

Creating a Strong Password and Evaluate Its Strength

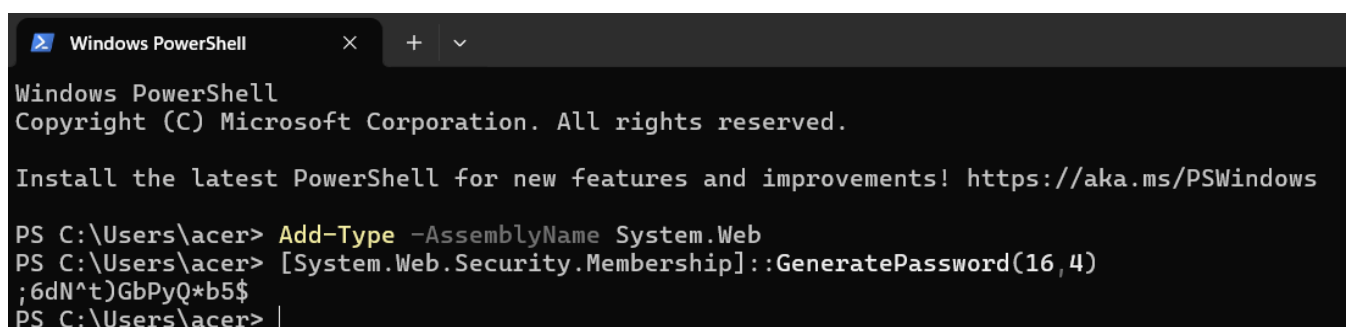
Step 1 Create an Initial Strong Password Candidate

Firstly, generate one password that follows strong password principles so we can test it later.

1. Understand what “strong” means (we’ll prove it later with tools):
 - At least 12–16 characters long.
 - Includes upper-case, lower-case, numbers, and special characters.
 - Not based on dictionary words or personal info.
 - Random/unpredictable.
2. On Windows or Kali (either works), open a terminal or password generator:
 - On Windows: Use PowerShell:

```
Add-Type -AssemblyName System.Web  
[System.Web.Security.Membership]::GeneratePassword(16,4)
```

(This creates a 16-character password with 4 special characters.)

A screenshot of a Windows PowerShell terminal window. The title bar says "Windows PowerShell" with a close button and a dropdown menu. The terminal text includes the copyright notice for Microsoft Corporation, a link to update PowerShell, and the execution of the command `Add-Type -AssemblyName System.Web` followed by `[System.Web.Security.Membership]::GeneratePassword(16,4)`. The output of the command is a 16-character password: `;6dN^t)GbPyQ*b5$`.

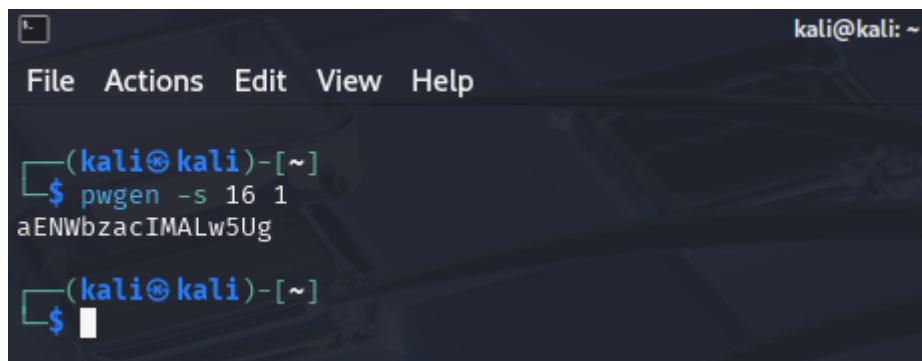
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\acer> Add-Type -AssemblyName System.Web
PS C:\Users\acer> [System.Web.Security.Membership]::GeneratePassword(16,4)
;6dN^t)GbPyQ*b5$
PS C:\Users\acer> |
```

The Created Password- ;6dN^t)GbPyQ*b5\$

- On Kali: Use pwgen (if installed):
`pwgen -s 16 1`
(-s for secure, 16 length, 1 = number of passwords)

A terminal window with a dark background and a menu bar at the top containing 'File', 'Actions', 'Edit', 'View', and 'Help'. The terminal title is 'kali@kali: ~'. The prompt is '(kali@kali)-[~]'. The user enters the command '\$ pwgen -s 16 1'. The output is 'aENWbzacIMALw5Ug'. The prompt is then shown again with a cursor ready for input.

```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ pwgen -s 16 1  
aENWbzacIMALw5Ug  
(kali@kali)-[~]  
$
```

The Created Password- aENWbzacIMALw5Ug

Step 2 Test the Passwords Using the Provided Tool

1. Go to <https://passwordmeter.com/>

The Password Meter

Test Your Password		Minimum Requirements	
Password:	<input type="text"/>	<ul style="list-style-type: none">• Minimum 8 characters in length• Contains 3/4 of the following items:<ul style="list-style-type: none">- Uppercase Letters- Lowercase Letters- Numbers- Symbols	
Hide:	<input checked="" type="checkbox"/>		
Score:	0%		
Complexity:	Too Short		

Additions	Type	Rate	Count	Bonus
<input checked="" type="checkbox"/> Number of Characters	Flat	$+(n*4)$	0	0
<input checked="" type="checkbox"/> Uppercase Letters	Cond/Incr	$+(len-n)*2$	0	0
<input checked="" type="checkbox"/> Lowercase Letters	Cond/Incr	$+(len-n)*2$	0	0
<input checked="" type="checkbox"/> Numbers	Cond	$+(n*4)$	0	0
<input checked="" type="checkbox"/> Symbols	Flat	$+(n*6)$	0	0
<input checked="" type="checkbox"/> Middle Numbers or Symbols	Flat	$+(n*2)$	0	0
<input checked="" type="checkbox"/> Requirements	Flat	$+(n*2)$	0	0

Deductions	Type	Rate	Count	Bonus
<input checked="" type="checkbox"/> Letters Only	Flat	$-n$	0	0
<input checked="" type="checkbox"/> Number			0	0
<input checked="" type="checkbox"/> Repeat			0	0

2. For each password:

- Enter it into the “Password” box.
- Note the score percentage and Complexity rating (e.g., “Very Strong”).
- For Windows Password:

The Password Meter

Test Your Password		Minimum Requirements	
Password:	<input type="password"/>	<ul style="list-style-type: none">• Minimum 8 characters in length• Contains 3/4 of the following items:<ul style="list-style-type: none">- Uppercase Letters- Lowercase Letters- Numbers- Symbols	
Hide:	<input checked="" type="checkbox"/>		
Score:	100%		
Complexity:	Very Strong		

Additions	Type	Rate	Count	Bonus
<input checked="" type="checkbox"/> Number of Characters	Flat	$+(n*4)$	17	+ 68
<input checked="" type="checkbox"/> Uppercase Letters	Cond/Incr	$+(len-n)*2$	4	+ 26
<input checked="" type="checkbox"/> Lowercase Letters	Cond/Incr	$+(len-n)*2$	5	+ 24
<input checked="" type="checkbox"/> Numbers	Cond	$+(n*4)$	2	+ 8
<input checked="" type="checkbox"/> Symbols	Flat	$+(n*6)$	5	+ 30
<input checked="" type="checkbox"/> Middle Numbers or Symbols	Flat	$+(n*2)$	5	+ 10
<input checked="" type="checkbox"/> Requirements	Flat	$+(n*2)$	5	+ 10

Deductions	Type	Rate	Count	Bonus

- For Kali-linux Password:

The Password Meter

Test Your Password		Minimum Requirements			
Password:	<input type="password" value="*****"/>	<ul style="list-style-type: none"> • Minimum 8 characters in length • Contains 3/4 of the following items: <ul style="list-style-type: none"> - Uppercase Letters - Lowercase Letters - Numbers - Symbols 			
Hide:	<input checked="" type="checkbox"/>				
Score:	<div><div>95%</div></div>				
Complexity:	Very Strong				

Additions		Type	Rate	Count	Bonus
⚙	Number of Characters	Flat	$+(n*4)$	<input type="text" value="16"/>	+ 64
⚙	Uppercase Letters	Cond/Incr	$+(len-n)*2$	<input type="text" value="8"/>	+ 16
⚙	Lowercase Letters	Cond/Incr	$+(len-n)*2$	<input type="text" value="7"/>	+ 18
✓	Numbers	Cond	$+(n*4)$	<input type="text" value="1"/>	+ 4
✗	Symbols	Flat	$+(n*6)$	<input type="text" value="0"/>	0
✓	Middle Numbers or Symbols	Flat	$+(n*2)$	<input type="text" value="1"/>	+ 2
✓	Requirements	Flat	$+(n*2)$	<input type="text" value="4"/>	+ 8

Deductions

3. Save them as:

- windows_password_test.pdf
- Kali_password_test.pdf

4. Record the results in a quick table

Password Source	Length	Score (%)	Complexity	Notes
Windows	16	100%	Very Strong	Includes upper/lower/numbers/symbols, high character variety, minimal deductions
Kali	16	95%	Very Strong	No symbols → lost points, deductions for consecutive uppercase/lowercase letters

Step 3 Improve the Weaker Password

To make the Kali password stronger by fixing weaknesses, then retest.

Why:

Even though the Kali password scored 95% and was rated “Very Strong,” in real-world terms it’s more vulnerable to certain attacks:

- **Dictionary attack:** While it’s random, the lack of symbols narrows the search space.

Why it matters here:

- Even random-looking strings without symbols can sometimes appear in cracked password databases or generated pattern lists.
 - Symbols disrupt predictable patterns and help avoid dictionary hits.
 - Placing symbols in the **middle** of the password (not just at the ends) makes it harder for attackers using “mangling rules” in tools like Hashcat or John the Ripper.
- **Brute force attack:** Fewer character types = fewer permutations for an attacker to guess.

Why it matters here:

- Your original Kali password had 3 character sets (upper, lower, number) but **no symbols**, so the search space is smaller.
- Adding symbols increases the number of possible characters from ~62 to ~95, drastically increasing permutations.
- This makes brute forcing far more time-consuming — often jumping from hours to centuries in cracking estimates.

For improvement:

1. **Length** — Keep at least 16 characters (recommended: 12+ minimum, 16+ for strong protection).
2. **Character Variety** — Use uppercase, lowercase, numbers, and symbols.
3. **Avoid Patterns** — No “abc”, “123”, keyboard patterns like “qwerty”.
4. **Random Placement of Symbols** — Not clustered; scattered throughout.
5. **No Personal Info** — No names, birthdays, or known words.

The Password Meter

Test Your Password		Minimum Requirements
Password:	<input type="password" value="....."/>	<ul style="list-style-type: none"> Minimum 8 characters in length Contains 3/4 of the following items: <ul style="list-style-type: none"> - Uppercase Letters - Lowercase Letters - Numbers - Symbols
Hide:	<input checked="" type="checkbox"/>	
Score:	<div><div>100%</div></div>	
Complexity:	Very Strong	

Additions		Type	Rate	Count	Bonus
	Number of Characters	Flat	$+(n*4)$	<input type="text" value="19"/>	+ 76
	Uppercase Letters	Cond/Incr	$+(len-n)*2$	<input type="text" value="8"/>	+ 22
	Lowercase Letters	Cond/Incr	$+(len-n)*2$	<input type="text" value="7"/>	+ 24
	Numbers	Cond	$+(n*4)$	<input type="text" value="2"/>	+ 8
	Symbols	Flat	$+(n*6)$	<input type="text" value="2"/>	+ 12
	Middle Numbers or Symbols	Flat	$+(n*2)$	<input type="text" value="4"/>	+ 8
	Requirements	Flat	$+(n*2)$	<input type="text" value="5"/>	+ 10
Deductions					
	Letters Only	Flat	$-n$	<input type="text" value="0"/>	0

Step 4 Password Attack Simulation (Kali Linux)

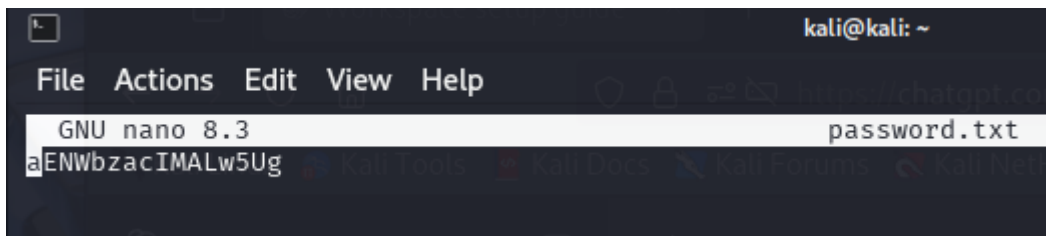
We'll use **John the Ripper** (comes preinstalled on Kali) with a simple wordlist to demonstrate:

- How a **dictionary attack** works.
- Why the weaker password is easier to crack.
- Why the improved password is harder.

4.1 Prepare a hash of your password

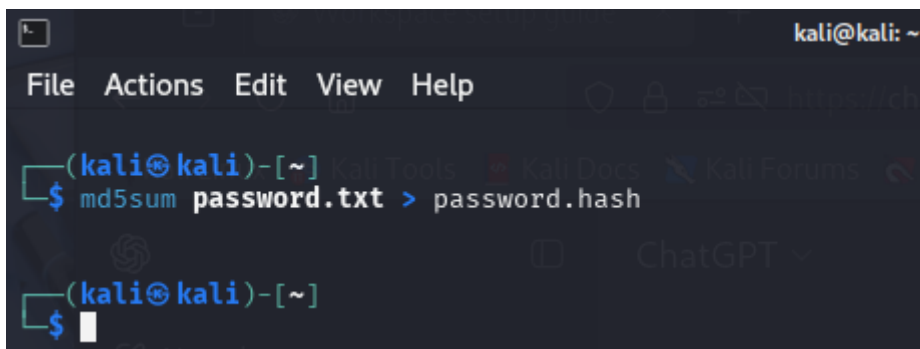
We'll never store the plain password in the test, we'll hash it.

1. On Kali, open a terminal and create a file password.txt containing the password you want to test (the original Kali password).
nano password.txt



```
kali@kali: ~  
File Actions Edit View Help  
GNU nano 8.3 password.txt  
aENWbzacIMALw5Ug
```

2. Generate an **MD5 hash** of the password:
md5sum password.txt > password.hash



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ md5sum password.txt > password.hash  
(kali@kali)-[~]  
$
```

This creates a file with the MD5 hash of your password.

4.2 Use John the Ripper with a dictionary attack

1. Use the built-in rockyou.txt wordlist (already in Kali):

```
john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 password.hash
```

```

(kali@kali)-[~]
$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 password.hash

Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 512/512 AVX512BW 16x3]) john --wordlist=/usr/share/wordlist
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:00 DONE (2025-08-12 09:48) 0g/s 25163Kp/s 25163Kc/s 25163KC/s fuckyooh21..*7;Vamos!
Session completed.

(kali@kali)-[~] practice
$

```

2. If the password is simple or close to a common word, John will crack it quickly.

- If it can't crack it, that's your proof of strength.

4.3 Attack simulation results:

1. Original Kali Password

- If you run the test on the original password and it still shows "0g" > note it's already strong against RockYou dictionary.
- If it cracks > highlight the weakness.

2. Improved Kali Password

- "0g" result proves stronger defense due to added complexity and symbol variety.

3. Explanation

- Mention that John's RockYou dictionary contains millions of real-world leaked passwords.
- If a password resists this, it's resistant to common **dictionary attacks** but could still be brute-forced given enough time and resources.

Step 5 Hashcat Brute Force Test

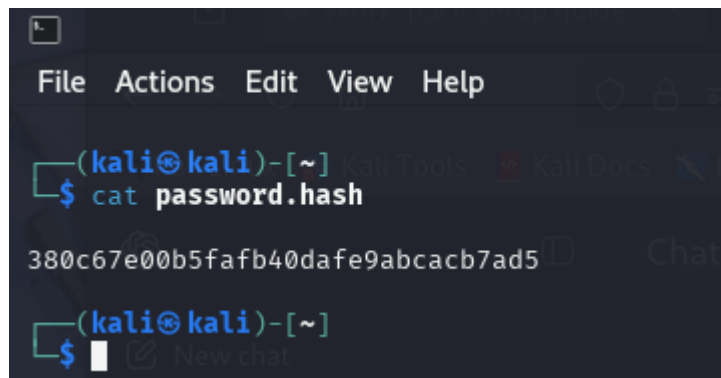
5.1 Prepare the hash file

We already have your MD5 hash in password.hash.

Hashcat also needs the raw hash in a text file (same as John).

Make sure it contains only the hash, no extra text:

cat password.hash

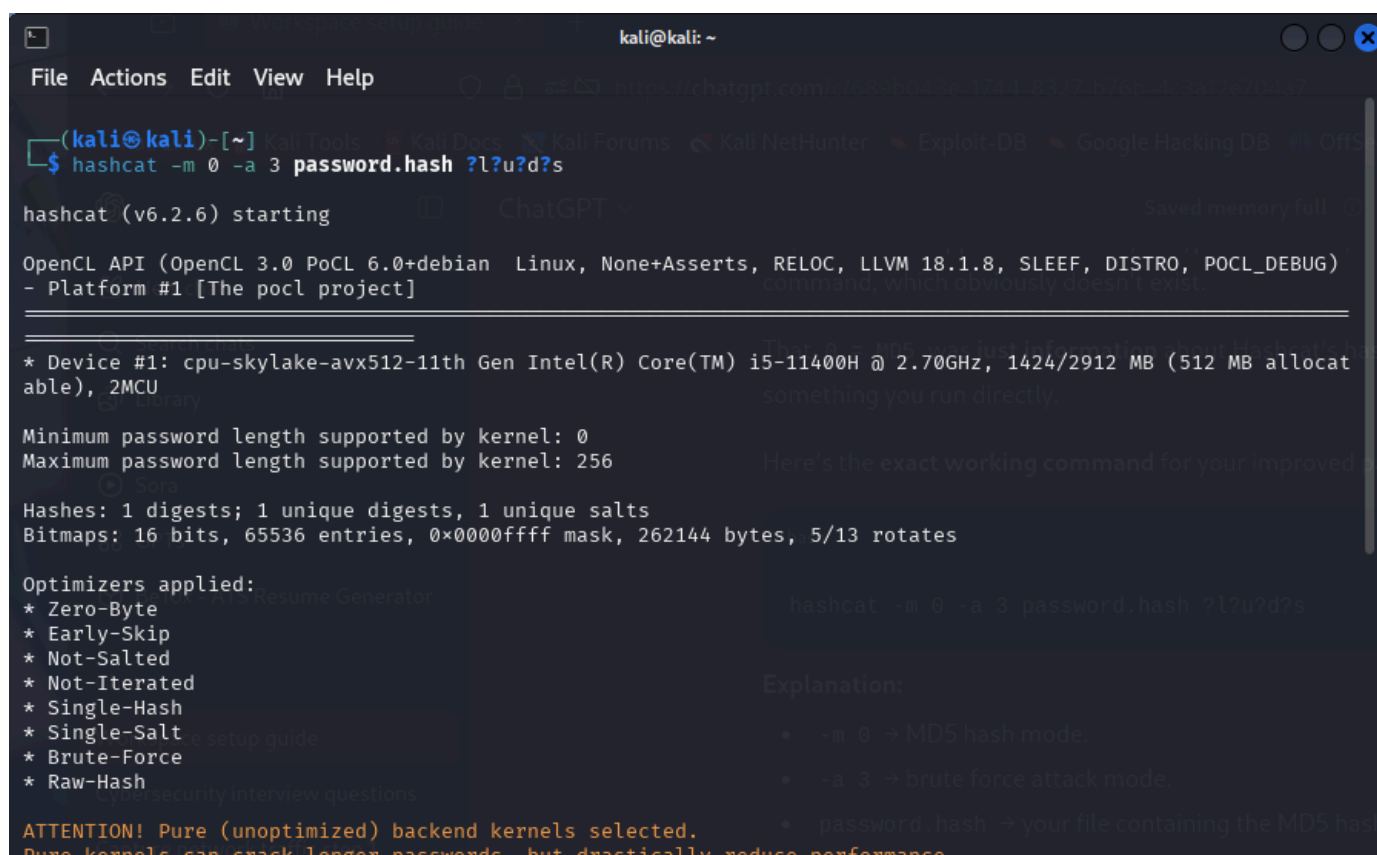


```
(kali㉿kali)-[~]  
$ cat password.hash  
380c67e00b5fafb40dafa9abcacb7ad5  
  
(kali㉿kali)-[~]  
$
```

5.2 Run brute force

Warning: This will try *all possible combinations* for a given length and charset — it can be very fast for short, weak passwords, but impractical for long, strong ones.

hashcat -m 0 -a 3 password.hash ?l?u?d?s



```
kali@kali: ~  
File Actions Edit View Help  
  
(kali㉿kali)-[~]  
$ hashcat -m 0 -a 3 password.hash ?l?u?d?s  
  
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-skylake-avx512-11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz, 1424/2912 MB (512 MB allocated), 2MCU  
  
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.  
  
hashcat -m 0 -a 3 password.hash ?l?u?d?s  
  
Explanation:  
* -m 0 -> MD5 hash mode.  
* -a 3 -> brute force attack mode.  
* password.hash -> your file containing the MD5 hash
```

```
kali@kali: ~  
File Actions Edit View Help  
Pure kernels can crack longer passwords, but drastically reduce performance.  
If you want to switch to optimized kernels, append -O to your commandline. pivot-DB Google Hacking DB OffS  
See the above message to find out about the exact limits.  
Watchdog: Temperature abort trigger set to 90c  
Host memory required for this attack: 0 MB  
Approaching final keyspace - workload adjusted.  
Session.....: hashcat  
Status.....: Exhausted  
Hash.Mode.....: 0 (MD5)  
Hash.Target.....: 380c67e00b5fafb40dfe9abcacb7ad5  
Time.Started.....: Tue Aug 12 09:58:26 2025 (0 secs)  
Time.Estimated...: Tue Aug 12 09:58:26 2025 (0 secs)  
Kernel.Feature...: Pure Kernel  
Guess.Mask.....: ?l?u?d?s [4]  
Guess.Queue.....: 1/1 (100.00%)  
Speed.#1.....: 1164.3 kH/s (0.18ms) @ Accel:256 Loops:26 Thr:1 Vec:16  
Recovered.....: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new) 0 -a 3 password.hash ?l?u?d?s  
Progress.....: 223080/223080 (100.00%)  
Rejected.....: 0/223080 (0.00%)  
Restore.Point...: 8580/8580 (100.00%)  
Restore.Sub.#1...: Salt:0 Amplifier:0-26 Iteration:0-26  
Candidate.Engine.: Device Generator  
Candidates.#1....: sE3} → xQ8~  
Hardware.Mon.#1..: Util: 54%  
Started: Tue Aug 12 09:57:49 2025  
Stopped: Tue Aug 12 09:58:28 2025  
(kali@kali)-[~] practice  
$
```

Explanation:

- -m 0 → MD5 hash mode.
- -a 3 → brute force attack mode.
- password.hash → your file containing the MD5 hash.
- ?l?u?d?s → tells Hashcat to try lowercase, uppercase, digits, and symbols.

5.3 Weak Password Demo (Fast Crack)

This one is short & simple so Hashcat cracks it quickly, perfect for showing brute force success.

Create weak password hash

```
echo -n "Pass12" | md5sum | awk '{print $1}' > weak.hash
```

Brute force: 1 uppercase, 3 lowercase, 2 digits

```
hashcat -m 0 -a 3 weak.hash ?u?!?!?!?d?d
```

```

(kali@kali)-[~]
$ # Create weak password hash
echo -n "Pass12" | md5sum | awk '{print $1}' > weak.hash

# Brute force: 1 uppercase, 3 lowercase, 2 digits
hashcat -m 0 -a 3 weak.hash ?u?l?l?l?d?d

hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEP, DISTRO, POCL_DEBUG)
- Platform #1 [The pocl project]

=====

* Device #1: cpu-skylake-avx512-11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz, 1424/2912 MB (512 MB allocatable), 2MCU

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

Host memory required for this attack: 0 MB

9fb6c877704a27519fa8e32a0c32a4b1:Pass12

Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 0 (MD5)
Hash.Target.....: 9fb6c877704a27519fa8e32a0c32a4b1
Time.Started.....: Tue Aug 12 10:03:06 2025 (0 secs)

```

```

Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 0 (MD5)
Hash.Target.....: 9fb6c877704a27519fa8e32a0c32a4b1
Time.Started.....: Tue Aug 12 10:03:06 2025 (0 secs)
Time.Estimated...: Tue Aug 12 10:03:06 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.....: ?u?l?l?l?d?d [6]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 119.7 MH/s (0.55ms) @ Accel:256 Loops:169 Thr:1 Vec:16
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 4585984/45697600 (10.04%)
Rejected.....: 0/4585984 (0.00%)
Restore.Point....: 6656/67600 (9.85%)
Restore.Sub.#1...: Salt:0 Amplifier:0-169 Iteration:0-169
Candidate.Engine.: Device Generator
Candidates.#1....: Manp56 -> Glpy12
Hardware.Mon.#1..: Util: 55%

Started: Tue Aug 12 10:03:04 2025
Stopped: Tue Aug 12 10:03:08 2025

```

5.4 Strong Password Demo (Slow / Impractical Crack)

This is your improved 16-character password, Hashcat will attempt brute force but it's infeasible.

Brute force: all lowercase, uppercase, digits, and symbols

hashcat -m 0 -a 3 --increment --increment-min=1 --increment-max=6 password.hash ?l?u?d?s

```
(kali@kali)-[~]
$ # Brute force: all lowercase, uppercase, digits, and symbols
hashcat -m 0 -a 3 --increment --increment-min=1 --increment-max=6 password.hash ?l?u?d?s

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-skylake-avx512-11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz, 1424/2912 MB (512 MB allocatable), 2MCU

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

Host memory required for this attack: 0 MB

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

Session.....: hashcat
Status.....: Exhausted
```

```
Session.....: hashcat
Status.....: Exhausted
Hash.Mode.....: 0 (MD5)
Hash.Target.....: 380c67e00b5fafb40dafe9abcacb7ad5
Time.Started.....: Tue Aug 12 10:08:30 2025 (0 secs)
Time.Estimated...: Tue Aug 12 10:08:30 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.....: ?l [1]
Guess.Queue.....: 1/4 (25.00%)
Speed.#1.....: 88003 H/s (0.00ms) @ Accel:256 Loops:26 Thr:1 Vec:16
Recovered.....: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.....: 26/26 (100.00%)
Rejected.....: 0/26 (0.00%)
Restore.Point....: 1/1 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-26 Iteration:0-26
Candidate.Engine.: Device Generator
Candidates.#1....: s → x
Hardware.Mon.#1..: Util: 53%
```

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: <https://hashcat.net/faq/morework>

Approaching final keyspace - workload adjusted.

```
Session.....: hashcat
Status.....: Exhausted
Hash.Mode.....: 0 (MD5)
Hash.Target.....: 380c67e00b5fafb40dafe9abcacb7ad5
Time.Started.....: Tue Aug 12 10:08:30 2025 (0 secs)
Time.Estimated...: Tue Aug 12 10:08:30 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.....: ?l?u [2]
Guess.Queue.....: 2/4 (50.00%)
Speed.#1.....: 2419.5 kH/s (0.06ms) @ Accel:256 Loops:26 Thr:1 Vec:16
Recovered.....: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.....: 676/676 (100.00%)
Rejected.....: 0/676 (0.00%)
Restore.Point....: 26/26 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-26 Iteration:0-26
Candidate.Engine.: Device Generator
Candidates.#1....: sA → xQ
Hardware.Mon.#1..: Util: 69%
```

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).

Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: <https://hashcat.net/faq/morework>

Approaching final keyspace - workload adjusted.

```
Session.....: hashcat
Status.....: Exhausted
Hash.Mode.....: 0 (MD5)
Hash.Target.....: 380c67e00b5fafb40dafe9abcacb7ad5
Time.Started.....: Tue Aug 12 10:08:30 2025 (0 secs)
Time.Estimated...: Tue Aug 12 10:08:30 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.....: ?l?u?d [3]
Guess.Queue.....: 3/4 (75.00%)
Speed.#1.....: 36788.9 kH/s (0.06ms) @ Accel:256 Loops:26 Thr:1 Vec:16
Recovered.....: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.....: 6760/6760 (100.00%)
Rejected.....: 0/6760 (0.00%)
Restore.Point....: 260/260 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-26 Iteration:0-26
Candidate.Engine.: Device Generator
Candidates.#1....: sA1 → xQ8
Hardware.Mon.#1..: Util: 61%
```

Approaching final keyspace - workload adjusted.

```
Session.....: hashcat
Status.....: Exhausted
Hash.Mode.....: 0 (MD5)
Hash.Target.....: 380c67e00b5fafb40dafe9abcacb7ad5
Time.Started.....: Tue Aug 12 10:08:30 2025 (0 secs)
Time.Estimated...: Tue Aug 12 10:08:30 2025 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Mask.....: ?l?u?d?s [4]
Guess.Queue.....: 4/4 (100.00%)
Speed.#1.....: 51067.0 kH/s (0.11ms) @ Accel:256 Loops:26 Thr:1 Vec:16
Recovered.....: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.....: 223080/223080 (100.00%)
Rejected.....: 0/223080 (0.00%)
Restore.Point....: 8580/8580 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-26 Iteration:0-26
Candidate.Engine.: Device Generator
Candidates.#1....: sE3} → xQ8~
Hardware.Mon.#1..: Util: 55%
```

```
Started: Tue Aug 12 10:08:28 2025
Stopped: Tue Aug 12 10:08:32 2025
```

Brute Force Attack Simulation Result:

The difference in resistance between weak and strong passwords.

Password Type	Length	Charset Used	Hashcat Result	Time Taken	Notes
Weak Password (Pass12)	6	Upper, lower, digits	Cracked	Seconds	Easily guessed due to short length & simple pattern.
Strong Password (Improved)	16	Upper, lower, digits, symbols	Not Cracked	N/A (all short masks exhausted)	Resistant to brute force due to length & complexity.

Explanation:

- Brute force tries every possible combination until it finds the match.
- For Pass12 (6 chars, $\sim 56^6$ combinations) > feasible for attackers.
- For your improved password (16 chars, $\sim 95^{16}$ combinations) > would take billions of years at current speeds.
- Your Hashcat runs show **real attack speed** and why length + complexity matter.