

Capturing of network traffic in the local network

Laboratory protocol Exercise 10: Capturing of network traffic in the local network



Figure 1: Grouplogo

Subject: ITSI
Class: 3AHITN
Name: Stefan Fürst, Justin Tremurici
Gruppenname/Nummer: Name here/12
Supervisor: SPAC, ZIVK
Exercise dates: 11.04.2025 | 25.04.2025 | 09.05.2025
Submission date: 16.05.2025

Contents

1 Task definition	3
2 Summary	3
3 Complete network topology of the exercise	4
4 Exercise Execution	5
4.1 Building the Topologies	5
4.1.1 Mirroring traffic in RouterOS v7	5
4.1.2 Comparing the traffic before and after the configurion	7
4.2 Packet Sniffing on the Local Device	8
4.3 Capturing a Ping Between Two Targets	9
4.4 Capturing Plain Text Passwords	10
4.5 Capturing a VoIP Call	11
5 References	14
6 List of figures	15

1 Task definition

This task focused on the passive interception of network traffic in a local network using either a hub or a managed switch with mirror ports. The objective was to analyze unaltered communications using **Wireshark** on both attacker and victim machines. Two topologies were tested: a hub-based setup, which allowed full traffic visibility, and a switch-based setup, where traffic was mirrored from victim ports to the attacker's port. Devices were assigned static IP addresses from a private range, and VoIP communication was simulated using either software-based or physical IP phones.

Three types of traffic were examined: ICMP echo requests (Ping), HTTP authentication involving plaintext credentials, and VoIP calls between two endpoints. Each case was recorded in a separate **Wireshark** capture. In the hub scenario, the focus was on visibility and potential stability issues under high traffic. For the switch, mirroring was configured and traffic was captured before and after to assess changes.

Further tasks involved filtering ICMP traffic by attacker IP, observing ping communication between victim devices from the attacker's perspective, capturing HTTP login attempts to extract credentials, and intercepting a VoIP call, which was exported as an MP3 file. All relevant captures and the audio file were submitted as part of the final documentation.¹

2 Summary

In this exercise, two distinct network topologies were implemented to investigate passive network traffic interception. The first topology utilized personal hardware, specifically a **Mikrotik RB5009** router, to configure port mirroring. The client devices were older laptops running **Proxmox**, with one laptop hosting an **nginx** container configured to demonstrate basic HTTP authentication. The attacker device was another laptop connected to the mirrored ports on the router, which allowed it to receive a complete copy of the network traffic between the clients and the server.

The initial step involved performing local ICMP ping requests from the attacker to the clients to observe the captured traffic and verify network connectivity. Following this, the two client laptops pinged each other, while the attacker monitored and recorded the exchanged packets. This demonstrated the attacker's ability to intercept traffic not directly addressed to it due to the port mirroring setup. Furthermore, the attacker was able to capture and analyze the HTTP basic authentication process, successfully extracting plaintext credentials transmitted from the client to the **nginx** server.

In the second part of the exercise, a VoIP call was established using two IP phones connected via a network hub instead of a switch with port mirroring. This topology allowed the attacker laptop to capture the audio stream of the call directly from the network traffic. The recorded audio was then exported and post-processed using **Audacity** and **Adobe Podcast Speech Enhancer** to clean and enhance the recording, resulting in a clear and intelligible audio file.

Throughout the exercise, **Wireshark** was extensively used to capture, filter, and analyze the network traffic from the attacker's perspective. This practical approach provided insight into how network devices like hubs and switches with port mirroring impact the visibility of traffic and the feasibility of passive interception attacks within a local network environment.²

¹The task definition was created by ChatGPT.

²The summary was created after providing a draft of bullet points to ChatGPT.

3 Complete network topology of the exercise

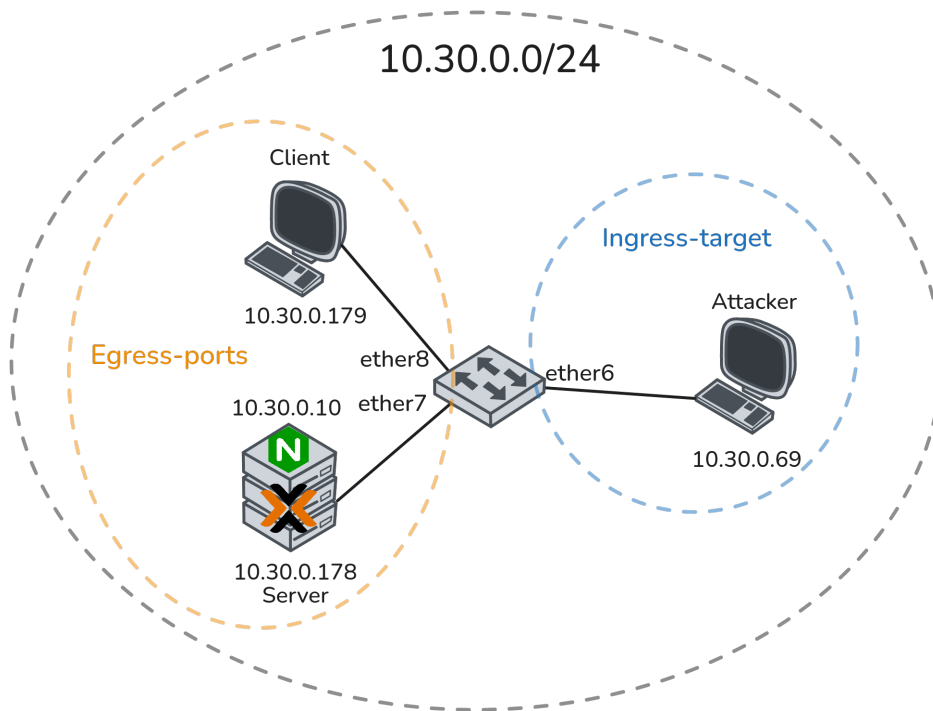


Figure 2: Complete network topology of the exercise using a switch

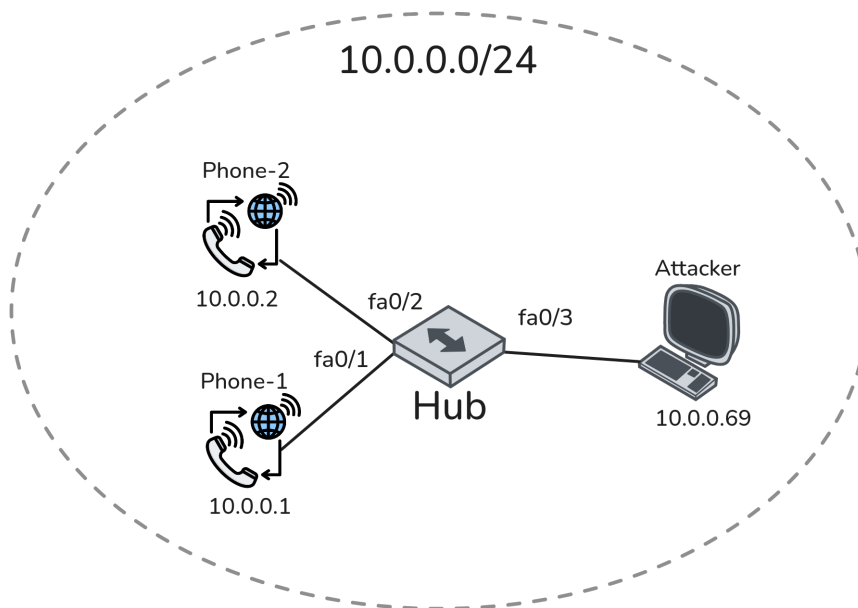


Figure 3: Complete network topology of the exercise using a Hub

4 Exercise Execution

4.1 Building the Topologies

To build the topology from [Figure 2](#), I chose the following hardware: a Mikrotik RB 5009 to act as the main "switch" due to RouterOS offering extensive settings in what I consider the best GUI to manage any network device.

For the server and clients, I used two old laptops running Proxmox, one of which has a Debian server VM running an Nginx web server with basic authentication set up. All of the devices have static IPs configured in the range 10.30.0.0/24. The attacker simply runs Linux with Wireshark to capture the traffic. The used IP addresses can be found in the addressing table below.

Deivce	IP
Attacker	10.30.0.69
Server	10.30.0.179
Webserver	10.30.0.10
Client	10.30.0.179

Table 1: Addressing Table

4.1.1 Mirroring traffic in RouterOS v7

To configure the router, there are three options: either use the WebGUI, SSH into it, or use their program called WinBox, which is the option I went with. After connecting a port on the router, it automatically detects available ports, and I can simply select one of them and configure everything as needed via the MAC address, as seen in Figure 4.

Note that it shows 192.168.88.1, which is the default IP of the router, but this won't be used, as all the devices already have their network setup. It can therefore be ignored, since no additional configuration with VLANs is needed—it's just plug and play.

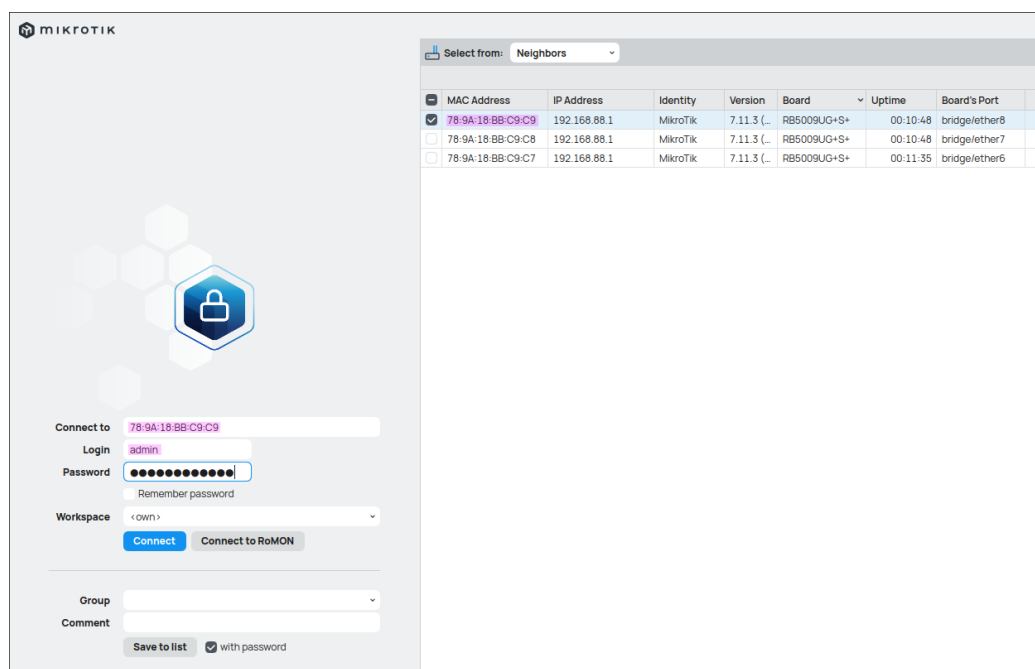


Figure 4: Connecting to the Router via Winbox

Now that we are in the router's configuration, we see a number of top-level options to choose from. To mirror traffic, we go to the **Switch** section and head to the **Port** tab, where we select the ports we want to mirror. If we double-click on an interface, it opens the port window, where we can choose whether to mirror only ingress traffic, egress traffic, or both.

We also specify an ingress target, which in this case is **ether6**, where the attacker's laptop is plugged in so that it receives all the mirrored traffic. The configuration for both **ether7** and **ether8** is the same, which is why only one is shown in Figure 5 below. Lastly, under the "Mirror Ingress"/"Mirror Egress" columns in the switch window table, we can see a **yes** in both columns, indicating that the configuration has been successfully applied. [1]

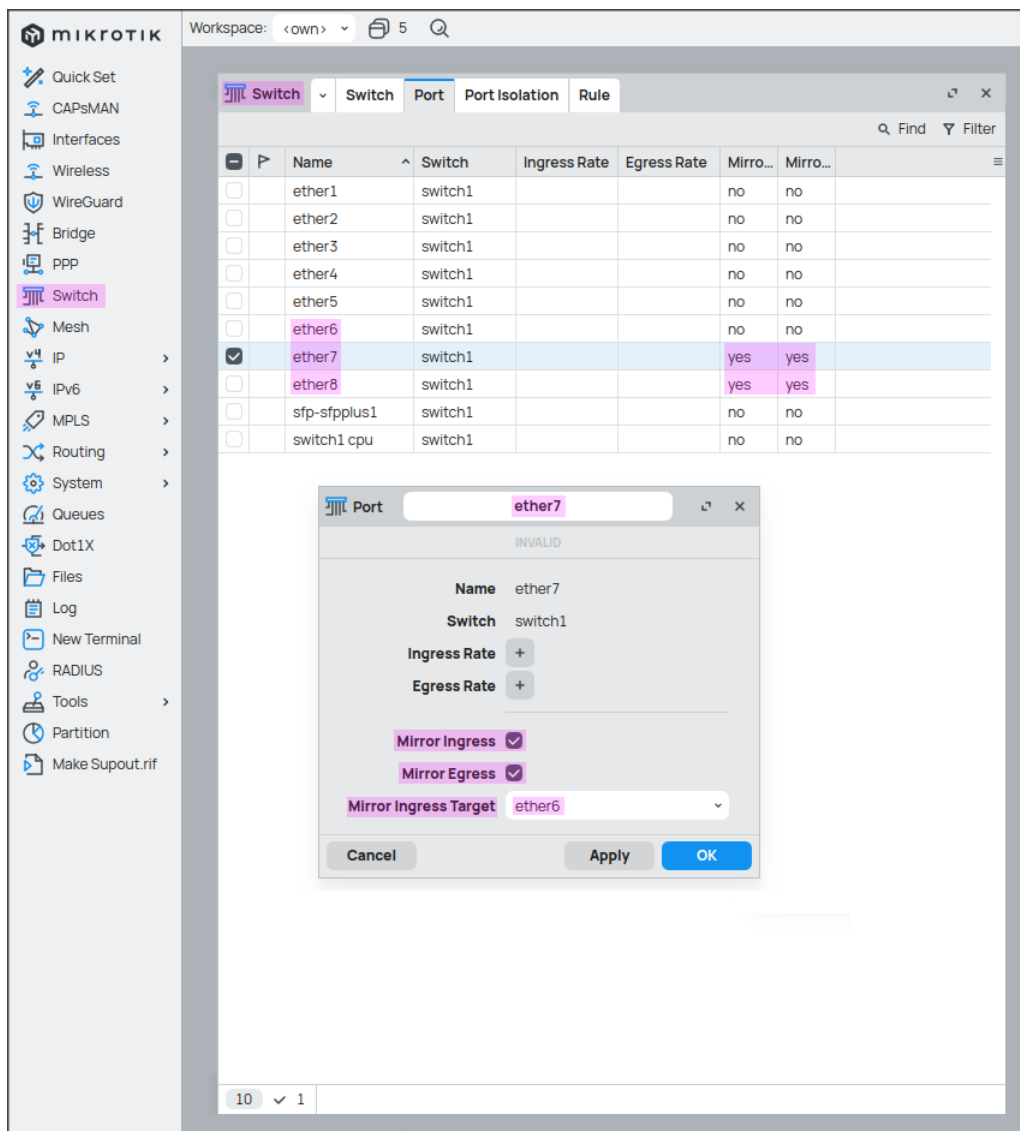
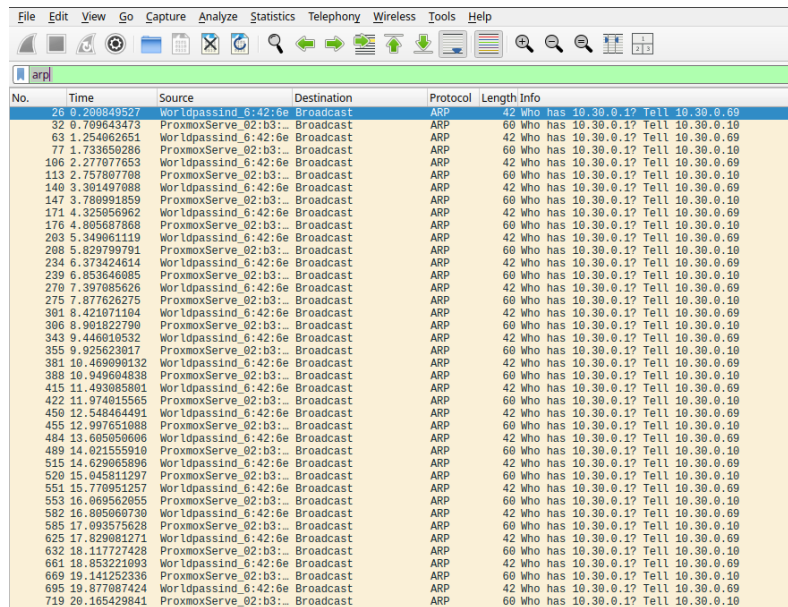


Figure 5: Examining the traffic mirror configuration

4.1.2 Comparing the traffic before and after the configuration

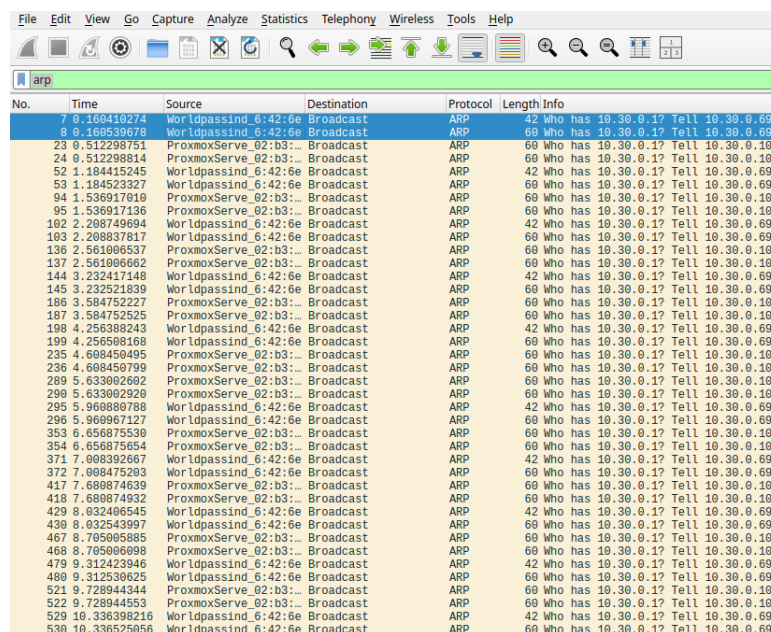
Now we can use Wireshark on the attacker's laptop to compare the traffic captured with and without mirroring. When everything is idle and only ARP traffic is occurring in the background, the only difference is that instead of receiving each broadcast once, it is received twice—once from the connection itself and once from the mirroring. Figure 6 shows ARP traffic without mirroring, while Figure 7 shows it with mirroring, in which all the broadcasts appear twice. These duplicated frames either appear directly one after the other or with a few in between. The interruptions in frame order are caused by me having WinBox open, which continuously streams frames and therefore causes some variation in the captured frame sequence.



The screenshot shows the Wireshark interface with the 'arp' filter applied. The packet list shows a series of ARP broadcast frames. Each frame has a source of 'Worldpassind_6:42:6e' and a destination of 'Broadcast'. The protocol is 'ARP' and the length is 42 bytes. The information field for each packet is '42 Who has 10.30.0.1? Tell 10.30.0.69'. The frames are numbered sequentially from 26 to 719.

No.	Time	Source	Destination	Protocol	Length	Info
26	0.200849527	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
32	0.709645473	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
63	1.254062551	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
77	1.733650286	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
106	2.277077653	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
113	2.757807788	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
140	3.381497088	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
147	3.788991859	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
171	4.325056962	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
176	4.805687868	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
203	5.349961119	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
208	5.829799791	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
234	6.373424614	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
239	6.853646085	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
270	7.397085626	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
275	7.877626275	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
301	8.422071104	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
306	8.901822790	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
343	9.446010532	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
355	9.925623017	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
381	10.468909132	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
388	10.949604838	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
415	11.493085801	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
422	11.974015565	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
450	12.548464491	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
455	12.997651088	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
484	13.605059066	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
489	14.021555910	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
515	14.629065896	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
520	15.045811297	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
551	15.770951257	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
553	16.069562655	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
582	16.805060730	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
585	17.093575628	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
625	17.829001271	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
632	18.117727428	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
661	18.853221093	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
669	19.141252336	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
695	19.877087424	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
719	20.165429841	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10

Figure 6: Examining the arp traffic without a mirror configuration



The screenshot shows the Wireshark interface with the 'arp' filter applied. The packet list shows a series of ARP broadcast frames, each appearing twice due to mirroring. Each frame has a source of 'Worldpassind_6:42:6e' and a destination of 'Broadcast'. The protocol is 'ARP' and the length is 42 bytes. The information field for each packet is '42 Who has 10.30.0.1? Tell 10.30.0.69'. The frames are numbered sequentially from 7 to 539.

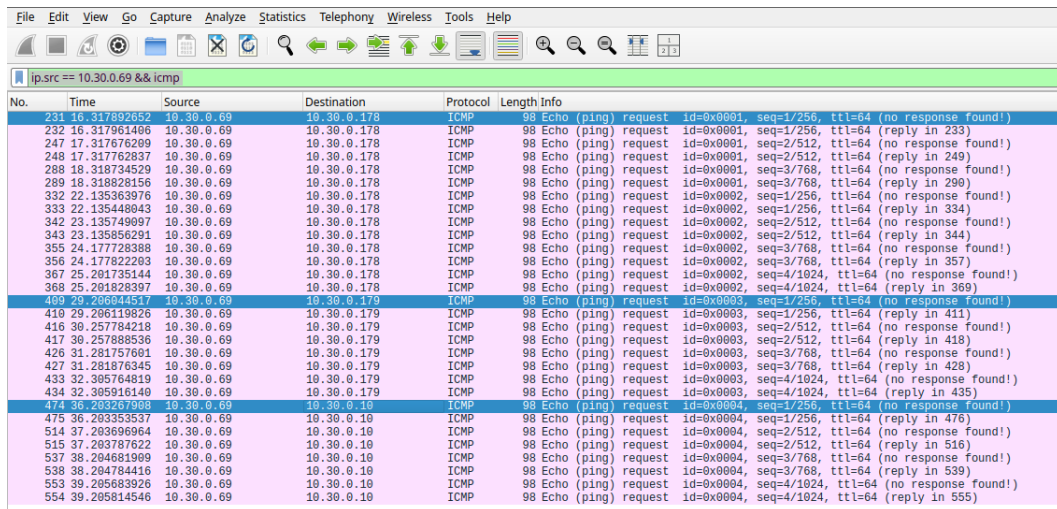
No.	Time	Source	Destination	Protocol	Length	Info
7	0.160410274	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
8	0.160539376	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
23	0.512290751	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
24	0.512298814	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
52	1.184415245	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
53	1.184523327	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
94	1.536917910	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
95	1.536917136	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
102	2.208749694	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
103	2.208837817	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
136	2.561006537	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
137	2.561006662	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
144	3.232417148	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
145	3.232521839	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
186	3.584752227	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
187	3.584752525	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
198	4.256388243	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
199	4.256508168	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
235	4.608450495	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
236	4.608450799	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
289	5.633002602	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
290	5.633002920	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
295	5.960880788	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
296	5.960967127	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
353	6.656875530	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
354	6.656875654	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
371	7.008392667	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
372	7.008475203	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
417	7.608074639	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
418	7.608074932	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
429	8.032406545	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
430	8.032543997	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
467	8.705005885	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
468	8.705006098	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
479	9.312423046	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
480	9.312530625	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
521	9.728944344	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
522	9.728944553	ProxmoxServe_02:b3:...	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10
529	10.336398216	Worldpassind_6:42:6e	Broadcast	ARP	42	Who has 10.30.0.1? Tell 10.30.0.69
530	10.336525056	Worldpassind_6:42:6e	Broadcast	ARP	60	Who has 10.30.0.1? Tell 10.30.0.10

Figure 7: Examining the arp traffic with a mirror configuration

4.2 Packet Sniffing on the Local Device

Now, with mirroring enabled, every device on the network is pinged so we can examine the behavior using the following filter: `ip.src == 10.30.0.69 && icmp`. This filter shows only ICMP frames with the source IP of the attacker's laptop.

The output of the filter displays all the issued ping requests, which can be verified in Figure 8 below.

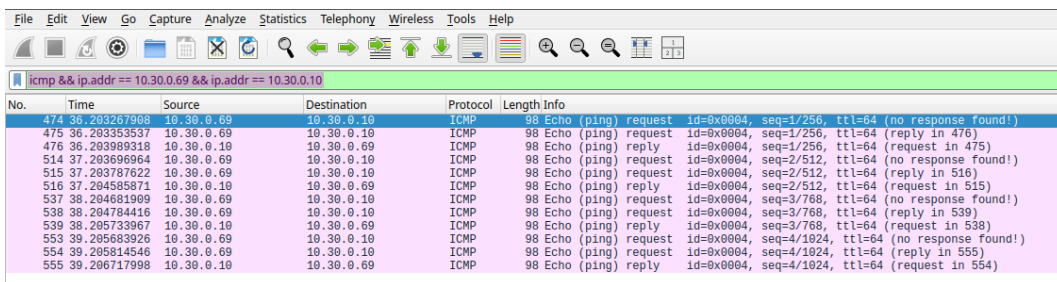


No.	Time	Source	Destination	Protocol	Length	Info
231	16.317892652	10.30.0.69	10.30.0.178	ICMP	68	Echo (ping) request id=0x0001, seq=1/256, ttl=64 (no response found!)
232	16.317961406	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0001, seq=1/256, ttl=64 (reply in 233)
247	17.317676209	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0001, seq=2/512, ttl=64 (no response found!)
248	17.317762837	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0001, seq=2/512, ttl=64 (reply in 249)
288	18.318734529	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0001, seq=3/768, ttl=64 (no response found!)
289	18.318828156	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0001, seq=3/768, ttl=64 (reply in 290)
332	22.135363976	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0002, seq=1/256, ttl=64 (no response found!)
333	22.135448043	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0002, seq=1/256, ttl=64 (reply in 334)
342	23.135749097	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0002, seq=2/512, ttl=64 (no response found!)
343	23.135856291	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0002, seq=2/512, ttl=64 (reply in 344)
355	24.177726388	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0002, seq=3/768, ttl=64 (no response found!)
356	24.177822293	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0002, seq=3/768, ttl=64 (reply in 357)
367	25.201735144	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0002, seq=4/1024, ttl=64 (no response found!)
368	25.201828397	10.30.0.69	10.30.0.178	ICMP	98	Echo (ping) request id=0x0002, seq=4/1024, ttl=64 (reply in 369)
409	29.206044517	10.30.0.69	10.30.0.179	ICMP	98	Echo (ping) request id=0x0003, seq=1/256, ttl=64 (no response found!)
410	29.206119826	10.30.0.69	10.30.0.179	ICMP	98	Echo (ping) request id=0x0003, seq=1/256, ttl=64 (reply in 411)
416	30.257784218	10.30.0.69	10.30.0.179	ICMP	98	Echo (ping) request id=0x0003, seq=2/512, ttl=64 (no response found!)
417	30.257888536	10.30.0.69	10.30.0.179	ICMP	98	Echo (ping) request id=0x0003, seq=2/512, ttl=64 (reply in 418)
426	31.281757681	10.30.0.69	10.30.0.179	ICMP	98	Echo (ping) request id=0x0003, seq=3/768, ttl=64 (no response found!)
427	31.281876345	10.30.0.69	10.30.0.179	ICMP	98	Echo (ping) request id=0x0003, seq=3/768, ttl=64 (reply in 428)
433	32.305764819	10.30.0.69	10.30.0.179	ICMP	98	Echo (ping) request id=0x0003, seq=4/1024, ttl=64 (no response found!)
434	32.305916140	10.30.0.69	10.30.0.179	ICMP	98	Echo (ping) request id=0x0003, seq=4/1024, ttl=64 (reply in 435)
474	36.203267908	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=1/256, ttl=64 (no response found!)
475	36.203353537	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=1/256, ttl=64 (reply in 476)
476	36.203989318	10.30.0.10	10.30.0.69	ICMP	98	Echo (ping) reply id=0x0004, seq=1/256, ttl=64 (request in 475)
514	37.203696964	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=2/512, ttl=64 (no response found!)
515	37.203787622	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=2/512, ttl=64 (reply in 516)
516	37.204585871	10.30.0.10	10.30.0.69	ICMP	98	Echo (ping) reply id=0x0004, seq=2/512, ttl=64 (request in 515)
537	38.204681909	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=3/768, ttl=64 (no response found!)
538	38.204784416	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=3/768, ttl=64 (reply in 539)
539	38.205733957	10.30.0.10	10.30.0.69	ICMP	98	Echo (ping) reply id=0x0004, seq=3/768, ttl=64 (request in 538)
553	39.205683926	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=4/1024, ttl=64 (no response found!)
554	39.205814546	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=4/1024, ttl=64 (reply in 555)
555	39.206717998	10.30.0.10	10.30.0.69	ICMP	98	Echo (ping) reply id=0x0004, seq=4/1024, ttl=64 (request in 554)

Figure 8: Displaying the pings to every device on the Network

To display only the full connection between the two devices, the following filter can be used to show only the complete exchange, including replies: `icmp && ip.addr == 10.30.0.69 && ip.addr == 10.30.0.10`.

This filter first includes only all ICMP frames, and then narrows it down to frames that involve both the IP address of the attacker's laptop and the target being pinged. This way, only the connection between the two—both requests and replies—is shown, as illustrated in Figure 9.



No.	Time	Source	Destination	Protocol	Length	Info
474	36.203267908	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=1/256, ttl=64 (no response found!)
475	36.203353537	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=1/256, ttl=64 (reply in 476)
476	36.203989318	10.30.0.10	10.30.0.69	ICMP	98	Echo (ping) reply id=0x0004, seq=1/256, ttl=64 (request in 475)
514	37.203696964	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=2/512, ttl=64 (no response found!)
515	37.203787622	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=2/512, ttl=64 (reply in 516)
516	37.204585871	10.30.0.10	10.30.0.69	ICMP	98	Echo (ping) reply id=0x0004, seq=2/512, ttl=64 (request in 515)
537	38.204681909	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=3/768, ttl=64 (no response found!)
538	38.204784416	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=3/768, ttl=64 (reply in 539)
539	38.205733957	10.30.0.10	10.30.0.69	ICMP	98	Echo (ping) reply id=0x0004, seq=3/768, ttl=64 (request in 538)
553	39.205683926	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=4/1024, ttl=64 (no response found!)
554	39.205814546	10.30.0.69	10.30.0.10	ICMP	98	Echo (ping) request id=0x0004, seq=4/1024, ttl=64 (reply in 555)
555	39.206717998	10.30.0.10	10.30.0.69	ICMP	98	Echo (ping) reply id=0x0004, seq=4/1024, ttl=64 (request in 554)

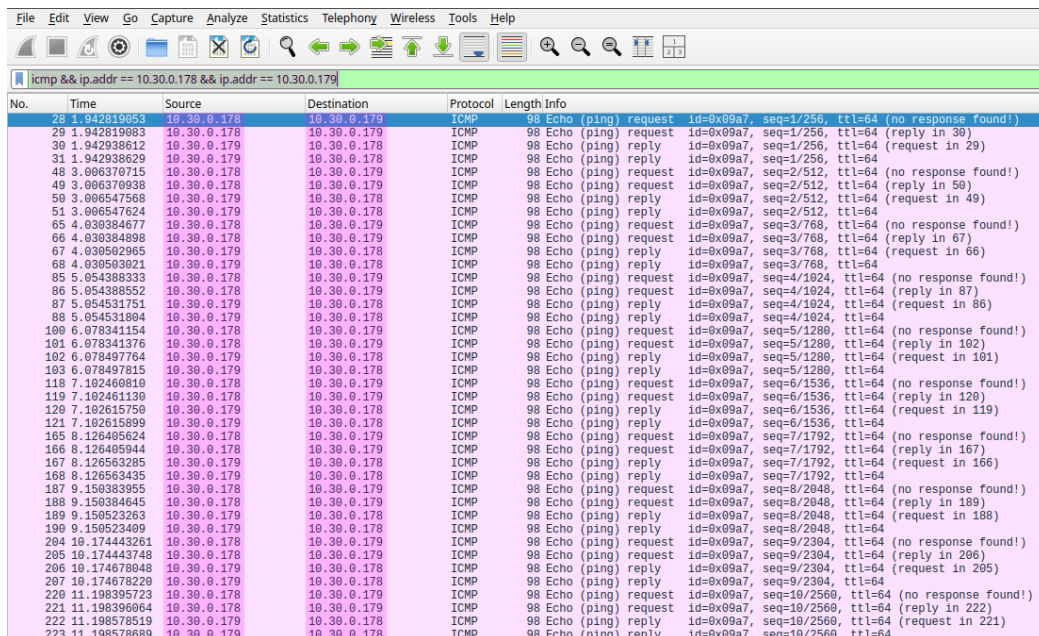
Figure 9: Displaying the full ping between the attacker and a client

4.3 Capturing a Ping Between Two Targets

Since all ingress and egress traffic is being mirrored to the attacker's port, it is possible to observe the entire ICMP exchange between the two victim machines directly from the attacker's PC using Wireshark. If a ping is initiated between the two devices, we can apply the same filter as before—replacing the IP addresses with those of the communicating victims—to capture and analyze the exchanged packets.

```
ip.addr == <Victim1_IP> && ip.addr == <Victim2_IP>
```

As shown in Figure 10, this traffic is visible **only** from the attacker's Wireshark capture. The source and destination fields in the packets correspond to the two victim machines—at no point does the attacker's IP address appear in the captured communication. This interception is possible solely due to port mirroring: all network traffic to and from the mirrored ports is duplicated to the attacker's port. The two clients are unaware of this and communicate normally, while the attacker silently captures their traffic.



No.	Time	Source	Destination	Protocol	Length	Info
28	1.942819653	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=1/256, ttl=64 (no response found!)
29	1.942819683	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=1/256, ttl=64 (reply in 30)
30	1.942938612	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=1/256, ttl=64 (request in 29)
31	1.942938629	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=1/256, ttl=64
48	3.006370715	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=2/512, ttl=64 (no response found!)
49	3.006370938	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=2/512, ttl=64 (reply in 50)
50	3.006547568	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=2/512, ttl=64 (request in 49)
51	3.006547624	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=2/512, ttl=64
65	4.030384677	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=3/768, ttl=64 (no response found!)
66	4.030384898	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=3/768, ttl=64 (reply in 67)
67	4.030502965	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=3/768, ttl=64 (request in 66)
68	4.030503021	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=3/768, ttl=64
85	5.054388333	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=4/1024, ttl=64 (no response found!)
86	5.054388552	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=4/1024, ttl=64 (reply in 87)
87	5.054531751	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=4/1024, ttl=64 (request in 86)
88	5.054531804	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=4/1024, ttl=64
100	6.078341154	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=5/1280, ttl=64 (no response found!)
101	6.078341376	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=5/1280, ttl=64 (reply in 102)
102	6.078497764	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=5/1280, ttl=64 (request in 101)
103	6.078497815	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=5/1280, ttl=64
118	7.102460910	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=6/1536, ttl=64 (no response found!)
119	7.102461130	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=6/1536, ttl=64 (reply in 120)
120	7.102615750	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=6/1536, ttl=64 (request in 119)
121	7.102615899	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=6/1536, ttl=64
165	8.126405624	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=7/1792, ttl=64 (no response found!)
166	8.126405944	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=7/1792, ttl=64 (reply in 167)
167	8.126563285	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=7/1792, ttl=64 (request in 166)
168	8.126563435	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=7/1792, ttl=64
187	9.150383955	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=8/2048, ttl=64 (no response found!)
188	9.150384645	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=8/2048, ttl=64 (reply in 189)
189	9.150523263	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=8/2048, ttl=64 (request in 188)
190	9.150523409	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=8/2048, ttl=64
204	10.174443261	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=9/2304, ttl=64 (no response found!)
205	10.174443748	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=9/2304, ttl=64 (reply in 206)
206	10.174678048	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=9/2304, ttl=64 (request in 205)
207	10.174678220	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=9/2304, ttl=64
220	11.198395723	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=10/2560, ttl=64 (no response found!)
221	11.198396064	10.30.0.178	10.30.0.179	ICMP	98	Echo (ping) request id=0x09a7, seq=10/2560, ttl=64 (reply in 222)
222	11.198578519	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=10/2560, ttl=64 (request in 221)
223	11.198578689	10.30.0.179	10.30.0.178	ICMP	98	Echo (ping) reply id=0x09a7, seq=10/2560, ttl=64

Figure 10: Observing a Ping Between Two Clients That Don't Involve the Attacker

4.4 Capturing Plain Text Passwords

But let's not stop at having two targets ping each other—we can also make use of the web server VM, which is simply the default Nginx page protected with basic authentication. I already explained in detail how to set this up in Exercise 6, under the “Securing Nginx with Basic Authentication” section in my documentation for Exercise 6 [2].

If we make a request to the HTTP server—using either a web browser, `curl`, or any other method—and pass the **Authorization** header, it will contain **Basic**, which is the scheme name, followed by a **Base64**-encoded UTF-8 string of the username and password separated by a colon :, as described in RFC 7617 on pages 4 and 5 [3].

The server then checks whether the provided credentials match an entry in the credentials file. If no match is found, as shown in Figure 11, an HTTP status code 401 **Unauthorized** is returned. [4]

Since the attacker can view the target's traffic, we can examine the provided credentials in the frame that sends the GET request to the server. This frame contains the **Authorization** header, which Wireshark conveniently and automatically decodes from **Base64** to UTF-8, revealing the credentials string in plain text, as shown in Figure 11 below.

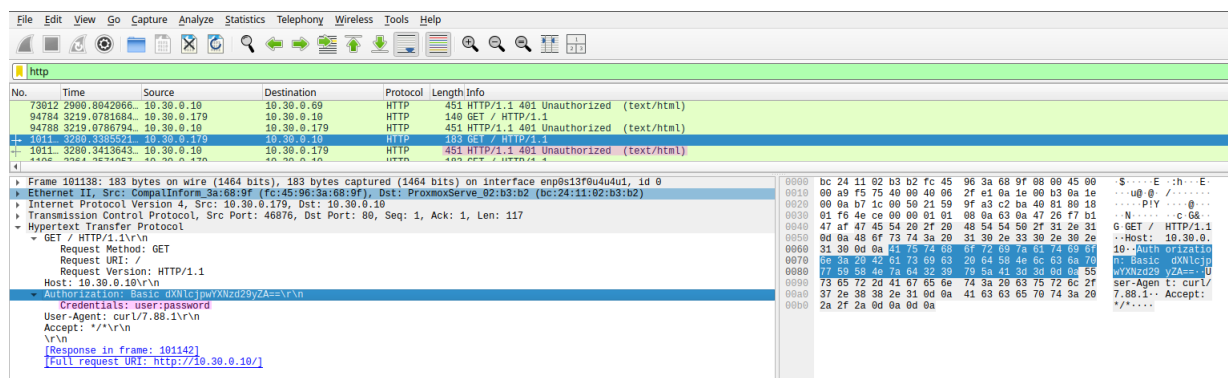


Figure 11: Viewing the plain text password from the authentication

Later, a successful authentication is made, where the server instead returns status code 200, which indicates that the request has succeeded [5].

Again, we can see the credentials used in the request headers and now know that the credentials for this web server are **user3:password123**, as shown in Figure 12. In addition, we receive the entire HTML code returned in the response from the server, which we can also view in plain text—essentially allowing us to see the same content as the client, as shown in Figure 13.

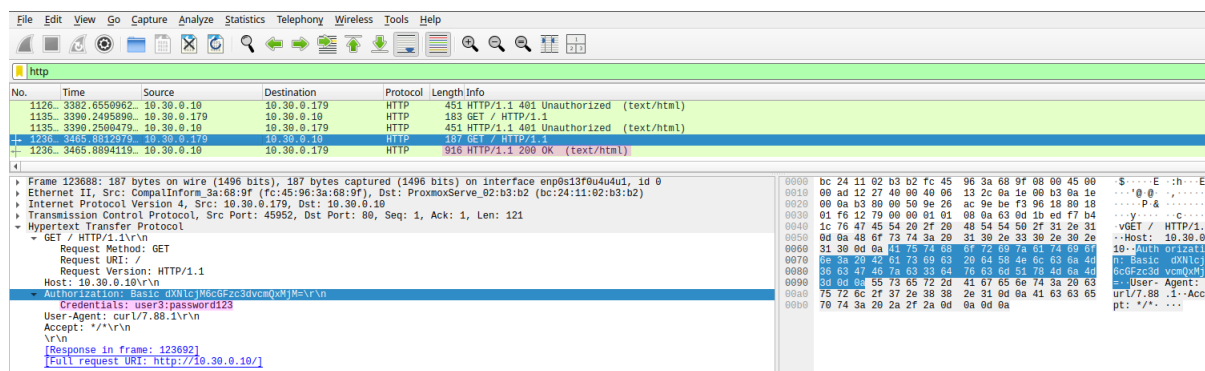


Figure 12: Viewing the correct plain text password from the authentication

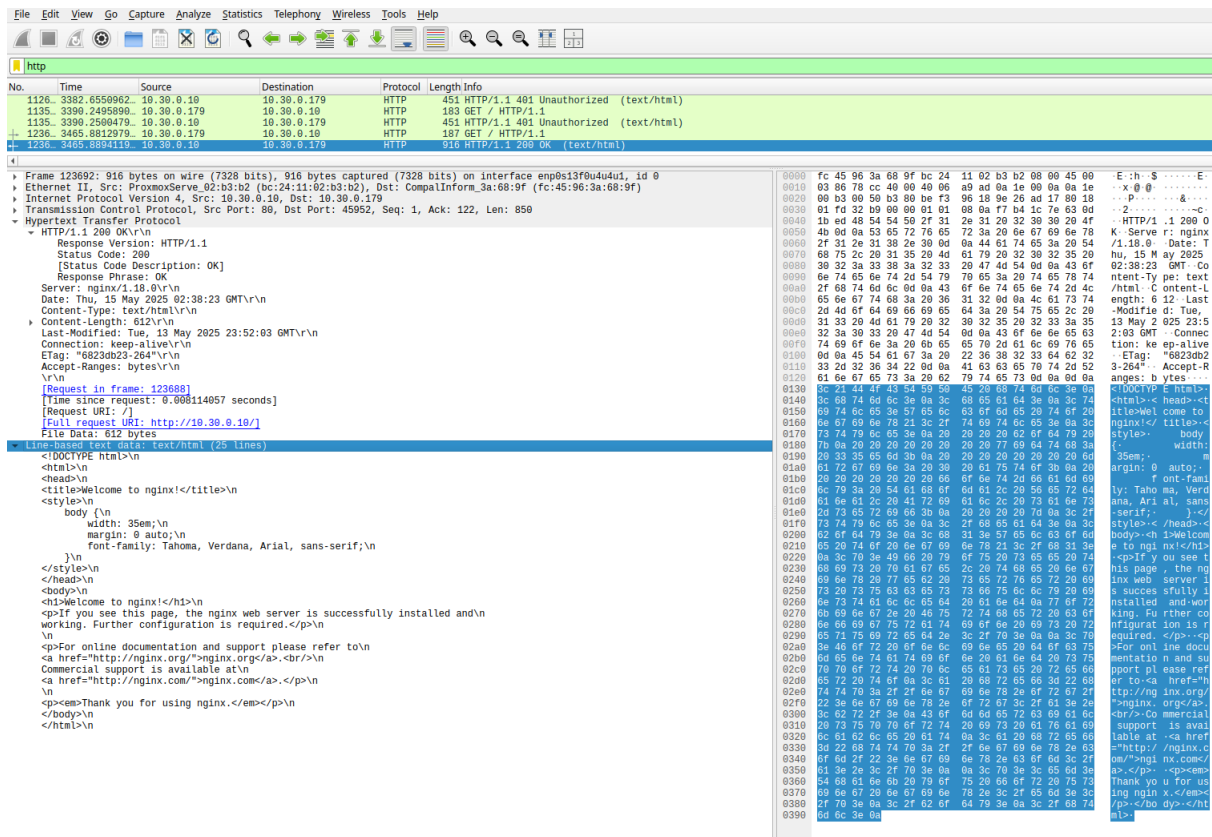


Figure 13: Viewing the returned HTML

4.5 Capturing a VoIP Call

Lastly, VoIP traffic was captured and analyzed using Wireshark. For this, a different topology was used, as shown in Figure 3, since I do not own any VoIP phones. This part of the experiment was conducted in the school's networking lab, where we used a hub and the address range 10.0.0.0/24. The attacker had the address 10.0.0.69, while the two phones had 10.0.0.1 and 10.0.0.2. Since a hub was used, no port mirroring had to be configured.

Voice over IP is an unencrypted protocol that uses the Real-time Transport Protocol (RTP) to transmit application data, which Wireshark has built-in tools to follow and even convert back into audio [6, 7].

Wireshark provides these tools under Telephony → VoIP, which automatically detects the relevant streams and identifies the speakers. In the window that opens (as shown in Figure 14 below), we have several options, such as viewing the Flow Sequence, which shows when the call was ringing and who was speaking when. However, we are more interested in the "Play Streams" button, which displays the waveform of the call (as shown in Figure 15), and allows us to export the audio as an MP3 file [6].

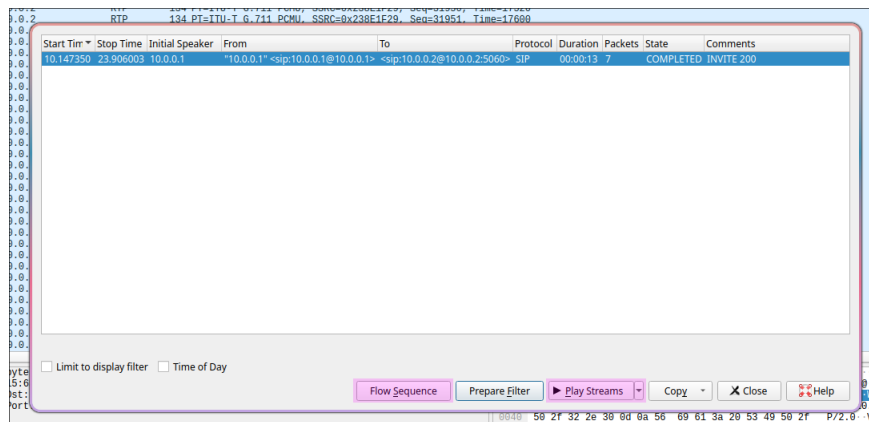


Figure 14: Viewing the VoIP menu in Wireshark

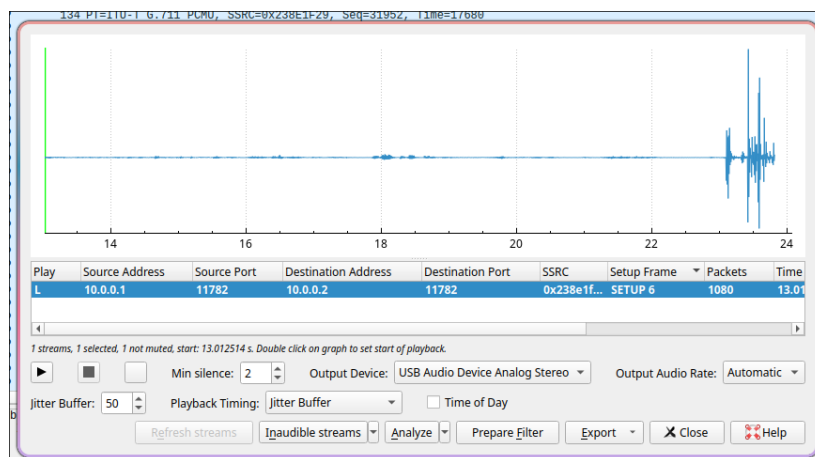


Figure 15: Viewing the Waveform of the call

However, the audio levels of the MP3 were initially unbalanced—the beginning was far too quiet—so I boosted the volume using Audacity, as shown in Figure 16, to normalize the audio levels. I then used the Adobe Podcast AI Audio Enhancer, as shown in Figure 17, to remove background noise and isolate the conversation. The result was a surprisingly clean and understandable audio file, even though the microphone of the other phone was quite far away when I was speaking. I'm actually quite surprised by the quality of the final result.

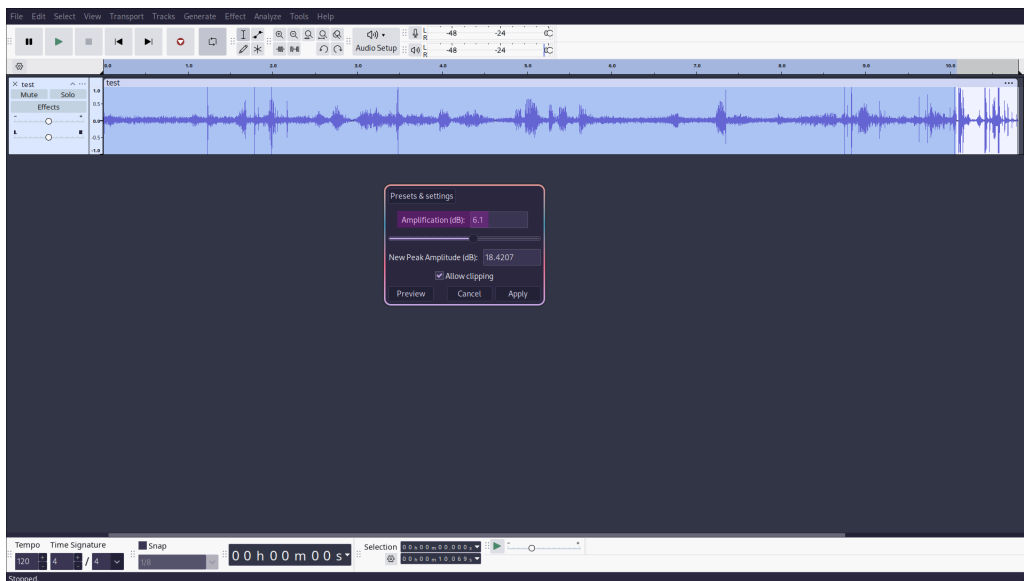


Figure 16: Fixing the Audio Levels in Audacity

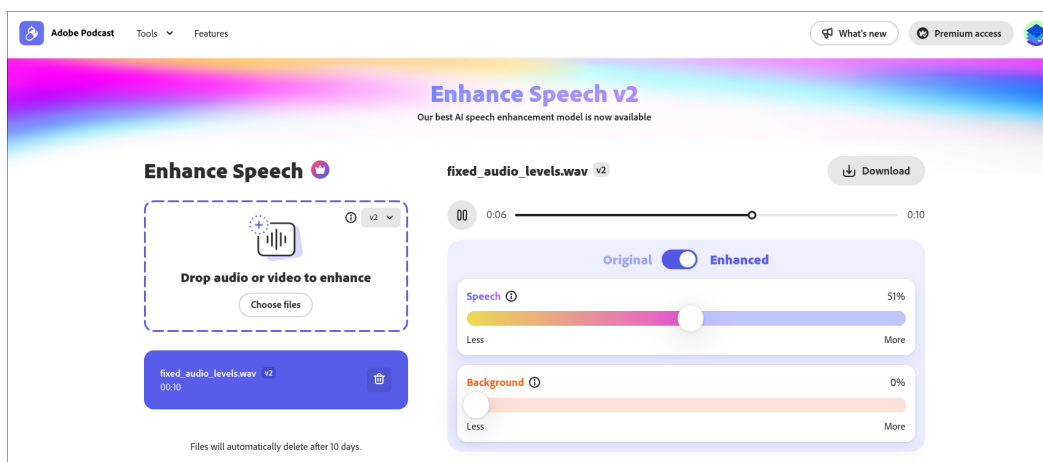


Figure 17: Removing Background Noise Using Adobe's Podcast Tool

5 References

References

- [1] Joey, “Mirroring ports on Mikrotik,” *Blogger*, Oct. 2015. [Online]. Available: <https://www.technicallyinsane.com/2015/10/mirroring-ports-on-mikrotik.html>
- [2] stefi, “Hardening a Linux Webserver,” *silly blog*, Jan. 2025. [Online]. Available: <https://stefanistkuhl.github.io/posts/itsi/year-3/exercise-6/linux-hardening-nginx>
- [3] Oct. 2015, [Online; accessed 16. May 2025]. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc7617.txt>
- [4] “401 Unauthorized - HTTP | MDN,” Mar. 2025, [Online; accessed 16. May 2025]. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status/401>
- [5] “200 OK - HTTP | MDN,” Mar. 2025, [Online; accessed 16. May 2025]. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status/200>
- [6] “VoIP_calls - Wireshark Wiki,” May 2025, [Online; accessed 16. May 2025]. [Online]. Available: https://wiki.wireshark.org/VoIP_calls
- [7] “RTP - Wireshark Wiki,” May 2025, [Online; accessed 16. May 2025]. [Online]. Available: <https://wiki.wireshark.org/RTP>

6 List of figures

List of Figures

1	Grouplogo	1
2	Complete network topology of the exercise using a switch	4
3	Complete network topology of the exercise using a Hub	4
4	Connecting to the Router via Winbox	5
5	Examining the traffic mirror configuration	6
6	Examining the arp traffic without a mirror configuration	7
7	Examining the arp traffic with a mirror configuration	7
8	Displaying the pings to every device on the Network	8
9	Displaying the full ping between the attacker and a client	8
10	Observing a Ping Between Two Clients That Don't Involve the Attacker	9
11	Viewing the plain text password from the authentication	10
12	Viewing the correct plain text password from the authentication	10
13	Viewing the returned HTML	11
14	Viewing the VoIP menu in Wireshark	12
15	Viewing the Waveform of the call	12
16	Fixing the Audio Levels in Audacity	13
17	Removing Background Noise Using Adobe's Podcast Tool	13