Ask-a-Muse: How Can Cfqueryparam Protect Me?

Posted At : July 28, 2008 7:19 PM | Posted By : Mark Kruger
Related Categories: Coldfusion Security

> *Muse Reader Asks:*
> If you want to allow someone to search your site by keyword, how do you protect against an SQL injection? CFqueryParam is great if testing for an integer, but what about for a string? Surely there's got to be a way to do it since all kinds of sites let you perform keyword searches. Thanks!

Whoa... slow down there. Do my ears deceive me? Did my reader just indicate that he (or she) thinks that cfqueryparam "tests" for a string? I hate to break it to you, but the purpose of Cfqueryparam is *not* to insure that the value passed into the tag is one thing or another. The validation that occurs is more of a by-product of binding. Sure, the tag will error out when you try to pass "abc" instead of "123" to a param of the "integer" type, but that is a result of type binding. It's simply trying to bind variables of type for the driver to use, so naturally it errors out. But pass in a decimal like 123.123 and it says "okey dokey - that will work". Testing to see what a form element contains is the job of the developer, not the job of a magic box tag.

But to answer your question more specifically, cfqueryparam will protect you from those malicious hack attempts anyway - even if the attack is *passed to the database*. Let's examine a working case and see if we can figure out what is happening.

To illustrate we need an SQLi attack that is targeting a character field. Let's borrow one from my previous post on **SQLi Using a Character Field**. It looks like this:

```
<cfparam name="form.song"
    default="I can\'; drop table ignominiousHits -- t get no satisfaction"/>
<cfquery ...>
SELECT *
FROM IgnominiousHits
WHERE Title = '#form.song#'
</cfquery>
```

For those of you playing along at home, this attack will succeed if you are using MySQL with backslash escaping turned on (which is the default) and the ability to run multiple statements in a single query turned on. Running this code would result in the following query being run against the DB.

```
SELECT    *

FROM   IgnominiousHits

WHERE   Title = 'I can\''; drop table ignominiousHits -- t get no satisfaction
```

Coldfusion plays an unwitting role in this drama by escaping the single quote. Since the backslash escapes the first quote the single quote added by ColdFusion terminates the string. the semi-colon is honored as a statement terminator and the rest of the code is executed - viola! To put it another way, the statement above is the equivalent of the following:

```
SELECT     *
```

```
    FROM    IgnominiousHits

    WHERE    Title = 'I can''';


drop table ignominiousHits  -- t get no satisfaction
```

Obviously this is a *bad* thing. To prevent it, I have suggested here and many many (many) other places that each parameter being used in the query should be bound to a variable through the use of cfqueryparam like so:

```
<cfparam name="form.song"
    default="I can\'; drop table ignominiousHits -- t get no satisfaction"/>
<cfquery ...>
SELECT *
FROM IgnominiousHits
WHERE Title = <cfqueryparam cfsqltype="CF_SQL_CHAR" value="#form.song#"/>
</cfquery>
```

In the case of the readers question above, how does this help me? After all, it *does allow the malicious string to be passed to the database*... doesn't it? Why yes, yes it does. But think about what is really happening here. Let's pretend I'm the MySQL dispatcher and I'm going to spell out what I want the DB to do.

Dispatcher:	I have a new query for you.
DB Hamsters:	Dude, we are spinning this wheel as fast as we can but whatever.. .bring it on.
Dispatcher:	Ok... first, I want you to select all the columns from the IgnominiousHits table and...
DB Hamsters:	Hang on a minute... (all the columns - a plague on whoever thought up that stupid asterisk)... Let's see, system schema, check, ignoble, ignat (nope that's a view) .. ah here we are - ignominiousHits, all the columns got it. Ok what's next dispatch?
Dispatcher:	Right... ok, now we have a variable to work with of the type "character".
DB Hamsters:	Check... let's call it @param1 - ready to receive.
Dispatcher:	Ok... here's comes the character array... *I can\'; drop table ignominiousHits -- t get no satisfaction*
DB Hamsters:	Ok I got it and it's stored in my little @param1 box. What do we do with it.

Dispatcher:	Right.. ok, now compare the column *TITLE* to @param1 looking for exact matches.
DB Hamsters:	Got it, ok... executing now... sorry dispatch but that returns zero records. Tell your user to go back to the drawing board. Meanwhile I have to spit out these seeds - my cheeks are killing me.

The point is that the string being passed is being "boxed up" in a variable "of type char". The database will not do anything with it unless specifically instructed to evaluate it as if it were valid executable SQL. In short, any variety of malicious code passed into the query through the use of this variable *cannot succeed in injecting*.

To come full circle back to my user's question, cfqueryparam when used *will protect* your search from injection by "binding" the values passed as URL or form variables to a specific "type" known to the database. Even if you are under attack and malicious code is being presented to your binding, the attack cannot succeed. The variable cannot be anything other than the declared type (i.e. it cannot be executed), and therefore you are protected.

I hope this reassures you. Keep those questions coming.