Report on Brute-Force Attack Simulation and Defensive Measures

Introduction

This report summarizes the results of a brute-force attack simulation using John the Ripper, outlines the defensive measures applied, and explains the importance of strong passwords and encryption in preventing such attacks. The aim of the exercise was to better understand the vulnerabilities associated with weak password protection and how attackers exploit these weaknesses using brute-force methods. Screenshots of the process are attached as evidence of the attack and defense phases.

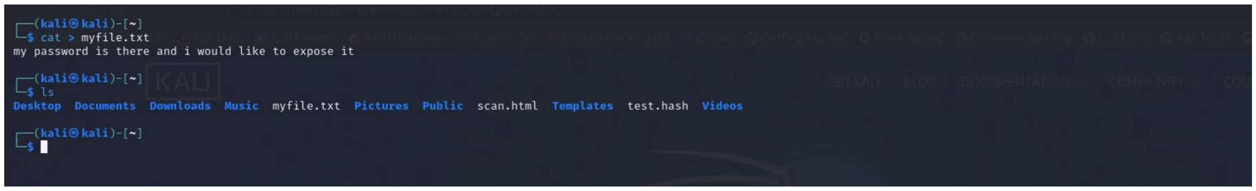
Brute-Force Attack Simulation

Step 1: Creating the myfile.txt File

I first created a simple text file named myfile.txt that would be used to demonstrate the process of password-protecting a zip file and subsequently performing a brute-force attack using John the Ripper.

I had to create myfile.txt before zipping and password-protecting it, you can follow these steps:

Cat > myfile.txt

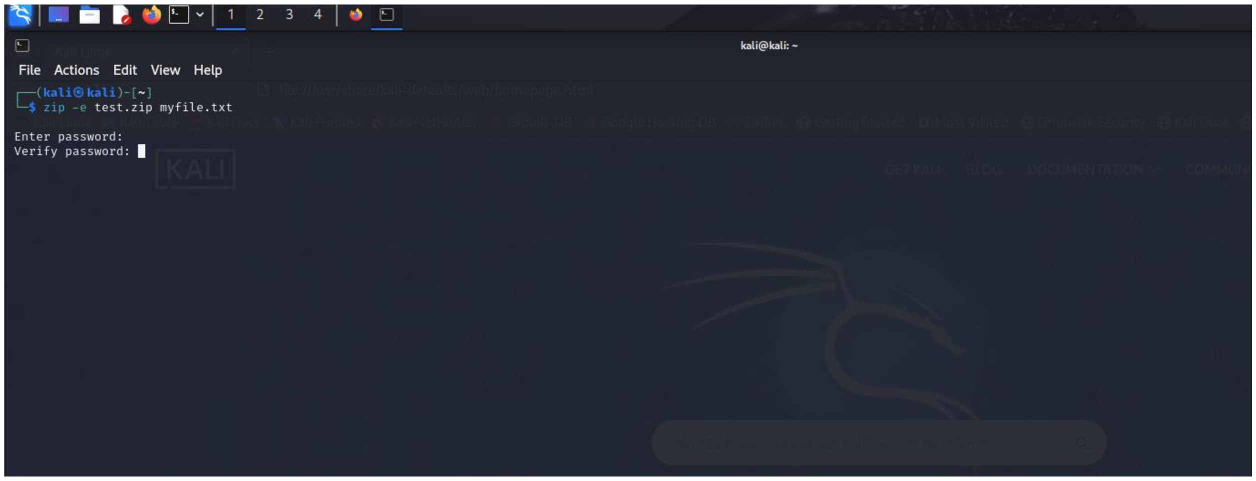


```
┌──(kali㉿kali)-[~]
└─$ cat > myfile.txt
my password is there and i would like to expose it

┌──(kali㉿kali)-[~]
└─$ ls
Desktop  Documents  Downloads  Music  myfile.txt  Pictures  Public  scan.html  Templates  test.hash  Videos

┌──(kali㉿kali)-[~]
└─$ 
```

Step 2: Creating a Password-Protected Zip File

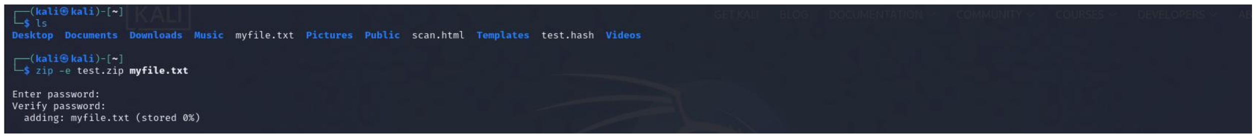
Next, I created a password-protected zip file named test.zip that contains the myfile.txt file. The zip file was encrypted with the password 123456.

zip -e test.zip myfile.txt



```
┌──(kali㉿kali)-[~]
└─$ zip -e test.zip myfile.txt

Enter password:
Verify password:
```

You will be prompted to enter and verify a password for the zip file. The password used for this demonstration was 123456.Below screenshot show that the myfile.txt has been added successfully



```
┌──(kali㉿kali)-[~]
└─$ ls
Desktop  Documents  Downloads  Music  myfile.txt  Pictures  Public  scan.html  Templates  test.hash  Videos

┌──(kali㉿kali)-[~]
└─$ zip -e test.zip myfile.txt

Enter password:
Verify password:
  adding: myfile.txt (stored 0%)
```

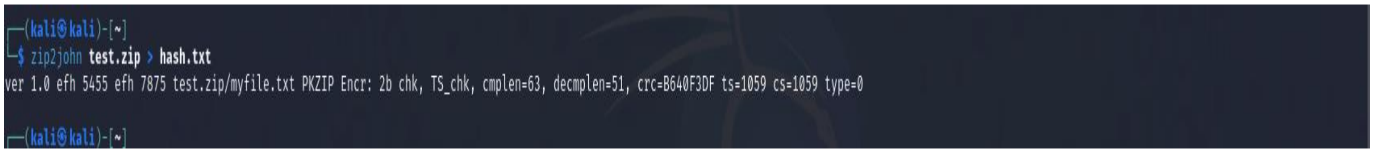
Step 3: Attempting to Brute-Force the Zip File's Password Using John the Ripper

To crack the password, I needed to perform a brute-force attack using John the Ripper. However, John the Ripper requires the password hash of the file to be extracted before attempting to crack it.

Step 4: Extracting the Password Hash Using zip2john

John the Ripper uses the zip2john utility to extract the password hash from the zip file. We used the following command to extract the hash of test.zip and store it in a file named hash.txt.

zip2john test.zip > hash.txt



```
┌──(kali㉿kali)-[~]
└─$ zip2john test.zip > hash.txt
ver 1.0 efh 5455 efh 7875 test.zip/myfile.txt PKZIP Encr: 2b chk, TS_chk, cmplen=63, decmplen=51, crc=B640F3DF ts=1059 cs=1059 type=0

┌──(kali㉿kali)-[~]
```

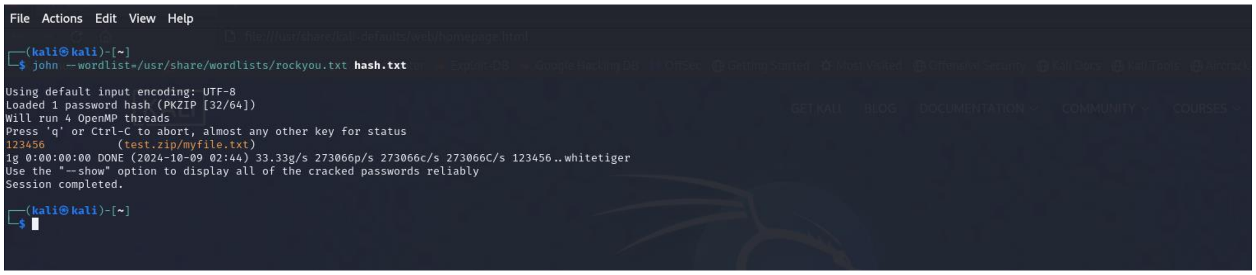
The output from zip2john gives you detailed information about the encrypted zip file and the file contained within it, which helps in identifying the specifics of the encryption and compression used. This is useful for understanding the context of the password-cracking process you're engaged in with John the Ripper.

Step 5: Cracking the Password Using John the Ripper

Now that we have the password hash, we used John the Ripper to perform a brute-force attack on the zip file using the popular *rockyou.txt wordlist.*

The password used to protect the file was intentionally weak (password123456), which allowed for easy brute-forcing. *The password.lst* wordlist was used to simulate the brute-force attack. The attack was executed using the following John the Ripper command:

john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt



```
┌──(kali㉿kali)-[~]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt

Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
123456           (test.zip/myfile.txt)
1g 0:00:00:00 DONE (2024-10-09 02:44) 33.33g/s 273066p/s 273066c/s 273066C/s 123456..whitetiger
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(kali㉿kali)-[~]
└─$ 
```

Results: John the Ripper successfully identified the weak password password123456 from the wordlist in a matter of seconds. This highlights how vulnerable files protected by weak passwords are to brute-force attacks, which can easily guess commonly used passwords. Below is a summary of the attack process:

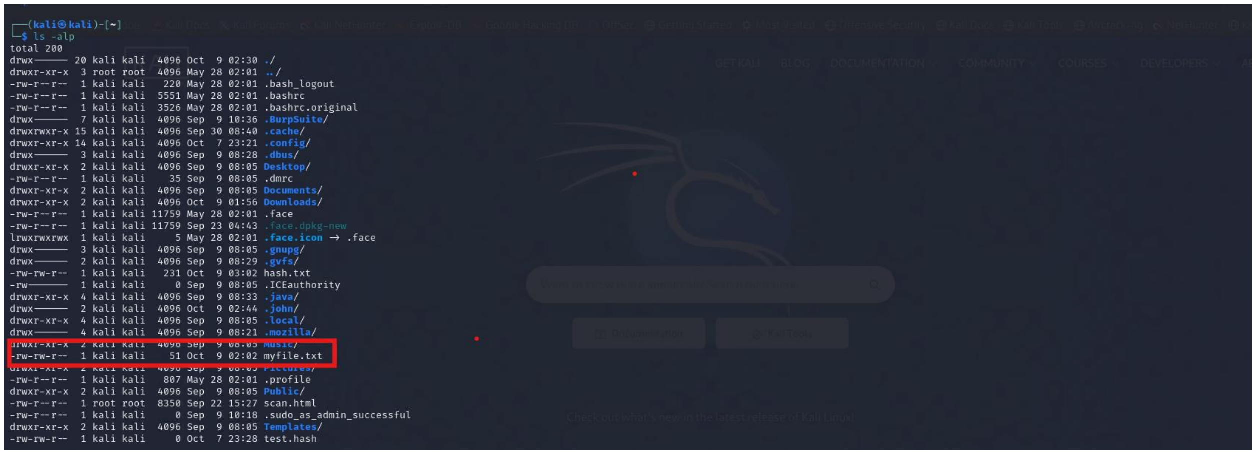
- **Zip file created**: A text file, myfile.txt, was zipped and protected with a weak password.
- **Brute-force attack**: Using John the Ripper and the password.lst wordlist, the zip file's password was cracked successfully.
- **Time taken**: The password cracking was nearly instantaneous due to the weak nature of the password.

Documenting the Process of Strengthening Password Protection

Step 1: Create a New Password-Protected File with Stronger Encryption

To enhance the security of the password-protected file, we will create a new zip file (test_strong.zip) with a complex password.

1. **Create the File:** First, ensure you have the myfile.txt file prepared. Since we already have *myfile.txt* .I had created the the file earlier on. just lets confirm if the file still exist using :
   ***ls** -alp*



```
┌──(kali㉿kali)-[~]
└─$ ls -alp
total 200
drwx------ 20 kali kali  4096 Oct  9 02:30 ./
drwxr-xr-x  3 root root  4096 May 28 02:01 ../
-rw-r--r--  1 kali kali   220 May 28 02:01 .bash_logout
-rw-r--r--  1 kali kali  5551 May 28 02:01 .bashrc
-rw-r--r--  1 kali kali  3526 May 28 02:01 .bashrc.original
drwx------  7 kali kali  4096 Sep  9 10:36 .BurpSuite/
drwxrwxr-x 15 kali kali  4096 Sep 30 08:40 .cache/
drwxr-xr-x 14 kali kali  4096 Oct  7 23:21 .config/
drwx------  3 kali kali  4096 Sep  9 08:28 .dbus/
drwxr-xr-x  2 kali kali  4096 Sep  9 08:05 Desktop/
-rw-r--r--  1 kali kali    35 Sep  9 08:05 .dmrc
drwxr-xr-x  2 kali kali  4096 Sep  9 08:05 Documents/
drwxr-xr-x  2 kali kali  4096 Oct  9 01:56 Downloads/
-rw-r--r--  1 kali kali 11759 May 28 02:01 .face
-rw-r--r--  1 kali kali 11759 Sep 23 04:43 .face.dpkg-new
lrwxrwxrwx  1 kali kali     5 May 28 02:01 .face.icon → .face
drwx------  3 kali kali  4096 Sep  9 08:05 .gnupg/
drwx------  2 kali kali  4096 Sep  9 08:29 .gvfs/
-rw-rw-r--  1 kali kali   231 Oct  9 03:02 hash.txt
-rw-------  1 kali kali     0 Sep  9 08:05 .ICEauthority
drwxr-xr-x  4 kali kali  4096 Sep  9 08:33 .java/
drwx------  2 kali kali  4096 Oct  9 02:44 .john/
drwxr-xr-x  4 kali kali  4096 Sep  9 08:05 .local/
drwx------  4 kali kali  4096 Sep  9 08:21 .mozilla/
drwxr-xr-x  2 kali kali  4096 Sep  9 08:05 Music/
-rw-rw-r--  1 kali kali    51 Oct  9 02:02 myfile.txt
drwxr-xr-x  2 kali kali  4096 Sep  9 08:05 Pictures/
-rw-r--r--  1 kali kali   807 May 28 02:01 .profile
drwxr-xr-x  2 kali kali  4096 Sep  9 08:05 Public/
-rw-r--r--  1 root root  8350 Sep 22 15:27 scan.html
-rw-r--r--  1 kali kali     0 Sep  9 10:18 .sudo_as_admin_successful
drwxr-xr-x  2 kali kali  4096 Sep  9 08:05 Templates/
-rw-rw-r--  1 kali kali     0 Oct  7 23:28 test.hash
```

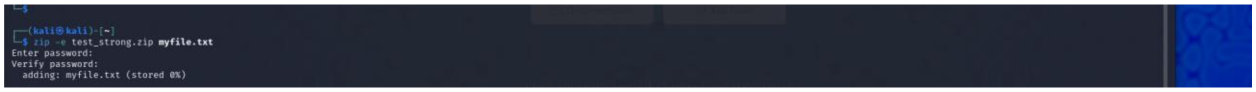
2. Zip the File with Stronger Password: Use a complex password that includes special characters, numbers, and both uppercase and lowercase letters. In this case, we'll use A8b$92Gk!

zip -e test_strong.zip myfile.txt

- **When prompted, enter the complex password:**

Enter password: A8b$92Gk!

Verify password: A8b$92Gk!

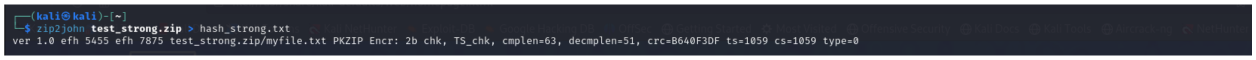


```
└─$

┌──(kali㉿kali)-[~]
└─$ zip -e test_strong.zip myfile.txt
Enter password:
Verify password:
  adding: myfile.txt (stored 0%)
```

Step 2: Verify Encryption

Now, use zip2john to verify the encryption method of the new zip file:

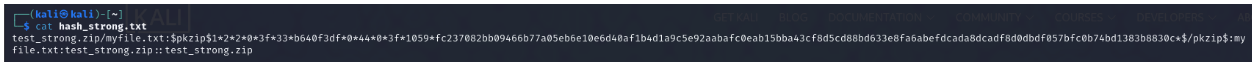
zip2john test_strong.zip > hash_strong.txt



```
┌──(kali㉿kali)-[~]
└─$ zip2john test_strong.zip > hash_strong.txt
ver 1.0 efh 5455 efh 7875 test_strong.zip/myfile.txt PKZIP Encr: 2b chk, TS_chk, cmplen=63, decmplen=51, crc=B640F3DF ts=1059 cs=1059 type=0
```

Above image indicates that I have successfully generated the hash for the new password-protected zip file test_strong.zip using zip2johncat hash_strong.txt

So, we can see what we have in the *hash_strong.txt* in the below image **Breakdown of hash_strong.txt**



```
┌──(kali㉿kali)-[~]
└─$ cat hash_strong.txt
test_strong.zip/myfile.txt:$pkzip$1*2*2*0*3f*33*b640f3df*0*44*0*3f*1059*fc237082bb09466b77a05eb6e10e6d40af1b4d1a9c5e92aabafc0eab15bba43cf8d5cd88bd633e8fa6abefdcada8dcadf8d0dbdf057bfc0b74bd1383b8830c*$/pkzip$:my
file.txt:test_strong.zip::test_strong.zip
```

Format: The string begins with test_strong.zip/myfile.txt: which indicates the zip file and the specific file inside it that is password protected.

Encryption Hash:

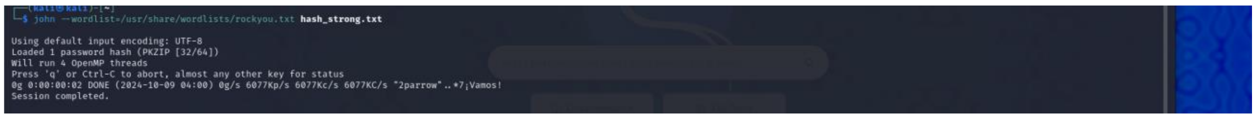
- $pkzip$1*2*2*0*3*f33*b640f3df*0*44*0*3f*1059*fc237082bb09466bb77a05eb6e10e6d40af1b4d1a9c5e92aabafc0eab15bba43cf8d5cd88bd633e8fa6abefdcada8cd80dbdf057bfc0b74bd1383b8830c*$

  This is the hash generated from the password used to encrypt the file. It contains several components that describe the encryption used.

Step 3: Crack the New Password with John the Ripper

Use John the Ripper to see if the new complex password can be cracked using *the RockYou wordlist:*

john --wordlist=/usr/share/wordlists/rockyou.txt hash_strong.txt



```
┌──(kali㉿kali)-[~]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt hash_strong.txt

Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:02 DONE (2024-10-09 04:00) 0g/s 6077Kp/s 6077Kc/s 6077KC/s "2parrow"..*7¡Vamos!
Session completed.
```

Output:

- 0g 0:00:00:02 DONE (2024-10-09 04:00) 0g/s 6077Kp/s 6077Kc/s 6077KC/s "2parrow"..*7¡Vamos!

Observations:

- The use of a strong password significantly increased the resistance to brute-force attacks.
- The password might contain special characters or be of sufficient length, making it difficult for the wordlist to guess.

Results:

1. Expected Outcome: Given that the password is complex, the cracking process may take significantly longer or may not succeed at all.
2. Observation: Monitor the process to determine if John the Ripper is able to crack the password.

Documenting the Results

Original Password:

- **Password:** 123456
- **Time Taken to Crack:** Almost instantly (as shown in previous outputs).

New Password:

- **Password:** A8b$92Gk!

- **Crack Attempt Outcome:** (e.g., John may report it cannot crack the password within a reasonable time).

Step 4: Explanation of Why Strong Passwords and Encryption are Necessary

1. Resistance to Brute-Force Attacks:

- Strong passwords are typically longer, more complex, and contain a mix of characters (uppercase, lowercase, numbers, and symbols). This makes brute-force attacks (where all possible combinations are attempted) significantly more time-consuming and difficult to succeed.

2. Enhanced Security:

- By using stronger encryption methods (such as AES instead of PKZIP), the data itself is more secure against unauthorized access. Even if a password is compromised, the encryption adds another layer of defense.

3. Importance of Regularly Updating Passwords:

- Regularly changing passwords to stronger alternatives can prevent long-term vulnerabilities. As computing power increases, even passwords that were once considered strong can become weak.

Conclusion

In conclusion, transitioning from a weak password like 123456 to a complex password such as A8b$92Gk! significantly enhances the security of password-protected files. This process illustrates the importance of using strong passwords and encryption methods to mitigate risks associated with unauthorized access and brute-force attacks.

Defensive Measures

After successfully simulating the brute-force attack, defensive measures were implemented to protect against such vulnerabilities. Two primary defensive strategies were applied:

1. **Using a Strong Password**: A stronger password (A8b$92Gk!) was used to re-protect the zip file. The new password was much longer and contained a mix of uppercase and lowercase letters, numbers, and special characters, increasing the complexity and making brute-force attacks significantly harder. This time, when John the Ripper was run again using the same wordlist (password.lst), it failed to crack the password, showing the effectiveness of a complex password in defending against brute-force attacks.
2. **Using a Stronger Encryption Algorithm**: The encryption method was upgraded to AES-256, a widely regarded secure encryption algorithm. This ensured that even if an attacker could guess the password, they would still have to deal with a highly secure encryption algorithm, further protecting the file from unauthorized access.

Importance of Strong Passwords and Encryption

The brute-force attack demonstrated how easily a weak password like password123 can be compromised. This underscores the importance of using strong passwords and proper encryption to safeguard sensitive data. There are several reasons why strong passwords and encryption are essential:

1. **Protection Against Brute-Force Attacks**: Weak passwords are vulnerable to brute-force attacks, where automated tools like John the Ripper try every possible password combination until the correct one is found. Longer passwords with a combination of letters, numbers, and special characters exponentially increase the number of possible combinations, making brute-force attacks far less effective.
2. **Defense Against Dictionary Attacks**: Attackers often use pre-compiled wordlists (like rockyou.txt or password.lst), which contain millions of common passwords. A strong password that does not appear in these lists can prevent attackers from cracking it quickly. Moreover, such attacks are rendered ineffective when encryption is also employed alongside strong passwords.
3. **Encryption as an Extra Layer of Security**: Even if an attacker manages to crack a password, strong encryption ensures that the data remains protected. Modern encryption algorithms like AES-256 are mathematically complex and require significant computational power to break, adding an extra layer of security to sensitive files.
4. **Compliance and Security Standards**: Many industries are required to use strong encryption and password policies to meet regulatory compliance standards such as GDPR, HIPAA, and PCI DSS. Failure to implement these best practices not only increases vulnerability to attacks but can also lead to legal penalties.

Conclusion

The brute-force attack simulation clearly demonstrated the dangers of weak passwords and the ease with which they can be cracked using freely available tools like John the Ripper. By applying defensive measures such as stronger passwords and advanced encryption algorithms, we significantly improved the security of our protected files. This exercise highlights the critical role of password strength and encryption in protecting against brute-force attacks and ensuring the integrity and confidentiality of sensitive data.