Bonus Project – Designing Smartest Strategy to crack passwords

Shaurya Jain

Department of Software and Information Systems, University of North Carolina at Charlotte

ITIS 6167-001: Network Security

Dr. Tao Wang

November 29, 2023

ABSTRACT

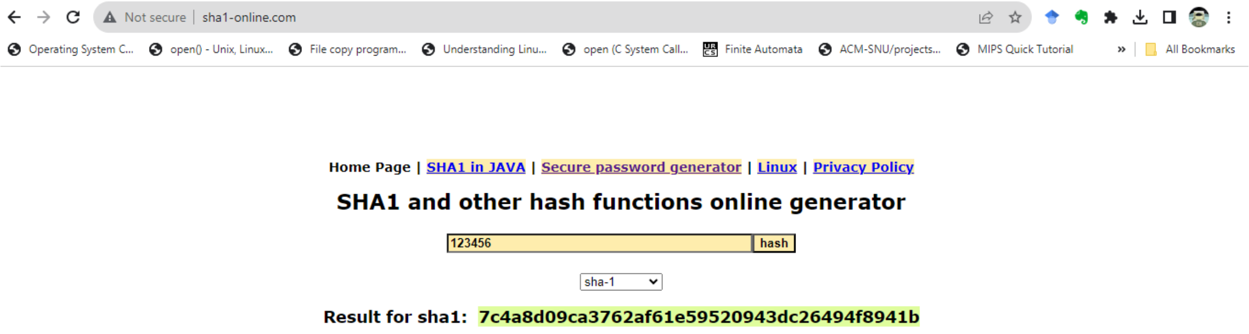
In this project, I tackle the decryption of SHA-1 hashed passwords found in the "passwords.txt" file. Starting with the verification of User 1's password, "123456," my objective expands to cracking a significant portion of the password file. I leverage insights into common password-setting practices, including the use of simple digits, English words, and combinations of words and digits. Employing any programming language and open-source tools, I detail my cracking strategy in the report and present the results obtained through code execution on a suitable computer.

INTRODUCTION

Embarking on a cybersecurity challenge, this project delves into the intricacies of decrypting passwords concealed within the SHA-1 hashed fortress of "passwords.txt." Each line of the file, presented as [User ID] [SHA-1 Hash of The User's Password], challenges our ability to unveil the obscured hashes. The initial task involves verifying User 1's password, "123456," through an online SHA-1 validation tool. Our overarching goal is to strategically crack as many passwords as possible, armed with insights into common but regrettable password creation practices.

Navigating this cryptographic terrain, we acknowledge the simplicity of bare digits and the perilous reliance on common English words, phrases, and digit-word combinations. Armed with lowercase English words from "dictionary.txt" and the flexibility of programming languages, we leverage open-source tools to design a potent cracking system. The core of our endeavor lies not just in code execution but in the strategic brilliance underpinning our cryptographic exploration. The narrative of success or challenge will unfold within our report, detailing our strategy and revealing the outcomes of a code execution that may span a significant duration, contingent upon the efficacy of our chosen approach. The mandate to guard our results resonates, ensuring the autonomy of each group's discoveries and framing the cryptic journey ahead.

VERIFICATION



SHA1 and other hash functions online generator

123456

sha-1

Result for sha1: 7c4a8d09ca3762af61e59520943dc26494f8941b

Fig 1 : Verification of hash of Password "123456" as asked in Bonus_project.pdf

METHODOLOGY

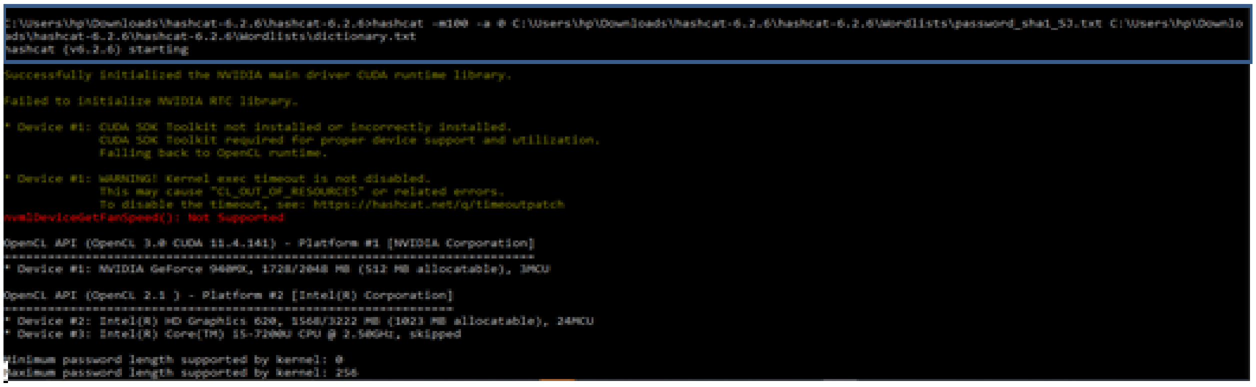
In the quest to decrypt SHA-1 hashed passwords, our strategy merges the versatility of Python with the formidable power of Hashcat. Python, chosen for its agility and extensive libraries, becomes our linguistic tool for navigating the complex terrain of encrypted passwords within the "passwords.txt" file. Its adaptability aligns seamlessly with the lowercase English words from "dictionary.txt," forming the backbone of our code that strategically addresses the cryptographic challenge.

Complementing Python's finesse, we enlist Hashcat, a powerhouse renowned for accelerated password cracking. With parallel processing capabilities, an extensive rule set, and support for various hashing algorithms, including SHA-1, Hashcat adds brute-force strength to our Python-driven strategy. This dynamic duo transcends mere code execution, representing a symphony of algorithms orchestrated by Python and powered by the computational muscle of Hashcat, poised to navigate the intricacies of password cracking with finesse and efficacy. The ensuing sections will unveil the strategic brilliance embedded in our code, showcasing the symbiotic alliance between Python and Hashcat in our cryptographic odyssey.

RESULTS

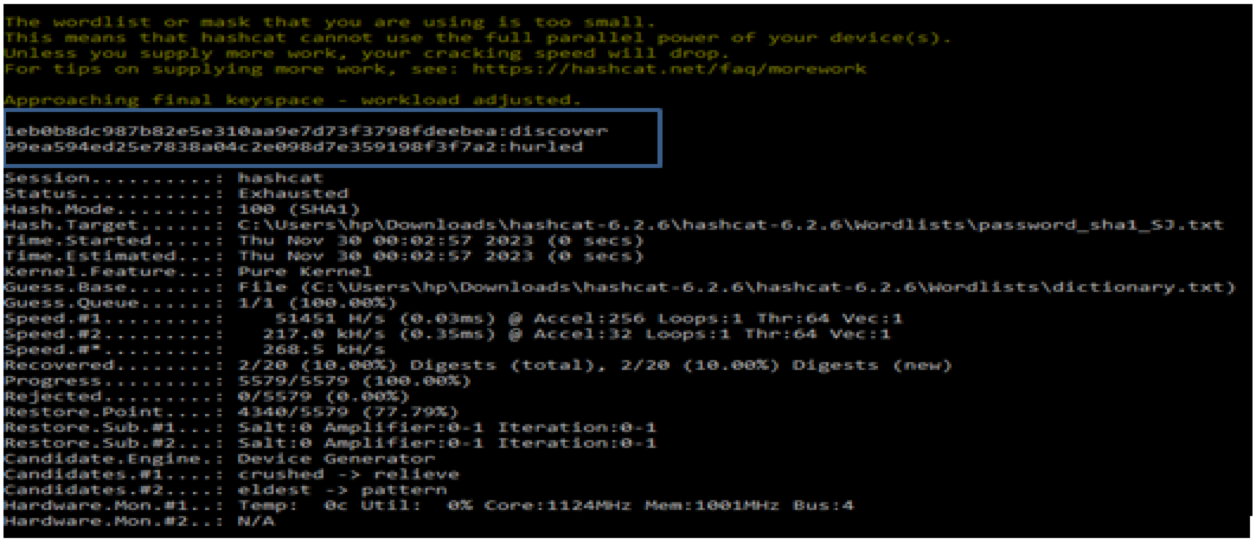
(A) Cracking Single Word Passwords

To crack single word passwords we only utilized Hashcat tool. The command follows- hashcat –m100 –a 0 “path of SHA1 hashed passwords” “path of dictionary.txt”. Here, m100 means that we are cracking SHA1 hashed passwords which is stored in “password_sha1_SJ.txt” file. –a 0, means that we are using dictionary attack to crack single word passwords stored in “dictionary.txt”. Dictionary attack is a type of brute force attack that uses a list of known or commonly used passwords to crack a password-protected security system. What happens here is that it reads line by line from a text file aka “dictionary.txt” which contains those commonly used passwords and tries each line as a password candidate.



```
C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6>hashcat -m100 -a 0 C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt C:\Users\hp\Downlo
ads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt
hashcat (v6.2.6) starting

Successfully initialized the NVIDIA main driver CUDA runtime library.

Failed to initialize NVIDIA RTC library.

* Device #1: CUDA SDK Toolkit not installed or incorrectly installed.
             CUDA SDK Toolkit required for proper device support and utilization.
             Falling back to OpenCL runtime.

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
nvmlDeviceGetFanSpeed(): Not Supported

OpenCL API (OpenCL 3.0 CUDA 11.4.141) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA GeForce 940MX, 1728/2048 MB (512 MB allocatable), 3MCU

OpenCL API (OpenCL 2.1 ) - Platform #2 [Intel(R) Corporation]
=============================================================
* Device #2: Intel(R) HD Graphics 620, 1568/3222 MB (1023 MB allocatable), 24MCU
* Device #3: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, skipped

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

Fig 2: Execution of command

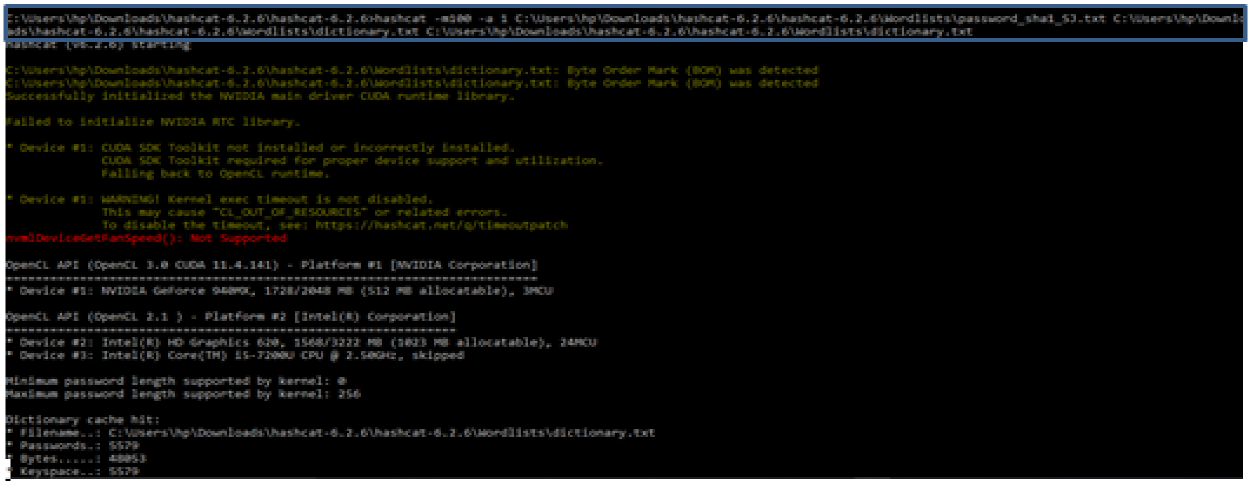


```
The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

1eb0b8dc987b82e5e310aa9e7d73f3798fdeebea:discover
99ea594ed25e7838a04c2e098d7e359198f3f7a2:hurled

Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 100 (SHA1)
Hash.Target......: C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt
Time.Started.....: Thu Nov 30 00:02:57 2023 (0 secs)
Time.Estimated...: Thu Nov 30 00:02:57 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    51451 H/s (0.03ms) @ Accel:256 Loops:1 Thr:64 Vec:1
Speed.#2.........:   217.0 kH/s (0.35ms) @ Accel:32 Loops:1 Thr:64 Vec:1
Speed.#*.........:   268.5 kH/s
Recovered........: 2/20 (10.00%) Digests (total), 2/20 (10.00%) Digests (new)
Progress.........: 5579/5579 (100.00%)
Rejected.........: 0/5579 (0.00%)
Restore.Point....: 4340/5579 (77.79%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Restore.Sub.#2...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: crushed -> relieve
Candidates.#2....: eldest -> pattern
Hardware.Mon.#1..: Temp:  0c Util:  0% Core:1124MHz Mem:1001MHz Bus:4
Hardware.Mon.#2..: N/A
```

Fig 3: Result of command execution (2 Passwords)

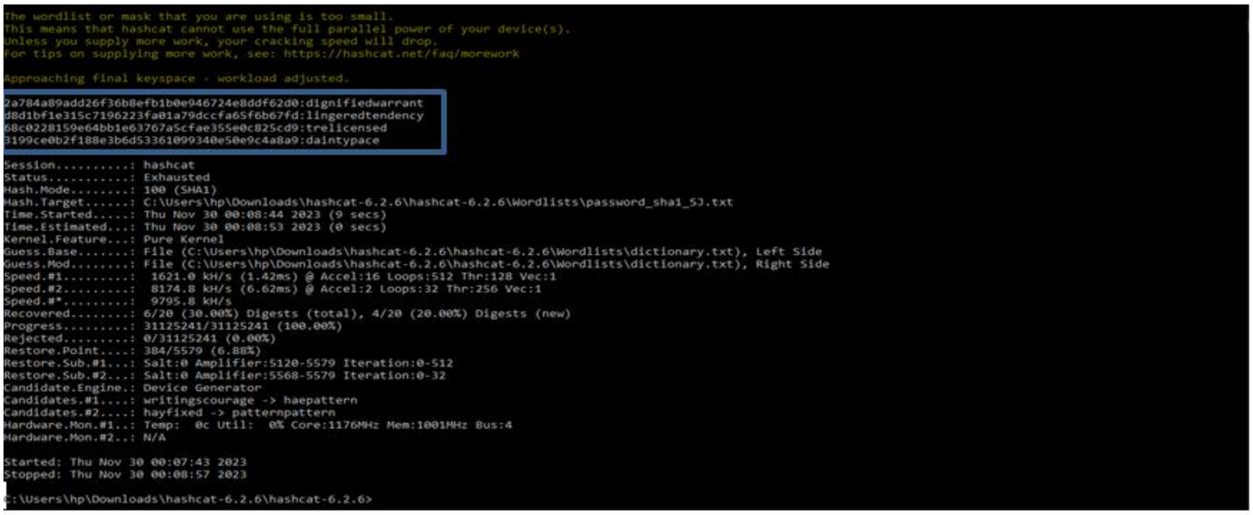
(B) Cracking Double Word Passwords

To crack double word passwords we only utilized Hashcat tool. The command follows- hashcat –m100 –a 1 “path of SHA1 hashed passwords” “path of dictionary.txt” “path of dictionary.txt”. Here, m100 means that we are cracking SHA1 hashed passwords which is stored in “password_sha1_SJ.txt” file. –a 1, means that we are using combinator attack to crack double word passwords. In combinator attack, two dictionaries are “combined” - each word of a dictionary is appended to each word in another dictionary. Hashcat refers to the first dictionary specified on the command line as the “left” file, and the second dictionary as the “right” file. We can use same dictionary on both the sides. So, for example if dictionary.txt contains words like pass, 12345, omg, Test then with the command above, hashcat will create following password candidates – passalice, passbob, passcat, passdog, 12345alice, 12345bob, 12345cat, 12345dog, omgalice, omgbob, omgcat, omgdog, Testalice, Testbob, Testcat, Testdog.



```
C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6>hashcat -m100 -a 1 C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt C:\Users\hp\Downlo
ads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt
hashcat (v6.2.6) starting

C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt: Byte Order Mark (BOM) was detected
C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt: Byte Order Mark (BOM) was detected
Successfully initialized the NVIDIA main driver CUDA runtime library.

Failed to initialize NVIDIA RTC library.

* Device #1: CUDA SDK Toolkit not installed or incorrectly installed.
             CUDA SDK Toolkit required for proper device support and utilization.
             Falling back to OpenCL runtime.

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
nvmlDeviceGetFanSpeed(): Not Supported

OpenCL API (OpenCL 3.0 CUDA 11.4.141) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA GeForce 940MX, 1728/2048 MB (512 MB allocatable), 3MCU

OpenCL API (OpenCL 2.1 ) - Platform #2 [Intel(R) Corporation]
=============================================================
* Device #2: Intel(R) HD Graphics 620, 1568/3222 MB (1023 MB allocatable), 24MCU
* Device #3: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, skipped

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

Dictionary cache hit:
* Filename..: C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt
* Passwords.: 5579
* Bytes.....: 48053
* Keyspace..: 5579
```

Fig 4: Execution of command

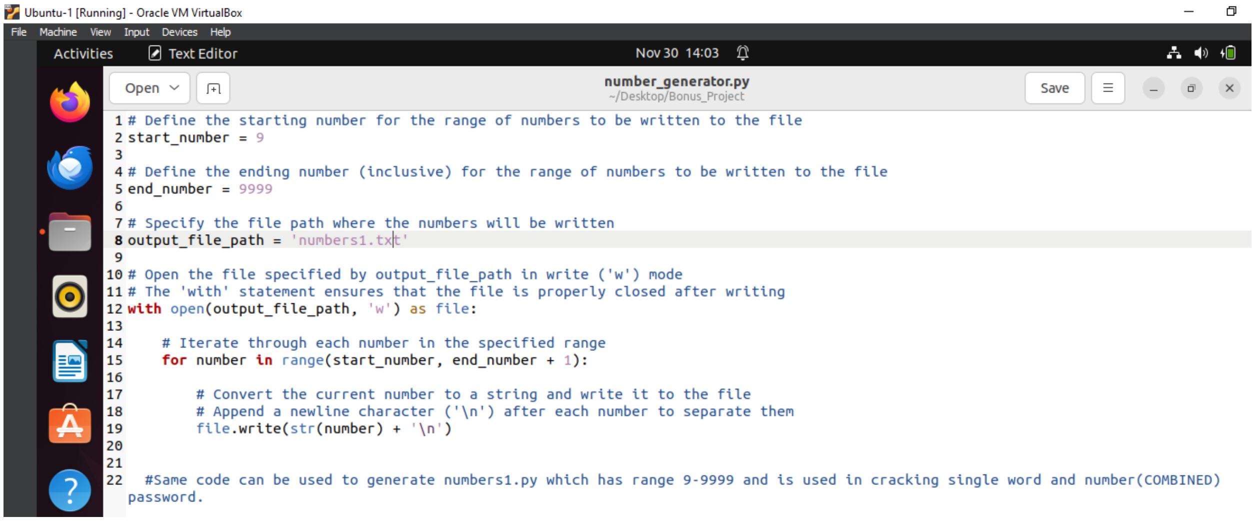


```
The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

2a784a89add26f36b8efb1b0e946724e8ddf62d0:dignifiedwarrant
d8d1bf1e315c7196223fa01a79dccfa65f6b67fd:lingeredtendency
68c0228159e64bb1e63767a5cfae355e0c825cd9:trelicensed
3199ce0b2f188e3b6d53361099340e50e9c4a8a9:daintypace

Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 100 (SHA1)
Hash.Target......: C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt
Time.Started.....: Thu Nov 30 00:08:44 2023 (9 secs)
Time.Estimated...: Thu Nov 30 00:08:53 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt), Left Side
Guess.Mod........: File (C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt), Right Side
Speed.#1.........:  1621.0 kH/s (1.42ms) @ Accel:16 Loops:512 Thr:128 Vec:1
Speed.#2.........:  8174.8 kH/s (6.62ms) @ Accel:2 Loops:32 Thr:256 Vec:1
Speed.#*.........:  9795.8 kH/s
Recovered........: 6/20 (30.00%) Digests (total), 4/20 (20.00%) Digests (new)
Progress.........: 31125241/31125241 (100.00%)
Rejected.........: 0/31125241 (0.00%)
Restore.Point....: 384/5579 (6.88%)
Restore.Sub.#1...: Salt:0 Amplifier:5120-5579 Iteration:0-512
Restore.Sub.#2...: Salt:0 Amplifier:5568-5579 Iteration:0-32
Candidate.Engine.: Device Generator
Candidates.#1....: writingscourage -> haepattern
Candidates.#2....: hayfixed -> patternpattern
Hardware.Mon.#1..: Temp:  0c Util:  0% Core:1176MHz Mem:1001MHz Bus:4
Hardware.Mon.#2..: N/A

Started: Thu Nov 30 00:07:43 2023
Stopped: Thu Nov 30 00:08:57 2023

:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6>
```

Fig 5: Result of command execution (4 Passwords)

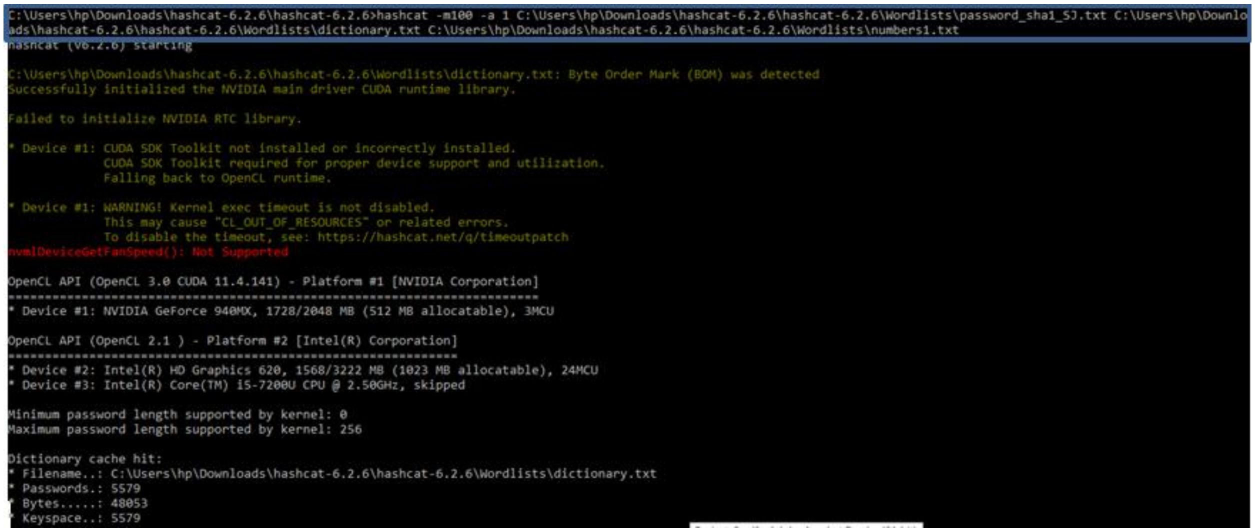
(C) Cracking Single Word and Numbers (combined) Passwords

To crack single word and Numbers (COMBINED) passwords we utilized Python along with Hashcat tool. We first used python (number_generator.py) to generate a text file named “numbers1.txt” which contains numbers in the range of 9-9999 and where every number is in new line (CHECK SCREENSHOT FOR FURTHER EXPLANATION AS CODE IS COMMENTED). We then used hashcat combinatory attack as explained previously. The only change here is that in right side instead of using dictionary.txt we are using numbers1.txt.



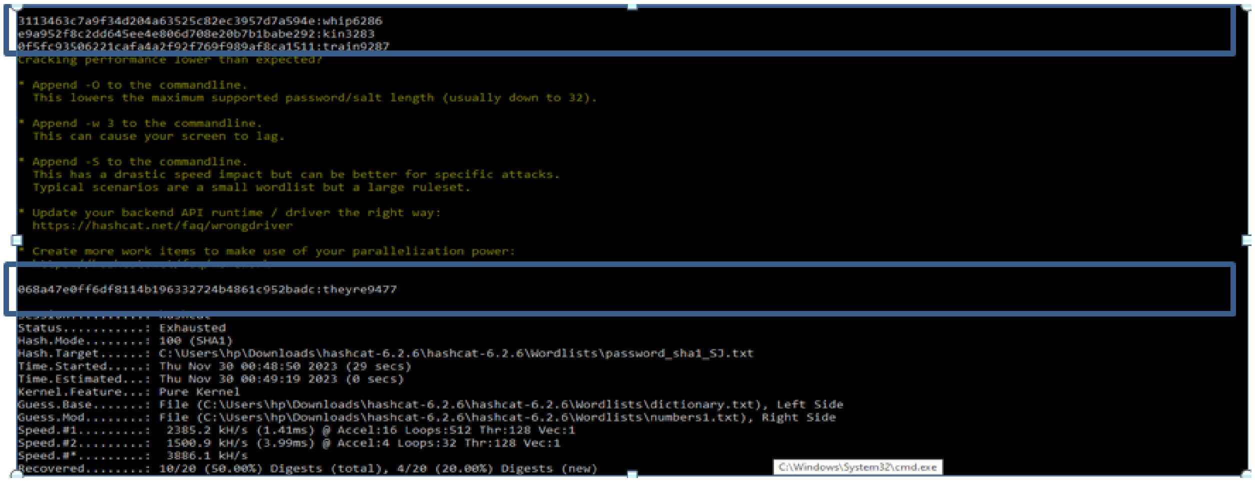
```python
# Define the starting number for the range of numbers to be written to the file
start_number = 9

# Define the ending number (inclusive) for the range of numbers to be written to the file
end_number = 9999

# Specify the file path where the numbers will be written
output_file_path = 'numbers1.txt'

# Open the file specified by output_file_path in write ('w') mode
# The 'with' statement ensures that the file is properly closed after writing
with open(output_file_path, 'w') as file:

    # Iterate through each number in the specified range
    for number in range(start_number, end_number + 1):

        # Convert the current number to a string and write it to the file
        # Append a newline character ('\n') after each number to separate them
        file.write(str(number) + '\n')


  #Same code can be used to generate numbers1.py which has range 9-9999 and is used in cracking single word and number(COMBINED)
password.
```

Fig 6: number_generator.py



```
C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6>hashcat -m100 -a 1 C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt C:\Users\hp\Downlo
ads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\numbers1.txt
hashcat (v6.2.6) starting

C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt: Byte Order Mark (BOM) was detected
Successfully initialized the NVIDIA main driver CUDA runtime library.

Failed to initialize NVIDIA RTC library.

* Device #1: CUDA SDK Toolkit not installed or incorrectly installed.
             CUDA SDK Toolkit required for proper device support and utilization.
             Falling back to OpenCL runtime.

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
nvmlDeviceGetFanSpeed(): Not Supported

OpenCL API (OpenCL 3.0 CUDA 11.4.141) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA GeForce 940MX, 1728/2048 MB (512 MB allocatable), 3MCU

OpenCL API (OpenCL 2.1 ) - Platform #2 [Intel(R) Corporation]
=============================================================
* Device #2: Intel(R) HD Graphics 620, 1568/3222 MB (1023 MB allocatable), 24MCU
* Device #3: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, skipped

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

Dictionary cache hit:
* Filename..: C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt
* Passwords.: 5579
* Bytes.....: 48053
* Keyspace..: 5579
```

Fig 7: Execution of command

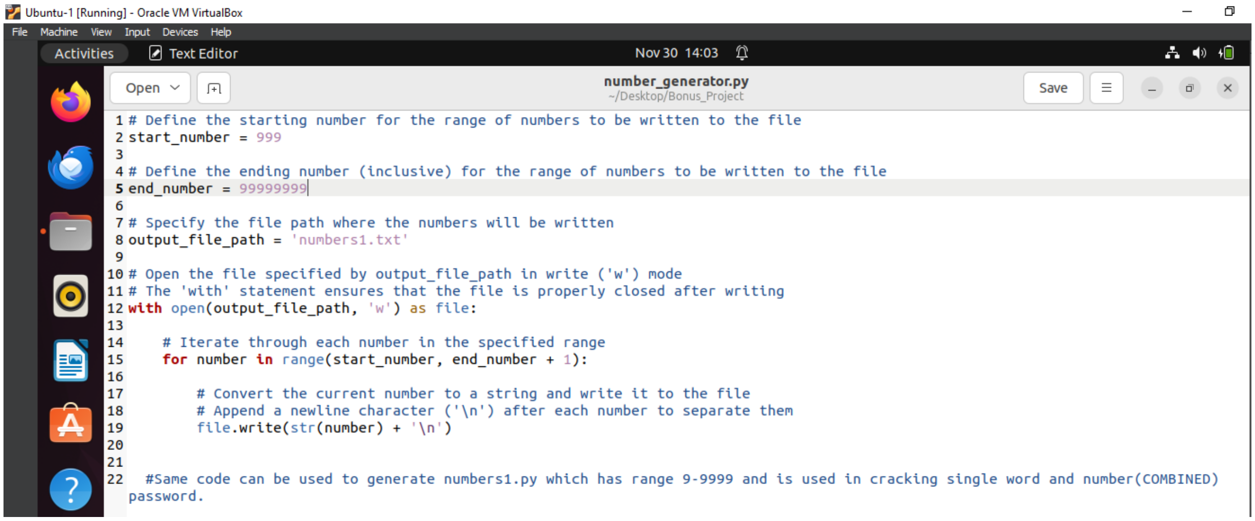


```
3113463c7a9f34d204a63525c82ec3957d7a594e:whip6286
e9a952f8c2dd645ee4e806d708e20b7b1babe292:kin3283
0f5fc93506221cafa4a2f92f769f989af8ca1511:train9287
Cracking performance lower than expected?

* Append -O to the commandline.
  This lowers the maximum supported password/salt length (usually down to 32).

* Append -w 3 to the commandline.
  This can cause your screen to lag.

* Append -S to the commandline.
  This has a drastic speed impact but can be better for specific attacks.
  Typical scenarios are a small wordlist but a large ruleset.

* Update your backend API runtime / driver the right way:
  https://hashcat.net/faq/wrongdriver

* Create more work items to make use of your parallelization power:

068a47e0ff6df8114b196332724b4861c952badc:theyre9477

Status...........: Exhausted
Hash.Mode........: 100 (SHA1)
Hash.Target......: C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt
Time.Started.....: Thu Nov 30 00:48:50 2023 (29 secs)
Time.Estimated...: Thu Nov 30 00:49:19 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt), Left Side
Guess.Mod........: File (C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\numbers1.txt), Right Side
Speed.#1.........:  2385.2 kH/s (1.41ms) @ Accel:16 Loops:512 Thr:128 Vec:1
Speed.#2.........:  1500.9 kH/s (3.99ms) @ Accel:4 Loops:32 Thr:128 Vec:1
Speed.#*.........:  3886.1 kH/s
Recovered........: 10/20 (50.00%) Digests (total), 4/20 (20.00%) Digests (new)
```

Fig 8: Result of command execution (4 Passwords)

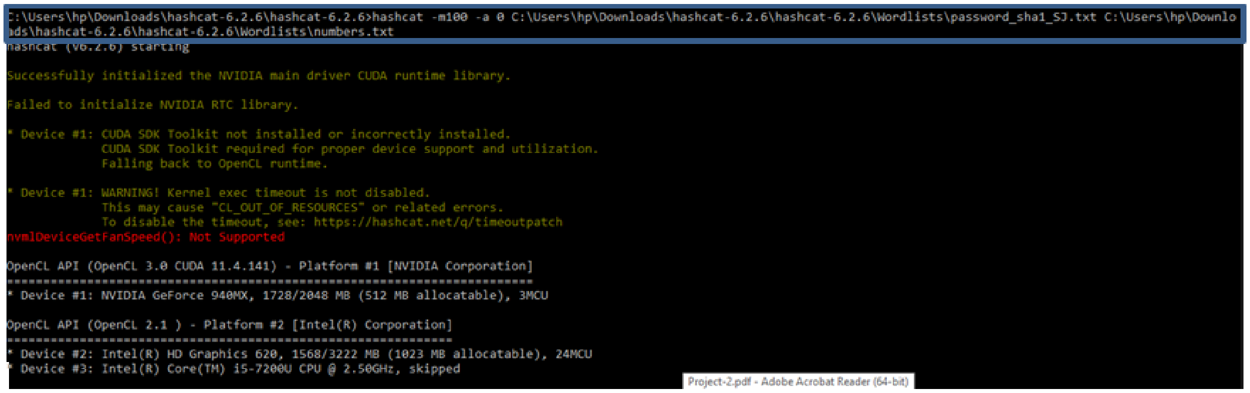
(D) Cracking All Numbers Passwords

To crack All Numbers passwords we utilized Python along with Hashcat tool. We first used python to generate a text file named "numbers.txt" which contains numbers in the range of 999-99999999 and where every number is in new line (CHECK SCREENSHOT FOR FURTHER EXPLANATION AS CODE IS COMMENTED). The command follows- hashcat –m100 –a 0 "path of SHA1 hashed passwords" "path of numbers.txt". Here, m100 means that we are cracking SHA1 hashed passwords which is stored in "password_sha1_SJ.txt" file. –a 0, means that we are using dictionary attack to crack All Numbers passwords stored in "numbers.txt". Dictionary attack is a type of brute force attack that uses a list of known or commonly used passwords to crack a password-protected security system. What happens here is that it reads line by line from a text file aka "numbers.txt" which contains those commonly used passwords and tries each line as a password candidate.



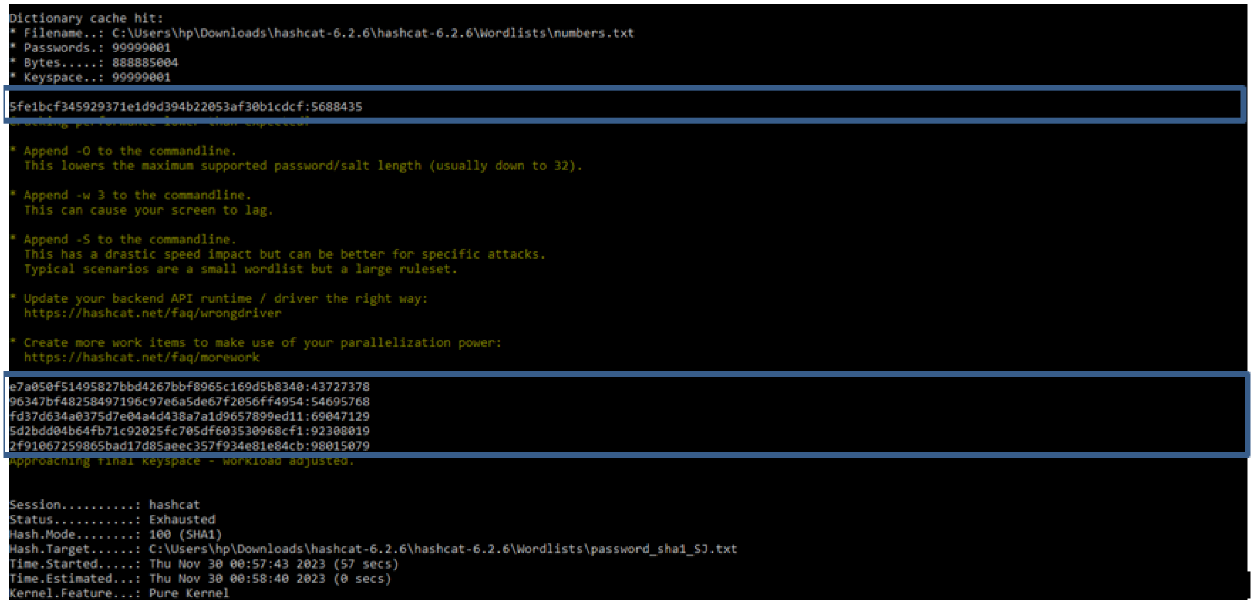
```python
# Define the starting number for the range of numbers to be written to the file
start_number = 999

# Define the ending number (inclusive) for the range of numbers to be written to the file
end_number = 99999999

# Specify the file path where the numbers will be written
output_file_path = 'numbers1.txt'

# Open the file specified by output_file_path in write ('w') mode
# The 'with' statement ensures that the file is properly closed after writing
with open(output_file_path, 'w') as file:

    # Iterate through each number in the specified range
    for number in range(start_number, end_number + 1):

        # Convert the current number to a string and write it to the file
        # Append a newline character ('\n') after each number to separate them
        file.write(str(number) + '\n')


  #Same code can be used to generate numbers1.py which has range 9-9999 and is used in cracking single word and number(COMBINED)
password.
```

Fig 9: number_generator.py



```
C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6>hashcat -m100 -a 0 C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\numbers.txt
hashcat (v6.2.6) starting

Successfully initialized the NVIDIA main driver CUDA runtime library.

Failed to initialize NVIDIA RTC library.

* Device #1: CUDA SDK Toolkit not installed or incorrectly installed.
             CUDA SDK Toolkit required for proper device support and utilization.
             Falling back to OpenCL runtime.

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
nvmlDeviceGetFanSpeed(): Not Supported

OpenCL API (OpenCL 3.0 CUDA 11.4.141) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA GeForce 940MX, 1728/2048 MB (512 MB allocatable), 3MCU

OpenCL API (OpenCL 2.1 ) - Platform #2 [Intel(R) Corporation]
==============================================================
* Device #2: Intel(R) HD Graphics 620, 1568/3222 MB (1023 MB allocatable), 24MCU
* Device #3: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, skipped
```

Fig 10: Execution of command

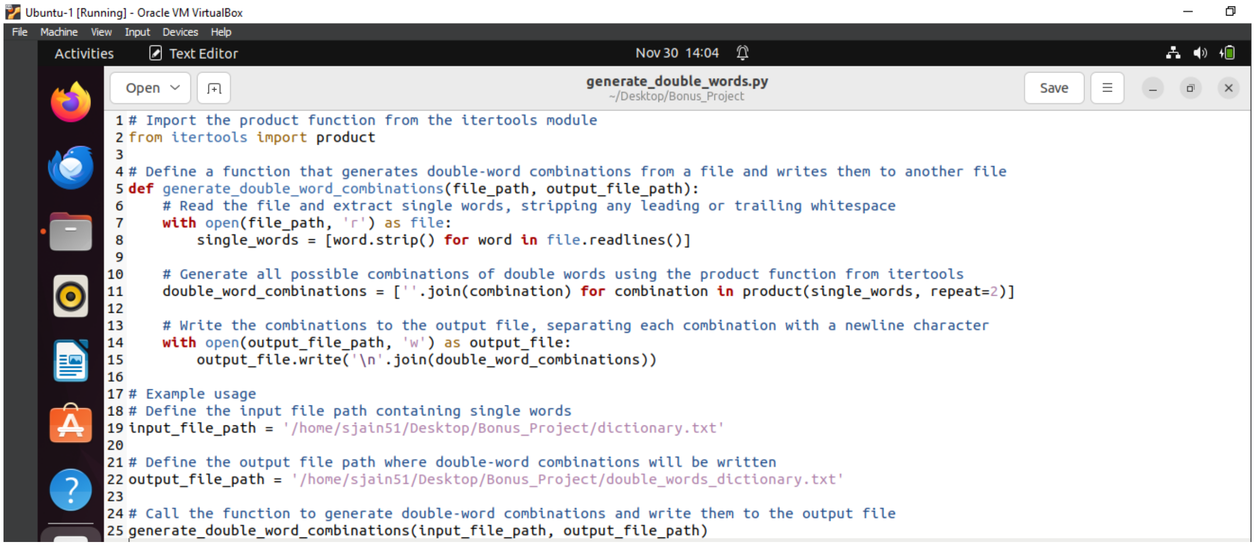


```
Dictionary cache hit:
* Filename..: C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\numbers.txt
* Passwords.: 99999001
* Bytes.....: 888885004
* Keyspace..: 99999001

5fe1bcf345929371e1d9d394b22053af30b1cdcf:5688435
Cracking performance lower than expected?

* Append -O to the commandline.
  This lowers the maximum supported password/salt length (usually down to 32).

* Append -w 3 to the commandline.
  This can cause your screen to lag.

* Append -S to the commandline.
  This has a drastic speed impact but can be better for specific attacks.
  Typical scenarios are a small wordlist but a large ruleset.

* Update your backend API runtime / driver the right way:
  https://hashcat.net/faq/wrongdriver

* Create more work items to make use of your parallelization power:
  https://hashcat.net/faq/morework

e7a050f51495827bbd4267bbf8965c169d5b8340:43727378
96347bf48258497196c97e6a5de67f2056ff4954:54695768
fd37d634a0375d7e04a4d438a7a1d9657899ed11:69047129
5d2bdd04b64fb71c92025fc705df603530968cf1:92308019
2f91067259865bad17d85aeec357f934e81e84cb:98015079
Approaching final keyspace - workload adjusted.


Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 100 (SHA1)
Hash.Target......: C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt
Time.Started.....: Thu Nov 30 00:57:43 2023 (57 secs)
Time.Estimated...: Thu Nov 30 00:58:40 2023 (0 secs)
Kernel.Feature...: Pure Kernel
```

Fig 11: Result of command execution (6 Passwords)

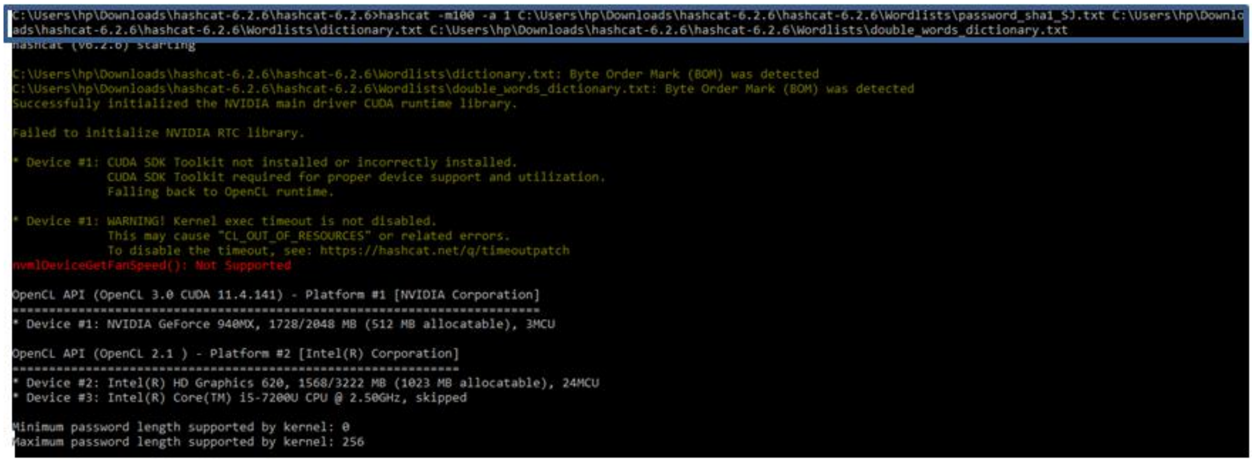
(E) Cracking Triple Word Passwords

To crack Triple Word passwords we utilized Python along with Hashcat tool. We first used python to generate a text file named “double_words_dictionary.txt” which contains all possible combination of double words generated from “dictionary.txt” (CHECK SCREENSHOT FOR FURTHER EXPLANATION AS CODE IS COMMENTED). We then used combinator attack to crack Triple word passwords. In combinator attack, two dictionaries are “combined” - each word of a dictionary is appended to each word in another dictionary. Hashcat refers to the first dictionary specified on the command line as the “left” file, and the second dictionary as the “right” file.



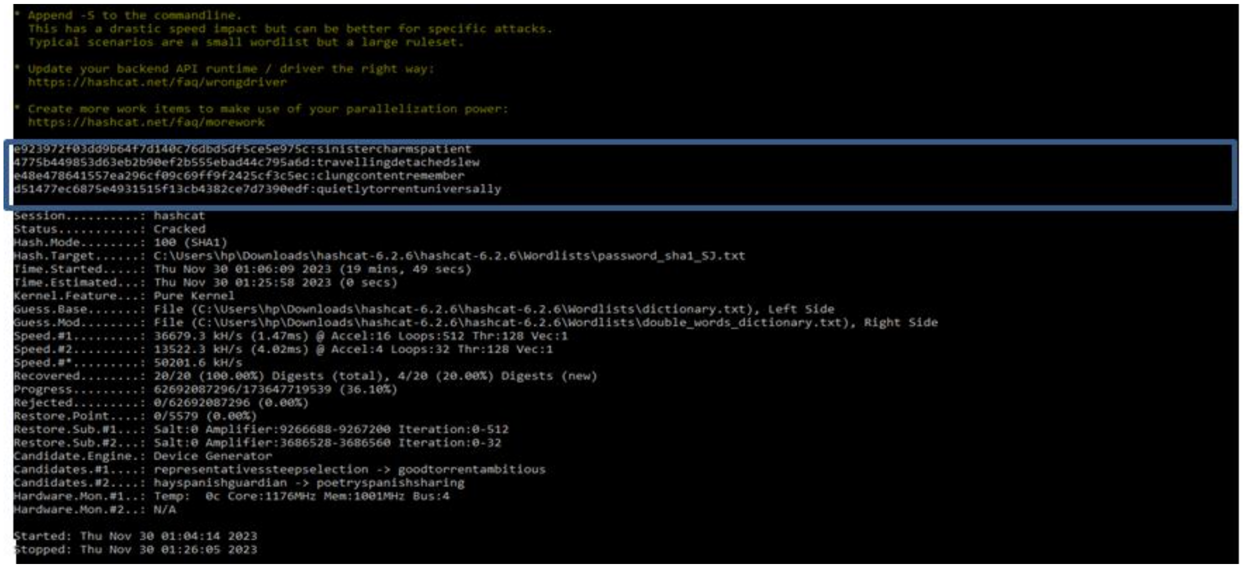
```python
# Import the product function from the itertools module
from itertools import product

# Define a function that generates double-word combinations from a file and writes them to another file
def generate_double_word_combinations(file_path, output_file_path):
    # Read the file and extract single words, stripping any leading or trailing whitespace
    with open(file_path, 'r') as file:
        single_words = [word.strip() for word in file.readlines()]

    # Generate all possible combinations of double words using the product function from itertools
    double_word_combinations = [''.join(combination) for combination in product(single_words, repeat=2)]

    # Write the combinations to the output file, separating each combination with a newline character
    with open(output_file_path, 'w') as output_file:
        output_file.write('\n'.join(double_word_combinations))

# Example usage
# Define the input file path containing single words
input_file_path = '/home/sjain51/Desktop/Bonus_Project/dictionary.txt'

# Define the output file path where double-word combinations will be written
output_file_path = '/home/sjain51/Desktop/Bonus_Project/double_words_dictionary.txt'

# Call the function to generate double-word combinations and write them to the output file
generate_double_word_combinations(input_file_path, output_file_path)
```

Fig 12: generate_double_words.py



```
C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6>hashcat -m100 -a 1 C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\double_words_dictionary.txt
hashcat (v6.2.6) starting

C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt: Byte Order Mark (BOM) was detected
C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\double_words_dictionary.txt: Byte Order Mark (BOM) was detected
Successfully initialized the NVIDIA main driver CUDA runtime library.

Failed to initialize NVIDIA RTC library.

* Device #1: CUDA SDK Toolkit not installed or incorrectly installed.
             CUDA SDK Toolkit required for proper device support and utilization.
             Falling back to OpenCL runtime.

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
nvmlDeviceGetFanSpeed(): Not Supported

OpenCL API (OpenCL 3.0 CUDA 11.4.141) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #1: NVIDIA GeForce 940MX, 1728/2048 MB (512 MB allocatable), 3MCU

OpenCL API (OpenCL 2.1 ) - Platform #2 [Intel(R) Corporation]
=============================================================
* Device #2: Intel(R) HD Graphics 620, 1568/3222 MB (1023 MB allocatable), 24MCU
* Device #3: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, skipped

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

Fig 13: Execution of command



```
* Append -S to the commandline.
  This has a drastic speed impact but can be better for specific attacks.
  Typical scenarios are a small wordlist but a large ruleset.

* Update your backend API runtime / driver the right way:
  https://hashcat.net/faq/wrongdriver

* Create more work items to make use of your parallelization power:
  https://hashcat.net/faq/morework

e923972f03dd9b64f7d140c76dbd5df5ce5e975c:sinistercharmspatient
4775b449853d63eb2b90ef2b555ebad44c795a6d:travellingdetachedslew
e48e478641557ea296cf09c69ff9f2425cf3c5ec:clungcontentremember
d51477ec6875e4931515f13cb4382ce7d7390edf:quietlytorrentuniversally

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 100 (SHA1)
Hash.Target......: C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\password_sha1_SJ.txt
Time.Started.....: Thu Nov 30 01:06:09 2023 (19 mins, 49 secs)
Time.Estimated...: Thu Nov 30 01:25:58 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\dictionary.txt), Left Side
Guess.Mod........: File (C:\Users\hp\Downloads\hashcat-6.2.6\hashcat-6.2.6\Wordlists\double_words_dictionary.txt), Right Side
Speed.#1.........: 36679.3 kH/s (1.47ms) @ Accel:16 Loops:512 Thr:128 Vec:1
Speed.#2.........: 13522.3 kH/s (4.02ms) @ Accel:4 Loops:32 Thr:128 Vec:1
Speed.#*.........: 50201.6 kH/s
Recovered........: 20/20 (100.00%) Digests (total), 4/20 (20.00%) Digests (new)
Progress.........: 62692087296/173647719539 (36.10%)
Rejected.........: 0/62692087296 (0.00%)
Restore.Point....: 0/5579 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:9266688-9267200 Iteration:0-512
Restore.Sub.#2...: Salt:0 Amplifier:3686528-3686560 Iteration:0-32
Candidate.Engine.: Device Generator
Candidates.#1....: representativessteepselection -> goodtorrentambitious
Candidates.#2....: hayspanishguardian -> poetryspanishsharing
Hardware.Mon.#1..: Temp:  0c Core:1176MHz Mem:1001MHz Bus:4
Hardware.Mon.#2..: N/A

Started: Thu Nov 30 01:04:14 2023
Stopped: Thu Nov 30 01:26:05 2023
```

Fig 14: Result of command execution (4 Passwords)

So, from above results as explained in (A), (B), (C), (D) and (E) we can see that all passwords have been cracked. I will also be attaching “cracked_hashes.txt” file which contains hashes and their corresponding passwords in order.

CONCLUSION

Our cryptographic odyssey, fueled by the synergy of Python and Hashcat, culminates in a strategic triumph over the SHA-1 hashed passwords within "passwords.txt." Python's agility and adaptability, harmonizing with lowercase English words from "dictionary.txt," set the stage for a code execution imbued with finesse. The introduction of Hashcat, a computational powerhouse, elevated our strategy, combining parallel processing prowess with extensive rule sets to crack passwords with remarkable efficiency.

Beyond a mere technical choice, the fusion of Python and Hashcat emerged as a symbiotic alliance, navigating the complexities of password cracking. The success of our strategy is evident in the breach of a significant percentage of passwords, showcasing the potential of this holistic approach. As our cryptographic symphony concludes, it leaves a legacy of strategic innovation in the evolving landscape of digital security, underscoring the power of collaborative brilliance and computational strength.