Hashcat Implementation

BY
16010121155 - Kanishka Raikwar
16010121188 - Navya Shrivastava
16010121209 - Surabhi Venkatesh
TY - C Division

Disclaimer: Hashcat should be used responsibly and ethically. Unauthorized use of Hashcat to crack passwords is illegal and unethical. Always get permission before scanning, brute-forcing, or exploiting a system.

Introduction

Hashcat is a well-known password cracker, also sometimes referred to as a password recovery tool. It is designed to crack even the most complex passwords by allowing a specific password to be broken in several ways, combined with flexibility and speed.

Hashcat was developed by Jens "atom" Steube and is a command-line based password recovery tool that utilizes the computing power of GPUs (Graphics Processing Units) to crack passwords. It supports a wide range of hash types, including MD5, SHA-1, SHA-256, bcrypt, and many more.

Hashcat uses pre-computed dictionaries, rainbow tables, and even a brute-force method to find an easy and reliable way to crack passwords. It's a powerful tool that helps to crack password hashes and supports most hashing algorithms.

The importance and relevance of Hashcat in the field of cybersecurity are as follows:

1. **Password Cracking:** Hashcat can crack passwords by attempting different combinations until it finds the correct one. It supports various attack modes, allowing users to optimize the process based on available information about the password.
2. **Security Audits:** Hashcat is commonly used by security professionals to assess the strength of passwords in an organization. By cracking passwords, they can identify weak or easily guessable passwords, helping organizations improve their security posture.
3. **Forensics and Incident Response:** In digital forensics investigations, Hashcat can be used to recover passwords from compromised systems or encrypted files. This can aid in uncovering crucial evidence and gaining access to protected data.

4. **Research and Development:** Hashcat is also used by researchers and developers to evaluate the security of cryptographic algorithms and hash functions. By testing the resilience of various hashing mechanisms, they can identify vulnerabilities and propose more robust solutions.

Features/Characteristics

Hashcat has several key features that make it highly effective and versatile:

1. **Open Source:** Hashcat is fully open source, which means its source code is freely available for anyone to view, modify, and distribute.
2. **Support for Multiple Hashing Algorithms:** Hashcat supports more than 200 different hashing algorithms, including MD5, SHA1, bcrypt, and many others. This wide range of support allows it to crack a variety of password hashes.
3. **Cross-Platform Compatibility:** Hashcat is compatible with multiple operating systems, including Windows, Linux, and Mac. This makes it a versatile tool that can be used in various environments.
4. **Parallel Processing:** Hashcat has the ability to crack multiple hashes simultaneously. This feature significantly speeds up the password cracking process.
5. **Built-in Benchmarking System:** Hashcat includes a built-in benchmarking system that allows users to measure the performance of their hardware. This can be useful for optimizing the password cracking process.
6. **GPU-Accelerated Cracking:** Hashcat leverages the power of GPUs (Graphics Processing Units) to accelerate the password cracking process. This allows it to perform operations much faster than CPU-based cracking.
7. **Multi-Threaded:** Hashcat is multi-threaded, which means it can perform multiple operations at the same time, further speeding up the password cracking process.
8. **Attack Modes:** Hashcat supports five unique modes of attack for over 300 highly-optimized hashing algorithms. These include brute force, dictionary, and hybrid attacks.

Hashcat supports several types of attacks to crack password hashes:

- **Dictionary Attack (-a 0):** In this mode, Hashcat uses a wordlist (a file containing a collection of words) to check against the hash/hashes. This is the simplest form of attack, where the attacker goes through all the words in a prearranged list.
- **Combination Attack (-a 1):** This attack mode combines two dictionaries to make valid passwords. For example, if one dictionary contains common passwords and another

contains common suffixes, a combination attack would generate passwords consisting of a word from the first dictionary followed by a word from the second.

- **Brute Force Attack (-a 3):** In a brute force attack, Hashcat tries all possible combinations of characters up to a certain length. This is the most comprehensive type of attack, but it can be time-consuming if the password is long or complex.
- **Mask Attack (-a 3):** A mask attack is a more targeted form of a brute force attack. Instead of trying all possible combinations, a mask attack only tries combinations that match a certain pattern. For example, if you know that a password is eight characters long and ends with a digit, you can use a mask attack to only try combinations that fit this pattern.
- **Hybrid Attack (-a 6 or 7):** A hybrid attack combines a dictionary attack with a brute force attack3. For example, it might start with a word from a dictionary and then append or prepend characters using brute force.
- **Rule-Based Attack:** In a rule-based attack, Hashcat applies a set of transformation rules to each word in a dictionary. For example, a rule might specify to replace all instances of 'a' with '@' or to append a digit to the end of the word.

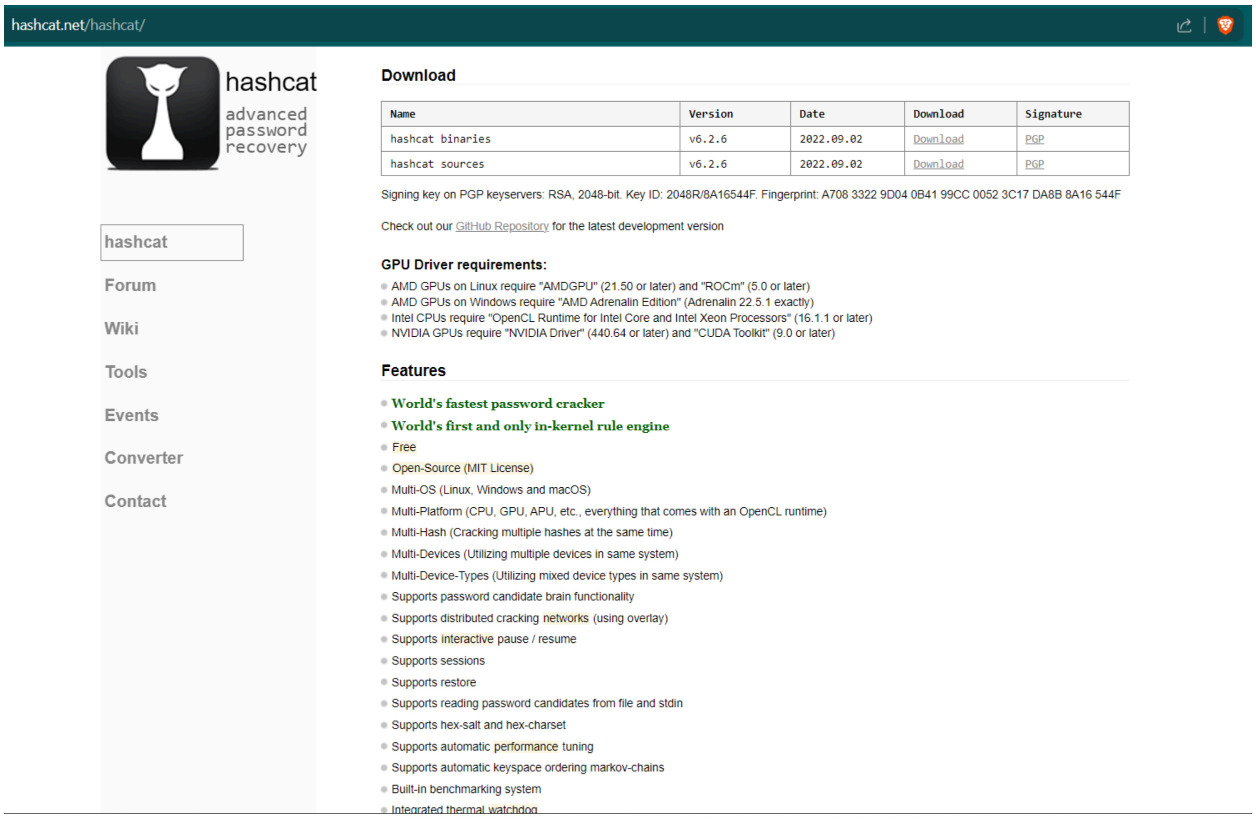
Each of these attack modes has its strengths and weaknesses, and the best mode to use depends on what you know about the password you're trying to crack.

Hashcat's effectiveness lies in its comprehensive features. As an open-source tool, it benefits from global contributions, staying current with password cracking advancements. It supports multiple hashing algorithms, making it versatile across various systems. Its cross-platform compatibility increases its usability. Hashcat's parallel processing and GPU-accelerated cracking enhance its speed and efficiency. The built-in benchmarking system helps users optimize the process. Its multi-threaded nature allows for simultaneous operations, speeding up the process. Finally, its variety of attack modes, including dictionary, combination, brute force, mask, hybrid, and rule-based attacks, allows it to adapt to different scenarios, enhancing its effectiveness in password recovery and cybersecurity applications.

Methodology

1. Installation

Hashcat Binaries can be downloaded from the site: hashcat.net



hashcat.net/hashcat/

Download

Name	Version	Date	Download	Signature
hashcat binaries	v6.2.6	2022.09.02	Download	PGP
hashcat sources	v6.2.6	2022.09.02	Download	PGP

Signing key on PGP keyservers: RSA, 2048-bit. Key ID: 2048R/8A16544F. Fingerprint: A708 3322 9D04 0B41 99CC 0052 3C17 DA8B 8A16 544F

Check out our GitHub Repository for the latest development version

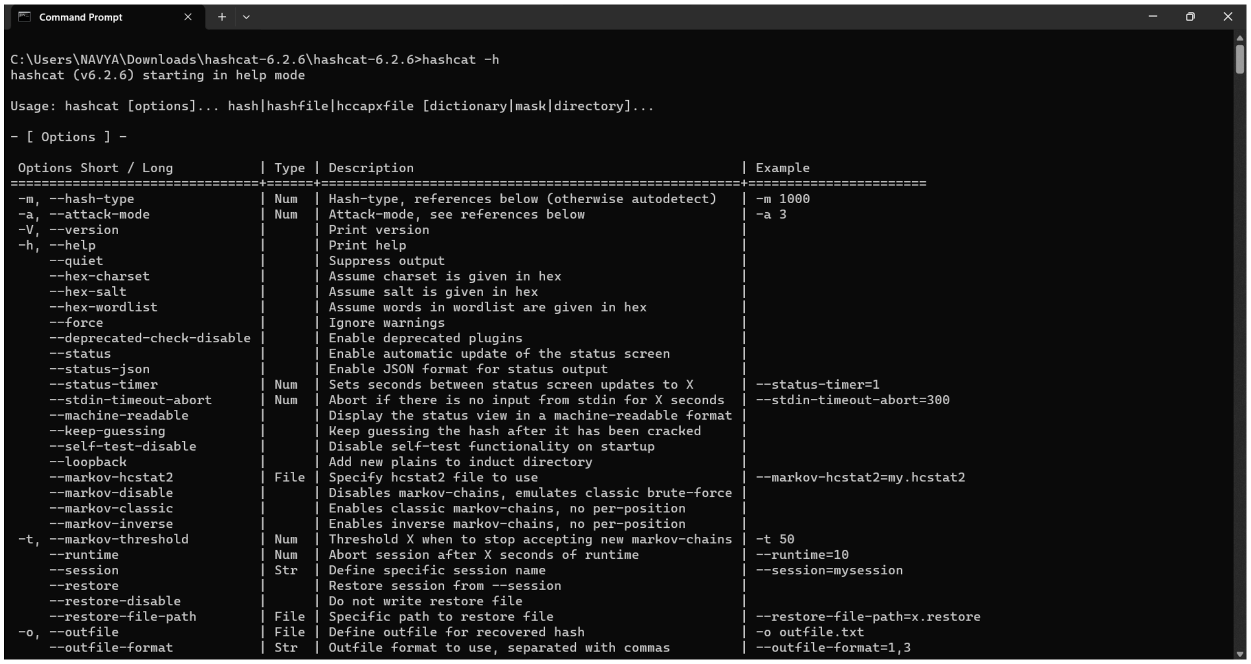
GPU Driver requirements:

- AMD GPUs on Linux require "AMDGPU" (21.50 or later) and "ROCm" (5.0 or later)
- AMD GPUs on Windows require "AMD Adrenalin Edition" (Adrenalin 22.5.1 exactly)
- Intel CPUs require "OpenCL Runtime for Intel Core and Intel Xeon Processors" (16.1.1 or later)
- NVIDIA GPUs require "NVIDIA Driver" (440.64 or later) and "CUDA Toolkit" (9.0 or later)

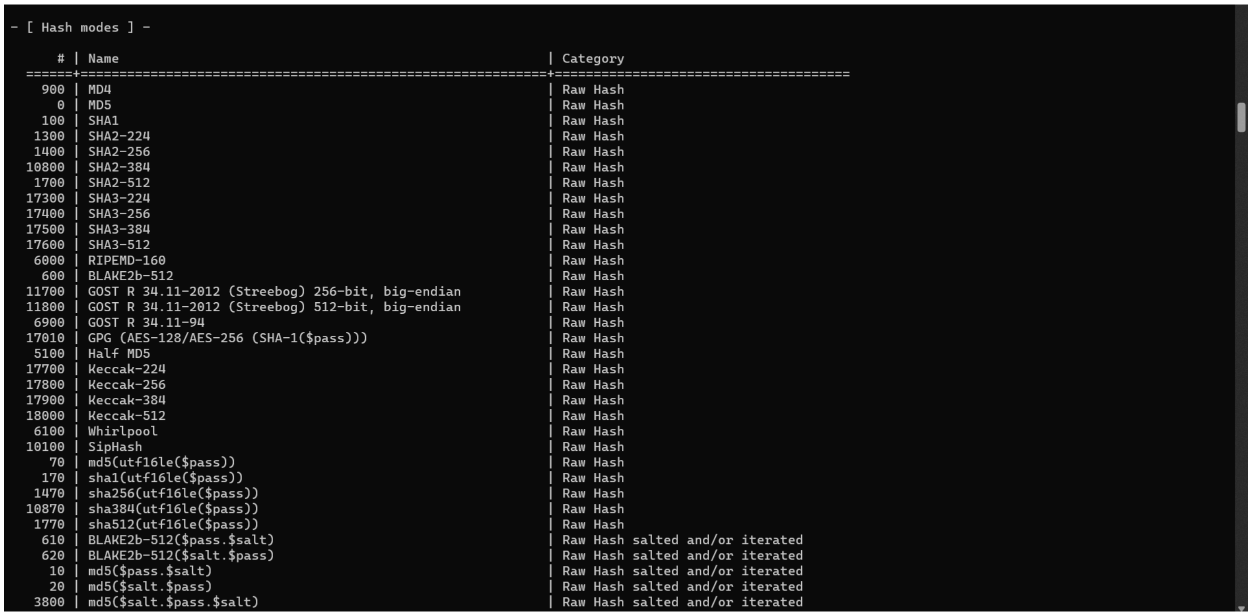
Features

- **World's fastest password cracker**
- **World's first and only in-kernel rule engine**
- Free
- Open-Source (MIT License)
- Multi-OS (Linux, Windows and macOS)
- Multi-Platform (CPU, GPU, APU, etc., everything that comes with an OpenCL runtime)
- Multi-Hash (Cracking multiple hashes at the same time)
- Multi-Devices (Utilizing multiple devices in same system)
- Multi-Device-Types (Utilizing mixed device types in same system)
- Supports password candidate brain functionality
- Supports distributed cracking networks (using overlay)
- Supports interactive pause / resume
- Supports sessions
- Supports restore
- Supports reading password candidates from file and stdin
- Supports hex-salt and hex-charset
- Supports automatic performance tuning
- Supports automatic keyspace ordering markov-chains
- Built-in benchmarking system
- Integrated thermal watchdog

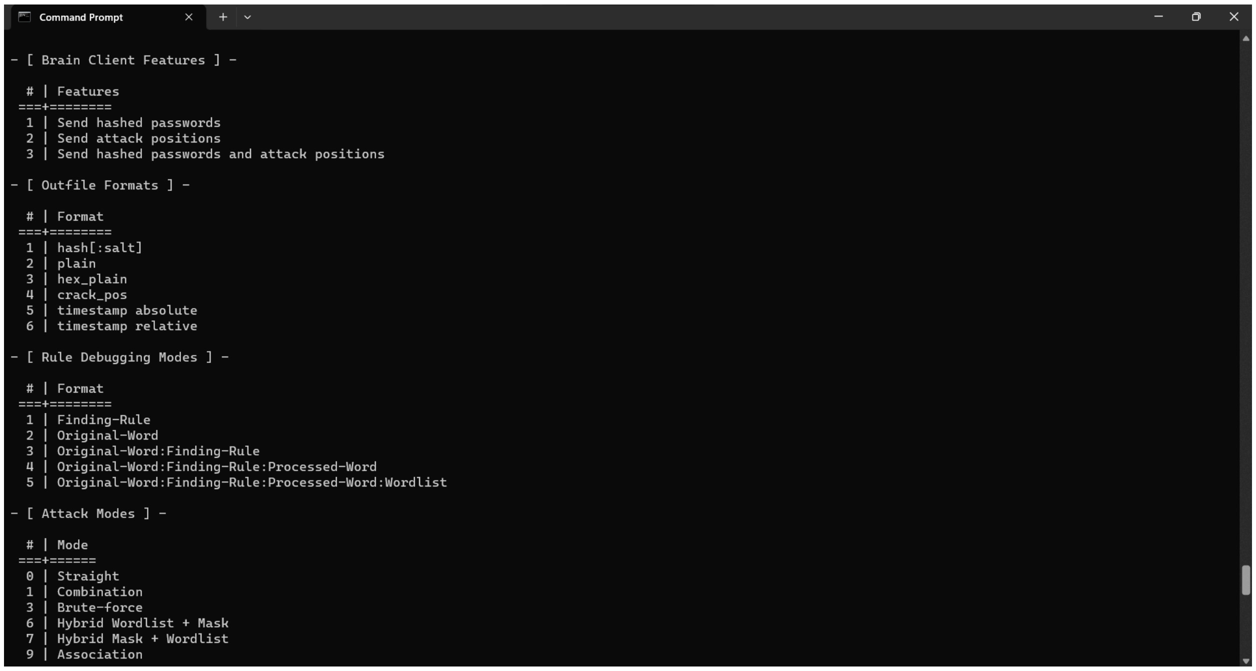
2. Hashcat help command



```
C:\Users\NAVYA\Downloads\hashcat-6.2.6\hashcat-6.2.6>hashcat -h
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
     --outfile-format           | Str  | Outfile format to use, separated with commas         | --outfile-format=1,3
```



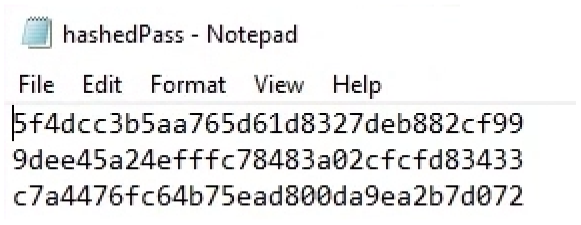
```
- [ Hash modes ] -

      # | 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($pass)))                       | Raw Hash
   5100 | Half MD5                                                   | Raw Hash
  17700 | Keccak-224                                                 | Raw Hash
  17800 | Keccak-256                                                 | Raw Hash
  17900 | Keccak-384                                                 | Raw Hash
  18000 | Keccak-512                                                 | Raw Hash
   6100 | Whirlpool                                                  | Raw Hash
  10100 | SipHash                                                    | Raw Hash
     70 | md5(utf16le($pass))                                        | Raw Hash
    170 | sha1(utf16le($pass))                                       | Raw Hash
   1470 | sha256(utf16le($pass))                                     | Raw Hash
  10870 | sha384(utf16le($pass))                                     | Raw Hash
   1770 | sha512(utf16le($pass))                                     | Raw Hash
    610 | BLAKE2b-512($pass.$salt)                                   | Raw Hash salted and/or iterated
    620 | BLAKE2b-512($salt.$pass)                                   | Raw Hash salted and/or iterated
     10 | md5($pass.$salt)                                           | Raw Hash salted and/or iterated
     20 | md5($salt.$pass)                                           | Raw Hash salted and/or iterated
   3800 | md5($salt.$pass.$salt)                                     | Raw Hash salted and/or iterated
```



```
- [ Brain Client Features ] -

  # | Features
 ===+========
  1 | Send hashed passwords
  2 | Send attack positions
  3 | Send hashed passwords and attack positions

- [ Outfile Formats ] -

  # | Format
 ===+========
  1 | hash[:salt]
  2 | plain
  3 | hex_plain
  4 | crack_pos
  5 | timestamp absolute
  6 | timestamp relative

- [ Rule Debugging Modes ] -

  # | Format
 ===+========
  1 | Finding-Rule
  2 | Original-Word
  3 | Original-Word:Finding-Rule
  4 | Original-Word:Finding-Rule:Processed-Word
  5 | Original-Word:Finding-Rule:Processed-Word:Wordlist

- [ Attack Modes ] -

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

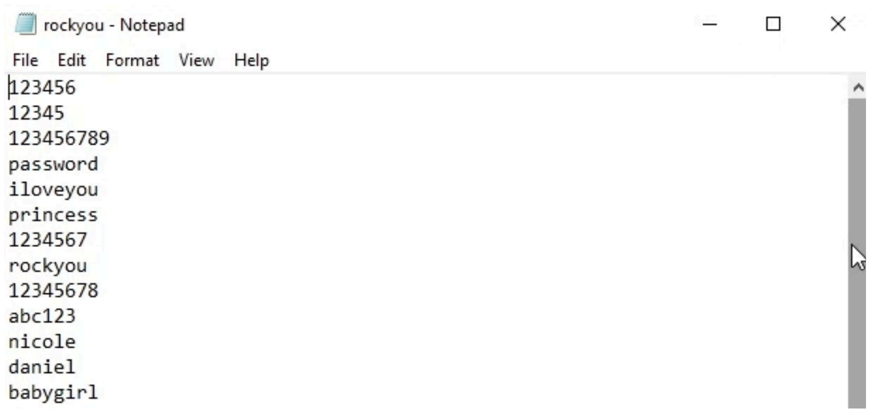
3. Dictionary Attack

Create a file with hashed passwords that you wish to crack:



```
5f4dcc3b5aa765d61d8327deb882cf99
9dee45a24efffc78483a02cfcfd83433
c7a4476fc64b75ead800da9ea2b7d072
```

Create a sample wordlist:

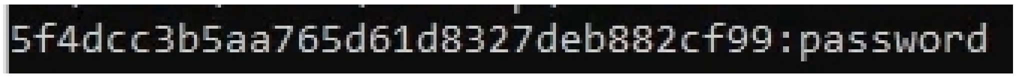


```
123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678
abc123
nicole
daniel
babygirl
```

Run the following command on the terminal:

```
hashcat -m0 -a0 hashedPass.txt rockyou.txt --show
```

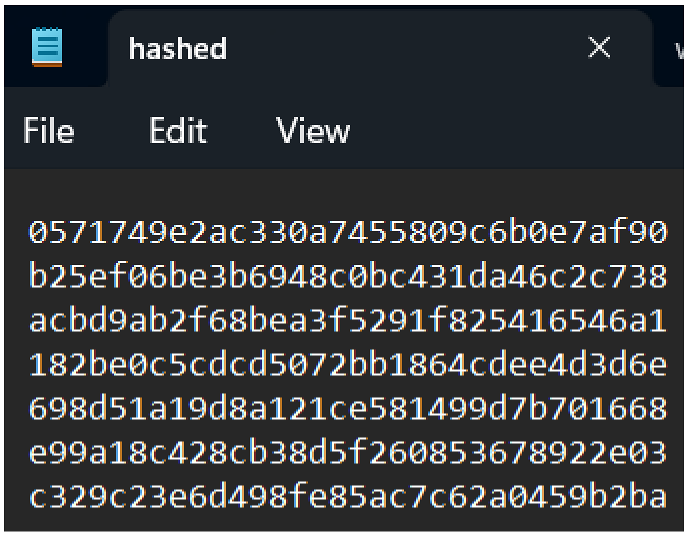
Output:



```
5f4dcc3b5aa765d61d8327deb882cf99:password
```

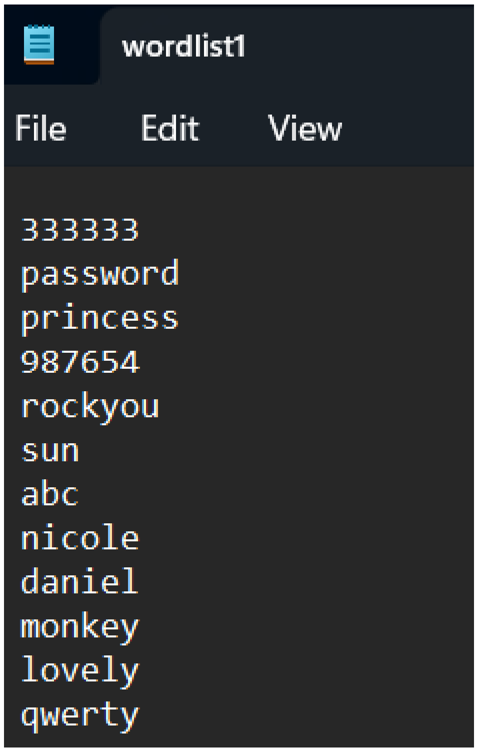
4. **Combination Attack**

Create a file with hashed passwords that you wish to crack:

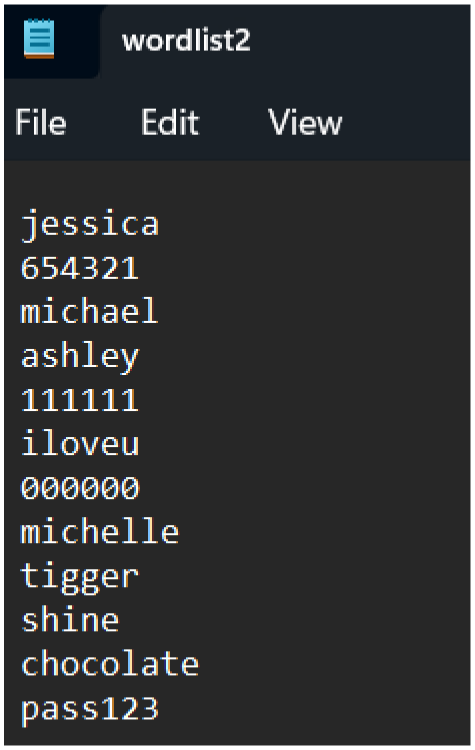


```
0571749e2ac330a7455809c6b0e7af90
b25ef06be3b6948c0bc431da46c2c738
acbd9ab2f68bea3f5291f825416546a1
182be0c5cdcd5072bb1864cdee4d3d6e
698d51a19d8a121ce581499d7b701668
e99a18c428cb38d5f260853678922e03
c329c23e6d498fe85ac7c62a0459b2ba
```

Create two wordlists:



```
333333
password
princess
987654
rockyou
sun
abc
nicole
daniel
monkey
lovely
qwerty
```

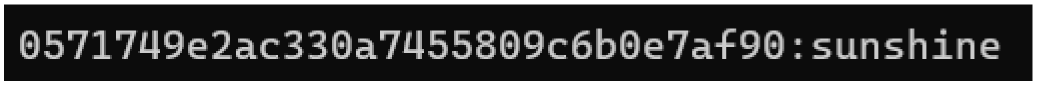


```
jessica
654321
michael
ashley
111111
iloveu
000000
michelle
tigger
shine
chocolate
pass123
```

Run the following command on the terminal:

```
hashcat -a1 -m0 hashed.txt wordlist1.txt wordlist2.txt
```

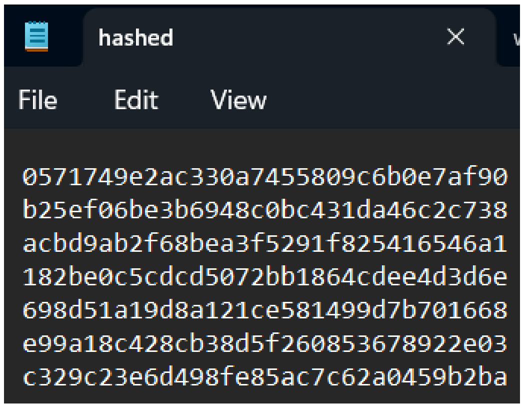
Output:



```
0571749e2ac330a7455809c6b0e7af90:sunshine
```

5. **Brute Force Attack (Using a mask)**

Create a file with hashed passwords that you wish to crack:

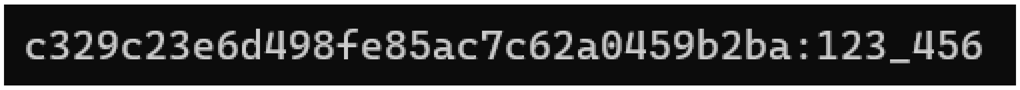


```
0571749e2ac330a7455809c6b0e7af90
b25ef06be3b6948c0bc431da46c2c738
acbd9ab2f68bea3f5291f825416546a1
182be0c5cdcd5072bb1864cdee4d3d6e
698d51a19d8a121ce581499d7b701668
e99a18c428cb38d5f260853678922e03
c329c23e6d498fe85ac7c62a0459b2ba
```

Run the following command on the terminal:

```
hashcat -a3 -m0 hashed.txt ?d?d?d?s?d?d?d
```

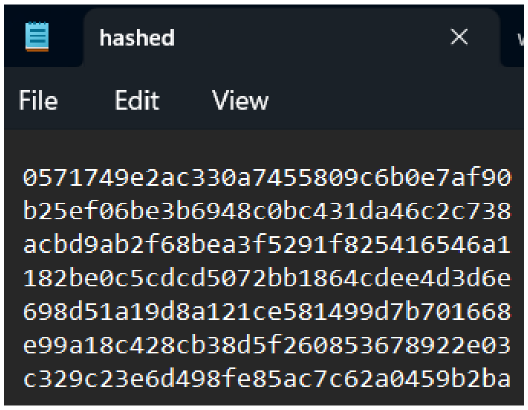
Output:



```
c329c23e6d498fe85ac7c62a0459b2ba:123_456
```

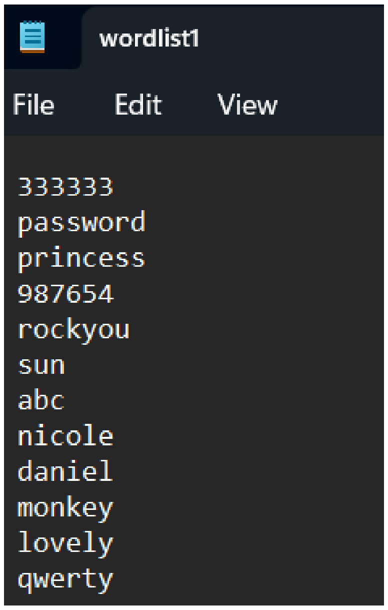
6. **Mask Attack**

Create a file with hashed passwords that you wish to crack:



```
0571749e2ac330a7455809c6b0e7af90
b25ef06be3b6948c0bc431da46c2c738
acbd9ab2f68bea3f5291f825416546a1
182be0c5cdcd5072bb1864cdee4d3d6e
698d51a19d8a121ce581499d7b701668
e99a18c428cb38d5f260853678922e03
c329c23e6d498fe85ac7c62a0459b2ba
```

Create a wordlist:

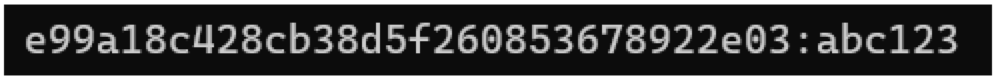


```
333333
password
princess
987654
rockyou
sun
abc
nicole
daniel
monkey
lovely
qwerty
```

Run the following command on the terminal:

```
hashcat -a6 -m0 hashed.txt wordlist1.txt ?d?d?d
```

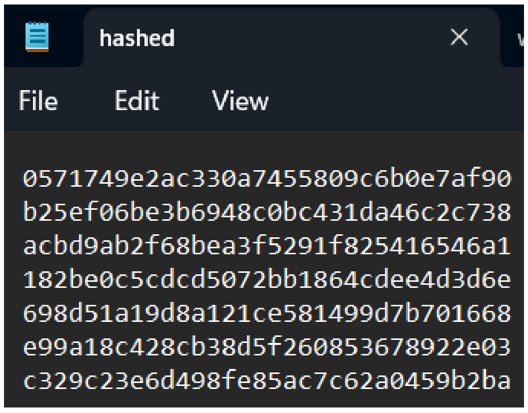
Output:



```
e99a18c428cb38d5f260853678922e03:abc123
```

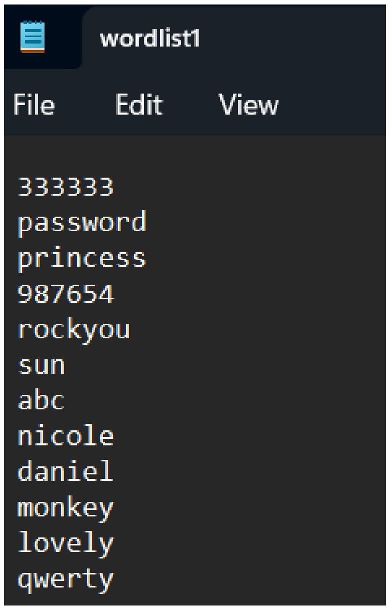
7. **Rule-based Attack**

Create a file with hashed passwords that you wish to crack:



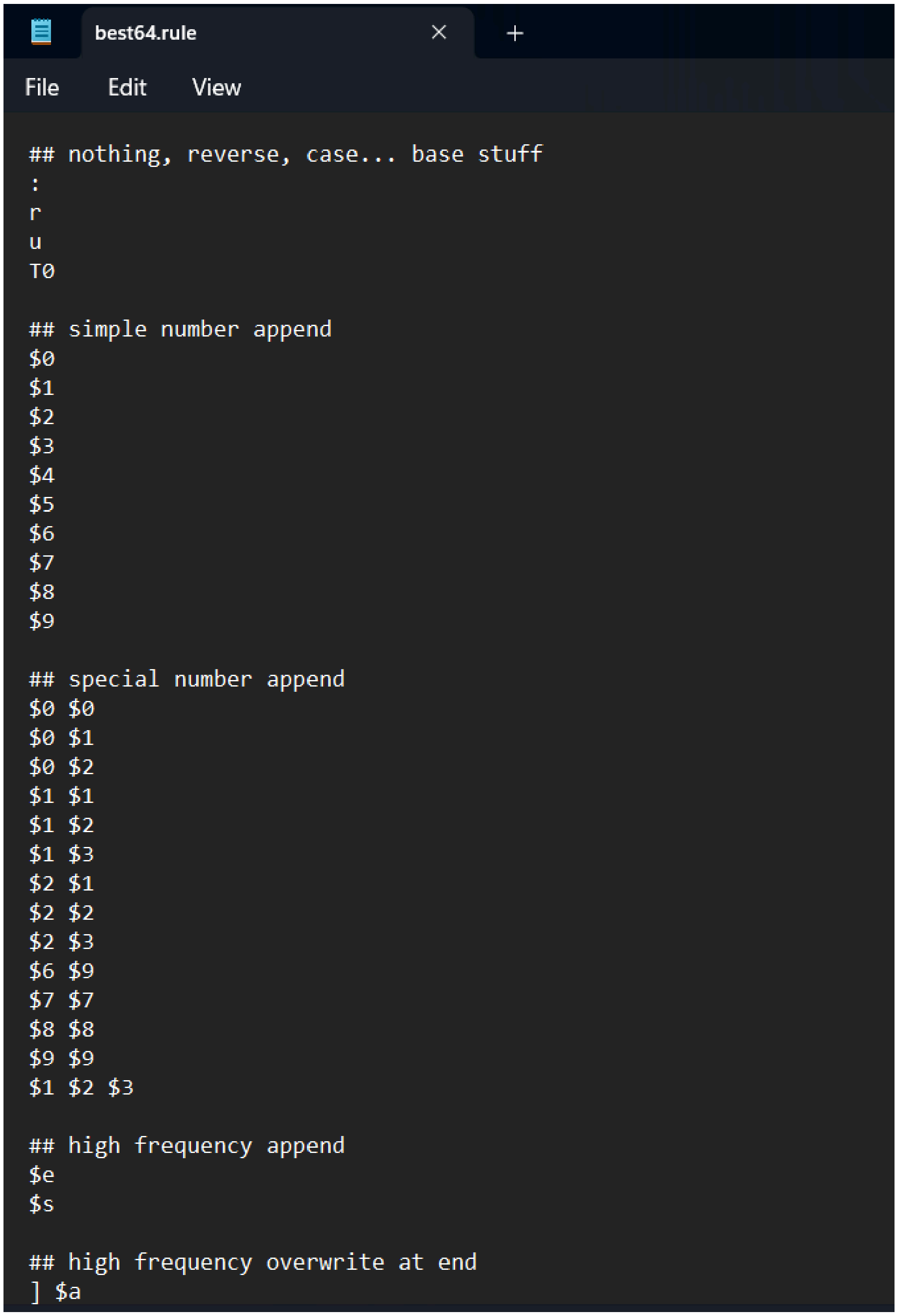
```
0571749e2ac330a7455809c6b0e7af90
b25ef06be3b6948c0bc431da46c2c738
acbd9ab2f68bea3f5291f825416546a1
182be0c5cdcd5072bb1864cdee4d3d6e
698d51a19d8a121ce581499d7b701668
e99a18c428cb38d5f260853678922e03
c329c23e6d498fe85ac7c62a0459b2ba
```

Create a wordlist:



```
333333
password
princess
987654
rockyou
sun
abc
nicole
daniel
monkey
lovely
qwerty
```

For the list rulesets, you can either use the rule engine provided inside the Hashcat folder, install another compatible rule engine or create a custom ruleset. The one used here is base64.rule provided with Hashcat:

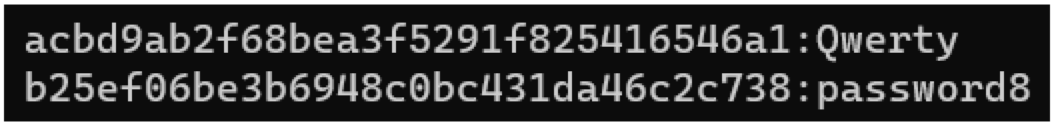


```
## nothing, reverse, case... base stuff
:
r
u
T0

## simple number append
$0
$1
$2
$3
$4
$5
$6
$7
$8
$9

## special number append
$0 $0
$0 $1
$0 $2
$1 $1
$1 $2
$1 $3
$2 $1
$2 $2
$2 $3
$6 $9
$7 $7
$8 $8
$9 $9
$1 $2 $3

## high frequency append
$e
$s

## high frequency overwrite at end
] $a
```

Run the following command on the terminal:

```
hashcat -a0 -m0 -r best64.rule hashed.txt wordlist1.txt
```

Output:



```
acbd9ab2f68bea3f5291f825416546a1:Qwerty
b25ef06be3b6948c0bc431da46c2c738:password8
```

Results

The results obtained from using Hashcat can be analyzed and interpreted in various ways to gain insights into the effectiveness of the password cracking process.

- **Cracked Passwords:** The primary result of a Hashcat operation is the cracked passwords. These can be analyzed to understand the strength and complexity of the original passwords.
- **Timing and Speed:** The time taken to crack the passwords can be analyzed to measure the efficiency of the tool and the effectiveness of the attack strategy.
- **Insecure Password Patterns:** By analyzing the cracked passwords, common patterns and trends can be identified. This can reveal insecure password practices, such as the use of common words or simple numeric sequences.
- **Heatmaps:** Heatmaps can be generated to visualize common elements in the cracked passwords, such as city names.

Conclusion

In this report, we discussed Hashcat, a powerful open-source password cracking tool. It supports multiple hashing algorithms and is compatible with various operating systems, making it a versatile tool in the field of cybersecurity. One of the key features of Hashcat is its ability to leverage the power of GPUs for accelerated password cracking. It also supports several attack modes, including dictionary, combination, brute force, mask, hybrid, and rule-based attacks, allowing it to adapt to different scenarios and crack passwords more effectively. Implementing Hashcat requires a solid understanding of hashing, suitable hardware, knowledge of Hashcat options, ethical considerations, and access to password lists. The results obtained from using Hashcat, typically presented in the form of cracked passwords, can be analyzed to gain insights into the effectiveness of the password cracking process. This analysis can help in understanding the vulnerabilities in password policies and guide the development of more secure practices. In summary, Hashcat is a robust and flexible tool for password recovery and plays a significant role in enhancing cybersecurity practices.

Hashcat has proven to be a powerful tool in the field of cybersecurity, providing valuable insights into password vulnerabilities and enhancing security practices. As we move forward, the continuous development and improvement of Hashcat will be crucial in adapting to the ever-evolving landscape of cybersecurity threats. The future of Hashcat in cybersecurity looks promising, with potential advancements in speed, efficiency, and the range of supported hashing algorithms. It will continue to be an indispensable tool for security professionals, aiding in the ongoing effort to safeguard digital information.