Whitepaper – FilterExpression Injection attacks against ASP.NET applications

Takeshi Terada / Mitsui Bussan Secure Directions, Inc.

March 2014

1. Introduction

FilterExpression is a SQL-like filter language built in ASP.NET framework. Like SQL, injection attacks are possible if an application utilizes FilterExpression in an improper manner, which can result in data leakages under certain situations. We call such vulnerability or attack “FilterExpression Injection”, and we’ll present its mechanism, impact, detection method and countermeasure in this paper.

From a viewpoint of pentesters, FilterExpression Injection is troublesome, because it looks almost the same as SQL Injection at a glance. We think that it is necessary for pentesters to understand the mechanism of FilterExpression Injection and to distinguish it properly from SQL Injection, in order to avoid wasting time for pentest and to evaluate risk of detected issues more accurately (usually the risk of FilterExpression Injection is lower than that of SQL Injection).

Experienced security folks may already understand what FilterExpression Injection is like, but we decided to publish this paper since there is little information about security of FilterExpression on the Web.

2. Usage of FilterExpression

FilterExpression is a filter language for filtering DataSource. DataSource is an object used in ASP.NET, which represents collection of data records extracted from data store such as SQL database, XML file, and so on.

The following is a sample of vulnerable program (and its explanation) using FilterExpression to filter data from SQL database.

```
[aspx]
<%-- Step1. Define DataSource --%>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:ntestdb1ConnectionString %>"
    SelectCommand="SELECT [id], [user_id], [title], [body], [create_date], [password], [status]
                   FROM [comment_tbl1]" />

<%-- Step2. Define GridView --%>
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="id" DataSourceID="SqlDataSource1" AllowPaging="True" Visible="False">
    <Columns>
        <asp:BoundField DataField="title" HeaderText="title" SortExpression="title" />
        <asp:BoundField DataField="body" HeaderText="body" SortExpression="body" />
        <asp:BoundField DataField="create_date" HeaderText="date" SortExpression="create_date" /
>    </Columns>
</asp:GridView>

<%-- Step3. Define textbox and button for search --%>
<asp:TextBox ID="txtSearchWord" runat="server" />
<asp:Button ID="btnFiltering" runat="server" OnClick="btnFiltering_Click" Text="Do Filtering" />
```

Step1. Define DataSource

DataSource (SqlDataSource1) is defined here. The application puts data, which is extracted from MS SQL Server database by the following SQL query, into DataSource.

```
SELECT id, user_id, title, body, create_date, password, status FROM comment_tbl1
```

In the later steps, FilterExpression is applied to this DataSource.

Step2. Define GridView

GridView is an ASP.NET UI component used to display values of DataSource in a HTML table. Here the application defines GridView (GridView1) with DataSource1 as its source of data. The GridView displays three items (title, body and create_data).

Step3. Define textbox and button for search

A textbox and a button used to narrow down DataSource are defined here. The event is fired when the button is clicked.

Step4. Click event for search button

Here, the application builds a FilterExpression string based on user-supplied string (txtSearchWord.Text) and session variable (Session["user_id"]). By applying the FilterExpression string to DataSource, filtered data is displayed in the GridView.

```
[C# source code]
public partial class GridViewTest1 : System.Web.UI.Page
{
    // Step4. Click event for search button
    protected void btnFiltering_Click(object sender, EventArgs e)
    {
        String expr = "user_id=" + Session["user_id"]
            + " AND title LIKE '%" + txtSearchWord.Text + "%'";
        SqlDataSource1.FilterExpression = expr;
        GridView1.Visible = true;
    }
}
```

3. Syntax of FilterExpression

In Step4 of above sample, the application builds FilterExpression as the following.

```
user_id=123 AND title LIKE '%abcd%'
```

As you can see, the syntax of FilterExpression is quite similar to that of SQL's WHERE clause. However, there are some differences between the two. The following is a list of FilterExpression's key characteristics.

- FilterExpression is executed not by database server but by ASP.NET.
- Compared to SQL, there are little functions available in FilterExpression.
  In FilterExpression, only CONVERT(), LEN(), ISNULL(), IIF(), TRIM(), SUBSTRING() and some aggregate functions are available.
- Like SQL, string literal is quoted by apostrophe (U+0027).
- Like SQL, you can use logical operators such as "AND" and "OR" etc.
- Like SQL, string concatenation operator is "+".
- Unlike SQL, You have no way to read the data that is not included in DataSource.

Details of FilterExpression's syntax are described in Microsoft's Web page [1].

4. Attack Examples

As explained in Section 3, attacker cannot read the data not included in DataSource through FilterExpression Injection. This is a truly big limitation of FilterExpression Injection attack, so that the risk of FilterExpression Injection is usually smaller than that of SQL Injection. However, in certain situations, FilterExpression Injection enables attackers to steal hidden

data in DataSource. There are two types of attack.

Type A. Steal hidden records

If the target application hides data records in DataSource by FilterExpression, injection attack can uncover such hidden records.

Input (txtSearchWord)	FilterExpression
`' OR 1=1 OR 'a'='`	`user_id=123 AND title LIKE '%' OR 1=1 OR 'a'='%'`

By manipulating input value as above, the condition "user_id=123" is invalidated, and all records in the DataSource are displayed in GridView.

Type B. Steal hidden items

Applications sometimes hide data items contained in DataSource.

In the sample code in Section 2, DataSource contains seven items listed below.

```
SELECT id, user_id, title, body, create_date, password, status FROM comment_tbl1
```

However, GridView displays only parts of these items (title, body, create_date).

```
<Columns>
    <asp:BoundField DataField="title" HeaderText="title" SortExpression="title" />
    <asp:BoundField DataField="body" HeaderText="body" SortExpression="body" />
    <asp:BoundField DataField="create_date" HeaderText="date" SortExpression="create_date" />
</Columns>
```

In such a situation, by manipulating input value as below, attackers can gain information about the items that are not visible in GridView.

Input (txtSearchWord)	FilterExpression
`abcd%' AND password LIKE 'a`	`user_id=123 AND title LIKE '%abcd%' AND password LIKE 'a%'`

As for type B attack, information on item names (such as "password") is required for successful attacks.

We have experienced both types of vulnerabilities several times in our past pentest works.

5. Identifying the Vulnerability

You can verify whether your target application has FilterExpression Injection flaw or not, by sending values such as ones in the following table.

#	Input (txtSearchWord)	FilterExpression	Result
1	abcd	… LIKE '%abcd%'	200 OK
2	abcd'	… LIKE '%abcd'%'	500 Error
3	abcd'+'	… LIKE '%abcd'+'%'	200 OK
4	abcd'+SUBSTRING('',1,1)+'	… LIKE '%abcd'+SUBSTRING('',1,1)+'%'	200 OK
5	abcd'+USSBRTNIG('',1,1)+'	… LIKE '%abcd'+USSBRTNIG('',1,1)+'%'	500 Error
6	abcd'+ltrim('')+'	… LIKE '%abcd'+ltrim('')+'%'	500 Error
7	abc'+CONVERT('d','System.String')+'	… LIKE '%abc'+CONVERT('d','System.String')+'%'	200 OK
8	abc'+CONVERT('d','yStsme.tSirgn')+'	… LIKE '%abc'+CONVERT('d','yStsme.tSirgn')+'%'	500 Error

The important parts are #6 to #8, because one cannot distinguish FilterExpression Injection from normal SQL Injection by #1 to #5 values. Additional values (#6 to #8) are required for verification.

In #7 value, CONVERT() function of FilterExpression is used. As you know, MS SQL Server also has a function with the same name, but the syntax is different from that of FilterExpression:

MS SQL Server: `CONVERT(nchar(1), 'd')`
FilterExpression: `CONVERT('d', 'System.String')`

You can use the difference to distinguish FilterExpression Injection from SQL Injection.

6. Countermeasures

If you are using FilterExpression for security purpose, protection measures against injection attacks should be taken. Roughly speaking, there are two types of measures against FilterExpression Injection.

Measure A. Output escaping / type conversion approach

If the application builds FilterExpression string in the application code (like sample code in Section 2), this approach is possible.

For string-typed values, escape special characters in the string. Just doubling apostrophe is enough (unfortunately it seems not to be mentioned in Microsoft documents).

If you put an input value into a pattern given to the LIKE operator, you may also escape wildcard characters “%”, “*” and “[“ to “[%]”, “[*]” and “[[]”. Take care as special characters are not the same as those of SQL LIKE operator. Regarding LIKE operator, it is known that wildcard DoS attack against MS SQL Server is possible [2]. As for FilterExpression, such attacks seem to be quite difficult, because LIKE operator of FilterExpression is, in its syntax, more restrictive than that of SQL.

If numeric input is expected, you should validate or convert the input value (string) to the expected type (int, double and so on).

Annoying thing is that such output escaping / type conversion approach cannot be simply applied, if your code uses FilterExpression’s placeholder in aspx file like the following.

```
<asp:SqlDataSource id="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:MyNorthwind%>"
    SelectCommand="SELECT EmployeeID,FirstName,LastName,Title FROM Employees"
    FilterExpression="FirstName LIKE '%{0}%'">
    <FilterParameters>
        <asp:ControlParameter Name="search" ControlId="txtSearch" PropertyName="Text"/>
    </FilterParameters>
</asp:SqlDataSource>
```

Unfortunately, ASP.NET does NOT automatically escape values to be embedded in placeholders of the FilterExpression attribute value. In this case, you have to implement escaping / type conversion in your own code during the Filtering event of DataSource. [3][4]

Measure B. Filter by SQL not by FilterExpression

You can replace FilterExpression filtering with SQL filtering (if the data comes from SQL Server), since FilterExpression language is roughly a subset of SQL. Filtering by SQL is a natural approach and may result in better performance. Obviously you have to pay attention to avoid SQL Injection vulnerabilities by using Parameterized Query or other means.

7. Our Test Environment

ASP.NET version 4.0.30319.18446 (Windows 7)

8. References

[1] http://msdn.microsoft.com/library/system.data.datacolumn.expression.aspx

[2] http://hax.tor.hu/read/MSSQL_DoS/wildcard_attacks.pdf

[3] http://msdn.microsoft.com/library/system.web.ui.webcontrols.sqldatasource.filterparameters.aspx

[4] http://msdn.microsoft.com/library/system.web.ui.webcontrols.objectdatasource.filtering(v=vs.100).aspx