Technical Report: Cracking Hashes with Hashcat

1. Introduction

The technique of creating cryptographic hashes for passwords and then trying to crack them with the Hashcat program is described in this report. The goal is to illustrate the process of password cracking, emphasizing Hashcat's capabilities as well as some hashing methods' vulnerabilities to brute-force attacks.

2. Tools Used

Hashcat is an open-source, powerful password cracking tool. It uses CPU or GPU processing for faster performance and supports a variety of hashing techniques and cracking modes.

3. Step-by-Step Execution

The following steps outline the process of generating an SHA256 hash and cracking it with Hashcat:

3.1 Generating the MD5 Hash

Step 1: Open a terminal or command-line interface.

Step 2: Execute the following command to generate the MD5 hash of the string "abc123":

Bash

```
echo -n "abc123" | md5sum | cut -d' ' -f1 > hashes.txt
```

- echo -n "abc123": Outputs the string "abc123" without a newline.
- | md5sum: Pipes the output to the md5sum command to calculate the hash.
- | cut -d' ' -f1: Extracts the hash value from the md5sum output.
- > hashes.txt: Saves the hash to a file named "hashes.txt".

Step 3: Display the contents of "hashes.txt" to verify the hash:

Bash

```
cat hashes.txt
```

3.1.2 Cracking the Hash with Hashcat

Step 1: Execute the following Hashcat command to initiate a brute-force attack:

Bash

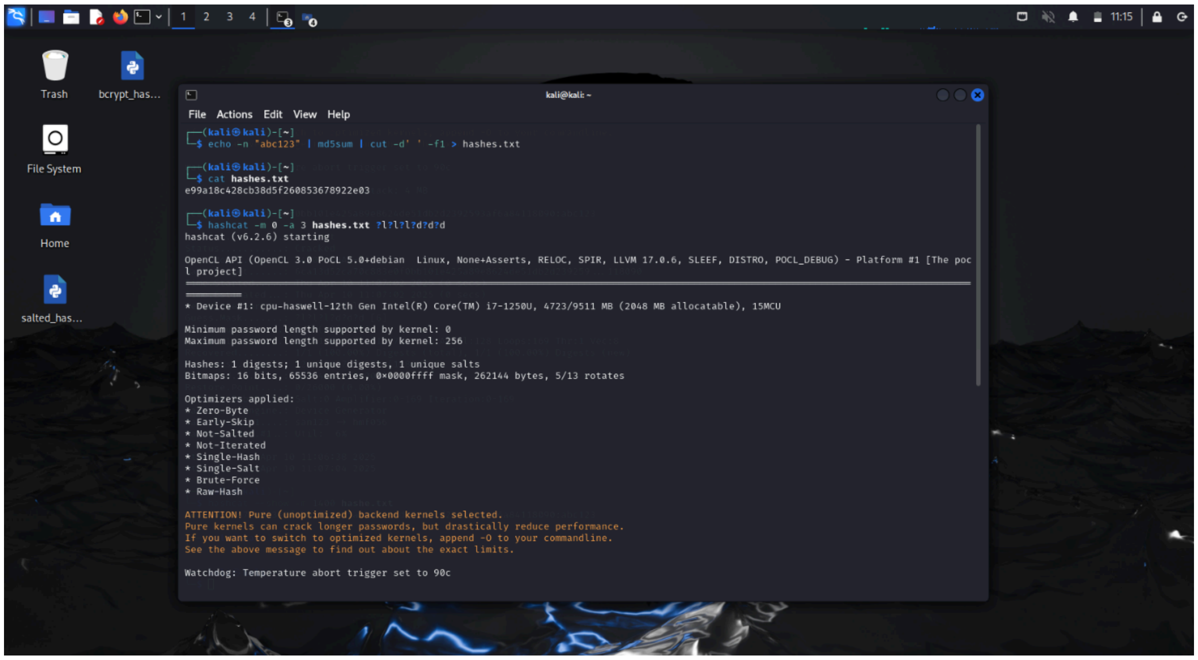
hashcat -m 0 -a 3 hash.txt ?a?a?a?a?a?a

- -m 0: Specifies MD5 as the hash type.
- -a 3: Selects the brute-force attack mode.
- hashes.txt: Indicates the file containing the hash.
- ?a?a?a?a?a?a: Defines a mask to try all 6-character combinations.

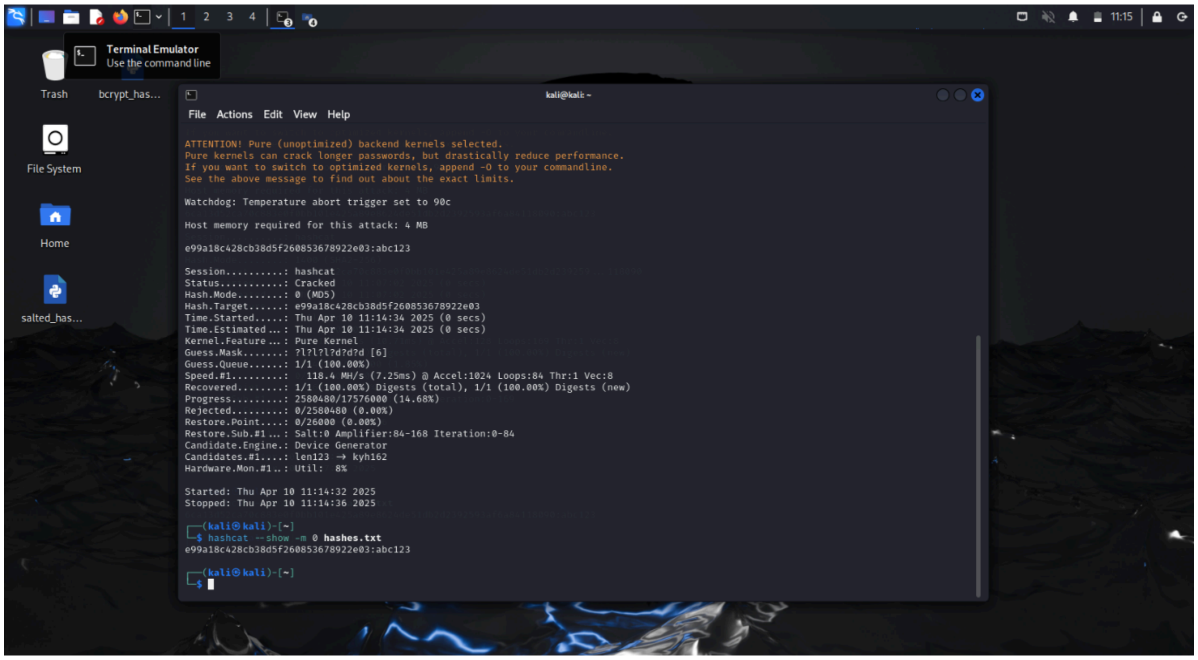
Step 2: Once Hashcat completes, use the following command to display the cracked password:

Bash

hashcat --show hashes.txt



```
┌──(kali㉿kali)-[~]
└─$ echo -n "abc123" | md5sum | cut -d' ' -f1 > hashes.txt

┌──(kali㉿kali)-[~]
└─$ cat hashes.txt
e99a18c428cb38d5f260853678922e03

┌──(kali㉿kali)-[~]
└─$ hashcat -m 0 -a 3 hashes.txt ?l?l?l?d?d?d
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 5.0+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The poc
l project]
==========================================================================================================================================
==========
* Device #1: cpu-haswell-12th Gen Intel(R) Core(TM) i7-1250U, 4723/9511 MB (2048 MB allocatable), 15MCU

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

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Brute-Force
* 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
```



```
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: 4 MB

e99a18c428cb38d5f260853678922e03:abc123

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: e99a18c428cb38d5f260853678922e03
Time.Started.....: Thu Apr 10 11:14:34 2025 (0 secs)
Time.Estimated...: Thu Apr 10 11:14:34 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?l?l?l?d?d?d [6]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   118.4 MH/s (7.25ms) @ Accel:1024 Loops:84 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 2580480/17576000 (14.68%)
Rejected.........: 0/2580480 (0.00%)
Restore.Point....: 0/26000 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:84-168 Iteration:0-84
Candidate.Engine.: Device Generator
Candidates.#1....: len123 → kyh162
Hardware.Mon.#1..: Util:  8%

Started: Thu Apr 10 11:14:32 2025
Stopped: Thu Apr 10 11:14:36 2025

┌──(kali㉿kali)-[~]
└─$ hashcat --show -m 0 hashes.txt
e99a18c428cb38d5f260853678922e03:abc123

┌──(kali㉿kali)-[~]
└─$ 
```

3.2 Generating the SHA1 Hash

Step 1: Open a terminal.

Step 2: Generate the SHA1 hash:

Bash

echo -n "123" | openssl sha1 | cut -d' ' -f2 > hash.txt

- echo -n "123": Outputs "123" without a newline.
- | openssl sha1: Calculates the SHA1 hash.
- | cut -d' ' -f2: Extracts the hash.
- > hash.txt: Saves the hash to "hash.txt".

Step 3: Verify the hash:

Bash

cat hash.txt

3.2.1 Cracking the Hash with Hashcat

Step 1: Crack the hash:

Bash

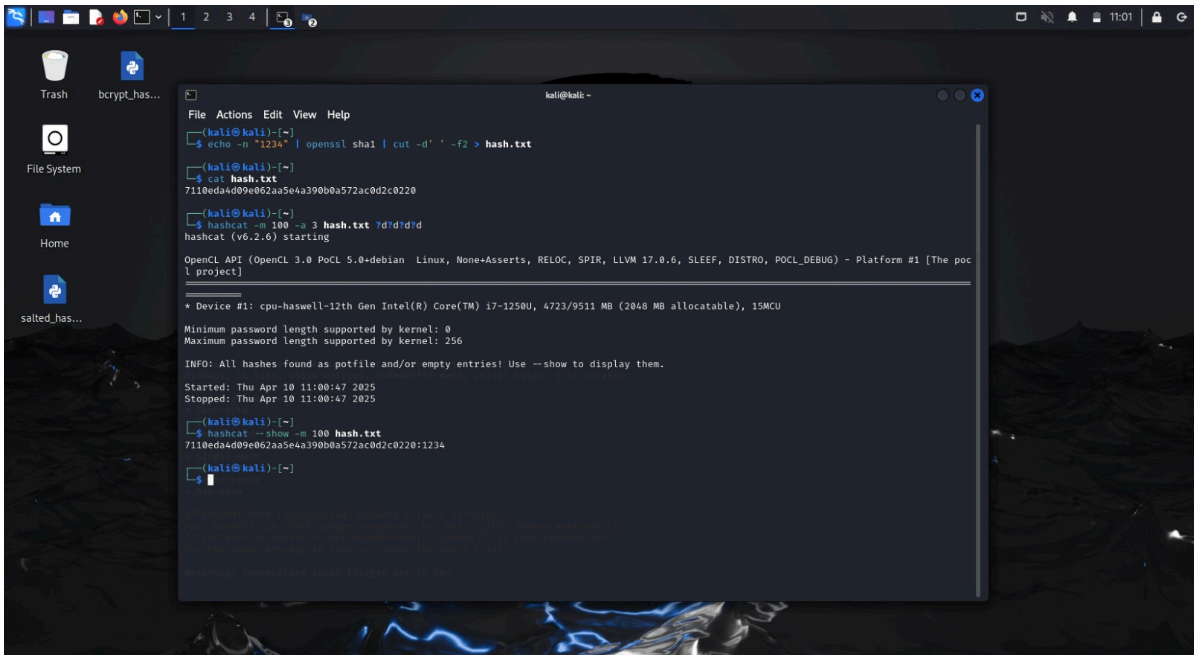
hashcat -m 100 -a 3 hash.txt ?a?a?a

- -m 100: Specifies SHA1.
- -a 3: Brute-force attack.
- hash.txt: Hash file.
- ?a?a?a: 3-character mask.

Step 2: Show the cracked password:

Bash

hashcat --show hash.txt



```
┌──(kali㉿kali)-[~]
└─$ echo -n "1234" | openssl sha1 | cut -d' ' -f2 > hash.txt

┌──(kali㉿kali)-[~]
└─$ cat hash.txt
7110eda4d09e062aa5e4a390b0a572ac0d2c0220

┌──(kali㉿kali)-[~]
└─$ hashcat -m 100 -a 3 hash.txt ?d?d?d?d
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 5.0+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The poc
l project]
==================================================================================================================================
=========
* Device #1: cpu-haswell-12th Gen Intel(R) Core(TM) i7-1250U, 4723/9511 MB (2048 MB allocatable), 15MCU

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

INFO: All hashes found as potfile and/or empty entries! Use --show to display them.

Started: Thu Apr 10 11:00:47 2025
Stopped: Thu Apr 10 11:00:47 2025

┌──(kali㉿kali)-[~]
└─$ hashcat --show -m 100 hash.txt
7110eda4d09e062aa5e4a390b0a572ac0d2c0220:1234

┌──(kali㉿kali)-[~]
└─$ 
```

3.3 Generating the SHA256 Hash

Step 1: Open a terminal or command-line interface.

Step 2: Execute the following command to generate the SHA256 hash of the string "abc123":

Bash

echo -n "abc123" | openssl sha256 | cut -d' ' -f2 > hashe.txt

- echo -n "abc123": Outputs the string "abc123" without a newline.

• | openssl sha256: Pipes the output to the OpenSSL sha256 command to calculate the hash.

• | cut -d' ' -f2: Extracts the hash value from the OpenSSL output.

• > hashe.txt: Saves the hash to a file named "hashe.txt".

Step 3: Display the contents of "hashe.txt" to verify the hash:

Bash

```
cat hashe.txt
```

3.3.1 Cracking the Hash with Hashcat

Step 1: Execute the following Hashcat command to initiate a brute-force attack:

Bash

```
hashcat -m 1400 -a 3 hash.txt ?a?a?a?a?a?a
```

• -m 1400: Specifies SHA256 as the hash type.

• -a 3: Selects the brute-force attack mode.

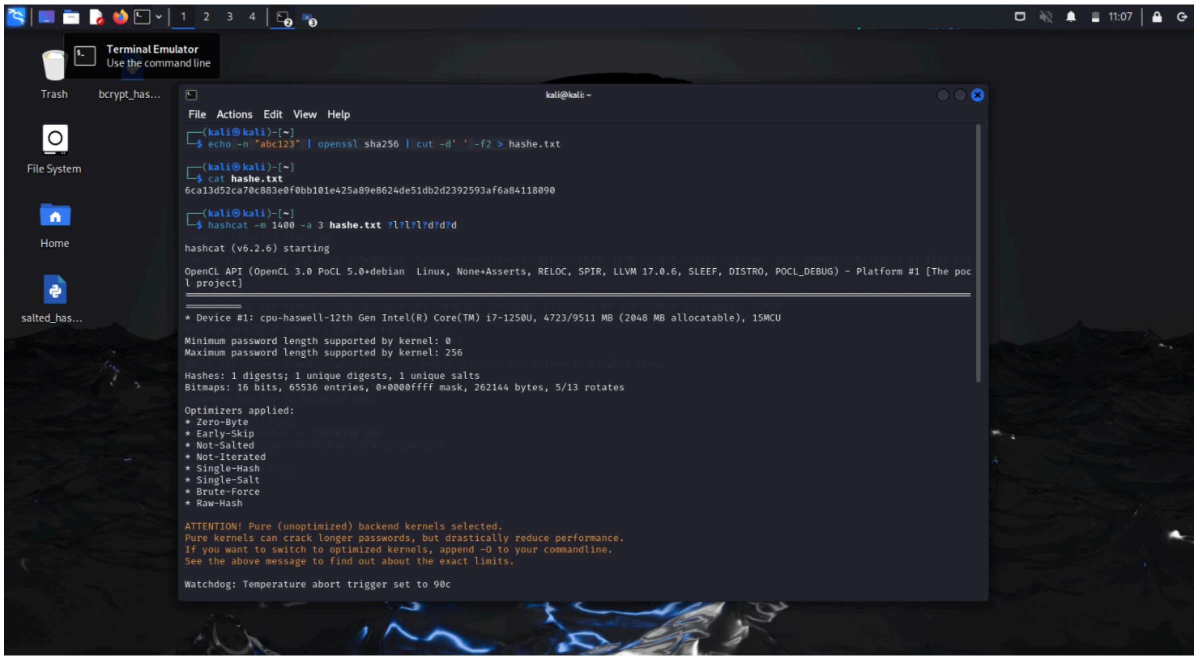
• hash.txt: Indicates the file containing the hash.

• ?a?a?a?a?a?a: Defines a mask to try all 6-character combinations (alphanumeric and symbols).

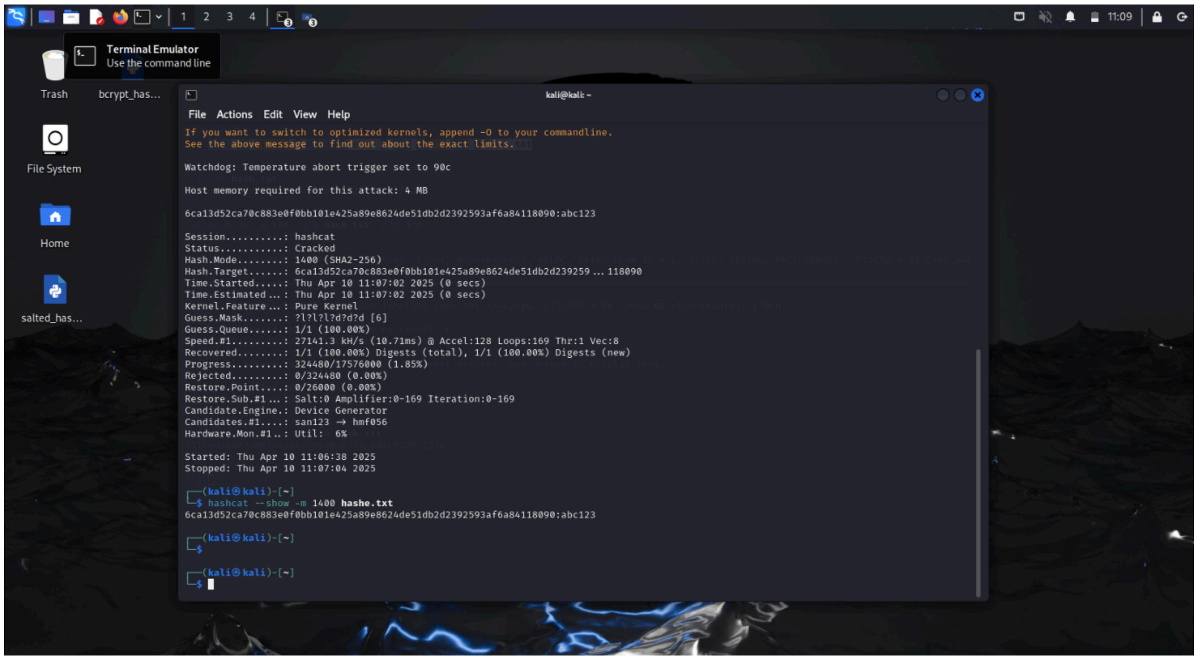
Step 2: Once Hashcat completes, use the following command to display the cracked password:

Bash

```
hashcat --show hashe.txt
```



```
┌──(kali㉿kali)-[~]
└─$ echo -n "abc123" | openssl sha256 | cut -d' ' -f2 > hashe.txt

┌──(kali㉿kali)-[~]
└─$ cat hashe.txt
6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090

┌──(kali㉿kali)-[~]
└─$ hashcat -m 1400 -a 3 hashe.txt ?l?l?l?d?d?d

hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 5.0+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The poc
l project]
==========================================================================================================================================
==========
* Device #1: cpu-haswell-12th Gen Intel(R) Core(TM) i7-1250U, 4723/9511 MB (2048 MB allocatable), 15MCU

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

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Brute-Force
* 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
```



```
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: 4 MB

6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090:abc123

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 1400 (SHA2-256)
Hash.Target......: 6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d239259...118090
Time.Started.....: Thu Apr 10 11:07:02 2025 (0 secs)
Time.Estimated...: Thu Apr 10 11:07:02 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?l?l?l?d?d?d [6]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 27141.3 kH/s (10.71ms) @ Accel:128 Loops:169 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 324480/17576000 (1.85%)
Rejected.........: 0/324480 (0.00%)
Restore.Point....: 0/26000 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-169 Iteration:0-169
Candidate.Engine.: Device Generator
Candidates.#1....: san123 -> hmf056
Hardware.Mon.#1..: Util:  6%

Started: Thu Apr 10 11:06:38 2025
Stopped: Thu Apr 10 11:07:04 2025

┌──(kali㉿kali)-[~]
└─$ hashcat --show -m 1400 hashe.txt
6ca13d52ca70c883e0f0bb101e425a89e8624de51db2d2392593af6a84118090:abc123

┌──(kali㉿kali)-[~]
└─$

┌──(kali㉿kali)-[~]
└─$ 
```

4. Findings & Analysis

The SHA256 hash of the password "abc123" was successfully broken by the Hashcat attack.

• Even with a powerful hashing algorithm like SHA256, the cracking procedure shows that it is

possible to recover comparatively short and straightforward passwords.
• The "?a?a?a?a?a?a" mask emphasizes how crucial password complexity is. A password that is longer or more complicated would take a lot longer to crack.
•The processing capacity of the hardware (CPU or GPU) used affects how quickly the cracking process proceeds.

5. Recommendations

- Password Complexity: Implement strong password rules that require a minimum length and a mix of capital, lowercase, and numeric characters, as well as symbols.
- Employ Unique Salts for Hashing: Prior to hashing, always add a distinct, random salt to every password. Rainbow table assaults are made ineffective by this technique, which guarantees that identical passwords produce distinct hash values.
- Select More safe Algorithms: Think about switching from SHA256 to more safe algorithms like Argon2, Scrypt, or Bcrypt. By adding computational delays, these are especially made to withstand brute-force and GPU-based cracking.
- Use Login Attempt Controls: Limit the number of times an account can be logged in by temporarily locking it or adding delays. As a result, automated assaults that attempt several password combinations are less feasible.
- Multi-Factor Authentication (MFA): Employ MFA to add an extra layer of security beyond passwords.

6. Conclusion

This experiment demonstrated how to use Hashcat to crack a password hash. The results highlight how crucial it is to use secure password practices, appropriate hashing methods, and other security measures to guard against password-based attacks. Hashcat is a useful tool for auditing and testing security, but it also draws attention to the dangers of using weak passwords.