What is Hashcat?

Hashcat is an open-source, high-performance password recovery tool designed for professionals in cybersecurity, digital forensics, and ethical hacking. It supports a wide range of hash algorithms and leverages the power of CPUs and GPUs to perform efficient password cracking.

Some Features of Hashcat:

1. It is compatible with Windows, Linux, and macOS.
2. We can also use GPUs for faster processing.
3. There are various multiple attack modes which offers various strategies like dictionary, brute-force, and hybrid attacks.
4. Hashcat has an extensive hash support. It supports over 300 hash types, including MD5, SHA variants, bcrypt, and more.

How you can Install Hashcat on Linux

Usually, Hashcat tool comes pre-installed with Kali Linux but if we need to install it write down the given command in the terminal**.**

```
┌──(root㉿Krish)-[/home/kali/Desktop]
└─# sudo apt-get install hashcat
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
hashcat is already the newest version (6.2.6+ds2-1).
hashcat set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 1266 not upgraded.
```

You can confirm the installation of Hashcat by running hashcat command

```
┌──(root㉿Krish)-[/home/kali/Desktop]
└─# hashcat
Usage: hashcat [options] ... hash|hashfile|hccapxfile [dictionary|mask|directory] ...

Try --help for more help.
```

You can run the --help command to bring up the extensive hashcat help guide.

```
┌──(root㉿Krish)-[/home/kali/Desktop]
└─# hashcat --help
hashcat (v6.2.6) starting in help mode

Usage: hashcat [options]... hash|hashfile|hccapxfile [dictionary|mask|directory]...

- [ Options ] -

 Options Short / Long          | Type | Description                                          | Example
===============================+======+======================================================+=======================
 -m, --hash-type               | Num  | Hash-type, references below (otherwise autodetect)   | -m 1000
 -a, --attack-mode             | Num  | Attack-mode, see references below                    | -a 3
 -V, --version                 |      | Print version                                        |
 -h, --help                    |      | Print help                                           |
     --quiet                   |      | Suppress output                                      |
     --hex-charset             |      | Assume charset is given in hex                       |
     --hex-salt                |      | Assume salt is given in hex                          |
     --hex-wordlist            |      | Assume words in wordlist are given in hex            |
     --force                   |      | Ignore warnings                                      |
     --deprecated-check-disable|      | Enable deprecated plugins                            |
     --status                  |      | Enable automatic update of the status screen         |
     --status-json             |      | Enable JSON format for status output                 |
     --status-timer            | Num  | Sets seconds between status screen updates to X      | --status-timer=1
     --stdin-timeout-abort     | Num  | Abort if there is no input from stdin for X seconds  | --stdin-timeout-abort=300
     --machine-readable        |      | Display the status view in a machine-readable format |
     --keep-guessing           |      | Keep guessing the hash after it has been cracked     |
     --self-test-disable       |      | Disable self-test functionality on startup           |
     --loopback                |      | Add new plains to induct directory                   |
     --markov-hcstat2          | File | Specify hcstat2 file to use                          | --markov-hcstat2=my.hcstat2
     --markov-disable          |      | Disables markov-chains, emulates classic brute-force |
     --markov-classic          |      | Enables classic markov-chains, no per-position       |
     --markov-inverse          |      | Enables inverse markov-chains, no per-position       |
 -t, --markov-threshold        | Num  | Threshold X when to stop accepting new markov-chains | -t 50
     --runtime                 | Num  | Abort session after X seconds of runtime             | --runtime=10
     --session                 | Str  | Define specific session name                         | --session=mysession
     --restore                 |      | Restore session from --session                       |
     --restore-disable         |      | Do not write restore file                            |
     --restore-file-path       | File | Specific path to restore file                        | --restore-file-path=x.restore
 -o, --outfile                 | File | Define outfile for recovered hash                    | -o outfile.txt
```

Syntax of hashcat command:

hashcat [options]... hashfile [dictionary|mask|directory]

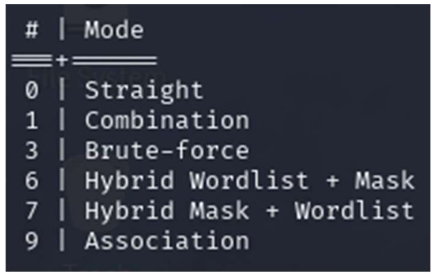
Some options you can use are:

1. -m: Specifies the hash type.

```
    # | Name                                                | Category
======+=====================================================+============
  900 | MD4                                                 | Raw Hash
    0 | MD5                                                 | Raw Hash
  100 | SHA1                                                | Raw Hash
 1300 | SHA2-224                                            | Raw Hash
 1400 | SHA2-256                                            | Raw Hash
10800 | SHA2-384                                            | Raw Hash
 1700 | SHA2-512                                            | Raw Hash
17300 | SHA3-224                                            | Raw Hash
17400 | SHA3-256                                            | Raw Hash
17500 | SHA3-384                                            | Raw Hash
17600 | SHA3-512                                            | Raw Hash
 6000 | RIPEMD-160                                          | Raw Hash
  600 | BLAKE2b-512                                         | Raw Hash
11700 | GOST R 34.11-2012 (Streebog) 256-bit, big-endian    | Raw Hash
11800 | GOST R 34.11-2012 (Streebog) 512-bit, big-endian    | Raw Hash
 6900 | GOST R 34.11-94                                     | Raw Hash
17010 | GPG (AES-128/AES-256 (SHA-1($pass)))                | Raw Hash
 5100 | Half MD5                                            | Raw Hash
17700 | Keccak-224                                          | Raw Hash
17800 | Keccak-256                                          | Raw Hash
```

These are just some of the hash types that you can crack using hashcat. There are over 300 types of hashes that can be cracked using hashcat.

2. -a: Defines the attack mode.



#	Mode
0	Straight
1	Combination
3	Brute-force
6	Hybrid Wordlist + Mask
7	Hybrid Mask + Wordlist
9	Association

We will be talking about these in detail later.

3. -o: Output file for cracked passwords.
4. --show: Displays cracked passwords.

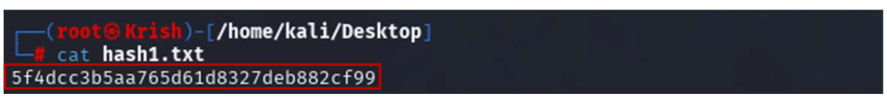
hashcat -m 0 -a 0 hashes.txt rockyou.txt

This command attempts to crack MD5 hashes (-m 0) using a dictionary attack (-a 0) with the rockyou.txt wordlist.

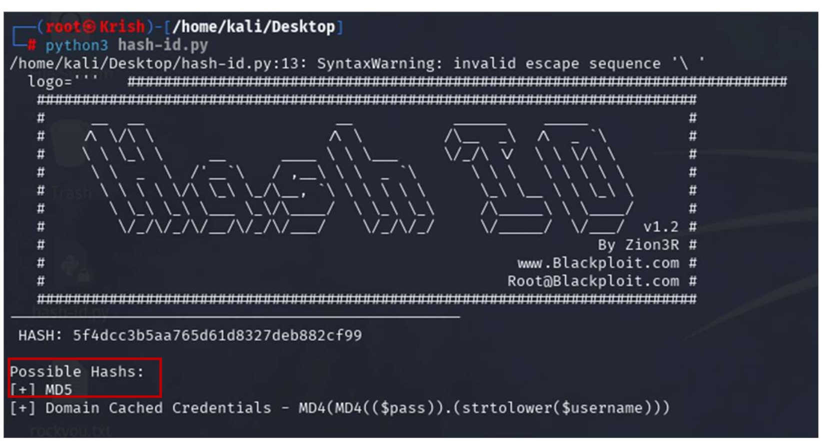
Attack Modes in Hashcat

1. **Straight Attack (Mode 0):**

This is a dictionary attack, where each word in a wordlist is hashed and compared to the target hash. Optionally, rules can be applied to modify each word. Here, you have a wordlist and you want to check if any plain entry present in it matches the hash.



This is the hash that we need to crack. In order to crack it we need to first know what kind of hash it is. For that we can use hash-id. You can download it from GitHub.



Using hash-id, now we know that it is a md5 hash. So now using hashcat we can crack this password.

```
┌──(root㉿Krish)-[/home/kali/Desktop]
└─# hashcat -a 0 -m 0 hash1.txt rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================================================================
* Device #1: cpu-sandybridge-AMD Ryzen 5 5625U with Radeon Graphics, 2913/5890 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1 MB

Dictionary cache built:
* Filename..: rockyou.txt
* Passwords.: 14344391
```

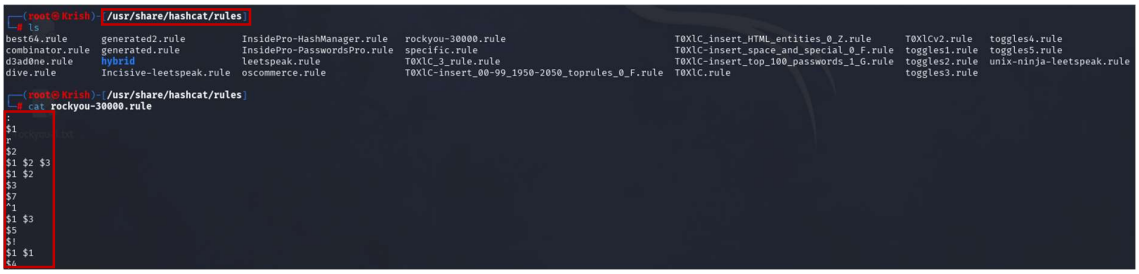
```
5f4dcc3b5aa765d61d8327deb882cf99:password

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: 5f4dcc3b5aa765d61d8327deb882cf99
Time.Started.....: Mon May 26 03:49:47 2025 (0 secs)
Time.Estimated...: Mon May 26 03:49:47 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   609.6 kH/s (0.76ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 2048/14344384 (0.01%)
Rejected.........: 0/2048 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 → lovers1
Hardware.Mon.#1..: Util: 29%

Started: Mon May 26 03:49:45 2025
Stopped: Mon May 26 03:49:49 2025
```

Voila! We got the cracked password which is password itself. There are many other attributes which is displayed by hashcat. To summarize, it says:

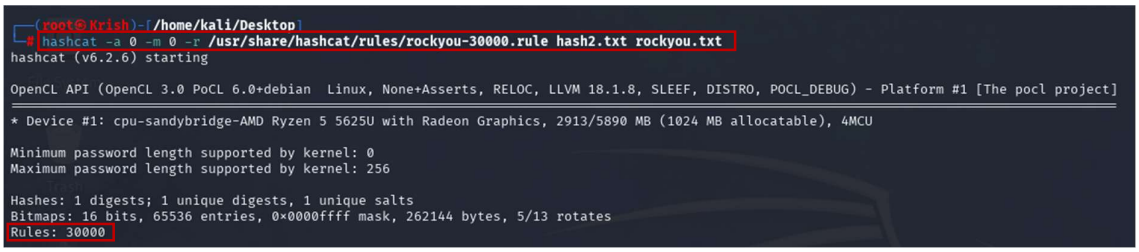
Hashcat loaded our hash and began testing passwords from rockyou.txt. It checked only 2048 entries out of 14M+ before finding password as the match. The hash mode was MD5 (-m 0), and the attack mode was Straight Dictionary (-a 0). My GPU was under low load (29%) and running at approximately 600k hashes/sec.

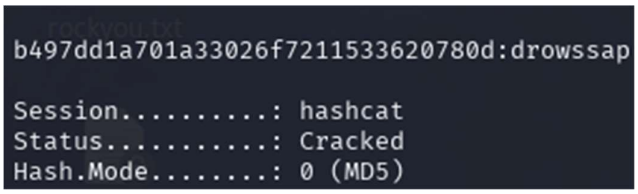
With rules:

You can apply rules to modify each word. For example, you have a simple wordlist with the word password in it but the password we need to crack can have 1 appended to it. So we can apply rules to also test such conditions. You can find already present rules:



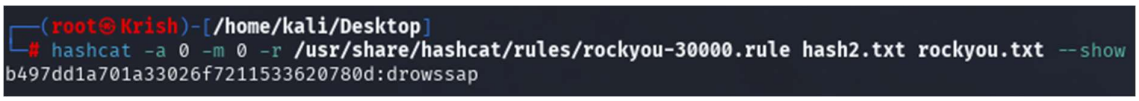
These are the rules already present. Here $1 means that if the wordlist has the word hello, append 1 to it and check if hello1 is the match. r means reverse the string you are checking for and so on. You can also create your own custom rule file.





And just like that we also got our reversed password by applying some rules.

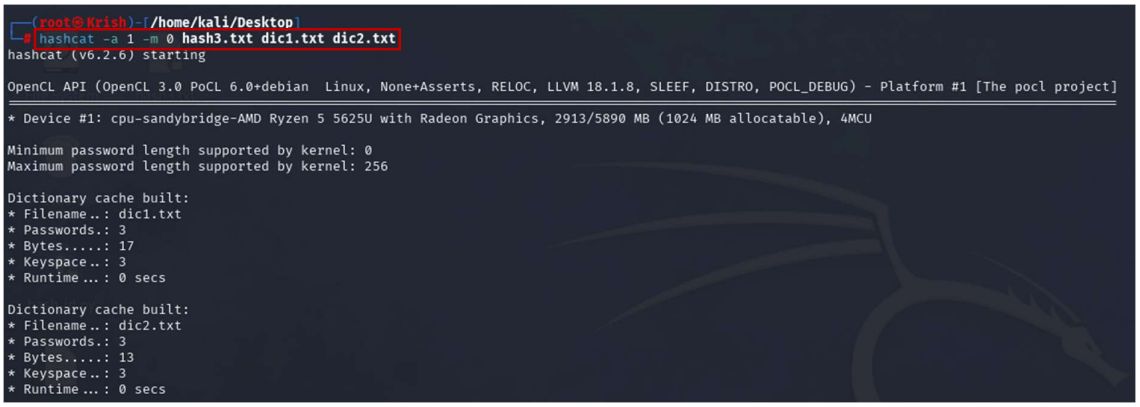
You can also use the --show command to see only the password that has been cracked.

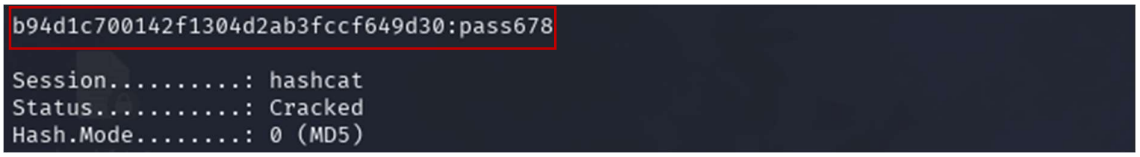


2. Combination Attack (Mode 1)

Here, we take two dictionaries and concatenate each word from the first list with each word from the second list.

Useful for passwords like password123, where the first part and the number are in different lists.





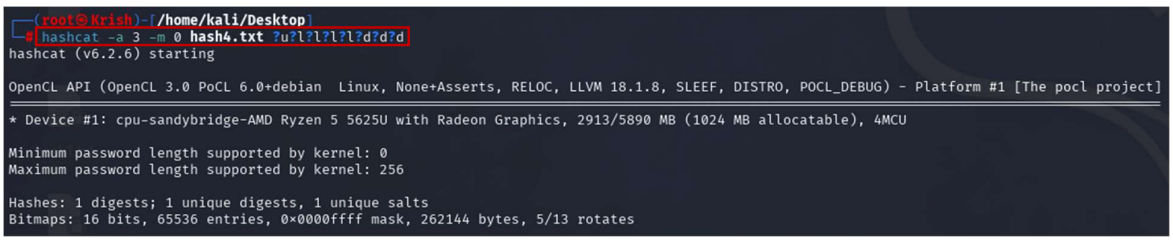
So this is how we were able to find pass678 which was not available in either of the wordlists but could be formed by concatenating the wordlists.

3. **Brute-force Attack (Mode 3)**

Here, we try every combination of characters based on a mask (?a, ?l, ?d, etc.). It is mostly used when we know the password length and/or format (e.g. 6-digit PIN or 8-character password).

Mask Characters:

- ?l: Lowercase
- ?u: Uppercase
- ?d: Digits
- ?s: Special characters
- ?a: All



Tries all passwords with 1uppercase, 4 lowercase letters followed by 3 digits.

```
3d512cab765d1f222a873bc62369a9e9:Krish123

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
```

This is how I was able to crack this password as it may be difficult to find such passwords in a wordlist.

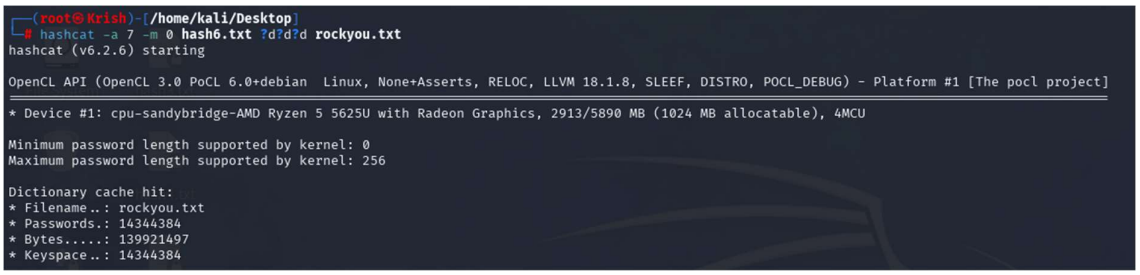
4. **Hybrid Wordlist + Mask (Mode 6)**

This appends a mask to every word in a dictionary. Good for passwords with a base word + suffix (e.g., summer2024). It is useful when most passwords follow a format like word123 or letmein!.


```
faa4b0d94da61f59fb40c180a90fda47:iloveyou123

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: faa4b0d94da61f59fb40c180a90fda47
```

5. **Hybrid Mask + Wordlist (Mode 7)**

Similar to Mode 6 it prepends rather that postpend a mask to every word in a dictionary. Good for passwords like 12summer.




```
87a1bdf256ed35f67dbbb1f054f78039:123summer

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: 87a1bdf256ed35f67dbbb1f054f78039
```

6. Association Attack (Mode 9)

This is a special-purpose mode not commonly used. It attempts to crack hashes by using known pairs (associated plaintext and hash combinations). When you have partial or structured knowledge of the relationship between data (e.g., hash pairs with correlated patterns).

Rarely used in regular password cracking. It's most useful in special forensic cases or capture-the-flag (CTF) puzzles.

hashcat -a 9 -m <mode> hashes.txt associated.txt

You will likely need a custom format for the associated.txt and specialized understanding to use it effectively.

Output Handling

Saving Cracked Passwords:

```
┌──(root㉿Krish)-[/home/kali/Desktop]
└─# hashcat -a 0 -m 0 -o output.txt hash7.txt rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
* Device #1: cpu-sandybridge-AMD Ryzen 5 5625U with Radeon Graphics, 2913/5890 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
```

Displaying Cracked Passwords:

```
┌──(root㉿Krish)-[/home/kali/Desktop]
└─# hashcat -m 0 hash4.txt --show
3d512cab765d1f222a873bc62369a9e9:Krish123
```

Optimizing Performance

In Hashcat, benchmarking refers to measuring the speed at which the program can process hashes. This is typically done to assess the performance of a device (like a GPU or CPU) and to determine the program's potential cracking speed. Benchmarking:

```
┌──(root㉿Krish)-[/home/kali/Desktop]
└─# hashcat -b
hashcat (v6.2.6) starting in benchmark mode

Benchmarking uses hand-optimized kernel code by default.
You can use it in your cracking session by setting the -O option.
Note: Using optimized kernel code limits the maximum supported password length.
To disable the optimized kernel code in benchmark mode, use the -w option.

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
* Device #1: cpu-sandybridge-AMD Ryzen 5 5625U with Radeon Graphics, 2913/5890 MB (1024 MB allocatable), 4MCU

Benchmark relevant options:
* --optimized-kernel-enable

* Hash-Mode 0 (MD5)

Speed.#1.........:   159.6 MH/s (12.73ms) @ Accel:512 Loops:1024 Thr:1 Vec:8

* Hash-Mode 100 (SHA1)

Speed.#1.........: 83114.3 kH/s (23.64ms) @ Accel:512 Loops:1024 Thr:1 Vec:8
```

Workload Profiles:

1. -w 1: Low
2. -w 2: Default
3. -w 3: High
4. -w 4: Nightmare (maximum performance)

hashcat -m 0 -a 0 -w 3 hashes.txt rockyou.txt

Some more additional features

1. Salting Support: Handles hashes with salts.
2. Session Management: Use --session to name sessions and --restore to resume.
3. Multi-Hash Cracking: Crack multiple hashes simultaneously.

One more thing

I found you can just use the following command to create a md5 hashed password for testing in the terminal itself. I was using websites before this. But this is much handier, at least for me.