ColdFusion: Code Security Best Practices

Presented at CCFUG Mar 2016

By Denard Springle

Who Am I?

- Denard Springle
- CEO – Virtual Solutions Group LLC
- Over two decades of IT experience
- Developing in CFML since version 4
- Node.js, Python, jQuery, Bootstrap, etc.
- Lucee as primary CFML engine
- denard.springle@gmail.com
- @ddspringle (Twitter, Slack)
- blog.vsgcom.net

Presentation Outline

- Obfuscation
- Encryption
- Attack Vectors (XSS, CSRF, SQL Injection, etc.)
- Secure Authentication
- Two-Factor Authentication

Obfuscation

```
1  <!--- How you might normally populate a URL in your application --->
2  <cfset myURL = "myApp.cfm?uid=42" />
3  <a href="#myURL#">Normal URL</a>
4
5  <!--- How you can use hashing to obfuscate variable names in a URL --->
6  <cfset myURL = "myApp.cfm?v#hash( 'uid', 'SHA-384' )#=42" />
7  <a href="#myURL#">Hashed URL</a>
8
9  <!--- How you might normally populate a form field in your application --->
10 <input type="hidden" name="uid" value="42" />
11
12 <!--- How you can use hashing to obfuscate hidden field names --->
13 <input type="hidden" name="f#hash( 'uid', 'SHA-512' )#" value="42" />
14
15 <!--- How you might normally param a URL variable --->
16 <cfparam name="URL.uid" default="0" />
17
18 <!--- How you can param a hashed variable name --->
19 <cfparam name="URL[ 'v' & hash( 'uid', 'SHA-384' ) ]" default="0" />
20
21 <!--- How you might normally param a URL variable --->
22 <cfparam name="FORM.uid" default="0" />
23
24 <!--- How you can param a hashed variable name --->
25 <cfparam name="FORM[ 'f' & hash( 'uid', 'SHA-512' ) ]" default="0" />
```

Encryption Primer

- ColdFusion defaults to ECB (electronic code book) block cipher mode
- In ECB mode, the message is divided into blocks, and each block is encrypted separately. Can be decrypted in parallel.
- In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point.
- You *must* specify CBC mode by passing it as an additional option to ‘algorithm’

Encryption

```
<!--- How you might normally populate a URL in your application --->
<cfset myURL = "myApp.cfm?uid=42" />
<a href="#myURL#">Normal URL</a>

<!--- How you can use hashing to obfuscate variable names in a URL --->
<cfset myURL = "myApp.cfm?uid=#encrypt( '42', '<key>', 'AES/CBC/PKCS5Padding', 'HEX' )#" />
<a href="#myURL#">Hashed URL</a>

<!--- How you might normally populate a form field in your application --->
<input type="hidden" name="uid" value="42" />

<!--- How you can use hashing to obfuscate hidden field names --->
<input type="hidden" name="uid" value="#encrypt( '42', '<key>', 'BLOWFISH/CBC/PKCS5Padding', 'HEX' )#" />

<!--- How you might normally param a URL variable --->
<cfparam name="URL.uid" default="0" />

<!--- How you can param a hashed variable name --->
<cfparam name="URL.uid" default="#encrypt( '0', '<key>', 'AES/CBC/PKCS5Padding', 'HEX' )#" />

<!--- How you might normally param a URL variable --->
<cfparam name="FORM.uid" default="0" />

<!--- How you can param a hashed variable name --->
<cfparam name="FORM.uid" default="#encrypt( '0', '<key>', 'BLOWFISH/CBC/PKCS5Padding', 'HEX' )#" />
```

Obfuscated and Encrypted

```
<!--- How you might normally populate a URL in your application --->
<cfset myURL = "myApp.cfm?uid=42" />
<a href="#myURL#">Normal URL</a>

<!--- How you can use hashing to obfuscate variable names in a URL --->
<cfset myURL = "myApp.cfm?v#hash( 'uid', 'SHA-384' )#=#encrypt( '42', '<key>', 'AES/CBC/PKCS5Padding', 'HEX' )#" />
<a href="#myURL#">Hashed URL</a>

<!--- How you might normally populate a form field in your application --->
<input type="hidden" name="uid" value="42" />

<!--- How you can use hashing to obfuscate hidden field names --->
<input type="hidden" name="f#hash( 'uid', 'SHA-512' )#" value="#encrypt( '42', '<key>', 'BLOWFISH/CBC/PKCS5Padding', 'HEX' )#" />

<!--- How you might normally param a URL variable --->
<cfparam name="URL.uid" default="0" />

<!--- How you can param a hashed variable name --->
<cfparam name="URL[ 'v' & hash( 'uid', 'SHA-384' ) ]" default="#encrypt( '0', '<key>', 'AES/CBC/PKCS5Padding', 'HEX' )#" />

<!--- How you might normally param a URL variable --->
<cfparam name="FORM.uid" default="0" />

<!--- How you can param a hashed variable name --->
<cfparam name="FORM[ 'f' & hash( 'uid', 'SHA-512' ) ]" default="#encrypt( '0', '<key>', 'BLOWFISH/CBC/PKCS5Padding', 'HEX' )#" />
```

Too much of a good thing

- DO NOT attempt to hash and encrypt *everything* in the request context
- DO NOT expect good performance dynamically hashing and encrypting large lists of data
- DO pick and choose important data (database id's) to obfuscate and encrypt
- DO hash() keys and encrypt() values before looping
- DO use pagination for large lists of data that require hash() and encrypt()
- DO fall back to ECB and/or 128 bit keys if performance is an issue – better some security than none

Attack Vectors Overview

- SQL Injection
- XSS (Cross-Site Scripting)
- CSRF (Cross-Site Request Forgery)
- Cookies
- Tidbits
  - Cflocation
  - File upload validation
  - Form Methods
  - File Injection
  - Application Naming

SQL Injection

```
1  <!--- Vulnerable to SQL Injection --->
2  <cfquery name="qGetSomething" datasource="myDSN">
3      SELECT oneThing, twoThing
4      FROM someTable
5      WHERE someId = #url.someId#
6  </cfquery>
7
8  <!--- Not vulnerable to SQL injection --->
9  <cfquery name="qGetSomething" datasource="myDSN">
10     SELECT oneThing, twoThing
11     FROM someTable
12     WHERE someId = <cfqueryparam value="#url.someId#" cfsqltype="cf_sql_integer" />
13 </cfquery>
14
15 myApp.cfm?someId=1  DELETE FROM someTable
16
```

XSS (Cross-Site Scripting)

```
1  <cfparam name="user" default="0" />
2
3  <!--- Vulnerable to XSS attack --->
4  <cfoutput>Hello #user#!</cfoutput>
5
6  <!--- CF9 or below Protection --->
7  <cfoutput>Hello #htmlEditFormat( user )#</cfoutput>
8
9  <!--- CF10+ (or CF9 w/ ESAPI loaded), Railo, Lucee Protection --->
10 <cfoutput>Hello #encodeForHtml( user )#</cfoutput>
11
12 myApp.cfm?user=<script>alert('HaX0R3d!');</script>
13
14 this.scriptProtect = 'all' - Application.cfc
15
```

CSRF (Cross-Site Request Forgery)

```
<!--- FORM without CSRF protection --->
<form method="post">
    <input type="text" name="yourName" value="">
    <input type="submit" name="btnSubmit" value="Go!">
</form>

<!--- FORM *with* CSRF protection --->
<form method="post">
    <cfoutput>
    <input type="hidden" name="token" value="#CSRFGenerateToken()#">
    </cfoutput>
    <input type="text" name="yourName" value="">
    <input type="submit" name="btnSubmit" value="Go!">
</form>

<!--- Check for valid token on form processing to ensure a valid token --->
<cfif NOT CSRFVerifyToken( form.token )>
    <!--- form is not from a valid source --->
</cfif>
```

Cookies. Yummy.

```
1  <!--- cookie without protection --->
2  <cfcookie name="someName" value="someValue" />
3
4  <!--- cookie with httpOnly protection --->
5  <cfcookie name="someName" value="someValue" httpOnly="true" />
6
7  <!--- secure (SSL) cookie with httpOnly protection --->
8  <cfcookie name="someName" value="someValue" httpOnly="true" secure="true" />
9
```

Other Tidbits

```
<!--- cflocation sending CFID and CFTOKEN in URL --->
<cflocation url="somePlace.cfm" /> <!--- default is 'true' --->
<cflocation url="somePlace.cfm" addtoken="true" />

<!--- cflocation *not* sending CFID and CFTOKEN in URL --->
<cflocation url="somePlace.cfm" addtoken="false" />

<!--- Always do file upload validation --->
<cffile action="upload" filefield="photo" accept="image/gif,image/png,image/jpg"
    destination="#getTempDirectory()#" nameconflict="overwrite" strict="true">

<!--- Vulnerable form methods --->
<form action="someValidator.cfm"><!--- default method is 'get' --->
<form action="someValidator.cfm" method="get">

<!--- Always use 'post' method for forms --->
<form action="someValidator.cfm" method="post">

<!--- Vunerable file injection vectors --->
<!--- attacker could pass in 'section=../../server-keys.xml' --->
<cfinclude template="includes/#section#" />
<cffile action="write" file="#section#" [...] />

<!--- Always name your applications --->
<cfapplication name="myAppName">

this.name = 'myAppName';
```

HTTP Headers for Security

```
// use HTTP headers to help protect against common attack vectors
getPageContext().getResponse().addHeader( 'X-Frame-Options', 'deny' );
getPageContext().getResponse().addHeader( 'X-XSS-Protection', '1; mode=block' );
getPageContext().getResponse().addHeader( 'X-Content-Type-Options', 'nosniff' );
```

- The **X-Frame-Options** HTTP response header can be used to indicate whether or not a browser should be allowed to render a page in a <frame> , <iframe> or <object>
- The **X-XSS-Protection** HTTP response header enables the Cross-site scripting (XSS) filter built into most recent web browsers.
- The **X-Content-Type-Options** HTTP response header has only one defined value, "nosniff", which prevents Internet Explorer and Google Chrome from MIME-sniffing (drive-by download prevention).

- SEE ALSO: Content Security Policy (CSP) and Check Your Headers (http://cyh.herokuapp.com/cyh)

Secure Authentication

- https://sa.vsgcom.net/ - DEMO
- https://github.com/ddspringle/framework-one-secure-auth - FOSS Code (CF10+, Lucee 4.5+, FW/1 3.5)

Multi-Factor Authentication

- There are three factors:
  - Something the user knows (password, etc.)
  - Something the user has (phone, smartcard, etc.)
  - Something the user is (biometrics – iris, fingerprint, etc.)
- We'll use two of the three factors:
  - Something the user knows (password)
  - Something the user has (phone)

Two-Factor Authentication

- https://tfa.vsgcom.net/ - DEMO
- https://github.com/ddspringle/framework-one-two-factor-auth - FOSS Code (CF10+, Lucee 4.5+, FW/1 3.5)

Additional Resources

- css.dvdmenubacks.com – Multi-Factor Auth Preso's and code (tag based)
- blog.vsgcom.net – Security related blog posts (obfuscation and encryption)
- cfdocs.org/security – Security documentation
- www.owasp.org – Open Web Application Security Project – makers of ESAPI
- www.petefreitag.com – CFML security blog, FuseGuard and HackMyCF developer.