SQL Injection Part III - Don't Forget Sorting

Posted At : July 21, 2008 12:53 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Security

So... you have diligently added CFQUERYPARAM to every input variable. Your database is secure and safe from SQL Injection - right? Well... maybe not. Did you remember to account for the ORDER BY Clause? Let me explain.

Let's suppose we have created a simple search and drill down. Searching gives us back a record set which we present in a tabular format - like so:

User	Email	City
Dick	archiesfriend@aol.com	Allentown
Bob	Bob@cubandisco.com	Springfield
Harry	Mommasboy@myspace.com	Toledo

This is ok, but sometimes when there are 60 or 70 results we wish to make them sortable. Perhaps we don't want to use client side sorting for this so we decide to simply re-run the query with a different sort order. We choose to do the following:

First, we change the header for each of the items to include a link to the results and a URL variable for order - something likes this.

```
<th>
<a href="rs.cfm?eml=#form.keyword#&orderby=U.User">User</a>
</th>
```

Next, in our query code we add something like this.

```
<cfparam name="url.orderby" default="C.city"/>
<cfquery ....>
    SELECT    U.User, U.Email
            C.City
    FROM    Users U JOIN contact C
            ON U.user_id = C.user_id
    WHERE    U.email LIKE
            (<cfqueryparam cfsqltype="CF_SQL_CHAR" value="%#form.eml#%"/>)
    ORDER BY #url.orderBy#
</cfquery>
```

Simple right? Actually, we have just opened up a gaping hole in our DB. Someone could easily pass something like:

```
?orderby=C.city; truncate table contact
```

We can't use CFQUERYPARAM on an ORDER BY clause - so how do we solve this problem (short of moving all of our ordering to the client side). Here are two possible approaches.

Fix 1 - Validate

The simplest way to do this is to simply make sure that URL.orderby matches a list of potential strings:

```
<cfparam name="url.orderby" default="C.city"/>
<cfset oList = "U.User,U.Email,C.City"/>
<cfquery ....>
    SELECT    U.User, U.Email
            C.City
    FROM    Users U JOIN contact C
            ON U.user_id = C.user_id
    WHERE    U.email LIKE
            (<cfqueryparam cfsqltype="CF_SQL_CHAR" value="%#form.eml#%"/>)
    <cfif listfindnocase(oList,url.orderby)>
        ORDER BY #url.orderBy#
    </cfif>
</cfquery>
```

This is fine if we are writing our application out of whole cloth - but what if we are repairing legacy code and we want to secure it as fast as possible? We don't have time to go and look up every column name in the DB and create lists. Here's a trick to try that serves as a nice work around.

Fix 2 - Q of a Q

The real danger of Injection is that a malicious user can change the contents of the DB. We can, however, use Q of a Q to take the process of ordering out of the picture altogether.

```
<cfparam name="url.orderby" default="city"/>
<cfquery name="test" datasource="blah">
    SELECT    U.User, U.Email
            C.City
    FROM    Users U JOIN contact C
            ON U.user_id = C.user_id
    WHERE    U.email LIKE
            (<cfqueryparam cfsqltype="CF_SQL_CHAR" value="%#form.eml#%"/>)
</cfquery>
<!--- now order it --->
<cfquery name="test" dbtype="query">
    SELECT    *
    FROM    test
    ORDER BY #Url.Orderby#
</cfquery>
```

See what I mean? No live data was harmed in the making of this ORDER BY clause.

Of course we will need good error trapping to insure that malicious attempts (which will now generate CF errors rather than JDBC errors) are caught.

One more note on this second approach. We might consider using it anyway if we are heavily caching data. If the sample DB query above is cached for, say an hour, and a user clicks on all three columns to re-order it using the *first* approach we will end up with three separate cached queries - identical save for ordering. If, however, we are using the second approach, we will have 1 cached query that is simply reordered in memory for each sort request. For some applications this is an excellent choice - especially if they are heavily used and DB resources are scarce.