Global Information Assurance Certification Paper

Web Application Security,

With a Focus on ColdFusion

Version 1

In fulfillment of GSEC certification requirements

Joseph Higgins

Table of Contents

Abstract 2
Introduction 3
Security Risks Inherent in ColdFusion 3
  Default Installations 3
  Two Step Attacks 4
  Remote Development 4
Security Risks in Code 5
  Casing the Store 5
  Securing Code 6
    Users Executing Commands 6
    Nested SQL Statements 7
    Cross Site Scripting 7
    State Management Security 8
    Protecting Templates 9
    Securing the Browser Traffic 9
    Inner Workings of an Application 10
Conclusion 10

Abstract

Security is often overlooked in web application development. Web applications must be secured 'in depth' because they are dependent on the hardware, the OS, the web server, the database, the scripting language, and finally the application code. Although web application security is not product specific we will focus on the last two layers using ColdFusion (CF) and the code. This paper covers default installation, two-step attacks, remote development, and security holes in the code, input encryption, which are the major issues in most web applications. Consult the latest product documentation for new security risks.

Web Application Security, With a Focus on ColdFusion

Introduction

Security is needed and often overlooked in web application development. Web application security should use a graded approach have a risk analysis as its base. Web applications must be secured 'in depth' because they are dependent on the hardware, the OS, the web server, the database, the scripting language, and finally the application code. Although web application security is not product specific we will focus on the last two layers using ColdFusion (CF) and the code. This paper will not cover securing the other layers, policy, or web application risk assessments, which would be a great topic for another GSEC paper. Also you should consult the latest product documentation for security risks.

Security Risks Inherent in ColdFusion

There are some security risks inherent with CF that are application specific.

Any web application will have general risks associated with the scripting language used. Apply the appropriate countermeasures available. According to Malcolm Gin[1], Macromedia's security manager, five top security risks for CF are:

- ColdFusion Administrator on Production Servers
- Unvalidated Browser Input
- Sample Applications and Documentation on Production Servers
- CFFILE, CFFTP, and CFPOP
- ColdFusion Studio and RDS with Production Servers

Default Installations

There are a number of issues with a default installation of CF on a production server.[2] This seems to be the case with software like operating systems and the like; you need to follow a best practice for securing them before deployment.

CF server has a Java applet that starts and stops the service. When using basic security features this utility is protected with the administrator password. However, when using advanced security it is not protected at all. A fix for this is to turn on directory security in IIS for this administrative directory. A ***best practice*** for the ultra paranoid is to

completely remove the programs from the CF admin directory on the production environment and protect the directory with Web Server and operating system file system security. Instructions for doing this are available on Allaire's[3] website. This way, in case you need to temporarily place the admin files there to change the server settings it will already be protected with Windows authentication. CF Server only has a browser front end for administration. There are potentially harmful tags in the admin directory that would not be available to the hacker if they accessed the directory. A single password to protect the admin area is not sufficient in any production environment.

A security axiom: Hardened servers should only run the processes they need.

In older versions of CF the default install included sample applications and documentation whose code could be exploited. ***Best Practice:*** Remove the sample applications and documentation directories. You should refer to the security bulletin for this as each product has its own requirements.[4] Sample Java servlets should also be removed. The code in these sample applications could be exploited to allow a simple DOS attack or to gain access to your server using a two-step attack.

Two Step Attacks

Attacks of this nature involve a file or code being uploaded to the server, then getting the server to execute the file in some way. This is not an uncommon approach. There are a small group of CF tags that allow the upload of files to the server, CFFILE, CFFTP, and CFPOP. If the files are not checked to make sure they are the proper MIME type you could get hit. Even if the user uploads CFML (Cold Fusion Markup Language (code)) it could be executed arbitrarily.

Best Practice[5]: First, upload the files to a safe directory, one that is not accessible to the web server for execution. The directory should also be limited by the operating system to read-only. Verify that the file being uploaded is the type of file it is supposed to be. You can use the 'accept' attribute to limit the mime-type. You may need to develop or use a custom CFX tag to validate the file. The file may have a virus payload as well. Use a command line CFEXECUTE with a virus scanner to scan the file for viruses. You may also want to run scheduled virus scans of the directories for added protection.

Remote Development

CF allows the use of Remote Development Services (RDS) with its CFStudio application. It uses HTTP or FTP to connect developers to the server. ***Best Practice:*** CF RDS should only be used in a secure intranet and only on development servers. If you need to use RDS consider using a VPN technology to secure the network traffic.

Security Risks in Code

The security officers, policies makers, and programmers are often different people in an organization. Depending upon the organization there may be a gap between programmers and security; programmers may not be focused on an application's security model or only think of security as user authentication. A web application allows inputs from forms, URL query strings, server variables, and cookies. If a hacker can gain access to any of these then they essentially have a small foothold into your application. Not all web applications will have the same set of security requirements. But you should consider answering questions like: If a user tried to send a malformed request how would I know? If my application were broken how would I know? How would I know if user an authentication attempt failed? What if someone was trying a brute-force attack against a user account? How can I keep users from getting data that they are not supposed to get?

There are some web application security risks that are not application specific like buffer overflows. Although CF is not as susceptible to buffer overflows as other web application scripting languages it has its own set of malformed input risks. Web applications can expose susceptibilities of every component of your application, from the scripting language to the web server to the OS, and must be properly secured. CF is a powerful scripting language for building web applications and as such gives the same power to hackers if hijacked.

Casing the Store

A security-oriented person might case a store when they walk into it without the intent of breaking in, just because that is the type of person they are. When you 'walk into' a web application (which may be an online store) you may start casing it as well. Some questions you may ask are: What scripting language are they using? Look at the URL string for cgi-bin, asp, jsp, shtml, cfm, php or if they are using fake extensions to hide which scripting language they are using. Which web server are they using? When you create a 404 error what does the server tell you? Someone could be stealthy about gathering data or be a script kiddie who uses a tool to test

your web server's configuration remotely. It would be difficult to determine that someone is casing your application just from sever logs. How are they maintaining state in the stateless html environment? Are they using session ID's are the session numbers random or are they incremental? You may look to see if they stored a cookie in your cache and try a cookie-viewing program (http://www.karenware.com/powertools/ptcookie.html) to quickly view the contents of the cookies. Are they storing a userid in the cookie to authenticate a user? Or are they simply placing a userid in the URL? I recently found a security hole in a web application that used a userid in the URL, the easiest place to get at it. All I had to do was change the userid until I found a super user or admin. Is the website allowing people to upload files, or post messages? Have they programmed their application to strip malicious code from user input in web forms and url strings? Have they allowed people to defame a sight by posting embarrassing material? What about legal implications of someone posting illegal files on the site?

Securing Code

Browser input validation is not an issue that is unique to CF. I thought it would be difficult for someone to inject malformed information into a packet stream until I found a utility called Achilles from Roberto Cardona[6] that acts as a proxy server that captures and holds the information going from a browser to a server and lets you inject or change information in the stream. Achilles will even let you do a man in the middle for SSL because it will negotiate two different SSL connections, one between it and the server and one between it and the client this allowing clear text in the middle. Remember that you cannot always trust your users and you may not know if a user account has been hijacked. The vulnerabilities listed here are not comprehensive. They are meant to give you an idea of the various methods used to attack web applications. Consult the latest security bulletins for your application software.

Users Executing Commands[7]

If you are using Access as a database driver then your web application is probably only serving a small portion of people and not budgeted for security. Many CF projects start out this way and migrate to SQL or Oracle or mySQL as the project grows.

Some versions of the Microsoft Access ODBC driver allow for appending VBA commands to a SQL string. The VBA commands are appended by using the pipe character.

Example:

```
'|shell("cmd /c 1 > c:\temp\foo.txt")|'
```

This string could be passed to an application using a URL variable, so the page could be called as follows:

```
http://myserver/page.cfm?x='|shell("cmd /c 1 > c:\temp\foo.txt")|'
```

This code, when executed as part of the following dynamically created query, will cause a file to be created at the location c:\temp\foo.txt.

```
SELECT   *
FROM     USERS
WHERE    lname = '#URL.X#'
```

To fix this you can throw errors in your code when you encounter a pipe and log it as a high risk. Also update your ODBC driver. Access does not allow stored procedures but your could change your queries to stored procedures.

Nested SQL Statements

This is another common vulnerability exposed by the scripting language but is actually related to the database component. It is possible to append a malicious SQL statement to a SQL statement in the code. Take a look at this example:

```
SELECT * FROM usertable
WHERE userid = 1 DELETE FROM userstable
```

The above statement would delete every user in your database, and if you had cascading deletes, well can you say backup? All the malicious user would have to do is know or guess the name of a table in your database and create a query like this:

```
http://servername/page.cfm?uid=23%20DELETE%20FRO
M%20userstable
```

Pretty easy to do. CF code is only susceptible if the query uses a number as input in a query or uses a string that protects single quotes using the PreserveSingleQuotes() function. To protect your code you would use the val() function around the variable. The val() function returns 0 if the expression is not numeric:

```
<CFQUERY DATASOURCE="DBName" NAME="Test">
        SELECT * FROM usertable
        WHERE userid = #Val(uid)#
</CFQUERY>
```

Cross Site Scripting[8]

This vulnerability is also not inherent to CF. This allows the insertion of malicious code using html tags that embed code like <SCRIPT>, <OBJECT>, <APPLET>, and <EMBED>. The potential impact is:

- SSL-Encrypted Connections May Be Exposed
- Attacks May Be Persistent Through Poisoned Cookies
- Attacker May Access Restricted Web Sites from the Client
- Domain Based Security Policies May Be Violated
- Use of Less-Common Character Sets May Present Additional Risk
- Attacker May Alter the Behavior of Forms

The main thing here is that scripts should filter all input for unwanted code. CERT has published an article on mitigating such code, http://www.cert.org/tech_tips/malicious_code_mitigation.html/.[9] Specifically for CF is a custom tag available at http://download.allaire.com/patches/inputfilter.zip that allows your to dynamically select the variable scopes, the characters, and the tags that you want filtered. The custom tag is invoked using the following:

```
<cf_inputFilter
 scopes = "[FORM][,COOKIE][,URL]"
 chars = "list_of_chars"
 tags = "ALL|list_of_tags">
```

State Management Security

How does your site maintain state? State is usually maintained using a session variable. If someone accesses the session variable they can mimic a user. So programmers use several approaches to mitigate this.

- Do not pass session id data on URL strings.
- If using url session data do not create links to other sites, the HTTP_REFERER http header information will contain the session id.
- Use other information like IP addresses as part of the session variable. CF does not allow this with the native session architecture that is part of the server.
- Keep session timeouts short.
- User education on logging out of applications on public terminals to prevent the use of the back and history to get into sessions.

If you are using native CF session tracking then you will be using

CFID and CFTOKEN, which are passed on each request in a cookie. If you do not use cookies then you must append them onto the end of every url, form request, and redirect. However these numbers could be pulled by a sniffer off a network and used to impersonate someone in an application. There are custom tags available on CF developers exchange that modify the native session management for more security. One such tag is aSession[10] will prevents attempts to use someone else's session by providing three additional validation levels of session security: Secret Key, client's IP address, and referrer data. This tag is great in the fact that it allows for the management of sessions on a server and audit logs for session spoofing. The Borg will definitely want to assimilate advanced session management.

Protecting Templates

CF has the ability to encrypt your source code files. The server will then store a decrypted version in memory as it needs them. You can use this, but only to discourage amateurs. The encryption algorithm is simple and has been broken. Future versions of CF promise advanced algorithms for template encryption.

Securing the Browser Traffic

You can obviously use SSL to protect your browser sessions. In addition to SSL you can use IIS's authentication mechanisms to protect your applications. This means that you can use everything from unencrypted basic authentication, to integrated windows authentication with certificates/smartcards.

To handle the management of user passwords, that in this context are now actually NT user accounts, you can use a C++ library and call it as a custom tag in CF. This custom tag is available from the CF developers exchange: http://www.allaire.com/developer.

```
<!--- Create a WinNT account. --->
<CFX_USERDB ACTION="USERADD"
        USERNAME=""
        FULLNAME=""
        PASSWORD=""
        PRIVS=""
        GROUPS=""
        COMMENT=""
```

```
ACCOUNTDISABLE="0"
PASSWD_NOTREQD="0"
PASSWD_CANT_CHANGE="0"
LOCKOUT = "0"
DONT_EXPIRE_PASSWD="0"
MUST_CHANGE_PASSWD="0">
```

This could be dangerous if your code is hijacked. But users often expect that their passwords can be changed in a web application. It may be a compromise, remember that security often involves compromise and accepting a level of risk.

Inner Workings of an Application

Ok, so you have built a hypothetical application, lets call it Acme Sales. This application will have groups. The Sales group will only have access to their own customer's information. You are the application programmer and have used all of the applicable security features mentioned so far. Great, now you need to program your application so that users cannot manipulate any variables that are used for input. One way to do this in a comprehensive manner is to encrypt URL, cookie, and form variables. Have you noticed major web sites encrypting or disguising their URL strings so that people cannot querystring surf? Steve Nelson of SecretAgents.com (http://www.sercretagents.com) developed a custom tag that works in conjunction with the Fusebox (http://www.fusebox.org) web application methodology to encrypt all URL querystrings, form variables, and cookies. Depending on the level of security needed you could extent his tag to use a custom encryption algorithm commensurate with your needs. Remember that it will be very processor intensive. Security is a tradeoff with convenience.

An additional way to protect small pieces of code is to use a security framework like that proposed by Hal Helms.[11] He uses a custom tag called <CF_Secure> that will accepts the users permissions, the permission required to execute the code, a security model (list or bit), and an action to perform if they are not allowed in. You may want to modify it to log permission failures. He has made this model extensible and has included 'list' and 'bit' validation models. Security is easier when your development team follows a programming methodology.

Conclusion

You corporate intranet may consist of web applications and your extranet can be an extension of your intranet. In securing corporate networks the security focus is first placed on perimeter defense, leaving the intranet for the next round of budgeting. The highest risk for a system is from the inside. That is what you are trying to do is keep the hackers from gaining access to the inside of your network. If we remember ‘defense in depth’ we should take steps to protect our web applications at the code level. Steps should be taken to protect your application from insiders as well.

End Notes

[1] Gin, Malcom. "Top Five ColdFusion Security Issues." 2 January 2002,. URL:http://allaire.com/Handlers/index.cfm?ID=19491&Method=Full&Cache=Off

[2] (No Author). Allaire Security Bulletin (ASB99-07). "Solution Available for Denial-of-Service Attack Using CF Admin." URL:http://allaire.com/handlers/index.cfm?ID=10968&Method=Full (May 19, 1999)

- (No Author). Allaire Security Bulletin (ASB99-10). "Addressing Potential Security Issues with Undocumented CFML Tags and Functions Used in the ColdFusion Administrator." 2 January 2002. URL: http://allaire.com/handlers/index.cfm?ID=11714&Method=Full (July 29, 1999)
- (No Author). "Allaire Security Best Practice: Securing the ColdFusion Administrator"2 January 2002. URL: http://allaire.com/Handlers/index.cfm?ID=10954&Method=Full (No Date)
- (No Author). "Allaire Security Best Practice: Setting Up ColdFusion Administrator Security" 2 January 2002. URL: http://allaire.com/Handlers/index.cfm?ID=8300&Method=Full (No Date)

[3] (No Author). "Allaire Security Best Practice: Securing ColdFusion Pages through IIS." 2 January 2002. URL: http://allaire.com/Handlers/index.cfm?ID=1533&Method=Full (No date).

[4] (No Author). "Security Best Practice: Removing Sample Applications and Online Documentation from Production Servers." 2 January 2002. URL: http://allaire.com/Handlers/index.cfm?ID=16258&Method=Full (No date).

[5] (No Author). "Security Best Practice: Evaluating the Risks of Allowing Uploading and of Attached Files on Your Server." 2 January 2002. URL: http://allaire.com/Handlers/index.cfm?ID=17407&Method=Full (No Date)

[6] DigiZen Security Group web site, 2 January 2002. http://www.digizen-security.com

[7] (No Author). "Allaire Security Bulletin (ASB99-09)." 2 January 2002. URL: http://allaire.com/handlers/index.cfm?id=11069&method=full. June 8, 1999.

[8] "CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests." 2 January 2002. URL: http://www.cert.org/advisories/CA-2000-02.html February 3, 2000.

[9] (No Author). "Understanding Malicious Content Mitigation for Web Developers." 2 January 2002. URL: http://www.cert.org/tech_tips/malicious_code_mitigation.html Feb 2, 2000.

[10] KUZNETSKY, ALEX. "aSession Cold Fusion Custom Tag." 2 January 2002. URL:

http://devex.allaire.com/developer/gallery/info.cfm?ID=AE230230-45FF-11D4-AA9800508B94F380&method=Full January 02, 2002.

[11] http://halhelms.com/writings/ProposedSecurityModel.pdf