



INSTITUTO SUPERIOR TÉCNICO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

FORENSICS CYBER-SECURITY

MEIC, METI

Wireshark Tutorial

2023/2024

afonso.gomes@tecnico.ulisboa.pt

Introduction

This guide aims to provide some examples of techniques that you are expected to use when analyzing traffic in Wireshark. For the third assignment of CSF, it is extremely important that you understand and are comfortable with the basic filtering capabilities of Wireshark, so that you can purposefully search for the information you want to find.

It is also very important for you to be able to identify what details might be suspicious in certain protocols, not only so that you can notice them when you see them, but also so that you can purposefully search for them.

1 Setting up the SSL Key log file

To be able to inspect the contents of the SSL/TLS packets in our capture, you must first set up the SSL key log file that was generated during the capture. To do this, you may follow the following steps:

1. Open Wireshark.
2. On top of the window, go to **Edit**, followed by **Preferences**.
3. Expand **Protocols** and search for **TLS** in the list of protocols.
4. Finally, find the field **(Pre)-Master-Secret log filename** and type there the path to the SSL key log file.

2 Time Shift

Wireshark provides users with a tool that may be useful to convert the times shown in the whole capture to different timezones. This tool is called **Time Shift**, and it can be accessed by going to the top of the window to **Edit**, followed by **Time Shift...**

Figure 1 shows the Time Shift window and its options. In this specific example, we are shifting all packets 14 days back in time.

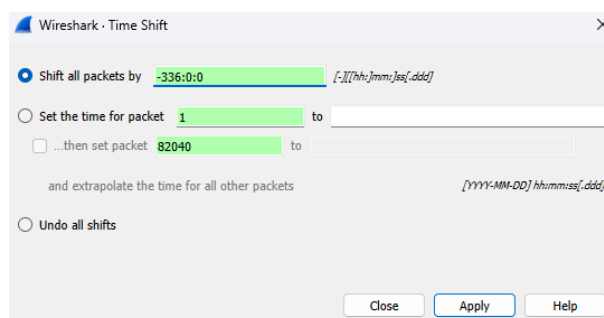


Figure 1: Time Shift Window

After using the Time Shift tool, if you ever want to revert and recover the original times, you may simply go back to the Time Shift window and select **Undo all shifts**.

3 Name Resolution

During your analysis, it might sometimes be useful to search for a specific domain name, such as `www.google.com` instead of its corresponding address, which is usually not recognizable. For this purpose, Wireshark provides options for name resolution. To access these options, you simply have to go to the top of the window to **View**, followed by **Name Resolution**, as shown in Figure 2.

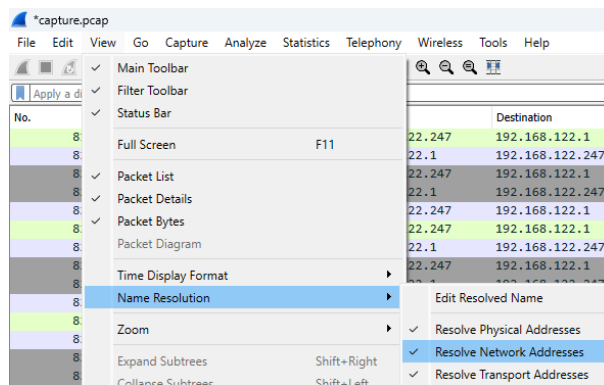


Figure 2: How to access the Name Resolution options

Here you have three options:

- **Resolve Physical Addresses:** Wireshark will attempt to translate MAC addresses to their respective names, for example, by identifying the name of the vendor to which the Organizationally Unique Identifier (OUI) belongs to.
- **Resolve Network Addresses:** Wireshark will use DNS servers to attempt to convert IP addresses to the hostnames associated with them.
- **Resolve Transport Addresses:** Wireshark will convert TCP and UDP ports to their well-known names, if they exist. For example, port 80 will be identified as HTTP.

Furthermore, while performing your analysis, you may have trouble keeping track of who each address belongs to. To help with this, you may right-click on an address you want to give a name to, and select **Edit Resolved Name**. After doing so, a text box will appear at the top of the packet list where you can type your desired name and press **Ok** to apply it. From then on, all instances of that IP address in the packet list will be replaced with your chosen name, making it easier for you to identify the entities involved in the various communications.

4 Find Tool

The find tool is present in various applications and Wireshark is no different. As is common, you can access this tool by pressing the **CTRL + F** key combination, or by going to **Edit -> Find Packet...**

However, before you can start accurately searching for what you desire, you must select where your search will be performed. As you may have noticed, Wireshark is divided into three **Panes**, the Packet List, the Packet Details, and the Packet Bytes. Figure 3 shows each of these panes, together with their names.

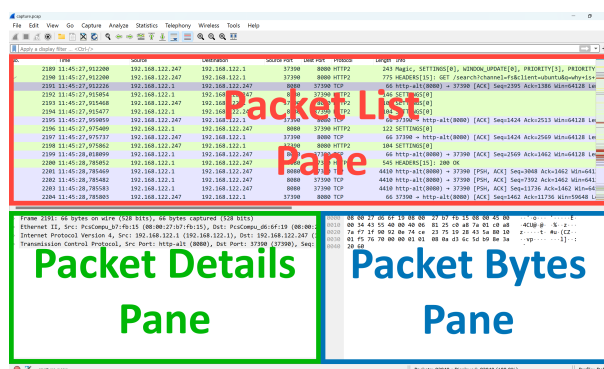


Figure 3: Wireshark panes

When performing a search using the find tool, you must specify in which of the three panes the search should be made. Additionally, you must also specify what type of data you would like to search for: Hex Value, String or Regular Expression. If you are searching for a string, you may also specify whether the search should be case-sensitive or not, and what character encoding(s) to use in the search.

5 Export Packet Bytes

One very useful thing that Wireshark allows you to do is export specific parts of a packet's data to a file. Some specific cases in which this can be helpful are to export plaintext data, HTML data, JSON data, etc.

You can do this by simply right-clicking on what you want to extract in the **Packet details Pane**, followed by **Export Packet Bytes...**

Figure 4 shows how you can export, for example, JSON data from an HTTP request.

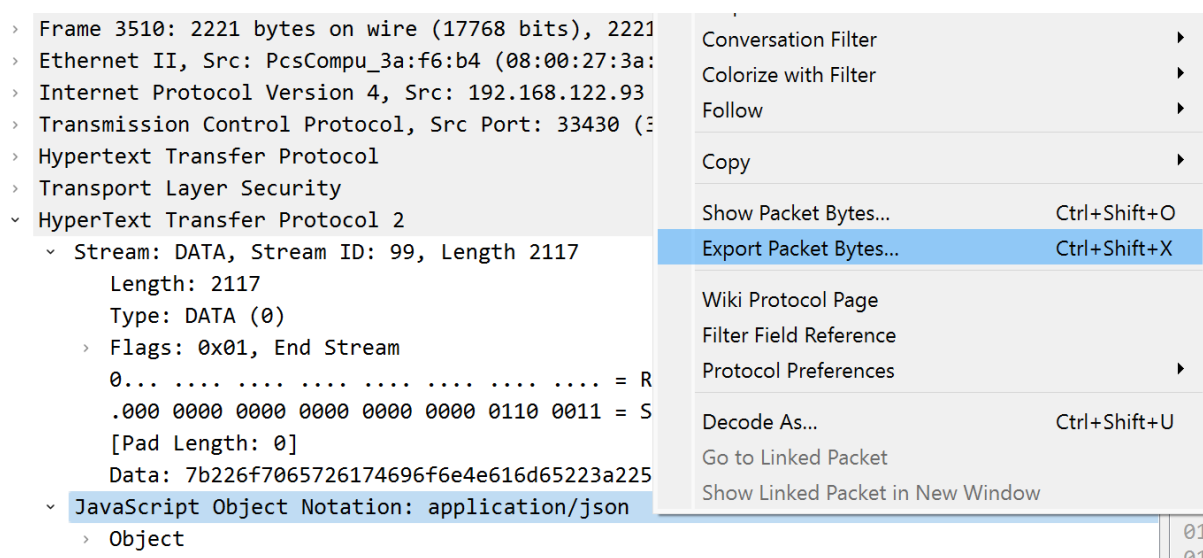


Figure 4: Exporting JSON data from an HTTP response

6 Follow Stream

In some cases, such as with HTTP connections, it might be useful to see the contents of a whole “conversation” that spans multiple packets. To do this, you may use the "Follow _ Stream" tool, which allows you to see the whole conversation to which a packet belongs. There are multiple options of streams to follow: TCP, UDP, TLS, HTTP, HTTP/2, etc.

Figure 5 shows how to access this tool for a certain packet.

No.	Time	Source	Destination	Source Port	Dest Port	Protocol	Length	Info
28196	11:52:10,359370	192.168.122.113	192.168.122.1	34276	8080	HTTP	319	POST /mail/client/start;jsessionid=59D8114D09144F59026DE0A91D7DFE82-n1.lxa04b?uc=SUCCESS&navigator_theme=mailcomblue&navigator_bg=mailcomblue HTTP/1.1
28197	11:52:10,359376	192.168.122.1	192.168.122.113	8080	34276	TCP	66	http->
28198	11:52:10,359385	192.168.122.1	192.168.122.113	8080	34276	TCP	66	http->
28270	11:52:10,885733	192.168.122.1	192.168.122.113	8080	34276	TLSv1.2	870	[TLS ...
28271	11:52:10,886055	192.168.122.1	192.168.122.1	34276	8080	TCP	66	34276
28272	11:52:10,886067	192.168.122.1	192.168.122.113	8080	34276	HTTP/XML	295	HTTP/...
28273	11:52:10,886341	192.168.122.1	192.168.122.1	34276	8080	TCP	66	34276
28303	11:52:11,413960	192.168.122.113	192.168.122.1	34276	8080	HTTP	5464	GET /r...
28304	11:52:11,413996	192.168.122.1	192.168.122.113	8080	34276	TCP	66	http->
28328	11:52:11,708662	192.168.122.1	192.168.122.113	8080	34276	HTTP	716	HTTP/...
28329	11:52:11,709855	192.168.122.113	192.168.122.1	34276	8080	TCP	66	34276
28330	11:52:11,713478	192.168.122.113	192.168.122.1	34276	8080	TCP	4410	34276
28331	11:52:11,713495	192.168.122.1	192.168.122.113	8080	34276	TCP	66	http->
28332	11:52:11,713565	192.168.122.113	192.168.122.1	34276	8080	HTTP	1122	GET /r...
28333	11:52:11,713571	192.168.122.1	192.168.122.113	8080		TCP Stream		Ctrl+Alt+Shift+T
28383	11:52:12,347252	192.168.122.1	192.168.122.113	8080		UDP Stream		Ctrl+Alt+Shift+U
28384	11:52:12,347387	192.168.122.1	192.168.122.113	8080		DCCP Stream		Ctrl+Alt+Shift+E
28385	11:52:12,347397	192.168.122.1	192.168.122.113	8080		TLS Stream		Ctrl+Alt+Shift+S
28386	11:52:12,347727	192.168.122.113	192.168.122.1	34276		HTTP Stream		Ctrl+Alt+Shift+H
28387	11:52:12,347740	192.168.122.1	192.168.122.113	8080		HTTP/2 Stream		
28391	11:52:12,389851	192.168.122.113	192.168.122.1	34276		HTTP/2 Stream		

Figure 5: Following an HTTP Stream

A simple example is an HTTP conversation, which allows you to see the sequence of requests and their respective responses. Figure 6 shows an example of the window that appears when you follow a stream.

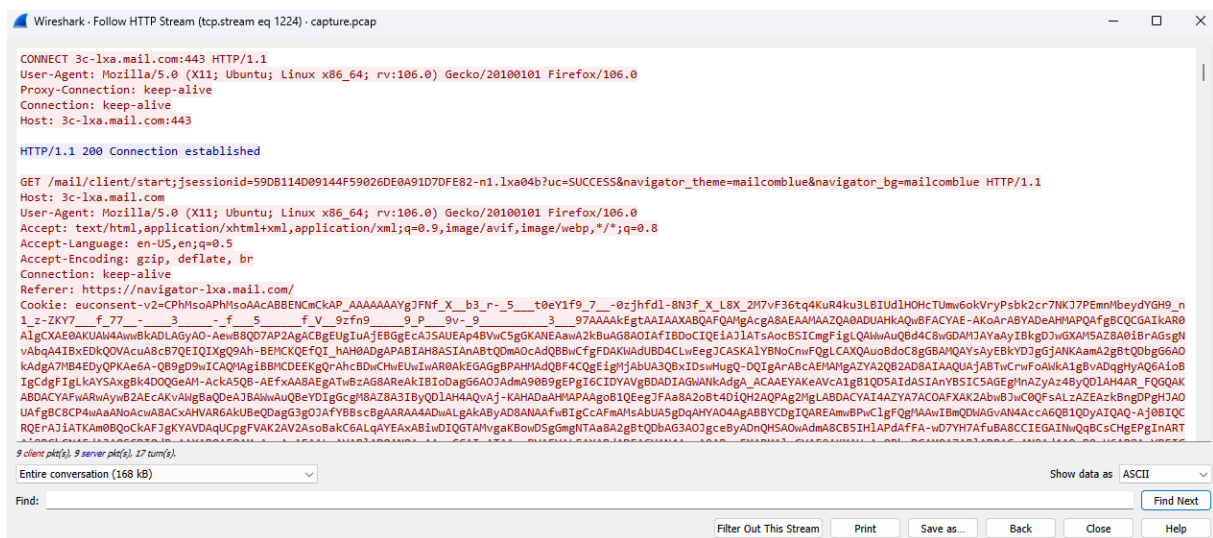


Figure 6: Example of an HTTP conversation seen when following an HTTP stream

On the bottom of the follow stream window, you are provided with some useful options, such as the direction specifier, which allows you to specify whether to show the whole conversation or just the packets going in one direction, and the data presentation mode, which allows you to specify how the data should be shown, for example, as **Raw** data, as an **Hex Dump**, decoded as **ASCII**, etc.

Finally, you can also export the currently shown contents of this window to a file by using the **Save as...** button. Pay attention to the fact that the data is exported based on your current presentation mode, so, for example, if you have selected the data to be shown as a Hex Dump, the exported file will be formatted as a Hex Dump.

7 Export Objects

Another way to export files found in a capture is the Export Objects Tool. To use it, you simply have to go to the top of the window to **File**, followed by **Export Objects** and choose the protocol for which you want to search for and extract objects. Figure 7 shows an example of a list of HTTP objects found by Wireshark through this tool.

Wireshark · Export · HTTP object list

Text Filter: Content Type: All Content-Types

Packet	Hostname	Content Type	Size	Filename
53842	home.navigato-lxa.mail.com	text/html	81 kB	X2NvbW1vbi9jb250ZW50LnR3aWc_eq_?sid=23d03f2072
53901	3c-lxa.mail.com	text/html	71 kB	folderjsessionid=F1887FC62C16D7218D4D027AD520698
53996	home.navigato-lxa.mail.com	application/json	1915 bytes	mailsf?sid=23d03f2072fda68259c621798f47ae928207644
54270	home.navigato-lxa.mail.com	application/json	870 bytes	restfs?sid=23d03f2072fda68259c621798f47ae928207644
54341	3c-lxa.mail.com	text/xml	685 bytes	folderjsessionid=F1887FC62C16D7218D4D027AD520698
54364	ib.adnxs.com	text/plain	7378 bytes	prebid
54430	ad.yieldlab.net	text/html	8520 bytes	2x2?ts=2419621930&type=h&gdpr=1&consent=CPhMsoAPhMsoAAcABB
54610	t.uimserv.net	image/gif	42 bytes	mam_optin_p
54634	3c-lxa.mail.com	application/x-www-form-urlencoded	269 bytes	folderjsessionid=F1887FC62C16D7218D4D027AD520698&n1.lxa12b72-1.l
54774	x.bidswitch.net	image/gif	42 bytes	sync?ssp=yieldlab
54846	t.uimserv.net	image/gif	43 bytes	folder&att1=700&att2=773&att3=]&d=78300107&lpos=1
54853	ib.adnxs.com	image/gif	43 bytes	setuid?bidder=yieldlab&uid=e41725b1-5883-40ca-9cca-65166fab53f0
54861	3c-lxa.mail.com	text/xml	69 bytes	folderjsessionid=F1887FC62C16D7218D4D027AD520698&n1.lxa12b72-1.l
55031	ib.adnxs.com	text/plain	7248 bytes	prebid
55038	ad.yieldlab.net	text/html	8520 bytes	2x2?ts=9069571525&type=h&gdpr=1&consent=CPhMsoAPhMsoAAcABB

Save Save All Preview Close Help

Figure 7: Example of a list of HTTP objects

From this list, you can either export just specific objects or export all objects shown. Additionally, you can also filter only certain types of objects and export those.

8 Statistics

Wireshark offers various statistical views that aid in the detection of unusual patterns within network traffic. These views analyze packet flow rates, protocol distribution, and conversation patterns, providing insights on network behavior and potential issues in the captured data.

8.1 Number of packets per unit time

You can create a graph that plots the number of packets per unit of time by clicking on **Statistics > I/O Graphs**. You can then identify the periods with highest activity in terms of number of packets sent/received, and click directly on the peaks to get a closer look at what is causing that unusual behavior.

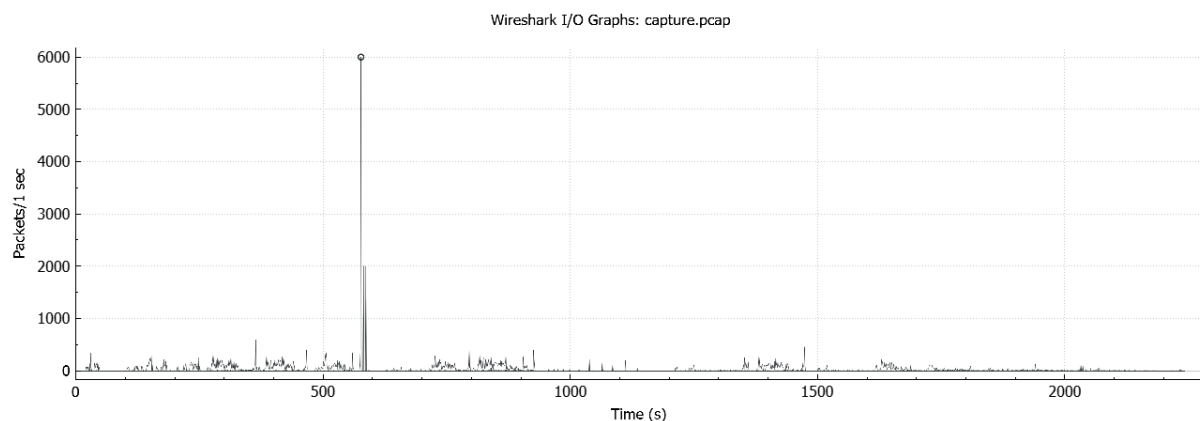


Figure 8: Example of an I/O Graph with a suspicious peak

8.2 Statistics on endpoints

Wireshark also provides a range of statistics on the amount of data (packets, bytes) sent/received by the different endpoints present on the capture. You can access this by clicking on **Statistics -> Endpoints**. This allows you to get a better idea of which hosts are more active in the capture (Figure 9), and which ports were used the most to transfer the largest amount of data (Figure 10). This can provide useful information if there are unusual ports gathering large volumes of data. Knowing this information can help narrow the search by applying the specific host and port as filters.

Ethernet · 8	IPv4 · 14	IPv6 · 5	TCP · 5307	UDP · 93
Address	Packets	Bytes	Tx Packets	
192.168.122.1	68.056 KiB	28.593 MiB	32.936 K	
192.168.122.93	41.553 KiB	22.581 MiB	21.343 K	
192.168.122.113	37.840 KiB	17.143 MiB	19.334 K	
192.168.122.247	8.661 KiB	1.930 MiB	4.405 K	
34.120.208.123	1.718 KiB	726.042 KiB	945 bytes	
8.8.8.8	869 bytes	78.454 KiB	420 bytes	
35.232.111.17	128 bytes	10.057 KiB	60 bytes	
35.224.170.84	111 bytes	8.396 KiB	50 bytes	
34.160.144.191	39 bytes	8.087 KiB	20 bytes	
34.122.121.32	68 bytes	5.324 KiB	30 bytes	
91.189.94.4	8 bytes	720 bytes	4 bytes	
224.0.0.251	2 bytes	320 bytes	0 bytes	
185.125.190.56	2 bytes	180 bytes	1 bytes	
10.0.2.2	2 bytes	140 bytes	2 bytes	

Apply as Filter

Prepare as Filter

Find

Colorize

Copy Endpoint table

Resize all columns to content

Selected

Not Selected

...and Selected

...or Selected

...and Not Selected

...or Not Selected

Figure 9: Example of statistics per host

Ethernet · 8		IPv4 · 14		IPv6 · 5		TCP · 5307		UDP · 93	
Address	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes		
192.168.122.1	8080	65,984 KiB	28.473 MiB	31,900 KiB	20.149 MiB	34,084 KiB	8.324 MiB		
192.168.122....	1337	824 bytes	5.668 MiB	368 bytes	26.891 KiB	456 bytes	5.642 MiB		
192.168.122.93	57910	214 bytes	2.314 MiB	119 bytes	2.308 MiB	95 bytes	6.221 KiB		
192.168.122.93	47154	293 bytes	2.130 MiB	118 bytes	9.151 KiB	175 bytes	2.122 MiB		
192.168.122.93	51046	176 bytes	1.691 MiB	93 bytes	1.686 MiB	83 bytes	5.447 KiB		
192.168.122.93	54084	130 bytes	974.871 KiB	81 bytes	971.525 KiB	49 bytes	3.346 KiB		
192.168.122....	43470	525 bytes	798.188 KiB	256 bytes	40.412 KiB	269 bytes	757.776 KiB		
34.120.208.123	443	1.718 KiB	726.042 KiB	945 bytes	266.104 KiB	814 bytes	459.938 KiB		
192.168.122.93	53060	81 bytes	560.253 KiB	43 bytes	566.706 KiB	38 bytes	2.547 KiB		

Figure 10: Example of statistics per host and port

9 Filter Examples

Finally, the most versatile tool and the main method for finding information in a network capture is the **Display Filters**. To help you understand what types of things you may be able to find using this tool, this section provides you with some examples of filters and their resulting packet lists, as well as some questions related to those results.

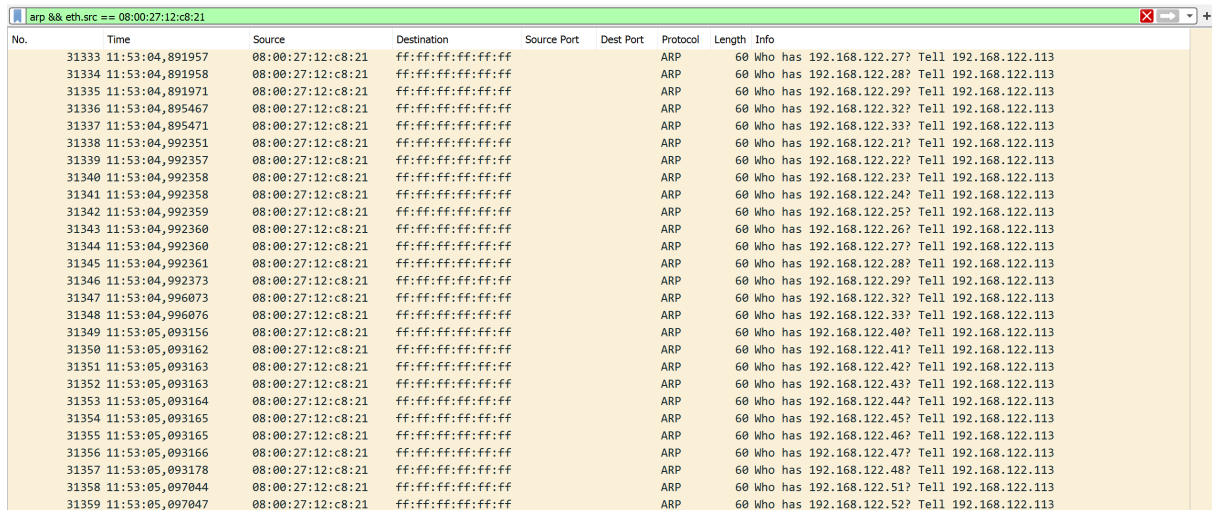
Most of these examples focus on a user whose computer has the MAC address `08:00:27:12:c8:21` and the IP address `192.168.122.113`.

Try to answer the questions related to each example to the best of your ability so that you can understand the purpose of each filter. Remember to use Google to search for any concept you may not fully understand.

9.1 ARP Example

Figure 11 shows the result of using the following filter(s):

- `arp`, which shows only packets corresponding to the Address Resolution Protocol (ARP).
- `eth.src == 08:00:27:12:c8:21` which shows only packets where the source MAC address is the specified one.



No.	Time	Source	Destination	Source Port	Dest Port	Protocol	Length	Info
31333	11:53:04,891957	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.27? Tell 192.168.122.113
31334	11:53:04,891958	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.28? Tell 192.168.122.113
31335	11:53:04,891971	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.29? Tell 192.168.122.113
31336	11:53:04,895467	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.32? Tell 192.168.122.113
31337	11:53:04,895471	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.33? Tell 192.168.122.113
31338	11:53:04,992351	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.21? Tell 192.168.122.113
31339	11:53:04,992357	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.22? Tell 192.168.122.113
31340	11:53:04,992358	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.23? Tell 192.168.122.113
31341	11:53:04,992358	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.24? Tell 192.168.122.113
31342	11:53:04,992359	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.25? Tell 192.168.122.113
31343	11:53:04,992360	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.26? Tell 192.168.122.113
31344	11:53:04,992360	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.27? Tell 192.168.122.113
31345	11:53:04,992361	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.28? Tell 192.168.122.113
31346	11:53:04,992373	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.29? Tell 192.168.122.113
31347	11:53:04,996073	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.32? Tell 192.168.122.113
31348	11:53:04,996076	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.33? Tell 192.168.122.113
31349	11:53:05,003156	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.40? Tell 192.168.122.113
31350	11:53:05,003162	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.41? Tell 192.168.122.113
31351	11:53:05,003163	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.42? Tell 192.168.122.113
31352	11:53:05,003163	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.43? Tell 192.168.122.113
31353	11:53:05,003164	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.44? Tell 192.168.122.113
31354	11:53:05,003165	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.45? Tell 192.168.122.113
31355	11:53:05,003165	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.46? Tell 192.168.122.113
31356	11:53:05,003166	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.47? Tell 192.168.122.113
31357	11:53:05,003178	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.48? Tell 192.168.122.113
31358	11:53:05,007044	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.51? Tell 192.168.122.113
31359	11:53:05,007047	08:00:27:12:c8:21	ff:ff:ff:ff:ff:ff			ARP	60	Who has 192.168.122.52? Tell 192.168.122.113

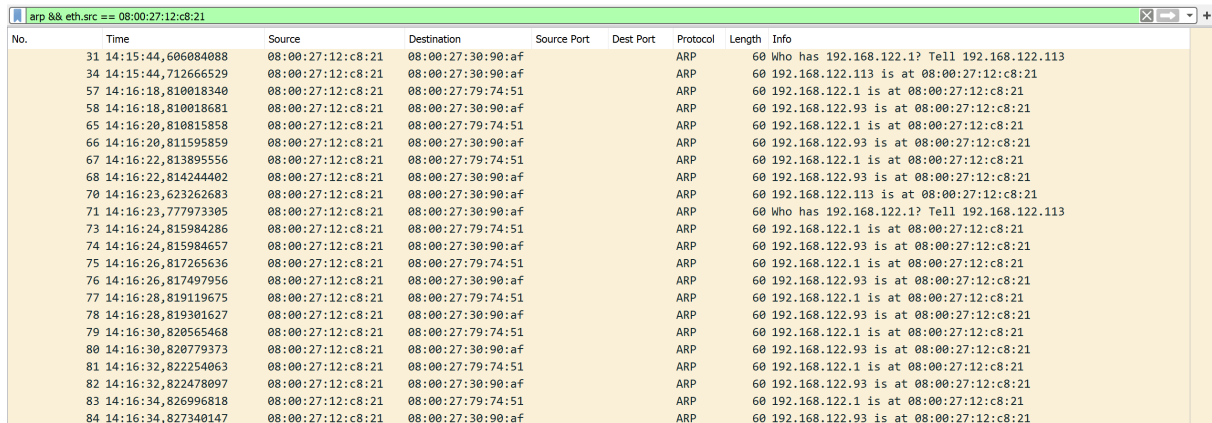
Figure 11: ARP Example

1. Is the resulting list of packets normal, or do you find anything suspicious about it?
2. What kinds of user actions may such a filter be used to identify?

9.2 Another ARP Example

Figure 12 shows the result of using the same filters as the previous example, but in a different capture:

- `arp`, which shows only packets corresponding to the Address Resolution Protocol (ARP).
- `eth.src == 08:00:27:12:c8:21` which shows only packets where the source MAC address is the specified one.



No.	Time	Source	Destination	Source Port	Dest Port	Protocol	Length	Info
31	14:15:44,606084088	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	Who has 192.168.122.1? Tell 192.168.122.113
34	14:15:44,71266529	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.113 is at 08:00:27:12:c8:21
57	14:16:18,810018340	08:00:27:12:c8:21	08:00:27:79:74:51			ARP	60	192.168.122.1 is at 08:00:27:12:c8:21
58	14:16:18,810018681	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.93 is at 08:00:27:12:c8:21
65	14:16:20,810815858	08:00:27:12:c8:21	08:00:27:79:74:51			ARP	60	192.168.122.1 is at 08:00:27:12:c8:21
66	14:16:20,811595859	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.93 is at 08:00:27:12:c8:21
67	14:16:22,813895556	08:00:27:12:c8:21	08:00:27:79:74:51			ARP	60	192.168.122.1 is at 08:00:27:12:c8:21
68	14:16:22,814244402	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.93 is at 08:00:27:12:c8:21
70	14:16:23,623262683	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.113 is at 08:00:27:12:c8:21
71	14:16:23,777973305	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	Who has 192.168.122.1? Tell 192.168.122.113
73	14:16:24,815984286	08:00:27:12:c8:21	08:00:27:79:74:51			ARP	60	192.168.122.1 is at 08:00:27:12:c8:21
74	14:16:24,815984657	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.93 is at 08:00:27:12:c8:21
75	14:16:26,817265636	08:00:27:12:c8:21	08:00:27:79:74:51			ARP	60	192.168.122.1 is at 08:00:27:12:c8:21
76	14:16:26,817497956	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.93 is at 08:00:27:12:c8:21
77	14:16:28,819119675	08:00:27:12:c8:21	08:00:27:79:74:51			ARP	60	192.168.122.1 is at 08:00:27:12:c8:21
78	14:16:28,819301627	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.93 is at 08:00:27:12:c8:21
79	14:16:30,820565468	08:00:27:12:c8:21	08:00:27:79:74:51			ARP	60	192.168.122.1 is at 08:00:27:12:c8:21
80	14:16:30,820779373	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.93 is at 08:00:27:12:c8:21
81	14:16:32,822254063	08:00:27:12:c8:21	08:00:27:79:74:51			ARP	60	192.168.122.1 is at 08:00:27:12:c8:21
82	14:16:32,822478097	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.93 is at 08:00:27:12:c8:21
83	14:16:34,826996818	08:00:27:12:c8:21	08:00:27:79:74:51			ARP	60	192.168.122.1 is at 08:00:27:12:c8:21
84	14:16:34,827340147	08:00:27:12:c8:21	08:00:27:30:90:af			ARP	60	192.168.122.93 is at 08:00:27:12:c8:21

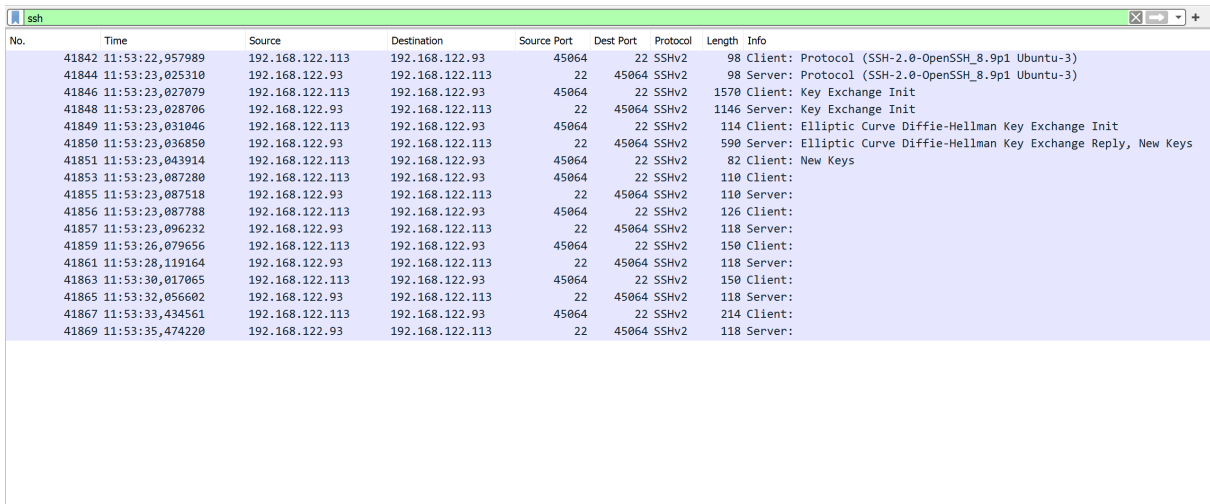
Figure 12: Another ARP Example

1. Is the resulting list of packets normal, or do you find anything suspicious about it?
2. Can you identify what type of attack may have been performed during this capture? If so, who performed it and what might be the objective of the attack?

9.3 SSH Example

Figure 13 shows the result of using the following filter(s):

- `ssh`, which shows only packets corresponding to the Secure Shell (SSH) protocol.



The image shows a Wireshark packet capture window with the filter 'ssh' applied. The packet list displays 18 packets, all of which are SSH-related. The packet details pane on the right shows the structure of the selected packet (No. 41842), including the SSH Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3), Key Exchange Init, and New Keys.

No.	Time	Source	Destination	Source Port	Dest Port	Protocol	Length	Info
41842	11:53:22,957989	192.168.122.113	192.168.122.93	45064	22	SSHv2	98	Client: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3)
41844	11:53:23,025310	192.168.122.93	192.168.122.113	22	45064	SSHv2	98	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3)
41846	11:53:23,027079	192.168.122.113	192.168.122.93	45064	22	SSHv2	1570	Client: Key Exchange Init
41848	11:53:23,028706	192.168.122.93	192.168.122.113	22	45064	SSHv2	1146	Server: Key Exchange Init
41849	11:53:23,031046	192.168.122.113	192.168.122.93	45064	22	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
41850	11:53:23,036850	192.168.122.93	192.168.122.113	22	45064	SSHv2	590	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
41851	11:53:23,043914	192.168.122.113	192.168.122.93	45064	22	SSHv2	82	Client: New Keys
41853	11:53:23,087280	192.168.122.113	192.168.122.93	45064	22	SSHv2	110	Client:
41855	11:53:23,087518	192.168.122.93	192.168.122.113	22	45064	SSHv2	110	Server:
41856	11:53:23,087788	192.168.122.113	192.168.122.93	45064	22	SSHv2	126	Client:
41857	11:53:23,096232	192.168.122.93	192.168.122.113	22	45064	SSHv2	118	Server:
41859	11:53:26,079656	192.168.122.113	192.168.122.93	45064	22	SSHv2	150	Client:
41861	11:53:28,119164	192.168.122.93	192.168.122.113	22	45064	SSHv2	118	Server:
41863	11:53:30,017065	192.168.122.113	192.168.122.93	45064	22	SSHv2	150	Client:
41865	11:53:32,056602	192.168.122.93	192.168.122.113	22	45064	SSHv2	118	Server:
41867	11:53:33,434561	192.168.122.113	192.168.122.93	45064	22	SSHv2	214	Client:
41869	11:53:35,474220	192.168.122.93	192.168.122.113	22	45064	SSHv2	118	Server:

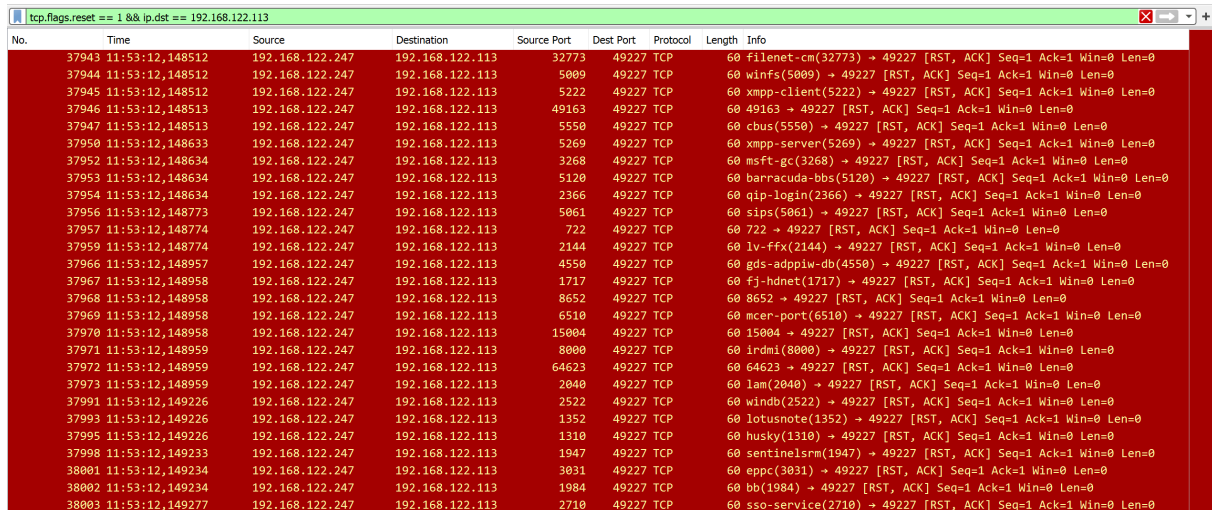
Figure 13: SSH Example

1. Is it usual to see SSH traffic in a capture? Could it have been generated automatically by a device?
2. Can you obtain any information just from this capture, even though it is encrypted?
3. Do you have any hypotheses related to what may have been done by the user? What method could you use to confirm your hypotheses?

9.4 TCP Reset Flag Example

Figure 14 shows the result of using the following filter(s):

- `tcp.flags.reset == 1`, which shows only packets with the TCP reset flag set.
- `ip.dst == 192.168.122.113` which shows only packets where the destination IP address is the specified one.



The image shows a Wireshark packet capture window with the filter `tcp.flags.reset == 1 && ip.dst == 192.168.122.113` applied. The packet list shows 30 packets, all of which are TCP RST (Reset) packets. Each packet has a source IP of 192.168.122.247 and a destination IP of 192.168.122.113. The source ports vary, and the destination ports also vary. The packet details pane on the right shows the structure of a TCP RST packet, including the reset sequence number and the reason for the reset (e.g., 'RST, ACK Seq=1 Ack=1 Win=0 Len=0').

No.	Time	Source	Destination	Source Port	Dest Port	Protocol	Length	Info
37943	11:53:12,148512	192.168.122.247	192.168.122.113	32773	49227	TCP	60	60 filenet-cm(32773) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37944	11:53:12,148512	192.168.122.247	192.168.122.113	5009	49227	TCP	60	60 winfs(5009) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37945	11:53:12,148512	192.168.122.247	192.168.122.113	5222	49227	TCP	60	60 xmpp-client(5222) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37946	11:53:12,148513	192.168.122.247	192.168.122.113	49163	49227	TCP	60	60 49163 → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37947	11:53:12,148513	192.168.122.247	192.168.122.113	5550	49227	TCP	60	60 cbus(5550) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37950	11:53:12,148633	192.168.122.247	192.168.122.113	5269	49227	TCP	60	60 xmpp-server(5269) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37952	11:53:12,148634	192.168.122.247	192.168.122.113	3268	49227	TCP	60	60 msft-gc(3268) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37953	11:53:12,148634	192.168.122.247	192.168.122.113	5120	49227	TCP	60	60 barracuda-bbs(5120) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37954	11:53:12,148634	192.168.122.247	192.168.122.113	2366	49227	TCP	60	60 qip-login(2366) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37956	11:53:12,148773	192.168.122.247	192.168.122.113	5061	49227	TCP	60	60 sips(5061) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37957	11:53:12,148774	192.168.122.247	192.168.122.113	722	49227	TCP	60	60 722 → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37959	11:53:12,148774	192.168.122.247	192.168.122.113	2144	49227	TCP	60	60 lv-ffx(2144) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37966	11:53:12,148957	192.168.122.247	192.168.122.113	4550	49227	TCP	60	60 gds-adppiw-db(4550) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37967	11:53:12,148958	192.168.122.247	192.168.122.113	1717	49227	TCP	60	60 fj-hdnet(1717) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37968	11:53:12,148958	192.168.122.247	192.168.122.113	8652	49227	TCP	60	60 8652 → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37969	11:53:12,148958	192.168.122.247	192.168.122.113	6510	49227	TCP	60	60 mcer-port(6510) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37970	11:53:12,148958	192.168.122.247	192.168.122.113	15004	49227	TCP	60	60 15004 → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37971	11:53:12,148959	192.168.122.247	192.168.122.113	8000	49227	TCP	60	60 irndai(8000) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37972	11:53:12,148959	192.168.122.247	192.168.122.113	64623	49227	TCP	60	60 64623 → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37973	11:53:12,148959	192.168.122.247	192.168.122.113	2040	49227	TCP	60	60 lam(2040) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37991	11:53:12,149226	192.168.122.247	192.168.122.113	2522	49227	TCP	60	60 windb(2522) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37993	11:53:12,149226	192.168.122.247	192.168.122.113	1352	49227	TCP	60	60 lotusnote(1352) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37995	11:53:12,149226	192.168.122.247	192.168.122.113	1310	49227	TCP	60	60 husky(1310) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
37998	11:53:12,149233	192.168.122.247	192.168.122.113	1947	49227	TCP	60	60 sentinelsrm(1947) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
38001	11:53:12,149234	192.168.122.247	192.168.122.113	3031	49227	TCP	60	60 eppc(3031) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
38002	11:53:12,149234	192.168.122.247	192.168.122.113	1984	49227	TCP	60	60 bb(1984) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
38003	11:53:12,149277	192.168.122.247	192.168.122.113	2710	49227	TCP	60	60 sso-service(2710) → 49227 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

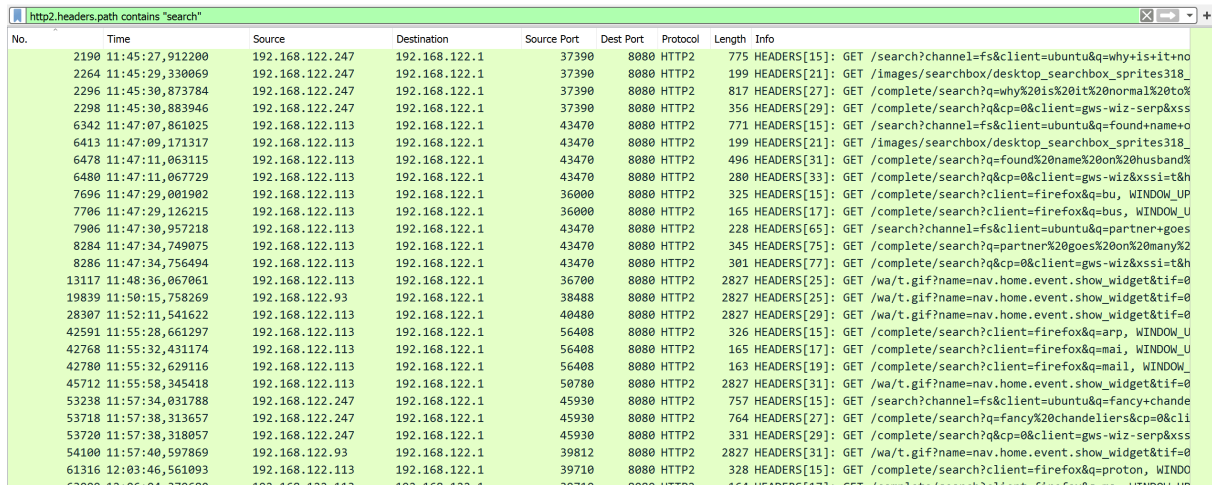
Figure 14: TCP Reset Example

1. What do you think is the objective of such a filter? Explain in simple terms what you think the filter is looking for.
2. Is the resulting packet list normal or do you find anything suspicious about it?

9.5 HTTP Path Example

Figure 15 shows the result of using the following filter(s):

- `http2.headers.path contains "search"`, which shows only packets whose HTTP/2 path contains the string “search”.



No.	Time	Source	Destination	Source Port	Dest Port	Protocol	Length	Info
2190	11:45:27,912200	192.168.122.247	192.168.122.1	37390	8080	HTTP2	775	HEADERS[15]: GET /search?channel=fs&client=ubuntu&q=why+is+it+no
2264	11:45:29,330069	192.168.122.247	192.168.122.1	37390	8080	HTTP2	199	HEADERS[21]: GET /images/searchbox/desktop_searchbox_sprites318_
2296	11:45:30,873784	192.168.122.247	192.168.122.1	37390	8080	HTTP2	817	HEADERS[27]: GET /complete/search?q=why%20is%20it%20normal%20to%20
2298	11:45:30,883946	192.168.122.247	192.168.122.1	37390	8080	HTTP2	356	HEADERS[29]: GET /complete/search?q=cp=0&client=gws-wiz-serp&xss
6342	11:47:07,861025	192.168.122.113	192.168.122.1	43470	8080	HTTP2	771	HEADERS[15]: GET /search?channel=fs&client=ubuntu&q=found+name+o
6413	11:47:09,171317	192.168.122.113	192.168.122.1	43470	8080	HTTP2	199	HEADERS[21]: GET /images/searchbox/desktop_searchbox_sprites318_
6478	11:47:11,063115	192.168.122.113	192.168.122.1	43470	8080	HTTP2	496	HEADERS[31]: GET /complete/search?q=found%20name%20on%20husband%20
6480	11:47:11,067729	192.168.122.113	192.168.122.1	43470	8080	HTTP2	280	HEADERS[33]: GET /complete/search?q=cp=0&client=gws-wiz&xssi=t&h
7696	11:47:29,001902	192.168.122.113	192.168.122.1	36000	8080	HTTP2	325	HEADERS[15]: GET /complete/search?client=firefox&q=bu, WINDOW_UP
7706	11:47:29,126215	192.168.122.113	192.168.122.1	36000	8080	HTTP2	165	HEADERS[17]: GET /complete/search?client=firefox&q=bus, WINDOW_U
7906	11:47:30,957218	192.168.122.113	192.168.122.1	43470	8080	HTTP2	228	HEADERS[65]: GET /search?channel=fs&client=ubuntu&q=partner+goes
8284	11:47:34,749075	192.168.122.113	192.168.122.1	43470	8080	HTTP2	345	HEADERS[75]: GET /complete/search?q=partner%20goes%20on%20many%20
8286	11:47:34,756494	192.168.122.113	192.168.122.1	43470	8080	HTTP2	301	HEADERS[77]: GET /complete/search?q=cp=0&client=gws-wiz&xssi=t&h
13117	11:48:36,067061	192.168.122.113	192.168.122.1	36700	8080	HTTP2	2827	HEADERS[25]: GET /wa/t.gif?name=nav.home.event.show_widget&tif=0
19839	11:50:15,758269	192.168.122.93	192.168.122.1	38488	8080	HTTP2	2827	HEADERS[25]: GET /wa/t.gif?name=nav.home.event.show_widget&tif=0
28307	11:52:11,541622	192.168.122.113	192.168.122.1	40480	8080	HTTP2	2827	HEADERS[29]: GET /wa/t.gif?name=nav.home.event.show_widget&tif=0
42591	11:55:28,661297	192.168.122.113	192.168.122.1	56408	8080	HTTP2	326	HEADERS[15]: GET /complete/search?client=firefox&q=arp, WINDOW_U
42768	11:55:32,431174	192.168.122.113	192.168.122.1	56408	8080	HTTP2	165	HEADERS[17]: GET /complete/search?client=firefox&q=mail, WINDOW_U
42780	11:55:32,629116	192.168.122.113	192.168.122.1	56408	8080	HTTP2	163	HEADERS[19]: GET /complete/search?client=firefox&q=mail, WINDOW_U
45712	11:55:58,345418	192.168.122.113	192.168.122.1	50780	8080	HTTP2	2827	HEADERS[31]: GET /wa/t.gif?name=nav.home.event.show_widget&tif=0
53238	11:57:34,031788	192.168.122.247	192.168.122.1	45930	8080	HTTP2	757	HEADERS[15]: GET /search?channel=fs&client=ubuntu&q=fancy+chande
53718	11:57:38,313657	192.168.122.247	192.168.122.1	45930	8080	HTTP2	764	HEADERS[27]: GET /complete/search?q=fancy%20chandeliers&cp=0&cli
53720	11:57:38,318057	192.168.122.247	192.168.122.1	45930	8080	HTTP2	331	HEADERS[29]: GET /complete/search?q=cp=0&client=gws-wiz-serp&xss
54100	11:57:40,597869	192.168.122.93	192.168.122.1	39812	8080	HTTP2	2827	HEADERS[31]: GET /wa/t.gif?name=nav.home.event.show_widget&tif=0
61316	12:03:46,561093	192.168.122.113	192.168.122.1	39710	8080	HTTP2	328	HEADERS[15]: GET /complete/search?client=firefox&q=proton, WINDO
62000	12:05:04,370500	192.168.122.113	192.168.122.1	39710	8080	HTTP2	164	HEADERS[17]: GET /complete/search?client=firefox&q=proton, WINDO

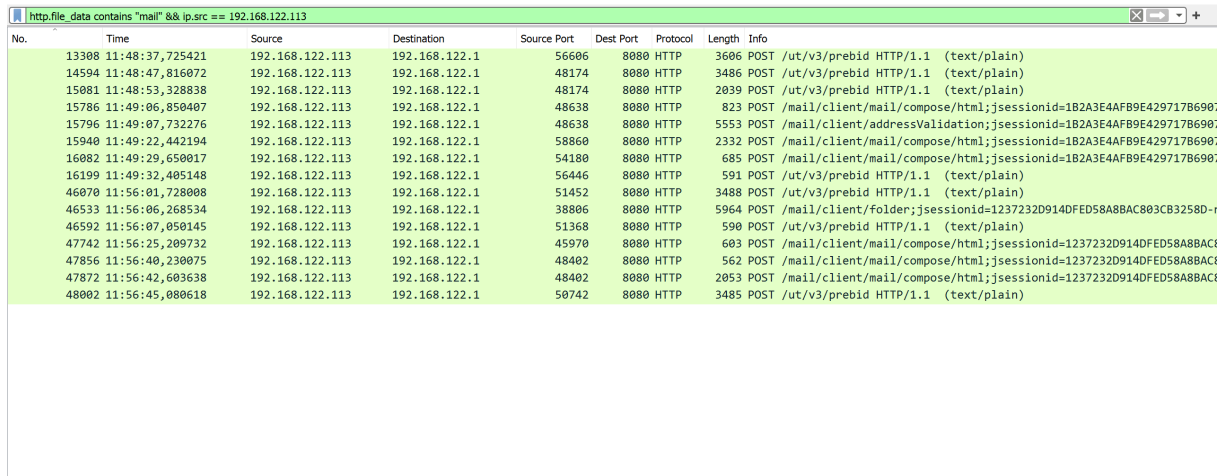
Figure 15: HTTP Path Example

1. What do you think is the objective of such a filter? Explain what you think the filter is looking for.
2. From what you can see in this list, can you give an example of a piece of information that you can learn?

9.6 HTTP File Data Example

Figure 16 shows the result of using the following filter(s):

- `http.file_data contains "mail"`, which shows only packets that contain the string “mail” in the data of a file transported by the HTTP protocol.



The image shows a Wireshark packet capture window with a filter applied: `http.file_data contains "mail" && ip.src == 192.168.122.113`. The packet list displays 18 packets, all of which are HTTP POST requests from 192.168.122.113 to 192.168.122.1. The packet details pane shows the structure of an HTTP POST request, including the status bar (200 OK), headers (Host, User-Agent, Content-Type, Content-Length), and the body (text/plain). The packet bytes pane shows the raw data of the request body, which contains the string "mail".

No.	Time	Source	Destination	Source Port	Dest Port	Protocol	Length	Info
13308	11:48:37,725421	192.168.122.113	192.168.122.1	56606	8080	HTTP	3606	POST /ut/v3/prebid HTTP/1.1 (text/plain)
14594	11:48:47,816072	192.168.122.113	192.168.122.1	48174	8080	HTTP	3486	POST /ut/v3/prebid HTTP/1.1 (text/plain)
15081	11:48:53,328838	192.168.122.113	192.168.122.1	48174	8080	HTTP	2039	POST /ut/v3/prebid HTTP/1.1 (text/plain)
15786	11:49:06,850407	192.168.122.113	192.168.122.1	48638	8080	HTTP	823	POST /mail/client/mail/compose/html;jsessionid=1B2A3E4AFB9E429717B690;
15796	11:49:07,732276	192.168.122.113	192.168.122.1	48638	8080	HTTP	5553	POST /mail/client/addressValidation;jsessionid=1B2A3E4AFB9E429717B690;
15940	11:49:22,442194	192.168.122.113	192.168.122.1	58860	8080	HTTP	2332	POST /mail/client/mail/compose/html;jsessionid=1B2A3E4AFB9E429717B690;
16082	11:49:29,650017	192.168.122.113	192.168.122.1	54180	8080	HTTP	685	POST /mail/client/mail/compose/html;jsessionid=1B2A3E4AFB9E429717B690;
16199	11:49:32,405148	192.168.122.113	192.168.122.1	56446	8080	HTTP	591	POST /ut/v3/prebid HTTP/1.1 (text/plain)
46070	11:56:01,728008	192.168.122.113	192.168.122.1	51452	8080	HTTP	3488	POST /ut/v3/prebid HTTP/1.1 (text/plain)
46533	11:56:06,268534	192.168.122.113	192.168.122.1	38806	8080	HTTP	5964	POST /mail/client/folder;jsessionid=1237232D914DFED58A8BAC803CB3258D-r
46592	11:56:07,050145	192.168.122.113	192.168.122.1	51368	8080	HTTP	590	POST /ut/v3/prebid HTTP/1.1 (text/plain)
47742	11:56:25,209732	192.168.122.113	192.168.122.1	45970	8080	HTTP	603	POST /mail/client/mail/compose/html;jsessionid=1237232D914DFED58A8BAC
47856	11:56:40,230075	192.168.122.113	192.168.122.1	48402	8080	HTTP	562	POST /mail/client/mail/compose/html;jsessionid=1237232D914DFED58A8BAC
47872	11:56:42,603638	192.168.122.113	192.168.122.1	48402	8080	HTTP	2053	POST /mail/client/mail/compose/html;jsessionid=1237232D914DFED58A8BAC
48002	11:56:45,080618	192.168.122.113	192.168.122.1	50742	8080	HTTP	3485	POST /ut/v3/prebid HTTP/1.1 (text/plain)

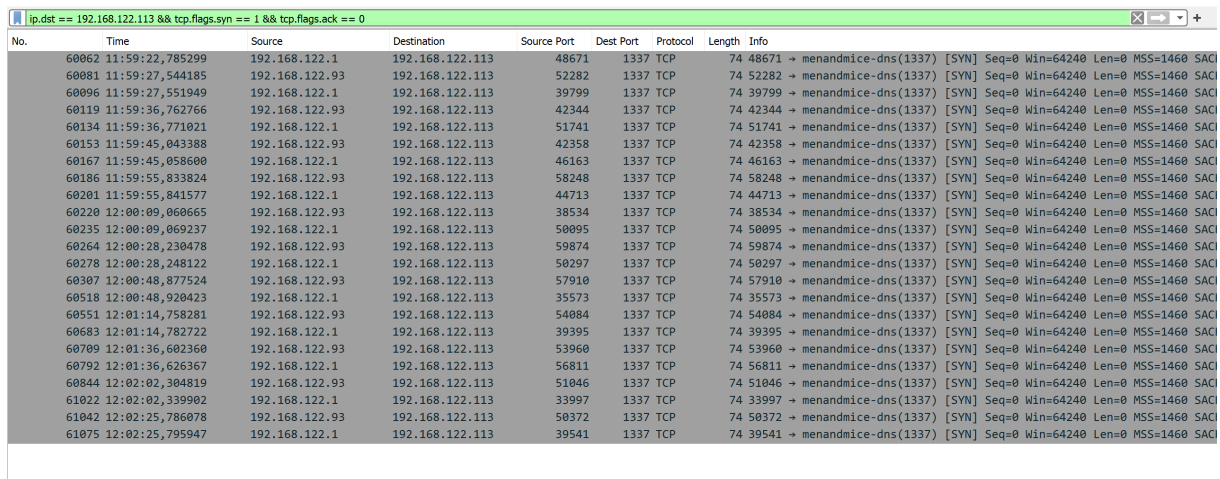
Figure 16: HTTP File Data Example

1. What do you think is the objective of such a filter? Explain what you think the filter is looking for.
2. How can you modify the filter to perform a more targeted search?

9.7 TCP Syn Flag Example

Figure 17 shows the result of using the following filter(s):

- `tcp.flags.syn == 1`, which shows only packets with the TCP syn flag set.
- `tcp.flags.ack == 0`, which shows only packets with the TCP ack flag unset.
- `ip.dst == 192.168.122.113`, which shows only packets where the destination IP address is the specified one.



No.	Time	Source	Destination	Source Port	Dest Port	Protocol	Length	Info
60062	11:59:22,785299	192.168.122.1	192.168.122.113	48671	1337	TCP	74	48671 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60081	11:59:27,544185	192.168.122.93	192.168.122.113	52282	1337	TCP	74	52282 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60096	11:59:27,551949	192.168.122.1	192.168.122.113	39799	1337	TCP	74	39799 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60119	11:59:36,762766	192.168.122.93	192.168.122.113	42344	1337	TCP	74	42344 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60134	11:59:36,771021	192.168.122.1	192.168.122.113	51741	1337	TCP	74	51741 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60153	11:59:45,043388	192.168.122.93	192.168.122.113	42358	1337	TCP	74	42358 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60167	11:59:45,058600	192.168.122.1	192.168.122.113	46163	1337	TCP	74	46163 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60186	11:59:55,833824	192.168.122.93	192.168.122.113	58248	1337	TCP	74	58248 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60201	11:59:55,841577	192.168.122.1	192.168.122.113	44713	1337	TCP	74	44713 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60220	12:00:09,060665	192.168.122.93	192.168.122.113	38534	1337	TCP	74	38534 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60235	12:00:09,069237	192.168.122.1	192.168.122.113	50095	1337	TCP	74	50095 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60264	12:00:28,238478	192.168.122.93	192.168.122.113	59874	1337	TCP	74	59874 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60278	12:00:28,248122	192.168.122.1	192.168.122.113	50297	1337	TCP	74	50297 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60307	12:00:48,877524	192.168.122.93	192.168.122.113	57910	1337	TCP	74	57910 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60518	12:00:48,920423	192.168.122.1	192.168.122.113	35573	1337	TCP	74	35573 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60551	12:01:14,758281	192.168.122.93	192.168.122.113	54084	1337	TCP	74	54084 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60683	12:01:14,782722	192.168.122.1	192.168.122.113	39395	1337	TCP	74	39395 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60709	12:01:36,602360	192.168.122.93	192.168.122.113	53960	1337	TCP	74	53960 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60792	12:01:36,626367	192.168.122.1	192.168.122.113	56811	1337	TCP	74	56811 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
60844	12:02:02,304819	192.168.122.93	192.168.122.113	51046	1337	TCP	74	51046 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
61022	12:02:02,339902	192.168.122.1	192.168.122.113	33997	1337	TCP	74	33997 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
61042	12:02:25,786078	192.168.122.93	192.168.122.113	50372	1337	TCP	74	50372 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
61075	12:02:25,795947	192.168.122.1	192.168.122.113	39541	1337	TCP	74	39541 → menandmice-dns(1337) [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC

Figure 17: TCP Syn Flag Example

1. Is the resulting list of packets normal, or do you find anything suspicious about it?
2. What kind of packet does the combination `tcp.flags.syn == 1 && tcp.flags.ack == 0` filter for? Explain in simple terms.
3. Why would it be relevant to perform such a search with the user's computer as destination?