A Better Blacklist Function for SQLi

Posted At : July 28, 2008 10:29 AM | Posted By : Mark Kruger
Related Categories: Coldfusion Security

Please note - I have not changed my stance on the use of CFQUERYPARAM. The real "fix" for injection is validation routines for form inputs and binding variables using Cfqueryparam. A blacklist function (a function that checks for "known bad" input) is useful in that it provides protection on the perimeter. It can help you intercept hack attempts before they reach your DB - where presumably they would fail in any case. They are also useful for thwarting immediate threats if you discover a security flaw that might take some time to fix. The recent spate of attacks caused a proliferation of blacklist techniques from simple to complex. In my own post on the vulnerability of using string concatenated SQL I published a snippet that made use of the iSQLInject function from CF Lib. There is a better approach however.

Now it turns out that Mary Jo Sminkey (creator of the popular CF Webstore) has a regular expression that she uses for this purpose. Gabriel Read (of Evolution 7) has added to it and I think the result is a better approach than the isSQLInject( ) function (which can generate false positives rather easily). The link to Mary Jo's Regex and function can be found here. In fact, Mary Jo has provided an include that you can just drop into your Application.cfm page (or you might need to fiddle with OnRequest() if you are using application.cfc). It will automatically examine the URL, Form, CGI and Cookie Scopes for you.

Mary Jo's function utilizes ColdFusion's "ReFindNoCase()" function against a complex regex pattern. Gabriel Read took it a step futher and is using a Java class - *java.util.regex.Pattern*. Here's his approach.

```
<cfscript>
// Short list of db objects to protect
DBObj.short = 'database|function|procedure|role|table|trigger|user|view';

// Sql Threat Indicators
blackList = '@@|' &
'(?:alter.*?(#DBObj.short#))|' &
'cast.*?\(|' &
'char.*?\([\w]{2}\)|' &
'(?:create.*?(#DBObj.short#))|' &
'(?:declare.*?@|cursor)|' &
'delete.*?from.*?_|' &
'(?:drop.*?(#DBObj.short#))|' &
'exec.*?\(|' &
'insert.*?values.*?\(+?|' &
'schema[^\w]+?|' &
'sysObjects|' &
'truncate.*?table|' &
'update.*?set+?|' &
'[sx]p_[\w_]+?|' &
'\''.*?-{2}|-{2}.*?\'''' &
'/\*.*?\*/';

// Build the java pattern matcher
rePattern = createObject('java', 'java.util.regex.Pattern');
rePattern = rePattern.compile(blackList);
reMatcher = rePattern.matcher('');
```

```
result = reMatcher.reset(lcase('truncate')).find();
</cfscript>
```

The "result" in the code of above returns either YES or NO. Why would you use this approach? Without testing I can't be sure, but I suspect that this approach might be beneficial in that this fairly large regex does not have to be compiled with each new check. Once the pattered has been added to the matcher object you can just use the reset().find() function as many times as needed for form, url, cgi and cookie variables - like so:

```
<cfloop collection="#url#" item="uItem">
    <cfif reMatcher.reset(lcase(url[uItem])).find()>
        <h4>Intruder... exterminate...exterminate....</h4>
        <cfabort>
    </cfif>
</cfloop>
```

You would want to test this of course, but it's possible that this approach would have less overhead than the one using ColdFusion's native ReFindNoCase. It could also be a negligible difference as well. I'm going to run a few tests myself and see what I can find out.