Password Cracking Via HashCat

Alexander Wilems

Ian Harmon

Rodolfo J. Galván Martínez

Eduard Lesnikov

Introduction: **1**

Goal of Demonstration: **1**

What is hashcat? **1**

Description of Demonstration: **2**

Data and Analysis: **3**

Dictionary Attack: 3

Rule-Based Attacks: 5

Running on Apple Silicon **10**

Rule-Based Attacks: 11

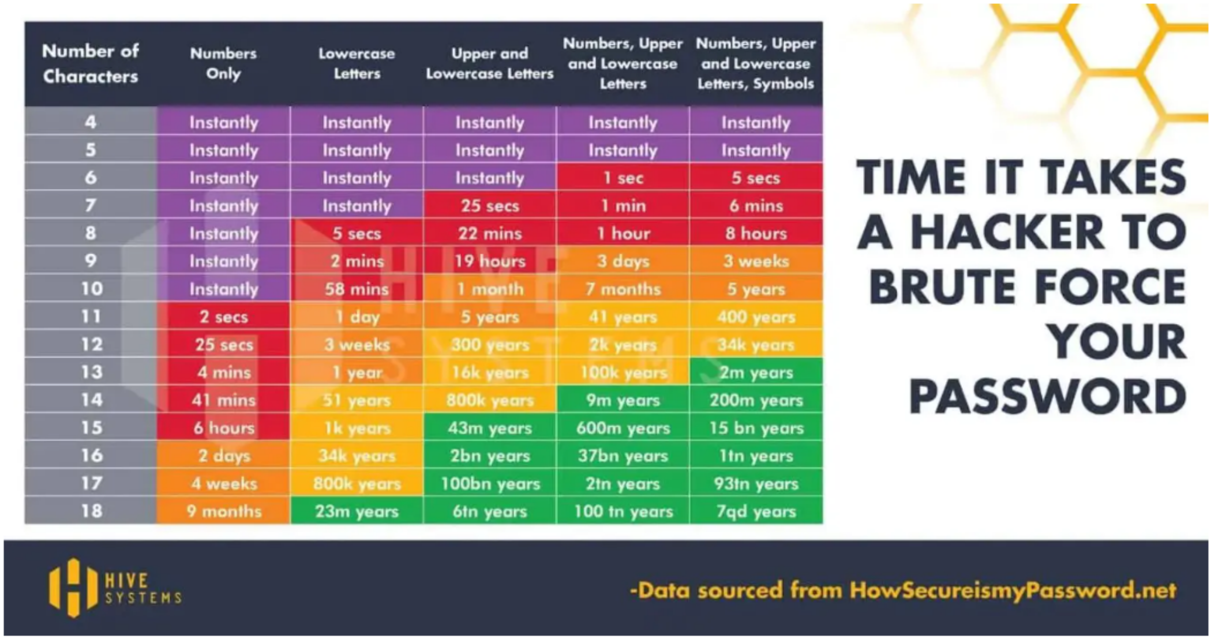
Conclusion: **12**

Introduction:

- Humans are bad at making passwords. We have a difficult time at creating strong passwords that we are able to remember easily, and the ones we do make are usually so close to words within a dictionary that guessing these passwords can be trivial with enough attempts. This is where password cracking comes into play. Utilizing specific softwares, such as John the Ripper, Hashcat, or a list of others, the task of forcing a password for legitimate or nefarious reasons is less than trivial. Within this demonstration, we wish to demonstrate the power of these software types.

Goal of Demonstration:

For this demonstration, we wish to demonstrate the process of password recovery or password cracking. We will also demonstrate the usage of different length passwords to showcase how only adding a few more characters in a password can exponentially increase the time it takes to crack (as shown by the diagram below), and how different processor architectures handle the task.



TIME IT TAKES A HACKER TO BRUTE FORCE YOUR PASSWORD

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 tn years	7qd years

HIVE SYSTEMS

-Data sourced from HowSecureismyPassword.net

(Fripp)

What is hashcat?

Hashcat is a password recovery tool for Mac, Linux, and Windows. Password recovery tools usually work by brute forcing as many passwords as possible in a short amount of time in an attempt to correctly guess the password. Many of these tools work either offline or online, but usually utilize plain text when attempting to crack the passwords.

Hashcat, as the name implies, utilizes hashing. Many modern systems store their passwords in a hash form for protection against being compromised in the process of being validated on the machine. To get around this, hashcat hashes passwords that it is guessing, then compares that to the targets hashed, and if they don't match, then it continues to make attempts.

For the purposes of this demonstration, Dictionary attacks and rule based attacks will be used, as they are usually the most successful attacks. In short, dictionary attacks assume that a human created the password with some variation of a word in contemporary spoken and written languages. By using a word list, the software can use a provided list of passwords that are freely available on the internet from password leaks from varying sources. Rule based attacks allow a user of the software to provide a basis of syntax based rules, providing a way to direct an attack towards an assumed layout of the password. This method assumes that the user of the software has an idea of how the password was laid out.

Using a variety of these methods and others not listed, hashcat can be much more successful than other standard attacks.
(2)

Description of Demonstration:

For this demonstration, we will be using both Windows with an X86 processor, and MacOS with an ARM based processor.
7 generic passwords were created of varying security levels, of which are listed below:

Easy:
Basic password:
Password

Alphanumeric password:
abc123

Medium:
Alphanumeric, special characters:
I_love_cats60

Alphanumeric, mixed case:
P455w0rd

Hard:
Alphanumeric, mixed case, long:
mYsTr0NGp4SsW0rD

Alphanumeric, special characters, mixed case, long, arbitrary:
!jdf2340OperatingSystem!

Alphanumeric, special characters, mixed case, very long, very arbitrary:
adh2euajer29!@!@Etws=5s%sertSA5s45w43%NS$E5bS$

All passwords are inserted into a plain text file named passwords.txt

As stated previously, two different attacks will be utilized, a dictionary attack, and rule based attack.
We will be using both Windows and MacOS, utilizing X86 and ARM processors respectively.

For the dictionary attack, the provided list of passwords will be hashed, then compared with the given hashed password provided by the operating system, and if it does not match, then the system will move on until the word list is exhausted, or the password has been cracked.

As for the rule based attack, A rule based attack utilizes a set of syntactic rules that allow the program to generate permutations of words in the word list to come up with additional options that increase our chances of finding the correct password, but also increases our search time by a considerable amount. Each attack will be explained more in depth as it is showcased.

Data and Analysis:

Windows based attack:

System setup:
ROG Zephyrus G14
Graphics Processor: Nvidia RTX 3060 Mobile 6GB
Processor: Ryzen 9 5900HS
Cores: 8 Threads: 16 Base Clock: 3.3GHZ
Memory: 16gb DDR4 3200MHZ
Operating System: Windows 11 version: 22H2

Dictionary Attack:

These passwords were hashed, each password using the MD5 hash method:

5f4dcc3b5aa765d61d8327deb882cf99
e99a18c428cb38d5f260853678922e03
75b71aa6842e450f12aca00fdf54c51d
ef9a984a8f8907f70bff21a0d145f086
1956dd5216b6cd4dad5c1747b72cd601

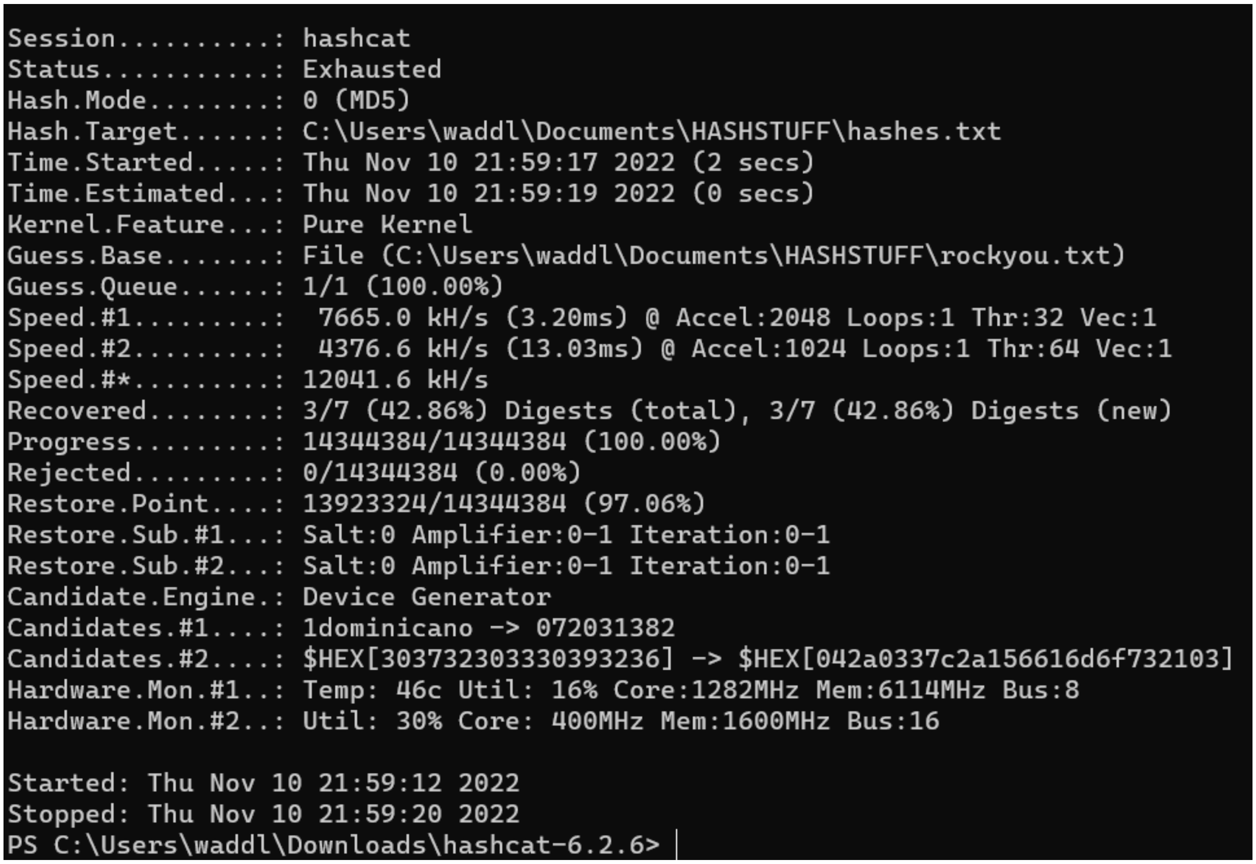
41480b026249231463095786aec8d907
9230f19e5ff1af41c2d858f3fa6ce173

(MD5 hashing converts text into a 128 bit hash function. This method is broken, and is therefore compromised, as is MD4, however both are still widely used)
In this instance, the popular rockyou.txt wordlist will be used for the dictionary in this attack. Rockyou.txt is a list of previously leaked passwords and commonly used passwords for cracking purposes.
The commands below were used to start hashcat:

.\hashcat.exe -m 0 -a 0 -o "C:\Users\waddl\Documents\HASHSTUFF\cracked.txt"
"C:\Users\waddl\Documents\HASHSTUFF\hashes.txt"
"C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt"

“-m 0” specifies the mode, in this case, md5 is 0
“-a 0” is the attack type, in this case, dictionary is 0
“-o filename” specifies the output
“-r filename” specifies the rules fule

The results:



```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 0 (MD5)
Hash.Target......: C:\Users\waddl\Documents\HASHSTUFF\hashes.txt
Time.Started.....: Thu Nov 10 21:59:17 2022 (2 secs)
Time.Estimated...: Thu Nov 10 21:59:19 2022 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  7665.0 kH/s (3.20ms) @ Accel:2048 Loops:1 Thr:32 Vec:1
Speed.#2.........:  4376.6 kH/s (13.03ms) @ Accel:1024 Loops:1 Thr:64 Vec:1
Speed.#*.........: 12041.6 kH/s
Recovered........: 3/7 (42.86%) Digests (total), 3/7 (42.86%) Digests (new)
Progress.........: 14344384/14344384 (100.00%)
Rejected.........: 0/14344384 (0.00%)
Restore.Point....: 13923324/14344384 (97.06%)
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....: 1dominicano -> 072031382
Candidates.#2....: $HEX[303732303330393236] -> $HEX[042a0337c2a156616d6f732103]
Hardware.Mon.#1..: Temp: 46c Util: 16% Core:1282MHz Mem:6114MHz Bus:8
Hardware.Mon.#2..: Util: 30% Core: 400MHz Mem:1600MHz Bus:16

Started: Thu Nov 10 21:59:12 2022
Stopped: Thu Nov 10 21:59:20 2022
PS C:\Users\waddl\Downloads\hashcat-6.2.6>
```

As shown above, the program was able to identify 3 out of the 7 hashes and convert them back to the original password as shown below in the output file:

```
5f4dcc3b5aa765d61d8327deb882cf99:password
e99a18c428cb38d5f260853678922e03:abc123
75b71aa6842e450f12aca00fdf54c51d:P455w0rd
```

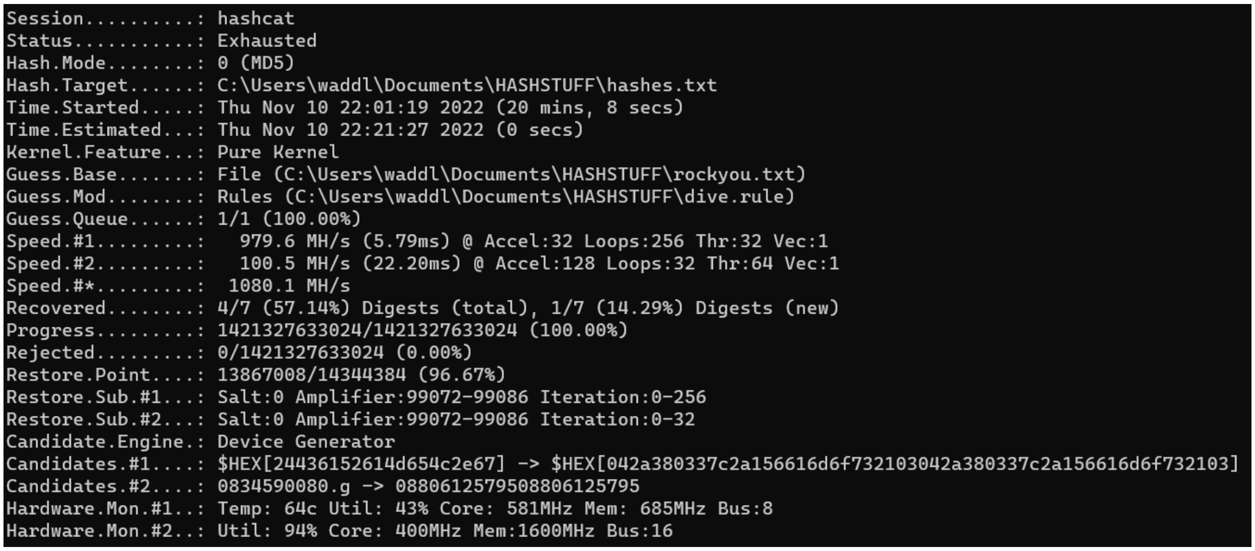
Rule-Based Attacks:

Additionally, **Rules** can be set to tell the program to try permutations of each password in the word list. This increases our cracking time exponentially, but it will allow us to crack far more passwords.

```
.\hashcat.exe -m 0 -a 0 -o "C:\Users\waddl\Documents\HASHSTUFF\cracked.txt"
```

```
"C:\Users\waddl\Documents\HASHSTUFF\hashes.txt"
"C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt" -r
"C:\Users\waddl\Documents\HASHSTUFF\dive.rule"
```

The results:



```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 0 (MD5)
Hash.Target......: C:\Users\waddl\Documents\HASHSTUFF\hashes.txt
Time.Started.....: Thu Nov 10 22:01:19 2022 (20 mins, 8 secs)
Time.Estimated...: Thu Nov 10 22:21:27 2022 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt)
Guess.Mod........: Rules (C:\Users\waddl\Documents\HASHSTUFF\dive.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   979.6 MH/s (5.79ms) @ Accel:32 Loops:256 Thr:32 Vec:1
Speed.#2.........:   100.5 MH/s (22.20ms) @ Accel:128 Loops:32 Thr:64 Vec:1
Speed.#*.........:  1080.1 MH/s
Recovered........: 4/7 (57.14%) Digests (total), 1/7 (14.29%) Digests (new)
Progress.........: 1421327633024/1421327633024 (100.00%)
Rejected.........: 0/1421327633024 (0.00%)
Restore.Point....: 13867008/14344384 (96.67%)
Restore.Sub.#1...: Salt:0 Amplifier:99072-99086 Iteration:0-256
Restore.Sub.#2...: Salt:0 Amplifier:99072-99086 Iteration:0-32
Candidate.Engine.: Device Generator
Candidates.#1....: $HEX[24436152614d654c2e67] -> $HEX[042a380337c2a156616d6f732103042a380337c2a156616d6f732103]
Candidates.#2....: 0834590080.g -> 0880612579508806125795
Hardware.Mon.#1..: Temp: 64c Util: 43% Core: 581MHz Mem: 685MHz Bus:8
Hardware.Mon.#2..: Util: 94% Core: 400MHz Mem:1600MHz Bus:16
```

As shown above, all 20 minutes of that got us exactly one more password. Not exactly the fancy 20 second hacking they show in the movies.

Next, we will attempt to crack a windows user's password. It's a little different in some ways, but is roughly the same process as the above with a little extra work we have to do.

Windows uses the older less secure MD4 hash, with a salt – usually the machine's name.

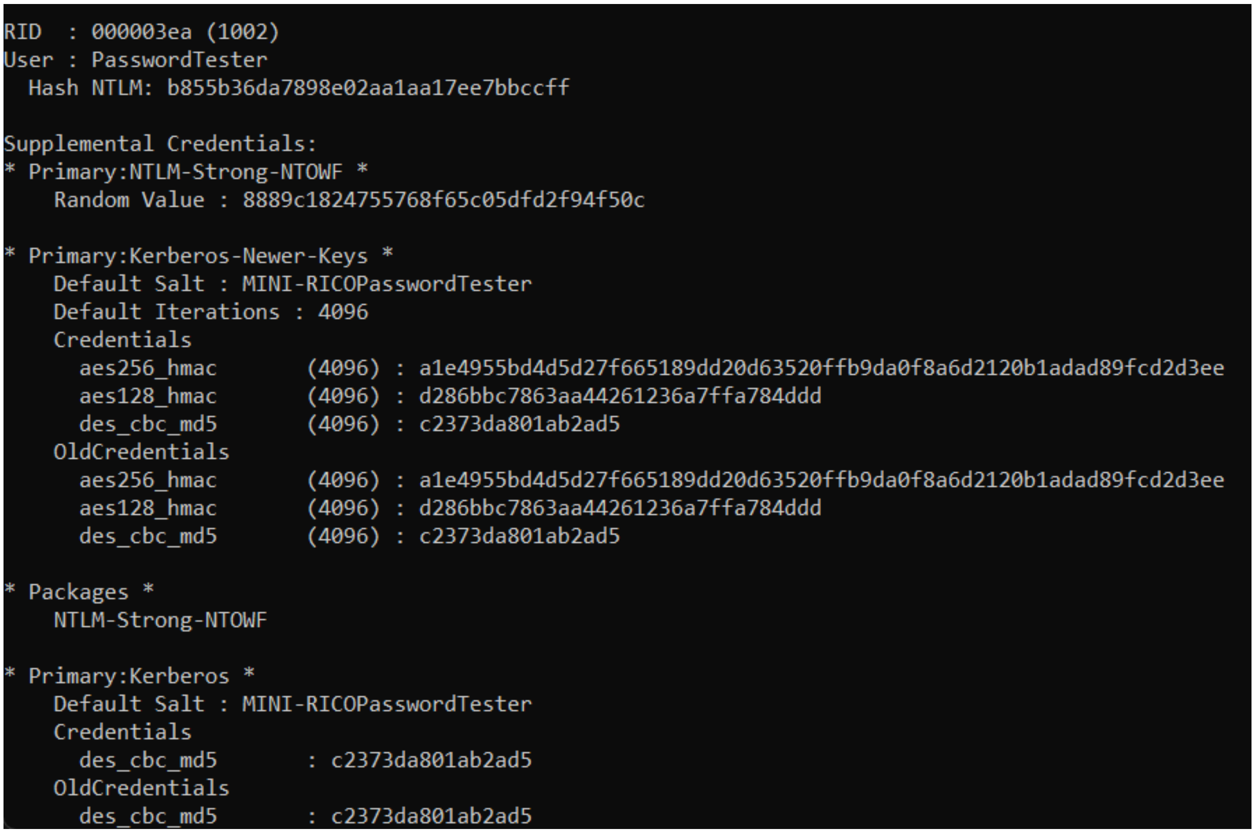
First, a testing user is created that can have an insecure password. The password set is doglover24. Next, Mimikatz is used to pull the NTLM password hash from windows registries. Mimikatz is used in actual exploits, which windows does not like for obvious reasons, so windows antivirus will be disabled temporarily as otherwise windows will automatically quarantine it. Next, mimikatz is run as admin using the commands:

```
Privilege::debug
Token::elevate
```

```
Lsadump::sam SAM.hiv SYSTEM.hiv
```

The first two give mimikatz elevated permissions and the 3rd command is what prints out the extracted information.

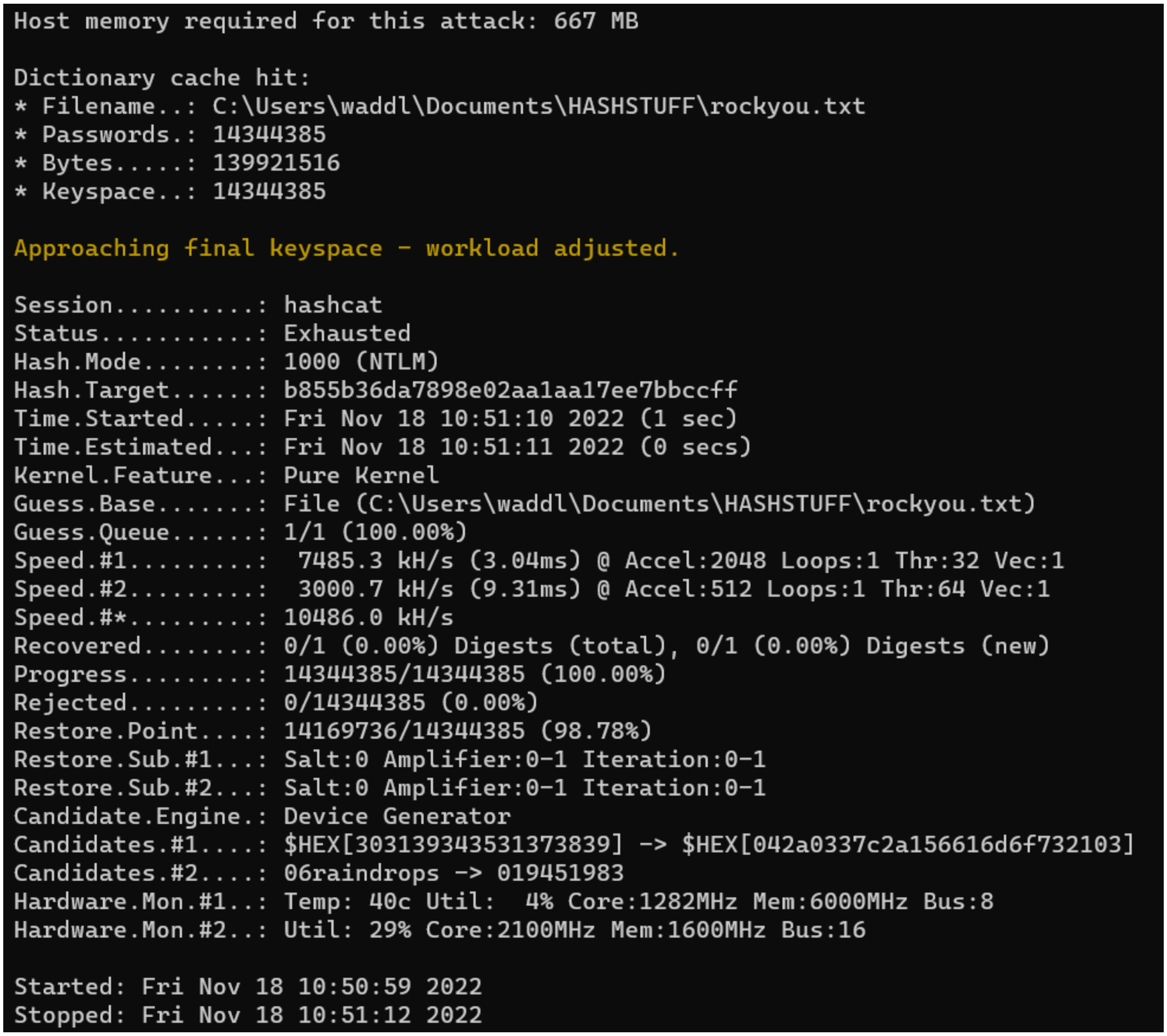
The results:



```
RID  : 000003ea (1002)
User : PasswordTester
  Hash NTLM: b855b36da7898e02aa1aa17ee7bbccff

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : 8889c1824755768f65c05dfd2f94f50c

* Primary:Kerberos-Newer-Keys *
    Default Salt : MINI-RICOPasswordTester
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : a1e4955bd4d5d27f665189dd20d63520ffb9da0f8a6d2120b1adad89fcd2d3ee
      aes128_hmac       (4096) : d286bbc7863aa44261236a7ffa784ddd
      des_cbc_md5       (4096) : c2373da801ab2ad5
    OldCredentials
      aes256_hmac       (4096) : a1e4955bd4d5d27f665189dd20d63520ffb9da0f8a6d2120b1adad89fcd2d3ee
      aes128_hmac       (4096) : d286bbc7863aa44261236a7ffa784ddd
      des_cbc_md5       (4096) : c2373da801ab2ad5

* Packages *
    NTLM-Strong-NTOWF

* Primary:Kerberos *
    Default Salt : MINI-RICOPasswordTester
    Credentials
      des_cbc_md5       : c2373da801ab2ad5
    OldCredentials
      des_cbc_md5       : c2373da801ab2ad5
```

This time mode 1000 was used, which is listed in hashcat as the NTLM hash mode. First, the basic rockyou.txt wordlist was used.

```
.\hashcat -m 1000 -a 0 -o cracked.txt C:\Users\waddl\Documents\HASHSTUFF\real.txt
C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt
```

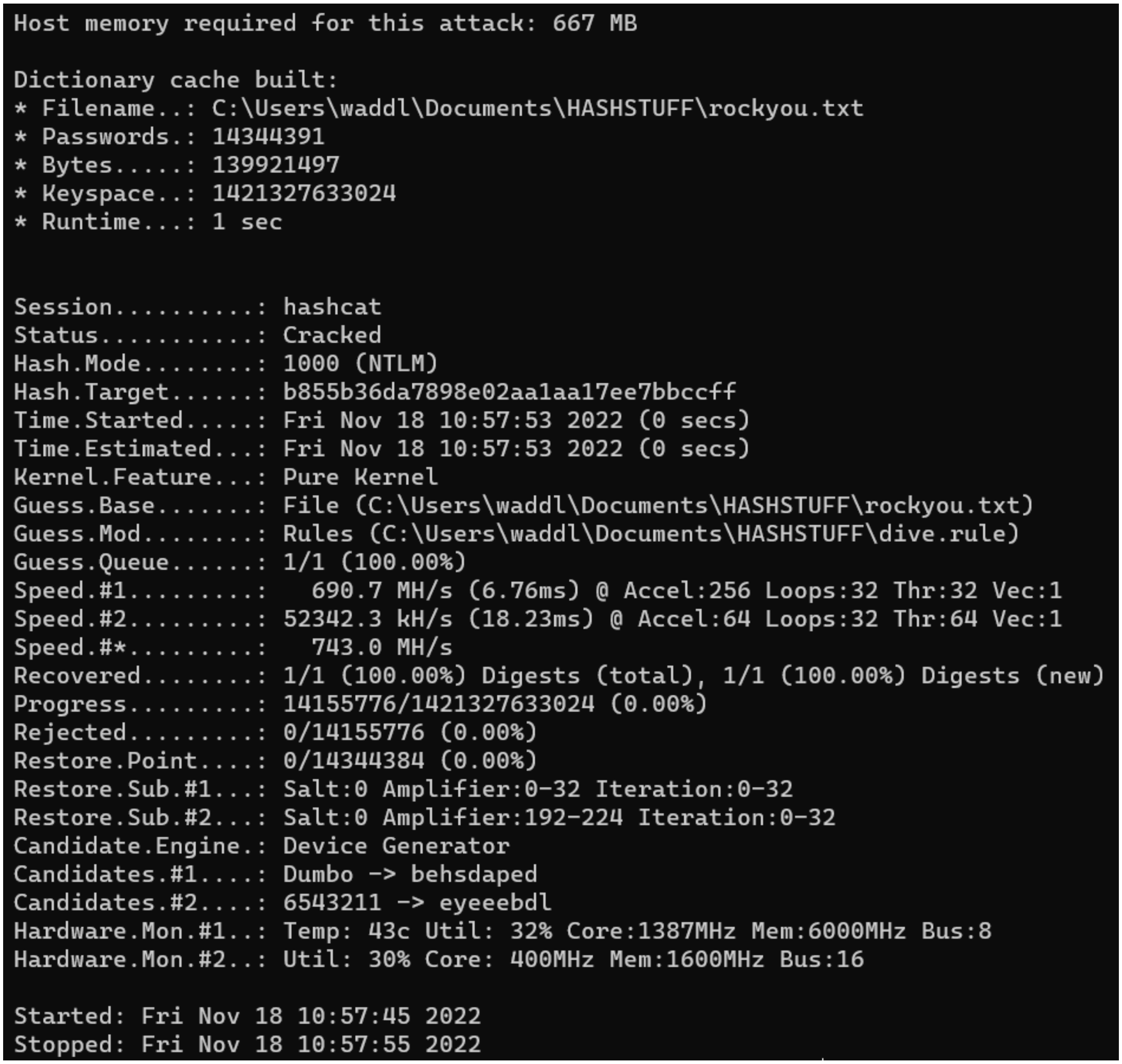


```
Host memory required for this attack: 667 MB

Dictionary cache hit:
* Filename..: C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921516
* Keyspace..: 14344385

Approaching final keyspace - workload adjusted.

Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 1000 (NTLM)
Hash.Target......: b855b36da7898e02aa1aa17ee7bbccff
Time.Started.....: Fri Nov 18 10:51:10 2022 (1 sec)
Time.Estimated...: Fri Nov 18 10:51:11 2022 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  7485.3 kH/s (3.04ms) @ Accel:2048 Loops:1 Thr:32 Vec:1
Speed.#2.........:  3000.7 kH/s (9.31ms) @ Accel:512 Loops:1 Thr:64 Vec:1
Speed.#*.........: 10486.0 kH/s
Recovered........: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.........: 14344385/14344385 (100.00%)
Rejected.........: 0/14344385 (0.00%)
Restore.Point....: 14169736/14344385 (98.78%)
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....: $HEX[303139343531373839] -> $HEX[042a0337c2a156616d6f732103]
Candidates.#2....: 06raindrops -> 019451983
Hardware.Mon.#1..: Temp: 40c Util:  4% Core:1282MHz Mem:6000MHz Bus:8
Hardware.Mon.#2..: Util: 29% Core:2100MHz Mem:1600MHz Bus:16

Started: Fri Nov 18 10:50:59 2022
Stopped: Fri Nov 18 10:51:12 2022
```

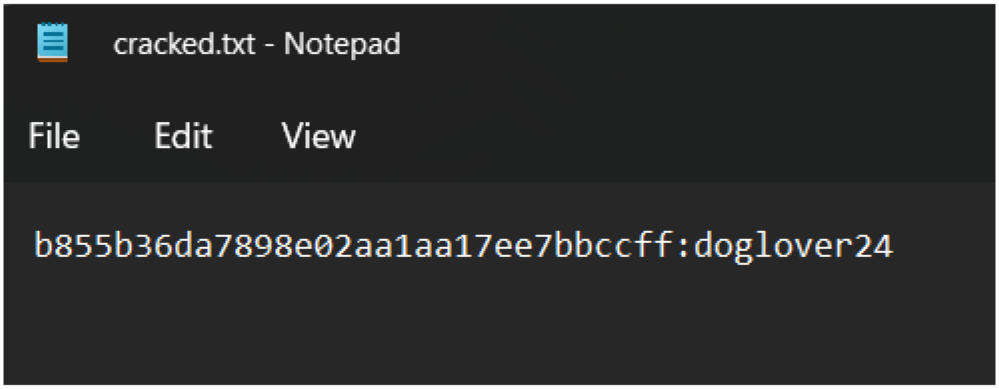
This did not look promising. It appeared as though the basic example password was not able to be cracked.

So, next, the dive.rule ruleset was assigned:

.\hashcat -m 1000 -a 0 -o cracked.txt C:\Users\waddl\Documents\HASHSTUFF\real.txt
C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt -r
C:\Users\waddl\Documents\HASHSTUFF\dive.rule



```
Host memory required for this attack: 667 MB

Dictionary cache built:
* Filename..: C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt
* Passwords.: 14344391
* Bytes.....: 139921497
* Keyspace..: 1421327633024
* Runtime...: 1 sec


Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 1000 (NTLM)
Hash.Target......: b855b36da7898e02aa1aa17ee7bbccff
Time.Started.....: Fri Nov 18 10:57:53 2022 (0 secs)
Time.Estimated...: Fri Nov 18 10:57:53 2022 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (C:\Users\waddl\Documents\HASHSTUFF\rockyou.txt)
Guess.Mod........: Rules (C:\Users\waddl\Documents\HASHSTUFF\dive.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   690.7 MH/s (6.76ms) @ Accel:256 Loops:32 Thr:32 Vec:1
Speed.#2.........: 52342.3 kH/s (18.23ms) @ Accel:64 Loops:32 Thr:64 Vec:1
Speed.#*.........:   743.0 MH/s
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 14155776/1421327633024 (0.00%)
Rejected.........: 0/14155776 (0.00%)
Restore.Point....: 0/14344384 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-32 Iteration:0-32
Restore.Sub.#2...: Salt:0 Amplifier:192-224 Iteration:0-32
Candidate.Engine.: Device Generator
Candidates.#1....: Dumbo -> behsdaped
Candidates.#2....: 6543211 -> eyeeebdl
Hardware.Mon.#1..: Temp: 43c Util: 32% Core:1387MHz Mem:6000MHz Bus:8
Hardware.Mon.#2..: Util: 30% Core: 400MHz Mem:1600MHz Bus:16

Started: Fri Nov 18 10:57:45 2022
Stopped: Fri Nov 18 10:57:55 2022
```



```
b855b36da7898e02aa1aa17ee7bbccff:doglover24
```

As shown above, the attack was successful with less than a second to crack the password doglover24.

Running on Apple Silicon

System setup:

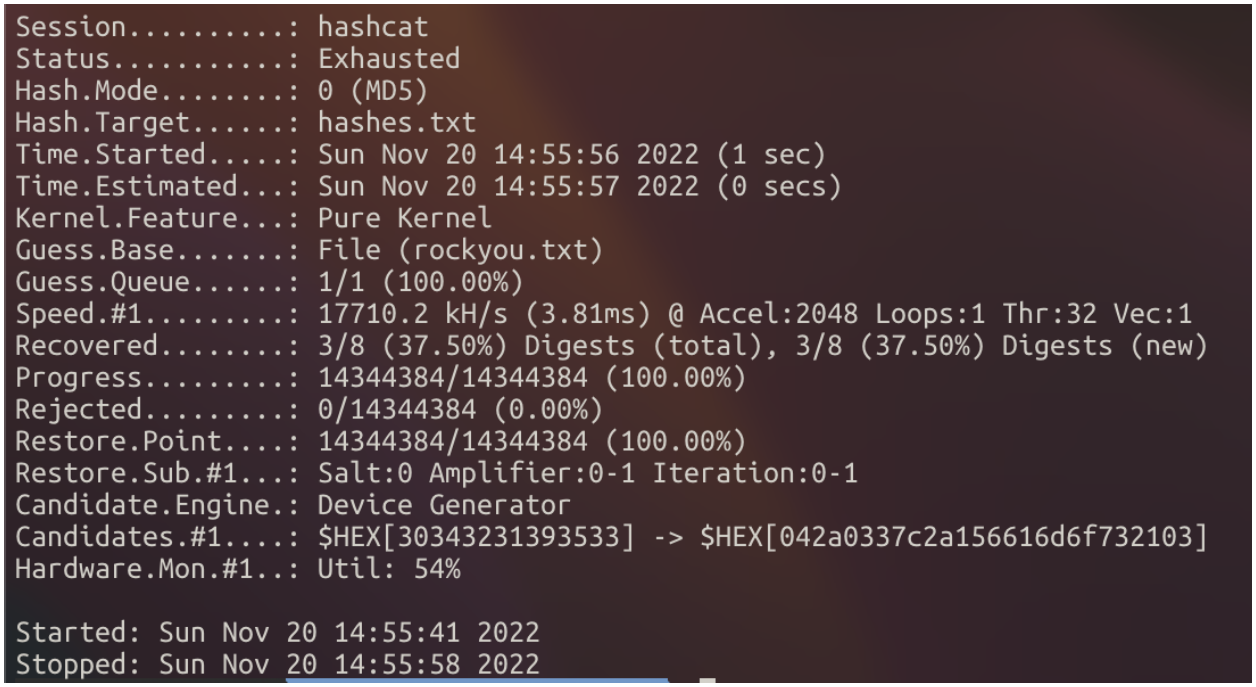
MacBook Air
Graphics Processor: 10 Core Built-in Apple M2 GPU
Processor: Apple M2
Cores: 8 (4 performance and 4 efficiency) Threads: 8 Base Clock: 3.49GHz
Memory: 16gb DDR5
Operating System: macOS version: Ventura 13.0 (22A380)

Dictionary Attack

This time running

hashcat -m 0 -a 0 -o "out.txt"
"hashes.txt"
"rockyou.txt"

we get the following output



```
Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 0 (MD5)
Hash.Target......: hashes.txt
Time.Started.....: Sun Nov 20 14:55:56 2022 (1 sec)
Time.Estimated...: Sun Nov 20 14:55:57 2022 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........: 17710.2 kH/s (3.81ms) @ Accel:2048 Loops:1 Thr:32 Vec:1
Recovered........: 3/8 (37.50%) Digests (total), 3/8 (37.50%) Digests (new)
Progress.........: 14344384/14344384 (100.00%)
Rejected.........: 0/14344384 (0.00%)
Restore.Point....: 14344384/14344384 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: $HEX[30343231393533] -> $HEX[042a0337c2a156616d6f732103]
Hardware.Mon.#1..: Util: 54%

Started: Sun Nov 20 14:55:41 2022
Stopped: Sun Nov 20 14:55:58 2022
```

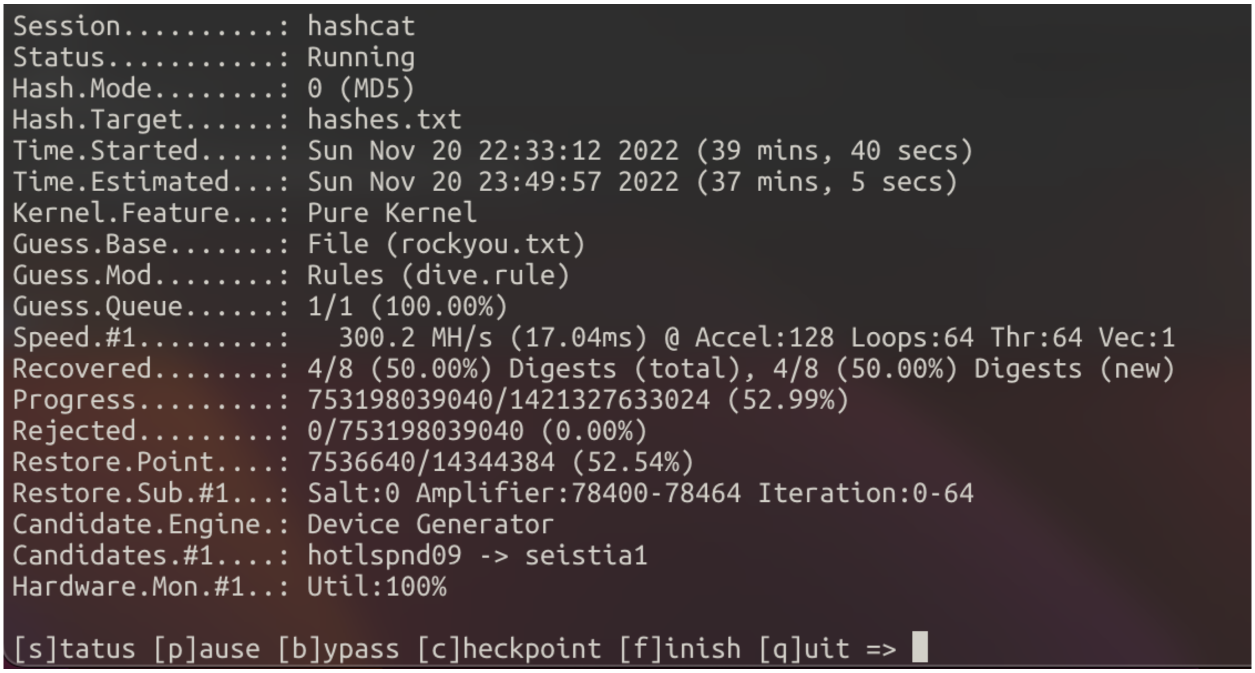
This time, the program was able to identify 3 out of the 8 hashes and convert them back to the original password as shown below in the output file:

5f4dcc3b5aa765d61d8327deb882cf99:password
e99a18c428cb38d5f260853678922e03:abc123
75b71aa6842e450f12aca00fdf54c51d:P455w0rd

Rule-Based Attacks:

hashcat -m 0 -a 0 -o "cracked.txt"
“hashes.txt”
"rockyou.txt" -r
"dive.rule"

Results:



```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 0 (MD5)
Hash.Target......: hashes.txt
Time.Started.....: Sun Nov 20 22:33:12 2022 (39 mins, 40 secs)
Time.Estimated...: Sun Nov 20 23:49:57 2022 (37 mins, 5 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (rockyou.txt)
Guess.Mod........: Rules (dive.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   300.2 MH/s (17.04ms) @ Accel:128 Loops:64 Thr:64 Vec:1
Recovered........: 4/8 (50.00%) Digests (total), 4/8 (50.00%) Digests (new)
Progress.........: 753198039040/1421327633024 (52.99%)
Rejected.........: 0/753198039040 (0.00%)
Restore.Point....: 7536640/14344384 (52.54%)
Restore.Sub.#1...: Salt:0 Amplifier:78400-78464 Iteration:0-64
Candidate.Engine.: Device Generator
Candidates.#1....: hotlspnd09 -> seistia1
Hardware.Mon.#1..: Util:100%

[s]tatus [p]ause [b]ypass [c]heckpoint [f]inish [q]uit =>
```

In the case of the Apple Silicon rule-based attack, we were able to recover one more password in about 40 minutes. This is about twice as long as we observed on the ROG Zephyrus G14 system.

Conclusion:

Through the process of our demonstrations here, we demonstrated the vulnerability of short passwords, length of time longer passwords can take to be broken, and the importance of creating a password that will protect a system from attacks more successfully. We see that more high powered cores means less time needed for the action, and how different types of attacks can provide an increased efficacy for speed of attack based on the many different parameters demonstrated today. Many companies now create recommended passwords that do not contain dictionary based words which helps mitigate the vulnerabilities shown today. It is highly recommended to use these, as you will be more secure, at least until new vulnerabilities are discovered, or an alternative to the dreaded password is introduced and becomes widely used.

Work Cited:

1) Fripp, Charlie. *Use This Chart to See How Long It'll Take Hackers to Crack Your Passwords. Komando.com*, https://www.komando.com/security-privacy/check-your-password-strength/783192/. Accessed 22 March 2021.

2) https://www.csoonline.com/article/3542630/hashcat-explained-why-you-might-need-this-password-cracker.html

3) https://cryptokait.com/2020/02/24/password-cracking-with-hashcat/