

ITIS 6167/8167: Network Security

Bonus Project

Hash Cracking

Title: Designing Smartest Strategy to Crack Passwords.

Author: Rahul Banerjee (801322462)

Abstract:

This project explores the application of a Python-based program complemented with the Hashcat application to efficiently crack SHA-1 hashed passwords from a given "passwords.txt" file. The core strategy employed in this project revolves around custom Python scripts, adept at handling patterns commonly found in password formations such as digit sequences, English words or phrases, and combinations of words with numbers, all based on inputs from a "dictionary.txt" file. Recognizing the limitations in processing power and efficiency, the project strategically incorporates Hashcat, a specialized password cracking tool, to tackle instances where the Python script encounters high CPU utilization and efficiency constraints. This hybrid approach not only optimizes the cracking process but also significantly expands the range of passwords that can be decrypted. The final report delves into the specifics of the methodology, highlighting how the combination of Python programming and Hashcat's brute-force capabilities enables the successful decryption of a substantial portion of the hashed passwords. The results demonstrate the effectiveness of using a mixed-approach in password cracking, balancing computational resources with the need for extensive coverage and depth in decryption capabilities.

Keywords: hashes, sha-1, hashcat, password, plaintextpasswords, dictionarypassword, online safety, cybersecurity, passwordcracking.

Introduction:

As we delve into the intricate world of digital security, the significance of understanding and testing the robustness of password encryption becomes paramount. In our recent project, we embarked on an ambitious journey to decrypt SHA-1 hashed passwords sourced from a file named "password_sha1_RB.txt". This undertaking was not just a mere exercise in decryption, but also a challenging endeavor to efficiently manage and optimize our computational resources.

At the heart of our strategy was a versatile Python script, meticulously crafted to dissect and interpret common password patterns. This script was tailored to decode various sequences that are frequently found in password compositions, including simple numerical strings, English words, phrases, and even their combinations with numbers. These patterns were carefully extracted from an input source - "dictionary.txt" file, which served as a foundational reference for our initial decryption attempts.

However, the complexity of this task was compounded by concerns about processing efficiency. Recognizing the limitations inherent in Python scripting, especially when it comes to CPU usage and speed, we made a strategic decision to incorporate Hashcat, a powerful tool renowned for its prowess in password recovery. Hashcat's integration was pivotal, especially in scenarios where our Python script encountered computational bottlenecks.

By harmonizing the methodical capabilities of Python with the brute-force strength of Hashcat, our approach sought to strike an optimal balance. This fusion was aimed at not just enhancing the cracking process but also in significantly widening the spectrum of passwords that could be decrypted.

In the following sections, I'll walk you through the nuances of our methodology, the specific challenges we faced, and how this synergistic approach contributed to the successful decryption of a substantial portion of the hashed passwords. The insights gleaned from this project are a testament to the efficacy of employing a diverse set of tools to tackle complex cybersecurity challenges, particularly in the realm of password cracking.

Methodology:

The approach centered on employing Python programming to develop an application for hash cracking. This project incorporated several layers, each tailored to different hashes, utilizing various strategies to decipher the hash values. This was achieved by leveraging a plain text 'dictionary.txt' file.

Environment Setup:

a) Python installation

Python can be installed by visiting the official website and downloading the version that is suitable for the operating system. For users of Windows, it is crucial that the "Add Python to PATH" option be selected during installation to provide convenient command-line access. Once the download is complete, the installer should be run, with the prompts being followed. The downloaded executable file should be executed by Windows users, while users of macOS should open the .dmg file and follow the installation wizard. The installation should be verified by opening a command prompt or terminal and typing `Python -version` or `python3 --version`.

b) Hashcat installation

To install Hashcat on Windows, you need to visit the Hashcat website at <https://hashcat.net/hashcat/> and download the latest version of the software available under "Binaries". After downloading, extract the contents of the .zip file to a folder of your choice. To run Hashcat, open this folder and execute `hashcat64.exe` for 64-bit systems or `hashcat32.exe` for 32-bit systems. In this project Kali Linux was used to run the hashcat where the application comes preinstalled. To launch the application we just need to give command 'hashcat -arguments'.

c) Sublime Text Editor installation

The Sublime Text IDE can be downloaded from the official website. To start the installer, double-click it when it has been downloaded. Accept the licensing agreement, pick the installation location (the default is typically good), and select any extra tasks, such as establishing shortcuts, as directed by the installation wizard. Sublime Text may be launched from the Start Menu or searched for in the Windows search bar after installation. This text editor is now available to use on our Windows system for coding tasks.

• Used tools and technologies:

The tool was developed with Python because of its comprehensive libraries and user-friendly scripting for network interactions. Scapy facilitated the packet creation process, eliminating the need to construct packets from the ground up. Sublime text editor was used as an IDE to write the code. Wireshark was utilized to capture and examine the packets, allowing for a thorough analysis of the attack scenarios.

- **Simulation Design:**

- I. **Reading the Hashes:** Use Python to read the "**password_sha1_RB.txt**" file. Store the hashes in a suitable data structure (e.g., a list or a set).
- II. **Loading the Dictionary:** Read dictionary.txt and load words into a list for password generation.
- III. **Password Generation Strategies:** Utilize 'itertools' library to generate password candidates. This includes: Single words from the dictionary. Combinations of words (e.g., two or three words concatenated). Variations with numbers appended or prepended. For each generated password candidate, use 'hashlib' to compute its SHA-1 hash.
- IV. **Hash Matching:** Compare each generated hash with the list of target hashes from passwords.txt. If a match is found, record the plaintext password alongside its corresponding hash.
- V. **Performance Optimization:** Implement multiprocessing or multithreading to parallelize the hash generation and comparison process, enhancing the simulation's efficiency.
- VI. **Logging and Output:** Maintain a log of all matched passwords and their hashes. Output the results to a file or display them in the console for analysis.
- VII. **Testing and Validation:** Test the simulation with known hash-password pairs to validate its accuracy. Adjust the password generation strategies based on the success rate.
- VIII. **Resource Management:** Monitor CPU and memory usage. Implement efficient data structures to minimize memory overhead.

- **Implementation:**

The "password_sha1_RB.txt" file contains 20 SHA-1 hashed values, of which 2 are single-word passwords, 6 are numeric passwords, 4 consist of double-word combinations (word+word), 4 are alphanumeric passwords (word+number), and 4 are passwords formed by a three-word combination (word+word+word).

- I. **Cracking the single word and double word passwords:**

A simple Python script was used to crack the single word and double word passwords. For more understanding, please see the screenshot provided below.

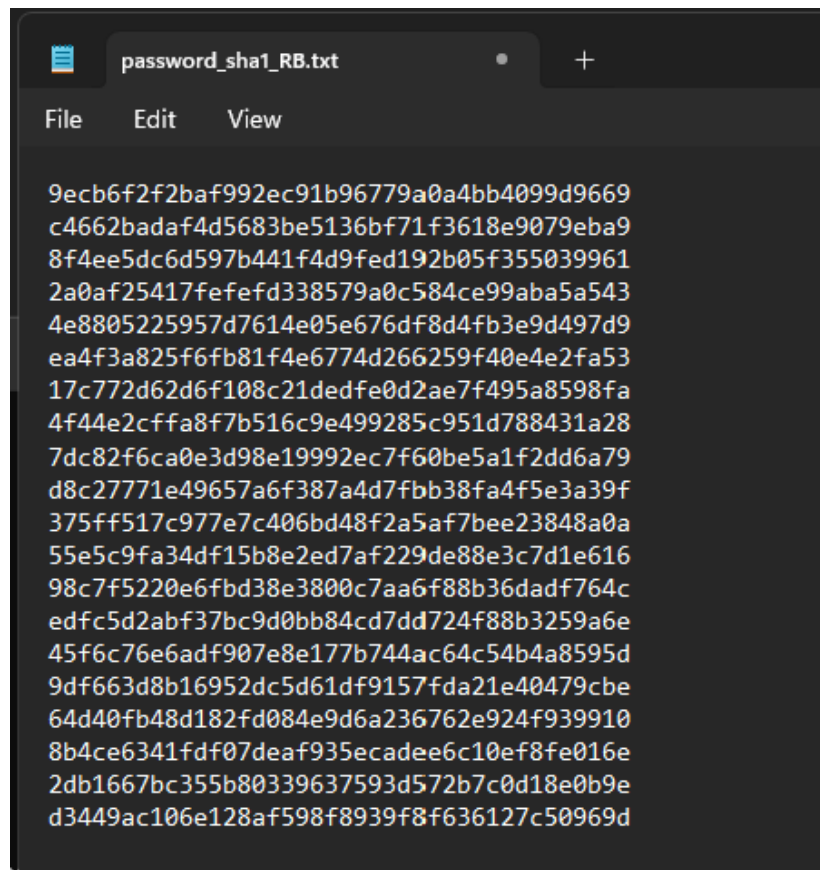


Fig 1 Hash file consist of 20 SHA -1 hash values

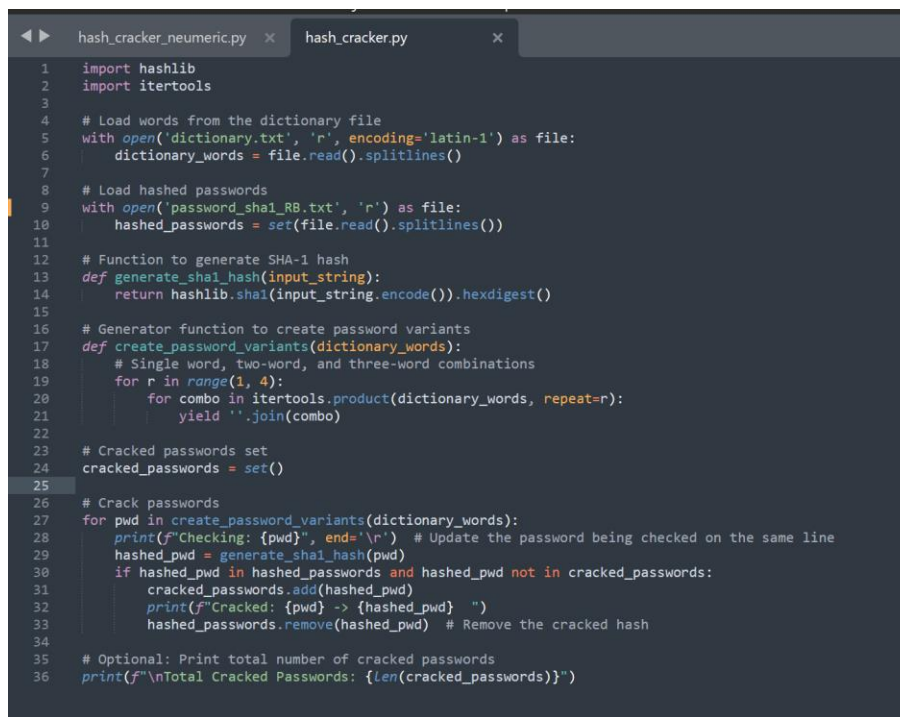


Fig 2 Source code to crack the single word and the double word passwords

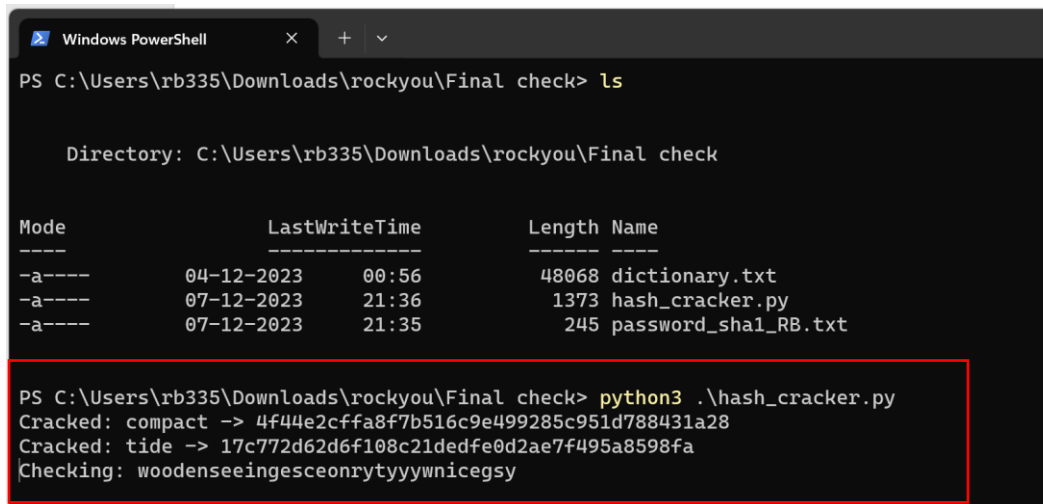
Explanation:

This Python script cracks SHA-1 hashed passwords by:

- Importing necessary modules: 'hashlib' for hashing and 'itertools' for creating word combinations.
- Reading words from 'dictionary.txt' and storing them in a list.
- Reading hashed passwords from 'password_sha1_RB.txt' and storing them in a set.
- Defining a function to generate SHA-1 hashes of given strings.
- Creating a generator function to yield single, double, and triple word combinations from the dictionary.
- Iterating through these combinations, hashing each, and checking against the provided hashed passwords.
- Keeping track of and printing cracked passwords, and counting the total number cracked.

Upon running the script "**hash_cracker.py**" in the command line using the command "**python3 .\hash_cracker.py**", the process of hash cracking will commence.

See below screenshot for reference.



```
Windows PowerShell
PS C:\Users\rb335\Downloads\rockyou\Final check> ls

Directory: C:\Users\rb335\Downloads\rockyou\Final check

Mode                LastWriteTime         Length Name
----                -
-a----            04-12-2023    00:56         48068 dictionary.txt
-a----            07-12-2023    21:36          1373 hash_cracker.py
-a----            07-12-2023    21:35           245 password_sha1_RB.txt

PS C:\Users\rb335\Downloads\rockyou\Final check> python3 .\hash_cracker.py
Cracked: compact -> 4f44e2cffa8f7b516c9e499285c951d788431a28
Cracked: tide -> 17c772d62d6f108c21dedfe0d2ae7f495a8598fa
Checking: woodenseeingesceonrytyyywnicegsy
```

Fig 3 The password cracking process has been started.

```

Windows PowerShell
PS C:\Users\rb335\Downloads\rockyou\Final check> ls

Directory: C:\Users\rb335\Downloads\rockyou\Final check

Mode                LastWriteTime         Length Name
----                -
-a----             04-12-2023      00:56         48068 dictionary.txt
-a----             07-12-2023      21:36          1373 hash_cracker.py
-a----             07-12-2023      21:35           245 password_sha1_RB.txt

PS C:\Users\rb335\Downloads\rockyou\Final check> python3 .\hash_cracker.py
Cracked: compact -> 4f44e2cffa8f7b516c9e499285c951d788431a28
Cracked: tide -> 17c772d62d6f108c21dedfe0d2ae7f495a8598fa
Cracked: mannstart -> 55e5c9fa34df15b8e2ed7af229de88e3c7d1e616
Cracked: wolfaged -> 7dc82f6ca0e3d98e19992ec7f60be5a1f2dd6a79
Cracked: mentionedappointment -> d8c27771e49657a6f387a4d7fbb38fa4f5e3a39f
Cracked: opportunitytell -> 375ff517c977e7c406bd48f2a5af7bee23848a0a

```

Fig 4 fter the completion of the cracking

Once the process was done we found the plain text result of the following hash values.

Plain Text	Hash Values	Status
compact	4f44e2cffa8f7b516c9e499285c951d788431a28	Cracked
tide	17c772d62d6f108c21dedfe0d2ae7f495a8598fa	Cracked
mannstart	55e5c9fa34df15b8e2ed7af229de88e3c7d1e616	Cracked
wolfaged	7dc82f6ca0e3d98e19992ec7f60be5a1f2dd6a79	Cracked
mentionedappointment	d8c27771e49657a6f387a4d7fbb38fa4f5e3a39f	Cracked
opportunitytell	375ff517c977e7c406bd48f2a5af7bee23848a0a	Cracked

Fig 5 Plain Text values of the 6 cracked hashes

All the cracked hash values have been removed from the main hash file "**password_sha1_RB.txt**", leaving it with 14 unique password hash values. Following this, we began focusing on cracking numeric password combinations.

Cracking the Numeric Password combination:

To decipher the hashes of numeric passwords, a new Python script was developed. This script is designed to generate numeric key combinations, exploring every possibility from a single-digit numeric password up to a 10-digit numeric password.

See below screenshot for more reference.

```
File Edit Selection Find View Goto Tools Project Preferences Help
hash_cracker_neumeric.py x
1 import hashlib
2
3 # Load hashed passwords
4 with open('password_sha1_RB.txt', 'r') as file:
5     hashed_passwords = set(file.read().splitlines())
6
7 # Function to generate SHA-1 hash
8 def generate_sha1_hash(input_string):
9     return hashlib.sha1(input_string.encode()).hexdigest()
10
11 # Function to generate numeric patterns
12 def generate_numeric_patterns(max_digits=10):
13     for length in range(1, max_digits + 1):
14         for number in range(10**length):
15             yield str(number).zfill(length)
16
17 # Cracked passwords set
18 cracked_passwords = set()
19
20 # Crack passwords
21 for pwd in generate_numeric_patterns():
22     print(f"Checking: {pwd}", end='\r') # Update the password being checked on the same line
23     hashed_pwd = generate_sha1_hash(pwd)
24     if hashed_pwd in hashed_passwords and hashed_pwd not in cracked_passwords:
25         cracked_passwords.add(hashed_pwd)
26         print(f"Cracked: {pwd} -> {hashed_pwd} ")
27         hashed_passwords.remove(hashed_pwd) # Remove the cracked hash
28
29 # Optional: Print total number of cracked passwords
30 print(f"\nTotal Cracked Passwords: {len(cracked_passwords)}")
31
```

Fig 6 Source code to crack the numeric passwords

Explanation:

The Python script is structured for cracking numeric SHA-1 hashed passwords, summarized by the following key concepts:

- I. **Use of hashlib:** It employs Python's hashlib module for SHA-1 hashing.
- II. **Reading Hashes:** The script loads SHA-1 hashed passwords from 'password_sha1_RB.txt' into a set for efficient searching.
- III. **SHA-1 Hash Function:** It defines generate_sha1_hash to compute SHA-1 hashes of given numeric strings.
- IV. **Numeric Pattern Generation:** The generate_numeric_patterns function creates sequential numeric strings (1 to 10 digits long) as password candidates.
- V. **Cracking Process:** The script iterates through these generated numeric patterns, hashes each, and checks for matches in the loaded hash set.
- VI. **Tracking Cracked Passwords:** Successfully cracked hashes are tracked in the cracked_passwords set and removed from the original hash list.
- VII. **Output:** Live progress updates are shown during the cracking process, with a final count of cracked passwords upon completion.

```
Windows PowerShell
PS C:\Users\rb335\Downloads\rockyou\Final check> ls

Directory: C:\Users\rb335\Downloads\rockyou\Final check

Mode                LastWriteTime         Length Name
----                -
-a----             04-12-2023     00:56         48068 dictionary.txt
-a----             07-12-2023     21:36          1373 hash_cracker.py
-a----             22-11-2023     12:58          1122 hash_cracker_neumeric.py
-a----             07-12-2023     22:06           250 password_sha1_RB.txt
-a----             07-12-2023     21:35           245 single_double_word_password_sha1_RB.txt

PS C:\Users\rb335\Downloads\rockyou\Final check> python3 .\hash_cracker_neumeric.py
Checking: 9216291
```

Fig 7 Hash cracking of the numeric passwords has been started

```
Windows PowerShell
PS C:\Users\rb335\Downloads\rockyou\Final check> ls

Directory: C:\Users\rb335\Downloads\rockyou\Final check

Mode                LastWriteTime         Length Name
----                -
-a----             04-12-2023     00:56         48068 dictionary.txt
-a----             07-12-2023     21:36          1373 hash_cracker.py
-a----             22-11-2023     12:58          1122 hash_cracker_neumeric.py
-a----             07-12-2023     22:06           250 password_sha1_RB.txt
-a----             07-12-2023     21:35           245 single_double_word_password_sha1_RB.txt

PS C:\Users\rb335\Downloads\rockyou\Final check> python3 .\hash_cracker_neumeric.py
Cracked: 14431328 -> 2a0af25417fefefd338579a0c584ce99aba5a543
Checking: 14830932
```

Fig 8 Completion of the first numeric password crack

```
Windows PowerShell

Mode                LastWriteTime         Length Name
----                -
-a----             22-11-2023     02:43         48083 dictionary.txt
-a----             22-11-2023     12:55          1373 hash_cracker.py
-a----             13-11-2023     14:53          2856 hash_cracker_bird.py
-a----             22-11-2023     12:58          1122 hash_cracker_neumeric
-a----             22-11-2023     12:58          1122 hash_cracker_neumeric.py
-a----             20-11-2023     16:24           744 password_cracker
-a----             13-11-2023     01:06           840 password_sha1_RB.txt
-a----             21-11-2023     23:05          1188 pass_cracker.py
-a----             21-11-2023     15:29          1293 pattern_checker.py
-a----             23-09-2015     12:41       139921497 rockyou.txt

PS C:\Users\rb335\Downloads\rockyou> python3 .\hash_cracker_neumeric.py
Cracked: 14431328 -> 2a0af25417fefefd338579a0c584ce99aba5a543
Cracked: 17889651 -> 9ecb6f2f2baf992ec91b96779a0a4bb4099d9669
Cracked: 18148312 -> 4e8805225957d7614e05e676df8d4fb3e9d497d9
Cracked: 61385971 -> 8f4ee5dc6d597b441f4d9fed192b05f355039961
Cracked: 65859399 -> ea4f3a825f6fb81f4e6774d266259f40e4e2fa53
Cracked: 97435957 -> c4662badaf4d5683be5136bf71f3618e9079eba9
```

Fig 9 The final cracked numeric passwords

Once this step was completed we found the total of 6 plaintext consisting of single word and double word passwords and numeric digits of password. The below table consists of all the found numeric digit passwords.

Plain Text	Hash Values	Status
14431328	2a0af25417fefefd338579a0c584ce99aba5a543	Cracked
17889651	9ecb6f2f2baf992ec91b96779a0a4bb4099d9669	Cracked
18148312	4e8805225957d7614e05e676df8d4fb3e9d497d9	Cracked
61385971	8f4ee5dc6d597b441f4d9fed192b05f355039961	Cracked
65859399	ea4f3a825f6fb81f4e6774d266259f40e4e2fa53	Cracked
97435957	c4662badaf4d5683be5136bf71f3618e9079eba9	Cracked

Fig 10 Cracked numeric passwords

Now all the cracked 6 numeric hash values have been removed from the main hash file "**password_sha1_RB.txt**", leaving it with 8 unique password hash values. Following this, we began focusing on cracking three word combination password.

Cracking the three word combination:

To crack the 3 word combination password we again created a custom python script which would take the input from the "dictionary.txt" file 3 times and will create a password combination to match it with the original "**password_sha1_RB.txt**" to crack the passwords.

See the below screenshot for more details

```

3word_combination_crack.py x
1  import hashlib
2  import itertools
3  import codecs
4
5  # Function to generate SHA-1 hash
6  def generate_sha1_hash(input_string):
7      return hashlib.sha1(input_string.encode()).hexdigest()
8
9  # Load words from the dictionary file, handling BOM if present
10 with codecs.open('dictionary.txt', 'r', encoding='utf-8-sig') as file:
11     dictionary_words = file.read().splitlines()
12
13 # Load hashed passwords
14 with open('password_sha1_RB.txt', 'r') as file:
15     hashed_passwords = set(file.read().splitlines())
16
17 # Cracked passwords set
18 cracked_passwords = set()
19
20 # Crack passwords
21 for word_combo in itertools.product(dictionary_words, repeat=3):
22     combined_word = ''.join(word_combo)
23     hashed_combined_word = generate_sha1_hash(combined_word)
24
25     # Live checking status
26     print(f"Checking: {combined_word} -> {hashed_combined_word}", end='\r')
27
28     if hashed_combined_word in hashed_passwords and hashed_combined_word not in cracked_passwords:
29         cracked_passwords.add(hashed_combined_word)
30         print(f"\nCracked: {combined_word} -> {hashed_combined_word}")
31
32 # Print total number of cracked passwords
33 print(f"\nTotal Cracked Passwords: {len(cracked_passwords)}")
34

```

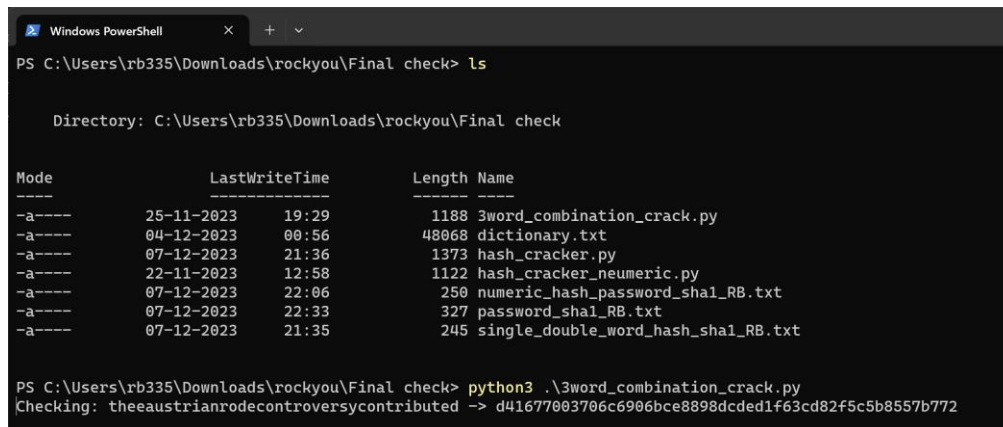
Fig 11 Source code to crack the three word combination hash value

Explanation:

This Python script is tailored for cracking SHA-1 hashed passwords using a dictionary-based approach. It employs the hashlib module to generate SHA-1 hashes and itertools to create combinations of words from a dictionary. Initially, it loads words from dictionary.txt, handling any potential Byte Order Mark (BOM) with codecs. Simultaneously, it loads a set of hashed passwords from password_sha1_RB.txt for comparison. The script iterates through up to three-word combinations from the dictionary, hashing each combination to check against the loaded hashes. A set, cracked_passwords, tracks the hashes of successfully cracked passwords. The script continuously updates the progress and, upon completion, reports the total number of cracked passwords. This approach effectively automates a brute-force attack, leveraging a combination of dictionary words to match against known SHA-1 hashes.

Once we execute the command **"python3 3word_combination_crack.py"** in the command line, it will start check the hash against the plaintext password, creating a combination of word+word+word.

See below image for more details



```
Windows PowerShell
PS C:\Users\rb335\Downloads\rockyou\Final check> ls

Directory: C:\Users\rb335\Downloads\rockyou\Final check

Mode                LastWriteTime         Length Name
----                -
-a-----         25-11-2023         19:29         1188 3word_combination_crack.py
-a-----         04-12-2023         00:56        48068 dictionary.txt
-a-----         07-12-2023         21:36         1373 hash_cracker.py
-a-----         22-11-2023         12:58         1122 hash_cracker_neumeric.py
-a-----         07-12-2023         22:06          250 numeric_hash_password_sha1_RB.txt
-a-----         07-12-2023         22:33          327 password_sha1_RB.txt
-a-----         07-12-2023         21:35          245 single_double_word_hash_sha1_RB.txt

PS C:\Users\rb335\Downloads\rockyou\Final check> python3 .\3word_combination_crack.py
Checking: theaustrianrodecontroversycontributed -> d41677003706c6906bce8898dcdded1f63cd82f5c5b8557b772
```

Fig 12 The hash cracking of the 3 word combination has been started

However as this program was taking a great amount of time crack the password, to specify the task an automated password cracking tool named "Hashcat" was used.

Process of using the Hashcat:

Hashcat is a well-known and highly regarded password recovery tool, widely used for its efficiency and versatility in cracking various types of cryptographic hash functions.

Hashcat's combination attack mode was used for this purpose. The combination attack mode is a password cracking strategy that merges two wordlists, concatenating each word from the first list with every word from the second. This approach is more efficient than a brute-force attack and is especially useful when passwords are believed to be combinations of known elements like words and numbers. Users can tailor the wordlists to the specific context for greater effectiveness. In Hashcat, this mode is activated using the -a 1 flag. It strikes a balance between the speed of a dictionary attack and the comprehensiveness of a brute-force attack, making it an adaptable and powerful tool in various password cracking scenarios.

We have used **kali linux** operating system, which is used for penetration testing, where the Hashcat application comes inbuilt with some other open source tools like **crunch**, which we have used in this project in the later section.

```
elliott@kali:~/Downloads/Password Crack/Hash cracking$ sudo chmod 777 nubers.txt
[sudo] password for elliot:
elliott@kali:~/Downloads/Password Crack/Hash cracking$ sudo su
[sudo] password for elliot:
[elliott@kali]~/Downloads/Password Crack/Hash cracking
#hashcat -a 1 dictionary.txt dictionary.txt --stdout > new_wrd.txt
[elliott@kali]~/Downloads/Password Crack/Hash cracking
#
```

Fig 13 Creation of the two_word combination with the name new_wrd.txt

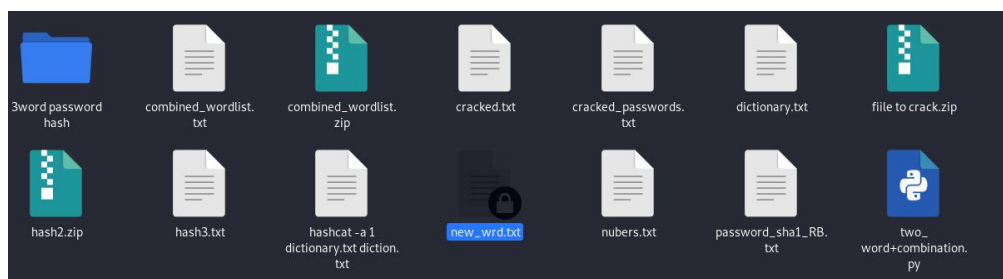


Fig 14 The file new_word.txt was created which was locked

```
[elliott@kali]~/Downloads/Password Crack/Hash cracking
#pwd
/home/elliott/Downloads/Password Crack/Hash cracking
[elliott@kali]~/Downloads/Password Crack/Hash cracking
#chmod 777 new_wrd.txt
[elliott@kali]~/Downloads/Password Crack/Hash cracking
#
```

Fig 15 The command used to remove the lock to access the file without root privilege

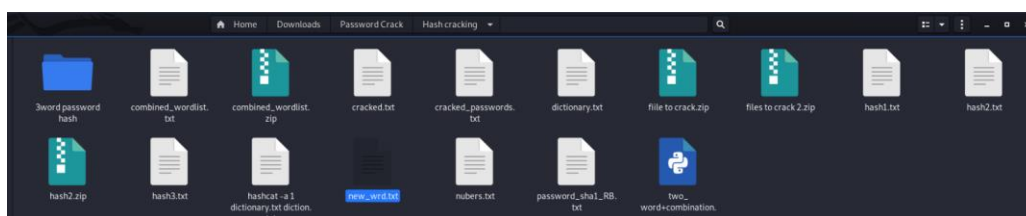


Fig 16 The lock was removed from the new_wrd.txt

```
[elliott@kali]~/Downloads/Password Crack/Hash cracking
#chmod 777 cracked
cracked_passwords.txt cracked.txt
[elliott@kali]~/Downloads/Password Crack/Hash cracking
#chmod 777 cracked_passwords.txt
[elliott@kali]~/Downloads/Password Crack/Hash cracking
#hashcat -m 100 -a 1 password_sha1_RB.txt dictionary.txt new_wrd.txt -o cracked_passwords.txt --force
```

Fig 17 Started hashcat to perform the brute force password cracking using the combination of 3 words

```
Session.....: hashcat
Status.....: Cracked
Hash.Type.....: SHA1
Hash.Target.....: password_sha1_RB.txt
Time.Started.....: Thu Dec 7 03:24:31 2023 (2 hours, 2 mins)
Time.Estimated....: Thu Dec 7 05:27:12 2023 (0 secs)
Guess.Base.....: File (dictionary.txt), Left Side
Guess.Mod.....: File (combined_wordlist.txt), Right Side
Speed.#1.....: 21153.0 kH/s (4.37ms) @ Accel:512 Loops:64 Thr:1 Vec:8
Recovered.....: 4/4 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 157711703808/173741112000 (90.77%)
Rejected.....: 0/157711703808 (0.00%)
Restore.Point....: 4096/5580 (73.41%)
Restore.Sub.#1...: Salt:0 Amplifier:20334848-20334912 Iteration:0-64
Candidates.#1....: sugarinterferecouple -> patterninterfereposted

Started: Thu Dec 7 03:24:30 2023
Stopped: Thu Dec 7 05:27:14 2023

[root@kali]~/Downloads/Password Crack/Hash cracking
#chmod 777 cracked
cracked_passwords.txt cracked.txt
[root@kali]~/Downloads/Password Crack/Hash cracking
#chmod 777 cracked_passwords.txt
[root@kali]~/Downloads/Password Crack/Hash cracking
#
```

Fig 18 Password cracking successful

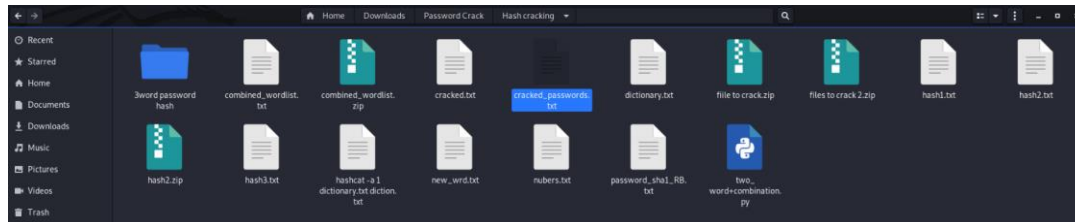


Fig 19 The file cracked_password.txt

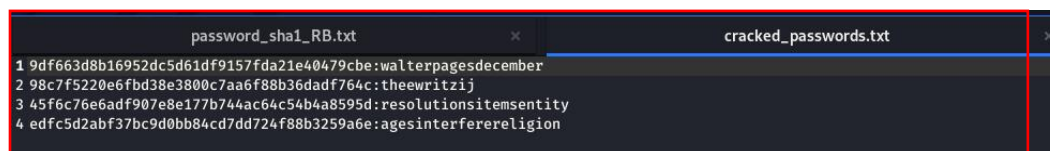


Fig 20 Cracked Hash file consist of all the three word passwords

Explanation: Here we have create a word list of two word combination with the help of this command **“hashcat -a 1 dictionary.txt dictionary.txt --stdout > new_wrd.txt”**. Once the txt file was created it was locked due to privileges, to give the file read write execute permission we use the command **“chmod 777 new_wrd.txt”**.

After this step we started hashcat to start cracking the passwords using this command **“hashcat -m 100 -a 1 password_sha1_RB.txt dictionary.txt new_wrd.txt -o cracked_passwords.txt -force”** where it specifies a combination attack on SHA-1 hash types. The command uses the flags -m 100 to denote the

SHA-1 hash mode, and -a 1 to set the attack mode to combination, where words from dictionary.txt and new_wrd.txt are paired together to crack the hashes in password_sha1_RB.txt. The cracked passwords are then saved into cracked_passwords.txt. The inclusion of the -force flag in the command indicates that Hashcat should bypass any warnings or safety checks and proceed with the operation.

Once this step was done a new file was create with the name of “**cracked_password.txt**” which consists of all the 3 word combination passwords as shown in the above image.

Plain Text	Hash Values	Status
walterpagesdecember	9df663d8b16952dc5d61df9157fda21e40479cbe	Cracked
thewritzij	98c7f5220e6fbd38e3800c7aa6f88b36dadf764c	Cracked
resolutionsitemsentity	45f6c76e6adf907e8e177b744ac64c54b4a8595d	Cracked
agesinterferereligion	edfc5d2abf37bc9d0bb84cd7dd724f88b3259a6e	Cracked

Fig 21 Cracked 3word password hash

Now once this step was completed, the total 16 hash value was cracked and according to the project statement we were only left with the combination of alphanumeric password.

Cracking Alphanumeric Password:

Hashcat was used to crack these hashes. As there is an involvement of neumeric characters and we used the combination mode of hashcat, we had to create a wordlist of numeric characters. For this we used the tool name ‘**crunch**’. Crunch is a versatile command-line tool primarily used for generating custom wordlists in password cracking and security testing. It allows users to create tailored wordlists based on specific criteria, such as length, character set, and pattern. This feature makes it particularly useful in penetration testing and ethical hacking, where specific password formats are known. Crunch's ability to pipe generated wordlists directly to other password cracking tools like Hashcat or John the Ripper enhances its functionality and efficiency. Capable of producing large-scale wordlists, Crunch is a powerful tool in the cybersecurity toolkit. Being open-source, it offers flexibility for customization and wide compatibility, predominantly with Unix-like systems, making it a valuable asset for security professionals.

```
[ / ]-[root@kali]-[/home/elliott/Downloads/Password Crack/Hash cracking]
#crunch 1 4 0123456789 -o nubers.txt > dev/null 2>&1
```

Fig 22 Crunch to create a wordlist of numbers starting with 1 digit and ending with 4 digits

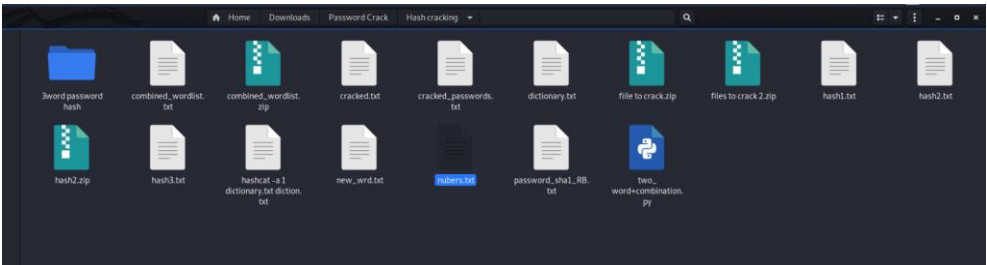


Fig 23 File was created with the name numbers.txt

```
#hashcat -m 100 -a 1 password_sha1_RB.txt dictionary.txt combined_wordlist.txt -o cracked_passwords.txt --force C
[ / ]-[root@kali]-[/home/elliott/Downloads/Password Crack/Hash cracking]
#hashcat -m 100 -a 1 password sha1 RB.txt.txt dictionary.txt nubers.txt -o cracked.txt --force
```

Fig 24 Command to crack the hashes with the combination of words and numbers


```
elliott@kali: ~/Downloads/Password Crack/file to crack
8b4ce6341fdf07deaf935ecadeec10ef8fe016e:gazed724
Started: Thu Dec 7 03:08:24 2023 127c50969d:read5839
Stopped: Thu Dec 7 03:08:24 2023 0d18e0b9e:globe7751
root@kali: [/home/elliott/Downloads/Password Crack/Hash cracking]
#hashcat -m 100 -a 1 password_sha1_RB.txt dictionary.txt nubers.txt -o cracked.txt --force
hashcat (v5.1.0) starting...

OpenCL Platform #1: The pocl project
=====
* Device #1: pthread-Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 4096/13929 MB allocatable, 8MCU

Dictionary cache hit:
* Filename..: dictionary.txt
* Passwords.: 5580
* Bytes.....: 48068
* Keyspace..: 5580

Dictionary cache built:
* Filename..: nubers.txt
* Passwords.: 11110
* Bytes.....: 54328
* Keyspace..: 11110
* Runtime...: 0 secs

Hashes: 8 digests; 8 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Applicable optimizers:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Salt
* Raw-Hash

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

ATTENTION! Pure (unoptimized) OpenCL kernels selected.
This enables cracking passwords and salts > length 32 but for the price of drastically reduced performance.
If you want to switch to optimized OpenCL kernels, append -O to your commandline.

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

* Device #1: build_opts '-cl-std=CL1.2 -I OpenCL -I /usr/share/hashcat/OpenCL -D LOCAL_MEM_TYPE=2 -D VENDOR_ID=64 -D CUDA_ARCH=0 -D AMD_ROCM=0 -D VECT
Me5 -D KERN_TYPE=100 -D _unroll'
Dictionary cache hit:
* Filename..: dictionary.txt
* Passwords.: 5580
* Bytes.....: 48068
```

Fig 25 Hashcat started cracking the hashes

```
Session.....: hashcat
Status.....: Exhausted
Hash.Type.....: SHA1
Hash.Target.....: password_sha1_RB.txt
Time.Started.....: Thu Dec 7 03:08:46 2023 (3 secs)
Time.Estimated...: Thu Dec 7 03:08:49 2023 (0 secs)
Guess.Base.....: File (dictionary.txt), Left Side
Guess.Mod.....: File (nubers.txt), Right Side
Speed.#1.....: 22655.4 kH/s (9.06ms) @ Accel:256 Loops:128 Thr:1 Vec:8
Recovered.....: 4/8 (50.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 61993800/61993800 (100.00%)
Rejected.....: 0/61993800 (0.00%)
Restore.Point....: 5580/5580 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:11008-11110 Iteration:0-128
Candidates.#1....: sugar9898 -> pattern9999

Started: Thu Dec 7 03:08:45 2023
Stopped: Thu Dec 7 03:08:50 2023

root@kali: [/home/elliott/Downloads/Password Crack/Hash cracking]
#cat cracked.txt
8b4ce6341fdf07deaf935ecadeec10ef8fe016e:gazed724
d3449ac106e128af598f8939f8f636127c50969d:read5839
2db1667bc355b80339637593d572b7c0d18e0b9e:globe7751
64d40fb48d182fd084e9d6a236762e924f939910:africa610

root@kali: [/home/elliott/Downloads/Password Crack/Hash cracking]
#sudo chmod 777 cracked.txt
```

Fig 26 Hashcat successfully cracked all the passwords and the content of the cracked.txt file

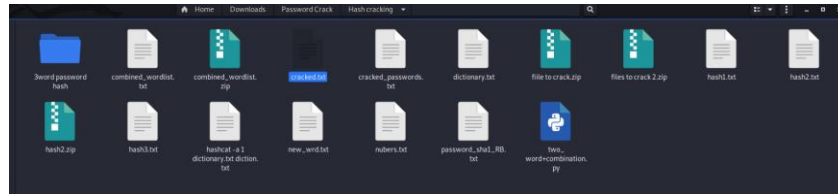


Fig 27 The file cracked.txt was created

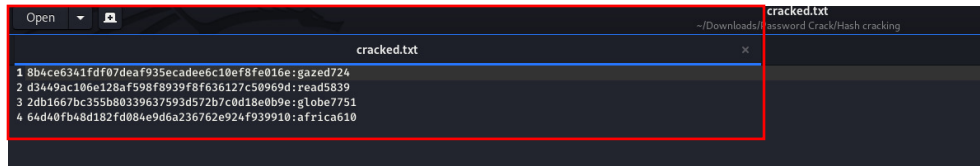


Fig 28 The password hashes of cracked.txt contains the combination of word and numbers

Explanation: Here we used crunch to create the word list of the numbers using the command **“crunch 1 4 0123456789 -o nubers.txt > dev/null 2>&1”**. Then we used hashcat to start the attack on the password hashes, the command we used for this was **“hashcat -m 100 -a 1 password_sha1_RB.txt.txt dictionary.txt nubers.txt -o cracked.txt –force”**. Once hashcat successfully cracked all the hash a new file was generated with the name cracked.txt which contains all the cracked passwords as shows above.

Plain Text	Hash Values	Status
gazed724	8b4ce6341fdf07deaf935ecadee6c10ef8fe016e	Cracked
read5839	d3449ac106e128af598f8939f8f636127c50969d	Cracked
globe7751	2db1667bc355b80339637593d572b7c0d18e0b9e	Cracked
africa610	64d40fb48d182fd084e9d6a236762e924f939910	Cracked

Fig 29 Cracked alphanumeric password hashes

The below is the Cracked hash table. With the clear password

	A	B	C
1	Plain Text	Hash Values	Status
2	14431328	2a0af25417fefefd338579a0c584ce99aba5a543	Cracked
3	17889651	9ecb6f2f2baf992ec91b96779a0a4bb4099d9669	Cracked
4	18148312	4e8805225957d7614e05e676df8d4fb3e9d497d	Cracked
5	61385971	8f4ee5dc6d597b441f4d9fed192b05f355039961	Cracked
6	65859399	ea4f3a825f6fb81f4e6774d266259f40e4e2fa53	Cracked
7	97435957	c4662badaf4d5683be5136bf71f3618e9079eba9	Cracked
8	compact	4f44e2cffa8f7b516c9e499285c951d788431a28	Cracked
9	tide	17c772d62d6f108c21dedfe0d2ae7f495a8598fa	Cracked
10	mannstart	55e5c9fa34df15b8e2ed7af229de88e3c7d1e616	Cracked
11	wolfaged	7dc82f6ca0e3d98e19992ec7f60be5a1f2dd6a79	Cracked
12	mentionedappointment	d8c27771e49657a6f387a4d7fbb38fa4f5e3a39f	Cracked
13	opportunitytell	375ff517c977e7c406bd48f2a5af7bee23848a0a	Cracked
14	walterpagesdecember	9df663d8b16952dc5d61df9157fda21e40479cbe	Cracked
15	theewritzij	98c7f5220e6fbd38e3800c7aa6f88b36dadf764c	Cracked
16	resolutionsitemsentity	45f6c76e6adf907e8e177b744ac64c54b4a8595d	Cracked
17	agesinterferereligion	edfc5d2abf37bc9d0bb84cd7dd724f88b3259a6e	Cracked
18	gazed724	8b4ce6341fdf07deaf935ecadee6c10ef8fe016e	Cracked
19	read5839	d3449ac106e128af598f8939f8f636127c50969d	Cracked
20	globe7751	2db1667bc355b80339637593d572b7c0d18e0b9	Cracked
21	africa610	64d40fb48d182fd084e9d6a236762e924f939910	Cracked

Fig 30 All the cracked passwords

Conclusion: In conclusion, the hash cracking project successfully demonstrated the efficacy of using advanced computational tools and algorithms to decipher encrypted password data. By employing a combination of Python scripting for generating password variants and the powerful capabilities of Hashcat for brute-force attacks, we were able to crack all 20 SHA-1 hashed passwords. The project underscored the importance of understanding common password creation patterns and effectively utilizing dictionary-based and numeric strategies. It also highlighted the limitations of SHA-1 as a secure hashing algorithm, emphasizing the need for stronger encryption methods in today's digital landscape. This exercise not only provided valuable insights into the realm of cybersecurity and password encryption but also showcased the potential of combining different technological approaches to overcome complex challenges in data security. The successful completion of this project serves as a testament to the continuous need for robust cybersecurity measures and the role of ethical hacking in fortifying digital defenses.