

Assignment 7

Analysis of Network Actions on Objectives

April 1, 2024

Table of Contents

EXECUTIVE SUMMARY.....	3
INTRODUCTION.....	4
BODY	4
PASSWORD DICTIONARY ATTACK.....	5
PASSWORD BRUTE FORCE ATTACK.....	13
DEFENCE STRATEGIES	15
CONCLUSION.....	17

Executive Summary

In my investigation of password vulnerabilities using John the Ripper, I focused on understanding the tool's dictionary and brute force capabilities. Understanding and knowing how these methods work is crucial for network security assessment.

I began by conducting a dictionary attack, which leveraged John the Ripper against a set of user accounts with varying password strengths on my Kali Linux machine. By inputting hashed passwords into the program and using the comprehensive **rockyou.txt** wordlist, I uncovered the plaintext of the passwords of the fake users I had created. These passwords ('LAKERS', 'dolphin', and '123abc') were quickly decrypted, which showcased the risks associated with choosing simple, guessable passwords. However, the password for my main account, cipherghost, remained secure against this attack, emphasizing the effectiveness of complex passwords that are not easily found in a typical wordlist.

With the dictionary attack providing valuable insights, I proceeded with a brute force attack, running John the Ripper in incremental mode for 40 hours. Initially, I attempted to crack the password for my main account for 9 hours, utilizing the cache from the dictionary attack. The process did not yield the desired result, highlighting the strength of my password. Following this, I cleared John's cache and reran the brute force attack, focusing on the simplicity of the fake accounts' passwords. After another 31 hours, only the password for fakeuser3 was compromised, reinforcing the notion that while brute force can be effective against basic passwords, it is inefficient and resource-intensive, especially against strong, complex passwords.

The critical findings from this assignment underline the importance of robust password policies. Simple and predictable passwords can be cracked using dictionary attacks. While brute force attacks may eventually succeed against such passwords, they require significant time and computing resources, making them less practical against strong passwords. This emphasizes creating secure passwords to safeguard network security against potential breaches. It is recommended that passwords be complex, unique, and changed regularly to prevent unauthorized access, ensuring the ongoing security of user accounts and sensitive data within a network.

Introduction

In my investigation into the Actions on Objectives phase in network security, my primary objective was to examine the resilience of password security against sophisticated cracking techniques.

This endeavour is vital as it illuminates the strengths and vulnerabilities within network systems. By simulating an attacker's methods, I aimed to understand better how password defences may be compromised and, consequently, how they can be reinforced.

Throughout this analysis, I adhered to ethical guidelines and standards, and my actions were confined to my controlled environment.

I used John the Ripper, known for its efficacy in deciphering passwords. I explored attack vectors using a dictionary and brute force attacks, understanding how each method interacts with different password complexities.

The dictionary attack findings underscored the risks of using guessable, simple passwords, which yield quickly to the wordlist-based strategy. However, my main account's complex password held strong against dictionary attacks.

Shifting to brute force tactics, I observed the limitations of this approach. Despite the extensive use of system resources over many hours, only the simplest password was compromised. This significant time and energy confirm the importance of robust password policies and practices.

In sum, my endeavour into password cracking revealed critical insights into network security. It highlighted the need for strong, unpredictable passwords as a fundamental shield against unauthorized access. These practices are invaluable in shaping security strategies responsive to the evolving landscape of threats.

Body

- **Password Dictionary Attack Summary:** In my test using John the Ripper, I conducted a dictionary attack against user accounts I created, simulating potential social engineering or common password choices. This attack quickly cracked simple passwords like 'LAKERS' and 'dolphin' but failed to decrypt the more complex password of my main

account, demonstrating the vulnerability of basic passwords and the importance of complexity for security.

- **Password Brute Force Attack Summary:** Transitioning from dictionary attacks, I evaluated the effectiveness of brute force attacks. After running John the Ripper in brute force mode for 40 hours, it only managed to crack the simplest password, '123abc', showcasing the limitations of brute force attacks against complex passwords and underscoring the need for robust password creation practices.
- **Defence Strategies Summary:** The defence strategies I outlined focus on preventing password attacks. They involve educating users on strong password creation, using passphrases, implementing regular password changes, enabling 2FA, and adopting tools like password managers. Additionally, instituting account lockout policies, monitoring suspicious activities, and conducting security audits to create a layered defence against potential password attacks.

Password Dictionary Attack

John the Ripper is a well-known tool for cracking passwords. The first part of the assignment focused on the dictionary attack method. Through this method, I aimed to understand the tool's capabilities and the vulnerabilities in common password practices.

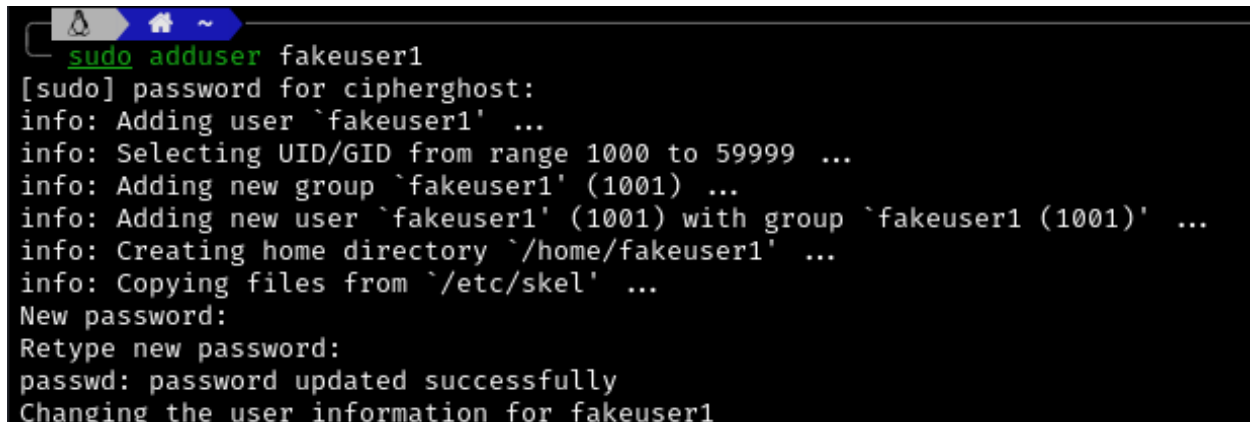
John the Ripper operates by taking hashed passwords as input and applying various techniques to uncover the plaintext. The dictionary attack method uses a list of pre-determined words, known as a wordlist, which contains password choices, including common words and phrases. This approach is predicated on the observation that many users opt for passwords that are easy to remember, making them susceptible to this form of attack.

In executing the attack, I observed how John the Ripper processes each entry from the wordlist, attempting to match it against the encrypted passwords. This process exploits the likelihood of individuals using simple and predictable passwords, leveraging a comprehensive wordlist to increase the chance of a match.

The dictionary attack's effectiveness lies in its simplicity and predictability of human behaviour. By generating hashes from the wordlist entries and comparing them to the target hashes, if a match is found, it indicates the successful cracking of the password. This method revealed to me the importance of choosing robust, complex passwords to mitigate vulnerability to such attacks.

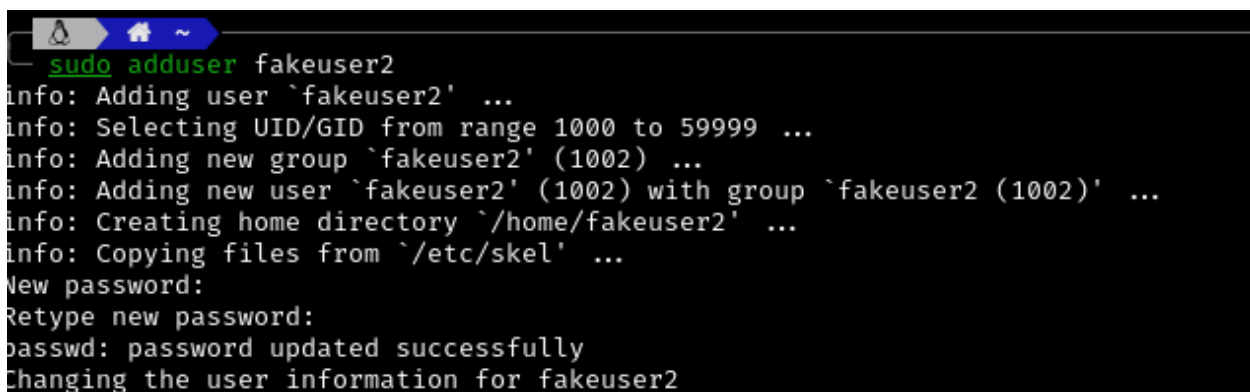
Techniques stressing the perpetual need to enhance password security practices.

In my experiment with John the Ripper on my Kali Linux machine, I crafted user accounts in addition to my main account with passwords that an attacker could guess using social engineering techniques or exploiting commonly used passwords.

A terminal window with a dark background and light-colored text. The prompt is a root symbol followed by a tilde (~). The command 'sudo adduser fakeuser1' is entered. The output shows the process of adding a new user: selecting a UID/GID, adding a new group 'fakeuser1' (1001), adding the user 'fakeuser1' (1001) to that group, creating a home directory '/home/fakeuser1', and copying files from '/etc/skel'. It then prompts for a new password, which is entered and confirmed. The final message is 'Changing the user information for fakeuser1'.

```
~  
[sudo] sudo adduser fakeuser1  
[sudo] password for cipherghost:  
info: Adding user `fakeuser1' ...  
info: Selecting UID/GID from range 1000 to 59999 ...  
info: Adding new group `fakeuser1' (1001) ...  
info: Adding new user `fakeuser1' (1001) with group `fakeuser1 (1001)' ...  
info: Creating home directory `/home/fakeuser1' ...  
info: Copying files from `/etc/skel' ...  
New password:  
Retype new password:  
passwd: password updated successfully  
Changing the user information for fakeuser1
```

For **fakeuser1**, I set the password to 'LAKERS,' aligning with the user's interest in a particular sports team. This kind of password is often vulnerable to dictionary attacks, as it is based on a noun easily associated with the user, making it a prime candidate for a wordlist compiled with social engineering insights in mind.

A terminal window with a dark background and light-colored text. The prompt is a root symbol followed by a tilde (~). The command 'sudo adduser fakeuser2' is entered. The output shows the process of adding a new user: selecting a UID/GID, adding a new group 'fakeuser2' (1002), adding the user 'fakeuser2' (1002) to that group, creating a home directory '/home/fakeuser2', and copying files from '/etc/skel'. It then prompts for a new password, which is entered and confirmed. The final message is 'Changing the user information for fakeuser2'.

```
~  
[sudo] sudo adduser fakeuser2  
info: Adding user `fakeuser2' ...  
info: Selecting UID/GID from range 1000 to 59999 ...  
info: Adding new group `fakeuser2' (1002) ...  
info: Adding new user `fakeuser2' (1002) with group `fakeuser2 (1002)' ...  
info: Creating home directory `/home/fakeuser2' ...  
info: Copying files from `/etc/skel' ...  
New password:  
Retype new password:  
passwd: password updated successfully  
Changing the user information for fakeuser2
```

Continuing with the theme of personalized passwords, I chose 'dolphin' for **fakeuser2**, which could be linked to the user's favourite animal, hobby, or pet name. Passwords like these are commonly used because they're memorable and have personal significance, but they also present security risks, as such words are typically included in comprehensive dictionary lists.

```
~  
sudo adduser fakeuser3  
info: Adding user `fakeuser3' ...  
info: Selecting UID/GID from range 1000 to 59999 ...  
info: Adding new group `fakeuser3' (1003) ...  
info: Adding new user `fakeuser3' (1003) with group `fakeuser3 (1003)' ...  
info: Creating home directory `/home/fakeuser3' ...  
info: Copying files from `/etc/skel' ...  
New password:  
Retype new password:  
passwd: password updated successfully  
Changing the user information for fakeuser3
```

For **fakeuser3**, I diverged from interest-based passwords and opted for '123abc', a password that, while seemingly more random, falls into the category of common and easily guessable patterns. This choice intentionally demonstrated how John the Ripper could handle passwords that are not necessarily dictionary words but are still weak due to their use of predictable sequences.

The chosen passwords were intended to reflect the range of common password strategies users might deploy, from the easily guessable to the seemingly random. This gave me a clear field to assess the tool's effectiveness in my environment.

```

sudo cat /etc/passwd
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/usr/sbin/nologin
_galera:x:100:65534::/nonexistent:/usr/sbin/nologin
mysql:x:101:102:MariaDB Server,,:/nonexistent:/bin/false
tss:x:102:103:TPM software stack,,:/var/lib/tpm:/bin/false
strongswan:x:103:65534::/var/lib/strongswan:/usr/sbin/nologin
systemd-timesync:x:992:992:systemd Time Synchronization:/usr/sbin/nologin
redsocks:x:104:105::/var/run/redsocks:/usr/sbin/nologin
rwhod:x:105:65534::/var/spool/rwho:/usr/sbin/nologin
_gophish:x:106:107::/var/lib/gophish:/usr/sbin/nologin
iodine:x:107:65534::/run/iodine:/usr/sbin/nologin
messagebus:x:108:108::/nonexistent:/usr/sbin/nologin
tcpdump:x:109:109::/nonexistent:/usr/sbin/nologin
miredo:x:110:65534::/var/run/miredo:/usr/sbin/nologin
redis:x:111:112::/var/lib/redis:/usr/sbin/nologin
usbmux:x:112:46:usbmux daemon,,:/var/lib/usbmux:/usr/sbin/nologin
mosquitto:x:113:114::/var/lib/mosquitto:/usr/sbin/nologin
sshd:x:114:65534::/run/sshd:/usr/sbin/nologin
_rpc:x:115:65534::/run/rpcbind:/usr/sbin/nologin
dnsmasq:x:116:65534:dnsmasq,,:/var/lib/misc:/usr/sbin/nologin
statd:x:117:65534::/var/lib/nfs:/usr/sbin/nologin
avahi:x:118:118:Avahi mDNS daemon,,:/run/avahi-daemon:/usr/sbin/nologin
stunnel4:x:991:991:stunnel service system account:/var/run/stunnel4:/usr/sbin/nologin
Debian-snmp:x:119:119::/var/lib/snmp:/bin/false
_gvm:x:120:120::/var/lib/ovnas:/usr/sbin/nologin
speech-dispatcher:x:121:29:Speech Dispatcher,,:/run/speech-dispatcher:/bin/false
sslm:x:122:121::/nonexistent:/usr/sbin/nologin
postgres:x:123:122:PostgreSQL administrator,,:/var/lib/postgresql:/bin/bash
pulse:x:124:124:PulseAudio daemon,,:/run/pulse:/usr/sbin/nologin
inetsim:x:125:126::/var/lib/inetsim:/usr/sbin/nologin
lightdm:x:126:127:Light Display Manager:/var/lib/lightdm:/bin/false
geoclue:x:127:128::/var/lib/geoclue:/usr/sbin/nologin
saned:x:128:130::/var/lib/saned:/usr/sbin/nologin
polkitd:x:989:989:polkit:/nonexistent:/usr/sbin/nologin
rtkit:x:129:131:RealtimeKit,,:/proc:/usr/sbin/nologin
colord:x:130:132:colord colour management daemon,,:/var/lib/colord:/usr/sbin/nologin
nm-openvpn:x:131:133:NetworkManager OpenVPN,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
nm-openconnect:x:132:134:NetworkManager OpenConnect plugin,,:/var/lib/NetworkManager:/usr/sbin/nologin
cipherghost:x:1000:1000:Mr.CipherGhost,,:/home/cipherghost:/usr/bin/zsh
fakeuser1:x:1001:1001:Fake User1,,:/home/fakeuser1:/bin/bash
fakeuser2:x:1002:1002:Fake User 2,,:/home/fakeuser2:/bin/bash
fakeuser3:x:1003:1003::/home/fakeuser3:/bin/bash

```

After setting up the user accounts with their respective passwords, I proceeded to the next step of my dictionary attack: obtaining the hashed versions of these passwords, as they would typically be stored in a system's password file. Using my Kali Linux machine, I used the command **sudo cat /etc/passwd** to display the contents of the **passwd** file, which contains user account information, including the location of each user's home directory and shell information. This file, however, does not contain the actual password hashes; it only indicates if the user has a hashed password set with the “x” after the username.

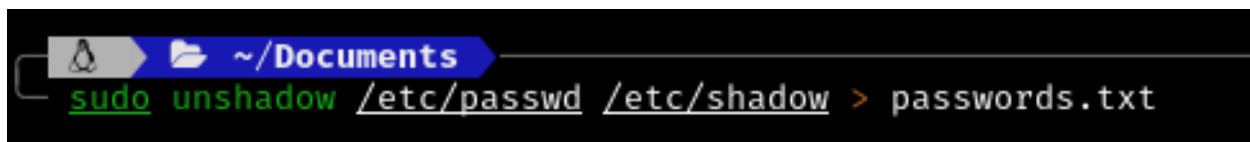

```

sudo cat /etc/shadow
root:!:19775:0:99999:7:::
daemon*:19775:0:99999:7:::
bin*:19775:0:99999:7:::
sys*:19775:0:99999:7:::
sync*:19775:0:99999:7:::
games*:19775:0:99999:7:::
man*:19775:0:99999:7:::
lp*:19775:0:99999:7:::
mail*:19775:0:99999:7:::
news*:19775:0:99999:7:::
uucp*:19775:0:99999:7:::
proxy*:19775:0:99999:7:::
www-data*:19775:0:99999:7:::
backup*:19775:0:99999:7:::
list*:19775:0:99999:7:::
irc*:19775:0:99999:7:::
_apt*:19775:0:99999:7:::
nobody*:19775:0:99999:7:::
systemd-networkd:!:19775:0:99999:7:::
_galera:!:19775:0:99999:7:::
mysql:!:19775:0:99999:7:::
ssss:!:19775:0:99999:7:::
strongswan:!:19775:0:99999:7:::
systemd-timesyncd:!:19775:0:99999:7:::
redsocks:!:19775:0:99999:7:::
rwhod:!:19775:0:99999:7:::
_gophish:!:19775:0:99999:7:::
iodine:!:19775:0:99999:7:::
messagebus:!:19775:0:99999:7:::
tcpdump:!:19775:0:99999:7:::
niredo:!:19775:0:99999:7:::
redis:!:19775:0:99999:7:::
jsbmux:!:19775:0:99999:7:::
mosquitto:!:19775:0:99999:7:::
sshd:!:19775:0:99999:7:::
_rpc:!:19775:0:99999:7:::
dnsmasq:!:19775:0:99999:7:::
statd:!:19775:0:99999:7:::
avahi:!:19775:0:99999:7:::
stunnel4:!:19775:0:99999:7:::
Debian-snmpp:!:19775:0:99999:7:::
_gvm:!:19775:0:99999:7:::
speech-dispatcher:!:19775:0:99999:7:::
ssllh:!:19775:0:99999:7:::
postgres:!:19775:0:99999:7:::
pulse:!:19775:0:99999:7:::
inetsim:!:19775:0:99999:7:::
lightdm:!:19775:0:99999:7:::
geoclue:!:19775:0:99999:7:::
saned:!:19775:0:99999:7:::
bolkitd:!:19775:0:99999:7:::
rtkit:!:19775:0:99999:7:::
colord:!:19775:0:99999:7:::
nm-openvpn:!:19775:0:99999:7:::
nm-openconnect:!:19775:0:99999:7:::
cipherghost:
fakeuser1:$y$j9T$VtV.02b0WY9xWPwbCao0/0$yvpH/.faOp4n9lkHUnTuwtENK/xPL410As/WKX1lE.C:19813:0:99999:7:::
fakeuser2:$y$j9T$xxKs0l75u9zunQk0TAMNL1$aZ0BrBhMrM3BOUzfhlTRjiD94kSZGW/LOhAZkKRpF32:19813:0:99999:7:::
fakeuser3:$y$j9T$HNdQ0MElfx3PscRmy8edz1$VvCzCzg1zssB0L6iHumqTj0bpvb2UuNCHMaFiWwIZ58:19813:0:99999:7:::

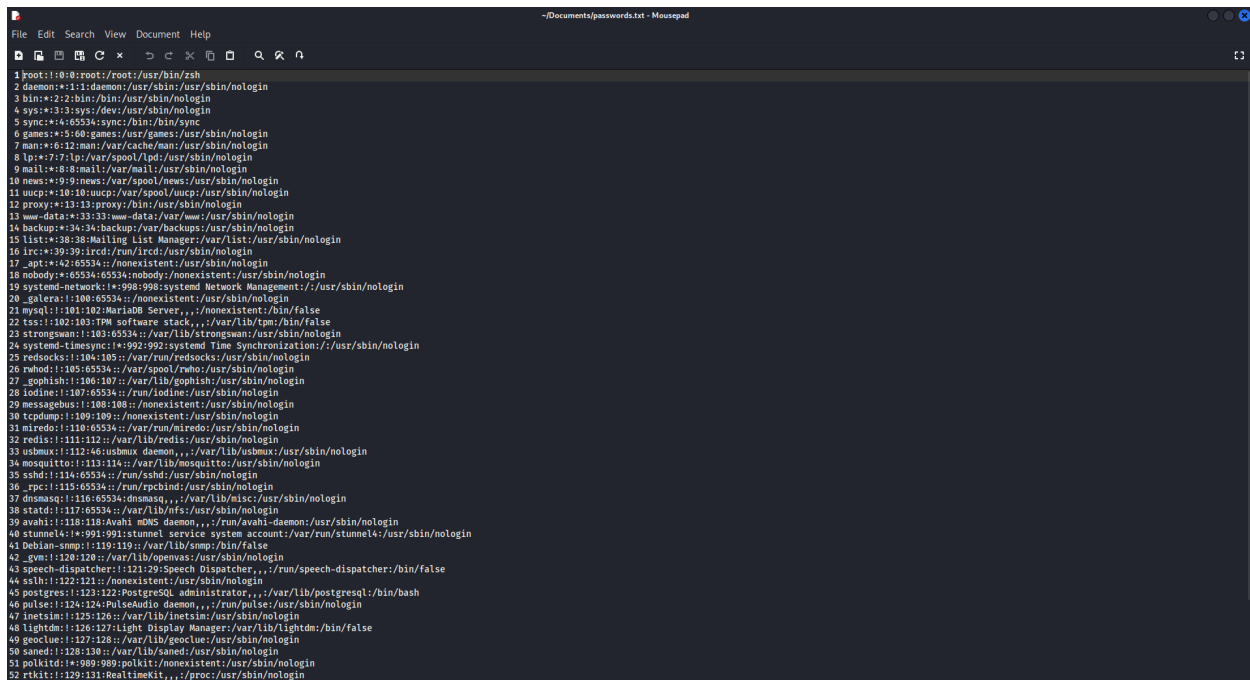
```

To get the actual password hashes, I ran **sudo cat /etc/shadow**, which is the file that stores the secure user account information, including the password hashes. This file is accessible with root or sudo privileges, ensuring unauthorized users cannot access sensitive password information. The shadow file displays each user's login name followed by a hashed password, the number of days since the password was last changed, and other settings.

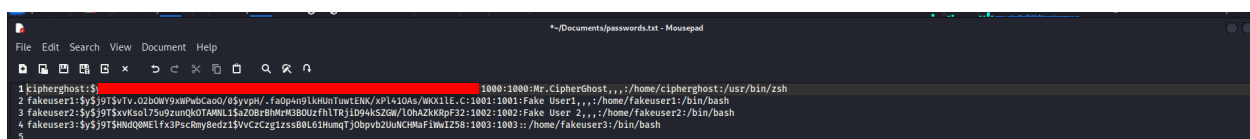
To prepare for the dictionary attack with John the Ripper, I needed to combine the information from the **passwd** and **shadow** files to create a format that John could work with effectively. I accomplished this by using the **unshadow** command. This command merges the two files, **passwd** and **shadow**, resulting in a single file containing the user information and the corresponding password hashes.



I executed the command **sudo unshadow /etc/passwd /etc/shadow > passwords.txt**, directing the output to a file named **passwords.txt**. This file now contained the necessary data for John the Ripper to perform its cracking attempts.



Inspecting the contents of **passwords.txt** with a text editor, I confirmed the presence of the user accounts along with their password hashes.



I decided to remove the information I was not interested in and kept the four user accounts and their password at the bottom of the text file. The first account is my user account with my own hashed password, and the other three accounts are the ones that I created at the beginning of this section.

I had the passwords.txt file ready with the hashed passwords. It set the stage for the actual dictionary attack, where I would be using John the Ripper's capabilities to match these hashes against a wordlist. This preparation mirrored what an attacker might do after obtaining access to

such sensitive files, highlighting the importance of secure password policies and encrypted storage of user data.

With the **passwords.txt** file containing the combined user and hash information, I was ready to launch John the Ripper for the dictionary attack. I began by opening my terminal in the directory where **passwords.txt** was located and typed the following command:

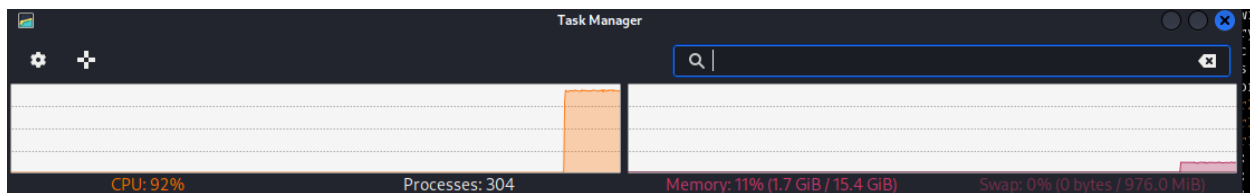
```
~ /Documents
sudo john --format=crypt --wordlist=/usr/share/wordlists/rockyou.txt passwords.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (crypt, generic crypt(3) [?/64])
Cost 1 (algorithm [1:descript 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 0 for all loaded hashes
Cost 2 (algorithm specific iterations) is 1 for all loaded hashes
Will run 16 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
dolphin (fakeuser2)
123abc (fakeuser3)
LAKERS (fakeuser1)
3g 0:01:26:47 12.45% (ETA: 03:23:36) 0.000576g/s 377.6p/s 381.3c/s 381.3C/s braden1234..brad13!
3g 0:01:30:37 13.03% (ETA: 03:22:08) 0.000551g/s 377.6p/s 381.2c/s 381.2C/s almondeyes..almencion
3g 0:01:42:03 14.57% (ETA: 03:27:11) 0.000489g/s 376.5p/s 379.7c/s 379.7C/s 25sassy..25girlie
3g 0:02:43:24 24.58% (ETA: 02:51:19) 0.000305g/s 378.5p/s 380.5c/s 380.5C/s skuize..skubba02
3g 0:02:51:41 25.97% (ETA: 02:47:37) 0.000291g/s 378.6p/s 380.6c/s 380.6C/s serajm1002..serafina29
3g 0:03:03:56 27.98% (ETA: 02:43:55) 0.000271g/s 378.9p/s 380.7c/s 380.7C/s romeu26..romerolili
3g 0:03:38:49 33.73% (ETA: 02:35:18) 0.000228g/s 379.6p/s 381.1c/s 381.1C/s numeros1:6..numeras07
3g 0:04:48:00 45.11% (ETA: 02:25:00) 0.000173g/s 380.4p/s 381.5c/s 381.5C/s killyoselfbitch..killusall
3g 0:07:02:42 67.38% (ETA: 02:13:55) 0.000118g/s 381.3p/s 382.1c/s 382.1C/s betlen..betitoboquita
3g 0:07:03:03 67.45% (ETA: 02:13:53) 0.000118g/s 381.3p/s 382.1c/s 382.1C/s beryss2..berwynne
Use the "--show" option to display all of the cracked passwords reliably
Session aborted
```

```
sudo john --format=crypt --wordlist=/usr/share/wordlists/rockyou.txt passwords.txt
```

This command was crafted with:

- **sudo** ensured root privileges, granting John the Ripper the necessary system permissions.
- **john** is the executable for John the Ripper.
- **--format=crypt** specified the expected format of the hashes, which is essential to apply the correct algorithm and optimize the cracking process.
- **--wordlist** pointed to the **rockyou.txt** wordlist, a comprehensive compilation of passwords exposed by real-world breaches. I chose it as it reflects a realistic array of commonly used passwords and is thus likely to be guessed.
- **passwords.txt** was the input file containing the password hashes to be tested.

The term "g/s" in the terminal output means "guesses per second." The rate at which the program makes guesses varies over time but averages around 378 guesses per second, as indicated by numbers like "377.6p/s". This means that the program attempts that many passwords every second to try and find the correct ones.



As I observed through the Task Manager, the command utilized extensive CPU resources while keeping memory usage modest. The CPU usage indicated the intense computational effort involved in the process.

The output in the terminal listed the passwords for **fakeuser1**, **fakeuser2**, and **fakeuser3** in less than 20 minutes, which were 'LAKERS', 'dolphin', and '123abc', respectively. These passwords were cracked almost instantaneously, highlighting their vulnerability and the power of dictionary attacks against weak passwords.

Notably, I ran the dictionary attack for **7 hours**, and the password for my main account, **cipherghost**, remained uncracked. This outcome showed that although dictionary attacks are powerful, they are not dependable. They might not have the password in their wordlist, which will cause them to waste time and resources. A strong, complex password can still withstand such an attack, reinforcing the value of good password practices.

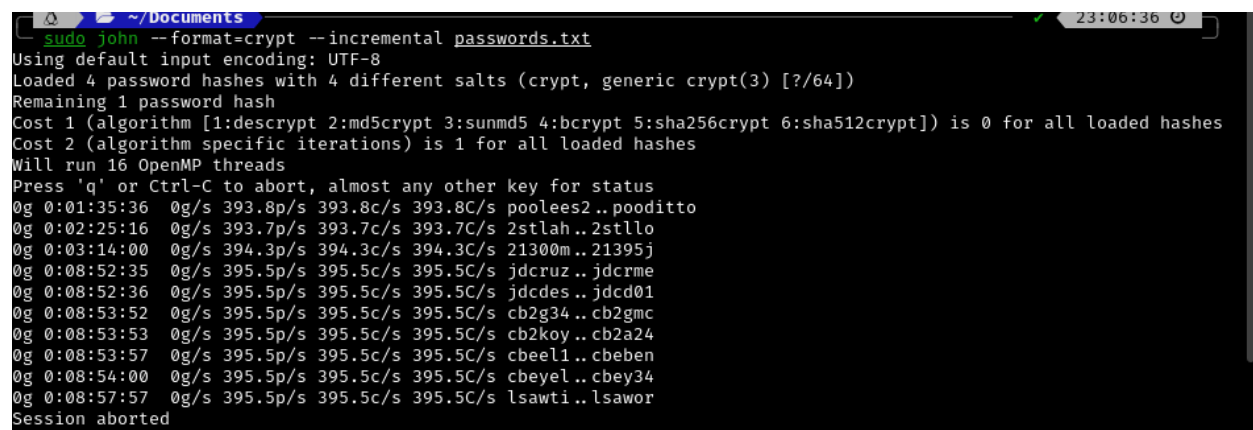
Based on my research, I have found that the **crypt** function is a standard Unix-based encryption algorithm. By specifying it, I directed John the Ripper to use the appropriate method for decrypting the hashes.

The **rockyou.txt** wordlist was selected due to its extensive nature and real-world application. It includes a wide array of common passwords, which challenges the security of user accounts. The resistance of the **cipherghost (my main account)** password against the dictionary attack is proof of the efficiency of complex passwords. It underscores the need for strong password policies, including combining uppercase and lowercase letters, numbers, and symbols and avoiding predictable patterns. This attack demonstrated the capabilities of John the Ripper and the importance of secure password creation to protect against unauthorized access.

Password Brute Force Attack

Brute-force attacks are often considered the last step because they are time-consuming. These attacks systematically attempt every possible combination of characters until the correct password is found. This method theoretically can crack any password if it has enough time and resources.

With this understanding, I tested John the Ripper's brute force capabilities on my Kali Linux system. My target was the password for my main account, alongside those of the simpler passwords of the fake user accounts created earlier.

A terminal window showing the execution of John the Ripper. The command 'sudo john --format=crypt --incremental passwords.txt' is entered. The output shows the program loading 4 password hashes with 4 different salts, using a crypt(3) format. It then displays a list of hashes being cracked, including 'poollees2..pooditto', '2stlah..2stllo', '21300m..21395j', 'jdcruz..jdcme', 'jdcdes..jdc01', 'cb2g34..cb2gmc', 'cb2koy..cb2a24', 'cbeel1..cbeben', 'cbeyel..cbey34', and 'lsawti..lsawor'. The program is running at approximately 395 guesses per second. The session is aborted.

```
~/.Documents
sudo john --format=crypt --incremental passwords.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (crypt, generic crypt(3) [?/64])
Remaining 1 password hash
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 0 for all loaded hashes
Cost 2 (algorithm specific iterations) is 1 for all loaded hashes
Will run 16 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:01:35:36 0g/s 393.8p/s 393.8c/s 393.8C/s poollees2..pooditto
0g 0:02:25:16 0g/s 393.7p/s 393.7c/s 393.7C/s 2stlah..2stllo
0g 0:03:14:00 0g/s 394.3p/s 394.3c/s 394.3C/s 21300m..21395j
0g 0:08:52:35 0g/s 395.5p/s 395.5c/s 395.5C/s jdcruz..jdcme
0g 0:08:52:36 0g/s 395.5p/s 395.5c/s 395.5C/s jdcdes..jdc01
0g 0:08:53:52 0g/s 395.5p/s 395.5c/s 395.5C/s cb2g34..cb2gmc
0g 0:08:53:53 0g/s 395.5p/s 395.5c/s 395.5C/s cb2koy..cb2a24
0g 0:08:53:57 0g/s 395.5p/s 395.5c/s 395.5C/s cbeel1..cbeben
0g 0:08:54:00 0g/s 395.5p/s 395.5c/s 395.5C/s cbeyel..cbey34
0g 0:08:57:57 0g/s 395.5p/s 395.5c/s 395.5C/s lsawti..lsawor
Session aborted
```

I intended to assess how the tool would perform against the password of my main account, **cipherghost**, and the simpler passwords of the fake user accounts created earlier.

I initiated the brute force attack by running the following command for approximately **9 hours**:

```
sudo john --format=crypt --incremental passwords.txt
```

Here, **--incremental** is the option that instructs John The Ripper to use brute force, trying all possible combinations of characters to crack the passwords. I used the same **passwords.txt** file from the dictionary attack as the input, containing the hashes for my main and fake user accounts.

The term "g/s" in the terminal output means "guesses per second." The rate at which the program makes guesses varies over time but averages around 395 guesses per second, as indicated by numbers like "395.5p/s". This means that the program attempts that many passwords every second to try and find the correct ones.

```
~  
sudo rm /root/.john/john.pot
```

After an extensive period without success in cracking the **cipherghost** password, I decided to clear John's cache. The passwords for the three fake accounts were stored in the cache from the dictionary attack. This was achieved by removing the **.pot** file, where John stores the cracked passwords.

Resetting the cache allowed me to conduct a fresh brute-force attack without any prior findings potentially affecting the process. This was useful for determining John's efficiency with shorter and simpler passwords (the three fake user accounts).

```
~/Documents  
sudo john --format=crypt --incremental passwords.txt  
[sudo] password for cipherghost:  
Using default input encoding: UTF-8  
Loaded 4 password hashes with 4 different salts (crypt, generic crypt(3) [?/64])  
Cost 1 (algorithm [1:descript 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 0 for all loaded hashes  
Cost 2 (algorithm specific iterations) is 1 for all loaded hashes  
Will run 16 OpenMP threads  
Press 'q' or Ctrl-C to abort, almost any other key for status  
123abc (fakeuser3)  
1g 0:00:01:36 0.01039g/s 97.76p/s 393.0c/s 393.0C/s 120607..110391  
1g 0:04:18:44 0.000064g/s 131.2p/s 394.4c/s 394.4C/s jarlovy..jarlyn2  
1g 0:06:06:48 0.000045g/s 128.8p/s 387.0c/s 387.0C/s sariang12..shockento  
1g 0:06:07:36 0.000045g/s 128.8p/s 387.0c/s 387.0C/s 19jrow..19jt18  
1g 0:06:07:43 0.000045g/s 128.8p/s 387.0c/s 387.0C/s 14m181..14m459  
1g 0:11:38:24 0.000023g/s 125.8p/s 377.6c/s 377.6C/s glalz..gljkj  
1g 0:12:02:14 0.000023g/s 125.7p/s 377.3c/s 377.3C/s pr41051..pr41530  
1g 0:14:41:54 0.000018g/s 125.1p/s 375.7c/s 375.7C/s mslinc..mslskb  
1g 0:15:48:46 0.000017g/s 124.9p/s 375.1c/s 375.1C/s hrvon..hrv 1  
1g 0:18:13:55 0.000015g/s 124.6p/s 374.2c/s 374.2C/s dobieb..dobsak  
1g 0:23:09:08 0.000011g/s 124.3p/s 373.0c/s 373.0C/s cjcsey..cj0v3s  
1g 1:04:45:15 0.000009g/s 124.0p/s 372.1c/s 372.1C/s p0pic1..p0piki  
1g 1:06:56:39 0.000008g/s 123.9p/s 371.8c/s 371.8C/s dwsd10..dwsdue  
1g 1:07:14:12 0.000008g/s 123.9p/s 371.7c/s 371.7C/s kn0MPI..kn0nic  
1g 1:07:14:14 0.000008g/s 123.9p/s 371.7c/s 371.7C/s kn0not..kc6904  
Use the "--show" option to display all of the cracked passwords reliably  
Session aborted
```

With the cache cleared, I ran John in brute force mode again. This time, after about a minute, it cracked the password '123abc' for **fakeuser3**, but it could not crack any of the other 3 passwords during the **31 hours** of running. The successful cracking of **fakeuser3**'s password reaffirmed the vulnerability of ordered and simple, common passwords to brute-force attacks.

Here, the program's guessing rate is averaged around 125 guesses per second.

In total, I invested **40 (9 + 31) hours** into brute force attacks. This substantial time commitment and the considerable amount of power consumed emphasize the impracticality of brute-force methods for cracking well-constructed passwords. The outcome contrast between the unsuccessful attempt against three accounts, including my secure cipherghost account, and the successful crack of fakeuser3's simplistic password demonstrated the variable efficacy of brute

force attacks. It emphasizes the need for stronger, more complex passwords to provide a barrier against brute force attacks, preserving the integrity of user data in the face of persistent threats.

Defence Strategies

Complexity and Variation: My main account's resilience against dictionary and brute-force attacks reinforces the necessity of complex passwords. Passwords should be a mix of uppercase and lowercase letters, numbers, and symbols and long enough to deter brute-force attacks due to the increased time required to crack them.

Regular Password Changes: Regular password changes can help mitigate the risks posed by potential data breaches. If a password is compromised but changed frequently, the chance for unauthorized access by an attacker is reduced.

Use of Passphrases: Instead of traditional passwords, passphrases (longer strings of words or sentences) can provide both memorability and security. A unique and random combination of words can be harder to predict and, therefore, to crack.

Two-Factor Authentication (2FA): Enabling 2FA provides an additional layer of security. Even if a password is compromised, the attacker still requires a second verification form to gain access.

Educating Users: Users must be educated about the importance of password security. This includes guidance on creating strong passwords, understanding the risks of using personal information or common words, and not reusing passwords across different services.

Avoiding Predictable Patterns: Passwords containing personal information, such as birthdays, anniversaries, or names, are more vulnerable. Users should be discouraged from using any predictable or easily researched information in their passwords.

Password Managers: Encouraging the use of reputable password managers can help users generate and store complex passwords, reducing the temptation to reuse passwords or create simple ones. Password managers can also help to mitigate the risk of physical device attacks such as keyloggers.

Account Lockout Policies: Implementing account lockout policies can mitigate brute force attacks by locking a user account after a certain number of failed login attempts. However, we should handle and avoid denial of service situations.

Monitoring and Alerts: Setting up systems that monitor and send alerts for unusual login attempts can help detect potential attacks early.

Security Audits and Penetration Testing: Regularly auditing systems for weak passwords and conducting penetration testing can help identify and mitigate risks proactively.

Conclusion

In understanding the Actions on Objectives phase in network security, my primary focus was testing password vulnerabilities using the John the Ripper tool. My objective was to measure passwords' vulnerability to a dictionary attack against the rockyou wordlist and evaluate their resilience against a brute force attack, which attempts to decode passwords through a trial-and-error method that systematically checks all possible combinations.

Password Dictionary Attack Findings:

During the dictionary attack phase, I learned that even secure passwords can be surprisingly weak if they are predictable or common. The simplicity of John the Ripper's operation, combined with the extensive 'rockyou.txt' wordlist, led to the rapid decryption of passwords for the user accounts 'fakeuser1', 'fakeuser2', and 'fakeuser3'. The contrast in the security outcomes between these accounts and my main account, 'cipherghost', was interesting. While the simpler passwords were easy to decrypt, my main account's password was not decrypted over the 7-hour attempt. This part of the assignment illustrates how attackers exploit passwords and highlights the importance of randomness and complexity in password creation to mitigate such threats.

Password Brute Force Attack Learnings:

Transitioning to brute force attacks, I realized the high computational task usage. Despite employing John the Ripper over an extensive period, the effort yielded only a single success, cracking the password '123abc' associated with 'fakeuser3'. This result underscored the inefficiency of brute force attacks against well-crafted passwords, as it took considerable time and computing power to achieve even one breakthrough.

Actionable Recommendations:

From these experiments, I can make several actionable recommendations to strengthen network security:

1. Passwords should avoid common words, phrases, or personal information. They should include a mix of characters, numbers, and symbols and be of a length that dissuades brute force attempts.
2. Regular training sessions for users on creating strong passwords and recognizing social engineering tactics can improve security.
3. Implementing MFA can be an essential barrier, even if passwords are compromised.

4. Utilizing reputable password managers can help users maintain strong passwords without the risk of forgetting them.
5. Encouraging or enforcing regular password changes can minimize the window of opportunity for an attacker if a breach does occur.
6. Employing account lockout policies after a certain number of incorrect attempts can prevent brute force attacks, and setting up monitoring systems for unusual login attempts can provide early warning signs of a breach.
7. Conducting periodic security audits and penetration testing to identify and strengthen weak points.

In conclusion, this assignment confirmed that password robustness is essential for network security. The difference in attack outcomes demonstrates the necessity of adopting multi-layered defence strategies to mitigate potential breaches. By integrating these defensive tactics, we can secure our networks against the password attacks.