

# Hashcat for Forensics

Rahul Parmar

Post Graduate Diploma in Cyber Security and Cyber Forensic, Rashtriya Raksha University,  
Lavad, Dahegam, Gandhinagar, Gujarat, India

## 1. Introduction

Hashcat is a password recovery tool. The essential characteristic of Hashcat is that it is a very high-speed brute-forcing tool for passwords, which is achieved through the simultaneous use of all video cards (GPU), as well as central processors (CPU) in the system. It is possible to work in several video cards or video adapters of different manufacturers are installed like AMD, NVIDIA, etc. Hashcat is available for Windows, macOS, and Linux with CPU, GPU, and generic OpenCL support which allows for FPGAs and other accelerator cards. Hashcat supports a large variety of hashing algorithms, including MD4, MD5, SHA-Family, LM Hash, NT Hash, TrueCrypt, and many more.

### Hashcat Features,

- World's fastest password cracker
- World's first and only in-kernel rule engine
- **Open-Source (MIT License)**
- Multi-Platform (Linux, Windows, and macOS)
- Multi-Platform (GPU, CPU, etc., everything that comes with an OpenCL runtime)
- Multi-Devices and Cracking multiple hashes at the same time
- Supports password candidate brain functionality
- Supports distributed cracking **networks** (using overlay)
- Supports **interactive** pause/resume
- Supports hex-salt and hex-charset
- Supports automatic **performance** tuning
- Supports automatic keyspace ordering markov-chains
- Built-in benchmarking system
- Integrated thermal **watchdog**
- **300+ Hash-types** implemented with performance in mind, and many more

## 2. Requirements

#### GPU Driver requirements:

- AMD GPUs on Linux OS requires the "RadeonOpenCompute (ROCm)" Software Platform (3.1 or later version).
- AMD GPUs on Windows OS require "AMD Radeon Adrenalin 2020 Edition" (20.2.2 or later Version)
- Intel CPUs require "OpenCL Runtime for Intel Core and Intel Xeon Processors" (16.1.1 or later version)
- NVIDIA GPUs require "NVIDIA Driver" (440.64 or later) and "CUDA Toolkit" (9.0 or later version)

#### Supported OpenCL runtimes

- AMD
- Apple
- Intel
- NVidia
- POCL
- ROCm

#### Supported OpenCL device types

- GPU
- CPU
- APU

#### Is your card supported?

To ensure that yours NVIDIA video card is supported by stream or CUDA. Visit Below link  
NVIDIA

<http://developer.nvidia.com/cuda-gpus>

### 3. Methodology

#### How to Download Hashcat for windows?

First go to the official Hashcat website: <https://hashcat.net/hashcat/>

Then Download Hashcat Binary: <https://hashcat.net/files/hashcat-6.1.1.7z> and Unzip downloaded file.

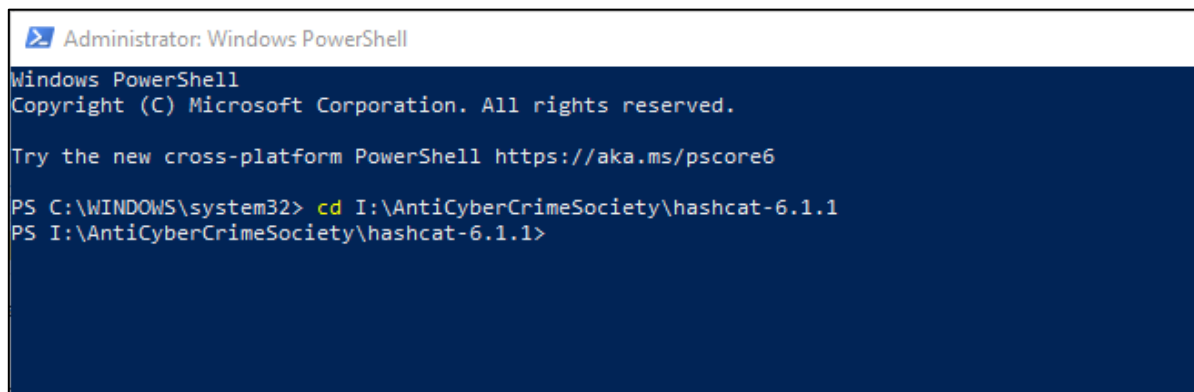
To start the Hashcat for windows, open the command window (or PowerShell) by pressing **Win+x**, and select Windows PowerShell.

For executing file there are two ways:

**First**, you can just drag-n-drop the executable file into the command window. The executable file is **hashcat.exe**

**Second**, you can locate the Hashcat file by changing directory Ex: My Hashcat file is located in I:\OSINT\hashcat-6.1.1\, to change the working directory you can use **cd**, after which the folder to which you specify the desired folder, Command looks like this in my case:

```
cd I:\OSINT\hashcat-6.1.1
```



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

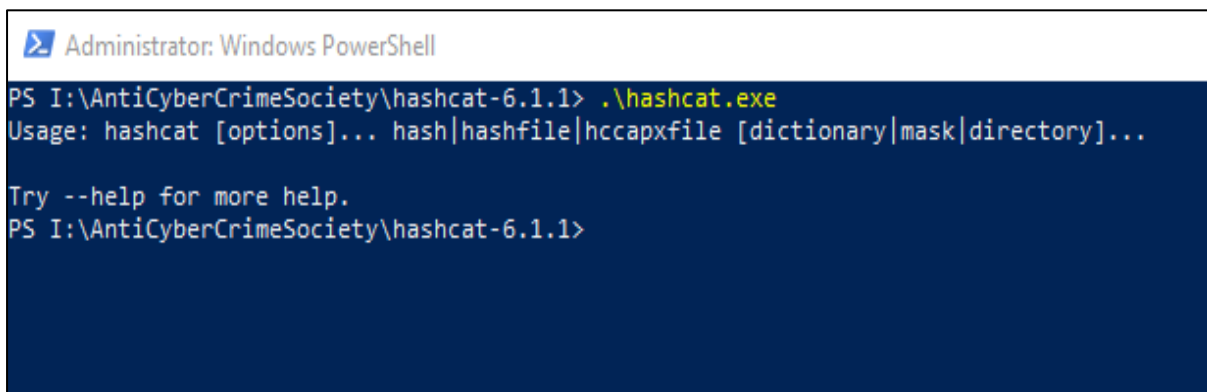
PS C:\WINDOWS\system32> cd I:\AntiCyberCrimeSociety\hashcat-6.1.1
PS I:\AntiCyberCrimeSociety\hashcat-6.1.1>
```

Fig 3.1: Changing directory

In the Above Figure, that folder C:\WINDOWS\system32 is changed to I:\AntiCyberCrimeSociet\hashcat-6.1.1.

Now to start Hashcat you just have to type the name of the executable file indicating the current folder. The current folder is indicated by a period ".", Then you need to put a backslash, command look like this:

```
.\hashcat.exe
```



```
Administrator: Windows PowerShell
PS I:\AntiCyberCrimeSociety\hashcat-6.1.1> .\hashcat.exe
Usage: hashcat [options]... hash[hashfile|hccapxfile [dictionary|mask|directory]...

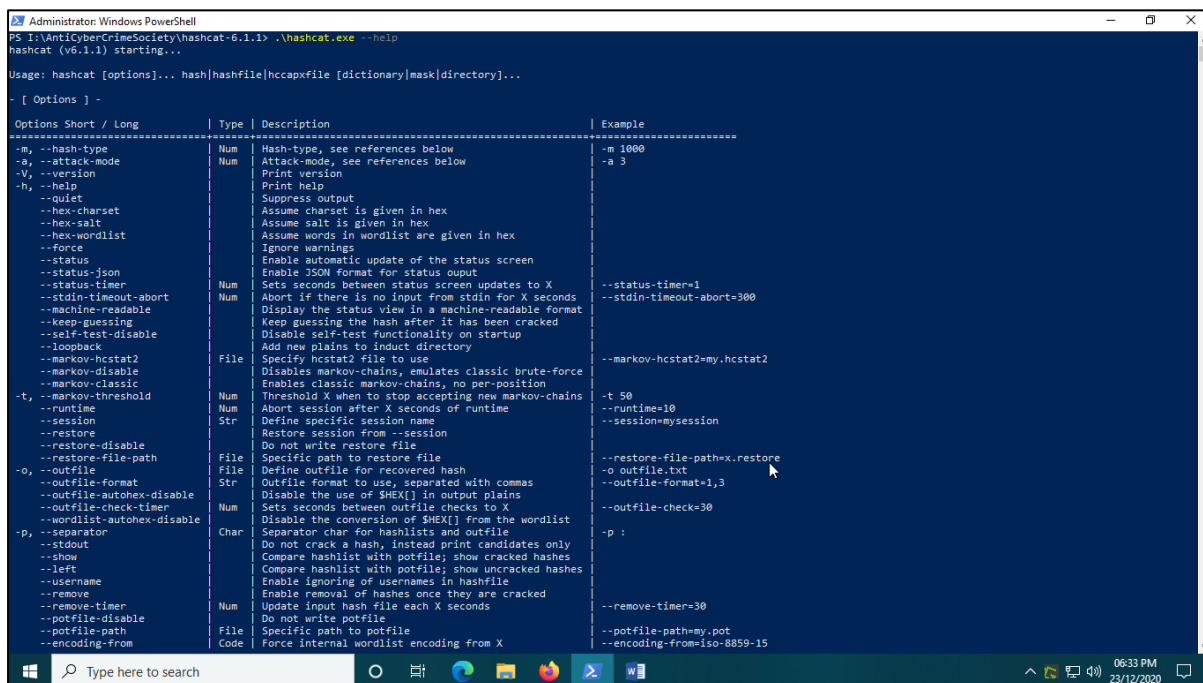
Try --help for more help.
PS I:\AntiCyberCrimeSociety\hashcat-6.1.1>
```

Fig 3.2: Executing Hashcat

Since we have not entered any option, nothing happens and displaying the only hint.

Now let's enter help command by entering **--help** command look like this:

**.\hashcat.exe --help**



```

PS I:\AntiCyberCrimeSociety\hashcat-6.1.1> .\hashcat.exe --help
hashcat (v6.1.1) starting...

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

- [ Options ] -

Options Short / Long | Type | Description | Example
-----
-m, --hash-type       Num | Hash-type, see references below | -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
--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
-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
--outfile-format      | Str | Outfile format to use, separated with commas | --outfile-format=1,3
--outfile-autohex-disable | Disable the use of $HEX[] in output plains
--outfile-check-timer | Num | Sets seconds between outfile checks to X | --outfile-check=30
--wordlist-autohex-disable | Disable the conversion of $HEX[] from the wordlist
-p, --separator       | Char | Separator char for hashlists and outfile | -p :
--stdout             | Do not crack a hash, instead print candidates only
--show               | Compare hashlist with potfile; show cracked hashes
--left              | Compare hashlist with potfile; show uncracked hashes
--username           | Enable ignoring of usernames in hashfile
--remove             | Enable removal of hashes once they are cracked
--remove-timer       | Num | Update input hash file each X seconds | --remove-timer=30
--potfile-disable    | Do not write potfile
--potfile-path       | File | Specific path to potfile | --potfile-path=my.pot
--encoding-from      | Code | Force internal wordlist encoding from X | --encoding-from=iso-8859-15
  
```

Fig: 3.3: Help Command In Hashcat

**Help** Command is displays summaries about a **command's** usage and syntax.

The Basic usage of Hashcat requires a minimum of four arguments.

[A] First is Hash-type, indicate by **-m** or **--hash-type**.

For example SHA1, MD5, BitLocker, etc.

Currently, support 300+ Hash-types, **--help** show full list of hash-type.

For example for SHA1 hash **"-m 100"**.

[B] The second is Attack-mode, indicate by **-a** or **--attack-mode**.

It tells Hashcat how to crack the password.

For example "hashcat -a 0 -m 400 phpass.hash wordlist.dict"

### Core attack modes:

- I. Dictionary attack - trying all words in a list; also called "straight" mode (attack mode 0, indicated by **-a 0**).

- II. Combination attack - concatenating words from multiple wordlists (mode 1, indicated by **-a 1**).
- III. Brute-force attack and Mask attack - trying all characters from given charsets, per position (mode 3, indicated by **-a 3**).
- IV. Hybrid attack - combining wordlists + masks (mode 6, indicated by **-a 6**) and masks + wordlists vice versa (mode 7, indicated by **-a 7**).

#### Other attacks

- I. Rule-based attack - applying rules to words from wordlists; combines with wordlist-based attacks (attack modes 0, 6, and 7).
- II. Toggle-case attack - toggling case of characters; now accomplished with rules.

#### [Attack Modes]

- 0 | Straight
- 1 | Combination
- 3 | Brute-force
- 6 | Hybrid Wordlist + Mask
- 7 | Hybrid Mask + Wordlist

[C] The third is the Hash file, specifying the file name containing the hash value you want to crack.

For example, **C:\Windows\System32\config\SAM** is a hash file of windows.

[D] Forth is specify dictionary, wordlist, mask, or directory.

For wordlist, dictionary and directory just give the path of a file that particular file. For rule **-r (rule file)**.

For example wordlist + rule "hashcat -a 0 -m 0 md5.hash wordlist.dict -r rules/best64.rule"

#### Let's Do some basic hash cracking practical with parrot OS.

For that, I have made one **hash.txt** file that contains some hash value. And for Brute-Force Attack, Dictionary attack, etc. We need one common password file so here I am using **RockYou.txt**. Rockyou.txt is a set of compromised passwords from the social media application developer RockYou.

**Dictionary attack:** For Dictionary attack I have converted one of the most common passwords into a SHA1 algorithm and pasted it into a hash.txt file. My hash.txt and rockyou.txt both the file are in ghostbit folder. For dictionary attack modes indicate by **-a 0** and hash type is SHA1 so **-m -100**. So our command looks like this.

```
hashcat -a 0 -m 100 ../ghostbit/hash.txt ../ghostbit/rockyou.txt
```

```
Dictionary cache built:
* Filename..: ../ghostbit/rockyou.txt
* Passwords.: 14344391
* Bytes.....: 139921497
* Keyspace..: 14344384
* Runtime...: 4 secs

8cb2237d0679ca88db6464eac60da96345513964:12345

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: SHA1
Hash.Target.....: 8cb2237d0679ca88db6464eac60da96345513964
Time.Started.....: Tue Dec 29 09:53:39 2020 (1 sec)
Time.Estimated...: Tue Dec 29 09:53:40 2020 (0 secs)
Guess.Base.....: File (../ghostbit/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 19318 H/s (0.31ms) @ Accel:1024 Loops:1 Thr:1
Recovered.....: 1/1 (100.00%) Digests
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
Candidates.#1....: 123456 -> lovers1

Started: Tue Dec 29 09:52:46 2020
Stopped: Tue Dec 29 09:53:41 2020
└─[ghostbit@parrot]─[~]
```

Fig: 3.4 Directory-Attack

In figure 3.4 hashcat cracked SHA1 hash and showing the password in cleartext. This password contains in rockyou.txt.

This method of cracking hash try all the password from the dictionary.

**Combination attack:** This type of mode is very useful when you have to guess a password from two wordlists or the same wordlist but two different words as a password. For this, we are using the same **hash.txt** file for hash value. And dictionary I am creating a new file called **password.txt** which contain a small amount of common password.

For testing purposes, I have put 10 words in password.txt like this. princess1, 12345, mylove, ilovedog, babygirl, pokemon, qwerty123, jessica, daniel, snowman.

For Combination attack indicate by **-a 1** and hash-type is MD5 so **-m 0**. Password.txt containing in ghostbit folder so the command looks like this.

```
hashcat -a 1 -m 0 ../ghostbit/hash.txt ../ghostbit/password.txt ../ghostbit/password.txt
```

```
85a27d0a174f77352b2ee738d5b6fe96:babygirljessica
Session.....: hashcat
Status.....: Exhausted
Hash.Name.....: MD5
Hash.Target.....: ../ghostbit/hash.txt
Time.Started.....: Tue Dec 29 11:11:05 2020 (0 secs)
Time.Estimated...: Tue Dec 29 11:11:05 2020 (0 secs)
Guess.Base.....: File (../ghostbit/password.txt), Left Side
Guess.Mod.....: File (../ghostbit/password.txt), Right Side
Speed.#1.....: 18844 H/s (0.09ms) @ Accel:1024 Loops:22 Thr:1 Vec:8
Recovered.....: 7/8 (87.50%) Digests
Progress.....: 484/484 (100.00%)
Rejected.....: 0/484 (0.00%)
Restore.Point....: 22/22 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-22 Iteration:0-22
Candidates.#1...: samsam -> jessicajessica
Started: Tue Dec 29 11:11:03 2020
Stopped: Tue Dec 29 11:11:07 2020
[x]-[ghostbit@parrot]-[~]
$password.txt selected (103 bytes). Free space: 17.2 GB
```

Fig: 3.5 Combination attack

If you noticed we don't have one-word **babygirljessica** in our password.txt but we have babygirl and jessica both different words. So this attack mode tries matching both the words from the same wordlist file or we can you two different wordlist files.

**Brute-force attack and Mask attack:** Hashcat is brute-forcing all hash with given wordlist or directory but this attack mode has some extra functionality. This attack mode trying all characters from given charsets, per position.

### Charset

- l | abcdefghijklmnopqrstuvwxyz
- u | ABCDEFGHIJKLMNOPQRSTUVWXYZ
- d | 0123456789
- h | 0123456789abcdef
- H | 0123456789ABCDEF
- s | !"#\$%&'()\*+,-./:;<=>?@[\]^\_`{|}~
- a | ?l?u?d?s
- b | 0x00 - 0xff

For Example, if you know that password is in lower case and four characters long then using this attack mode only try lower case characters. We can define lower case four characters with "?l?l?l?l". If a password is five characters long and in uppercase then we can use "?u?u?u?u?u". We can use multi charset also like "?u?l?l?u?s"



Let's do practical for three characters password but don't know case order. So command look like this.

```
hashcat -a 3 -m 0 ../ghostbit/hash.txt ?a?a?a
```

```
88ec7b741e9c3ccac7b61b1301ba11e8:BC1
5f184db5aa59fee4e952d5d5ac87471d:Rf3
Approaching final keyspace - workload adjusted.
82E12D1BB1DA4C023024AC64387C0E0B
Session.....: hashcat 9B7AC33BFAC
Status.....: Exhausted 4B6989349
Hash.Name.....: MD5
Hash.Target.....: ../ghostbit/hash.txt
Time.Started.....: Tue Dec 29 10:49:38 2020 (0 secs)
Time.Estimated...: Tue Dec 29 10:49:38 2020 (0 secs)
Guess.Mask.....: ?a?a?a [3]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 32699.4 kH/s (3.13ms) @ Accel:1024 Loops:95 Thr:1 Vec:8
Recovered.....: 6/7 (85.71%) Digests
Progress.....: 857375/857375 (100.00%)
Rejected.....: 0/857375 (0.00%)
Restore.Point....: 9025/9025 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-95 Iteration:0-95
Candidates.#1....: st? -> ~

Started: Tue Dec 29 10:49:32 2020
Stopped: Tue Dec 29 10:49:39 2020
[x]-[ghostbit@parrot]-[~]
$
```

Fig: 3.6 Brute-Force attacks and mask attack

Here, I have given two hash values in the hash.txt file. We have used ?a?a?a so its brute force with all the three characters with all charset. This attack mode is very useful when we know only password length or charset.

### Rule-based attack:

Rule base attack is more complicated than other attacks. In this attack, you have to create your rule file. What is the rule base attack? Let me explain to you with some examples.

Consider that you have a wordlist contains some word below:

jessica  
qwerty  
joe  
emma  
password



Now if you want to try the above password with some sort of pattern like after every word you want to add "123" in the end then your list is now like below:

jessica  
Jessica123  
qwerty  
qwerty123  
joe  
joe123  
emma  
emma123  
password  
password123

Now you want that all word in Uppercase then now your wordlist is like below:

jessica  
Jessica123  
JESSICA  
qwerty  
qwerty123  
QWERT  
joe  
joe123  
JOE  
emma  
emma123  
EMMA  
password  
password123  
PASSWORD

Now we have known that what is rule-based attack and how a rule works. But how to make a rule for that I am showing you some common rule functions which we can use store in our rule file.

Name	Function	Description	Example Rule	Input Word	Output Word
Nothing	:	Do nothing (passthrough)	:	p@ssW0rd	p@ssW0rd
Uppercase	u	Uppercase all letters	u	p@ssW0rd	P@SSWORD
Lowercase	l	Lowercase all letters	l	p@ssW0rd	p@ssword

Capitalize	c	Capitalize the first letter and lower the rest	c	p@ssW0rd	P@ssword
Toggle Case	t	Toggle the case of all characters in a word.	t	p@ssW0rd	P@SSw0RD

Let's do practical, in this, I am using **hob064.rule** file. You can download that from Github. I have created one small wordlist **pass.txt** which contains the top 100 common passwords in lowercase. I am here using Straight attack mode and my hash is md5 so the command looks like this:

```
hashcat -a 0 -m 0 -r ../ghostbit/hob064.rule ../ghostbit/hash.txt ../ghostbit/pass.txt
```

```
6814670f6479e391883cf8d6f588754e:SUNSHINE

Session.....: hashcat
Status.....: Exhausted
Hash.Name.....: MD5
Hash.Target.....: ../ghostbit/hash.txt
Time.Started.....: Wed Dec 30 10:54:06 2020 (0 secs)
Time.Estimated...: Wed Dec 30 10:54:06 2020 (0 secs)
Guess.Base.....: File (../ghostbit/pass.txt)
Guess.Mod.....: Rules (../ghostbit/hob064.rule)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 6007.1 kH/s (0.78ms) @ Accel:512 Loops:64 Thr:1 Vec:8
Recovered.....: 9/10 (90.00%) Digests
Progress.....: 6848/6848 (100.00%)
Rejected.....: 0/6848 (0.00%)
Restore.Point....: 107/107 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-64 Iteration:0-64
Candidates.#1....: jessica -> 16

Started: Wed Dec 30 10:54:05 2020
Stopped: Wed Dec 30 10:54:08 2020
└─[x]─[ghostbit@parrot]─[~]
```

Fig: 3.7 Rule-Based Attack

In figure 3.7 you can see that it converted our hash. Although we don't have uppercase in our wordlist.

These are some common rule file from GitHub:

<https://github.com/praetorian-inc/Hob0Rules/blob/master/hob064.rule>

[https://github.com/NotSoSecure/password\\_cracking\\_rules](https://github.com/NotSoSecure/password_cracking_rules)

[https://github.com/wpatookit/Hashcat-Rules-Reference/blob/master/hashcat\\_rules.txt](https://github.com/wpatookit/Hashcat-Rules-Reference/blob/master/hashcat_rules.txt)

One more feature that Hashcat appends all the cracked passwords in a **potfile** which you can see in the directory.

### WinRAR Cracking

For this, I have created one password-protected **Credential.rar**. For cracking the RAR file, we need a RAR file hash. Which we have to extract from the RAR file with help of **rar2john**. For that open terminal and enter **rar2hash filename**. Below you can see that rar2hash extract hash from the RAR file.

```
[ghostbit@parrot]-[~]  
$rar2john ../ghostbit/Credential.rar  
../ghostbit/Credential.rar:$rar5$16$e6ea9c52f3ff196d2ec91a8d86165928$15$be50adb2  
129db4884dca5e544b9729d4$8$90633517ad5b4285
```

Fig: 3.8 hash extraction

After extracting the hash from the RAR file. copy and paste in our Hashcat command with separator.

Here we are cracking WinRAR password so hash-type indicates with **-m 13000** and we are using Dictionary attack so attack mode becomes **-a 0** final command look like this.

```
Hashcat -m 13000 -a 0  
'$rar5$16$e6ea9c52f3ff196d2ec91a8d86165928$15$be50adb2129db4884dca5e544b9729  
d4$8$90633517ad5b4285' ../ghostbit/rockyou.txt
```

```
$rar5$l6$e6ea9c52f3ff196d2ec91a8d86165928$15$be50adb2129db4884dca5e544b9729d4$8$90633517ad5b4285:abc123
Session.....: hashcat
Status.....: Cracked
Hash.Name.....: RAR5
Hash.Target.....: $rar5$l6$e6ea9c52f3ff196d2ec91a8d86165928$15$be50ad...5b4285
Time.Started.....: Wed Dec 30 10:35:41 2020 (3 secs)
Time.Estimated...: Wed Dec 30 10:35:44 2020 (0 secs)
Guess.Base.....: File (./ghostbit/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 54 H/s (7.23ms) @ Accel:32 Loops:512 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 64/14344384 (0.00%)
Rejected.....: 0/64 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:32768-32799
Candidates.#1....: 123456 -> charlie

Started: Wed Dec 30 10:35:12 2020
Stopped: Wed Dec 30 10:35:46 2020
[ghostbit@parrot]-[-] selected (139.9 MB), Free space: 17.1 GB
$
```

Fig: 3.9 WinRAR Cracking.

In Figure: 3.9 you will find out the hash type from Hash.name also and in status, you can see its show cracked successfully. Like this, we can extract zip hash from the zip file with help of **zip2john**. After that, we only have to change the hash type in Hashcat.

These are some basic techniques for recovering the password from the hash. For more hash-type, you can go through the help command with **-help** and learn more about Hashcat. Here we have discussed some attacks. You can try more attacks like Hybrid attack with combining dictionary and mask attack you can set rule also in that.

## 4. Conclusion

Hashcat provides the fastest and most advanced password recovering tool which contains over 300+ highly-optimized hashing algorithms and different attack modes like a dictionary attack, mask attack, rule base attack, and many more. Which support CPUs, GPUs, and other hardware accelerators on Linux, Windows, and macOS. Hashcat is very useful when you have some guesses on passwords like it's in lowercase or uppercase. Then with a rule base attack, we can crack passwords easily. Hashcat is adding more and more new hashes and more features in it. In latest version of Hashcat 6.1.0 added Apple Keychain and XMPP SCRAM hashes also fixing in some attack mode like large mask in **-a 6** attack mode.

## 5. References

- [1] Official website of Hashcat [\*https://hashcat.net/hashcat/\*](https://hashcat.net/hashcat/)
- [2] Wikipedia [\*https://en.wikipedia.org/wiki/Hashcat\*](https://en.wikipedia.org/wiki/Hashcat)
- [3] Hash Crack: Password Cracking Manual Book by Joshua Picolet
- [4] [\*https://www.unix-ninja.com/p/A\\_guide\\_to\\_password\\_cracking\\_with\\_Hashcat\*](https://www.unix-ninja.com/p/A_guide_to_password_cracking_with_Hashcat)
- [5] Confessions of a crypto cluster operator, Dustin Heywood, Derbycon 2015
- [6] Hashcat State of the Union, EvilMog, Derbycon 2016