عنوان: **امنیت در LINQ to SQL**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۸/۱۵ ۱۳:۰۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Security

سؤال: LINQ to SQL تا چه میزان در برابر حملات تزریق SQL امن است؟
جواب کوتاه: بسیار زیاد!

توضیحات:

```
string query = @"SELECT * FROM USER_PROFILE
              WHERE LOGIN_ID = '"+loginId+@"' AND PASSWORD = '"+password+@"'";
```

گاهی از اوقات هر چقدر هم در مورد خطرات کوئری‌هایی از نوع فوق مقاله نوشته شود کافی نیست و باز هم شاهد این نوع جمع زدن‌ها و نوشتن کوئری‌هایی به شدت آسیب پذیر در حالت استفاده از ADO.Net کلاسیک هستیم. مثال فوق یک نمونه کلاسیک از نمایش آسیب پذیری در مورد تزریق اس کیوال است. یا نمونه‌ی بسیار متداول دیگری از این دست که با ورودی خطرناک می‌تواند تا نمایش کلیه اطلاعات تمامی جداول موجود هم پیش برود:

```
protected void btnSearch_Click(object sender, EventArgs e)
{
 String cmd = @"SELECT [CustomerID], [CompanyName], [ContactName]
  FROM [Customers] WHERE CompanyName ='" + txtCompanyName.Text
  + @"'";

 SqlDataSource1.SelectCommand = cmd;

 GridView1.Visible = true;
}
```

در اینجا فقط کافی است مهاجم با تزریق عبارت SQL مورد نظر خود، کوئری اولیه را کاملا غیرمعتبر کرده و از یک جدول دیگر در سیستم کوئری تهیه کند!
راه حلی که برای مقابله با آن در دات نت ارائه شده نوشتن کوئری‌های پارامتری است و در این حالت کار encoding اطلاعات ورودی به صورت خودکار توسط فریم ورک مورد استفاده انجام خواهد شد؛ همچنین برای مثال اس کیوال سرور، execution plan این نوع کوئری‌های پارامتری را همانند رویه‌های ذخیره شده، کش کرده و در دفعات آتی فراخوانی آن‌ها به شدت سریعتر عمل خواهد کرد. برای مثال:

```
SqlCommand cmd = new SqlCommand("SELECT UserID FROM Users WHERE UserName=@UserName AND
Password=@Password");
cmd.Parameters.Add(new SqlParameter("@UserName", System.Data.SqlDbType.NVarChar, 255, UserName));
cmd.Parameters.Add(new SqlParameter("@Password", System.Data.SqlDbType.NVarChar, 255, Password));
dr = cmd.ExecuteReader();
if (dr.Read()) userId = dr.GetInt32(dr.GetOrdinal("UserID"));
```

زمانیکه از کوئری پارامتری استفاده شود، مقدار پارامتر، هیچگاه فرصت و قدرت اجرا پیدا نمی‌کند. در این حالت صرفا به آن به عنوان یک مقدار معمولی نگاه خواهد شد و نه جزء قابل تغییر بدنه کوئری وارد شده که در حالت جمع زدن رشته‌ها همانند اولین کوئری معرفی شده، تا حد انحراف کوئری به یک کوئری دلخواه مهاجم قابل تغییر است.

اما در مورد LINQ to SQL چطور؟
این سیستم به صورت پیش فرض طوری طراحی شده است که تمام کوئری‌های SQL نهایی حاصل از کوئری‌های LINQ نوشته شده توسط آن، پارامتری هستند. به عبارت دیگر این سیستم به صورت پیش فرض برای افرادی که دارای حداقل اطلاعات امنیتی هستند به شدت امنیت بالایی را به همراه خواهد آورد.

برای مثال کوئری LINQ زیر را در نظر بگیرید:

```
var products = from p in db.products
      where p.description.StartsWith(_txtSearch.Text)
      select new
      {
          p.description,
          p.price,
          p.stock

      };
```

اکنون فرض کنید کاربر به دنبال کلمه sony باشد، آنچه که بر روی اس کیوال سرور اجرا خواهد شد، دستور زیر است (ترجمه نهایی کوئری فوق به زبان T-SQL) :

```
exec sp_executesql N'SELECT [t0].[description], [t0].[price], [t0].[stock]
FROM [dbo].[products] AS [t0]
WHERE [t0].[description] LIKE @p0',N'@p0 varchar(5)',@p0='sony%'
```

برای لاگ کردن این عبارات SQL یا می‌توان از SQL profiler استفاده نمود و یا خاصیت log زمینه مورد استفاده را باید مقدار دهی کرد:

```
db.Log = Console.Out;
```

و یا می‌توان بر روی کوئری مورد نظر در VS.Net یک break point قرار داد و سپس از debug visualizer مخصوص آن استفاده نمود.

همانطور که ملاحظه می‌کنید، کوئری نهایی تولید شده پارامتری است و در صورت ورود اطلاعات خطرناک در پارامتر p0 ، هیچ اتفاق خاصی نخواهد افتاد و صرفا رکوردی بازگشت داده نمی‌شود.

و یا همان مثال کلاسیک اعتبار سنجی کاربر را در نظر بگیرید:

```
public bool Validate(string loginId, string password)
{
 DataClassesDataContext db = new DataClassesDataContext();

 var validUsers = from user in db.USER_PROFILEs
           where user.LOGIN_ID == loginId
           && user.PASSWORD == password
           select user;

 if (validUsers.Count() > 0) return true;
 else return false;
}
```

کوئری نهایی T-SQL تولید شده توسط این ORM از کوئری LINQ فوق به شکل زیر است:

```
SELECT [t0].[LOGIN_ID], [t0].[PASSWORD]
FROM [dbo].[USER_PROFILE] AS [t0]
WHERE ([t0].[LOGIN_ID] = @p0) AND ([t0].[PASSWORD] = @p1)
```

و این کوئری پارامتری نیز در برابر حملات تزریق اس کیوال امن است.

تذکر مهم هنگام استفاده از سیستم LINQ to SQL :

اگر با استفاده از LINQ to SQL مجددا به روش قدیمی اجرای مستقیم کوئری‌های SQL خود همانند مثال زیر روی بیاورید (این امکان نیز وجود دارد)، نتیجه این نوع کوئری‌های حاصل از جمع زدن رشته‌ها، پارامتری "نبوده" و مستعد به تزریق اس کیوال هستند:

```
string sql = "select * from Trade where DealMember='" + this.txtParams.Text + "'";
var trades = driveHax.ExecuteQuery<Trade>(sql);
```

در اینجا باید در نظر داشت که اگر شخصی مجددا بخواهد از این نوع روش‌های کلاسیک استفاده کند شاید همان ADO.Net کلاسیک برای او کافی باشد و نیازی به تحمیل سربار یک ORM را به سیستم نداشته باشد. در این حالت برنامه از type safety کوئری‌های LINQ نیز محروم شده و یک لایه بررسی مقادیر و پارامترها را توسط کامپایلر نیز از دست خواهد داد.

اما روش صحیحی نیز در مورد بکارگیری متد ExecuteQuery وجود دارد. استفاده از این متد به شکل زیر مشکل را حل خواهد کرد:

```
IEnumerable<Customer> results = db.ExecuteQuery<Customer>(
 "SELECT contactname FROM customers WHERE city = {0}", "Tehran");
```

در این حالت، پارامترهای بکارگرفته شده (همان {0} ذکر شده در کوئری) به صورت خودکار به پارامترهای T-SQL ترجمه خواهند شد و مشکل تزریق اس کیوال برطرف خواهد شد (به عبارت دیگر استفاده از +، علامت مستعد بودن به تزریق اس کیوال است و بر عکس).