Adobe White Paper

ColdFusion 8 developer security guidelines

Erick Lee, Ian Melven, and Sarge Sargent

October 2007

Adobe Systems Incorporated
345 Park Avenue
San Jose, CA
95110-2704
USA
www.adobe.com

Contents

Adobe 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, Adobe Flash and Adobe Flex application connectivity, web services publishing, 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 (Adobe 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. This whitepaper will 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, use 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, and 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.

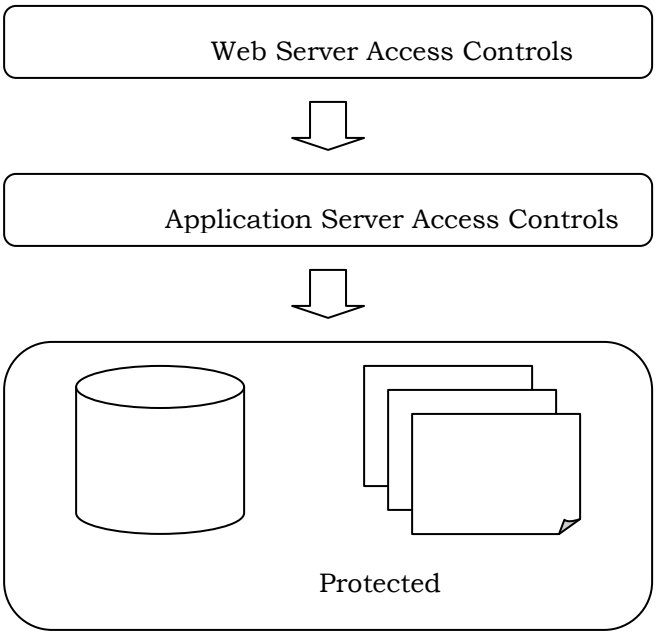

Figure 1: *Web server and application server authentication occur in sequence before access is granted to protected resources.*

ColdFusion enables you to authenticate against multiple system types. These types include LDAP, text files, Databases, NTLM, Client-Side certificates via LDAP, 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 parameters (`<cfprocparam>`).
- Database connections should be 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')>
      <cfset onSessionEnd(SESSION) />
    </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 = loginQuery.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="nusername" 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.nusername, "[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.nusername#"
      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>
  <!--- return role here --->
  <cfreturn retargs />
</cffunction>
```

LDAP

To set up authentication against an LDAP (Lightweight Directory Access Protocol) 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"
   throwontimeout="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>
```

Authenticating to another server

ColdFusion has a variety of tags that connect to another server, for example, another HTTP server or a Microsoft Exchange Server. Many of these tags require that the user specify the login credentials to authenticate to the server as an attribute to the tag. If an attacker can obtain the source to the application via some method or can download the generated class files via some method, these plain-text passwords can be recovered.

Best practices

If you need to use static passwords, put them inside an initialization file (such as settings.ini) in a secure filesystem location and use the `getProfileString()` function to retrieve the password. The file system location used to store initialization files should *not* be under the web root, this removes the possibility of these files being accessed remotely.

For example : getProfileString("C:\secured\admin.ini", "Admin_Pass", "password").

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.

Best practices

- If your application allows users to be logged in for long periods of time, ensure that controls are in place to always revalidate a user's authorization to a sensitive 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.
- ColdFusion 8 provides three new functions to help implement authentication in your application :
    - The `GetUserRoles` function – this function returns a list of roles for the current user. Only ColdFusion roles are returned, not roles set for the servlet API.
    - The `IsUserIsAnyRole` function – this function determines whether the user has any roles at all. It checks the roles set up by the `cflogin` tag and by the servlet API
    - The `IsUserLoggedIn` function – this function returns true if the user is currently logged into the ColdFusion security framework via `cfloginuser`, otherwise it returns false.

Note : Role names are NOT case sensitive.

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 = true />
      <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 = false />
     </cfif>
   <cfelse>
     <cfset retargs.authenticated = false />
   </cfif>
   <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. This 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, as a web service, or from Flash/Flex.

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 variable This.sessionManagement to true in Application.cfc.

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: *CFID=cfid_number&CFTOKEN=cftoken_num*
Session.SessionID	A combination of the application name and the session token that uniquely identifies the session: *applicationName_cfid_cftoken*

Pros:

- It is compatible with all versions of ColdFusion.
- It uses 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 JESSIONID as the session identifier. ColdFusion sends JESSIONID 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 or when the browser closes.

Table 2: *Default session scope variables:*

Variable	Description
Session.URLToken	A combination of the CFID, CFToken, and JESSIONID in URL format: *CFID=cfid_number&CFTOKEN=cftoken_num&JESSIONID=SessionID*
Session.SessionID	The JESSIONID value that uniquely identifies the session.

Pros:

- ColdFusion sessions data is shareable with JSP pages and Java servlets.
- It uses a session-specific JESSIONID 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.
- JESSIONID 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)`
- 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/go/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/go/tn_18133
- Avoid passing session identifiers (CFID/CFToken or JESSIONID) 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 8 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 maximum memory (heap) 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.
- Use `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 tag 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 validate dynamic SQL variables against database datatypes.
- Use CFLDAP for accessing LDAP servers. Avoid allowing native JNDI calls to connect to LDAP.

Best Practices 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 = true />
      <cfset Session.UserName = strUserName />
      <!-- Add code to get roles from database -->
      <cfelse>
      <cfset retargs.authenticated = false />
    </cfif>
  <cfelse>
    <cfset retargs.authenticated = false />
  </cfif>
  <cfreturn retargs />
</cffunction>
```

Ajax

ColdFusion 8 provides new functions and tags to aid in the development of Ajax (Asynchronous JavaScript and XML) applications. Often in Ajax applications data is transferred between the client and server using JSON (Javascript Object Notation). Coldfusion 8 also provides functionality to prevent JSON hijacking, an attack in which an attacker can modify JSON received by the client to execute arbitrary Javascript. Ajax can also be vulnerable to cross-site scripting attacks and ColdFusion 8 also includes safeguards to mitigate the risk of exposure to these attacks. ColdFusion 8 also allows creation of Javascript proxies to CFCs through the `cfajaxproxy` tag.

Best practices

- To prevent cross-site scripting, you cannot use remote URLs in code that execute on the client. For example, if you use a URL such as http://www.myco.com/mypage.cfm in a `cfwindow` tag source attribute, the remote page does not load in the window and the window shows an error message. If you must access remote URLs, you must do so in CFML code that executes on the server, for example, by using a `cfhttp` tag on the page specified by a source attribute.
- When a CFC function returns remote data in JSON format, by default, the data is sent without any prefix or wrapper. To help prevent cross-site scripting attacks where the attacker accesses the JSON data, you can tell ColdFusion to prefix the returned data with one or more characters. You can specify this behavior in several ways.

  The value of an item in the following list is determined by the preceding item in this list:

  a. In the Administrator, enable the Prefix Serialized JSON option on Server Settings > Settings page (the default value is false). You can also use this setting to specify the prefix characters. The default prefix is `//`, which is the JavaScript comment marker that turns the returned JSON code into a comment from the browser's perspective. The `//` prefix helps prevent security breaches because it prevents the browser from converting the returned value to the equivalent JavaScript objects

  b. Set the Application.cfc file `This.secureJSON` and `This.secureJSONPrefix` variable values, or set the `cfapplication` tag `secureJSON` and `secureJSONPrefix attributes`.

  c. Set the `cffunction` tag `secureJSON` attribute. (You cannot use the `cffunction` tag to set the prefix.)

  As a general rule, you should use one of these techniques for any CFC or CFML page that returns sensitive data, such as credit card numbers.

  When you use any of these techniques, the ColdFusion Ajax elements that call CFC functions, including bind expressions and the CFC proxies created by the `cfajaxproxy` tag, automatically remove the security prefix when appropriate. You do not have to modify your client-side code.

- ColdFusion provides capabilities that help prevent security attacks where an unauthorized party attempts to perform an action on the server, such as changing a password. You can use the following techniques to help make sure that a request to a CFML page or remote CFC function comes from a ColdFusion Ajax feature, such as a bind expression or CFC proxy, that is a valid part of your application:

  In the `cffunction` tag in a CFC function that returns data to an ColdFusion Ajax client (such as Ajax client code written using the ColdFuson Ajax tags), specify a `verifyClient` attribute with a value of yes.

  At the top of a CFML page or function that can be requested by a ColdFusion Ajax client, call the `VerifyClient` ColdFusion function. This function takes no parameters.

  The `VerifyClient` function and attribute tell ColdFusion to require an encrypted security token in each request. To use this function, you must enable client management or session management in your application; otherwise, you do not get an error, but ColdFusion does not verify clients.

  Enable client verification only for code that responds to ColdFusion Ajax client code, because only the ColdFusion Ajax library contains the client-side support code. Enabling client verification for clients other than ColdFusion Ajax applications can result in the client application not running.

  As a general rule, use this function for Ajax requests to the server to perform sensitive actions, such as updating passwords. You should typically not enable client verification for public APIs that do not need protection, for example, search engine web services. Also, do not enable client verification for the top-level page of an application, because the security token is not available when the user enters a URL in the browser address bar.

- ColdFusion 8 provides the `IsJSON` function to verify data follows correct JSON syntax. Use this function to validate JSON data is legitimate and correctly formed before passing it to the `DeserializeJSON` function or other functions that expect to be passed valid JSON data.

- The `deserializeJSON` function takes a parameter called `strictmapping`. By default, this parameter is true. Setting `strictmapping` to true stops the possibility of JSON object being converted to ColdFusion queries, which may result in unexpected or undesired behavior.

- By default, the `cfajaxproxy` tag uses GET to make Ajax calls. GET passes parameters directly in the URL which can lead to disclosure of information as well as malicious modification of data. You should use the `SetHttpMethod` function to specify POST as the HTTP method to use to make AJAX calls, particularly when sensitive data is being transferred. This is true for all AJAX calls as well as for usage of the `cfajaxproxy` tag.

Best practices in action

The following example shows using the `cfajaxproxy` tag to automatically create a javascript proxy to a CFC. Note using the `setHTTPMethod` to choose POST as the method of communicating with the server.

```
<!--- The cfajaxproxy tag creates a client-side proxy for the emp
   CFC.
View the generated page source to see the resulting JavaScript.
The emp CFC is in the components subdirectory of the directory that
   contains this page. --->
<cfajaxproxy cfc="components.emp" jsclassname="emp">
<html>
  <head>
    <script type="text/javascript">
    // Function to find the index in an array of the first entry
    // with a specific value.
    // It is used to get the index of a column in the column list.
      Array.prototype.findIdx = function(value) {
        for (var i=0; i < this.length; i++) {
          if (this[i] == value) {
            return i;
          }
        }
      }

    // Use an asynchronous call to get the employees for the
    // drop-down employee list from the ColdFusion server.

    var getEmployees = function() {

      // create an instance of the proxy.

      var e = new emp();

      // Setting a callback handler for the proxy automatically makes
      // the proxy's calls asynchronous.

      e.setHTTPMethod("POST");
      e.setCallbackHandler(populateEmployees);
      e.setErrorHandler(myErrorHandler);

      // The proxy getEmployees function represents the CFC
      // getEmployees function.

      e.getEmployees();

    }

    // Callback function to handle the results returned by the
    // getEmployees function and populate the drop-down list.

    var populateEmployees = function(res) {
      with(document.simpleAJAX){
        var option = new Option();
        option.text='Select Employee';
        option.value='0';
        employee.options[0] = option;
        for(i=0;i<res.DATA.length;i++){
          var option = new Option();
```

```
            option.text=res.DATA[i][res.COLUMNS.findIdx('FIRSTNAME')]
            + ' ' + res.DATA[i][[res.COLUMNS.findIdx('LASTNAME')]];
            option.value=res.DATA[i][res.COLUMNS.findIdx('EMP_ID')];
            employee.options[i+1] = option;
          }
        }
      }

      // Use an asynchronous call to get the employee details.
      // The function is called when the user selects an employee.

        var getEmployeeDetails = function(id) {
          var e = new emp();
          e.setCallbackHandler(populateEmployeeDetails);
          e.setErrorHandler(myErrorHandler);
          e.setHTTPMethod("POST");

          // This time, pass the employee name to the getEmployees CFC
          // function.

          e.getEmployees(id);
        }

      // Callback function to display the results of the
      // getEmployeeDetails function.
      var populateEmployeeDetails = function(employee) {
        var eId = employee.DATA[0][0];
        var efname = employee.DATA[0][1];
        var elname = employee.DATA[0][2];
        var eemail = employee.DATA[0][3];
        var ephone = employee.DATA[0][4];
        var edepartment = employee.DATA[0][5];
        with(document.simpleAJAX){
          empData.innerHTML =
          '<span style="width:100px">Employee Id:</span>'
          + '<font color="green"><span align="left">'
          + eId + '</font></span><br>'
          + '<span style="width:100px">First Name:</span>'
          + '<font color="green"><span align="left">'
          + efname + '</font></span><br>'
          + '<span style="width:100px">Last Name:</span>'
          + '<font color="green"><span align="left">'
          + elname + '</font></span><br>'
          + '<span style="width:100px">Email:</span>'
          + '<font color="green"><span align="left">'
          + eemail + '</span></font><br>'
          + '<span style="width:100px">Phone:</span>'
          + '<font color="green"><span align="left">'
          + ephone + '</font></span><br>'
          + '<span style="width:100px">Department:</span>'
          + '<font color="green"><span align="left">'
          + edepartment + '</font></span>';
        }
      }

      // Error handler for the asynchronous functions.
      var myErrorHandler = function(statusCode, statusMsg) {
        alert('Status: ' + statusCode + ', ' + statusMsg);
      }
    </script>
    </head>
```

```
  <body>
    <!--- The form to display the employee drop-down list and
    employee data. --->
    <form name="simpleAJAX" method="get">
      List of Employees:   
      <select name="employee"
    onChange="getEmployeeDetails(this.value)">
        <script language="javascript">
          getEmployees();
        </script>
      </select>
    <br><br>
      <span id="empData"></span>
    </form>
  </body>
</html>
```

The following component, which gets the data from the data source, must be in a file named emp.cfc in the component's subdirectory of the application directory.

The CFC uses the `cfdocexamples` data source that is installed with ColdFusion 8, if you install the documentation and sample applications.

```
<cfcomponent>
  <cfset this.dsn = "cfdocexamples">
  <cffunction name="getEmployees" access="remote" secureJSON = “true”
    verifyClient = “yes” returnFormat="json" output="false">
     <cfargument name="empid" required="no" type="string" default="0"
    />

     <cfquery name="qryEmp" datasource="#this.dsn#">
       select * from Employees
       <cfif arguments.empid NEQ 0>
         where Emp_ID = #empid#
       </cfif>
     </cfquery>
     <cfreturn qryEmp />
  </cffunction>
</cfcomponent>
```

The following example shows using the `IsJSON` function to verify returned data is legitimate JSON formatted data before deserializing it :

```
<!--- Get the JSON Feed --->
<cfhttp
   url="http://localhost:8500/My_Stuff/Ajax/Books/CreateJSON_NEW.cfm"
   />

<!--- JSON data is sometimes distributed as a JavaScript function.
The following REReplace functions strip the function wrapper. --->
<cfset theData=REReplace(cfhttp.FileContent,
   "^\s*[[:word:]]*\s*\(\s*","") />
<cfset theData=REReplace(theData, "\s*\)\s*$", "") />

<!--- Test to make sure you have JSON data. --->
```

```
<cfif ! IsJSON(theData)>
  <h3>The URL you requested does not provide valid JSON</h3>
  <cfdump var="#theData#">
<cfelse>
 <!--- If the data is in JSON format, deserialize it. --->
  <cfset cfData=DeserializeJSON(theData) />

 <!--- Parse the resulting array or structure and display the data.
  In this case, the data represents a ColdFusion query that has been
  serialized by the SerializeJSON function into a JSON structure with
  two arrays: an array column names, and an array of arrays,
  where the outer array rows correspond to the query rows, and the
  inner array entries correspond to the column fields in the row. ---
    >

  <!--- First, find the positions of the columns in the data array. -
    -->
  <cfset colList=ArrayToList(cfData.COLUMNS) />
  <cfset cityIdx=ListFind(colList, "City") />
  <cfset tempIdx=ListFind(colList, "Temp") />
  <cfset fcstIdx=ListFind(colList, "Forecasts") />
  <!--- Now iterate through the DATA array and display the data. --->
  <cfoutput>
    <cfloop index="i" from="1" to="#ArrayLen(cfData.DATA)#">
      <h3>#cfData.DATA[i][cityIdx]#</h3>
      Current Temperature: #cfData.DATA[i][tempIdx]#<br><br>
      <b>Forecasts</b><br><br>
      <cfloop index="j" from="1"
   to="#ArrayLen(cfData.DATA[i][fcstIdx])#">
        ADOBE COLDFUSION 8
        <b>Day #j#</b><br>
        Outlook: #cfData.DATA[i][fcstIdx][j].WEATHER#<br>
        High: #cfData.DATA[i][fcstIdx][j].HIGH#<br>
        Low: #cfData.DATA[i][fcstIdx][j].LOW#<br><br>
      </cfloop>
    </cfloop>
  </cfoutput>
</cfif>
```

PDF integration

ColdFusion 8 features new functions and tags for working with PDF files. PDFs can be created and manipulated in a variety of ways, including protecting them with a password and encrypting the document.

There are two kinds of passwords for PDF objects in ColdFusion:

ColdFusion password	Acrobat equivalent	Description
User password	Document Open password, user password	Anyone who tries to open the PDF document must enter the password that you specify. A user password does not allow a user to change restricted features in the PDF document.

Owner password	Permissions password, master password	Lets the person who enters the password restrict access to features in a PDF document.

The following table lists the permissions an owner can set for PDF documents:

Permissions	**Description**
All	There are no restrictions on the PDF document.
AllowAssembly	Users can add the PDF document to a merged document
AllowCopy	Users can copy text, images and other file content. This setting is required to generate thumbnail images with the "thumbnail" action of the `cfpdf` tag.
AllowDegradedPrinting	Users can print the document at low-resolution (150 DPI).
AllowFillIn	Users can enter data into PDF form fields. Users can sign PDF forms electronically.
AllowModifyAnnotations	Users can add or change comments in the PDF document.
AllowModifyContents	Users can change the file content. Users can add the PDF document to a merged document.
AllowPrinting	Users can print the document at high-resolution (print-production quality). This setting is required for use with the `cfprint` tag.
AllowScreenReaders	Users can extract content from the PDF document.
AllowSecure	Users can sign the PDF document (with an electronic signature).
None	Users can view the document only.

When you protect a PDF, ColdFusion updates the variable's saved password to the one you provide. However, if you provide both passwords, ColdFusion uses the owner password.

To set the permissions on the output file, you must set the `newOwnerPassword`. A user who enters the `ownerpassword` when accessing the PDF file is considered the owner of file. The following example shows how to set a new owner password:

```
<cfpdf action="protect" encrypt="AES_128"source="Book.pdf"
   destination="MysteryBook.pdf" overwrite="yes"
   newOwnerPassword="pssst" permissions="AllowDegradedPrinting">
```

Best practices

- The default encryption used by the "protect" action of the `cfpdf` tag is RC4_128. It is recommended to use AES_128 for stronger encryption *unless* the encrypted PDF must be compatible with Acrobat 6.0 or earlier.

- When using the `getinfo` action of the `cfpdf` tag to view permissions for a PDF document that is encrypted, specify the user password, *not* the owner password. If you specify the owner password, all permissions are set to allow.

- ColdFusion does not retain permissions: If you add a `newUserPassword` attribute, you also must set the permissions explicitly and specify the `newOwnerPassword` also.

- ColdFusion provides the `IsPDFFile` and `IsPDFObject` functions to validate PDF files and objects. Use these functions to validate that a PDF file or object is correctly formed before operating on it with other functions and tags.

- ColdFusion provides the `IsDDX` function to validate whether a DDX file or set of instructions is valid. Use this function to validate a DDX file or set of instructions before passing them to the processddx action of the **`cfpdf`** tag.

.NET integration

ColdFusion 8 now allows integration with .NET components through the `cfobject` tag.

Best practices

- Disable the `cfobject` tag if your application does not require it. This can be done by accessing the ColdFusion administration console, going to Security > Sandbox Security and then selecting to disable.
- If the `cfobject` tag will be accessing a .NET object on another server, use the `'secure = yes'` attribute to specify SSL should be used to interact with the remote .NET object.

HTTP

ColdFusion 8 adds the ability to use SSL to connect to other HTTP servers using the `cfhttp` tag. It also supports client certificates to enable authentication to the remote server.

Best practices

- Use SSL to connect to a server when sensitive information will be exchanged (such as login credentials, credit card information). This will both encrypt the network traffic to the server and verify that the server is the server it claims to be. This can be done by specifying the `'secure = yes'` attribute to the `cfhttp` tag.
- Where extra security is required, use client certificates to authenticate the ColdFusion server to the HTTP server. This can be done by specifying the `ClientCert` and `ClientCertPassword` attributes to the `cfhttp` tag. The `ClientCert` attribute is the full path to a PKCS12 format file containing the client certificate. The `ClientCertPassword` attribute specifies the password used to unlock the client certificate.

FTP

ColdFusion 8 adds the ability to use secure FTP to the cfftp tag by specifying the attribute `'secure = yes'.`

The `cfftp` tag supports two methods of authentication to the secure FTP server : password authentication and public/private key authentication.

Password authentication is done by specifying the password atttibute to the `cfftp` tag.

Public/private key authentication is done by specifying the private key to use in the key attribute of the `cfftp` tag and the passphrase used to decrypt the private key in the passphrase attribute of the `cfftp` tag.

Best practices

- Use secure FTP whenever possible as standard FTP passes login credentials in plain text across the network. Secure FTP uses an encrypted channel and login information is not sent in the clear.

- When additional security is required, the fingerprint attribute can be used with the cfftp tag when the attribute `'secure = yes'` is also specified. This attribute specifies the host fingerprint of the secure FTP server which is being connected to. If the fingerprint sent by the secure FTP server does not match the fingerprint that is specified in the fingerprint attribute, the connection is aborted. The fingerprint attribute can be used with either the public/private key or password authentication methods.

Best practices in action

The sample code below shows an example of how to connect to a secure FTP server. Note that the fingerprint is specified to help make sure ColdFusion is really talking to the intended secure FTP server and also that the password is obtained from a secured file using `GetProfileString` so the plain-text password cannot be recovered via inspecting the compiled class file. Note also that the `stopOnError` attribute is set to "no" so no detailed error information is returned, preventing information leakage.

```
<cfset pwvar = GetProfileString("C:\secured\admin.ini", "pw_section",
   "password") />
<cfftp action = "open"
connection = "Myconnection"
fingerprint = "65:c7:4d:5c::3b:a6:9b:1e"
password = "#pwvar#"
retryCount = "3"
secure = "yes"
server = "secure.ftp.example.com"
stopOnError = "no"
timeout = "10"
username = "example">
```

The C:\secured\admin.ini file looks like this:

```
[pw_section]
password=secrets
```

Error handling and logging

All applications have failures—whether they occur during compilation or run time. Most programming languages will throw run-time exceptions when executing invalid 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:

1. Template level (ColdFusion templates and components)
   - ColdFusion exception handling tags: `<cftry>`, `<cfcatch>`, `<cfthrow>`, and `<cfrethrow>`
   - `try` and `catch` statements in CFScript
2. 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
3. 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 8 Administrator; for more information, see the ColdFusion 8 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 descriptions[ii]:

Log file	Description
rdservice.log	Records errors that occur in the ColdFusion Remote Development Service (RDS). RDS provides remote HTTP-based access to files and databases.
application.log	Records every ColdFusion error reported to a user. Application page errors, including ColdFusion 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.
eventgateway.log	Records events and errors related to event gateways.
migration.log	Records errors related to upgrading from a previous version of ColdFusion.
migrationException.log	Records errors related to running ColdFusion applications after upgrading from a previous version of ColdFusion.
server.log	Records errors for ColdFusion.
customtag.log	Records errors generated in custom tag processing.
car.log	Records errors associated with site archive and restore operations.
mail.log	Records errors generated by an SMTP mail server.
mailsent.log	Records messages that ColdFusion sends.
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, and log to UNIX operating system logging facilities. 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 the cflog tag 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 = true />
        <cfset Session.UserName = strUserName />
        <cflog text="#loginQuery.strUserName# has logged in!"
          type="Information"
          file="access"
          application="yes" />
        <!-- Add code to get roles from database -->
      <cfelse>
        <cfset retargs.authenticated = false />
      </cfif>
    <cfelse>
      <cfset retargs.authenticated = false />
    </cfif>
    <cfreturn retargs />
    <cfcatch type="database">
      <cflog text="Error in dbLogin(). #cfcatch.details#"
        type="Error"
        log="Application"
        application="yes" />
      <cfset retargs.authenticated = false />
      <cfreturn retargs />
    </cfcatch>
  </cftry>
</cffunction>
…
<cffunction name="logout" access="remote" output="true">
  <cfargument name="logintype" type="string" required="yes">
  <cfset var userName = getAuthUser() />
  <cftry>
    <cflogout>
    <cfset StructClear(Session) />
    <cflog text="#userName# 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>
    <cfcatch type="any">
      <cflog text="Error in logout(). #cfcatch.details#"
        type="Error"
        log="Application"
        application="yes" />
    </cfcatch>.
  </cftry>
</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 practices

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 at the web server level.
- 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, .jsp, 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.

Image files :

- Use the `IsImageFile` function to validate that an image file is legitimate and correctly formed before performing actions on it with the `cfimage` tag.

PDF files :

- Use the `IsPDFFile` function to validate that a PDF file is legitimate and correctly formed before performing actions on it with any of the ColdFusion 8 PDF tags.

Cryptography

The following section describes ColdFusion's cryptography features. ColdFusion 8 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 parameters[iii]:

- `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 8 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 and `keysize` for the `encrypt` and `encryptBinary` functions.

```
generateSecretKey(algorithm [, keysize])
```

ColdFusion offers the following default algorithms for these functions[iv]

- `CFMX_COMPAT`: Generates a MD5 hash string identical to that generated by ColdFusion MX and ColdFusion MX 6.1 (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.

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]] )
```

The following algorithms are provided by default for the hash() function[v].

Note: SHA algorithms used in ColdFusion are NIST FIPS-180-2 compliant:

- `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 (recommended)
- `SHA-384`: Generates a 384-bit digest
- `SHA-512`: Generates a 512-bit digest

PBEwithMD5andTripleDES (and other algorithms if large key sizes are used) requires replacing two files in the runtime jre/lib/security/ directory: US_export_policy.jar and local_policy.jar.

The replacement files are in "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6" which is available at http://java.sun.com/javase/downloads. The download file is named jce_policy-6.zip.

Note: This download is not available in Cuba, Iran, North Korea, Syria, or Sudan per http://www.gpo.gov/bis/ear/pdf/740spir.pdf (August, 2007) Group E:1.

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

ColdFusion 8 Enterprise edition includes the RSA BSafe® Crypto-J library (version 3.6). This library includes the JSafeJCE provider which adds these additional algorithms:

- Secret-Key: DESX, RC5
- Password: PBEwithSHA1andDES, PBEwithSHA1andRC2
- Hash: SHA-224, RIPEMD160
- Random: DummyPRNG, FIPS186PRNG, MD5PRNG, OBFPRNG

Notes:

- The JSafeJCE provider replaces the Sun provider for these algorithms in CF8 Enterprise:

  AES, DESEDE, DES, RC2, RC4, PBEwithMD5andDES, MD5, MD2, SHA1, SHA-256, SHA-384, SHA-512, SHA1PRNG
- All SHA algorithms have synonyms without a hyphen. SHA224, SHA256, SHA384, SHA512
- FIPS186_PRNG and FIPS186PRNGxChangeNoticeGeneral are synonyms for FIPS186PRNG
- DummyPRNG always returns zero.
- TripleDES *is not* a synonym for DESEDE in the JsafeJCE provider. TripleDES will use the Sun provider.
- ARCFOUR *is not* a synonym for RC4 in the JsafeJCE provider. ARCFOUR will use the Sun provider.
- DESX, PBEwithSHA1andDES, PBEwithSHA1andRC2 require the "Unlimited Strength Jurisdiction Policy Files" as detailed above.
- FIPS186PRNG may exhaust the supply of entropy from /dev/random and hang on some Unix platforms, especially in Virtual Machines.

FIPS-140 compliant encryption

The RSA JsafeJCE provider included in the ColdFusion 8 Enterprise edition provides FIPS-140 Compliant Strong Cryptography. The following are the FIPS-140 approved algorithms that can be used with ColdFusion 8:

- AES – ECB, CBC, CFB (128), OFB (128) – [128, 192, 256 bit key sizes]
- AES – CTR
- FIPS 186-2 General Purpose [(x-Change Notice); (SHA-1)]
- FIPS 186-2 [(x-Change Notice); (SHA-1)]
- Secure Hash Standard (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)

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 web and LDAP 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 8 Administrator console web application (http://*servername*:*port*/CFIDE/administrator/index.cfm). If the console application is unavailable, you can modify these options by using the Administrator API or by manually 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]

Security > Administrator

- Enable separate user name and password authentication:
  By default the ColdFusion Administrator is secured by a single password. ColdFusion 8 provides multi-user access to the Administrator. When enabled, users must enter a user name and password to authenticate to the Administrator. Create user accounts and passwords with the ColdFusion 8 User Manager.
  1. Fill the radio button next to “Separate user name and password authentication (allows multiple users).”
  2. Click Submit Changes
- Use a strong password for the Root Administrator Password. The ColdFusion 8 Administrator is secured by root user name and root password. Ensure that the root password is strong and stored in a secure place.
  1. Enter and confirm a strong password string of eight characters or more.
  2. Click Submit Changes.

Note: See Adobe TechNote kb402459 at http://www.adobe.com/go/kb402459 for information on changing the Root Administrator Username.

Security > RDS

- Enable separate user name and password authentication:
  Developers can access ColdFusion resources (files and data sources) over HTTP from Adobe Dreamweaver and HomeSite+, and Eclipse through ColdFusion's Remote Development Services (RDS). RDS is protected by a single password by default. ColdFusion 8 provides multi-user access to RDS. Create user accounts and grant RDS access with the User Manager.
  1. Fill the radio button next to “Separate user name and password authentication (allows multiple users).”
  2. Click Submit Changes

Note: ColdFusion 8 adds the ability to use a single password for RDS for all users or to specify which users defined in the User Manager are allowed to have access through RDS. You should limit RDS use specifically to the users that need it and disallow it for all other users.

- Enable a strong RDS password:
  ColdFusion RDS is protected by a single password by default. Ensure that the default RDS password is strong and stored in a secure place.

  1. Enter and confirm a strong password string of eight characters or more.
  2. Click Submit Changes.

- Use RDS over SSL:
  During development, you should use SSL v3 to encrypt all RDS communications between RDS clients 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 8, 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.

  1. Back up the C:\ColdFusion8\wwwroot\WEB-INF\web.xml file.
  2. Open the web.xml file for editing.
  3. Comment out the RDSServlet mapping, as follows:

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

  4. Save the file.
  5. Restart ColdFusion.

Security > Sandbox Security

- 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.

Security > User Manager

The ColdFusion 8 Administrator includes the User Manager screen for configuring user-based Administrator and RDS access. User-based access allows administrators to create individual user accounts with individual passwords, and then assign various roles to those accounts. By default there are no configured user accounts. To enable user-based access you must first create user accounts and passwords, assign roles to those accounts, and then enable the "Separate user name and password authentication" Administrator and/or RDS authentication type.

Use the following steps to create a new user account[vii]:

1. Click the Add User button.
2. User Authentication section
   a. Enter a unique username.
   b. Enter and confirm a password.
   c. (optional) Enter a useful description.
3. User RDS and Administrator Access section
   a. To grant the user RDS permissions, select the Allow RDS Access option. If the "Separate user name and password authentication" type is not enabled for RDS authentication you will receive an alert message.
   b. Select the Allow Administrative Access option to grant the user access to Administrator resources.
      i. Decide whether to grant Administrator Console and API Access, or API Access only. Select the appropriate option.
      ii. Highlight the roles in the Prohibited Roles list that you want to grant to the user account and press the left arrow (<<) button to move them into the Allowed Roles list.
4. User Sandboxes section

a. Highlight any configured Sandboxes in the Prohibited Sandboxes list that you want to grant to the user account and press the left arrow (<<) button to move them into the Allowed Sandboxes list.

5. Click Add User button.

Server Settings > Settings

- Timeout Requests after (seconds):
  Enable this option and set a reasonable timeout value. 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.

  1. Fill the checkbox next to "Timeout Request after (seconds)".
  2. 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).

- Enable Per App Settings
  Enable this setting to allow developers to programmatically define application-specific ColdFusion settings, such as ColdFusion Mappings.

- 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.

- Disable access to internal ColdFusion Java components
  Enable this option to prevent unauthorized access to the ColdFusion ServiceFactory Java objects.

- Prefix serialized JSON with
  Enable this option to help protect web services returning JSON data from cross-site scripting attacks.

- Enable Global Script Protection: This is a security feature that was added 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 missing template handler:
  Provide a custom message page for HTTP 404 errors when the server cannot find the requested ColdFusion template.

- 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.

- 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.

Server Settings > Caching

- Trusted cache
  Enable Trusted cache in production environments. When enabled, ColdFusion will only server requested templates held in its memory cache. This provides performance gains but also prevents ColdFusion from running hacked or invalid templates.
- Save class files
  This option saves class files generated by ColdFusion to disk for reuse at server start up. Since these class files can be reverse engineered if your server is compromised, consider disabling this option.

Server Settings >Client Variables

- Default Client Storage Mechanism for Client Sessions
  Enable an option other than Registry for client variable storage. Adobe recommends adding a RDBMS as a client variable store. See TechNote 17919 at http://www.adobe.com/go/tn_17919 for more information.

Server Settings > Memory Variables

- 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.

Server Settings > Mail

- Require a user name and password to authenticate to your mail server.
- Set the connection timeout to 60 seconds (default)
- For securer communication between ColdFusion and the mail server enable either SSL socket or TLS connections.

Data & Services > Data Sources

- 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.

Data & Services > Flex Integration

- Enable Flash Remoting support
  Only enable this option if you want to allow Flash/Flex clients to connect to ColdFusion through ColdFusion Components.

Note: Disabling this option also disables the Flex-based Server Monitor.

- Enable Remote Adobe LiveCycle Data Management access
  Only enable this service if you want to allow remote LiveCycle Data Services ES to connect to ColdFusion over RMI (Remote Method Invocation).
- Enable RMI over SSL for Data Management
  Enable this option and configure the keystore path and password to provide SSL encryption for RMI communication between ColdFusion and Flex clients.
- Select IP addresses where LiveCycle Data Services are running
  If you enable remote Adobe LiveCycle data management access, enter IP Addresses here to restrict the access to those specific remote addresses.

Debugging & Logging > Debugging Output Settings

- Enable Robust Exception Information
  Disable this option for production servers. (default)

- Enable AJAX Debug Log Window
  Disable this option for production servers. (default)
- Enable Request Debugging Output
  Disable this option for production servers. (default)

Debugging & Logging > 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 (IPv4) or 0:0:0:0:0:0:0:1 (IPv6)) in the list on production machines.

Debugging & Logging > Debugger Settings

- Allow Line Debugging
  Disable this option for production servers. (default)

Maintenance

This section provides guidance on maintaining ColdFusion. 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 Adobe supported software release types[viii]:

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/releasenotes.html
    - The Adobe LiveDocs provides an online version of the ColdFusion 8 manuals (with customer comments): http://livedocs.adobe.com/coldfusion/8/index.html
- Ensure that the platform on which ColdFusion is running has the latest stable patches. This includes the operating system, Java Virtual Machine, J2EE application server, 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 8 server and applications:

- Forta, Ben. Adobe ColdFusion 8 Web Application Construction Kit. Vol 3. Berkley, CA: Adobe Press, 2007.
- 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 8 LiveDocs
http://livedocs.adobe.com/coldfusion/8/htmldocs/Errors_02.html

[ii] *Ibid.*
http://livedocs.adobe.com/coldfusion/8/htmldocs/basiconfig_21.html

[iii] *Ibid.*
http://livedocs.adobe.com/coldfusion/8/htmldocs/functions_m-r_22.html

[iv] *Ibid.*
http://livedocs.adobe.com/coldfusion/8/htmldocs/functions_e-g_01.html

[v] *Ibid.*
http://livedocs.adobe.com/coldfusion/8/htmldocs/functions_h-im_01.html

[vi] Configuring ColdFusion MX 7 Server Security.
http://www.adobe.com/devnet/coldfusion/articles/cf7_security.html

[vii] Adobe ColdFusion 8 Web Application Construction Kit. Vol 3. pp.

[viii] Adobe Software – Release Terminology.
http://www.adobe.com/support/updaters/terms.html