Exp05

Matteo Di Fabio: 264339

Luca Genovese: 264364

The goal of this experiment was to extract file from an HTTP session using a python script. Now we are going to explain all the step we followed during the developed phases. First of all we studied what is a dissector and which tools or library are available to carry out this kind of analysis. One of the more famous and used tools is Scapy: *“Scapy is a Python program that enables the user to send, sniff and dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks”*.   
There are also other library that make more or less the same things like:

* The jspcap project is an open source Python program focus on [PCAP](https://en.wikipedia.org/wiki/Pcap) parsing and analysis, which works as a stream PCAP file extractor. With support of [jsformat](https://github.com/JarryShaw/jsformat), it shall support multiple output report formats.
* dpkt is a python module for fast, simple packet creation / parsing, with definitions for the basic TCP/IP protocols.
* The [PyPCAPKit](https://pypcapkit.readthedocs.io/en/latest/pcapkit.html#module-pcapkit) project is an open source Python program focus on [PCAP](https://en.wikipedia.org/wiki/Pcap) parsing and analysis, which works as a stream PCAP file extractor. With support of DictDumper, it shall support multiple output report formats.

With this tool is in fact possible to filter all the traffic detected, using differents techniques and methods. Studying these library and tools we understand that all of them have more or less the same working structure, and the logic behind them is the very similar each other.

The logic is to analyse all the packet in the detected traffic and reading the headers of each packet (where are written all the basics informations of the packet) and knowing this informations is possible to group together all packages belonging to the same session. Doing that we are able to analyse all the traffic of a specific session and recognize all the request and reply that the server and the clients exchange each other. For example from the HTTP request we can read the name of the requested file, then a new file on local machine will be create (in this way we can export and write the transmitted file). At this point all the data read in TCP packet are written in the file, following the sequence number of the packet: it is very important to check that all the packets are present and the file is not corrupted, otherwise we won’t be able to open the file correctly, once it was totally written.

Doing other research we found another solution to extract the file from HTTP. In this case we are going to exploit tshark library, in particular exist this command:

tshark -nr INPUT\_FILE --export-objects http, OUTPUT\_DIRECTORY

that allow us to extract all the HTTP file present in the analysed traffic.

At the end we decided to use this second solution.

We will use the same format suggested in the notes of the experiment text:

*$ python exp05.py -r mytraffic.pcap*

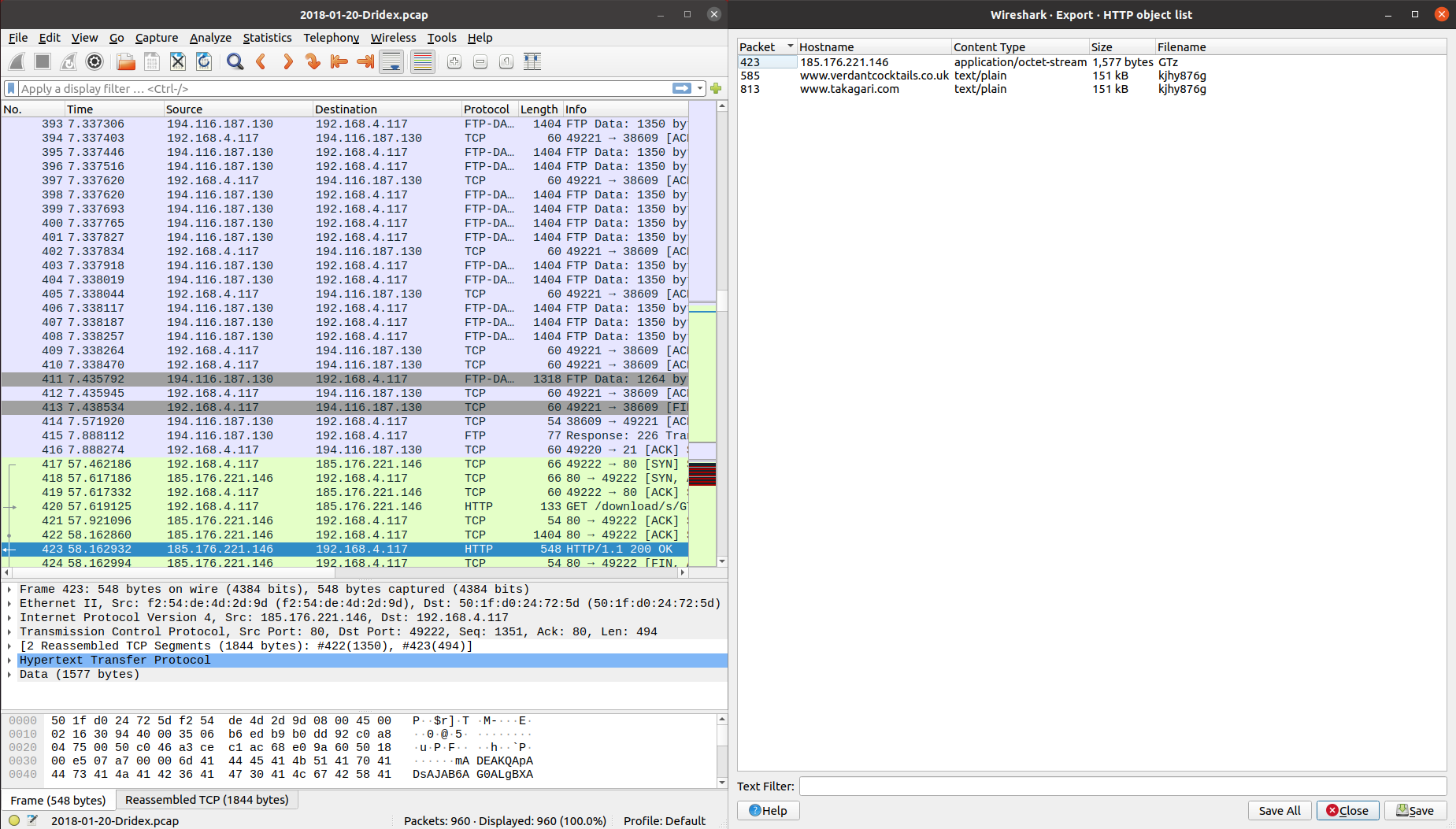
In this way we will dissect the file *mytraffic.pcap* and extracting all the files it contains into a folder called *temp* located in the directory where the programme is launched.

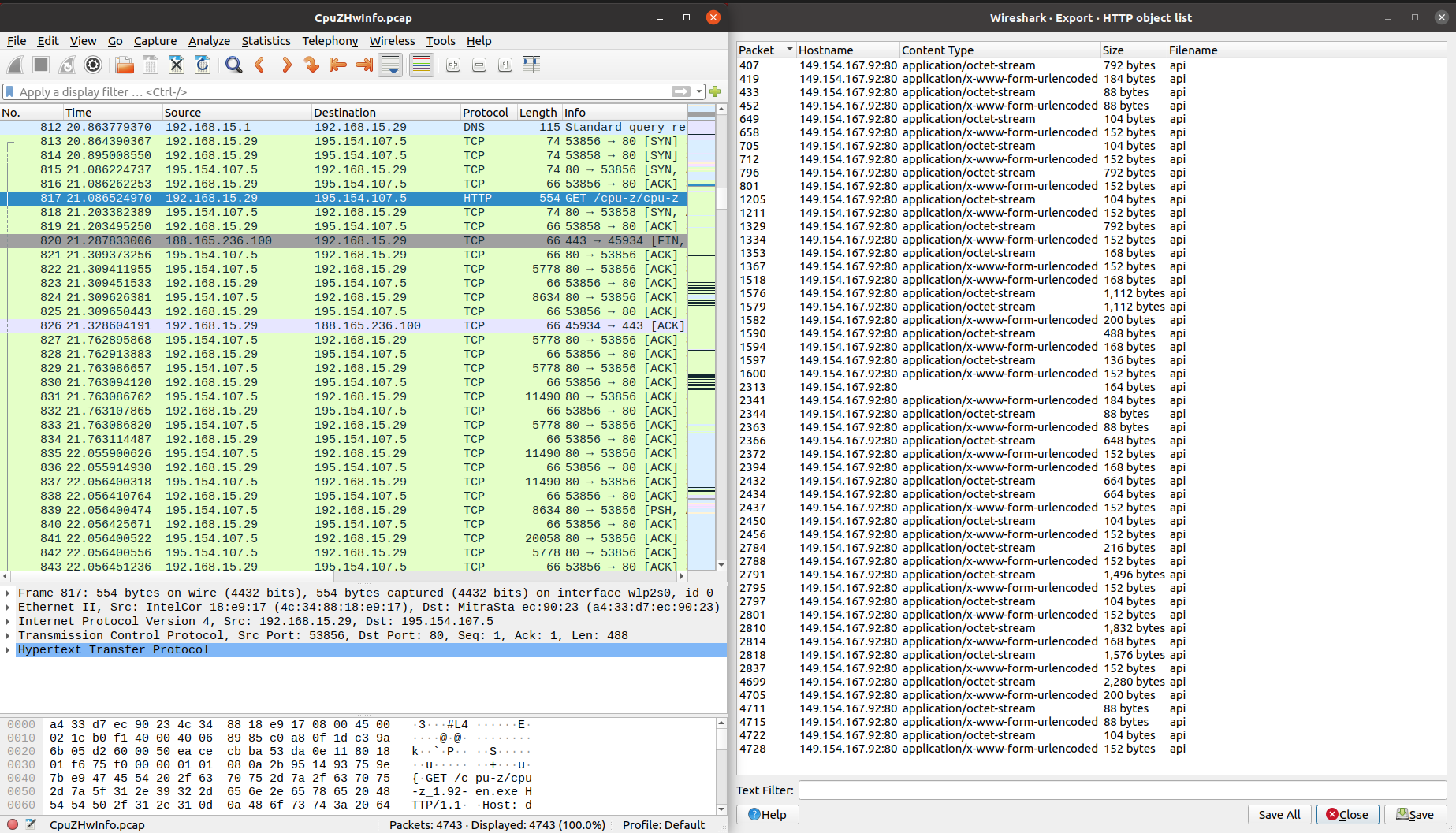
## **Test:**

To test our program we decided to submit different pcap capture to see how it works and if correctly extract all the file present in the session. To check that the extracted file are correct we decided to make a comparison between our dissector and wireshark extract tool.

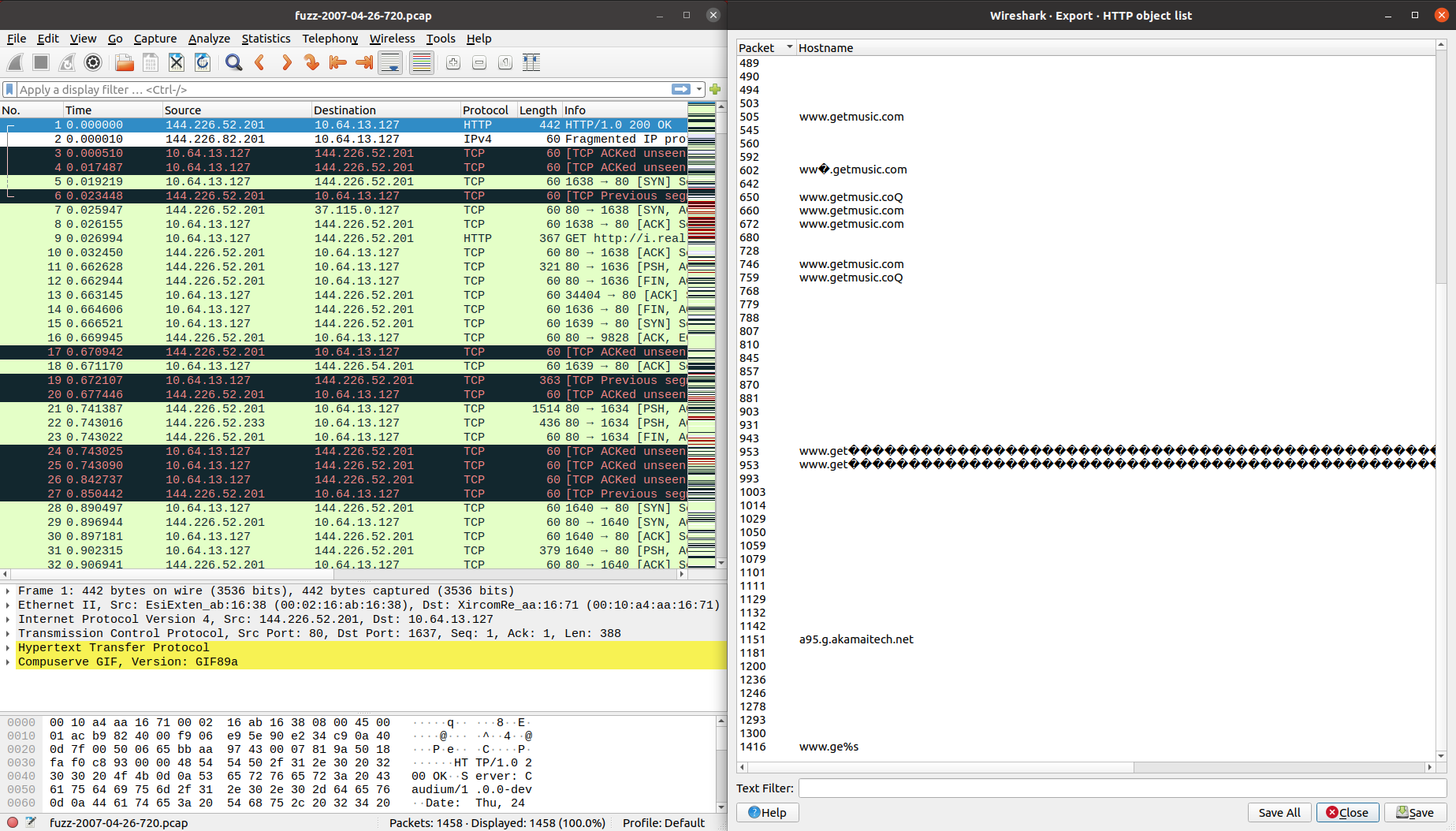
In all our tests, we have compared the extracted files from our program and the ones from wireshark: we observe that the files downloaded from our dissector where the same of wireshark.  
We have obtained all the same file listed in the HTTP dissector of wireshark (as you can see in the screenshot of each tested pcap file)

Test 1: 2018-01-20-Dridex.pcap

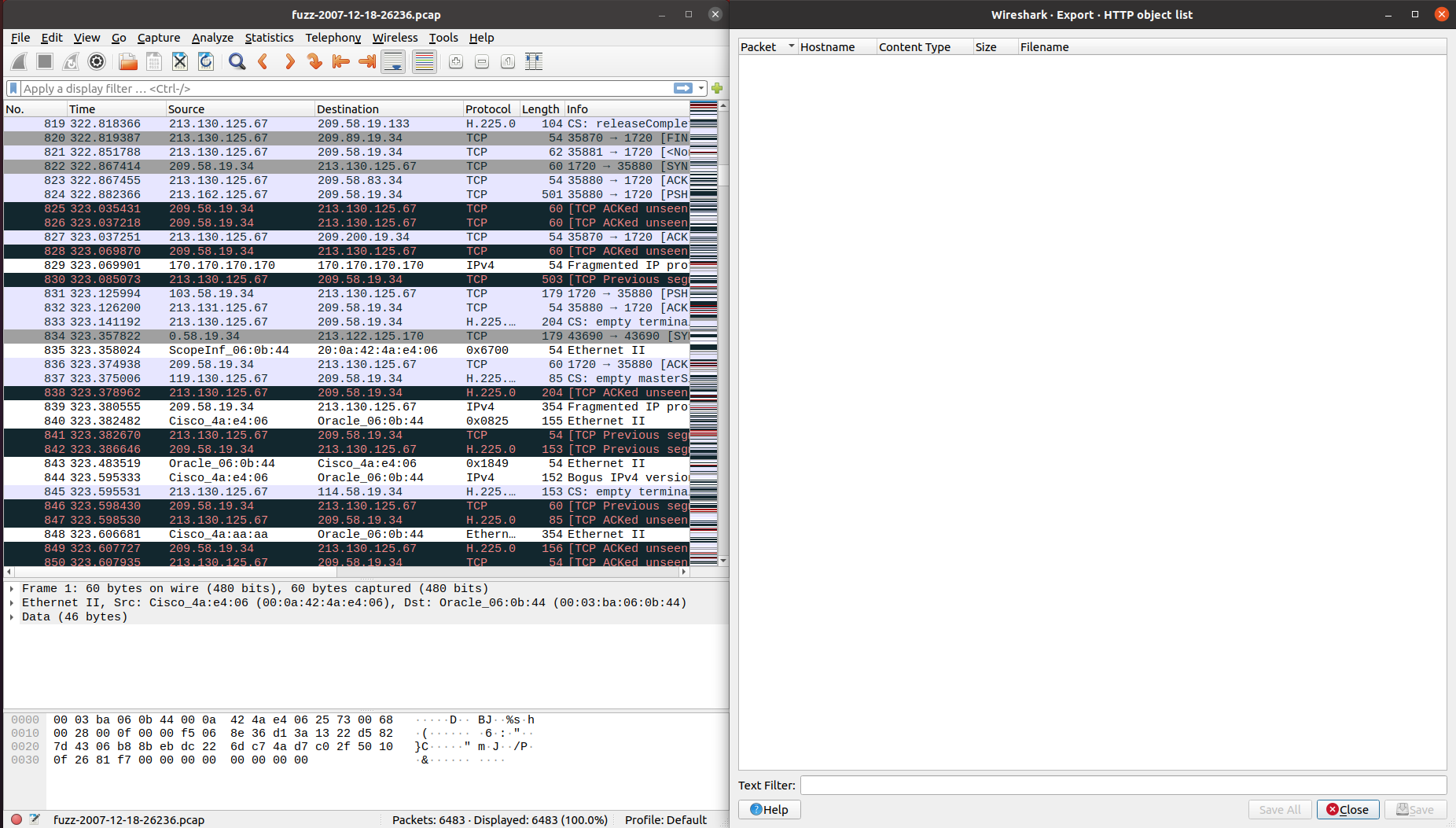


Test 2: CpuZHwInfo.pcap

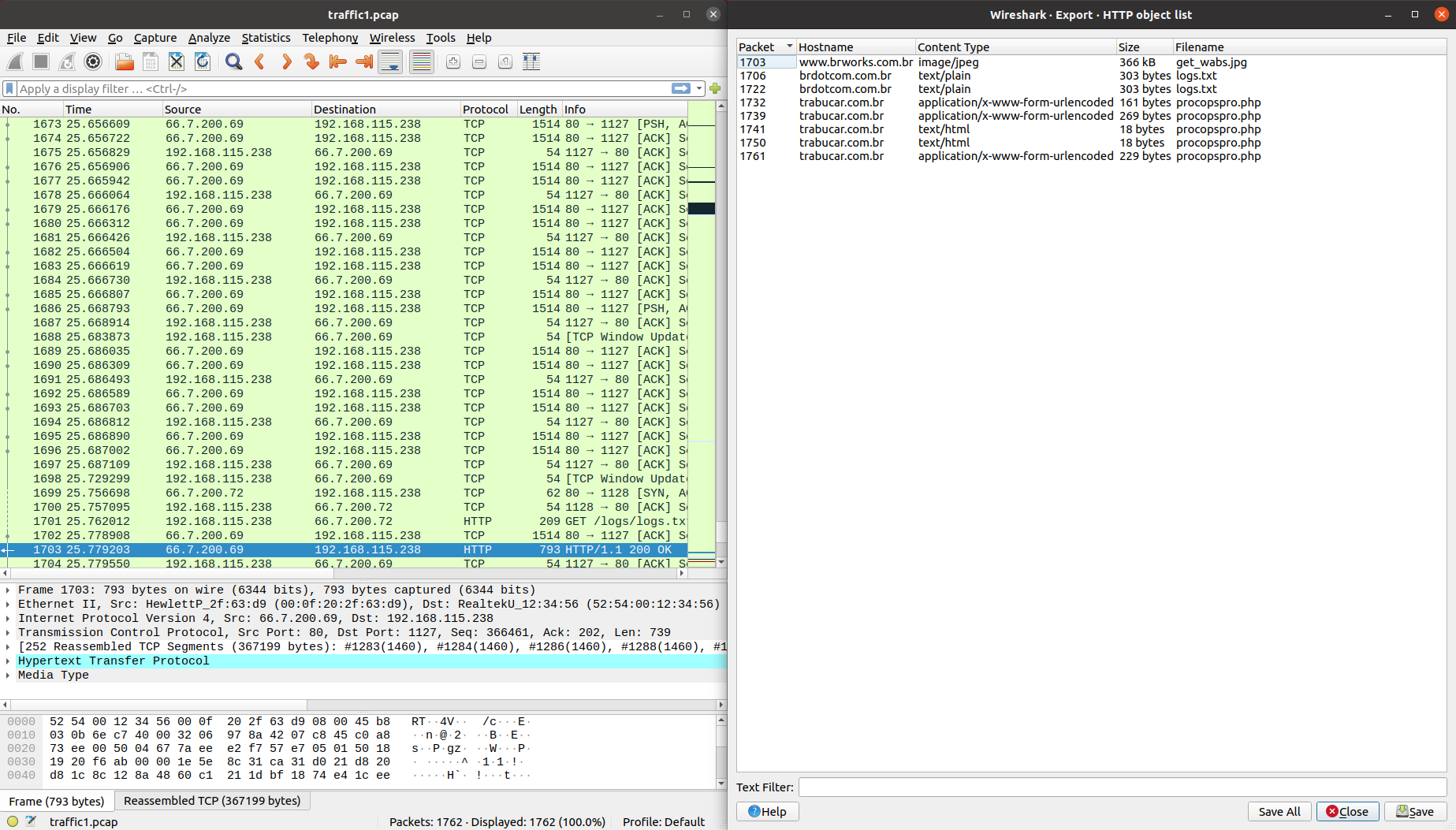
Test 3: fuzz-2007-04-26-720.pcap



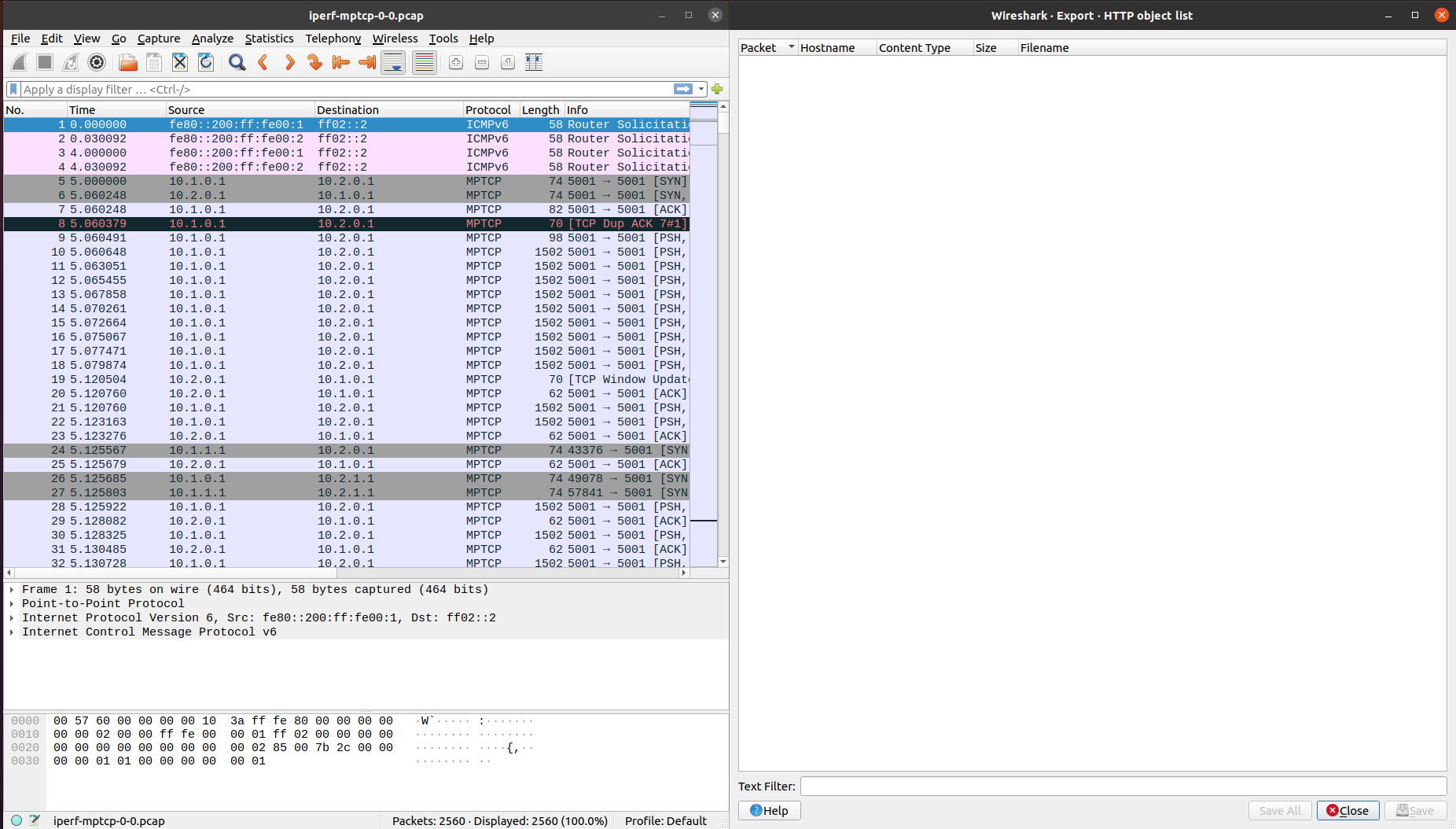
Test 4: fuzz-2007-12-18-26236.pcap



Test 5: traffic1.pcap



Test 6: iperf-mptcp-0-0.pcap



## Note: due to the use of tshark, our program has a dependencies, please install tshark with the following command: sudo apt-get install -y tshark

## HTTP Attack:

An attack that exploit dissector could be lead in this way: if the attacker can read all the traffic between one client and a server, analysing all the packets they exchange he could be able to reconstruct and export some files that are send through TCP connection. This protocol is critical because the data in the transmitted packets are in clear and everybody could read and export files as we saw. In this file could be present sensitive informations that the attacker could exploit with malicious intent. A simple countermeasure is for example to use the protocol HTTPS instead of HTTP because HTTPS is more secure as all the exchanged packets are encrypted and for this reason the attacker could anyway extract the file, but he can read and understand the content as it is encrypted.