John The Ripper

1. Introduction

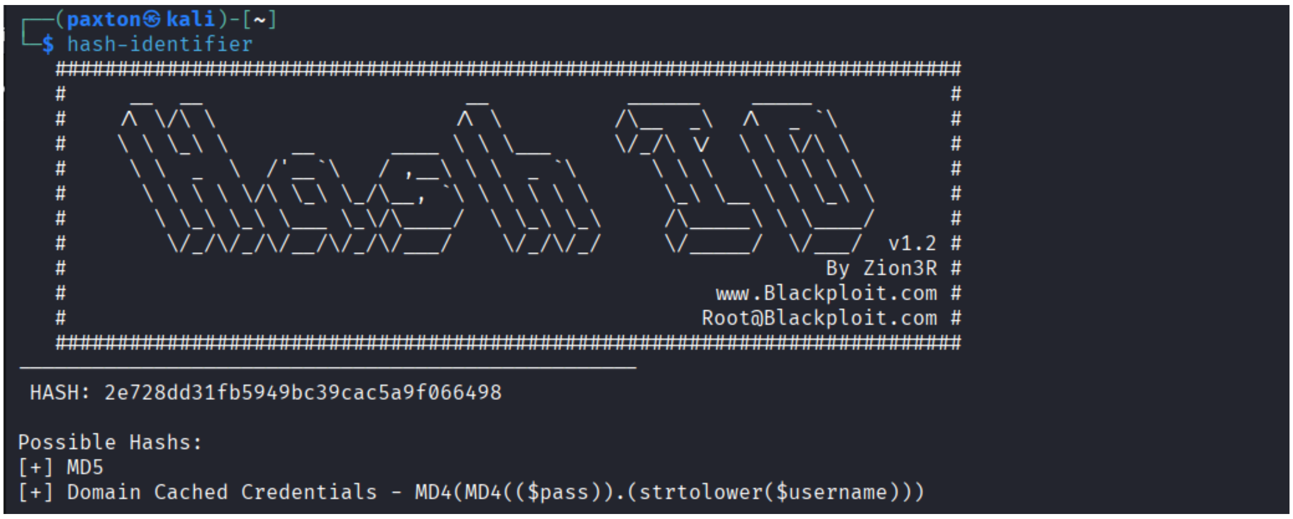
- John the Ripper (JtR) is a free, open-source, and cross-platform password cracking and security auditing tool primarily used by security professionals to test the strength of passwords and recover lost ones. It works by taking password hashes (encrypted representations of passwords) from systems like Unix, Windows, or encrypted files, and attempting to guess the original password using various highly efficient methods, including dictionary attacks and sophisticated rule-based attacks that intelligently modify common words. We use it to identify weak, easily guessed passwords in a system before malicious actors can, thereby strengthening overall digital security. Its key alternatives often include Hashcat (known for superior GPU acceleration) and Ophcrack (a Windows-focused tool).

2. Basic Cracking Mode

- **Checking hash type:**
  To efficiently crack a hash (determine the original input that generated it), the **first and most crucial step** is correctly identifying the **hashing algorithm** used. Different algorithms require different specialized tools and cracking modes.

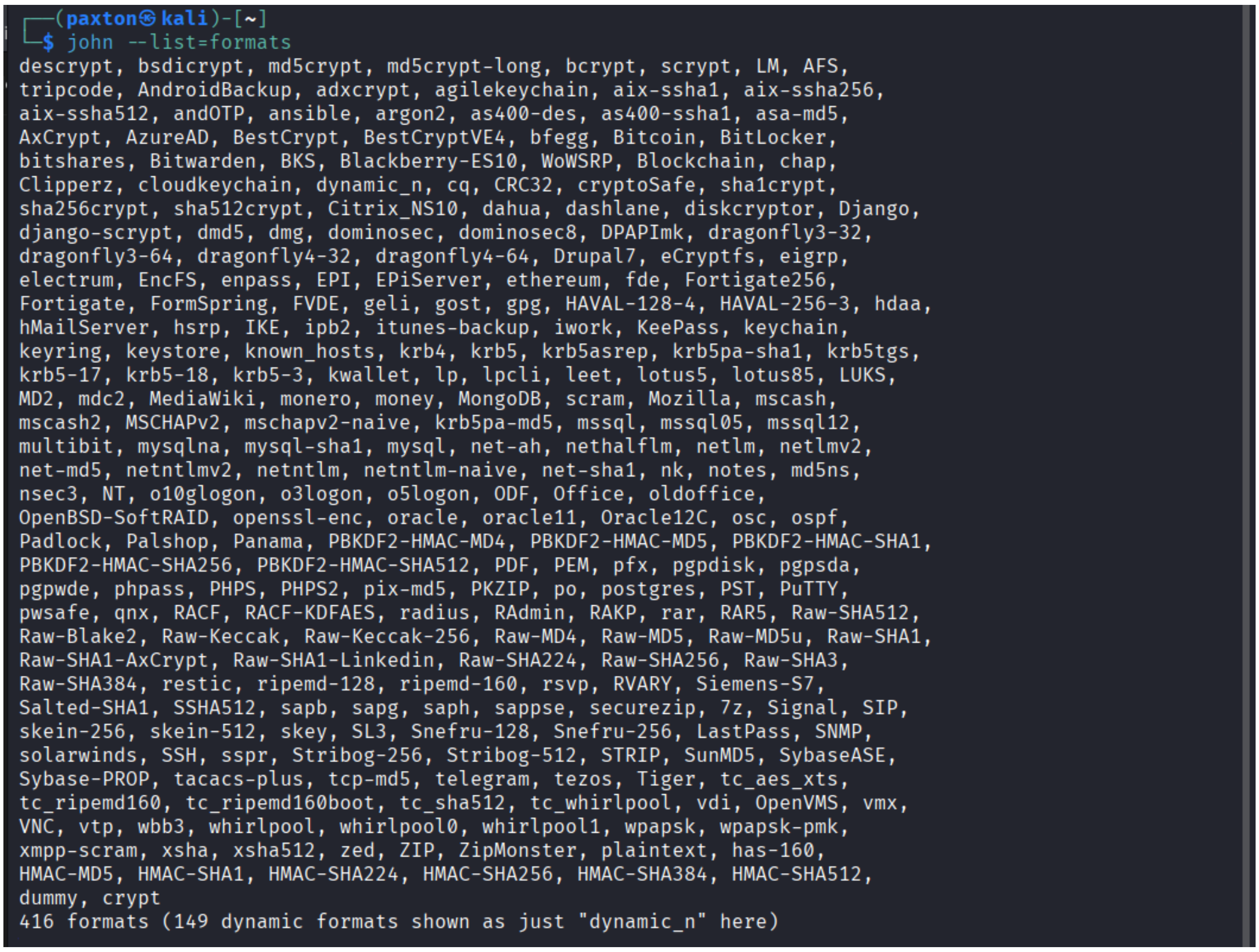
  Out of many tools and online sites to determine the hashes, one of them is **hash-identifier,** a command-line utility designed specifically for this purpose.

  How it works:
  Hash-identifier determines the type of hashing algorithm (like MD5 or SHA-256) by rapidly analyzing the structure of the unknown hash string. It works in three main steps: first, it checks the length of the hash, as many algorithms produce fixed-length outputs; second, it examines the character set (e.g., hexadecimal or Base64); and third, it searches for characteristic prefixes (or "signatures") like $6$ that indicate specific salted formats. By matching these properties against its database, the tool suggests the most probable hash type, which is essential information for correctly configuring password cracking software like John the Ripper.



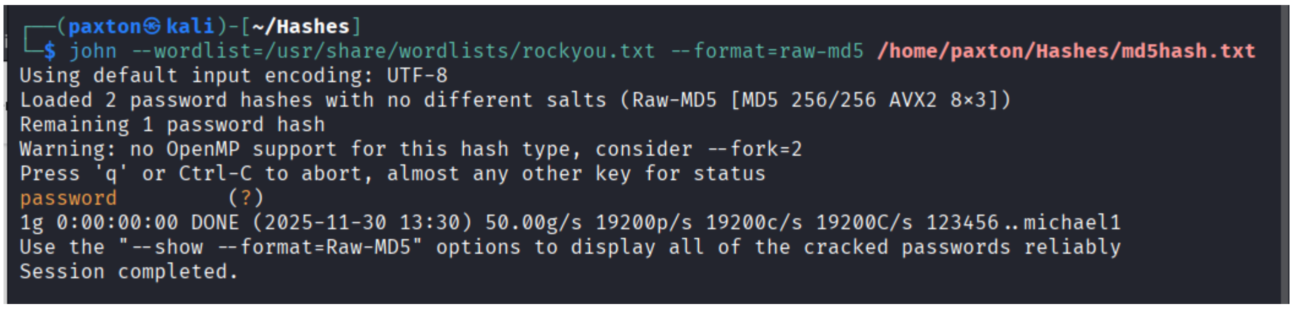
```
┌──(paxton㉿kali)-[~]
└─$ hash-identifier
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.2 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################
--------------------------------------------------
 HASH: 2e728dd31fb5949bc39cac5a9f066498

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

- **List all supported hash type**
- → **Command:** *john --list=formats*
- → **Explanation:** Shows every hash algorithm John can crack. This helps you identify the right format for your extracted hashes.



```
┌──(paxton㉿kali)-[~]
└─$ john --list=formats
descrypt, bsdicrypt, md5crypt, md5crypt-long, bcrypt, scrypt, LM, AFS,
tripcode, AndroidBackup, adxcrypt, agilekeychain, aix-ssha1, aix-ssha256,
aix-ssha512, andOTP, ansible, argon2, as400-des, as400-ssha1, asa-md5,
AxCrypt, AzureAD, BestCrypt, BestCryptVE4, bfegg, Bitcoin, BitLocker,
bitshares, Bitwarden, BKS, Blackberry-ES10, WoWSRP, Blockchain, chap,
Clipperz, cloudkeychain, dynamic_n, cq, CRC32, cryptoSafe, sha1crypt,
sha256crypt, sha512crypt, Citrix_NS10, dahua, dashlane, diskcryptor, Django,
django-scrypt, dmd5, dmg, dominosec, dominosec8, DPAPImk, dragonfly3-32,
dragonfly3-64, dragonfly4-32, dragonfly4-64, Drupal7, eCryptfs, eigrp,
electrum, EncFS, enpass, EPI, EPiServer, ethereum, fde, Fortigate256,
Fortigate, FormSpring, FVDE, geli, gost, gpg, HAVAL-128-4, HAVAL-256-3, hdaa,
hMailServer, hsrp, IKE, ipb2, itunes-backup, iwork, KeePass, keychain,
keyring, keystore, known_hosts, krb4, krb5, krb5asrep, krb5pa-sha1, krb5tgs,
krb5-17, krb5-18, krb5-3, kwallet, lp, lpcli, leet, lotus5, lotus85, LUKS,
MD2, mdc2, MediaWiki, monero, money, MongoDB, scram, Mozilla, mscash,
mscash2, MSCHAPv2, mschapv2-naive, krb5pa-md5, mssql, mssql05, mssql12,
multibit, mysqlna, mysql-sha1, mysql, net-ah, nethalflm, netlm, netlmv2,
net-md5, netntlmv2, netntlm, netntlm-naive, net-sha1, nk, notes, md5ns,
nsec3, NT, o10glogon, o3logon, o5logon, ODF, Office, oldoffice,
OpenBSD-SoftRAID, openssl-enc, oracle, oracle11, Oracle12C, osc, ospf,
Padlock, Palshop, Panama, PBKDF2-HMAC-MD4, PBKDF2-HMAC-MD5, PBKDF2-HMAC-SHA1,
PBKDF2-HMAC-SHA256, PBKDF2-HMAC-SHA512, PDF, PEM, pfx, pgpdisk, pgpsda,
pgpwde, phpass, PHPS, PHPS2, pix-md5, PKZIP, po, postgres, PST, PuTTY,
pwsafe, qnx, RACF, RACF-KDFAES, radius, RAdmin, RAKP, rar, RAR5, Raw-SHA512,
Raw-Blake2, Raw-Keccak, Raw-Keccak-256, Raw-MD4, Raw-MD5, Raw-MD5u, Raw-SHA1,
Raw-SHA1-AxCrypt, Raw-SHA1-Linkedin, Raw-SHA224, Raw-SHA256, Raw-SHA3,
Raw-SHA384, restic, ripemd-128, ripemd-160, rsvp, RVARY, Siemens-S7,
Salted-SHA1, SSHA512, sapb, sapg, saph, sappse, securezip, 7z, Signal, SIP,
skein-256, skein-512, skey, SL3, Snefru-128, Snefru-256, LastPass, SNMP,
solarwinds, SSH, sspr, Stribog-256, Stribog-512, STRIP, SunMD5, SybaseASE,
Sybase-PROP, tacacs-plus, tcp-md5, telegram, tezos, Tiger, tc_aes_xts,
tc_ripemd160, tc_ripemd160boot, tc_sha512, tc_whirlpool, vdi, OpenVMS, vmx,
VNC, vtp, wbb3, whirlpool, whirlpool0, whirlpool1, wpapsk, wpapsk-pmk,
xmpp-scram, xsha, xsha512, zed, ZIP, ZipMonster, plaintext, has-160,
HMAC-MD5, HMAC-SHA1, HMAC-SHA224, HMAC-SHA256, HMAC-SHA384, HMAC-SHA512,
dummy, crypt
416 formats (149 dynamic formats shown as just "dynamic_n" here)
```

- **Wordlist mode**
- → **Command:** *john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 /home/paxton/Hashes/md5hash.txt*
- → **Explanation:** --format=<format_name> command forces John to treat the hash as a specific type. Attempts to crack hashes by trying each word from the wordlist.
- → **Finding:** Here, the hash is saved in a file named md5hash.txt and the real password is "password" shown in different color in the screenshot.

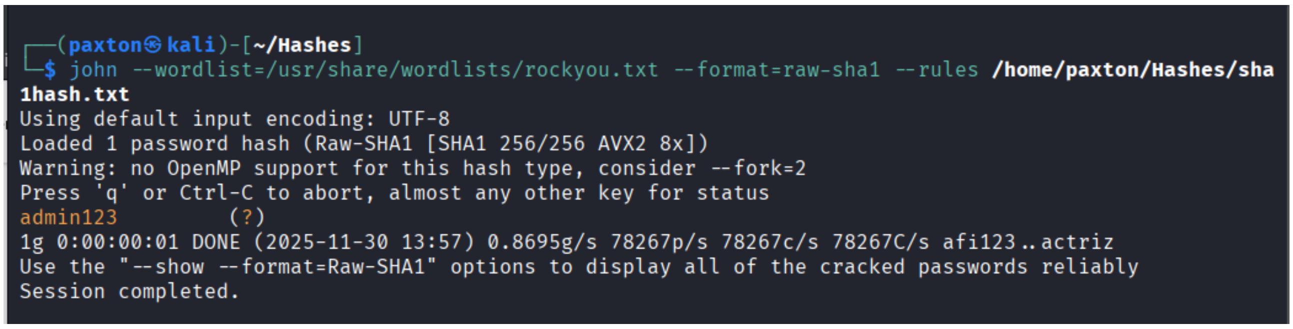


```
┌──(paxton㉿kali)-[~/Hashes]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 /home/paxton/Hashes/md5hash.txt
Using default input encoding: UTF-8
Loaded 2 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8×3])
Remaining 1 password hash
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
password         (?)
1g 0:00:00:00 DONE (2025-11-30 13:30) 50.00g/s 19200p/s 19200c/s 19200C/s 123456..michael1
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

- **Wordlist + Rules mode**

→ **Command:** *john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-sha1 --rules /home/paxton/Hashes/sha1hash.txt or john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-sha1 –rules=<rule_name> /home/paxton/Hashes/sha1hash.txt*

→ **Explanation:** Applies rules (like adding numbers or symbols) to each word in the wordlist. This helps simulate human password habits, making it far more effective.

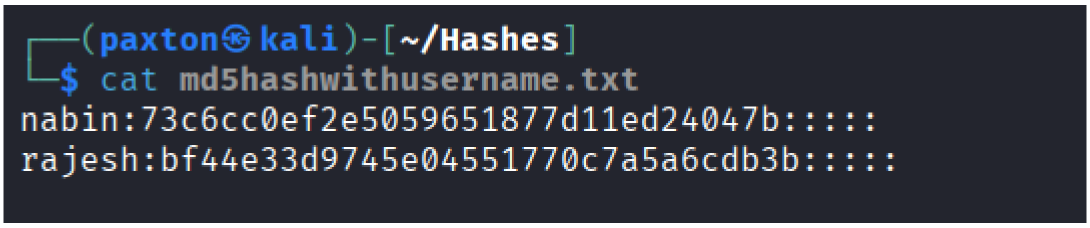


```
┌──(paxton㉿kali)-[~/Hashes]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-sha1 --rules /home/paxton/Hashes/sha
1hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 256/256 AVX2 8x])
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
admin123         (?)
1g 0:00:00:01 DONE (2025-11-30 13:57) 0.8695g/s 78267p/s 78267c/s 78267C/s afi123..actriz
Use the "--show --format=Raw-SHA1" options to display all of the cracked passwords reliably
Session completed.
```

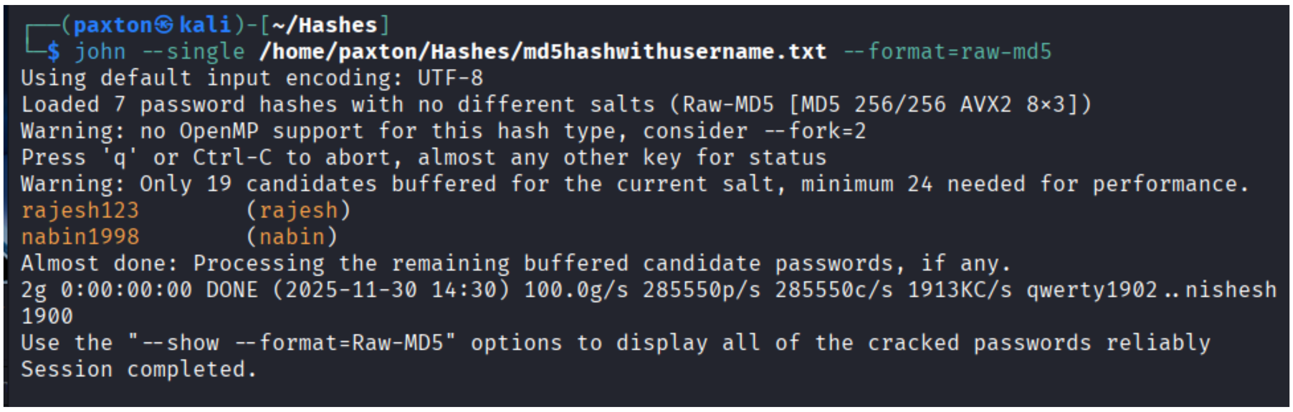
- **Single crack mode**

→ **Command:** *john --single /home/paxton/Hashes/md5hashwithusername.txt --format=raw-md5*

→ **Explanation:** Applies rules (like adding numbers or symbols) to each word in the wordlist. This helps simulate human password habits, making it far more effective.

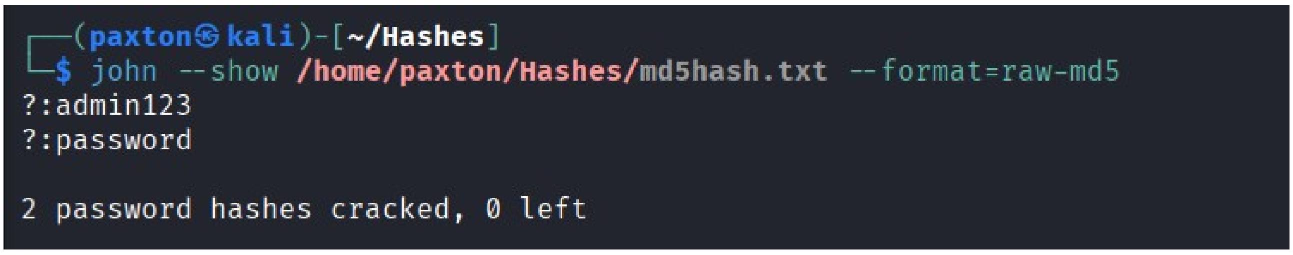


```
┌──(paxton㉿kali)-[~/Hashes]
└─$ cat md5hashwithusername.txt
nabin:73c6cc0ef2e5059651877d11ed24047b::::
rajesh:bf44e33d9745e04551770c7a5a6cdb3b::::
```



```
┌──(paxton㉿kali)-[~/Hashes]
└─$ john --single /home/paxton/Hashes/md5hashwithusername.txt --format=raw-md5
Using default input encoding: UTF-8
Loaded 7 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8×3])
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 19 candidates buffered for the current salt, minimum 24 needed for performance.
rajesh123        (rajesh)
nabin1998        (nabin)
Almost done: Processing the remaining buffered candidate passwords, if any.
2g 0:00:00:00 DONE (2025-11-30 14:30) 100.0g/s 285550p/s 285550c/s 1913KC/s qwerty1902..nishesh
1900
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

- **Show cracked password**
- → **Command:** *john --show /home/paxton/Hashes/md5hash.txt --format=raw-md5*
- → **Explanation:** Displays all passwords John has successfully cracked. This is useful for password audits and documentation.



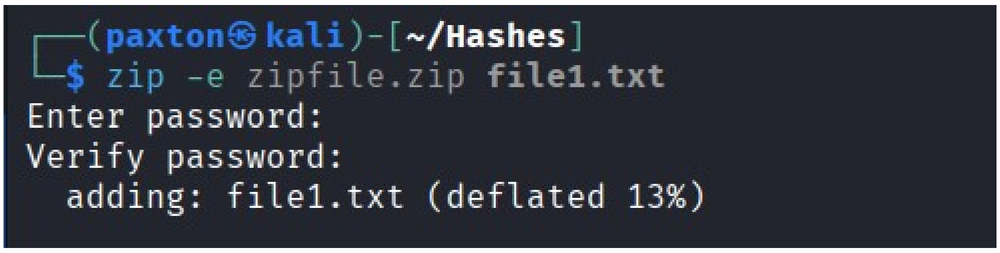
```
┌──(paxton㉿kali)-[~/Hashes]
└─$ john --show /home/paxton/Hashes/md5hash.txt --format=raw-md5
?:admin123
?:password

2 password hashes cracked, 0 left
```

- **Incremental mode**
- → **Command:** *john /home/paxton/Hashes/md5hashwithusername.txt --incremental --format=raw-md5*
- → **Explanation:** Tries every possible character combination. It is very thorough but slow, so it should be used only when wordlist methods fail.

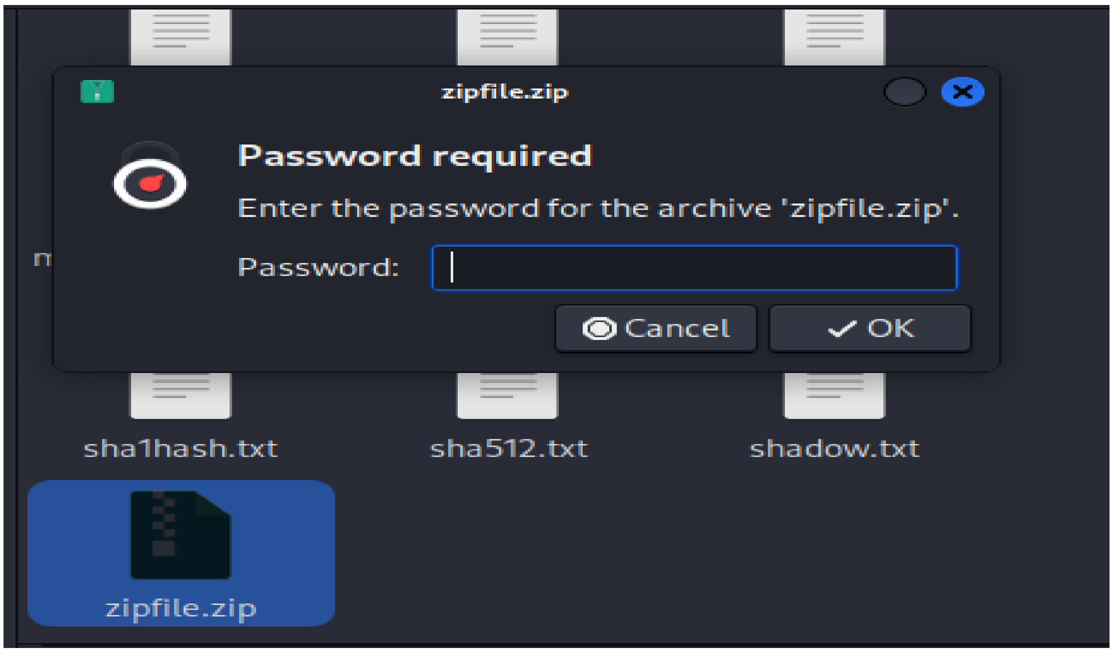
3. Cracking password protected files

- **Extract and Crack Hashes from /etc/shadow (your own VM only)**
- → **Command: 1** *sudo unshadow /etc/passwd /etc/shadow > myhashes.txt*
- → **Explanation:** Combines `/etc/passwd` and `/etc/shadow` into a file that John can crack.

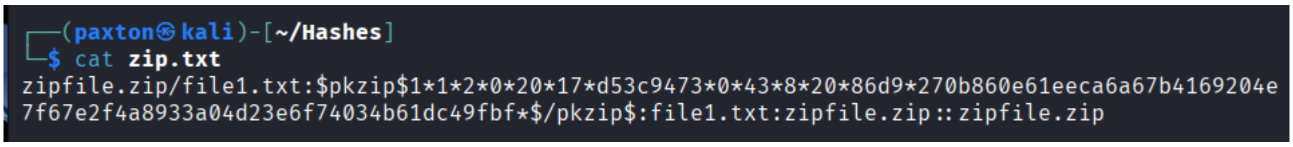
- **Extract hash and crack zip file**
- → **Command :** *zip -e zipfile.zip file1.txt*
- → **Explanation:** Convert file name *file1.txt* into zip file named *zipfile.zip* with password.



```
┌──(paxton㉿kali)-[~/Hashes]
└─$ zip -e zipfile.zip file1.txt
Enter password:
Verify password:
  adding: file1.txt (deflated 13%)
```

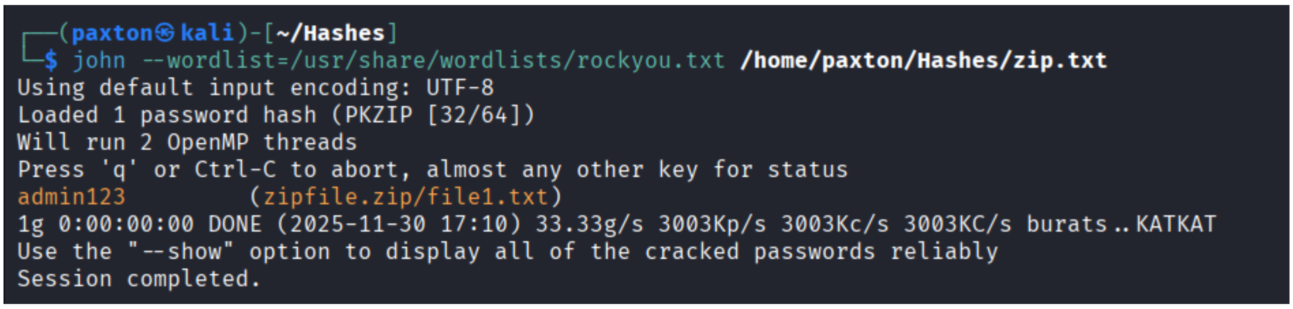


→ **Command :** *zip2john zipfile.zip > zip.txt*
→ **Explanation:** Converts a zip archive's password hash into John-compatible format. Useful for testing secure zip creation on your own files.



```
┌──(paxton㉿kali)-[~/Hashes]
└─$ cat zip.txt
zipfile.zip/file1.txt:$pkzip$1*1*2*0*20*17*d53c9473*0*43*8*20*86d9*270b860e61eeca6a67b4169204e
7f67e2f4a8933a04d23e6f74034b61dc49fbf*$/pkzip$:file1.txt:zipfile.zip::zipfile.zip
```

→ **Command:** *john --wordlist=/usr/share/wordlists/rockyou.txt /home/paxton/Hashes/zip.txt*
→ **Explanation:** cracks zip password hashes.



```
┌──(paxton㉿kali)-[~/Hashes]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt /home/paxton/Hashes/zip.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
admin123         (zipfile.zip/file1.txt)
1g 0:00:00:00 DONE (2025-11-30 17:10) 33.33g/s 3003Kp/s 3003Kc/s 3003KC/s burats..KATKAT
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

- **Cracking ssh private key**

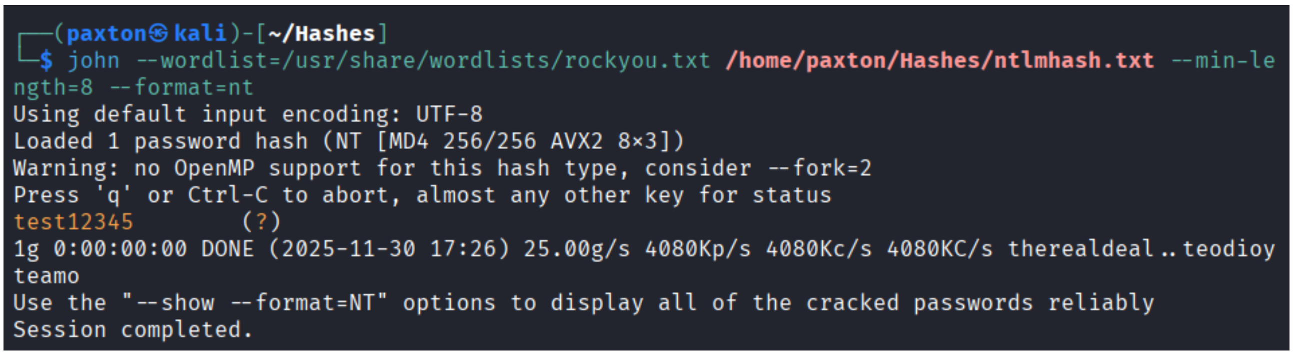
→ **Command:** ssh2john <id_rsa> > sshhash.txt
→ **Explanation:** Converts encrypted SSH private keys into a format John can test

4. Password policy testing

- **Minimum password length check**

→ **Command:** *john --wordlist=/usr/share/wordlists/rockyou.txt /home/paxton/Hashes/ntlmhash.txt --min-length=8 --format=nt*

→ **Explanation:** Forces John to only try passwords with a minimum length. Used to ensure your system enforces proper password policies.



```
┌──(paxton㉿kali)-[~/Hashes]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt /home/paxton/Hashes/ntlmhash.txt --min-le
ngth=8 --format=nt
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 256/256 AVX2 8×3])
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
test12345        (?)
1g 0:00:00:00 DONE (2025-11-30 17:26) 25.00g/s 4080Kp/s 4080Kc/s 4080KC/s therealdeal..teodioy
teamo
Use the "--show --format=NT" options to display all of the cracked passwords reliably
Session completed.
```

- **Maximum password length**

→ **Command:** *john --wordlist=/usr/share/wordlists/rockyou.txt /home/paxton/Hashes/ntlmhash.txt --max-length=15 --format=nt*

→ **Explanation:** Limits brute force to reasonable sizes, saving time. Good for testing realistic password ranges.

5. Firewall-aware & safe testing

Note: John doesn't interact with firewalls like web tools, but these apply for system logs and monitoring.

- **Simulate slow, realistic logins**

→ **Command:** *john --wordlist=mylist.txt --fork=1 myhashes.txt*

→ **Explanation:** Using one process simulates slow testing and avoids overwhelming your system logs. Default is also 1. When John forks multiple processes (e.g., `--fork=4`), it splits the work (like checking a wordlist or running an incremental attack) across those processes. Each process runs on a separate CPU core, allowing you to check multiple password candidates simultaneously.

- **Use quiet mode**

→ **Command:** *john --quiet myhashes.txt*

→ **Explanation:** Reduces console output and keeps logs minimal for learning.