**Name:** Hemant Jain

**Lab Progress Report Due Date:** 03/01/2021

**Current Week Since Start Date:** Week 6 (03/01/2021– 03/08/2021)

**Reporting Week:** From Feb 23, 2021 to Mar 01, 2021

**Summary about the TestOut Module-5 Learning:**

From the TestOut LabSim, I learnt about the Identity, Access and Account management techniques and various roles to be performed. The lessons covered included the topic of discussion namely on Access Control, Authentication, Authorization and Accounting – 4A’s.

Access Control Policies varies from Preventive, Detective, Corrective, Deterrent, Recovery and Compensative. The three different type s of access controls which helps us to include encryption, one-time passwords, and firewall rules.

Continuing with the access control best practices in details for the unknown resource access denials. To avoid the creeping privileges occurrences and best protect the security of information we have Account Creation, Active Accounts, and Old Accounts account’s life cycles.

Authentication included the access resources on a network, and the Multi-Factor Authentication (MFA) which requires more than one method of identification and uses factors and multiple attributes to considerations.

For the unique biometric authentications and technologies to consider we have some parameters like Universal, Unique, Permanent, Collectible, Circumvention, Accuracy. It could include Retina, Iris, Facial, Voice, Vein, Gait. Key Terms Authorization, Access Control List(ACL), Permission, DACL, SACL, Security Principal. Windows Operating System Roles Facts Stand-alone model, Workgroup Network Model, Client-Server Network Model. Local user accounts, Workgroup membership, Microsoft account sign-in, Domain account sign-in, Azure Active Directory account sign-in.

Learnt about creating Organizational Units(OUs), Deleting OUs, Use Group Policy, Create and link a GPO, Create User accounts, manage user accounts, Create a group, Create global groups.

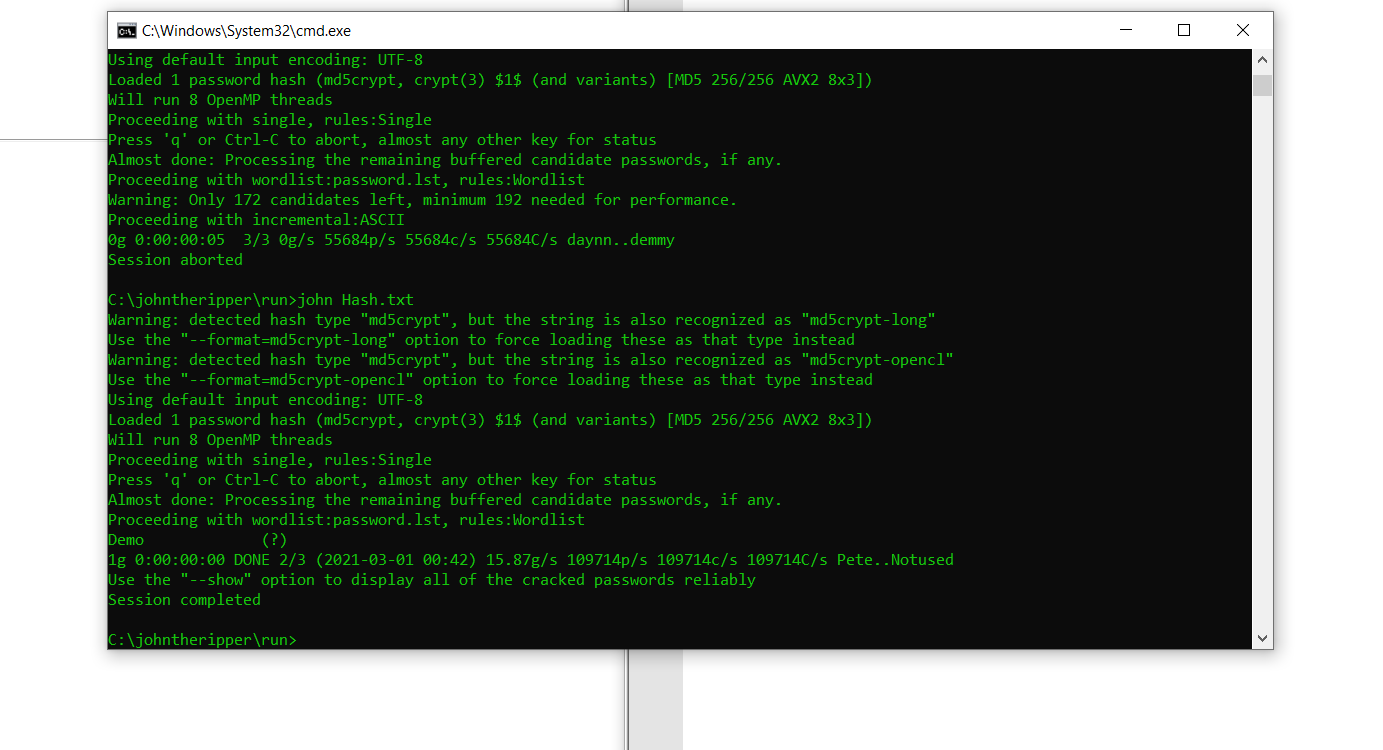
Group policy is a set of configuration setting applied to users or computers. Every GPO Structure has GPO categories including Computer Configuration, User Configuration.

Hardening Authentication Methods including the Password Policies, MFAs, Account Restrictions, Account Maintenance, Limit Remote Access, Account Lockout Policies. Smart Card Authentication Facts benefits a lot in tamper-resistant storage for a user’s private key and PIIs.

Hands-on practice on Linux platform was quite helpful along with the hands-on embedded labs to-dos. And at the end it was a quick fast learning about the Network Authentication Facts overviews the different protocols it include in itself.

**In-class Lab Homework:**

Cracking the MD5 hashed password using the JohnTheRipper password cracking tool for easy wording passwords.



**Part – 1 – Test Password Security**

**A computer screen capture

Description automatically generated with medium confidence**

**A computer screen capture

Description automatically generated with medium confidence**

**Part-2 – Check an Account for a Prior Data Breach**

**Question:** Was one of your accounts breached? If so, which one(s)? and what will you do to make it more secure?

**Ans:** Luckily, none of my accounts are hacked yet previously in some data breaches. To prevent them happening I should be enabling 2-Factor Authentication and keeping the password changed after every 30 days cycle to keep it healthy and safe from getting breached by some hackers.

**Part-3 – Sign Up for Two Factor Authentication**

**Question:** Which service did you enable 2FA for?

**Ans:** I do have enabled out my Northeastern Email Id and my other @northeastern.edu accounts linked using the Duo 2FA Authentication App.

Apart from that, I have all my Cloud accounts on AWS,GCP and my PayPal account secured using the 2FA Google Authenticator App.

**Part-4 – Install and Setup up a Password Manager**

Graphical user interface, application

Description automatically generated

**Part-5 – Online Password Attack**

**Question:** What was the password (Scan the results to find the line beginning with [443][http-get])?

Ans: [443][http-get] host: is.theorizeit.org login: istheory password: 9876543210

**Question:** Approximately how many passwords a second were you able to try? **Hint:** You may need to calculate this from the start and end time along with number of guesses made.

**Ans:**

Starting at 2021-02-28 20:30:59

Finished at 2021-02-28 20:31:33

Password tried 1236 of 14344399

Password tried/sec = 1236/94 = 13 passwords per sec

**Part-6 – Offline Attack using Hashcat**

**Step – 1:**

root@kali:~# python office2john.py hashcat.doc

hashcat.doc:$oldoffice$1\*b405d2e0bef836cd538b96de63d64cfd\*7c33fab607ed148ae5f2ca3ee8ca4c0b\*e0e9f79eabc501653af0543e027f0cad:::::hashcat.doc

**Step2 – Save the hash string in file**

root@kali:~# cat hashpassword.txt

$1\*b405d2e0bef836cd538b96de63d64cfd\*7c33fab607ed148ae5f2ca3ee8ca4c0b\*e0e9f79eabc501653af0543e027f0cad

**Step 3 – Crack the hash**

root@kali:~# hashcat --force -a 0 -m 9700 -o crack.txt hashpassword.txt /usr/share/wordlists/rockyou.txt

Output -

hashcat (v5.1.0) starting...

OpenCL Platform #1: The pocl project

====================================

\* Device #1: pthread-Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2048/5191 MB allocatable, 2MCU

/usr/share/hashcat/OpenCL/m09700\_a0-optimized.cl: Pure OpenCL kernel not found, falling back to optimized OpenCL kernel

Hashes: 1 digests; 1 unique digests, 1 unique salts

Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Rules: 1

Applicable optimizers:

\* Optimized-Kernel

\* Zero-Byte

\* Precompute-Init

\* Not-Iterated

\* Single-Hash

\* Single-Salt

Minimum password length supported by kernel: 0

Maximum password length supported by kernel: 15

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\_SIZE=8 -D DEVICE\_TYPE=2 -D DGST\_R0=0 -D DGST\_R1=1 -D DGST\_R2=2 -D DGST\_R3=3 -D DGST\_ELEM=4 -D KERN\_TYPE=9700 -D \_unroll'

\* Device #1: Kernel m09700\_a0-optimized.d794e54a.kernel not found in cache! Building may take a while...

Dictionary cache built:

\* Filename..: /usr/share/wordlists/rockyou.txt

\* Passwords.: 14344392

\* Bytes.....: 139921507

\* Keyspace..: 14344385

\* Runtime...: 2 secs

Session..........: hashcat

Status...........: Cracked

Hash.Type........: MS Office <= 2003 $0/$1, MD5 + RC4

Hash.Target......: $oldoffice$1\*b405d2e0bef836cd538b96de63d64cfd\*7c33f...7f0cad

Time.Started.....: Sun Feb 28 13:35:34 2021 (0 secs)

Time.Estimated...: Sun Feb 28 13:35:34 2021 (0 secs)

Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)

Guess.Queue......: 1/1 (100.00%)

Speed.#1.........: 297.2 kH/s (7.70ms) @ Accel:32 Loops:1 Thr:64 Vec:8

Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts

Progress.........: 225445/14344385 (1.57%)

Rejected.........: 165/225445 (0.07%)

Restore.Point....: 221347/14344385 (1.54%)

Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1

Candidates.#1....: flutesrock -> anusara

Started: Sun Feb 28 13:35:18 2021

Stopped: Sun Feb 28 13:35:35 2021

**Step-4 – view the output file for cracked hash**

root@kali:~# cat crack.txt

$oldoffice$1\*b405d2e0bef836cd538b96de63d64cfd\*7c33fab607ed148ae5f2ca3ee8ca4c0b\*e0e9f79eabc501653af0543e027f0cad:camp

**Question:** What is the password for hashcat.doc?

**Ans:** camp

**Step-5: Performing same operations for john.doc**

root@kali:~# python office2john.py john.doc

john.doc:$oldoffice$1\*16b19484f9276544547f7b94535fd9c3\*4df800da560ed22757622c804763ec5e\*1e53e6f37bf0f20fd4eb2c84815df1dc:::::john.doc

root@kali:~# vi johnhashstring.txt

root@kali:~# hashcat --force -a 0 -m 9700 -o johncrackedhash.txt johnhashstring.txt /usr/share/wordlists/rockyou.txt

hashcat (v5.1.0) starting...

OpenCL Platform #1: The pocl project

====================================

\* Device #1: pthread-Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2048/5191 MB allocatable, 2MCU

/usr/share/hashcat/OpenCL/m09700\_a0-optimized.cl: Pure OpenCL kernel not found, falling back to optimized OpenCL kernel

Hashes: 1 digests; 1 unique digests, 1 unique salts

Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Rules: 1

Applicable optimizers:

\* Optimized-Kernel

\* Zero-Byte

\* Precompute-Init

\* Not-Iterated

\* Single-Hash

\* Single-Salt

Minimum password length supported by kernel: 0

Maximum password length supported by kernel: 15

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\_SIZE=8 -D DEVICE\_TYPE=2 -D DGST\_R0=0 -D DGST\_R1=1 -D DGST\_R2=2 -D DGST\_R3=3 -D DGST\_ELEM=4 -D KERN\_TYPE=9700 -D \_unroll'

Dictionary cache hit:

\* Filename..: /usr/share/wordlists/rockyou.txt

\* Passwords.: 14344385

\* Bytes.....: 139921507

\* Keyspace..: 14344385

Session..........: hashcat

Status...........: Cracked

Hash.Type........: MS Office <= 2003 $0/$1, MD5 + RC4

Hash.Target......: $oldoffice$1\*16b19484f9276544547f7b94535fd9c3\*4df80...5df1dc

Time.Started.....: Sun Feb 28 14:45:32 2021 (0 secs)

Time.Estimated...: Sun Feb 28 14:45:32 2021 (0 secs)

Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)

Guess.Queue......: 1/1 (100.00%)

Speed.#1.........: 263.0 kH/s (7.96ms) @ Accel:32 Loops:1 Thr:64 Vec:8

Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts

Progress.........: 143448/14344385 (1.00%)

Rejected.........: 88/143448 (0.06%)

Restore.Point....: 139346/14344385 (0.97%)

Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1

Candidates.#1....: juragan -> 23102004

Started: Sun Feb 28 14:45:31 2021

Stopped: Sun Feb 28 14:45:34 2021

root@kali:~# cat johncrackedhash.txt

$oldoffice$1\*16b19484f9276544547f7b94535fd9c3\*4df800da560ed22757622c804763ec5e\*1e53e6f37bf0f20fd4eb2c84815df1dc:attica

**Question:** What is the password for john.doc?

**Ans:** attica

7. **Question:** How many passwords per second can Hashcat running on a Brutalis try on a .doc file (i.e., hashtype "MS Office <= 2003 MD5 + RC4, oldoffice$0, oldoffice$1")?

**Ans -**

Hashtype: MS Office <= 2003 MD5 + RC4, oldoffice$0, oldoffice$1

Speed.Dev.#1.: 219.6 MH/s (108.82ms)

Speed.Dev.#2.: 226.6 MH/s (104.06ms)

Speed.Dev.#3.: 221.1 MH/s (108.15ms)

Speed.Dev.#4.: 221.4 MH/s (107.97ms)

Speed.Dev.#5.: 225.8 MH/s (107.32ms)

Speed.Dev.#6.: 226.6 MH/s (104.02ms)

Speed.Dev.#7.: 225.9 MH/s (107.27ms)

Speed.Dev.#8.: 226.0 MH/s (107.24ms)

Speed.Dev.#\*.: 1792.9 MH/s i.e., 1792.9 Million hashes per second

**Question:** How much faster is Hashcat in cracking .doc MS Office documents (option 9700, "Hashtype: MS Office <= 2003 MD5 + RC4, oldoffice$0, oldoffice$1") compared to Office 2013 documents (option 9600, "Hashtype: Office 2013")?

**Ans –**

Hashtype: Office 2013

Speed.Dev.#1.: 8814 H/s (96.69ms)

Speed.Dev.#2.: 8678 H/s (98.62ms)

Speed.Dev.#3.: 8937 H/s (97.32ms)

Speed.Dev.#4.: 8882 H/s (96.88ms)

Speed.Dev.#5.: 8936 H/s (93.84ms)

Speed.Dev.#6.: 8740 H/s (97.52ms)

Speed.Dev.#7.: 8922 H/s (97.10ms)

Speed.Dev.#8.: 8976 H/s (96.88ms)

Speed.Dev.#\*.: 70884 H/s i.e., 70884 hashes per second

**Optional:** Install hashcat on your own machine (not the VM). See how your benchmarks compare against a Brutalis. Note that running benchmarks on the VM will break once it reaches scrypt before complete results are reported.

Command to run on prompt: hashcat -b --force

**Output on my local system: -**

C:\Users\jainh\Downloads\hashcat-6.1.1>hashcat -b --force

hashcat (v6.1.1) starting in benchmark mode...

Benchmarking uses hand-optimized kernel code by default.

You can use it in your cracking session by setting the -O option.

Note: Using optimized kernel code limits the maximum supported password length.

To disable the optimized kernel code in benchmark mode, use the -w option.

You have enabled --force to bypass dangerous warnings and errors!

This can hide serious problems and should only be done when debugging.

Do not report hashcat issues encountered when using --force.

\* Device #1: CUDA SDK Toolkit installation NOT detected.

CUDA SDK Toolkit installation required for proper device support and utilization

Falling back to OpenCL Runtime

nvmlDeviceGetFanSpeed(): Not Supported

OpenCL API (OpenCL 1.2 CUDA 11.1.114) - Platform #1 [NVIDIA Corporation]

========================================================================

\* Device #1: GeForce MX250, 3392/4096 MB (1024 MB allocatable), 3MCU

OpenCL API (OpenCL 2.1 ) - Platform #2 [Intel(R) Corporation]

=============================================================

\* Device #2: Intel(R) UHD Graphics, 6390/6454 MB (3227 MB allocatable), 24MCU

Benchmark relevant options:

===========================

\* --force

\* --optimized-kernel-enable

Hashmode: 0 - MD5

Speed.#1.........: 758.2 MH/s (264.98ms) @ Accel:64 Loops:1024 Thr:1024 Vec:1

Speed.#2.........: 251.3 MH/s (98.19ms) @ Accel:128 Loops:1024 Thr:8 Vec:4

Speed.#\*.........: 1009.5 MH/s

Hashmode: 100 - SHA1

Speed.#1.........: 339.5 MH/s (295.82ms) @ Accel:32 Loops:1024 Thr:1024 Vec:1

Speed.#2.........: 61917.1 kH/s (199.00ms) @ Accel:256 Loops:256 Thr:8 Vec:4

Speed.#\*.........: 401.4 MH/s

Hashmode: 1400 - SHA2-256

Speed.#1.........: 105.6 MH/s (237.57ms) @ Accel:16 Loops:512 Thr:1024 Vec:1

Speed.#2.........: 65530.3 kH/s (93.17ms) @ Accel:64 Loops:512 Thr:8 Vec:4

Speed.#\*.........: 171.1 MH/s

Hashmode: 1700 - SHA2-512

Speed.#1.........: 34791.6 kH/s (180.30ms) @ Accel:2 Loops:1024 Thr:1024 Vec:1

Speed.#2.........: 10864.1 kH/s (67.80ms) @ Accel:4 Loops:1024 Thr:8 Vec:1

Speed.#\*.........: 45655.7 kH/s

Hashmode: 22000 - WPA-PBKDF2-PMKID+EAPOL (Iterations: 4095)

Speed.#1.........: 14683 H/s (208.61ms) @ Accel:4 Loops:1024 Thr:1024 Vec:1

Speed.#2.........: 5013 H/s (74.86ms) @ Accel:8 Loops:1024 Thr:8 Vec:1

Speed.#\*.........: 19696 H/s

Hashmode: 1000 - NTLM

Speed.#1.........: 1287.2 MH/s (77.60ms) @ Accel:32 Loops:1024 Thr:1024 Vec:1

Speed.#2.........: 683.6 MH/s (72.64ms) @ Accel:512 Loops:512 Thr:8 Vec:4

Speed.#\*.........: 1970.7 MH/s

Hashmode: 3000 - LM

Speed.#1.........: 3120.8 MH/s (63.43ms) @ Accel:1024 Loops:1024 Thr:64 Vec:1

Speed.#2.........: 223.6 MH/s (109.84ms) @ Accel:128 Loops:1024 Thr:8 Vec:1

Speed.#\*.........: 3344.3 MH/s

Hashmode: 5500 - NetNTLMv1 / NetNTLMv1+ESS

Speed.#1.........: 2984.3 MH/s (66.83ms) @ Accel:64 Loops:1024 Thr:1024 Vec:1

Speed.#2.........: 348.2 MH/s (71.27ms) @ Accel:128 Loops:1024 Thr:8 Vec:4

Speed.#\*.........: 3332.5 MH/s

Hashmode: 5600 - NetNTLMv2

Speed.#1.........: 59317.4 kH/s (105.50ms) @ Accel:8 Loops:256 Thr:1024 Vec:1

Speed.#2.........: 22878.0 kH/s (67.44ms) @ Accel:64 Loops:128 Thr:8 Vec:4

Speed.#\*.........: 82195.4 kH/s

Hashmode: 1500 - descrypt, DES (Unix), Traditional DES

Speed.#1.........: 125.1 MH/s (49.24ms) @ Accel:32 Loops:1024 Thr:64 Vec:1

Speed.#2.........: 6922.2 kH/s (112.06ms) @ Accel:4 Loops:1024 Thr:8 Vec:1

Speed.#\*.........: 132.0 MH/s

Hashmode: 500 - md5crypt, MD5 (Unix), Cisco-IOS $1$ (MD5) (Iterations: 1000)

Speed.#1.........: 1376.9 kH/s (68.98ms) @ Accel:64 Loops:500 Thr:1024 Vec:1

Speed.#2.........: 206.5 kH/s (116.25ms) @ Accel:128 Loops:1000 Thr:8 Vec:4

Speed.#\*.........: 1583.4 kH/s

Hashmode: 3200 - bcrypt $2\*$, Blowfish (Unix) (Iterations: 32)

Speed.#1.........: 2722 H/s (23.04ms) @ Accel:4 Loops:16 Thr:11 Vec:1

Speed.#2.........: 504 H/s (92.22ms) @ Accel:1 Loops:4 Thr:16 Vec:1

Speed.#\*.........: 3225 H/s

Hashmode: 1800 - sha512crypt $6$, SHA512 (Unix) (Iterations: 5000)

Speed.#1.........: 22981 H/s (52.14ms) @ Accel:4 Loops:512 Thr:1024 Vec:1

Speed.#2.........: 1167 H/s (130.14ms) @ Accel:8 Loops:512 Thr:8 Vec:1

Speed.#\*.........: 24148 H/s

Hashmode: 7500 - Kerberos 5, etype 23, AS-REQ Pre-Auth

Speed.#1.........: 41395.7 kH/s (75.21ms) @ Accel:256 Loops:64 Thr:64 Vec:1

Speed.#2.........: 3029.5 kH/s (63.60ms) @ Accel:2 Loops:64 Thr:64 Vec:4

Speed.#\*.........: 44425.3 kH/s

Hashmode: 13100 - Kerberos 5, etype 23, TGS-REP

Speed.#1.........: 41393.1 kH/s (75.28ms) @ Accel:256 Loops:64 Thr:64 Vec:1

Speed.#2.........: 3422.2 kH/s (56.04ms) @ Accel:4 Loops:32 Thr:64 Vec:4

Speed.#\*.........: 44815.3 kH/s

Hashmode: 15300 - DPAPI masterkey file v1 (Iterations: 23999)

Speed.#1.........: 10159 H/s (51.04ms) @ Accel:8 Loops:512 Thr:1024 Vec:1

Speed.#2.........: 919 H/s (69.60ms) @ Accel:128 Loops:64 Thr:8 Vec:1

Speed.#\*.........: 11078 H/s

Hashmode: 15900 - DPAPI masterkey file v2 (Iterations: 12899)

Speed.#1.........: 4705 H/s (49.82ms) @ Accel:1 Loops:1024 Thr:1024 Vec:1

Speed.#2.........: 276 H/s (107.72ms) @ Accel:8 Loops:256 Thr:8 Vec:1

Speed.#\*.........: 4981 H/s

Hashmode: 7100 - macOS v10.8+ (PBKDF2-SHA512) (Iterations: 1023)

Speed.#1.........: 58903 H/s (49.80ms) @ Accel:32 Loops:31 Thr:1024 Vec:1

Speed.#2.........: 4368 H/s (33.48ms) @ Accel:4 Loops:255 Thr:8 Vec:1

Speed.#\*.........: 63271 H/s

Hashmode: 11600 - 7-Zip (Iterations: 16384)

Speed.#1.........: 52103 H/s (56.48ms) @ Accel:4 Loops:4096 Thr:1024 Vec:1

Speed.#2.........: 4546 H/s (81.88ms) @ Accel:8 Loops:4096 Thr:8 Vec:4

Speed.#\*.........: 56650 H/s

Hashmode: 12500 - RAR3-hp (Iterations: 262144)

Speed.#1.........: 6331 H/s (60.20ms) @ Accel:2 Loops:16384 Thr:1024 Vec:1

Speed.#2.........: 823 H/s (54.71ms) @ Accel:4 Loops:16384 Thr:8 Vec:4

Speed.#\*.........: 7154 H/s

Hashmode: 13000 - RAR5 (Iterations: 32799)

Speed.#1.........: 5132 H/s (72.08ms) @ Accel:4 Loops:1024 Thr:1024 Vec:1

Speed.#2.........: 436 H/s (103.23ms) @ Accel:32 Loops:256 Thr:8 Vec:1

Speed.#\*.........: 5568 H/s

Hashmode: 6211 - TrueCrypt RIPEMD160 + XTS 512 bit (Iterations: 1999)

Speed.#1.........: 38206 H/s (76.04ms) @ Accel:8 Loops:256 Thr:1024 Vec:1

Speed.#2.........: 2737 H/s (66.50ms) @ Accel:4 Loops:512 Thr:8 Vec:1

Speed.#\*.........: 40944 H/s

Hashmode: 13400 - KeePass 1 (AES/Twofish) and KeePass 2 (AES) (Iterations: 24569)

Speed.#1.........: 4605 H/s (222.00ms) @ Accel:8 Loops:1024 Thr:1024 Vec:1

Speed.#2.........: 441 H/s (68.66ms) @ Accel:8 Loops:512 Thr:8 Vec:1

Speed.#\*.........: 5046 H/s

Hashmode: 6800 - LastPass + LastPass sniffed (Iterations: 499)

Speed.#1.........: 327.8 kH/s (57.95ms) @ Accel:32 Loops:124 Thr:1024 Vec:1

Speed.#2.........: 39429 H/s (49.19ms) @ Accel:32 Loops:249 Thr:8 Vec:1

Speed.#\*.........: 367.2 kH/s

Hashmode: 11300 - Bitcoin/Litecoin wallet.dat (Iterations: 200459)

Speed.#1.........: 643 H/s (48.38ms) @ Accel:4 Loops:512 Thr:1024 Vec:1

Speed.#2.........: 31 H/s (59.98ms) @ Accel:4 Loops:512 Thr:8 Vec:1

Speed.#\*.........: 674 H/s

Started: Sun Feb 28 22:06:18 2021

Stopped: Sun Feb 28 22:19:17 2021

C:\Users\jainh\Downloads\hashcat-6.1.1>

**Question:** How does an offline password attack compare with the online hydra attack you attempted earlier?

**Ans –** In an offline secret password attack , the programmer is never truly endeavoring to login to the application specialist. This suggests it is subtle to the security bunch and logs. This moreover infers that fundamental securities, for example, account lockouts will not work. This can be on the grounds that the programmer will take it offline, find the mystery key/password, and a while later fair one right try will be enrolled by the application.

Whereas online password breaches are limited by the speed of the organization, offline attacks are confined essentially by the speed of the PC the programmer is utilizing to break them.

**Part 7 – Cracking LinkedIn Hashes Using Hashcat**

**Step 1 – download the hashfile**

Command to download the dataset on the local kali machine using wget package is as follows:

wget <https://raw.githubusercontent.com/deargle/security-assignments/master/labs/files/LinkedIn_HalfMillionHashes.txt>

**Step 2 – run following command to crack the hashes and store it in output file , --remove flag will remove the cracked hashes from the file that we downloaded in step 1**

**Command**

hashcat --force -m 100 --remove --outfile=LinkedIn\_cracked.txt LinkedIn\_HalfMillionHashes.txt /usr/share/wordlists/rockyou.txt

**Step 3 – count the number of hashes cracked**

**Command** – wc -l LinkedIn\_cracked.txt

**Output** - 144622 LinkedIn\_cracked.txt

**Count the remaining hashes**

**Command** - wc -l LinkedIn\_HalfMillionHashes.txt

**Output** - 355378 LinkedIn\_HalfMillionHashes.txt

**Question:** How many passwords were you able to recover using the Hashcat command above?

**Ans** – 144622

**Rule based password attack -**

root@kali:~# hashcat --force -m 100 --remove --outfile=LinkedIn\_cracked\_new.txt newLinkedHashes.txt -r /usr/share/hashcat/rules/best64.rule /usr/share/wordlists/rockyou.txt

hashcat (v5.1.0) starting...

OpenCL Platform #1: The pocl project

====================================

\* Device #1: pthread-Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2048/5191 MB allocatable, 2MCU

Hashes: 500000 digests; 500000 unique digests, 1 unique salts

Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Rules: 77

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.

INFO: Removed 144622 hashes found in potfile.

\* 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\_SIZE=8 -D DEVICE\_TYPE=2 -D DGST\_R0=3 -D DGST\_R1=4 -D DGST\_R2=2 -D DGST\_R3=1 -D DGST\_ELEM=5 -D KERN\_TYPE=100 -D \_unroll'

Dictionary cache hit:

\* Filename..: /usr/share/wordlists/rockyou.txt

\* Passwords.: 14344385

\* Bytes.....: 139921507

\* Keyspace..: 1104517645

Approaching final keyspace - workload adjusted.

Session..........: hashcat

Status...........: Exhausted

Hash.Type........: SHA1

Hash.Target......: newLinkedHashes.txt

Time.Started.....: Sun Feb 28 17:38:10 2021 (12 mins, 19 secs)

Time.Estimated...: Sun Feb 28 17:50:29 2021 (0 secs)

Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)

Guess.Mod........: Rules (/usr/share/hashcat/rules/best64.rule)

Guess.Queue......: 1/1 (100.00%)

Speed.#1.........: 1420.2 kH/s (12.70ms) @ Accel:128 Loops:77 Thr:1 Vec:8

Recovered........: 233812/500000 (46.76%) Digests, 0/1 (0.00%) Salts

Recovered/Time...: CUR:698,N/A,N/A AVG:7243,434593,10430250 (Min,Hour,Day)

Progress.........: 1104517645/1104517645 (100.00%)

Rejected.........: 0/1104517645 (0.00%)

Restore.Point....: 14344385/14344385 (100.00%)

Restore.Sub.#1...: Salt:0 Amplifier:0-77 Iteration:0-77

Candidates.#1....: $HEX[206b72697374656e616e6e65] -> $HEX[04a156616d6f]

Started: Sun Feb 28 17:38:06 2021

Stopped: Sun Feb 28 17:50:31 2021

**Question:** How many total passwords were you able to recover after using this rules based attack in combination with the earlier straight attack?

**Ans :** 233812

**Hybrid method that uses a dictionary attack combined with a “mask”**

root@kali:~# hashcat --force -m 100 --remove --outfile=LinkedIn\_cracked\_2.txt newLinkedHashes\_2.txt -i -a 6 /usr/share/wordlists/rockyou.txt ?d?d

hashcat (v5.1.0) starting...

OpenCL Platform #1: The pocl project

====================================

\* Device #1: pthread-Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 2048/5191 MB allocatable, 2MCU

Hashes: 500000 digests; 500000 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.

INFO: Removed 233812 hashes found in potfile.

\* 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\_SIZE=8 -D DEVICE\_TYPE=2 -D DGST\_R0=3 -D DGST\_R1=4 -D DGST\_R2=2 -D DGST\_R3=1 -D DGST\_ELEM=5 -D KERN\_TYPE=100 -D \_unroll'

\* Device #1: Kernel m00100\_a1-pure.91c72d73.kernel not found in cache! Building may take a while...

\* Device #1: Kernel markov\_le.2d9c9c68.kernel not found in cache! Building may take a while...

Dictionary cache hit:

\* Filename..: /usr/share/wordlists/rockyou.txt

\* Passwords.: 14344385

\* Bytes.....: 139921507

\* Keyspace..: 143443850

Approaching final keyspace - workload adjusted.

Session..........: hashcat

Status...........: Exhausted

Hash.Type........: SHA1

Hash.Target......: newLinkedHashes\_2.txt

Time.Started.....: Sun Feb 28 18:37:37 2021 (1 min, 19 secs)

Time.Estimated...: Sun Feb 28 18:38:56 2021 (0 secs)

Guess.Base.......: File (/usr/share/wordlists/rockyou.txt), Left Side

Guess.Mod........: Mask (?d) [1], Right Side

Guess.Queue.Base.: 1/1 (100.00%)

Guess.Queue.Mod..: 1/2 (50.00%)

Speed.#1.........: 1786.5 kH/s (5.61ms) @ Accel:512 Loops:10 Thr:1 Vec:8

Recovered........: 233812/500000 (46.76%) Digests, 0/1 (0.00%) Salts

Recovered/Time...: CUR:0,N/A,N/A AVG:0,0,0 (Min,Hour,Day)

Progress.........: 143443850/143443850 (100.00%)

Rejected.........: 0/143443850 (0.00%)

Restore.Point....: 14344385/14344385 (100.00%)

Restore.Sub.#1...: Salt:0 Amplifier:0-10 Iteration:0-10

Candidates.#1....: $HEX[206b72697374656e616e6e6531] -> $HEX[042a0337c2a156616d6f73210336]

Dictionary cache hit:

\* Filename..: /usr/share/wordlists/rockyou.txt

\* Passwords.: 14344385

\* Bytes.....: 139921507

\* Keyspace..: 1434438500

Approaching final keyspace - workload adjusted.

Session..........: hashcat

Status...........: Exhausted

Hash.Type........: SHA1

Hash.Target......: newLinkedHashes\_2.txt

Time.Started.....: Sun Feb 28 18:38:56 2021 (14 mins, 4 secs)

Time.Estimated...: Sun Feb 28 18:53:00 2021 (0 secs)

Guess.Base.......: File (/usr/share/wordlists/rockyou.txt), Left Side

Guess.Mod........: Mask (?d?d) [2], Right Side

Guess.Queue.Base.: 1/1 (100.00%)

Guess.Queue.Mod..: 2/2 (100.00%)

Speed.#1.........: 1747.2 kH/s (7.36ms) @ Accel:128 Loops:50 Thr:1 Vec:8

Recovered........: 252510/500000 (50.50%) Digests, 0/1 (0.00%) Salts

Recovered/Time...: CUR:66,N/A,N/A AVG:1330,79816,1915595 (Min,Hour,Day)

Progress.........: 1434438500/1434438500 (100.00%)

Rejected.........: 0/1434438500 (0.00%)

Restore.Point....: 14344385/14344385 (100.00%)

Restore.Sub.#1...: Salt:0 Amplifier:50-100 Iteration:0-50

Candidates.#1....: $HEX[206b72697374656e616e6e653133] -> $HEX[042a0337c2a156616d6f7321033638]

Started: Sun Feb 28 18:37:19 2021

Stopped: Sun Feb 28 18:53:02 2021

root@kali:~# wc -l LinkedIn\_cracked\_2.txt

**18698 LinkedIn\_cracked\_2.txt**

root@kali:~# wc -l newLinkedHashes\_2.txt

**247490 newLinkedHashes\_2.txt**

**Question:** How many total passwords were you able to recover after using this hybrid attack combination with the earlier straight and rules-based attacks?

**Ans:** 252510

**Part 8. Secure Password Hashing**

**Question:** How much slower is Hashcat in cracking Bcrypt hashes based on Brutalis Benchmark output ?

Hashtype: bcrypt, Blowfish(OpenBSD)

Speed.Dev.#\*.: 105.7 kH/s

Hashtype: SHA1

Speed.Dev.#\*.: 68771.0 MH/s

**Hashcat is 650624 times slower in cracking Bcrypt hashes than compared to SHA1 hashes**

**Question:** Imagine that Bcrypt is set to a work factor of 12. How many hashing rounds will Bcrypt go through to compute the final hash?

**Ans: 2^12 = > 4096 rounds**

**Function bcrypt**

**Input:**

cost: Number (4..31) log2(Iterations). e.g. 12 ==> 212 = 4,096 iterations

salt: array of Bytes (16 bytes) random salt

password: array of Bytes (1..72 bytes) UTF-8 encoded password

Output:

hash: array of Bytes (24 bytes)

**Question:** An attacker knows that a user generated their password using 8 random lowercase letters exclusively (so character space of 26, length of 8). On average, an attacker needs to try only half of all possible passwords to brute force the password. The attacker has access to a Brutalis. How long would it take to crack the password hash if SHA1 had been used? Bcrypt with the benchmarks shown for a Brutalis?

**Ans:** Possible password combinations - 26 choices for each character

Since the length of password is 8 lower case characters => 26^8 which

is close to 200 billion

Assuming that only half of passwords are required(i.e. 10^11 – 100 billion) to crack a sha1 or bcrypt hash, Brutalis will go through 100 billion passwords to crack the hashed password

Bcrypt’s cumulative speed is 105.7 kh/s

Sha1 cumulative speed 68771.0 MH/s

**Time taken to crack a bcrypt hash = 100 billion / 105.7 kh/s => 15767.90 mins i.e. approx. 11 days**

**Time taken to crack a SHA1 hash = 100 billion / 68771 Mh/s => 1.454 seconds**

**Part 9. Create a Targeted Wordlist Using CeWL**

**Step -1**

**Create a custom dictionary using CeWL for the website neurosecurity.byu.edu: cewl -v -d 2 -m 5 -w custom\_dict.txt https://neurosecurity.byu.edu Where:**

**-v runs CeWL in verbose mode.**

**-d is the depth to “spider” or crawl the website**

**-m is the minimum word length**

**-w custom\_dict.txt is the name of your new custom wordlist or dictionary.**

**Step-2**

**Check how many entries are in the custom\_dict.txt file:**

**wc -l custom\_dict.txt**

**Output –**

root@kali:~# wc -l custom\_dict.txt

2065 custom\_dict.txt

**Step- 3**

**Permute the words in the custom\_dict.txt wordlist using the “best64” rule, and append the output to custom\_dict.txt (all one line):**

**hashcat custom\_dict.txt -r /usr/share/hashcat/rules/best64.rule --stdout >> custom\_dict.txt**

**Step -4**

**Check how many entries are in the custom\_dict.txt file now:**

**wc -l custom\_dict.txt**

**Output –**

root@kali:~# wc -l custom\_dict.txt

161070 custom\_dict.txt

**Step-5**

Run Hashcat using custom\_dict against the MD5 hash (all one line):

hashcat --force -a 0 -m 0 cf4aff530715824c055892438a1ab6b2 custom\_dict.txt

**Output –**

Dictionary cache built:

\* Filename..: custom\_dict.txt

\* Passwords.: 161070

\* Bytes.....: 1401798

\* Keyspace..: 161070

\* Runtime...: 0 secs

cf4aff530715824c055892438a1ab6b2:ytirucesoruen

Session..........: hashcat

Status...........: Cracked

Hash.Type........: MD5

Hash.Target......: cf4aff530715824c055892438a1ab6b2

Time.Started.....: Sun Feb 28 09:00:06 2021 (0 secs)

Time.Estimated...: Sun Feb 28 09:00:06 2021 (0 secs)

Guess.Base.......: File (custom\_dict.txt)

Guess.Queue......: 1/1 (100.00%)

Speed.#1.........: 30342 H/s (0.79ms) @ Accel:1024 Loops:1 Thr:1 Vec:8

Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts

Progress.........: 4096/161070 (2.54%)

Rejected.........: 0/4096 (0.00%)

Restore.Point....: 2048/161070 (1.27%)

Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1

Candidates.#1....: larger -> Earglee

Started: Sun Feb 28 08:59:50 2021

Stopped: Sun Feb 28 09:00:07 2021

**Question:** What is the plaintext of the hash?

**Ans:** :ytirucesoruen

**Progress Embedded Image of Progress Report from LabSim:**

