

SQL Injection Attacks, Easy To Prevent, But Apparently Still Ignored

Posted At : December 21, 2005 4:27 PM | Posted By : Ben Forta

Related Categories: Tips (CF), ColdFusion

I was just on a web site (no, not a ColdFusion powered site, and no I will not name names) browsing for specific content. The URLs used typical name=value query string conventions, and so I changed the value to jump to the page I wanted. And I made a typo and added a character to the numeric value. The result? An invalid SQL error message.

That's bad. Very very bad. It means that I was able to create a SQL statement that was submitted to the database for processing, a SQL statement that was passed to the database as is, unchecked.

You'd think that by now we'd have learned to lock down our code so as to prevent SQL injection attacks, but apparently this is not the case. You do not know what a SQL injection attack is? Well, read on.

Consider the following simple dynamic ColdFusion query:

```
SELECT *
FROM Customers
WHERE CustID=#URL.custid#
```

Here a WHERE clause is being populated dynamically using a URL parameter. This type of code is common and popular, and is often used in data drill-down interfaces. If the URL was:

```
http://domain/path/file.cfm?custid=100
```

the resulting SQL statement would be:

```
SELECT *
FROM Customers
WHERE CustID=100
```

But what if someone tampered with that URL so that it read:

```
http://domain/path/file.cfm?custid=100;DELETE+Customers
```

Now the resulting SQL would be:

```
SELECT *
FROM Customers
WHERE CustID=100;
DELETE Customers
```

And depending on the DBMS being used, you could end up executing two statements - first the SELECT, and then DELETE Customers (which would promptly delete all data from the Customers table).

Scared? You should be. SQL statements are not just used for queries. They are also used by most DBMSs to create and drop tables, create user logins, change passwords, set security levels, manage scheduled events, even creating and dropping entire databases. And whatever features are supported by your DBMS may be accessible this way.

Before I go further I must point out that this is not a ColdFusion vulnerability at all. In fact, it is not even a bug or a hole. This is truly a feature - many DBMS do indeed allow queries to contain more than a single operation, this is legal and by design.

Of course, you should always be checking parameters anyway before passing them to your DBMS. Passing client supplied data (URL parameters, FORM fields, and even cookies) through unchecked is programmatic suicide. Attacks aside, it is flat out unsafe to ever assume that data submitted by a client can be used as is.

As such, you should already be using code like this:

```
<cfparam name="URL.CustID" type="integer">
```

This single line of code will lock SQL injection attacks out. How? Think about it, SQL injection (within ColdFusion apps) is really only an issue with non textual fields. If a text value is tampered with you'll end up with tampered text, but that text will all be part of the core string (within quotes) passed as a value, and will therefore not be executed as separate statements. Numbers, on the other hand, are not enclosed within quotes, and so extraneous text can be tampered with to create an additional SQL statement. And <cfparam> can protect you.

Of course, you may want more control, in which case you could use code like this:

```
<cfif IsDefined("URL.CustID")
and not IsNumeric(URL.CustID)>
... throw an error or something ...
</cfif>
```

And as an additional line of defense you can use <cfqueryparam>, as seen here:

```
<cfquery ...>
SELECT *
FROM Customers
WHERE CustID=<cfqueryparam value="#URL.CustID#" cfsqltype="CF_SQL_INTEGER">
</cfquery>
```

If the previous tampered URL was passed to the this query, the value would be rejected and an error would be thrown. The CFSQLTYPE (aside from binding variables) performs data type validation checks, and values that do not match the type are rejected. That's it, only integers allowed, and malicious tampered URL parameters are not integers.

The bottom line is that SQL injection attacks have been around for as long as dynamic SQL itself. ColdFusion has made it incredibly easy to protect yourself against such attacks. Be it <cfparam> or <cfqueryparam> or your own conditional processing, it's simple to protect yourself, and your responsibility to do so.

If you have not been paying attention to this risk, stop whatever you are doing, fire up your IDE, and do a search for every single <cfquery> in your code. Then quickly scan to find any that contain #'s in them (that are not enclosed in quotes or passed to <cfqueryparam>), and make a list of the variables used. If any of them are URL parameters or FORM fields, create a <cfparam> for each (at the top of the page, or before the <cfquery>). It's that simple. Really. There is no legitimate reason not to protect yourself, so just do it. Now! And I mean right now, before you leave for the day or take off for the holidays, and despite whatever project you are working on or deadline you are up against. No excuses (and if your boss complains about you switching gears to take care of this one, send him my way!).

Enough said! (I hope).

(UPDATED 07/24/2008)

Since this post was made, SQL injection attacks have evolved, and it is now known that even strings are vulnerable. See the more current related posts linked below.