The World of Passwords

Trey Atwood

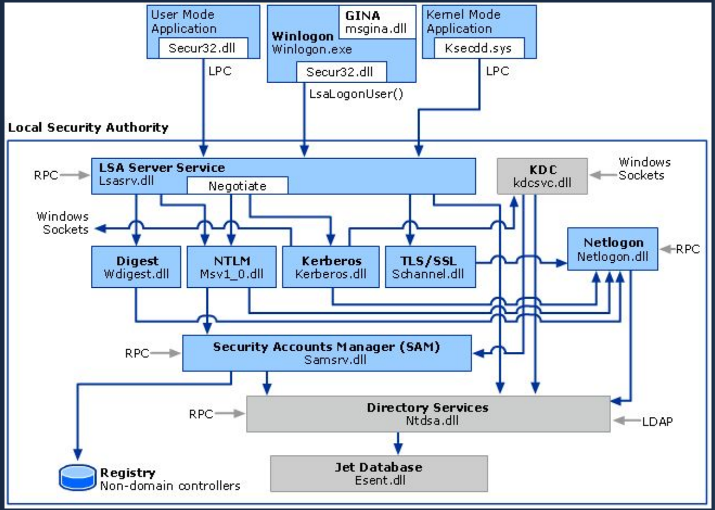
Password rules have become a normal standard part of life for most people, but few know the reasons why they need to have long, complex passwords and why their personal passwords shouldn't be reused on multiple different websites.

We are going to demonstrate how a hacker would intercept, locate, and crack a users password by eavesdropping on a shared network.

Doing this outside a simulation environment is illegal, and is a felony charge if caught

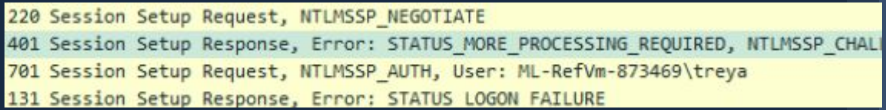
How does Windows Password authentication work?

Windows Password authentication works by hashing every password before it is transmitted across the network. It takes local or domain account credentials with Windows security protocols and encryption to ensure that whoever is trying to login is actually who they're logging in as. No password is ever stored in plaintext. The cryptographic keys that windows uses are stored in a central location within the system to make it scalable and easily maintainable. When you authenticate a service or person, the goal is to verify that the credentials presented are authentic.

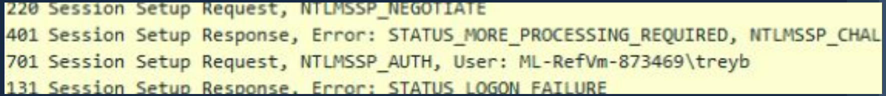


How do eavesdroppers capture data?

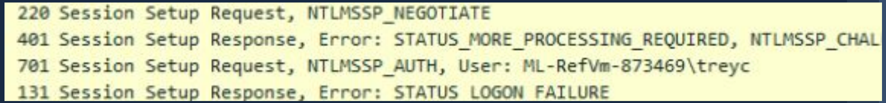
Eavesdroppers can capture data by physically intercepting the data streams that are sent across a network. Using tools like Wireshark you can gain access to a network to capture data that is sent across the network. Like we see in this exercise we capture data that is sent across a local network to “sniff” out the data that is sent in the packet. This is why public internet access isn't secure and is dangerous, people can be sniffing the network without you knowing and be able to capture packets that are being sent from your phone or laptop that could contain sensitive information that makes you vulnerable to someone stealing your password for example.



```
220 Session Setup Request, NTLMSSP_NEGOTIATE
401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHAL
701 Session Setup Request, NTLMSSP_AUTH, User: ML-RefVm-873469\treya
131 Session Setup Response, Error: STATUS_LOGON_FAILURE
```



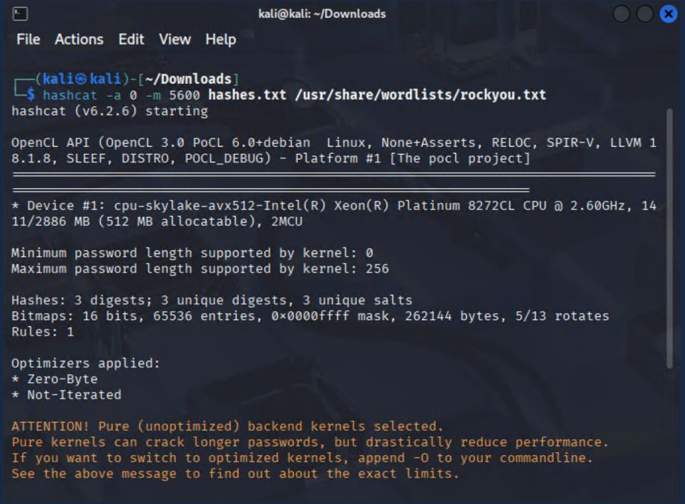
```
220 Session Setup Request, NTLMSSP_NEGOTIATE
401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHAL
701 Session Setup Request, NTLMSSP_AUTH, User: ML-RefVm-873469\treyb
131 Session Setup Response, Error: STATUS_LOGON_FAILURE
```



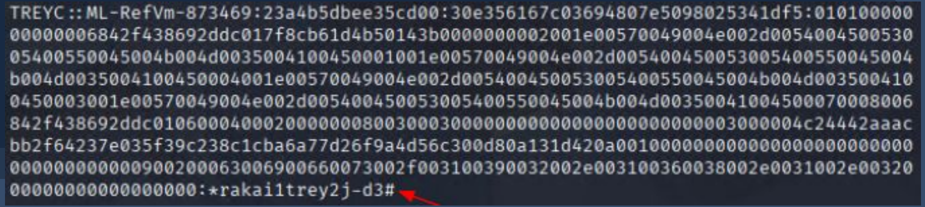
```
220 Session Setup Request, NTLMSSP_NEGOTIATE
401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHAL
701 Session Setup Request, NTLMSSP_AUTH, User: ML-RefVm-873469\treyc
131 Session Setup Response, Error: STATUS_LOGON_FAILURE
```

How do hackers crack passwords?

There are many tools available to crack a password, but hashcat is the most popular. Hashcat is popular because it uses the computers GPU mainly to crack a password, so the better GPU you have the faster the crack. Like we'll see in this exercise hashcat has different options to crack a password; searching a words list, hybrid-brute force, and brute force attacks. Hackers need the Username, Domain Name, NTLM Server Challenge, NTProofString, and the rest of the NTLMv2 response in order to crack a password. They will need to sniff out a network to get this information and you can use hashcat to use computing power to crack it and reveal the password.



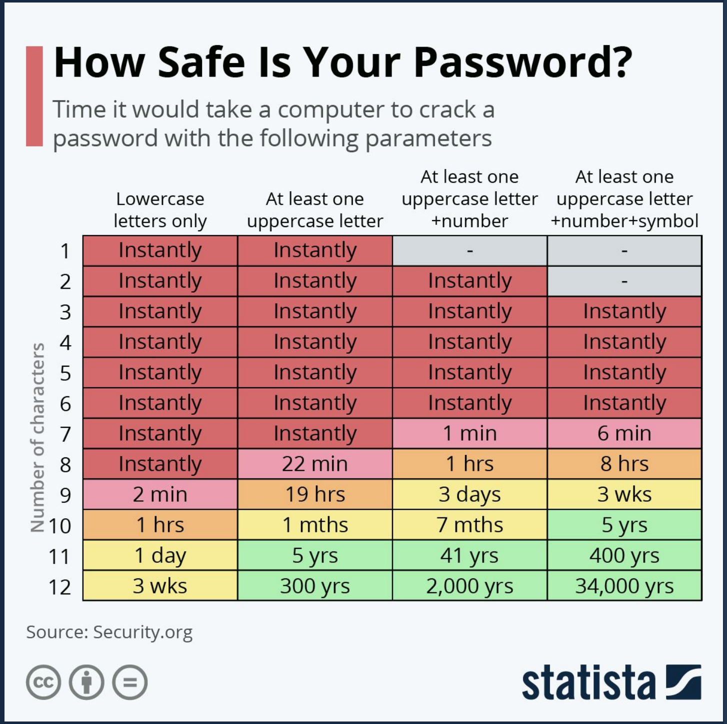
```
┌──(kali㉿kali)-[~/Downloads]
└─$ hashcat -a 0 -m 5600 hashes.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, SPIR-V, LLVM 1
8.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
==================================================================================
===================================================================
* Device #1: cpu-skylake-avx512-Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz, 14
11/2886 MB (512 MB allocatable), 2MCU

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

Hashes: 3 digests; 3 unique digests, 3 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Not-Iterated

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.
```



```
TREYC::ML-RefVm-873469:23a4b5dbee35cd00:30e356167c03694807e5098025341df5:010100000
00000006842f438692ddc017f8cb61d4b50143b0000000002001e00570049004e002d0054004500530
05400550045004b004d003500410045001001e00570049004e002d00540045005300540055004500 4
b004d003500410045004001e00570049004e002d00540045005300540055004500 4b004d003500410
0450003001e00570049004e002d00540045005300540055004500 4b004d0035004100450007000800 6
842f438692ddc01060004000200000008003000300000000000000000000000003000004c24442aaac
bb2f64237e035f39c238c1cba6a77d26f9a4d56c300d80a131d420a001000000000000000000000000
000000000000900200063006900660073002f003100390032002e003100360038002e0031002e00320
0000000000000000:*rakai1trey2j-d3#
```

My recommendations for password safety

My recommendations when it comes to password safety are as follows; make sure it's at least 12 characters, has a mix of lowercase and uppercase letters, numbers and at least one symbol. With this make sure that you aren't using words that are in the dictionary, or popular phrases. Forcing a hacker to have to brute force a password is what will deter them the most, they don't want to have to wait months for a password to finish cracking. They want instant access, the low hanging fruit whatever makes it easiest for them to get what they want is what they're going to do. Simple passwords that are all lowercase is what they want as that's the quickest to crack.



How Safe Is Your Password?

Time it would take a computer to crack a password with the following parameters

Number of characters	Lowercase letters only	At least one uppercase letter	At least one uppercase letter +number	At least one uppercase letter +number+symbol
1	Instantly	Instantly	-	-
2	Instantly	Instantly	Instantly	-
3	Instantly	Instantly	Instantly	Instantly
4	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	Instantly
7	Instantly	Instantly	1 min	6 min
8	Instantly	22 min	1 hrs	8 hrs
9	2 min	19 hrs	3 days	3 wks
10	1 hrs	1 mths	7 mths	5 yrs
11	1 day	5 yrs	41 yrs	400 yrs
12	3 wks	300 yrs	2,000 yrs	34,000 yrs

Source: Security.org

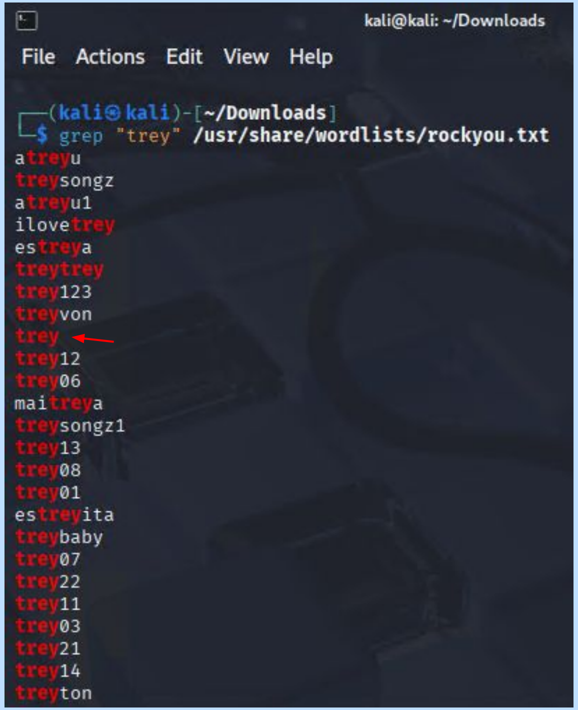
statista

In this exercise we are going to learn about passwords, we are going to simulate what it would be like to have your password caught in network traffic and getting your password compromised.

In order to start this, we will search the rockyou.txt list of compromised passwords for passwords to test this exercise on.

This is the command that we use to search the list for passwords that contain my name; “Trey”

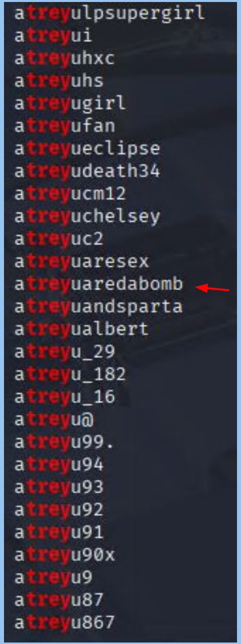
There are a roughly a thousand passwords here to choose from with my name, I'm going to find three passwords that are bad, good, and great. [none of these are good because they've been exposed already]



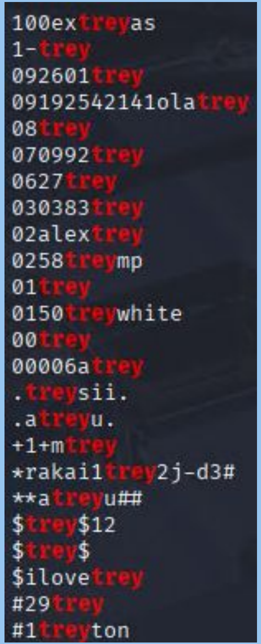
```
┌──(kali㉿kali)-[~/Downloads]
└─$ grep "trey" /usr/share/wordlists/rockyou.txt
atreyu
treysongz
atreyu1
ilovetrey
estreya
treytrey
trey123
treyvon
trey
trey12
trey06
maitreya
treysongz1
trey13
trey08
trey01
estreyita
treybaby
trey07
trey22
trey11
trey03
trey21
trey14
treyton
```

I'm going to create 3 fake users, treya, treyb, and treyc and tie 3 different passwords to each to later see if we can crack those passwords.

Treya - trey
Treyb - atreyuaredabomb
Treyc - *rakai1trey2j-d3#



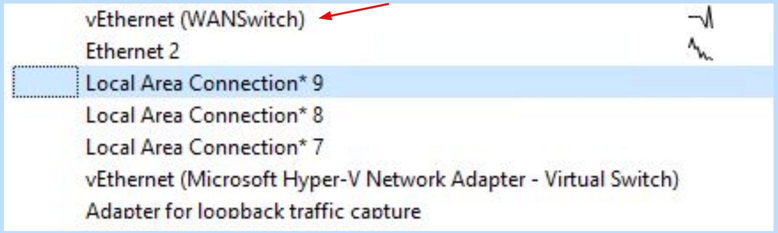
```
atreyulpsupergirl
atreyui
atreyuhxc
atreyuhs
atreyugirl
atreyufan
atreyueclipse
atreyudeath34
atreyucm12
atreyuchelsey
atreyuc2
atreyuaresex
atreyuaredabomb
atreyuandsparta
atreyualbert
atreyu_29
atreyu_182
atreyu_16
atreyu@
atreyu99.
atreyu94
atreyu93
atreyu92
atreyu91
atreyu90x
atreyu9
atreyu87
atreyu867
```



```
100extreyas
1-trey
092601trey
09192542141olatrey
08trey
070992trey
0627trey
030383trey
02alextrey
0258treymp
01trey
0150treywhite
00trey
00006atrey
.treysii.
.atreyu.
+1+mtrey
*rakai1trey2j-d3#
**atreyu##
$trey$12
$trey$
$ilovetrey
#29trey
#1treyton
```

To simplify the process of getting the passwords, we're going to use Wireshark and try to login to our shared folder that we created in our last exercise.

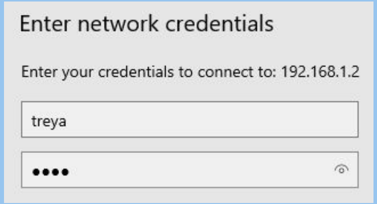
Logging into the shared folder across our WAN that we have set up will send packets to Wireshark and we will be able to inspect them and get the hash values.



vEthernet (WANSwitch)
Ethernet 2
Local Area Connection* 9
Local Area Connection* 8
Local Area Connection* 7
vEthernet (Microsoft Hyper-V Network Adapter - Virtual Switch)
Adapter for loopback traffic capture

In Wireshark we select our [WANSwitch] to start sniffing this connection to see the packets

In order to send our passwords across the network, we'll connect to our shared folder from the last exercise.

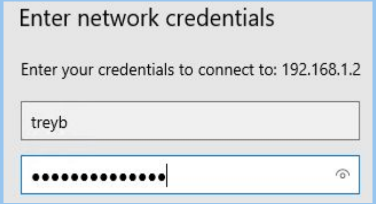


Enter network credentials

Enter your credentials to connect to: 192.168.1.2

treya

••••

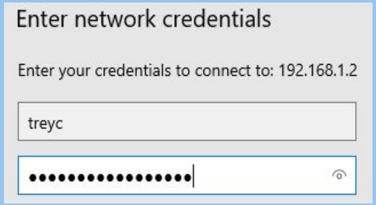


Enter network credentials

Enter your credentials to connect to: 192.168.1.2

treyb

••••••••••••••



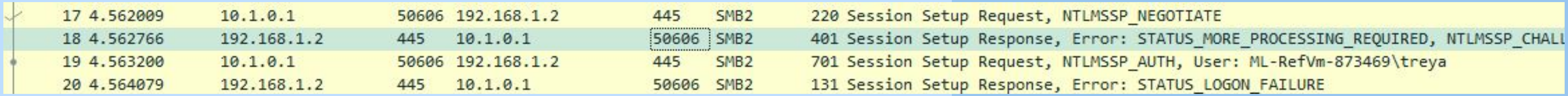
Enter network credentials

Enter your credentials to connect to: 192.168.1.2

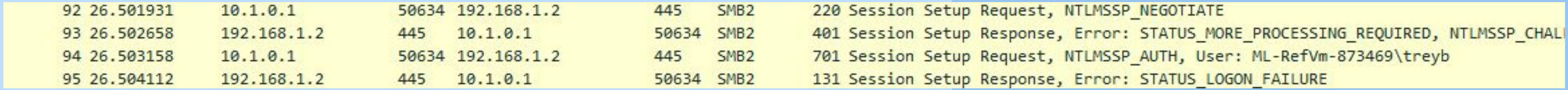
treyc

•••••••••••••••••

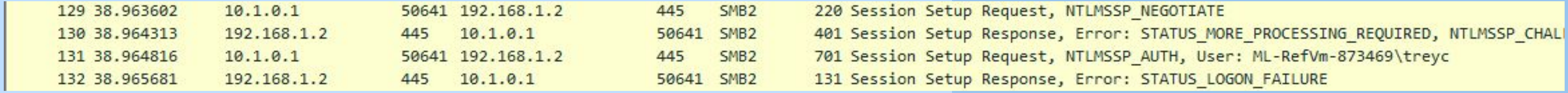
These logins are all going to fail, they aren't real accounts that we have setup. But attempting to submit them is going to send packets over our network that we can inspect with Wireshark. So let's do just that.



```
17 4.562009    10.1.0.1       50606 192.168.1.2    445    SMB2    220 Session Setup Request, NTLMSSP_NEGOTIATE
18 4.562766    192.168.1.2    445   10.1.0.1       50606  SMB2    401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALL
19 4.563200    10.1.0.1       50606 192.168.1.2    445    SMB2    701 Session Setup Request, NTLMSSP_AUTH, User: ML-RefVm-873469\treya
20 4.564079    192.168.1.2    445   10.1.0.1       50606  SMB2    131 Session Setup Response, Error: STATUS_LOGON_FAILURE
```



```
92 26.501931   10.1.0.1       50634 192.168.1.2    445    SMB2    220 Session Setup Request, NTLMSSP_NEGOTIATE
93 26.502658   192.168.1.2    445   10.1.0.1       50634  SMB2    401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHAL
94 26.503158   10.1.0.1       50634 192.168.1.2    445    SMB2    701 Session Setup Request, NTLMSSP_AUTH, User: ML-RefVm-873469\treyb
95 26.504112   192.168.1.2    445   10.1.0.1       50634  SMB2    131 Session Setup Response, Error: STATUS_LOGON_FAILURE
```

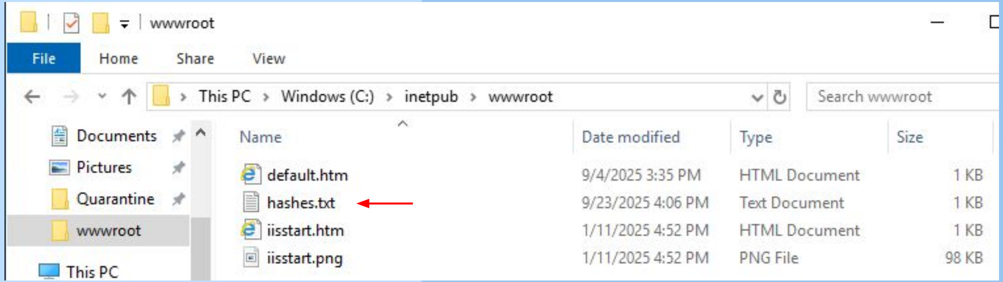


```
129 38.963602  10.1.0.1       50641 192.168.1.2    445    SMB2    220 Session Setup Request, NTLMSSP_NEGOTIATE
130 38.964313  192.168.1.2    445   10.1.0.1       50641  SMB2    401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHAL
131 38.964816  10.1.0.1       50641 192.168.1.2    445    SMB2    701 Session Setup Request, NTLMSSP_AUTH, User: ML-RefVm-873469\treyc
132 38.965681  192.168.1.2    445   10.1.0.1       50641  SMB2    131 Session Setup Response, Error: STATUS_LOGON_FAILURE
```

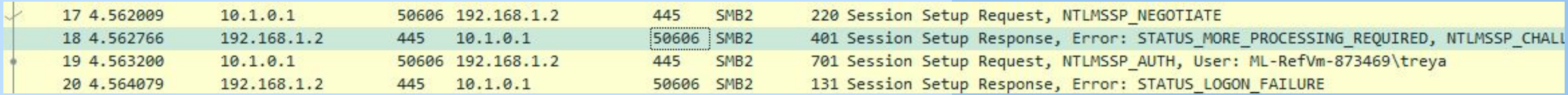
After attempting to login to my shared folder, with the three fake accounts, treya, treyb, and treyc we see these groups of packets that were sent across our network on Wireshark.

In order to keep track of all the information that we will get from these packets, we'll create a .txt file to keep all our information in.

I'm going to make it in the wwwroot folder that we have on our virtual machines.

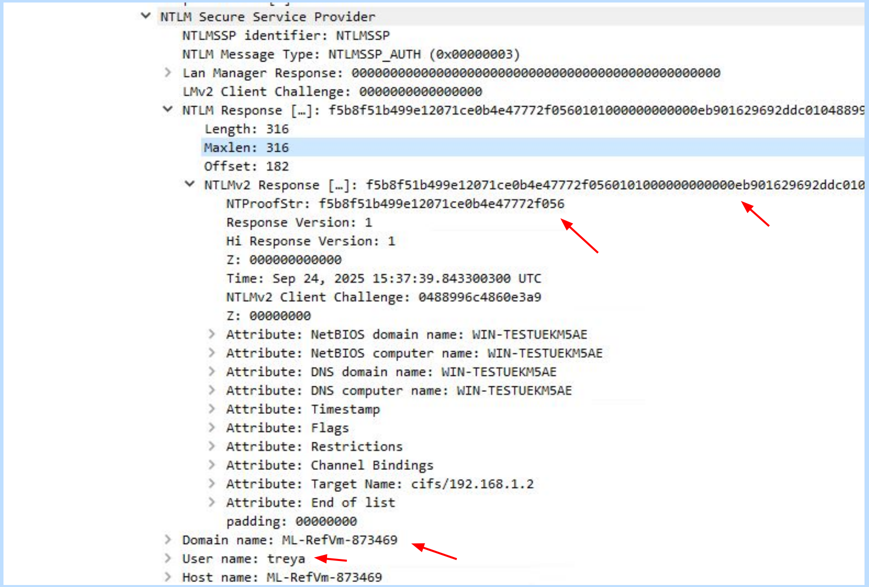


Name	Date modified	Type	Size
default.htm	9/4/2025 3:35 PM	HTML Document	1 KB
hashes.txt	9/23/2025 4:06 PM	Text Document	1 KB
iisstart.htm	1/11/2025 4:52 PM	HTML Document	1 KB
iisstart.png	1/11/2025 4:52 PM	PNG File	98 KB

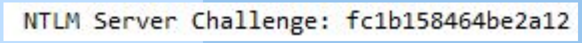


```
17 4.562009   10.1.0.1      50606 192.168.1.2   445    SMB2   220 Session Setup Request, NTLMSSP_NEGOTIATE
18 4.562766   192.168.1.2   445   10.1.0.1      50606  SMB2   401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALL
19 4.563200   10.1.0.1      50606 192.168.1.2   445    SMB2   701 Session Setup Request, NTLMSSP_AUTH, User: ML-RefVm-873469\treya
20 4.564079   192.168.1.2   445   10.1.0.1      50606  SMB2   131 Session Setup Response, Error: STATUS_LOGON_FAILURE
```

We'll start with the information from treya.
Starting with packet 19:



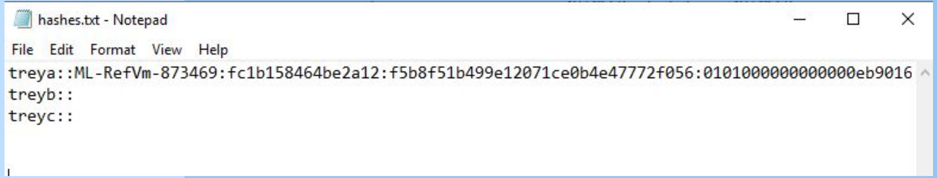
```
NTLM Secure Service Provider
  NTLMSSP identifier: NTLMSSP
  NTLM Message Type: NTLMSSP_AUTH (0x00000003)
> Lan Manager Response: 000000000000000000000000000000000000000000000000
  LMv2 Client Challenge: 0000000000000000
  NTLM Response […]: f5b8f51b499e12071ce0b4e47772f0560101000000000000eb901629692ddc01048899
    Length: 316
    Maxlen: 316
    Offset: 182
    NTLMv2 Response […]: f5b8f51b499e12071ce0b4e47772f0560101000000000000eb901629692ddc010
      NTProofStr: f5b8f51b499e12071ce0b4e47772f056
      Response Version: 1
      Hi Response Version: 1
      Z: 000000000000
      Time: Sep 24, 2025 15:37:39.843300300 UTC
      NTLMv2 Client Challenge: 0488996c4860e3a9
      Z: 00000000
    > Attribute: NetBIOS domain name: WIN-TESTUEKM5AE
    > Attribute: NetBIOS computer name: WIN-TESTUEKM5AE
    > Attribute: DNS domain name: WIN-TESTUEKM5AE
    > Attribute: DNS computer name: WIN-TESTUEKM5AE
    > Attribute: Timestamp
    > Attribute: Flags
    > Attribute: Restrictions
    > Attribute: Channel Bindings
    > Attribute: Target Name: cifs/192.168.1.2
    > Attribute: End of list
      padding: 00000000
> Domain name: ML-RefVm-873469
> User name: treya
> Host name: ML-RefVm-873469
```



```
NTLM Server Challenge: fc1b158464be2a12
```

This is the line that we needed from packet 18.

When we put this information into our .txt file there is a specific format that it must be put into.
[User Name]::[Domain Name]:[NTLM Server Challenge]:[NTProofString]:[Rest of the NTLMv2 Response]

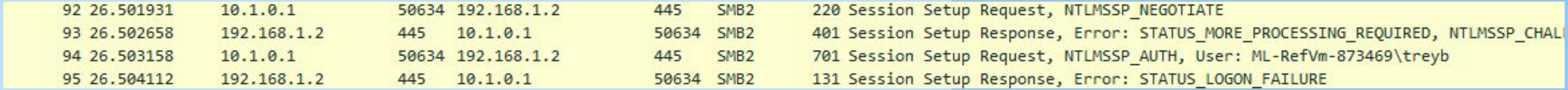


hashes.txt - Notepad

File Edit Format View Help

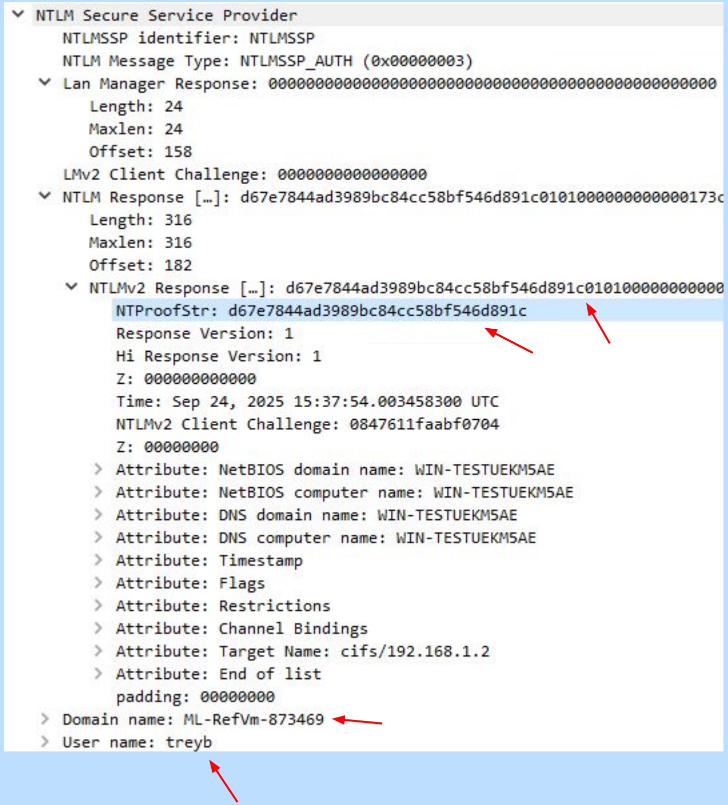
```
treya::ML-RefVm-873469:fc1b158464be2a12:f5b8f51b499e12071ce0b4e47772f056:0101000000000000eb9016
treyb::
treyc::
```

There is a lot of information here, and we need to pick parts of it out to put into a .txt file we created. We need the domain name, username, NTProofStr, and the NTLMv2 Response. We also need one more piece of information that isn't in this packet; we need the NTLM Server Challenge. To find this we will inspect packet 18 that ends with "… NTLMSSP_CHALLENGE".

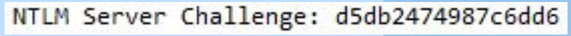


```
   92 26.501931     10.1.0.1           50634 192.168.1.2        445    SMB2       220 Session Setup Request, NTLMSSP_NEGOTIATE
   93 26.502658     192.168.1.2        445   10.1.0.1           50634  SMB2       401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHAL
   94 26.503158     10.1.0.1           50634 192.168.1.2        445    SMB2       701 Session Setup Request, NTLMSSP_AUTH, User: ML-RefVm-873469\treyb
   95 26.504112     192.168.1.2        445   10.1.0.1           50634  SMB2       131 Session Setup Response, Error: STATUS_LOGON_FAILURE
```

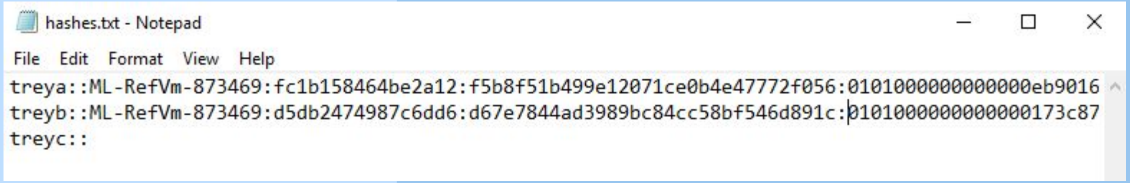
Next we will repeat the same process for treyb, starting with packet 94:



```
v NTLM Secure Service Provider
    NTLMSSP identifier: NTLMSSP
    NTLM Message Type: NTLMSSP_AUTH (0x00000003)
  v Lan Manager Response: 000000000000000000000000000000000000000000000000
      Length: 24
      Maxlen: 24
      Offset: 158
    LMv2 Client Challenge: 0000000000000000
  v NTLM Response […]: d67e7844ad3989bc84cc58bf546d891c0101000000000000173c
      Length: 316
      Maxlen: 316
      Offset: 182
    v NTLMv2 Response […]: d67e7844ad3989bc84cc58bf546d891c0101000000000000
        NTProofStr: d67e7844ad3989bc84cc58bf546d891c
        Response Version: 1
        Hi Response Version: 1
        Z: 000000000000
        Time: Sep 24, 2025 15:37:54.003458300 UTC
        NTLMv2 Client Challenge: 0847611faabf0704
        Z: 00000000
      > Attribute: NetBIOS domain name: WIN-TESTUEKM5AE
      > Attribute: NetBIOS computer name: WIN-TESTUEKM5AE
      > Attribute: DNS domain name: WIN-TESTUEKM5AE
      > Attribute: DNS computer name: WIN-TESTUEKM5AE
      > Attribute: Timestamp
      > Attribute: Flags
      > Attribute: Restrictions
      > Attribute: Channel Bindings
      > Attribute: Target Name: cifs/192.168.1.2
      > Attribute: End of list
        padding: 00000000
  > Domain name: ML-RefVm-873469
  > User name: treyb
```



```
NTLM Server Challenge: d5db2474987c6dd6
```

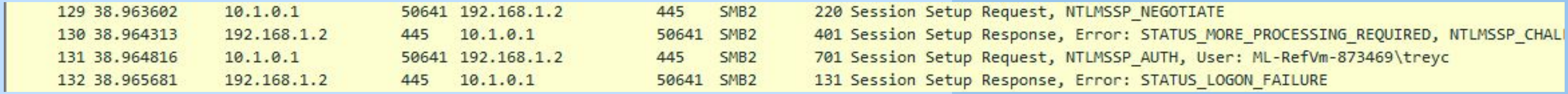


hashes.txt - Notepad

```
treya::ML-RefVm-873469:fc1b158464be2a12:f5b8f51b499e12071ce0b4e47772f056:0101000000000000eb9016
treyb::ML-RefVm-873469:d5db2474987c6dd6:d67e7844ad3989bc84cc58bf546d891c:0101000000000000173c87
treyc::
```

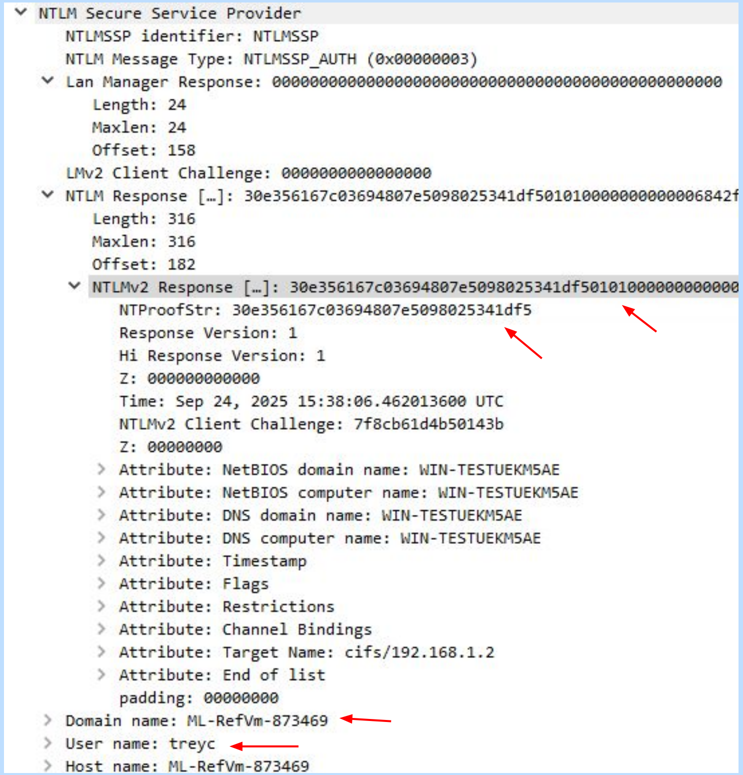
Here we have all our data from treyb that was captured and correctly formatted in into our .txt file.

Now we just need to repeat this process one more time for treyc. And then we can switch back over to our Kali Linux and try to crack these hashes with hashcat.

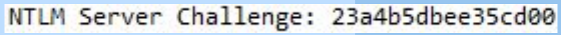


```
129 38.963602    10.1.0.1       50641 192.168.1.2      445    SMB2     220 Session Setup Request, NTLMSSP_NEGOTIATE
130 38.964313    192.168.1.2    445   10.1.0.1         50641  SMB2     401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHAL
131 38.964816    10.1.0.1       50641 192.168.1.2      445    SMB2     701 Session Setup Request, NTLMSSP_AUTH, User: ML-RefVm-873469\treyc
132 38.965681    192.168.1.2    445   10.1.0.1         50641  SMB2     131 Session Setup Response, Error: STATUS_LOGON_FAILURE
```

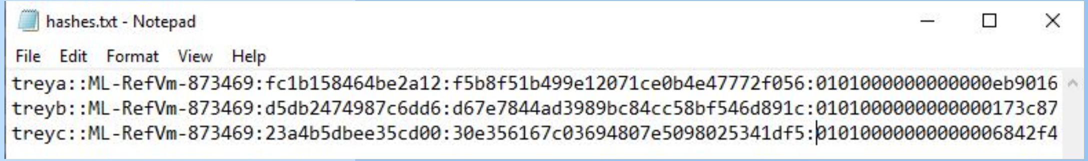
Lastly, we have the packets for the attempted login for treyc.
Let's start with packet 131:



```
v NTLM Secure Service Provider
    NTLMSSP identifier: NTLMSSP
    NTLM Message Type: NTLMSSP_AUTH (0x00000003)
  v Lan Manager Response: 000000000000000000000000000000000000000000000000
      Length: 24
      Maxlen: 24
      Offset: 158
    LMv2 Client Challenge: 0000000000000000
  v NTLM Response […]: 30e356167c03694807e5098025341df50101000000000000006842f
      Length: 316
      Maxlen: 316
      Offset: 182
    v NTLMv2 Response […]: 30e356167c03694807e5098025341df5010100000000000000
        NTProofStr: 30e356167c03694807e5098025341df5
        Response Version: 1
        Hi Response Version: 1
        Z: 000000000000
        Time: Sep 24, 2025 15:38:06.462013600 UTC
        NTLMv2 Client Challenge: 7f8cb61d4b50143b
        Z: 00000000
      > Attribute: NetBIOS domain name: WIN-TESTUEKM5AE
      > Attribute: NetBIOS computer name: WIN-TESTUEKM5AE
      > Attribute: DNS domain name: WIN-TESTUEKM5AE
      > Attribute: DNS computer name: WIN-TESTUEKM5AE
      > Attribute: Timestamp
      > Attribute: Flags
      > Attribute: Restrictions
      > Attribute: Channel Bindings
      > Attribute: Target Name: cifs/192.168.1.2
      > Attribute: End of list
        padding: 00000000
  > Domain name: ML-RefVm-873469
  > User name: treyc
  > Host name: ML-RefVm-873469
```



```
NTLM Server Challenge: 23a4b5dbee35cd00
```

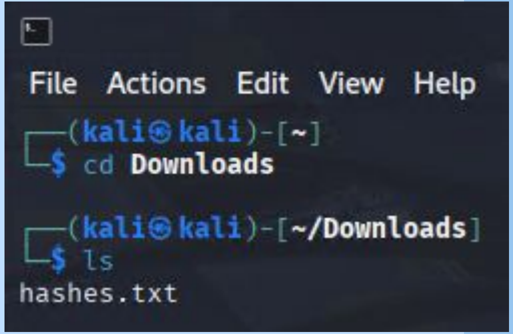


```
treya::ML-RefVm-873469:fc1b158464be2a12:f5b8f51b499e12071ce0b4e47772f056:0101000000000000eb9016
treyb::ML-RefVm-873469:d5db2474987c6dd6:d67e7844ad3989bc84cc58bf546d891c:0101000000000000173c87
treyc::ML-RefVm-873469:23a4b5dbee35cd00:30e356167c03694807e5098025341df5:0101000000000000006842f4
```

Now we have all of our necessary information that we will need to move forward and try to crack the hashes using hashcat.

Now that we have collected all the information we need for the next step, let's get started with hashcat.

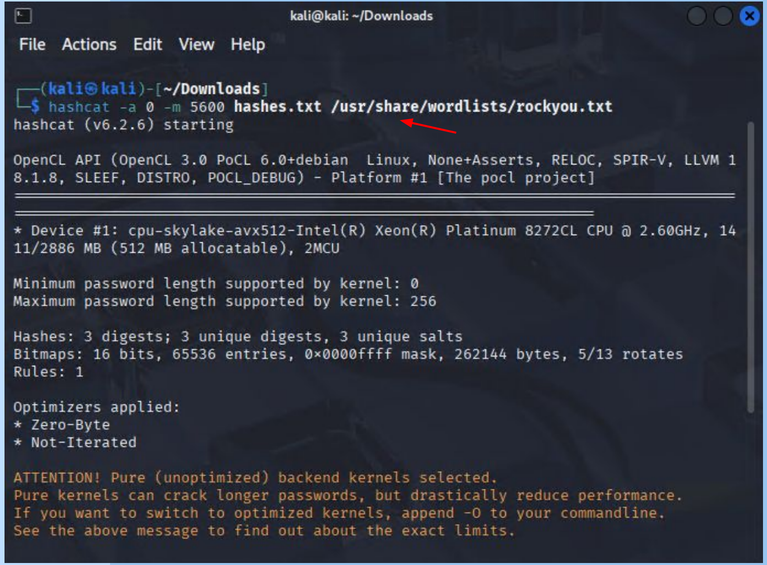
Inside of Kali we need to access the hashes.txt file that we created with our information from Wireshark. Luckily for us we put this file in our wwwroot folder, which we can access on the browser. Going to 192.168.1.2/hashes.txt will bring up our file and we can download it from there. Inside of the terminal in Kali, we can see that our hashes.txt file is in the downloads folder, so we can go ahead and attempt to crack the hashes.



```
┌──(kali㉿kali)-[~]
└─$ cd Downloads

┌──(kali㉿kali)-[~/Downloads]
└─$ ls
hashes.txt
```

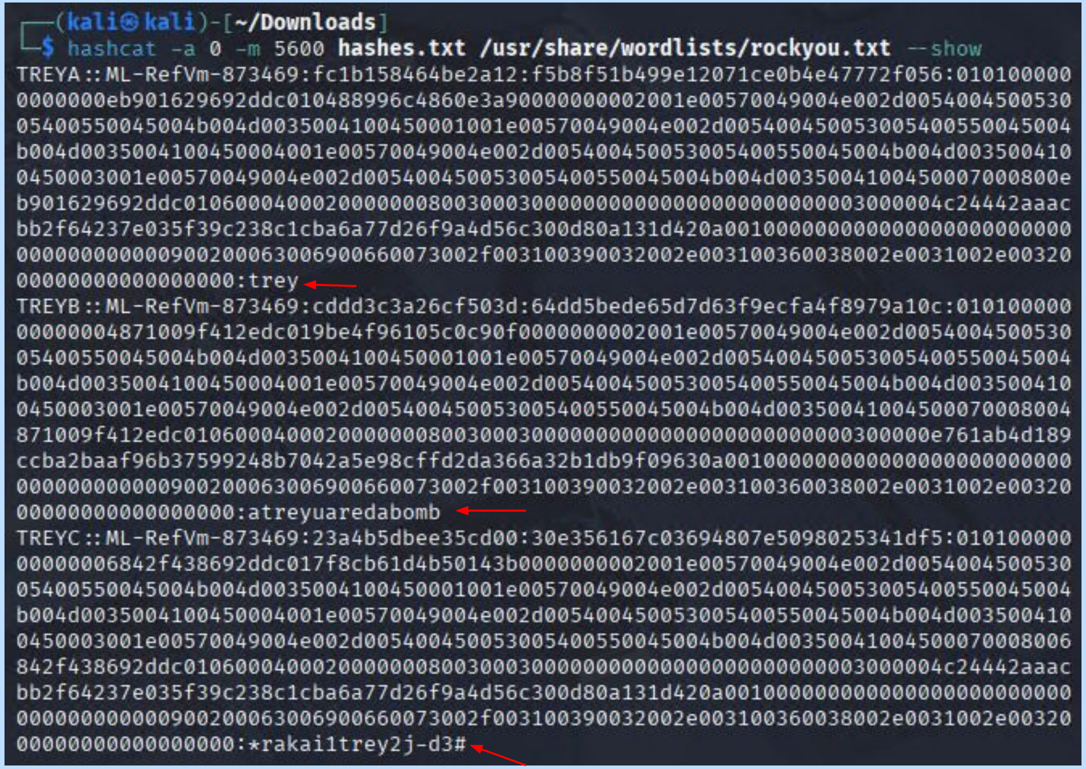
Inside of the Kali terminal we enter the command "hashcat -a 0 -m 5600 hashes.txt /usr/share/wordlists/rockyou.txt"

This command if you break it down has many different parts. "-a 0" the first part is the attack mode, the "0" is a straight attack, in this case using a words list. The next part "-m 5600" is the hash type, the "5600" is for NTLMv2 which is what our hashes are. Next is our filename "hashes.txt", lastly is the file path to find our rockyou.txt which is our words list that we are using to crack the hashes.



```
┌──(kali㉿kali)-[~/Downloads]
└─$ hashcat -a 0 -m 5600 hashes.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, SPIR-V, LLVM 1
8.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================
=================================================================

* Device #1: cpu-skylake-avx512-Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz, 14
11/2886 MB (512 MB allocatable), 2MCU

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

Hashes: 3 digests; 3 unique digests, 3 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Not-Iterated

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.
```

That command will take a while to finish, but once it does this is what we see. Each username with its respective password.



```
┌──(kali㉿kali)-[~/Downloads]
└─$ hashcat -a 0 -m 5600 hashes.txt /usr/share/wordlists/rockyou.txt --show
TREYA::ML-RefVm-873469:fc1b158464be2a12:f5b8f51b499e12071ce0b4e47772f056:010100000
0000000eb901629692ddc010488996c4860e3a90000000002001e00570049004e002d0054004500530
05400550045004b004d0035004100450001001e00570049004e002d00540045005300540055004500 4
b004d0035004100450004001e00570049004e002d0054004500530054005500450004b004d003500410
0450003001e00570049004e002d00540045005300540055004500 4b004d003500410045000700080 0e
b901629692ddc01060004000200000008003000300000000000000000000000003000004c24442aaac
bb2f64237e035f39c238c1cba6a77d26f9a4d56c300d80a131d420a001000000000000000000000000
00000000000090020006300690066007300 2f003100390032002e003100360038002e0031002e00320
000000000000000000:trey
TREYB::ML-RefVm-873469:cddd3c3a26cf503d:64dd5bede65d7d63f9ecfa4f8979a10c:010100000
00000004871009f412edc019be4f96105c0c90f0000000002001e00570049004e002d0054004500530
05400550045004b004d003500410045000100 1e00570049004e002d005400450053005400550045004
b004d0035004100450004001e00570049004e002d00540045005300540055004500 4b004d003500410
0450003001e00570049004e002d00540045005300540055004500 4b004d0035004100450007000800 4
871009f412edc01060004000200000008003000300000000000000000000000000300000e761ab4d189
ccba2baaf96b37599248b7042a5e98cffd2da366a32b1db9f09630a001000000000000000000000000
00000000000090020006300690066007300 2f003100390032002e003100360038002e0031002e00320
000000000000000000:atreyuaredabomb
TREYC::ML-RefVm-873469:23a4b5dbee35cd00:30e356167c03694807e5098025341df5:010100000
00000006842f438692ddc017f8cb61d4b50143b0000000002001e00570049004e002d0054004500530
05400550045004b004d003500410045000100 1e00570049004e002d005400450053005400550045004
b004d0035004100450004001e00570049004e002d00540045005300540055004500 4b004d003500410
0450003001e00570049004e002d00540045005300540055004500 4b004d0035004100450007000800 6
842f438692ddc01060004000200000008003000300000000000000000000000003000004c24442aaac
bb2f64237e035f39c238c1cba6a77d26f9a4d56c300d80a131d420a001000000000000000000000000
00000000000090020006300690066007300 2f003100390032002e003100360038002e0031002e00320
000000000000000000:*rakai1trey2j-d3#
```

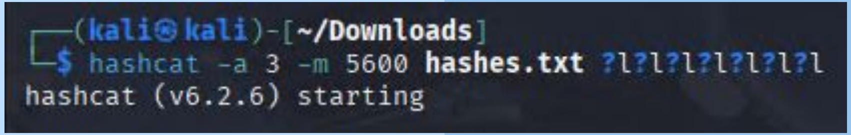
Regardless of the level of the password whether or not it's a "good" password, all 3 passwords showed up from the rockyou.txt list. Which we knew would happen since that's where we got these passwords from!

But this shows the danger of having dictionary words in passwords. There are different attacks like a hybrid-brute force attack where they could use a word from the dictionary and add commands like "?d", "?l", or "?u" for example. Let's say you want to hybrid-brute force a password with a common phrase of Appstate. The "?d", "?l", or "?u" you add to a command would add every combination of digits 0-9, letters a-z, and uppercase letters A-Z. Instead of a pure brute force attack where every character is tested at random there is patterns recognized like a common phrase then it's brute-forced from there.

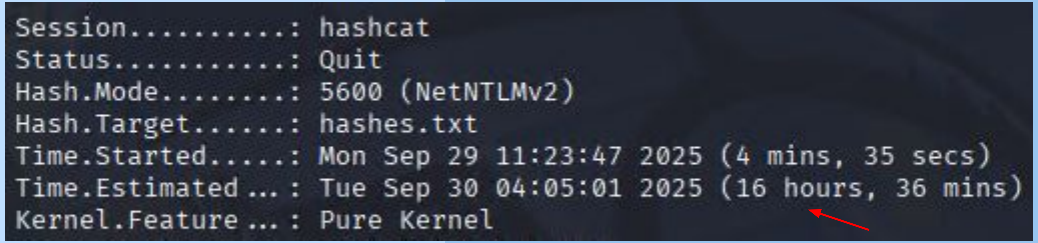
These three passwords, didn't take long to crack at all. We already knew they were in the words list so it would be quick. But other methods take much longer, brute force attacks take the longest, and hybrid-brute force are in between the two.

Now that we've cracked passwords from a word list, lets try cracking the same passwords but with different methods.

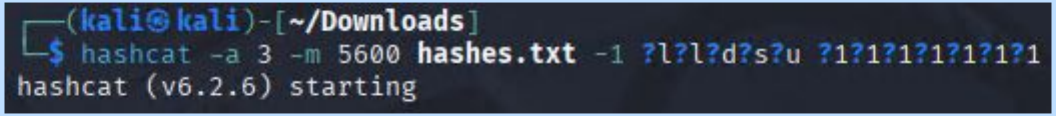
Brute force attacks take the longest to complete, since they test every possible character in every position of a password. For this example we're brute forcing a 7 letter all lowercase password. The length of a brute force attack depends on computing power, the stronger the GPU the faster it will go. Since we are doing this on a VM it's going to take a lot longer to brute force a password. Like we see here a 7 letter all lowercase password will take 16 hours, and 36 minutes to crack.



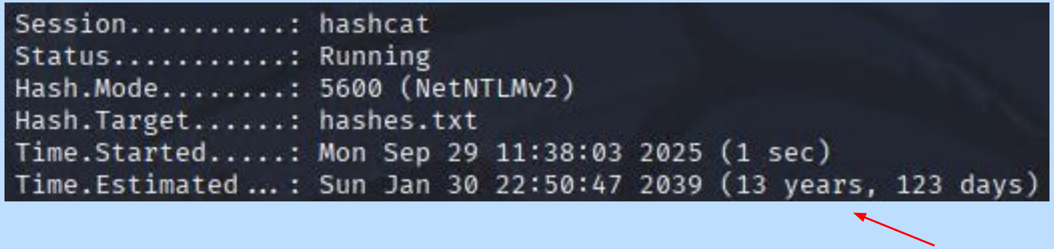
```
┌──(kali㉿kali)-[~/Downloads]
└─$ hashcat -a 3 -m 5600 hashes.txt ?l?l?l?l?l?l?l
hashcat (v6.2.6) starting
```



```
Session..........: hashcat
Status...........: Quit
Hash.Mode........: 5600 (NetNTLMv2)
Hash.Target......: hashes.txt
Time.Started.....: Mon Sep 29 11:23:47 2025 (4 mins, 35 secs)
Time.Estimated ...: Tue Sep 30 04:05:01 2025 (16 hours, 36 mins)
Kernel.Feature ...: Pure Kernel
```



```
┌──(kali㉿kali)-[~/Downloads]
└─$ hashcat -a 3 -m 5600 hashes.txt -1 ?l?l?d?s?u ?1?1?1?1?1?1?1
hashcat (v6.2.6) starting
```



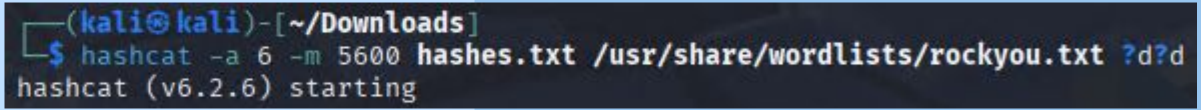
```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 5600 (NetNTLMv2)
Hash.Target......: hashes.txt
Time.Started.....: Mon Sep 29 11:38:03 2025 (1 sec)
Time.Estimated ...: Sun Jan 30 22:50:47 2039 (13 years, 123 days)
```

Here we are trying another brute force attack, but this time we are doing it with lowercase, uppercase, numbers, and special characters.

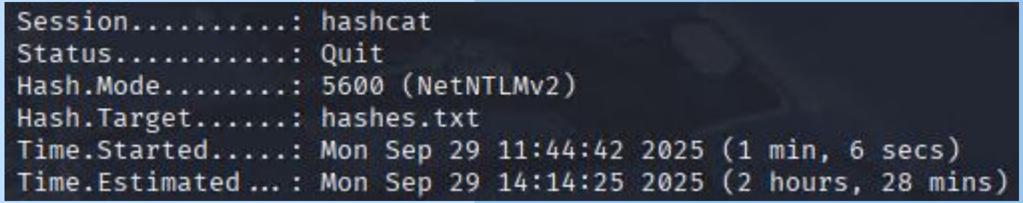
In this command we see it's formatted a little differently. Here we are using "1" to define a custom charset, that charset being all lowercase, uppercase, digits, and special characters. The "?1" at the end is telling hashcat that we want to crack passwords that are 7 digits in length with our custom charset. Like I mentioned before we're using a VM to crack these passwords, so it's going to take longer than normal but here we see it would take 13 years, and 123 days to finish

Now lets see how long it would take to do a hybrid-brute force attack.

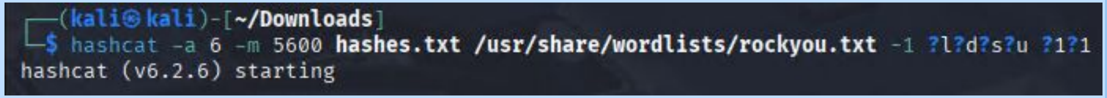
Here the command that we use searches the wordlist rockyou.txt and brute forces two digit spaces after the "?d" command. Since it's able to use a word list it's much faster than a normal brute force because it's only brute forcing two digit characters. As you can see it will only take 2 hours, and 28 minutes to crack the passwords in hashes.txt.



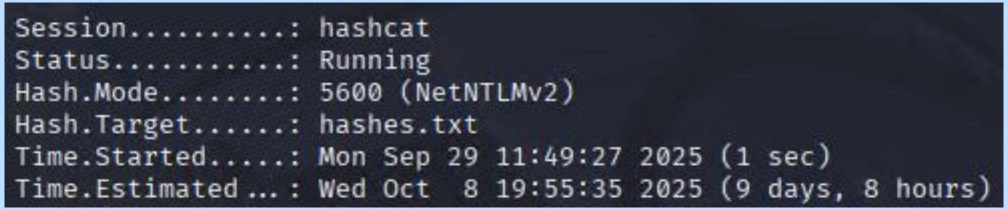
```
┌──(kali㉿kali)-[~/Downloads]
└─$ hashcat -a 6 -m 5600 hashes.txt /usr/share/wordlists/rockyou.txt ?d?d
hashcat (v6.2.6) starting
```



```
Session..........: hashcat
Status...........: Quit
Hash.Mode........: 5600 (NetNTLMv2)
Hash.Target......: hashes.txt
Time.Started.....: Mon Sep 29 11:44:42 2025 (1 min, 6 secs)
Time.Estimated ...: Mon Sep 29 14:14:25 2025 (2 hours, 28 mins)
```



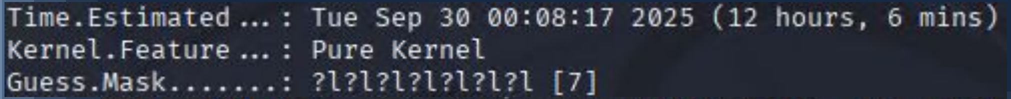
```
┌──(kali㉿kali)-[~/Downloads]
└─$ hashcat -a 6 -m 5600 hashes.txt /usr/share/wordlists/rockyou.txt -1 ?l?d?s?u ?1?1
hashcat (v6.2.6) starting
```



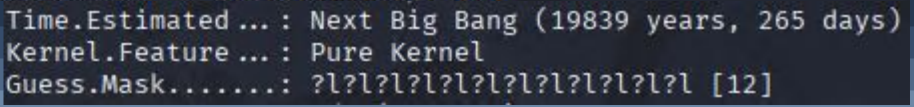
```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 5600 (NetNTLMv2)
Hash.Target......: hashes.txt
Time.Started.....: Mon Sep 29 11:49:27 2025 (1 sec)
Time.Estimated ...: Wed Oct  8 19:55:35 2025 (9 days, 8 hours)
```

Here we're just doing a more complex crack. Instead of just doing it for two digits, I created a custom charset for "1" like we did for the brute force attack. I only asked to crack 2 characters after the word in the words list so it isn't going to take too long, but just from this you can see how much longer it would take to crack it vs just wanting to try to crack a digit. This time it would take 9 days, and 8 hours to finish. Now this is a lot quicker than a brute force attack, but this is also only for 2 extra digits, so this time would increase if we were to do more.

Password security is VITAL for companies, not just for one person. If one person gets their password compromised the companies system as a whole are vulnerable. Someone being able to login to the systems as that user is dangerous depending on what permissions that user has within a system the attack can be crippling. In this exercise we went through a couple of different ways to crack a password using hashcat, and the dangers that come with it. Being on the other side of being the person cracking the password really shows you how easy it COULD be. I say could because there are great ways to make it harder. Simply just increasing the length of your password from 7 to 12 lowercase characters increases the length to brute force your password from 12 hours and 4 minutes to 19,839 years and 265 days or the next big bang. This number is greatly inflated since we're using a VM's computer power, and would be greatly reduced using a system that is made for this. But nonetheless it proves my point, simply just increasing the length of a password makes it more difficult.



```
Time.Estimated...: Tue Sep 30 00:08:17 2025 (12 hours, 6 mins)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?l?l?l?l?l?l?l [7]
```



```
Time.Estimated...: Next Big Bang (19839 years, 265 days)
Kernel.Feature...: Pure Kernel
Guess.Mask.......: ?l?l?l?l?l?l?l?l?l?l?l?l [12]
```

Assuming that you aren't using words that are in the dictionary, or commonly used phrases the hacker would have to brute force your password. There are many different words list .txt files out there that have millions of passwords on it to reference to speed up the cracking process for the hacker. Using random characters will increase the time even more. Having a 14 character long password with symbols, lowercase, uppercase, and digits in it would take an unfathomable amount of time to crack it. Simply increasing the length and complexity of a password makes it that much harder to crack. Having a password like that is annoying for the user to have to remember and to type it in, it comes with problems for the user themself. But having a complex password that's at least 14 characters long will deter any hacker from wanting to crack a password because it's simply just not worth their time. Why would they want to waste years of their time to crack a password. They want the low hanging fruit, they want the easy stuff they don't want to work hard for a password. Simply increasing the length and complexity is enough.