|  |  |
| --- | --- |
| **Ex.No.2**  **Date:26/07/24** | **Perform password extraction, cracking and recovery from target system** |

|  |
| --- |
| **AIM:** |

* **To use open source software tools for extraction, cracking and recovery from target system.**
* **To implement Dictionary attack in Java/Python**

|  |
| --- |
| **THEORY:** |

**Tool Selected: John The Ripper**

**About the Tool :John the Ripper (JTR) is a free, open-source software tool used by hackers, both ethical and otherwise, for password cracking**

**Software Requirements :The software is typically used in a UNIV/Linux and Mac OS X environment where it can detect weak passwords.**

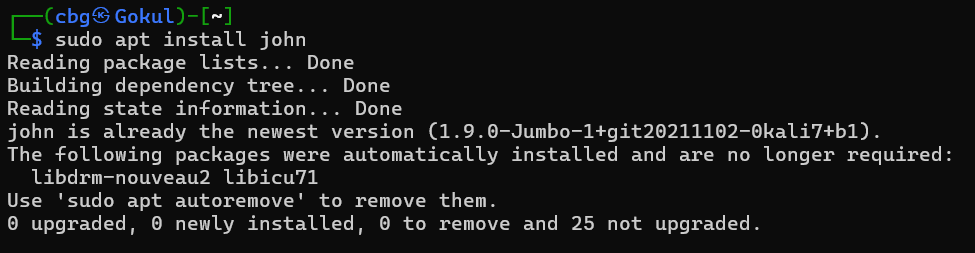
**Dictionary attack:**

**Dictionary attack is a technique used in cybersecurity, primarily for password cracking, where an attacker systematically enters every word in a pre-defined list (dictionary) to try and guess a password. This method exploits the tendency of users to choose common, easily guessable passwords.**

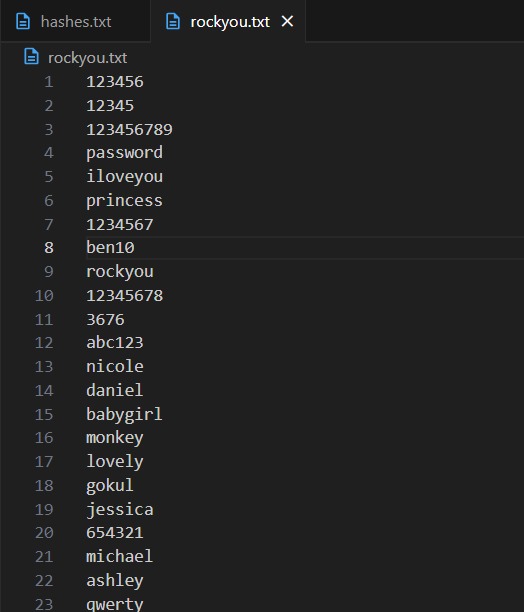
|  |
| --- |
| **Demonstrations (With Screen shots)** |

**Installation Procedure / Execution**

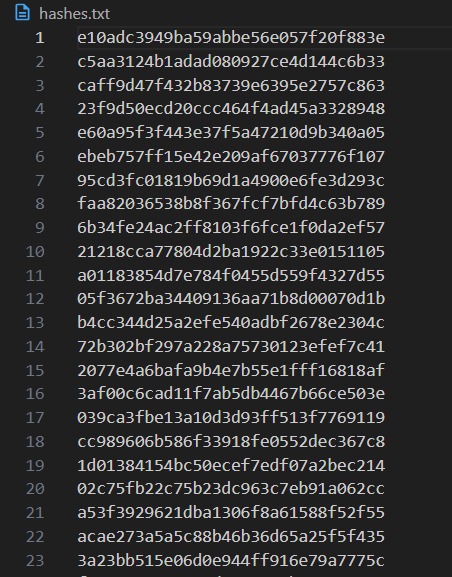
**apt install John**

****

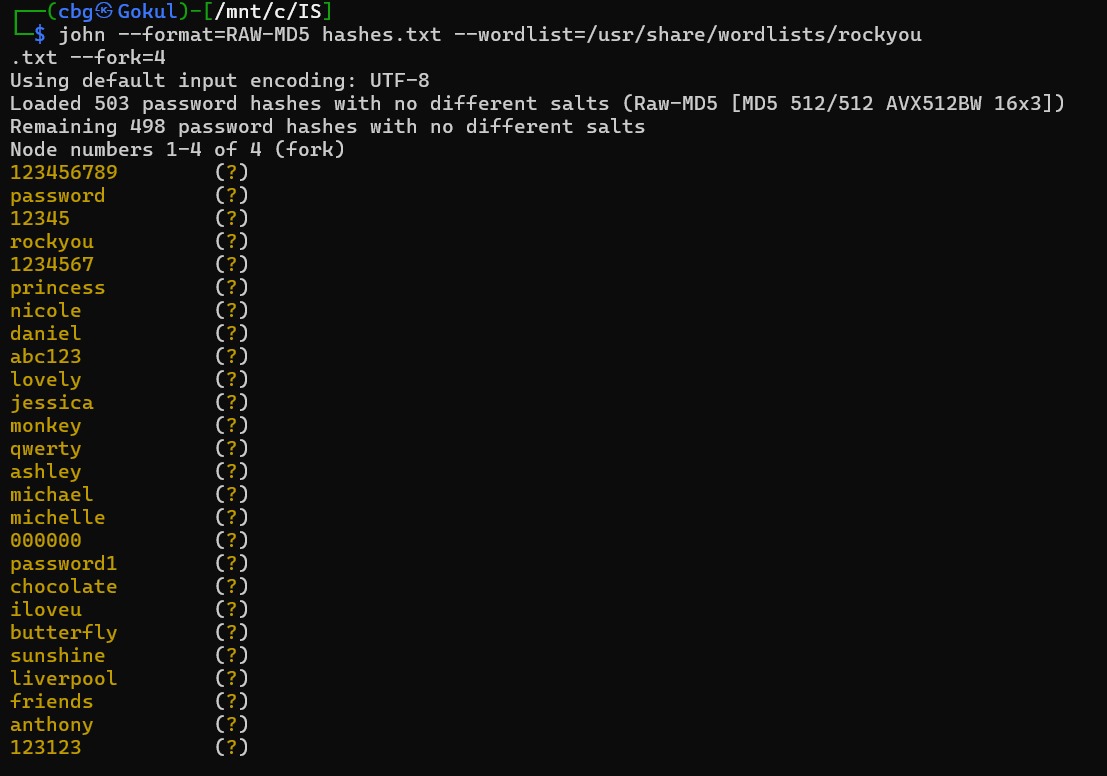
**dictionary of common passwords named rockyou.txt to compare it with the hash in hand.**

****

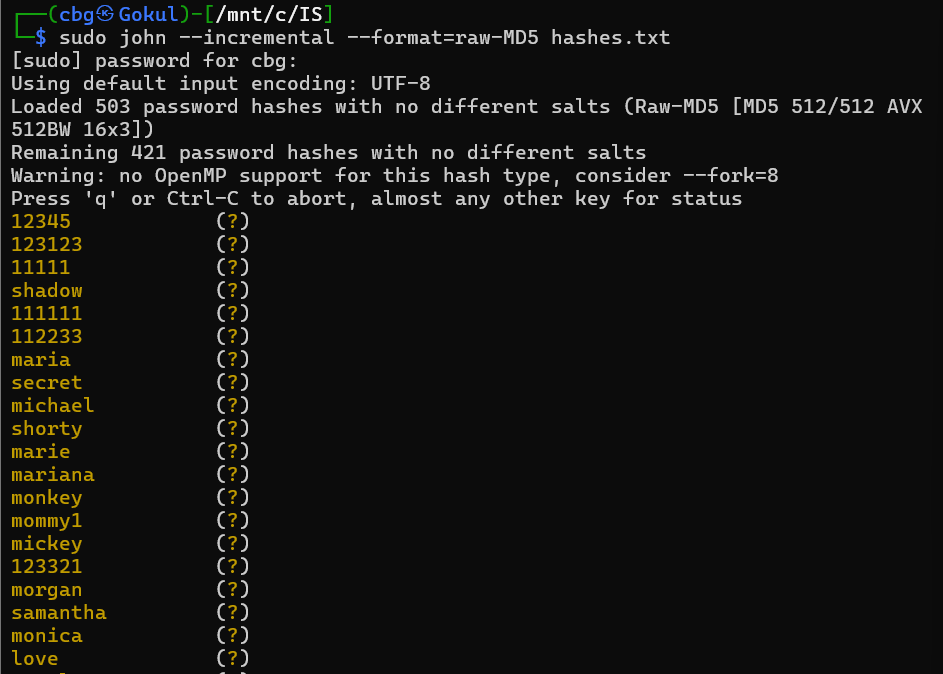
**Hashes:**

****

**Demonstrating Dictionary Attack**

****

**Demonstrating incremental attack (Brute Force)**

****

|  |
| --- |
| **Algorithm for Dictionary Attack** |

1. **Load the dictionary file containing possible passwords.**
2. **Initialize the cipher suite using the provided key.**
3. **For each password in the dictionary:**
   1. **Try to decrypt the target password.**
      1. **If decryption is successful and the password matches, return the password.**
      2. **If decryption fails, continue to the next password.**
   2. **Catch any error in case of any effort in opening the dictionary file or loading the hash.**
4. **If no password matches, output that the password was not found.**

|  |
| --- |
| **Coding** |

import hashlib

import itertools

import time

import matplotlib.pyplot as plt

import random

def hash\_password(password):

return hashlib.sha256(password.encode()).hexdigest()

def create\_dictionary(char\_set, max\_size):

word\_dict = {}

for combo in itertools.product(char\_set, repeat=max\_size):

word = ''.join(combo)

word\_dict[word] = hash\_password(word)

return word\_dict

def launch\_dict\_attack(word\_dict, password\_table):

username = 'user'

hashed\_password = password\_table[username]

for word, hashed\_word in word\_dict.items():

if hashed\_word == hashed\_password:

return word

return None

def draw\_plot(x\_data, y\_data, x\_title, y\_title):

plt.plot(x\_data, y\_data, marker='o', color='red', linestyle='-')

plt.xlabel(x\_title)

plt.ylabel(y\_title)

plt.title(f'{x\_title} vs {y\_title}')

plt.grid(True)

plt.show()

def charset\_vs\_time(charset):

pin\_size = 4

cur\_set = ""

time\_taken = []

sizes = [i + 1 for i in range(len(charset))]

for char in charset:

cur\_set += char

word\_dict = create\_dictionary(cur\_set, pin\_size)

random\_pin = "".join(random.choice(cur\_set) for \_ in range(pin\_size))

password\_table = {'user': hash\_password(random\_pin)}

start\_time = time.time()

launch\_dict\_attack(word\_dict, password\_table)

end\_time = time.time()

time\_taken.append((end\_time - start\_time) \* 1000) # Convert to milliseconds

print(time\_taken)

draw\_plot(sizes, time\_taken, x\_title='Character Set Size', y\_title='Time taken (ms)')

def create\_password\_table():

password\_table = {}

while True:

username = input("Enter username (or type 'done' to finish): ")

if username == 'done':

break

password = input("Enter password: ")

password\_table[username] = hash\_password(password)

return password\_table

def display\_password\_table(password\_table):

for username, hashed\_password in password\_table.items():

print(f"Username: {username}, Hashed Password: {hashed\_password}")

def dictionary\_attack(username, password\_table, word\_dict):

if username not in password\_table:

print("Username not found.")

return None

hashed\_password = password\_table[username]

for word, hashed\_word in word\_dict.items():

if hashed\_word == hashed\_password:

return word

return None

def plot\_time\_vs\_length(lengths, times):

plt.plot(lengths, times, marker='o', color='red', linestyle='-')

plt.xlabel('Password Length')

plt.ylabel('Time taken (milliseconds)')

plt.title('Password Length vs Time taken')

plt.grid(True)

plt.show()

def main():

char\_set = input("Enter character set (e.g., abcdefghijklmnopqrstuvwxyz): ")

length = int(input("Enter password length (e.g., 5): "))

while True:

print("\nMenu:")

print("1. Create dictionary of all possible combinations and their hashes")

print("2. Create password table")

print("3. Display password table")

print("4. Launch dictionary attack")

print("5. Plot Time vs Password Length")

print("6. Plot Time vs Character Set Size")

print("7. Exit")

choice = input("Enter your choice: ")

if choice == '1':

word\_dict = create\_dictionary(char\_set, length)

print("Dictionary created.")

elif choice == '2':

password\_table = create\_password\_table()

print("Password table created.")

elif choice == '3':

if 'password\_table' in locals():

display\_password\_table(password\_table)

else:

print("Password table not created yet.")

elif choice == '4':

username = input("Enter username to attack: ")

if 'password\_table' in locals() and 'word\_dict' in locals():

start\_time = time.time()

password = dictionary\_attack(username, password\_table, word\_dict)

end\_time = time.time()

if password:

print(f"Password found: {password}")

else:

print("Password not found.")

print(f"Time taken: {(end\_time - start\_time) \* 1000:.2f} milliseconds")

plot\_time\_vs\_length([length], [(end\_time - start\_time) \* 1000])

else:

print("Dictionary or password table not created yet.")

elif choice == '5':

lengths = [i for i in range(1, 6)]

length\_times = []

for length in lengths:

word\_dict = create\_dictionary(char\_set, length)

random\_pin = ''.join(random.choice(char\_set) for \_ in range(length))

password\_table = {'user': hash\_password(random\_pin)}

start\_time = time.time()

launch\_dict\_attack(word\_dict, password\_table)

end\_time = time.time()

length\_times.append((end\_time - start\_time) \* 1000) # Convert to milliseconds

plot\_time\_vs\_length(lengths, length\_times)

elif choice == '6':

charset\_vs\_time(char\_set)

elif choice == '7':

break

else:

print("Invalid choice. Please try again.")

if \_name\_ == "\_main\_":

main()

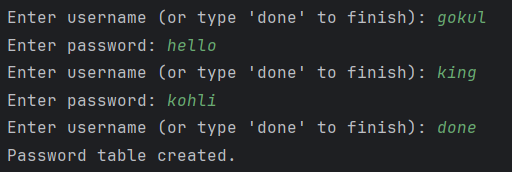
|  |
| --- |
| **Screenshots** |

**Creating dictionary of all possible combinations and their hashes**



****

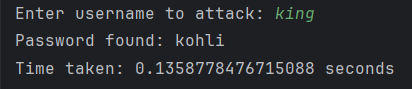
**Creating password table:**

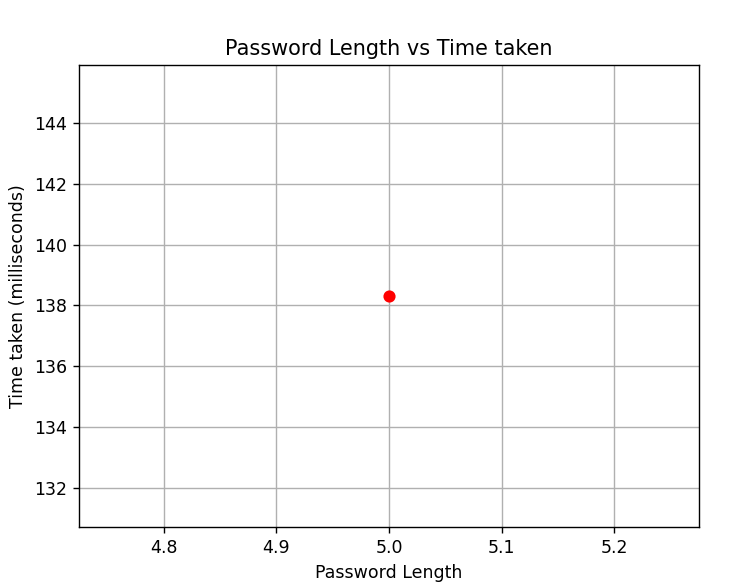
****

**Displaying password table:**

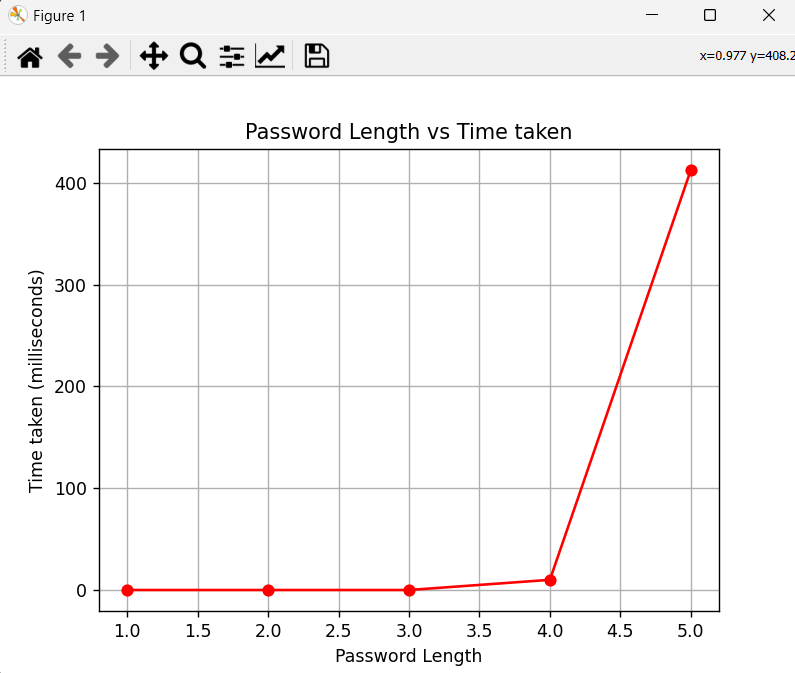
****

**Launching dictionary attack:**

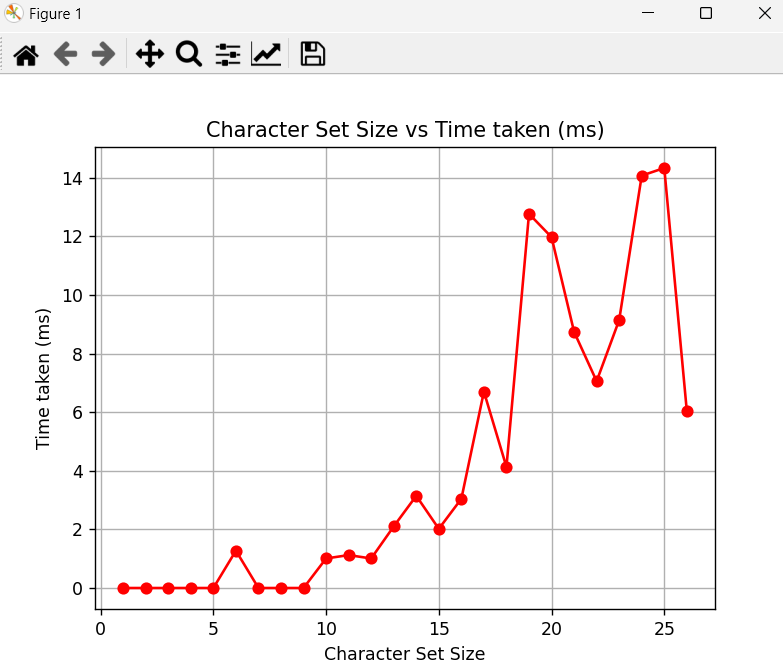
****

****

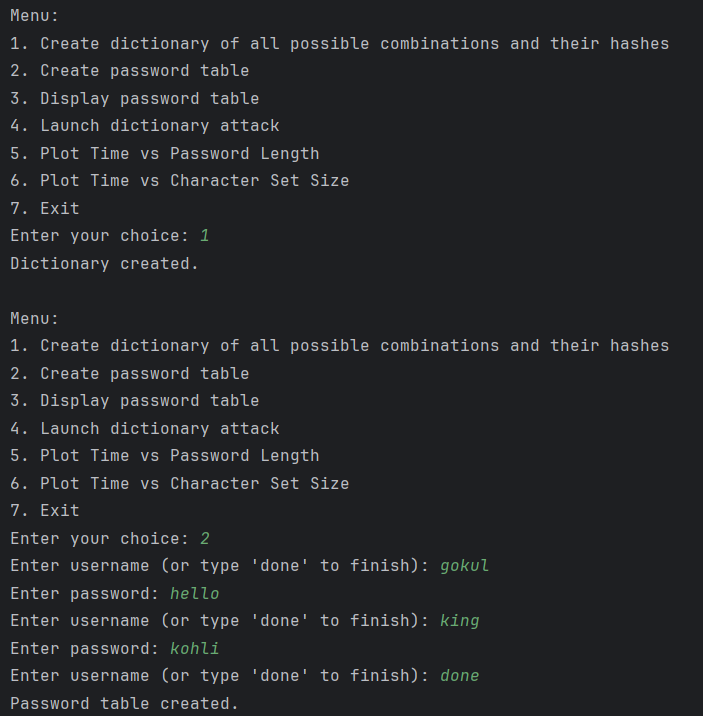
**Plot Time vs Password Length**

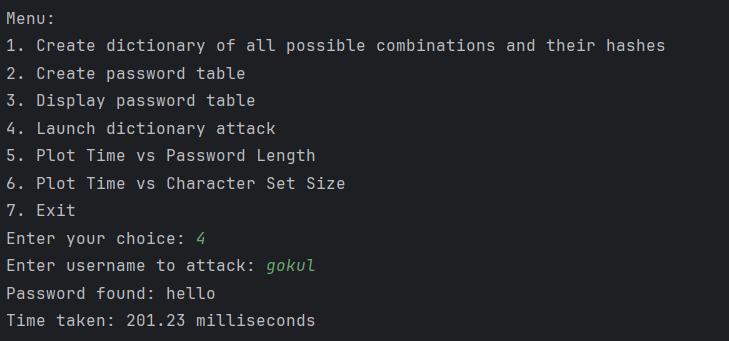
****

**Plot Time vs Character Set Size**

****

**Final Output:**

****

****

**Password cracking (Dictionary attack)**

1. Create hash for the password that you need to check for also for easy implementation know the key.
2. Now use dictionary attack to find the password from the list of passwords.

|  |
| --- |
| **Output** |

**Key takeaway from the installation and execution of password recovery tool:**

1. A good dictionary file is must for a smooth and easy password cracking.
2. The tool must be versatile supporting multiple encryption formats, using multiple cores of the system thus making the tedious process to be completed within minimal time.
3. It must be used in a legal and ethical way to crack passwords in case of need.

|  |
| --- |
| **RESULT:** |

The password cracking tool has been installed and password recovery has been done.

|  |
| --- |
| **Evaluation** |

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Max Marks** | **Marks Obtained** |
| **Uniqueness of the Tool (Installation and Exploration of Functionalities)** | **20** |  |
| **Uniqueness of Code for Dictionary Attack** | **15** |  |
| **Completion of experiment on time** | **5** |  |
| **Documentation** | **10** |  |
| **Total** | **50** |  |
| **Signature of the faculty with Date** |  |  |