP Retransmission] 80 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
2008	11.064190		64.13.134.52	172.16.0.8	TCP	60	22	36050	[TCP Retransmission] 22 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
2009	21.093215		64.13.134.52	172.16.0.8	TCP	60	53	36065	[TCP Retransmission] 53 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
2010	21.461180		64.13.134.52	172.16.0.8	TCP	60	80	36050	[TCP Retransmission] 80 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388
2011	23.085343		64.13.134.52	172.16.0.8	TCP	60	22	36050	[TCP Retransmission] 22 - 36050 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1388

Applying the filer `ip.src==64.13.134.5`, we can see the responses sent by 64.13.134.52. It is evident that we have received the SYN/ACK from ports 53, 80, and 22, which are open ports. We can also see that there has been network loss, and the sender has sent the packets again. Additionally, we can see **Reset Acknowledgment Packets (RST)** that denote misconfigurations or the application running on the not willing to connect: the reasons for such behavior can differ.

## Summary

Over the course of this chapter, we learned about the basics of network forensics. We used Wireshark to analyze a keylogger and packets from a port scan. We discovered various types of network evidence sources and also learned the basics methodology that we should follow when performing network forensics.

In the next chapter, we will look at the basics of protocols and other technical concepts and strategies that are used to acquire evidence, and we will perform hands-on exercises related to them.



All credits for this above capture file goes to Chris Sanders GitHub repository at <https://github.com/chrissanders/packets>.

## Questions and exercises

To improve your confidence in your network forensics skills, try answering the following questions:

1. What is the difference between the `ftp` and `ftp-data` display filter in Wireshark?
2. Can you build an `http` filter for webpages with specific keywords?
3. We saved files from the PCAP using NetworkMiner. Can you do this using Wireshark? (Yes/No)
4. Try repeating these exercises with Tshark.

## Further reading

Read more and watch videos on Wireshark