New issue

Jump to bottom

Modify installation to account for "CLR strict security" in SQL Server 2017 #25

Open **SqlQuantumLeap** opened this issue on Oct 12, 2017 · 3 comments

SqlQuantumLeap commented on Oct 12, 2017 • edited

SQL Server 2017 introduces a new server-level option, `"CLR strict security"`, that prevents unsigned assemblies from being created, even `SAFE` assemblies. Because of this, certain adjustments need to be made to the deployment process.

This issue was initially noted on Stack Overflow: Unable to create the tSQLtCLR assembly in SQL Server 2017

Because there is an existing **.snk** file and (I assume) related **.pfx** file, the option with the least impact on the current signing key and deployment process would be to use a certificate to sign the empty assembly used to create the asymmetric key. The overall difference as of SQL Server 2017 is that you used to be able to load the `tSQLtCLR` assembly and then later, optionally load the `tSQLtExternalAccessKey` assembly so that you could set the `tSQLtCLR` assembly to `EXTERNAL_ACCESS` for those using the `NewConnection`. Now, you can't create *any* assemblies unless the asymmetric key is there first. The asymmetric key is no longer optional (well, unless the database is set to `TRUSTWORTHY ON` but that is *highly* discouraged.

1. Create a certificate. There are a few ways to do this. I prefer the MAKECERT utility, but you can also use PowerShell or even SQL Server (if you use SQL Server, you will need to BACKUP CERTIFICATE including the private key). In the end, the certificate will be in both places: file system and SQL Server. Once created, you shouldn't ever need to do this again. But it might be a good idea to add the **.cer** file to the repository.

2. Use the PVK2PFX utility to combine the **.cer** and **.pvk** files into a password-protected **.pfx** file. Perhaps call it "InstallationKey.pfx" to distinguish it from "SigningKey.pfx"? Once combined, you shouldn't ever need to do this again.

3. This new **.pfx** file will be used to sign the tSQLtExternalAccessKey **DLL** using the SignTool utility. This can be automated rather easily by adding it as a Post-Build Event (which will be stored in the **tSQLtExternalAccessKey.csproj** file). This only needs to be run if the **tSQLtExternalAccessKey.dll** assembly is ever re-compiled, and there generally is no need to ever do that. But since you can't control whether or not people using this project will re-compile it, it can't hurt to have this part automated.

- All 3 steps are executed in `[master]`
- If you need the installation to be compatible for SQL Server versions prior to 2012, then wrap steps 1 - 3 in an `IF` block based on SQL Server "major" version. The two `DROP` statements, 8 and 9, should be wrapped in an `IF EXISTS` (or similar) check so those don't need to check for the version (if those objects weren't created due to running on SQL Server 2008, for example, then they just won't do anything).
- A full explanation of this approach, along with a link to a working demo script, can be found at: SQLCLR vs. SQL Server 2017, Part 2: "CLR strict security" – Solution 1

i. `CREATE CERTIFICATE [InstallationKey]` from a `VARBINARY` literal of the **.cer** file. You can use or update one of your helper utilities to do this, or you can use the free Binary Formatter utility that I wrote to do this (and to make sure that the output works well in scripts and when posted online).
ii. `CREATE LOGIN [tmp]` from that certificate.
iii. `GRANT` that cert-based login the `UNSAFE ASSEMBLY` permission.
iv. `CREATE ASSEMBLY [tSQLtExternalAccessKey] ...;`
v. `CREATE ASYMMETRIC KEY [tSQLtExternalAccessKey] ...;`
vi. `CREATE LOGIN` from that asymmetric key.
vii. `GRANT` that key-based login the `UNSAFE ASSEMBLY` permission.
viii. `DROP LOGIN [tmp];`
ix. `DROP CERTIFICATE [InstallationKey];`
x. `DROP ASSEMBLY [tSQLtExternalAccessKey];`

All of this might seem complicated at first, but this is all just a one-time setup.

10

ConstantineK commented on Dec 19, 2017

As a workaround you can also temporarily turn off CLR strict security (though though the above post is the correct way to do things) and append this to your SetClrEnabled.sql

```
EXEC sp_configure 'show advanced options', 1
RECONFIGURE;
EXEC sp_configure 'clr strict security', 0;
RECONFIGURE;
```

36

srutzky commented on Sep 2, 2018

```
EXEC tSQLt.InstallExternalAccessKey;
EXEC master.sys.sp_executesql N'GRANT UNSAFE ASSEMBLY TO [tSQLtExternalAccessKey];';
EXEC sp_configure 'clr strict security', 1; RECONFIGURE;
```

3 2

sidiqahmed commented on Feb 19, 2020

Thanks ConstantineK

dkultasev mentioned this issue on Jul 7, 2020

Installation of tSQLT on SQL Server 2019 #61

Open

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Linked pull requests

Successfully merging a pull request may close this issue.

None yet

4 participants