

# ColdFusion Application Security: The Next Step - Handout

Jason Dean  
<http://www.12robots.com>  
Boston CFUG – September 16<sup>th</sup>, 2009

## REQUEST FORGERIES

A request forgery, also sometimes called a Cross-Site (or On-Site) Request Forgery(XSRF), is an attack that is perpetrated against the user of a site who has authenticated access to that site

The user is unwittingly tricked into performing actions on a site through hidden code displayed to them and, therefore, executed in their browser.

### Form / View

```
<cfset session.deleteForm.CSRFToken = createUUID() />
<cfset session.deleteForm.tokenExpires = DateAdd('m', 10, Now()) />

<form action="deletePage.cfm" method="post">
    <input type="hidden" name="pageid" value="1" />
    <input type="hidden" name="key" value="#session.deleteForm.CSRFToken#" />
    <input type="submit" name="btnSubmit" value="Delete Page 1" />
</form>
```

### Action Page / Method

```
<cfif NOT StructKeyExists(form, "CSRFToken") OR
    NOT StructKeyExists(session.deleteForm, "CSRFToken") OR
    NOT StructKeyExists(session.deleteForm, "tokenExpires") OR
    NOT IsDate(session.deleteForm.tokenExpires) OR
    NOT session.deleteForm.CSRFToken EQ form.CSRFToken OR
    NOT DateDiff("s", Now(), session.deleteForm.tokenExpires) GT 0>

    <cflog file="Security" type="warning" text="Possible CSRF Attack">
    <cfthrow message="Access denied">
<cfelse>
    <cfset StructDelete(session.deleteForm, "token")>
    <cfset StructDelete(session.deleteForm, "tokenExpires")>
</cfif>

<!--- Continue Processing Form --->
```

## PASSWORD SECURITY

### Achieving a “Super Secure” Password

Require:

- Alpha and numeric characters
- Uppercase and lowercase characters
- Special characters
- Minimum Length (7 or 8 characters)
- Not contain the username
- Not a date
- Periodic password change (frequency should be based on how secure the site need be)

**NOTE:** Remember that “Password1!” can pass all of the above tests, so this is not enough.

### Best Practices

- Don't set a minimum length above 8 character
- Where possible, use SSL and Load the login form using SSL
- Don't send Login credentials on th URL string (except Web Services or similar, then only over SSL and preferably hashed or encrypted)
- NEVER store or transmit passwords in clear text
- Create an audit log of login attempts
- If you lock a user out after a certain number of login attempts, do not use a cookie or tie it to the user's session, do it in the database
- Hash and Salt your passwords

### Hashing

We can replace MD5 with a number of other hashing algorithms that produce different fixed-lengths

- MD5: (Default) Generates a 32-character, hexadecimal string, using the MD5 algorithm (The algorithm used in ColdFusion MX and prior releases)
- SHA: Generates a 40-character string using the Secure Hash Standard SHA-1 algorithm
- SHA-256: Generates a 64-character string using the SHA-256 algorithm
- SHA-384: Generates a 96-character string using the SHA-384 algorithm
- SHA-512: Generates an 128-character string using the SHA-512 algorithm

Notes about hashing:

- A specific string will ALWAYS result in the same hash value
- Collisions occur when two values have the same Hash value
- Strong hashing algorithms are going to have fewer collisions
- The longer the hash value, the less likely you will have collisions.

## Forgot My Password – Best Practices

- Never have your “Forgot My Password” function e-mail the users password (If you are hashing password you won't be able to anyway)
- Either reset the users password and email them the new password or send the user a temporary URL that can be used for them to reset the password
- Force the user to change their password after they first log in after a reset
- Keep a log of the last X hashes of the users password so they cannot reset their password to something that have used previously (Within reason)
- Make sure your Change Password functionality uses the same strength and hashing functions as your initial password set up
- Do not login a user from the “Forgot My Password” section. Always make them go through their e-mail.

## COOKIE SECURITY

### Cookie Parameters

- Name and Value – Pretty self-explanatory. No Security Concerns (except content)
- Expires – This value definitely carries a security concern with it, especially for session management cookies.
- Path – The path to which a cookie applies within a domain. If set, a domain must also be set. Default is for all pages on the domain that set the cookie to be able to access it.
- Domain – The domain to which the cookie applies. Must start with a period. Example: domain=“.12robots.com”. Only the specified domain can access the cookie. By default, the domain that set the cookie will be used.
- Secure – If set to “True” the cookie will only be submitted to the server over an SSL connection. No SLL, no cookie
- HTTPOnly – This feature is new to browsers (IE6+ and FF 2.0.0.5+). It is a flag that tells the browser to only submit the cookie via HTTP requests, which means it cannot be access via JavaScript

## SESSION MANAGEMENT

Sessions can be persisted in 3 ways

### Passing in a URL (Tough to maintain)

- Developer has to remember to always pass the URL string
- End user can easily lose their session by messing with the URL
- Session Token will be logged by the web server

Example:

`http://www.12robots.com/mypage.cfm?CFID=2&CFTOKEN=10666880`

### Passing in POST requests (Very difficult to maintain)

- EVERY request from page to page needs to be a <form>, even navigation

Example:

```
<input type="hidden" name="cfid" value="#session.cfid#">
<input type="hidden" name="cftoken" value="#session.cftoken#">
```

### Using a Token Cookie (Easy to do, easy to maintain, easier to secure)

- End user would have a hard time screwing it up
- Does require that your end users have cookies enabled

How can a Session Token become compromised?

- Physical Access to a machine
- Sessions Fixation
- XSS
- Social Engineering
- Brute-Force Guessing

## Session Token Best Practices

- Do NOT pass your session tokens in the URL string
- Use cookies as a best practice
- Set ColdFusion to use UUIDs for session tokens or use J2EE Session tokens
- Set token cookies to HTTPOnly (where possible) to prevent some XSS attacks
- Use SSL connection to prevent packet sniffing exploits
- Set token cookies to SECURE (where possible and where SSL is being used so they are only sent via SSL)
- Use DOMAIN and PATH attributes in your cookies to minimize where they are sent
- Set Session-Only cookies so that they expire when the browser is closed (CF Sessions Only, J2EE already does this)
- Don't use unnamed application
- Always use the SESSION prefix when accessing session variables
- Properly var scope within your functions

## Manipulating Session Token Cookies

### For CF Session Management - Application.cfc:

```
<cfset this.name = "sessionManagementApp">
<cfset this.sessionManagement = true>
<cfset this.sessionTimeout = CreateTimeSpan(0,0,20,0) />
<cfset this.setClientCookies = false />

<cffunction name="onSessionStart" output="false">
    <cfheader name="Set-Cookie" value="CFID=#session.CFID#;secure;HTTPOnly" />
    <cfheader name="Set-Cookie" value="CFTOKEN=#session.CFTOKEN#;secure;HTTPOnly" />
</cffunction>
```

### For JEE Session Management - Application.cfc

```
<cffunction name="onSessionStart" output="false">
    <!--- Expire the old Cookie --->
    <cfcookie name="jsessionid" expires="now"/>

    <!--- Get the HTTP Response Object --->
    <cfset response = getPageContext().getResponse() />

    <!--- Set the specifics for the cookie --->
    <cfset path = "/test" />
    <cfset domain = "my.local" />
    <cfset secure = "" /> <!--- Use val of "Secure" or leave blank --->
    <cfset HTTPOnly = "" /> <!--- Use val of "HTTPOnly" or leave blank --->

    <cfscript>
        header = "jsessionid" & "=" & session.sessionid & ";domain=" &
            domain & ";path=" & path & ";" & secure & ";" & HTTPOnly;
        response.addHeader("Set-Cookie", header);
    </cfscript>
</cffunction>
```