Title: Password Hash Cracking Tool

By :

CHAITANYA REDDY

CHETHAN KUMAR REDDY

Agenda

- Introduction
- Methodology
- Implementation
- Demonstration
- Results and Analysis
- Real-World Applications
- Discussion
- Conclusion

Introduction

Overview of Password Security and Cracking Techniques :

Passwords are a primary method of securing digital information. However, weak passwords can be easily compromised through various cracking techniques, such as brute-force attacks, dictionary attacks, and more advanced methods.

Purpose of the Tool :

Performs dictionary attacks on hashed passwords using pre-defined wordlists to identify the original passwords.

One of the main function is we can add 2 or more wordlists to identify the hashes.

It can decode the given hash and display the hash name.

Dictionary attack

Methodology

Dictionary Attacks :

Dictionary attacks use a wordlist to guess passwords. The tool hashes each word and compares it to the given hash, identifying the original password if a match is found. This method targets weak, common passwords.

Hashing Algorithms Used :

Supports:

✓ MD5: 32-character hash.
✓ SHA-1: 40-character hash.
✓ SHA-256: 64-character hash.
✓ SHA-512: 128-character hash.

This allows the tool to handle various hash types effectively.

Implementation

Key Functions and Code Structure :

- **Guess_hash_algorithm:** Guesses the hash algorithm based on the length of the hash string.
- **Hash_cracker:** Main function that performs the dictionary attack. It reads wordlists, hashes each word using the specified algorithm, and compares the hash to the target hash.
- **num_wordlists:** This function can handle multiple wordlists.
- **Print_header:** Displays the tool's header in the console.
- **Print_menu:** Displays the user menu for selecting options.

Manual Wordlist Creation

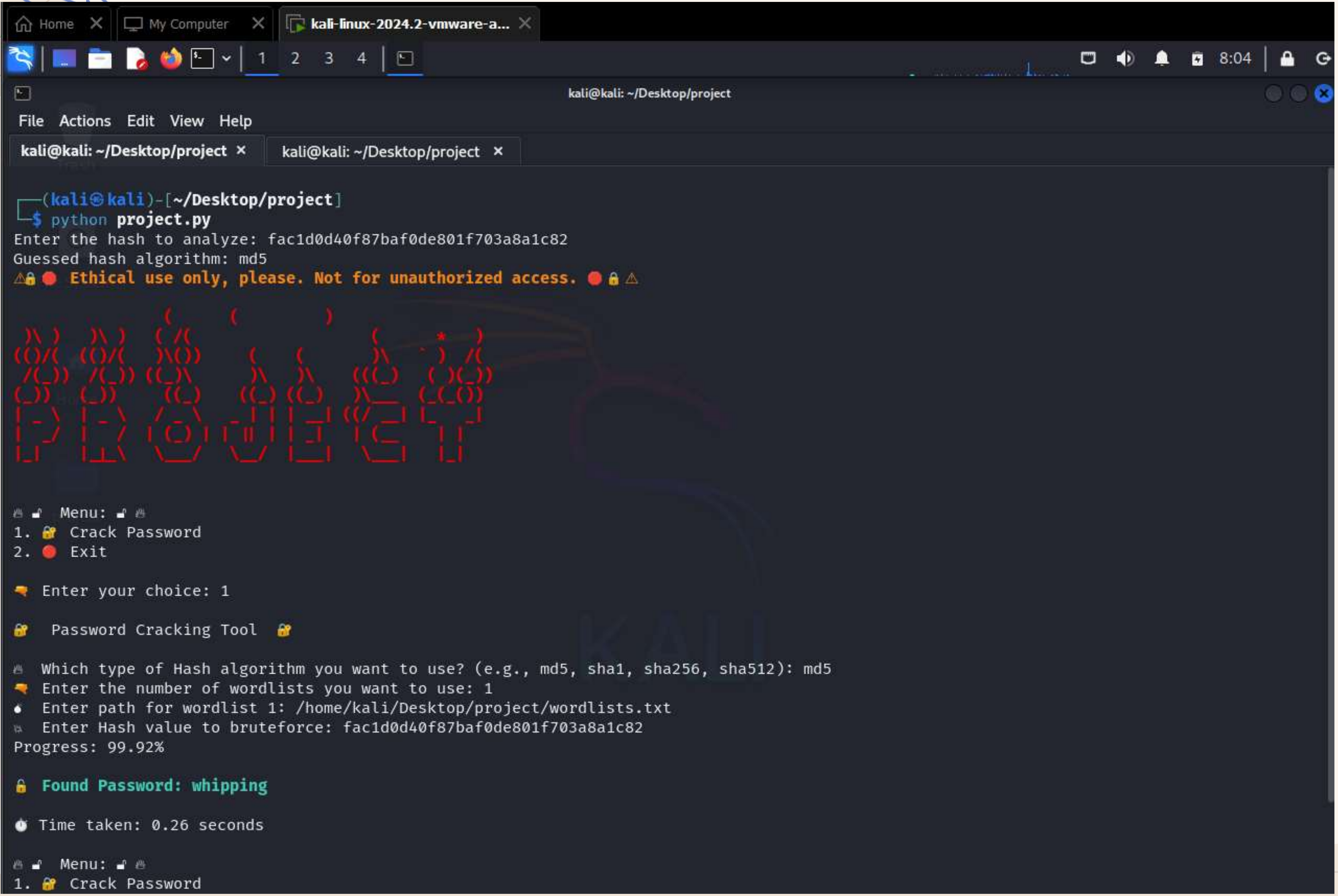
What is wordlist ?

Wordlists are collections of words, phrases, common Passwords and other strings of characters that are used in penetration testing and cybersecurity assessments to simulate attacks on systems.

How to Create a Wordlist ?

- Gather Passwords: Collect common or relevant passwords.
- Organize: List each password on a new line in a text file.
- Save: Save as .txt for use in attacks.

Demonstration of Tool

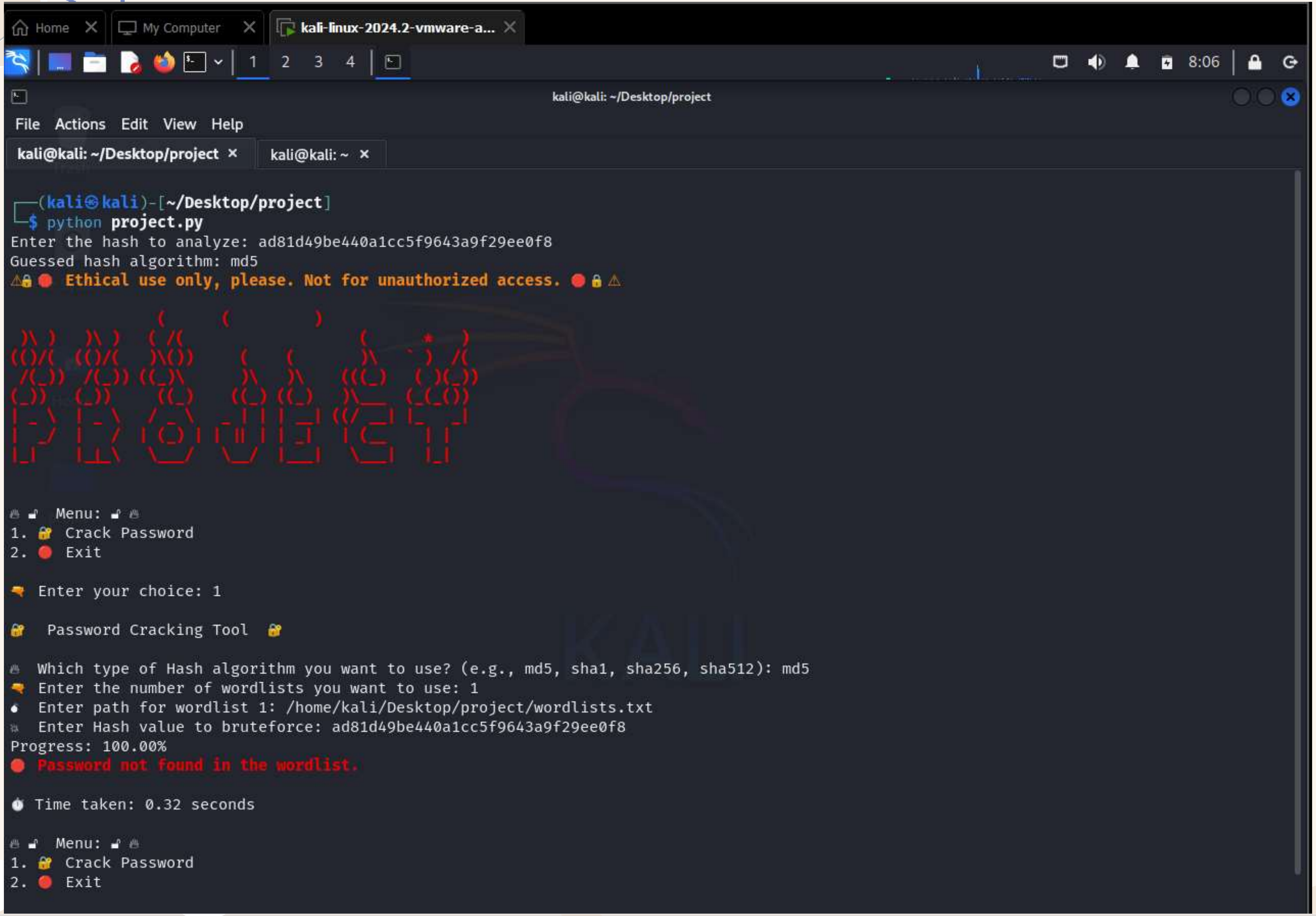


Home
My Computer
kali-linux-2024.2-vmware-a...
1 2 3 4
8:04
kali@kali: ~/Desktop/project
File Actions Edit View Help
kali@kali: ~/Desktop/project
kali@kali: ~/Desktop/project

```
┌──(kali㉿kali)-[~/Desktop/project]
└─$ python project.py
Enter the hash to analyze: fac1d0d40f87baf0de801f703a8a1c82
Guessed hash algorithm: md5
⚠🔒 🔴 Ethical use only, please. Not for unauthorized access. 🔴 🔒 ⚠

 🔓 Menu: 🔓
1. 🔐 Crack Password
2. 🔴 Exit

🔫 Enter your choice: 1

🔐  Password Cracking Tool  🔐

  Which type of Hash algorithm you want to use? (e.g., md5, sha1, sha256, sha512): md5
🔫 Enter the number of wordlists you want to use: 1
  Enter path for wordlist 1: /home/kali/Desktop/project/wordlists.txt
  Enter Hash value to bruteforce: fac1d0d40f87baf0de801f703a8a1c82
Progress: 99.92%

🔓 Found Password: whipping

⏱ Time taken: 0.26 seconds

 🔓 Menu: 🔓
1. 🔐 Crack Password
```

Here is successfully match the hash from the wordlist as shown in figure.

Demonstration of Tool

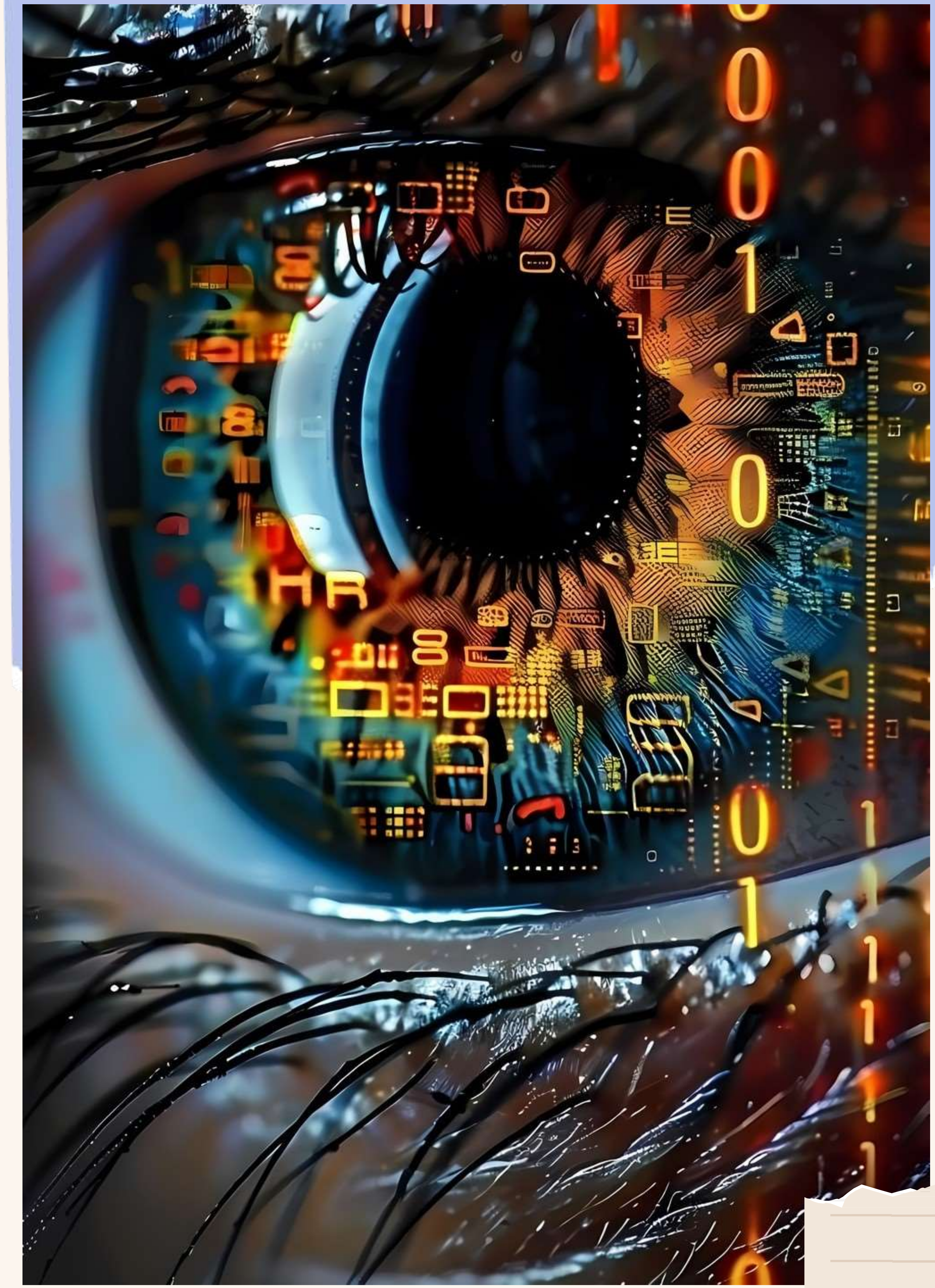

Home
My Computer
kali-linux-2024.2-vmware-a...
1
2
3
4
8:06
kali@kali: ~/Desktop/project
File
Actions
Edit
View
Help
kali@kali: ~/Desktop/project
kali@kali: ~

```
┌──(kali㉿kali)-[~/Desktop/project]
└─$ python project.py
Enter the hash to analyze: ad81d49be440a1cc5f9643a9f29ee0f8
Guessed hash algorithm: md5
⚠🔒🛑 Ethical use only, please. Not for unauthorized access. 🛑🔒⚠

🖐 🔓  Menu: 🔓 🖐
1. 🔐 Crack Password
2. 🛑 Exit

🔫 Enter your choice: 1

🔐  Password Cracking Tool  🔐

🖐  Which type of Hash algorithm you want to use? (e.g., md5, sha1, sha256, sha512): md5
🔫 Enter the number of wordlists you want to use: 1
💣 Enter path for wordlist 1: /home/kali/Desktop/project/wordlists.txt
☠ Enter Hash value to bruteforce: ad81d49be440a1cc5f9643a9f29ee0f8
Progress: 100.00%
🛑 Password not found in the wordlist.

⏱ Time taken: 0.32 seconds

🖐 🔓  Menu: 🔓 🖐
1. 🔐 Crack Password
2. 🛑 Exit
```

If Hash is not found
Displays as shown
in figure.

Real-World Applications and Scenarios

- ✓ Ethical Hacking: Find weak passwords in security assessments.
- ✓ Cybersecurity Training: Teach attack and defense techniques.
- ✓ Incident Response: Recover passwords during security breaches.
- ✓ Security Awareness: Show why strong passwords are essential.



Conclusion: The Future of Password Security

- **Summary :**
  - Developed a password hash-cracking tool using dictionary attacks with algorithms like MD5, SHA-1, SHA-256, and SHA-512. Demonstrates how common passwords can be cracked and highlights vulnerabilities.

- **Importance :**
  - Significance: Strong passwords are essential for protecting sensitive data. This project shows how weak passwords can be exploited, underscoring the need for complex and secure passwords.

- **Contribution :**
  - To Cybersecurity: Provides a practical tool for understanding password security and attacking methods. Helps in ethical hacking, training, and promoting better password practices.

CODE

```
import hashlib
import sys
import os
import time

def hash_cracker(wordlists, hash_to_decrypt, hash_algorithm):
    total_passwords = 0
    for wordlist in wordlists:
        if os.path.isfile(wordlist):
            total_passwords += sum(1 for _ in open(wordlist))
        else:
            print(f"'{wordlist}' is not a valid file.")

    if total_passwords == 0:
        print("No valid wordlists provided.")
        return None

    passwords_tried = 0

    for wordlist_path in wordlists:
        if not os.path.isfile(wordlist_path):
            continue

        try:
            with open(wordlist_path, 'r') as file:
                for line in file:
                    password = line.strip()
                    if hash_algorithm == 'md5':
                        hash_object = hashlib.md5(password.encode())
                    elif hash_algorithm == 'sha1':
                        hash_object = hashlib.sha1(password.encode())
                    elif hash_algorithm == 'sha256':
                        hash_object = hashlib.sha256(password.encode())
                    elif hash_algorithm == 'sha512':
                        hash_object = hashlib.sha512(password.encode())
                    else:
                        print(f"Hashing algorithm '{hash_algorithm}' not supported.")
                        return None

                    hashed_word = hash_object.hexdigest()
                    passwords_tried += 1
                    progress = (passwords_tried / total_passwords) * 100
                    print(f"\rProgress: {progress:.2f}%", end='', flush=True)

                    if hashed_word == hash_to_decrypt:
                        return password
        except FileNotFoundError:
            print(f"Wordlist file '{wordlist_path}' not found.")
            continue
        except Exception as e:
            print(f"Error reading wordlist file '{wordlist_path}': {str(e)}")
            continue

    return None

def print_header():
    title = """
           (    (       )
 )\ )  )\ )  ( /(             (     *  )
(()/(  (()/(  )\())    (   (      )\  ` )  /(
 /(_))  /(_)) ((_)\     )\  )\   (((_)  ( )(_))
(_))  (_))    ((_)   ((_) ((_)  )\___  (_(_())
| _ \ | _ \  / _ \  _ | | | __| ((/ __| |_   _|
|  _/ |   / | (_) | | || | | _|  | (__    | |
|_|   |_|_\  \___/   \__/  |___|  \___|   |_|
    """
    print("\033[1;31m" + title + "\033[0m")
```

```
def guess_hash_algorithm(hash_str):
    hash_length = len(hash_str)

    if hash_length == 32:
        return 'md5'
    elif hash_length == 40:
        return 'sha1'
    elif hash_length == 64:
        return 'sha256'
    elif hash_length == 128:
        return 'sha512'
    else:
        return 'Unknown or unsupported hash length'

# Example usage:
hash_to_decrypt = input("Enter the hash to analyze: ")
algorithm_guess = guess_hash_algorithm(hash_to_decrypt)
print(f"Guessed hash algorithm: {algorithm_guess}")


def print_menu():
    print("\n🔥🔓 Menu: 🔓🔥")
    print("1. 🔐 Crack Password")
    print("2. 🛑 Exit")

def main():
    attention_message = "⚠🔒🛑 Ethical use only, please. Not for unauthorized access. 🛑🔒⚠□"
    print("\033[1;33m" + attention_message + "\033[0m")

    print_header()
    while True:
        print_menu()
        choice = input("\n🔫 Enter your choice: ")

        if choice == '1':
            print("\n🔐 Password Cracking Tool 🔐\n")
            hash_algorithm = input("🔥 Which type of Hash algorithm you want to use? (e.g., md5, sha1, sha256, sha512): ").lower()
            if hash_algorithm not in ['md5', 'sha1', 'sha256', 'sha512']:
                print("⊘ Invalid hash algorithm.")
                continue
            num_wordlists = input("🔫 Enter the number of wordlists you want to use: ")
            if not num_wordlists.isdigit() or int(num_wordlists) <= 0:
                print("❌ Invalid number of wordlists.")
                continue
            num_wordlists = int(num_wordlists)
            wordlists = []
            for i in range(num_wordlists):
                wordlist_path = input(f"💣 Enter path for wordlist {i+1}: ")
                if not os.path.exists(wordlist_path):
                    print(f"🔍 Wordlist file '{wordlist_path}' not found.")
                    continue
                wordlists.append(wordlist_path)
            hash_to_decrypt = input("💥 Enter Hash value to bruteforce: ")

            start_time = time.time()
            cracked_password = hash_cracker(wordlists, hash_to_decrypt, hash_algorithm)
            end_time = time.time()

            if cracked_password:
                print(f"\n\n\033[1;32m🔓 Found Password: {cracked_password}\033[0m\n")
            else:
                print("\n\033[1;31m🛑 Password not found in the wordlist.\033[0m\n")

            print(f"□□ Time taken: {end_time - start_time:.2f} seconds")

        elif choice == '2':
            print("\n🚪 Exiting...")
            sys.exit()
        else:
            print("\n⛔ Invalid choice. Please select a valid option.")

if __name__ == "__main__":
    main()
```

Thanks!