

# How to Use the Script: Hashcat Cracker with Hashid

## 1. Prerequisites:

Before you begin using the script, ensure you have the following installed on your system:

- **Python 3.x:** Ensure Python is installed on your system.
- **Hashcat:** A powerful password recovery tool.
- **Hashid:** A tool to identify the type of hash.
- **Rockyou Wordlist:** A famous wordlist used by Hashcat for cracking passwords.

You can install `hashcat` and `hashid` using the following commands:

### Install hashcat:

bash

Copy code

```
sudo apt install hashcat
```

○

### Install hashid:

bash

Copy code

```
sudo apt install hashid
```

○

- **Wordlist (rockyou.txt):** Ensure you have the `rockyou.txt` wordlist on your system, which is used for cracking the hashes. Typically, it can be found at `/usr/share/wordlists/rockyou.txt`.

---

## 2. How to Run the Script:

- Download the script file (e.g., `hash_cracker.py`).
- Open your terminal and navigate to the directory where the script is located.

Run the script using Python:

bash

Copy code

```
python3 hash_cracker.py
```

- 

---

### 3. Working of the Script:

#### Step 1: Input Hash

The script will ask you to enter the hash that you want to crack:

```
plaintext
Copy code
Enter the hash: <Your hash here>
```

You can input any valid hash (MD5, SHA-1, NTLM, etc.).

#### Step 2: Hash Identification

Once you input the hash, the script will use **Hashid** to identify the possible hash modes based on the hash value. It will display something like this:

```
plaintext
Copy code
[*] Identified possible hashes:
    - MD5 (Mode: 0)
    - NTLM (Mode: 1000)
```

The script will then attempt to crack the hash using the identified modes.

#### Step 3: Cracking the Hash

For each hash mode, the script will attempt to crack the hash using **Hashcat** and the **rockyou.txt** wordlist. It will display the process as follows:

```
plaintext
Copy code
[*] Attempting to crack MD5 [Mode: 0] (1/2)
[*] Successfully cracked: password123
```

If the hash is successfully cracked, the script will log the result with a timestamp into a file called **hash\_cracking\_log.txt**:

plaintext

Copy code

[\*] Results logged in hash\_cracking\_log.txt

## Step 4: Log Files

The following log files will be generated:

1. **hashes.txt**: Contains the hash you entered for processing.
2. **hash\_cracking\_log.txt**: Logs the cracked results along with timestamps.

---

## 4. Manual for the Script:

### Functions Overview:

- **get\_user\_input():**
    - **Purpose:** Prompts the user to input a hash value.
    - **Input:** A valid hash.
    - **Output:** A hash string.
  - **save\_to\_file(filename, data):**
    - **Purpose:** Saves the entered hash to a file (e.g., **hashes.txt**).
    - **Input:** Filename and data (hash value).
    - **Output:** None.
  - **log\_results(log\_file, hash\_value, cracked\_value):**
    - **Purpose:** Logs the cracked hash result to a file with a timestamp.
    - **Input:** Log filename, hash value, and cracked value.
    - **Output:** None.
  - **identify\_all\_possible\_hash\_modes(hash\_value):**
    - **Purpose:** Identifies possible hash modes using the **hashid** tool.
    - **Input:** A hash value.
    - **Output:** A list of possible hash modes and their respective Hashcat mode values.
  - **fallback\_identify\_hash\_modes(hash\_value):**
    - **Purpose:** Provides fallback identification based on hash length.
    - **Input:** A hash value.
    - **Output:** A list of hash types and corresponding Hashcat modes.
  - **attempt\_cracking(hash\_file, mode):**
    - **Purpose:** Attempts to crack the hash using **Hashcat** and the **rockyou.txt** wordlist.
    - **Input:** The hash file and the hashcat mode.
    - **Output:** Cracked value or None.
-

## 5. Troubleshooting:

### Common Issues:

1. **Missing Dependencies:** If the script fails due to missing dependencies (like `hashcat` or `hashid`), ensure they are installed using the steps in the Prerequisites section.
  2. **Hash Type Not Identified:** If `hashid` cannot identify the hash type, the script will fall back on length-based identification. Ensure the hash is valid.
  3. **Wordlist Not Found:** If `rockyou.txt` is not found, make sure the wordlist exists in `/usr/share/wordlists/rockyou.txt`, or you can specify the path in the script.
  4. **No Cracking Results:** If the hash is not cracked, it may not exist in the wordlist, or the hash may require more advanced cracking techniques.
- 

## 6. Customization:

You can easily customize this script for specific use cases:

1. **Wordlist:** Change the wordlist location in the script by modifying the `wordlist` variable in the `attempt_cracking()` function.
2. **Hashcat Modes:** The script automatically uses Hashcat modes identified by `hashid`. If you want to adjust the hash modes, you can modify the `identify_all_possible_hash_modes()` function.

Example of using the tool:

Step1: Save the script in a .py name

Step2: Run the script using

Step3: Python3 script.py

Step4: Enter the hash and the script will do its job

```
(root@kali)-[/home/kali/thm/prj]
# python3 ad.py

          @@@@@@  @@@@@@  @@@@@@  @@@@@@  @@@  @@@  @@@@@@  @@@@@@
@@@@@  @@@@@@  @@@@@@  @@@@@@  @@@  @@@  @@@@@@  @@@@@@
!@@  !@@  @@!  @@  @@!  @@  @@!  !@@  @@!  @@
!@!  !@!  !@!  @!@  !@!  @!@  !@!  @!!  !@!  @!@
!!@@!!  !@!  @!@!!@  @!@!@!  @!@!@!  @!!!!  @!@!!@
!!@!!!  !!!  !!@!  !!!!!!  !!!!!!  !!!!!!  !!@!@!
!::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::
!::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::

A scripted cracker
@hashcat @hashid

Enter the hash: 482c811da5d5b4bc6d497ffa98491e38
[*] Hash saved to hashes.txt
[!] Hash modes not found using hashid. Falling back to length-based identification.
[*] Identified possible hashes:
- MD5 (Mode: 0)
[*] Attempting to crack MD5 [Mode: 0] (1/1)
[*] Successfully cracked: password123
[*] Results logged in hash_cracking_log.txt
```

Script-

```
import subprocess
import os
import datetime
```

```
# Banner
banner = ""
```

```
          @@@@@@  @@@@@@  @@@@@@  @@@@@@  @@@  @@@  @@@@@@  @@@@@@
@@@@@  @@@@@@  @@@@@@  @@@@@@  @@@  @@@  @@@@@@  @@@@@@
!@@  !@@  @@!  @@  @@!  @@  @@!  !@@  @@!  @@
!@!  !@!  !@!  @!@  !@!  @!@  !@!  @!!  !@!  @!@
!!@@!!  !@!  @!@!!@  @!@!@!  @!@!@!  @!@@!@!  @!!!!  @!@!!@
!!@!!!  !!!  !!@!  !!!!!!  !!!!!!  !!!!!!  !!@!@!
!::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::
!::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::

:::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::
::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::  ::
```

""

```
# Input and validation
def get_user_input():
    hash_value = input("Enter the hash: ").strip()
    if not hash_value or " " in hash_value:
        raise ValueError("[!] Invalid hash input.")
    return hash_value
```

```

# Save hash to file
def save_to_file(filename, data):
    with open(filename, "w") as file:
        file.write(data + "\n")
    print(f"[*] Hash saved to {filename}")

# Log results
def log_results(log_file, hash_value, cracked_value):
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    log_entry = f"[{timestamp}] Hash: {hash_value} | Cracked: {cracked_value}\n"
    with open(log_file, "a") as file:
        file.write(log_entry)
    print(f"[*] Results logged in {log_file}")

# Hash identification
def identify_all_possible_hash_modes(hash_value):
    try:
        result = subprocess.run(["hashid", hash_value], capture_output=True, text=True,
                                check=True)
        lines = result.stdout.strip().split("\n")
        hash_modes = []
        for line in lines:
            if "[Hashcat Mode:" in line:
                name = line.split("[")[0].strip() # This gets the name of the hash type
                mode = line.split("[Hashcat Mode: ")[-1].strip("]") # This extracts the mode
                hash_modes.append((name, mode))

        if not hash_modes:
            print("[!] Hash modes not found using hashid. Falling back to length-based
            identification.")
            return fallback_identify_hash_modes(hash_value)

        return hash_modes
    except Exception as e:
        print(f"[!] Error running hashid: {e}")
        return fallback_identify_hash_modes(hash_value)

# Fallback identification with NTLM rule
def fallback_identify_hash_modes(hash_value):
    if len(hash_value) == 32 and hash_value.isalnum() and any(c.isupper() for c in hash_value)
    and any(c.isdigit() for c in hash_value):
        return [("NTLM", "1000")]
    hash_length = len(hash_value)

```

```

length_map = {
    32: [("MD5", "0")],
    40: [("SHA-1", "100")],
    64: [("SHA-256", "1400")],
    128: [("SHA-512", "1700")]
}
return length_map.get(hash_length, [])

```

# Attempt cracking

```
def attempt_cracking(hash_file, mode):
```

```

    try:
        wordlist = "/usr/share/wordlists/rockyou.txt"
        if not os.path.exists(wordlist):
            raise FileNotFoundError("[!] Wordlist not found. Ensure rockyou.txt is present.")
        result = subprocess.run(
            ["hashcat", "-m", mode, hash_file, wordlist, "--quiet", "--potfile-disable"],
            capture_output=True, text=True, check=True
        )
        if result.stdout.strip():
            cracked_value = result.stdout.split("\n")[0].split(":")[-1].strip()
            return cracked_value
        return None
    except subprocess.CalledProcessError as e:
        print(f"[!] Hashcat error for mode {mode}: {e.stderr}")
        return None

```

# Main workflow

```
def main():
```

```

    try:
        print(banner)
        hash_value = get_user_input()
        save_to_file("hashes.txt", hash_value)

        hash_modes = identify_all_possible_hash_modes(hash_value)

        if not hash_modes:
            print("[!] Unable to identify hash modes. Please check the hash.")
            return

        print("[*] Identified possible hashes:")
        for name, mode in hash_modes:
            print(f"    - {name} (Mode: {mode})") # This line will now correctly print both name and
mode

```

```
for idx, (name, mode) in enumerate(hash_modes, 1):
    print(f"[*] Attempting to crack {name} [Mode: {mode}] ({idx}/{len(hash_modes)})")
    cracked_value = attempt_cracking("hashes.txt", mode)
    if cracked_value:
        print(f"[*] Successfully cracked: {cracked_value}")
        log_results("hash_cracking_log.txt", hash_value, cracked_value)
        return

    print("[!] Unable to crack the hash. Try alternative methods.")
except Exception as e:
    print(f"[!] An unexpected error occurred: {e}")

# Run the script
if __name__ == "__main__":
    main()
```