John the Ripper: Password cracking tool

John the Ripper is an Open-Source password security auditing and password recovery tool available for many operating systems. It is designed to detect weak passwords by performing dictionary attacks, brute force attacks, and rainbow table attacks.

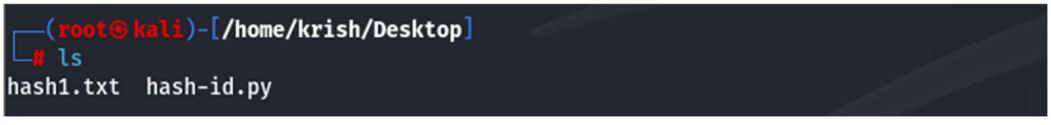
Key Features of John the Ripper:

- Multi-platform Support: Works on Linux, macOS, Windows, and other Unix-based systems.
- Supports Various Hash Formats: Can crack passwords hashed with MD5, SHA-1, SHA-256, SHA-512, bcrypt, NTLM, and many more.
- Modes of Operation:
    - Dictionary Attack: Uses a wordlist to try commonly used passwords.
    - Brute Force Attack: Tries all possible character combinations.
    - Incremental Mode: Tries passwords based on frequency analysis.
    - External Mode: Uses custom rules for password generation.
- Highly Optimized Performance: Utilizes CPU and GPU acceleration for faster cracking.

Using John the Ripper

1. We can identify the type of hash used by using hash-id. You can install hash-id by running the following command. The link is taken from the hash id Gitlab.

2. So now we have two files hash-id and hash1.txt(the file that contains the hashed password)



3. You can run hash-id using python3.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# python3 hash-id.py
/home/krish/Desktop/hash-id.py:13: SyntaxWarning: invalid escape sequence '\ '
  logo='''   #########################################################################
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \      __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \   /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \ /\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.2 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################
--------------------------------------------------
 HASH:
```

4. Provide the hash once the above screen appears. It tells us the most possible hash.

```
HASH: 25d55ad283aa400af464c76d713c07ad

Possible Hashs:
[+] MD5
[+] Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))
```

5. Once the hash used is known we can run John the Ripper to crack this hash. On Kali linux John is already installed. You can check the same by running the john command it should reply with the image attached below:

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john
John the Ripper 1.9.0-jumbo-1+bleeding-aec1328d6c 2021-11-02 10:45:52 +0100 OMP [linux-gnu 64-bit x86_64 AVX2 AC]
Copyright (c) 1996-2021 by Solar Designer and others
Homepage: https://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]

Use --help to list all available options.
```

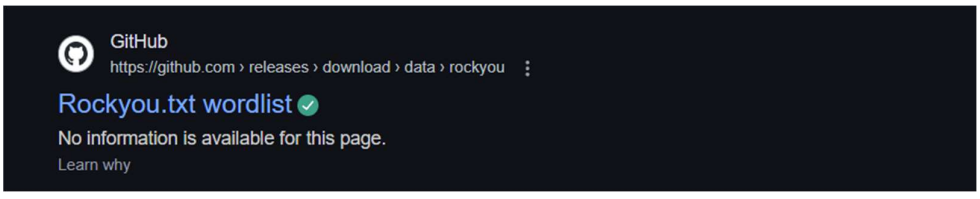
Dictionary Attacks

First, we will be running dictionary attacks using John the Ripper since they are very simple to run. For this you need a wordlist. Kali linux has already some wordlists installed for you. They can be found at the following path:

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# cd /usr/share/wordlists

┌──(root㉿kali)-[/usr/share/wordlists]
└─# ls
amass  dirb  dirbuster  dnsmap.txt  fasttrack.txt  fern-wifi  john.lst  legion  metasploit  nmap.lst  rockyou.txt.gz  sqlmap.txt  wfuzz  wifite.txt
```

You can also install rockyou.txt from github. As the one provided by kali is compressed, you can extract it or download directly from github.



Once you have the wordlist, you have everything to run a dictionary attack. Remember the path where you have saved your wordlist as you will need it.

1. Type john and the format of the hash. That is why we used hash-id to identify the type of hash.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --format=raw-md5
```

   Don't press enter yet you still need the wordlist and the file that contains the hash. Our format is md5. The raw here means the hash is unsalted. Salting is a technique used to enhance password security by adding a random value (salt) to a password before hashing it. This prevents attackers from easily cracking passwords using precomputed hash databases (like rainbow tables).

2. Enter the path of the wordlist and the file that contains the hash password. Don't forget you need to be at that folder where the file is present.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --format=raw-md5 --wordlist=/home/krish/Desktop/rockyou.txt  hash1.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
No password hashes left to crack (see FAQ)
```

   Congratulations the password has been cracked as indicated by the No password hashes left to crack.

3. Don't panic if you can't see the password. Just run the following command:

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --show --format=raw-md5 hash1.txt
?:12345678

1 password hash cracked, 0 left
```

Voila! You have successfully cracked the password.

What if the password is not hashed as md5? Just change the format that you are using. For example, if after running hash-id the hash has now come to be SHA1 do the following:

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --format=raw-sha1 --wordlist=/home/krish/Desktop/rockyou.txt hash2.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 256/256 AVX2 8x])
Warning: no OpenMP support for this hash type, consider --fork=6
Press 'q' or Ctrl-C to abort, almost any other key for status
12345678         (?)
1g 0:00:00:00 DONE (2025-03-14 14:57) 50.00g/s 800.0p/s 800.0c/s 800.0C/s 12345678..jessica
Use the "--show --format=Raw-SHA1" options to display all of the cracked passwords reliably
Session completed.
```

Successfully cracked again.

NTLM Hashes: The NTLM hash is the cryptographic format in which user passwords are stored on Windows systems. NTLM hashes are stored in the SAM (security account manager)

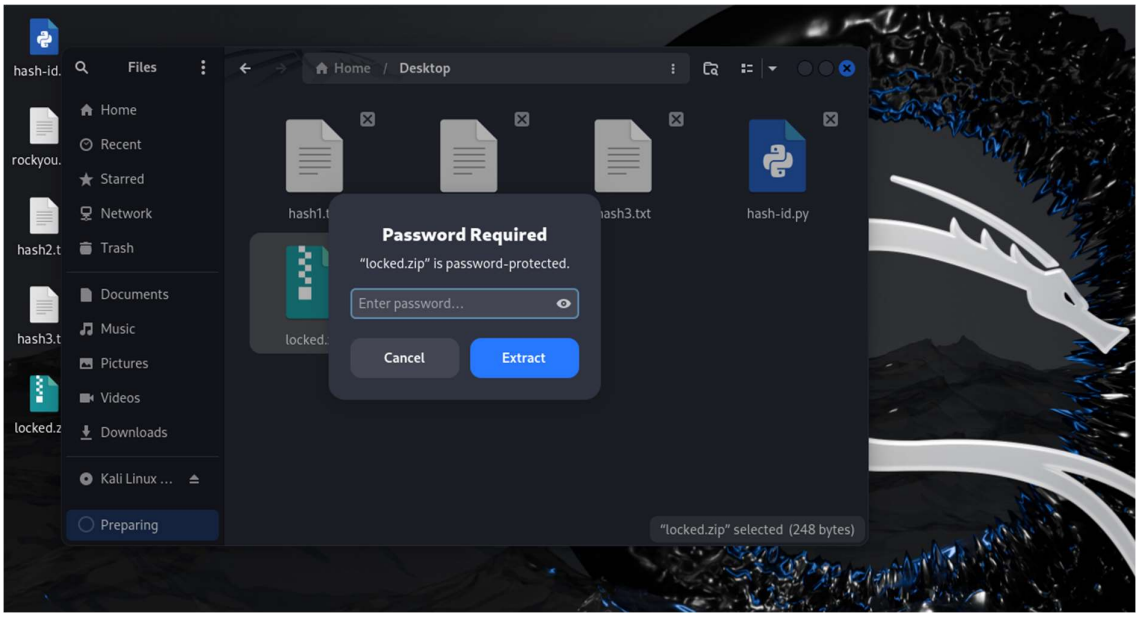
or NTDS file of a domain controller. They are a fundamental part of the mechanism used to authenticate a user through different communications protocols.

You can easily crack ntlm hashes also. Just change the format to nt.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --format=nt --wordlist=/home/krish/Desktop/rockyou.txt hash3.txt
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=6
Press 'q' or Ctrl-C to abort, almost any other key for status
12345            (?)
1g 0:00:00:00 DONE (2025-03-14 15:10) 33.33g/s 6400p/s 6400c/s 6400C/s 123456..november
Use the "--show --format=NT" options to display all of the cracked passwords reliably
Session completed.
```

How to get password for locked zip files.

You can also get password for locked zip files using John the Ripper. A zip may be protected in the following manner:



How do you get the password to access the content inside the files? You can use the zip2john feature. It is really simple just add a step and you will crack it. Follow the following steps:

1. Use zip2john and save its result in a new txt file

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# ls
hash1.txt  hash2.txt  hash3.txt  hash-id.py  locked.zip  rockyou.txt

┌──(root㉿kali)-[/home/krish/Desktop]
└─# zip2john locked.zip > unlock.txt
```

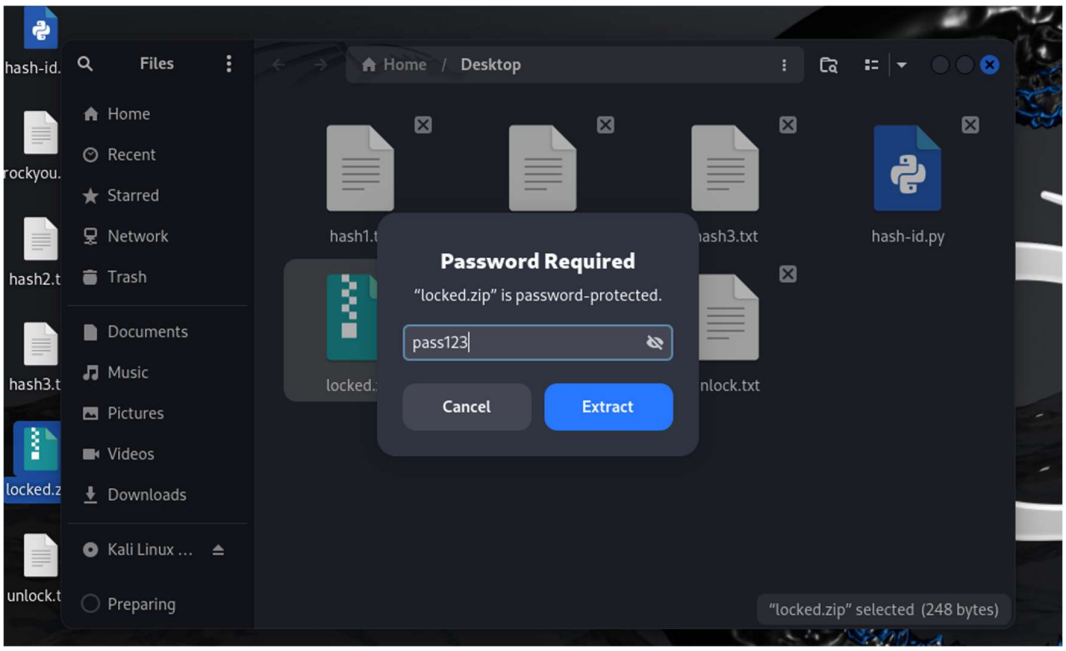
2. This zip.txt now contains the hash of the original file

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# cat unlock.txt
locked.zip/locked.txt:$zip2$*0*3*0*6a86f9733c57811022ee159701f424d6*3771*8*c47a3a9804d682b5*cc178d9b29e8969bc5e5*$/zip2$:locked.txt:locked.zip:locked.zip
```

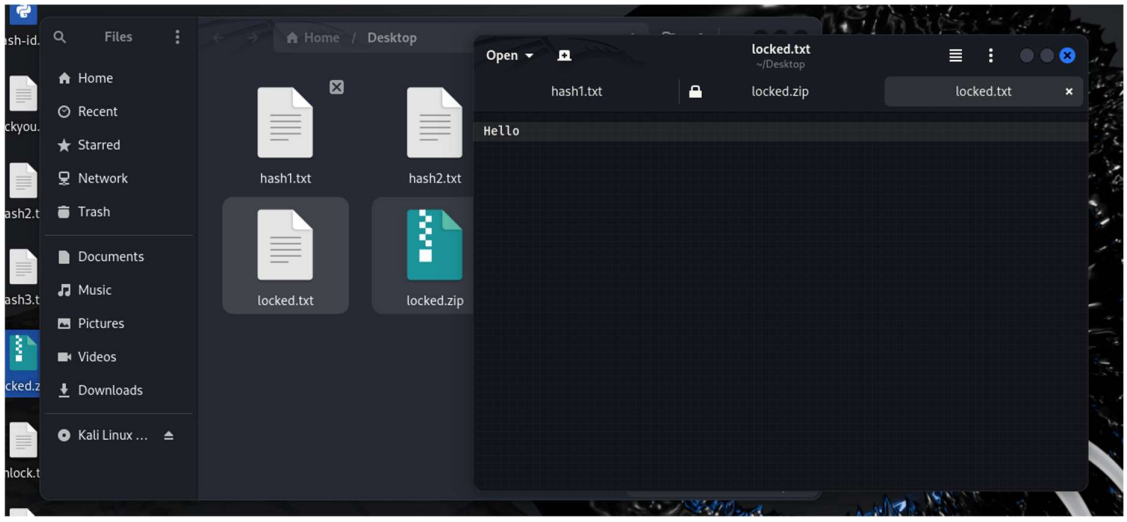
We can use John the ripper as we were using it before on this file to get the password of the zip file

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --wordlist=/home/krish/Desktop/rockyou.txt unlock.txt
Using default input encoding: UTF-8
Loaded 1 password hash (ZIP, WinZip [PBKDF2-SHA1 256/256 AVX2 8x])
Cost 1 (HMAC size) is 8 for all loaded hashes
Will run 6 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
pass123          (locked.zip/locked.txt)
1g 0:00:00:00 DONE (2025-03-14 15:26) 3.846g/s 47261p/s 47261c/s 47261C/s 123456..havana
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Voila this is it. We got the password



After extracting it creates a txt file which is same as the zip file. We can now open it to see its contents.



How to crack id_rsa files?

id_rsa is a private SSH key used for secure authentication in **SSH connections**. If an SSH key is **protected with a passphrase**, an attacker would need to crack it to gain unauthorized access. I will be generating my own id_rsa file for this and then I will show how to crack it.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# ssh-keygen -t rsa -b 2048 -m PEM -f id_rsa
Generating public/private rsa key pair.
Enter passphrase for "id_rsa" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa
Your public key has been saved in id_rsa.pub
The key fingerprint is:
SHA256:gZ4qzpNDwI3EMFMuwFOSuxACGjlf75zwvX45rxDUKIA root@kali
The key's randomart image is:
+---[RSA 2048]----+
|%++o.            |
|B@E .. . o       |
|*+*. .o + .      |
|o=.....+ .       |
|...  =ooS        |
| ..  .= ..       |
| ....   .. .     |
| o+.    ..+      |
|  oo   ....+.    |
+----[SHA256]-----+
```

This will create an id_rsa file that contains the passphrase

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# ls
hash1.txt  hash2.txt  hash3.txt  hash-id.py  id_rsa  id_rsa.pub  locked.zip  rockyou.txt  unlock.txt
```

To crack this file first use ssh2john command similar to the zip2john command and store the result in a new file.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# ssh2john id_rsa > ssh.txt
```

Use john command with the wordlist to crack the passphrase.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --wordlist=/home/krish/Desktop/rockyou.txt ssh.txt
Using default input encoding: UTF-8
Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 1 for all loaded hashes
Will run 6 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
admin123         (id_rsa)
1g 0:00:00:00 DONE (2025-03-14 15:40) 8.333g/s 750400p/s 750400c/s 750400C/s advanced..SAVAGE1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

If the private key (id_rsa) corresponds to a valid user on a server, and now since you have the passphrase, you can log in without needing a password.

Brute force Attacks

Now we are familiar with dictionary attacks. They use wordlists, are fast and simple but the problem is you may not be able to crack a password using this wordlist as people may have added their name, phone number, DOB as their password. Since they will not be present in a wordlist, we need to use the brute force approach to crack their passwords. A brute-force attack systematically tries all possible character combinations until it finds the correct password. Unlike a dictionary attack (where we used a predefined list like rockyou.txt), brute-force attacks attempt every possible password combination within a given set of rules.

It can be slow depending on how strong the password is and how little we know of the password. As we fill in the details it becomes much faster. Do we know the hash type of the password, its length, does it only use digits, lowercase or uppercase letters. All these factors determine the duration of a brute force attack. It may take minutes to months.

Now instead of using wordlist we will be using incremental.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# cat hash4.txt
806d26f7dcb1e95bbf9d8f93a6d1ee9b
```

This is the hash that I am trying to crack. We can use brute force for this.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --format=raw-md5  --incremental hash4.txt
```

Since I knew the format to be md5(using hash-id) I significantly reduce the time of the brute force attack.

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --format=raw-md5  --incremental hash4.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=6
Press 'q' or Ctrl-C to abort, almost any other key for status
Krish            (?)
1g 0:00:01:11 DONE (2025-03-14 17:31) 0.01403g/s 33955Kp/s 33955Kc/s 33955KC/s Kriev..Krixc
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

Here we have the password. You can change incremental settings depending on the information you have on the password.

For a password that only have digits you can set incremental=digits

```
┌──(root㉿kali)-[/home/krish/Desktop]
└─# john --format=raw-md5  --incremental=digits hash5.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=6
Press 'q' or Ctrl-C to abort, almost any other key for status
121222           (?)
1g 0:00:00:00 DONE (2025-03-14 17:45) 33.33g/s 115200p/s 115200c/s 115200C/s 220507..101121
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

Similarly, For lowercase letters only:

john --incremental=lower hash.txt

For uppercase + lowercase:

john --incremental=alpha hash.txt

For full brute-force (all character types):

john --incremental=all hash.txt

You can also use external rules for brute force. If you want more advanced brute-force patterns, use John's rules engine:

john --rules --incremental hash.txt

--rules apply John's intelligent password mutation rules, which prioritize common patterns before attempting full brute-force.