

Adobe White Paper

ColdFusion MX 7 developer security guidelines

Erick Lee and Sarge Sargent

October 2007

Adobe, the Adobe logo, and ColdFusion are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Linux is a registered trademark of Linus Torvalds in the U.S. and other countries. Windows is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries. UNIX is a registered trademark of The Open Group in the U.S. and other countries. Java is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries. All other trademarks are the property of their respective owners.

© 2007 Adobe Systems Incorporated. All rights reserved. Printed in the USA.

Adobe Systems Incorporated

345 Park Avenue

San Jose, CA

95110-2704

USA

www.adobe.com

Contents

Authentication	1
Best Practices	2
Best Practices In-Action.....	3
Application.cfc	3
protected.cfm	4
Login.cfm.....	4
Simple Authentication using a Database	4
NTLM	5
LDAP	6
Logout	6
Authorization	7
Best Practices	7
Best Practices in Action	8
ColdFusion Components (CFCs)	9
Best Practices	9
Session Management.....	9
ColdFusion Session Management	10
J2EE Session Management	10
Configuring Session Management.....	11
Application Code.....	11
Best Practices	12
Data Validation and Interpreter Injection.....	13
SQL Injection.....	13
LDAP Injection.....	13
XML Injection	13
Event Gateway, IM, and SMS Injection	14
Best Practices	14
Best Practice in Action	15
Error Handling and Logging.....	15
Error Handling	16
Best Practices	16
Logging.....	17
Best Practices	18
Best Practices in Action	18
File System	19
Best Practice	20
Cryptography.....	20

Pseudo-Random Number Generation.....	20
Symmetric Encryption	21
Pluggable Encryption.....	22
SSL 22	
Best Practices	23
Configuration	23
Best Practices.....	23
Data Sources screen	27
Maintenance	27
References.....	29

ColdFusion is a scripting language for creating dynamic Internet applications. It uses ColdFusion Markup Language (CFML), an XML tag-based scripting language, to connect to data providers, authentication systems, and other services. ColdFusion features built-in server-side file search, limited Flash movie rendering, and charting capabilities. ColdFusion is implemented on the Java platform and uses a Java 2 Enterprise Edition (J2EE) application server for many of its runtime services. ColdFusion can be configured to use an embedded J2EE server (JRun), or it can be deployed as a J2EE application on a third party J2EE application server such as Apache Tomcat, IBM WebSphere, and BEA WebLogic (which supports clustering). This whitepaper will attempt to provide guidance and clarifications on protecting your ColdFusion application from common threats facing Internet applications. Any example code given in this whitepaper may not work without modification for your particular environment.

ColdFusion is available at <http://www.adobe.com/products/coldfusion/>

Authentication

Web and application server authentication can be thought of as two different controls (see Figure 1). Web server authentication is controlled by the web server administration console or configuration files. These controls do not need to interact with the application code to function. For example, using Apache, you modify the http.conf or .htaccess files; or for IIS, modify the IIS Microsoft Management Console. Basic authentication works by sending a challenge request back to a user's browser consisting of the protected URI. The user must then respond with the user ID and password, separated by a single colon, encoded using base64 encoding. application-level authentication occurs at a layer after the web server access controls have been processed. This section examines how to use ColdFusion to authenticate and authorize users to resources at the application level.

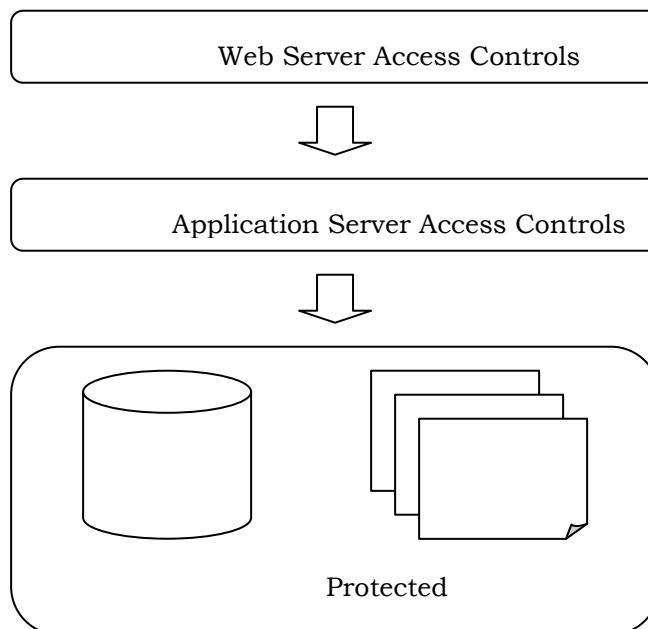


Figure 1: *Web server and application server authentication occur in sequence before access is granted to protected resources.*

ColdFusion enables you to authenticate against multiple system types. These types include LDAP, text files, Databases, NTLM, and others via custom modules. The section below describes using these credential stores according to best practices.

Best Practices

- When a user enters an invalid credential into a login page, do NOT return which item was incorrect; instead, show a generic message. For example, "Your login information was invalid!"
- Never submit login information via a GET request; always use POST.
- Use SSL to protect login page delivery and credential transmission.
- Remove dead code and client-side viewable comments from all pages.
- Set application variables in the Application.cfc file. The values you use ultimately depend on the function of your application; however, for best practices use the following.
 - `applicationTimeout = #CreateTimeSpan(0,8,0,0)#`
 - `loginStorage = session`
 - `sessionTimeout = #CreateTimeSpan(0,0,20,0)#`
 - `sessionManagement = True`
 - `scriptProtect = All`
 - `setClientCookies = False (Use JSESSIONID)`
 - `setDomainCookies = False`
 - `name` (This value is application-dependent; however, it should be set)
- Do not depend on client-side validation. Validate input parameters for type and length on the server, using regular expressions or string functions.
- Database queries must use parameterized queries (`<cfqueryparam>`) or properly constructed stored procedures (`<cfstoredproc>`).
- Database connections should be made created using a lower privileged account. Your application should not log into the database using sa or dbadmin.
- Hash passwords in a database or flat file using SHA-256 or greater with a random salt value for each password. For example, `Hash(password + salt, "SHA-256")`
- Call `StructClear(Session)` to completely clear a user's session. Issuing `<cflogout>` when using `LoginStorage=Session` removes the `SESSION.cfauthorization` variable from the Session scope, but does not clear the current user's session object.
- Prompt the user to close the browser session to ensure that header authentication information has been flushed.

Best Practices In-Action

To help demonstrate the use some of these best practices, assume you want to protect a page called protected.cfm. To protect this content you need the Application.cfc file, a login page (login.cfm), and code to perform your authentication and logout (Auth.cfc). Note that all the filenames and variables used in this section are arbitrary.

Application.cfc:

```
<cfcomponent >
  <cfscript>
    This.name = "OWASP_Sample";
    This.applicationTimeout = CreateTimeSpan(0,8,0,0);
    This.sessionManagement = true;
    This.sessionTimeout = CreateTimeSpan(0,0,20,0);
    This.loginStorage = "session";
    This.scriptProtect = "All";
  </cfscript>
  <cffunction name = "onRequestStart">
    <cfargument name = "thisRequest" required="true"/>
    <cfset Request.DSN = "owaspDB"><!--- <cflogout> --->
    <cfif isDefined('Form.logout')>
      <cfinvoke
        component="auth"
        method="logout"
        loginType="simple">
    </cfif>
    <cflogin>
      <cfif NOT isDefined('cflogin')>
        <cfinclude template="login.cfm"><cfabort>
      <cfelse>
        <cfif len(trim(cflogin.name)) AND
len(trim(cflogin.password))>
          <cfinvoke
            component="auth"
            method="LoginUser"
            returnvariable="result"
            strUserName="#cflogin.name#"
            strPassword="#cflogin.password#">
          </cfinvoke>
          <cfif result.authenticated>
            <cfloginuser
              name="#cflogin.name#"
              password="#cflogin.password#"
              roles="#result.roles#" >
          <cfelse>
            <cfset Request.boolError = true>
            <cfinclude template="login.cfm"><cfabort>
          </cfif>
        <cfelse>
          <cfset Request.boolError = true>
          <cfinclude template="login.cfm"><cfabort>
        </cfif>
      </cfif>
    </cflogin>
  </cffunction>
  <cffunction name="onSessionEnd">
    <cfargument name="thisSession" required="true"/>
    <cfargument name="thisApp" required="false" />
    <cfinvoke
```

```
        component="auth"
        method="logout"
        loginType="simple">
    </cffunction>
</cfcomponent>
```

protected.cfm:

```
<cfset strPath = ExpandPath("*.*)">
<cfset strDir = GetDirectoryFromPath(strPath)>
<html>
<body>
    <title>OWASP Security Test</title>
</body>
<b>You have successfully logged into the new application</b>
<ul>
    <li>This application directory called
        "<cfoutput>#strDir#</cfoutput>" is protected</li>
    <li>You can also remove any or all of this text and replace it
        with any valid browser code that you choose, such as CFML or
        HTML</li>
</ul>
<cfform name="logMeout" action="#CGI.script_name#" method="post">
    <cfinput type="submit" name="logout" value="Logout">
</cfform>
</html>
```

Login.cfm:

```
<cfparam name="Request.boolError" type="boolean" default="false">
<cfif Request.boolError>
    <span style="color: red">Your login information was invalid!</span>
</cfif>
<cfoutput>
<H2>You must login to access this restricted resource.</H2>
    <cfform name="loginform" action="protected.cfm" method="Post">
        <table>
            <tr>
                <td>username:</td>
                <td><cfinput type="text" name="j_username" required="yes"
                    message="Username required"></td>
            </tr>
            <tr>
                <td>password:</td>
                <td><cfinput type="password" name="j_password"
                    required="yes" message="Password required"></td>
            </tr>
        </table>
        <br>
        <input type="submit" value="Log In">
    </cfform>
</cfoutput>
```

Simple Authentication using a Database

After basic authentication, probably the second most widely used method of authenticating users on the web is database login. The code snippet below shows how to accomplish database authentication using best practices.

Auth.cfc:

```

<cffunction name="LoginUser" access="public" output="false"
  returntype="struct">
  <cfargument name="strUserName" required="true" type="string">
  <cfargument name="strPassword" required="true" type="string">
  <cfset var retargs = StructNew(>
  <cfif IsValid("regex", strUserName, "[A-Za-z0-9%]*") AND
    IsValid("regex", strPassword, "[A-Za-z0-9%]*")>
    <cfquery name="loginQuery" dataSource="#Request.DSN#" >
    SELECT hashed_password, salt
    FROM UserTable
    WHERE UserName =
    <cfqueryparam value="#strUserName#" cfsqltype="CF_SQL_VARCHAR"
    maxlength="25">
    </cfquery>
    <cfif loginQuery.hashed_password EQ Hash(strPassword &
    loginQuery.salt, "SHA-256" )>
      <cfset retargs.authenticated = true>
      <cfset Session.UserName = strUserName>
      <cfset retargs.roles = "Admin">
    <cfelse>
      <cfset retargs.authenticated = false>
    </cfif>
  <cfelse>
    <cfset retargs.authenticated false>
  </cfif>
  <cfreturn retargs>
</cffunction>

```

NTLM

In addition to using controls available via IIS and using the browser dialog box, ColdFusion enables you to authenticate users via a web form using NTLM. The code snippet below shows how to accomplish NTLM authentication using best practices.

Auth.cfc (cont'd):

```

<cffunction name="LoginUser" access="public" output="false"
  returntype="struct">
  <cfargument name="username" required="true" type="string">
  <cfargument name="npassword" required="true" type="string">
  <cfargument name="ndomain" required="true" type="string">
  <cfset var retargs = StructNew(>
  <cfif IsValid("regex", arguments.username, "[A-Za-z0-9%]*")
    AND IsValid("regex", arguments.npassword, "[A-Za-z0-9%]*")
    AND IsValid("regex", arguments.ndomain, "[A-Za-z0-9%]*")>
    <CFNTAuthenticate
      username="#arguments.username#"
      password="#arguments.npassword#"
      domain="#arguments.ndomain#"
      result="authenticated">
    <cfif findNoCase("success", authenticated.status)>
      <cfset retargs.authenticated = true>
    <cfelse>
      <cfset retargs.authenticated = false>
    </cfif>
  <cfelse>
    <cfset retargs.authenticated = false>
  </cfif>
  <cfreturn retargs>
</cffunction>

```

```
<!-- return role here -->
<cfreturn retargs>
</cffunction>
```

LDAP

To set up authentication against an LDAP server, including Active Directory:

Auth.cfc (cont'd):

```
<cffunction name="LoginUser" access="public" output="true"
  returntype="struct">
  <cfargument name="lServer" required="true" type="string">
  <cfargument name="lPort" type="numeric">
  <cfargument name="sUsername" required="true" type="string">
  <cfargument name="sPassword" required="true" type="string">
  <cfargument name="uUsername" required="true" type="string">
  <cfargument name="uPassword" required="true" type="string">
  <cfargument name="sQueryString" required="true" type="string">
  <cfargument name="lStart" required="true">
  <cfset var retargs = StructNew()>
  <cfset var username = replace(sQueryString, "{username}", uUserName)>
  <cfldap action="QUERY"
    name="userSearch"
    attributes="dn"
    start="#arguments.lStart#"
    server="#arguments.lServer#"
    port="#arguments.lPort#"
    username="#arguments.sUsername#"
    password="#arguments.sPassword#">
  <!-- If user search failed or returns 0 rows abort -->
  <cfif NOT userSearch.recordCount>
    <cfoutput>Error</cfoutput>
    <cfset retargs.authenticated = false>
  </cfif>
  <!-- pass the user's DN and password to see if the user
    authenticates
    and get the user's roles -->
  <cfldap
    action="QUERY"
    name="auth"
    attributes="dn,roles"
    start="#arguments.lStart#"
    server="#arguments.lServer#"
    port="#arguments.lPort#"
    username="#username#"
    password="#arguments.uPassword#" >
  <!-- If the LDAP query returned a record, the user is valid. -->
  <cfif auth.recordCount>
    <cfset retargs.authenticated = true>
  </cfif>
  <cfreturn retargs>
</cffunction>
```

Logout

Auth.cfc (cont'd):

```
<!-- Logout -->
```

```
<cffunction name="logout" access="remote" output="true">
  <cfargument name="logintype" type="string" required="yes">
  <cflogout>
  <cflock scope="session" type="exclusive" timeout="5"
throwtimeout="true">
  <cfset StructClear(Session)>
  </cflock>
  <cfif arguments.logintype eq "challenge">
  <cfset foo = closeBrowser()>
  <cfelse>
  <!--- replace this URL to a page logged out users should see --
->
  <cflocation url="login.cfm">
  </cfif>
</cffunction>

<!--- Close Browser --->
<cffunction name="closeBrowser" access="private" output="true">
  <script language="javascript">
    if(navigator.appName == "Microsoft Internet Explorer") {
      alert("The browser will now close to complete the logout.");
      window.close();
    }
    if(navigator.appName == "Netscape") {
      alert("To complete the logout you must close this browser.");
    }
  </script>
</cffunction>
</cfcomponent>
```

Authorization

Authorization ensures that the authenticated user has the appropriate privileges to access resources. The resources a user has access to depend on the user's role. For more information see the **Authorization** chapter of this document.

Best Practices

- If your application allows users to be logged in for long periods of time, ensure that controls are in place to revalidate a user's authorization to a resource. For example, if Bob has the role of "Top Secret" at 1:00, and at 2:00 while he is logged in his role is reduced to Secret, he should not be able to access "Top Secret" data anymore.
- Architect your services (for instance, the data source and the web service) to query a user's role directly from the credential store instead of trusting the user to provide an accurate listing of the user's roles.

Best Practices in Action

Continuing with the previous code, the following snippet extends the code in the dbLogin method of the auth.cfc file to return the user's roles. The roles are passed to the <cfloginuser> tag to provide authentication to ColdFusion's built-in login structure. The roles are also used to provide authorization to ColdFusion Components. (See the [ColdFusion Components](#) section of this document.)

Auth.cfc:

```
<cffunction name="dblogin" access="public" output="false"
    returntype="struct">
    <cfargument name="strUserName" required="true" type="string">
    <cfargument name="strPassword" required="true" type="string">
    <cfset var retargs = StructNew()>
    <cfif IsValid("regex", strUserName, "[A-Za-z0-9%]*") AND
        IsValid("regex", strPassword, "[A-Za-z0-9%]*")>
        <cfquery name="loginQuery" dataSource="#Request.DSN#" >
            SELECT userid, hashed_password, salt
            FROM UserTable
            WHERE UserName =
            <cfqueryparam value="#strUserName#" cfsqltype="CF_SQL_VARCHAR"
                maxlength="25">
        </cfquery>
        <cfif loginQuery.hashed_password EQ Hash(strPassword &
            loginQuery.salt, "SHA-256" )>
            <cfset retargs.authenticated="YES">
            <cfset Session.UserName = strUserName>
            <!-- Add code to get roles from database -->
            <cfquery name="authQuery" dataSource="#Request.DSN#" >
                SELECT roles.role
                FROM roles INNER JOIN (users INNER JOIN userroles ON
                users.userid = userroles.userid) ON roles.roleid =
                userroles.roleid
                WHERE ((users.usersid)='<cfqueryparam
                value="#loginQuery.userid#" cfsqltype="CF_SQL_INTEGER">'))
            </cfquery>
            <cfif authQuery.recordCount>
                <cfset retargs.roles = valueList(authQuery.role)>
            <cfelse>
                <cfset retargs.roles = "user">
            </cfif>
            <cfelse>
                <cfset retargs.authenticated="NO">
            </cfif>
        <cfelse>
            <cfset retargs.authenticated="NO">
        </cfif> ---><cfset retargs.authenticated = true><cfset
            retargs.roles = "Admin">
        <cfreturn retargs>
    </cffunction>
```

ColdFusion Components (CFCs)

This section provides guidance on using ColdFusion components (CFCs) without exposing your web application to unnecessary risk. ColdFusion provides two ways of restricting access to CFCs: role-based security and access control.

Role-based security is implemented by the **roles** attribute of the `<cffunction>` tag. The attribute contains a comma-delimited list of security roles that can call this method.

Access control is implemented by the **access** attribute of the `<cffunction>` tag. The possible values of the attribute in order of most restricted behavior are: `private` (strongest), `package`, `public` (default), and `remote` (weakest).

Private: The method is accessible only to methods within the same component. This is similar to the Object Oriented Programming (OOP) private identifier.

Package: The method is accessible only to other methods within the same package. This is similar to the OOP protected static identifier.

Public: The method is accessible to any CFC or CFM on the same server. This is similar to the OOP public static identifier.

Remote: Allows all the privileges of public, in addition to accepting remote requests from HTML forms, Flash, or a web services. This option is required to publish the function as a web service.

Best Practices

Do not use `THIS` scope inside a component to expose properties. Use a getter or setter function instead. For example, instead of using `THIS.myVar`, create a public function that sets the variable (for example, `setMyVar(value)`).

Do not omit the role attribute, as ColdFusion will not restrict user access to the function.

Avoid using `Access="Remote"` if you do not intend to call the component directly from a URL.

Session Management

The term *session* has two meanings for web applications. At the server level, *session* refers to the connections between a client (browser) and the server. At the application level, *session* refers to the activities of an individual user within a given application. ColdFusion uses tokens to identify unique browser sessions to the server. It will also use these same tokens to identify user sessions within an application. ColdFusion has two types of session management: ColdFusion (CFID/CFToken) and J2EE (JSESSIONID).

- CFID: a sequential four-digit integer
- CFToken: a random eight-digit integer by default. It can also be generated as a ColdFusion UUID (a 32-character alphanumeric string) for greater security. (See the [Configuration](#) section of this document.)

- JSESSIONID: a secure, random alphanumeric string

The two types cannot be used simultaneously. However, ColdFusion session management must be enabled in order to allow J2EE session management. When either type is enabled, client data is persisted in the session scope in ColdFusion's memory space. In order to use session-scope variables in application code, you must set the initialization variable to true in Application.cfc (`This.sessionManagement`).

ColdFusion Session Management

ColdFusion session management is enabled by default. It utilizes CFID and CFToken as session identifiers. It sends them to the browser as persistent cookies with every request. If cookies are disabled, developers must pass these values in the URL. Session variables are automatically cleared when the session timeout is reached—but not when the browser closes.

Table 1: *Default session scope variables:*

Variable	Description
Session.CFID	The value of the CFID client identifier
Session.CFToken	The value of the CFToken security token
Session.URLToken	A combination of the CFID and CFToken in URL format: <i>CFID=cfid_number&CFTOKEN=cftoken_num</i>
Session.SessionID	A combination of the application name and the session token that uniquely identifies the session: <i>applicationName_cfid_cftoken</i>

Pros:

- It is compatible with all versions of ColdFusion.
- It uses the same session identifiers as ColdFusion's client management.
- It is enabled by default.

Cons:

- CFID and CFToken are created as persistent cookies.
- You can only use one unnamed application per server instance.
- Sessions persist when the browser closes.
- ColdFusion session scope is not serializable.

J2EE Session Management

J2EE session management is disabled by default. J2EE sessions utilize JSESSIONID as the session identifier. ColdFusion sends JSESSIONID as a nonpersistent (or in-memory) cookie to the browser. If cookies are disabled, developers must pass the value in the URL. Session variables are automatically cleared when the session timeout is reached and when the browser closes.

Table 2: *Default session scope variables:*

Variable	Description
Session.URLToken	A combination of the CFID, CFToken, and JSESSIONID in URL format:

	<code>CFID=cfid_number&CFTOKEN=cftoken_num&JSESSIONID=SessionID</code>
Session.SessionID	The JSESSIONID value that uniquely identifies the session.

Pros:

- ColdFusion session data is shareable with JSP pages and Java servlets.
- It uses a session-specific JSESSIONID identifier.
- It can use multiple unnamed sessions.
- Sessions automatically expire when the browser closes.
- ColdFusion session scope is serializable.

Cons:

- It must be enabled manually.
- It is not backwards-compatible with earlier versions of ColdFusion.
- JSESSIONID is not used for client management.
- When J2EE session management is enabled, ColdFusion session management cannot be used.

Configuring Session Management

Session management must be enabled in two places in order to use session variables:

- 1 The ColdFusion Administrator (See the Configuration section of this document)
- 2 Application initialization code

Application Code

Application initialization code is created in the pseudo-constructor area (the section above the application event handlers) of the Application.cfc file or the `<cfapplication>` in the Application.cfm file. You should only use one of these application files, and ColdFusion will ignore the Application.cfm file if it finds an Application.cfc file in the same directory tree.

Application.cfc

Add the following code right below the opening `<cfcomponent>` tag:

- Set `This.sessionManagement` equal to a positive ColdFusion Boolean value; for example, `this.sessionManagement=true`
- Set `This.sessionTimeout` equal to a valid time value using the `CreateTimeSpan` function; for example, `This.sessionTimeout=CreateTimeSpan(0,0,20,0)`
- Optionally, provide the application name by using the `This.name` variable.

Application.cfm

Add the following attributes to the `<cfapplication>` tag:

- Set `sessionManagement` equal to a positive ColdFusion Boolean value; for example, `sessionManagement=true`
- Set `sessionTimeout` equal to a valid time value using the `CreateTimeSpan` function; for example, `sessionTimeout=CreateTimeSpan(0,0,20,0)`
- Optionally, provide an application name using the `name` attribute.

Note: Unnamed applications can facilitate session integration with JSPs and Java servlets for J2EE Session Management. However, they should not be used with ColdFusion session management because the application name is used to create the `Session.SessionID` variable.

Best Practices

- Use session timeout values of 20 minutes or less.
- For J2EE sessions, ensure the session-timeout parameter in `cf_root/WEB-INF/web.xml` is greater than or equal to ColdFusion's maximum session timeout.
- Only enable J2EE session variables if all applications on the server will be using it. Do not enable if applications require client management.
- Create `CFID` and `CFToken` as nonpersistent cookies.
 - See ColdFusion TechNote 17915: How to write `CFID` and `CFTOKEN` as per-session cookies at http://www.adobe.com/cfusion/knowledgebase/index.cfm?id=tn_17915.
- Enable `UUID CFToken` for stronger ColdFusion session identifiers.
 - See ColdFusion TechNote 18133: How to guarantee unique `CFToken` values at http://www.adobe.com/cfusion/knowledgebase/index.cfm?id=tn_18133.
- Avoid passing session identifiers (`CFID/CFToken` or `JSESSIONID`) on the URL.
 - Use cookies whenever possible.
 - If cookies are not available, use the `URLSessionFormat` function for links.
- Only use one unnamed application per ColdFusion server instance.
- Always use the `SESSION` prefix when accessing session variables.
- Lock read/write access to session variables that may cause race conditions.
- Do not overwrite the default session variables.
- Loop over `StructDelete(Session.variable)` instead of using `StructClear(Session)` to remove variables from the session scope.
- Use `<cflogout>` to remove login information from the session scope when using `loginStorage=Session`.
- If `clientManagement="Yes"` and `clientStorage="Cookie"`, do not store sensitive information in the client's cookie. Any information that could aid in identity theft if revealed to a third-party should not be included in a cookie; for example, passwords, phone numbers, or Social Security numbers.

Data Validation and Interpreter Injection

This section focuses on preventing injection in ColdFusion. Interpreter injection involves manipulating application parameters to execute malicious code on the system. The most prevalent of these is SQL injection, but the concept also includes other injection techniques, including LDAP, ORM, User Agent, XML, and more.. As a developer you should assume that all input is malicious. Before processing any input coming from a user, data source, component, or data service, you should validate it for type, length, and/or range. ColdFusion includes support for regular expressions and CFML tags that can be used to validate input.

SQL Injection

SQL Injection involves sending extraneous SQL queries as variables. ColdFusion provides the `<cfqueryparam>` and `<cfprocparam>` tags for validating database parameters. These tags nests inside `<cfquery>` and `<cfstoredproc>`, respectively. For dynamic SQL submitted in `<cfquery>`, use the `CFSQLTYPE` attribute of the `<cfqueryparam>` to validate variables against the expected database datatype. Similarly, use the `CFSQLTYPE` attribute of `<cfprocparam>` to validate the datatypes of stored procedure parameters passed through `<cfstoredproc>`.

You can also strengthen your systems against SQL injection by disabling the Allowed SQL operations for individual data sources. See the **Configuration** section in this document for more information.

LDAP Injection

ColdFusion uses the `<cfldap>` tag to communicate with LDAP servers. This tag has an `ACTION` attribute that dictates the query performed against the LDAP. The valid values for this attribute are: `add`, `delete`, `query` (default), `modify`, and `modifyDN`. All `<cfldap>` calls are turned into JNDI (Java Naming And Directory Interface) lookups. However, because `<cfldap>` wraps the calls, it will throw syntax errors if native JNDI code is passed to its attributes, making LDAP injection more difficult.

XML Injection

Two parsers exist for XML data: SAX and DOM. ColdFusion uses DOM, which reads the entire XML document into the server's memory. This requires the administrator to restrict the size of the JVM containing ColdFusion. ColdFusion is built on Java; therefore, by default, entity references are expanded during parsing. To prevent unbounded entity expansion, before a string is converted to an XML DOM, filter out `DOCTYPES` elements.

After the DOM has been read, to reduce the risk of XML injection, use the ColdFusion XML decision functions: `isXML()`, `isXmlAttribute()`, `isXMLElement()`, `isXmlNode()`, and `isXmlRoot()`. The `isXML()` function determines if a string is well-formed XML. The other functions determine if the passed parameter is a valid part of an XML document. Use the `xmlValidate()` function to validate external XML documents against a document type definition (DTD) or XML schema.

Event Gateway, IM, and SMS Injection

ColdFusion MX 7 enables Event Gateways, instant messaging (IM), and SMS (short message service) for interacting with external systems. Event Gateways are ColdFusion components that respond asynchronously to non-HTTP requests: instant messages, SMS text from wireless devices, and so on. ColdFusion provides Lotus Sametime and XMPP (Extensible Messaging and Presence Protocol) gateways for instant messaging. It also provides an event gateway for interacting with SMS text messages.

Injection along these gateways can happen when users (and/or systems) send malicious code to execute on the server. These gateways all utilize ColdFusion Components (CFCs) for processing. Use standard ColdFusion functions, tags, and validation techniques to protect against malicious code injection. Sanitize all input strings and do not allow unvalidated code to access backend systems.

Best Practices

- Use the XML functions to validate XML input.
- When performing XPath searches and transformations in ColdFusion, validate the source before executing.
- Use ColdFusion validation techniques to sanitize strings passed to `xmlSearch` for performing XPath queries.
- When performing XML transformations, use only a trusted source for the XSL stylesheet.
- Ensure that the memory size of the Java Sandbox containing ColdFusion can handle large XML documents without adversely affecting server resources.
 - Set the memory value to less than the amount of RAM on the server (`-Xmx`)
- Remove DOCTYPE elements from the XML string before converting it to an XML object.
- Using `scriptProtect` to thwart most attempts of cross-site scripting. Set `scriptProtect` to All in the `Application.cfc` file.
- Use `<cfparam>` or `<cfargument>` to instantiate variables in ColdFusion. Use these tags with the name and type attributes. If the value is not of the specified type, ColdFusion returns an error.
- To handle untyped variables use `IsValid()` to validate its value against any legal object type that ColdFusion supports.
- Use `<cfqueryparam>` and `<cfprocparam>` to valid dynamic SQL variables against database datatypes.

- Use CFLDAP for accessing LDAP servers. Avoid allowing native JNDI calls to connect to LDAP.

Best Practice in Action

The sample code below shows a database authentication function using some of the input validation techniques discussed in this section.

```
<cffunction name="dblogin" access="private" output="false"
    returntype="struct">
    <cfargument name="strUserName" required="true" type="string">
    <cfargument name="strPassword" required="true" type="string">
    <cfset var retargs = StructNew()>
    <cfif IsValid("regex", uUserName, "[A-Za-z0-9%]*") AND
        IsValid("regex", uPassword, "[A-Za-z0-9%]*")>
        <cfquery name="loginQuery" dataSource="#Application.DB#" >
            SELECT hashed_password, salt
            FROM UserTable
            WHERE UserName =
            <cfqueryparam value="#strUserName#" cfsqltype="CF_SQL_VARCHAR"
                maxlength="25">
        </cfquery>
        <cfif loginQuery.hashed_password EQ Hash(strPassword &
            loginQuery.salt, "SHA-256" )>
            <cfset retargs.authenticated="YES">
            <cfset Session.UserName = strUserName>
            <!-- Add code to get roles from database -->
        <cfelse>
            <cfset retargs.authenticated="NO">
        </cfif>
    <cfelse>
        <cfset retargs.authenticated="NO">
    </cfif>
    <cfreturn retargs>
</cffunction>
```

Error Handling and Logging

All applications have failures—whether they occur during compilation or run time. Most programming languages will throw run-time exceptions for illegally executing code (for example, syntax errors), often in the form of cryptic system messages. These failures and resulting system messages can lead to several security risks if not handled properly, including: enumeration, buffer attacks, sensitive information disclosure, and more. If an attack occurs, it is important that forensics personnel be able to trace the attacker's tracks via adequate logging.

ColdFusion provides structured exception handling and logging tools. These tools can help developers customize error handling to prevent unwanted disclosure, and provide customized logging for error tracking and audit trails. These tools should be combined with web server, J2EE application server, and operating system tools to create the full system/application security overview.

Error Handling

Hackers can use the information exposed by error messages. Even missing templates errors (HTTP 404) can expose your server to attacks: buffer overflow, cross-site scripting (XSS), and more. If you enable the Robust Exception Information debugging option, ColdFusion will display:

- Physical path of template
- URI of template
- Line number and line snippet
- SQL statement used (if any)
- Data source name (if any)
- Java stack trace

ColdFusion provides tags and functions for developers to use to customize error handling. Administrators can specify default templates in the ColdFusion Administrator (CFAM) to handle unknown or unhandled exceptions.

ColdFusion's structured exception handling works in the following order:

3 Template level (ColdFusion templates and components)

- ColdFusion exception handling tags: `<cftry>`, `<cfcatch>`, `<cfthrow>`, and `<cfrethrow>`
- `try` and `catch` statements in CFScript

4 Application level (Application.cfc/cfm files)

- Custom templates for individual exceptions types specified with the `<cferror>` tag
- `Application.cfc` `onError` method to handle uncaught application exceptions

5 System level (ColdFusion Administrator settings)

- Missing template handler executed when a requested ColdFusion template is not found
- Site-wide error handler executed globally for all unhandled exceptions on the server

Best Practices

- Do not allow exceptions to go unhandled.
- Do not allow any exceptions to reach the browser.
 - Display custom error pages to users with an email link for feedback.
- Do not enable "Robust Exception Information" in production.
- Specify custom pages for ColdFusion to display in each of the following cases:
 - When a ColdFusion page is missing (the Missing Template Handler page)

- When an otherwise-unhandled exception error occurs during the processing of a page (the Site-wide Error Handler page)

You specify these pages on the Settings page in the Server Settings are in the ColdFusion MX Administrator; for more information, see the ColdFusion MX Administrator Help.

- Use the `<cferror>` tag to specify ColdFusion pages to handle specific types of errors.
- Use the `<cftry>`, `<cfcatch>`, `<cfthrow>`, and `<cfrethrow>` tags to catch and handle exception errors directly on the page where they occur.
- In CFScript, use the `try` and `catch` statements to handle exceptions.
- Use the `onError` event in `Application.cfc` to handle exception errors that are not handled by `try/catch` code on the application pages.

Logging

Log files can help with application debugging and provide audit trails for attack detection. ColdFusion provides several logs for different server functions. It leverages the Apache Log4j libraries for customized logging. It also provides logging tags to assist in application debugging.

The following is a partial list of ColdFusion log files and their descriptionsⁱⁱ:

Log file	Description
application.log	Records every ColdFusion MX error reported to a user. Application page errors, including ColdFusion MX syntax, ODBC, and SQL errors, are written to this log file.
exception.log	Records stack traces for exceptions that occur in ColdFusion.
scheduler.log	Records scheduled events that have been submitted for execution. Indicates whether task submission was initiated and whether it succeeded. Provides the scheduled page URL, the date and time executed, and a task ID.
server.log	Records startup messages and errors for ColdFusion MX.
customtag.log	Records errors generated in custom tag processing.
mail.log	Records errors generated by an SMTP mail server.
mailed.log	Records messages sent by ColdFusion MX.
flash.log	Records entries for Flash Remoting.

The CFAM contains the Logging Settings and log viewer screens. Administrators can configure the log directory, maximum log file size, and maximum number of archives. It also allows administrators to log slow running pages, CORBA calls, and scheduled task execution. The log viewer allows viewing, filtering, and searching of any log files in the log directory (default is `cf_root/logs`). Administrators can archive, save, and delete log files as well.

The `<cflog>` and `<cftrace>` tags allow developer to create customized logging. The `<cflog>` tag can write custom messages to the `Application.log`, `Scheduler.log`, or a custom log file. The custom log file must be in the default log directory; if it does not exist, ColdFusion will create it. The `<cftrace>` tag tracks execution times, logic flow, and variable values at the time the tag executes. It records the data in the `cftrace.log` (in the default logs directory) and can display this info either inline or in the debugging output of the current page request. Use `<cflog>` to write custom error messages, track user logins, and record user activity to a custom log file. Use `<cftrace>` to track variables and application state within running requests.

Best Practices

- Use `<cflog>` for customized logging:
 - Incorporate into custom error handling.
 - Record application-specific messages.
- Actively monitor and fix errors in ColdFusion's logs.
- Optimize logging settings:
 - Rotate log files to keep them current.
 - Keep files size manageable.
 - Enable logging of slow-running pages.
 - Set the time interval lower than the configured Timeout Request value in the CFAM Settings screen.
 - Long-running page timings are recorded in the `server.log`.
- Use `<cftrace>` sparingly for audit trails.
 - Use with `inline="false"`.
 - Use it to track user input: Form and/or URL variables.

Best Practices in Action

The following code adds error handling and logging to the `dbLogin` and `logout` methods in the code from Authentication section.

```
<cffunction name="dblogin" access="private" output="false"
    returntype="struct">
    <cfargument name="strUserName" required="true" type="string">
    <cfargument name="strPassword" required="true" type="string">
    <cfset var retargs = StructNew()>
    <cftry>
        <cfif IsValid("regex", uUserName, "[A-Za-z0-9%]*") AND
            IsValid("regex", uPassword, "[A-Za-z0-9%]*")>
            <cfquery name="loginQuery" dataSource="#Application.DB#" >
                SELECT hashed_password, salt
                FROM UserTable
                WHERE UserName =
            <cfqueryparam value="#strUserName#"
                cfsqltype="CF_SQL_VARCHAR" maxlength="25">
            </cfquery>
```

```

        <cfif loginQuery.hashed_password EQ Hash(strPassword &
loginQuery.salt, "SHA-256" )>
        <cfset retargs.authenticated="YES">
        <cfset Session.UserName = strUserName>
        <cflog text="#getAuthUser()# has logged in!"
            type="Information"
            file="access"
            application="yes">
        <!-- Add code to get roles from database -->
        <cfelse>
        <cfset retargs.authenticated="NO">
        </cfif>
    <cfelse>
        <cfset retargs.authenticated="NO">
    </cfif>
    <cfcatch type="database">
        <cflog text="Error in dbLogin(). #cfcatch.details#"
            type="Error"
            log="Application"
            application="yes">
        <cfset retargs.authenticated="NO">
        <cfreturn retargs>
    </cfcatch>
</cftry>
<cfreturn retargs>
</cffunction>
...
<cffunction name="logout" access="remote" output="true">
    <cfargument name="logintype" type="string" required="yes">
    <cfif isDefined("form.logout")>
        <cflogout>
        <cfset StructClear(Session)>
        <cflog text="#getAuthUser()# has been logged out."
            type="Information"
            file="access"
            application="yes">
        <cfif arguments.logintype eq "challenge">
        <cfset foo = closeBrowser()>
        <cfelse>
        <!-- replace this URL to a page logged out users should see --
        -->
        <cflocation url="login.cfm">
        </cfif>
    </cfif>
</cffunction>

```

File System

Even with an authentication system in place to protect your content, if file permissions are set incorrectly, an attacker could browse directly to your application source code or protected documents. The section below gives guidance in setting file system permissions and directories to reduce your risk of exposure.

Best Practice

File Permissions:

- Restrict access of the following subdirectories under the \CFIDE directory to specific IP addresses and/or user groups/accounts:
 - adminapi
 - administrator
 - componentutils
 - wizards
- Remove the \cfdocs directory. Sample applications are installed by default in the cfdocs directory and are accessible to anyone. These applications should never be available on a production server.
- Ensure that directory browsing is disabled.
- Ensure that proper access controls are set on web application content. The following settings assume a user account called "cfuser" has been created to run the ColdFusion service. In addition, if you are using a directory or operating system authentication service these setting may need to be adjusted.
 - File types: Scripts (.cfm, .cfml, .cfc, .cfr, .jsp, .swf and others):
ACLs: cfuser (Execute); Administrators (Full)
 - File types: Static content (.txt, .gif, .jpg, .html, .xml):
ACLs: cfuser (Read); Administrators (Full)

File Upload:

- Upload files to a destination outside of the web application directory.
- Enable virus scan on the destination directory.
- Do not allow user input to specify the destination directory or file name of uploaded documents.

Cryptography

The following section describes the ColdFusion's cryptography features. ColdFusion MX leverages the Java Cryptography Extension (JCE) of the underlying J2EE platform for cryptography and random number generation. It provides functions for symmetric (or private-key) encryption. While it does not provide native functionality for public-key (asymmetric) encryption, it does use the Java Secure Socket Extension (JSSE) for SSL communication.

Pseudo-Random Number Generation

ColdFusion provides three functions for random number generation: `rand()`, `randomize()`, and `randRange()`. Function descriptions and syntax:

Rand: Use to generate a pseudo-random number:

```
rand([algorithm])
```

Randomize: Use to seed the pseudo-random number generator (PRNG) with an integer:

```
randomize(number [, algorithm])
```

RandRange: Use to generate a pseudo-random integer within the range of the specified numbers:

```
randrange(number1, number2 [, algorithm])
```

The following values are the allowed algorithm parametersⁱⁱⁱ:

- CFMX_COMPAT (default): Invokes java.util.rand.
- SHA1PRNG (recommended): Invokes java.security.SecureRandom using the Sun Java SHA1 PRNG algorithm.
- IBMSecureRandom: IBM WebSphere's JVM does not support the SHA1PRNG algorithm.

Symmetric Encryption

ColdFusion MX 7 provides six encryption functions: `decrypt()`, `decryptBinary()`, `encrypt()`, `encryptBinary()`, `generateSecretKey()`, and `hash()`. Function descriptions and syntax:

Decrypt: Use to decrypt encrypted strings with specified key, algorithm, encoding, initialization vector or salt, and iterations.

```
decrypt(encrypted_string, key[, algorithm[, encoding[, IVorSalt[, iterations]]]])
```

DecryptBinary: Use to decrypt encrypted binary data with specified key, algorithm, initialization vector or salt, and iterations.

```
decryptBinary(bytes, key[, algorithm[, IVorSalt[, iterations]]])
```

Encrypt: Use to encrypt string using specific algorithm, encoding, initialization vector or salt, and iterations

```
encrypt(string, key[, algorithm[, encoding[, IVorSalt[, iterations]]]])
```

EncryptBinary: Use to encrypt binary data with specified key, algorithm, initialization vector or salt, and iterations.

```
encryptBinary(bytes, key[, algorithm[, IVorSalt[, iterations]]])
```

GenerateSecretKey: Use to generate a secure key using the specified algorithm for the `encrypt` and `encryptBinary` functions.

```
generateSecretKey(algorithm)
```

Hash: Use for one-way conversion of a variable-length string to fixed-length string using the specified algorithm and encoding.

```
hash(string[, algorithm[, encoding]] )
```

ColdFusion offers the following default algorithms for these functions^{iv}:

- CFMX_COMPAT: the algorithm used in ColdFusion MX and prior releases. This algorithm is the least secure option (default).

- AES: the Advanced Encryption Standard specified by the National Institute of Standards and Technology (NIST) FIPS-197. (recommended)
- BLOWFISH: the Blowfish algorithm defined by Bruce Schneier.
- DES: the Data Encryption Standard algorithm defined by NIST FIPS-46-3.
- DESEDE: the "Triple DES" algorithm defined by NIST FIPS-46-3.
- PBEWithMD5AndDES: A password-based version of the DES algorithm that uses a MD5 hash of the specified password as the encryption key.
- PBEWithMD5AndTripleDES: A password-based version of the DESEDE algorithm that uses a MD5 hash of the specified password as the encryption key.

The following algorithms are provided by default for the hash() function. Note, SHA algorithms used in ColdFusion are NIST FIPS-180-2 compliant^v:

- CFMX_COMPAT: Generates a MD5 hash string identical to that generated by ColdFusion MX and ColdFusion MX 6.1 (default).
- MD5: Generates a 128-bit digest.
- SHA: Generates a 160-bit digest. (SHA-1)
- SHA-256: Generates a 256-bit digest
- SHA-384: Generates a 384-bit digest
- SHA-512: Generates a 512-bit digest

Pluggable Encryption

ColdFusion MX 7 introduced pluggable encryption for CFML. The JCE allows developers to specify multiple cryptographic service providers. ColdFusion can leverage the algorithms, feedback modes, and padding methods of third-party Java security providers to strengthen its cryptography functions. For example, ColdFusion can leverage the Bouncy Castle (<http://www.bouncycastle.org/>) crypto package and use the SHA-224 algorithm for the hash() function or the Serpent block encryption for the encrypt() function.

See Macromedia's Strong Encryption in ColdFusion MX 7 technote for information on installing additional security providers for ColdFusion at <http://www.macromedia.com/go/e546373d>.

SSL

ColdFusion does not provide tags and functions for public-key encryption, but it can communicate over Secure Sockets Layer (SSL). ColdFusion leverages the Sun JSSE to communicate over SSL with remote servers. ColdFusion uses the Java certificate database (jre_root/lib/security/cacerts) to store server certificates. It compares presented certificates of remote systems to those stored in the database. It also grabs the host system's certificate from this database and uses it to present to remote systems to initiate the SSL handshake. Certificate information is then exposed as CGI variables.

Best Practices

- Enable `/dev/urandom` for higher entropy for random number generation.
- Call the `randomize()` function before calling `rand()` or `randRange()` to seed the random number generator.
- Do *not* use the `CFMX_COMPAT` algorithms. Upgrade your application to use stronger cryptographic ciphers.
- Use AES or higher for symmetric encryption.
- Use SHA-256 or higher for the hash function.
- Use a salt (or random string) for password generation with the hash function.
- Always use `generateSecretKey()` to generate keys of the appropriate length for Block Encryption algorithms unless a customized key is required.
- Use separate key databases to store remote server certificates separately from the ColdFusion server's certificate.

Configuration

The following section describes some of the server-wide security-related options available to a ColdFusion administrator via the ColdFusion MX 7 Administrator console web application (<http://servername:port/CFIDE/administrator/index.cfm>). If the console application is unavailable, you can modify these options by editing the XML files in the `cf_root/lib/` (Server configuration) or `cf_web_root/WEB-INF/cfusion/lib` (J2EE configuration) directory; however, editing these files directly is not recommended.

Best Practices^{vi}

CF Admin Password screen:

- Enable a strong Administrator password:
The ColdFusion Administrator is the default interface for configuring the ColdFusion application server. It is secured by a single password. Ensure that the Administrator security is enabled and the password is strong and stored in a secure place.
- 6** Make sure the checkbox is filled.
 - 7** Enter and confirm a strong password string of eight characters or more.
 - 8** Click Submit Changes.

Sandbox Security screen:

- Enable Sandbox Security:
The ColdFusion Sandbox enables you to place access security restrictions on files, directories, methods, and data sources. Sandboxes make the most sense for a hosting provider or corporate intranet where multiple applications share the same server. Enable this option.

Next, a sandbox needs to be configured, because if not, all code in all directories will execute without restriction. Code in a directory and its subdirectories inherits the access controls defined for the sandbox. For example, if ABC Company creates multiple applications within their directory, all applications will have the same permissions as the parent. A sandbox applied to ABC-apps will apply to app1 and app2. Here is a sample directory structure:

```
D:\inetpub\wwwroot\ABC-apps\app1
```

```
D:\inetpub\wwwroot\ABC-apps\app2
```

Note: if a new sandbox is created for app2, then it will not inherit settings from ABC-apps.

Sandbox security configurations are application-specific; however, you should follow these general guidelines:

- Create a default restricted sandbox and copy its settings to each subsequent sandbox, removing restrictions as needed by the application, except in the case of files or directories where access is granted rather than restricted.
 - Restrict access to data sources that should not be accessed by the sandboxed application.
 - Restrict access to powerful tags; for example, CFREGISTRY and CFEXECUTE.
- Restrict file and directory access to limit the ability of tags and functions to perform actions to specified paths.
- **Every** application should have a sandbox.
- In multi-homed environments disable Java Server Pages (JSP), as ColdFusion is unable to restrict the functionality of the underlying Java server.

RDS Password screen:

- Enable a strong RDS password:

Developers can access ColdFusion resources (files and data sources) over HTTP from Macromedia Dreamweaver MX and HomeSite+ through ColdFusion's Remote Development Services (RDS). This feature is password-protected and should only be enabled in secure development environments.

 - a. Make sure the checkbox is filled.
 - b. Enter and confirm a strong password string of eight characters or more.
 - c. Click Submit Changes.
- Use RDS over SSL:

During development, you should use SSL to encrypt all RDS communications between Dreamweaver MX and the ColdFusion server. This includes remote access to server data sources and drives, provided that both are accessed through RDS.

- **Disable RDS in production:**
In production environments, you should not use RDS. In earlier versions of ColdFusion, RDS ran as a separate service or process and could be disabled by disabling the service. In ColdFusion MX, RDS is integrated into the main service. To disable it, you must disable the RDSServlet mapping in the web.xml file. The following procedure assumes that ColdFusion is installed in the default location.

- a. Back up the C:\CFusionMX7\wwwroot\WEB-INF\web.xml file.
- b. Open the web.xml file for editing.
- c. Comment out the RDSServlet mapping, as follows:

```
<!--  
<servlet-mapping>  
<servlet-name>RDSServlet</servlet-name>  
<url-pattern>/CFIDE/main/ide.cfm</url-pattern>  
</servlet-mapping>  
-->
```

- d. Save the file.
- e. Restart ColdFusion.

Settings Screen:

- **Enable a Request Timeout:**
ColdFusion processes requests simultaneously and queues all requests above the configured maximum number of simultaneous requests. If requests run abnormally long, this can tie up server resources and lead to DoS attacks. This setting will terminate requests when the configured timeout is reached.

- a. Fill the checkbox next to "Timeout Request after (seconds)".
- b. Enter the number of seconds for ColdFusion to allow threads to run.

To allow a valid template request to run beyond the configured timeout, place a `<cfsetting>` tag atop the base ColdFusion template and configure the `RequestTimeout` attribute for the necessary amount of time (in seconds).

- **Use UUID for cftoken:**

Best practice calls for J2EE session management. In the event that only ColdFusion session management is available, strong security identifiers must be used. Enable this setting to change the default eight-character CFToken security token string to a UUID.

- **Enable Global Script Protection:** This is a new security feature in ColdFusion MX 7 that isn't available in other web application platforms. It helps protect Form, URL, CGI, and Cookie scope variables from cross-site scripting attacks.
- **Specify a site-wide error handler:**
Prevent information leaks through verbose error messages. Specifying a site-wide error handler covers you when `<cftry>` and `<cfcatch>` are not used. This page should be a generic error message that you return to the user. Also, if the error handler displays user-input, it should be reviewed for potential cross-site scripting issues.

- Specify a missing template handler:
Provide a custom message page for HTTP 404 errors when the server cannot find the requested ColdFusion template.
- Configure a memory throttling:
To prevent file upload DoS attacks, Macromedia added new configuration settings to ColdFusion MX 7.0.1 that enable administrators to restrict the total upload size of HTTP POST operations. Configure these settings accordingly.
 - maximum size for post data:
This is the total size that ColdFusion will accept for any single HTTP POST request (including file uploads). ColdFusion will reject any request whose Content-size header exceeds this setting.
 - Request Throttle Threshold:
HTTP POST requests larger than this setting (default is 4MB) are included in the total concurrent request memory size and get queued if they exceed the Request Throttle Memory setting.
 - Request Throttle Memory:
This sets the total amount of memory (MB) ColdFusion reserves for concurrent HTTP POST requests. Any requests exceeding this limit are queued until enough memory is available.

Memory Variables screen:

- Enable J2EE session management and use J2EE session variables:
Best practice requires J2EE sessions because they are more secure than regular ColdFusion sessions. (See the [Session Management](#) section of this document.)
 - a. Select checkbox next to "Enable Session Variables".
 - b. Select checkbox next to "Enable J2EE session variables".
- Set the maximum session timeout to 20 minutes to limit the window of opportunity for session hijacking.
- Set the default session timeout to 20 minutes to limit the window of opportunity for session hijacking. (The default value is 20 minutes.)
- The session-timeout parameter in the cf_root/WEB-INF/web.xml file establishes the maximum J2EE session timeout. This setting should always be greater than or equal to ColdFusion's Maximum Session Timeout value.
- Set the maximum application timeout to 24 hours.
- Set the default application timeout to eight hours.

Data Sources screen

- Do not use an administrative account to connect ColdFusion to a data source. For example, do not use the *sa* account to connect to a MS SQL Server. The account accessing the database should be granted specific privileges to the objects it needs to access. In addition, the account created to connect to the database should be an OS-based account, not a SQL account. Operating system accounts have many more auditing, password, and other security controls associated with them. For example, account lockouts and password complexity requirements are built into the Windows operating system; however, a database would need custom code to handle these security-related tasks.
- Disable the following Allowed SQL options for all data sources:
 - Create
 - Drop
 - Grant
 - Revoke
 - Alter

As an administrator, you do not have control over what a developer sends to the database; however, there should be no circumstance where the previous commands need to be sent to a database from a web application.

Debugging Settings screen:

- Disable Robust Exception Information for production servers. (default)
- Disable Debugging for production servers. (default)

Debugging IP Addresses:

- Make sure that only the addresses of trusted clients are in the IP list.
- Allow only the localhost IP (127.0.0.1) in the list on production machines.

Mail screen:

- Require a user name and password to authenticate to your mail server.
- Set the connection timeout to 60 seconds (The default value is 60 seconds.)

Maintenance

This section provides guidance on maintaining ColdFusion MX. Even with securely coded applications, developers and hackers may find security flaws in the ColdFusion engine itself. Adobe routinely performs security checks and responds to customer-reported security incidents. The company provides software releases to address identified flaws and publishes security bulletins and technical briefs to provide customer notification of the issues and fixes.

The following is a partial list of software release types supported and tested by Adobe^{vii}.

Type	Description	Delivery
------	-------------	----------

Hot Fix	Fixes a specific problem that has been escalated through support or customer service. Requires a specific code base in order to apply. It is not guaranteed that any two Hot Fixes will work together properly.	Electronic delivery. Distributed by Support, Engineering Escalation Team (EET), or another Macromedia group.
Security Update	Fixes a specific security issue. Requires a specific code base in order to apply.	Electronic delivery. Distributed by Support or Engineering Escalation Team (EET) on the Security Zone.
Updaters	Includes all applicable Hot Fixes and Security Fixes to date. May include additional bug fixes. May include additional updates to drivers, databases, platform support, or other initiatives. Requires a specific code base in order to apply.	Electronic delivery. Usually posted on the product's Support Center.

Security updates for all Adobe software can be downloaded at <http://www.adobe.com/devnet/security/>. Find a list of the latest product updates for all Adobe products at <http://www.adobe.com/downloads/updates/>.

Best practices

- Use the Adobe Security Topic Center at: <http://www.adobe.com/devnet/security/>.
 - Subscribe to Adobe Security Notification Services at <http://www.adobe.com/cfusion/entitlement/index.cfm?e=szalert>.
 - Read the published Adobe Security Bulletins at <http://www.adobe.com/support/security/>.
 - Only implement the solutions provided in any security bulletin that are applicable to your environment.
 - Immediately notify Adobe if you find a bug or security vulnerability.
 - For security vulnerabilities use <http://www.adobe.com/support/security/alertus.html>.
 - For software bugs use <http://www.adobe.com/cfusion/mmform/index.cfm?name=wishform>.
- Utilize the Adobe ColdFusion Support Center.
 - Download and apply the latest ColdFusion updates.
 - Get ColdFusion Updaters and Hot Fixes from http://www.adobe.com/support/coldfusion/downloads_updates.html.
 - Only install any relevant hot fixes that are not already included in the latest ColdFusion Updater.
 - Search the ColdFusion Support Center for updated information from ColdFusion technical support
 - ColdFusion Technote Index: <http://www.adobe.com/support/coldfusion/>.
 - Read the ColdFusion documentation: <http://www.adobe.com/support/documentation/en/coldfusion/>

- Release notes contain a list of identified issues and bug fixes in each ColdFusion software release. The ColdFusion release notes are posted at <http://www.adobe.com/support/documentation/en/coldfusion/releases.html>
- The Adobe LiveDocs provides an online version the ColdFusion MX 7 manuals (with customer comments): <http://livedocs.adobe.com/coldfusion/7/index.html>
- Ensure that the platform on which ColdFusion is running has the latest stable patches. This includes the operating system and web server.

References

Developing secure applications is a large subject that has only been touched upon here. The following list of references should be consulted to help you administer and secure ColdFusion MX server and applications:

- Forta, Ben. *Advanced Macromedia ColdFusion MX 7 Application Development*. Berkley, CA: Macromedia Press, 2005.
- Lee, Erick. "Configuring ColdFusion MX 7 Server Security" Macromedia Inc. 3 November 2005 <http://www.adobe.com/devnet/coldfusion/articles/cf7_security.html>

ⁱ Adobe ColdFusion MX 7 Livedocs.

http://livedocs.adobe.com/coldfusion/7/htmldocs/wwhelp/wwhimpl/common/html/wwhelp.htm?context=ColdFusion_Documentation&file=00001131.htm

ⁱⁱ *Ibid.*

<http://livedocs.adobe.com/coldfusion/7/htmldocs/00001719.htm>

ⁱⁱⁱ *Ibid.*

http://livedocs.adobe.com/coldfusion/7/htmldocs/wwhelp/wwhimpl/common/html/wwhelp.htm?context=ColdFusion_Documentation&file=00001719.htm

^{iv} *Ibid.*

http://livedocs.adobe.com/coldfusion/7/htmldocs/wwhelp/wwhimpl/common/html/wwhelp.htm?context=ColdFusion_Documentation&file=00000452.htm

^v *Ibid.*

http://livedocs.adobe.com/coldfusion/7/htmldocs/wwhelp/wwhimpl/common/html/wwhelp.htm?context=ColdFusion_Documentation&file=00000503.htm

^{vi} Configuring ColdFusion MX 7 Server Security.

http://www.adobe.com/devnet/coldfusion/articles/cf7_security.html

^{vii} Adobe Software – Release Terminology.

<http://www.adobe.com/support/updaters/terms.html>