Deconstructing ColdFusion

BlackHat USA 2010
July 29, 2010

VERACODE

Bios

- Chris Eng
  - Senior Director of Research at Veracode
- Previously
  - Technical Director and Consultant at @stake (and Symantec, through acquisition)
  - Security Researcher, etc. at NSA
- Other
  - Frequent speaker at security conferences
  - Contributor to various CWE, OWASP, WASC projects
  - Advisory board for SOURCE Conferences (BOS, BCN)
  - Developed @stake WebProxy (for any old timers out there)

- Brandon Creighton
  - Security Researcher at Veracode
- Previously
  - Engineer/architect at VeriSign MSS (ex-Guardent); focus on high-volume security event storage & transmission
- Other
  - Operations/goon volunteer at several conferences (DEFCON, SOURCE BOS, HOPE 5)
  - Ninja Networks party badge firmware dev
  - Old stuff: Stint as the maintainer of OpenBSD/vax (~1999-2002)

Agenda

- ColdFusion Background and History
- Platform Architecture and CFML
- Practical Tips for Developers and Code Reviewers
- ColdFusion Behind the Curtain

ColdFusion Background and History

ColdFusion History

- Originally released in 1995 by Allaire
  - Motivation: make it easier to connect simple HTML pages to a database
  - Initially Windows only with built-in web server
- Migration to J2EE with ColdFusion 6 in 2002
  - Everything compiled to Java classes before being run
  - Apps can be bundled up as WARs/EARs, including admin interface if desired
  - Bundled with JRun
- Latest version is ColdFusion 9 released in 2009
  - Most recent features focus on integration with other technologies, e.g. Flash, Flex, AIR, Exchange, MS Office, etc.

Historical Vulnerabilities

- Within the past 12 months
  - Multiple XSS, some as recent as May 2010 (not much detail available)
  - "Double-encoded null character" CVE in August 2009
- Lots of XSS in sample apps, administrator UI, error pages
- Source code disclosure
  - JRun canonicalization mistakes, bugs in sample applications
- Authorization vulnerabilities related to administrative UI
- Sandbox escape scenarios
- Prior to ColdFusion 6 (Allaire/Macromedia days)
  - Arbitrary file retrieval
  - XOR used to encrypt passwords
  - Predictable session identifiers (may have been sequential, IIRC)
  - Various DoS conditions and buffer overflows

Source: National Vulnerability Database

Who Uses ColdFusion Anyway?

- Lots of people, believe it or not. Let's start by asking Google…

Search Term	Hits
ext:asp	1,170,000,000
ext:aspx	1,220,000,000
ext:cfm	**310,000,000**
ext:jsp	518,000,000
ext:php	5,060,000,000
ext:pl	139,000,000
ext:py	5,130,000
ext:rb	244,000

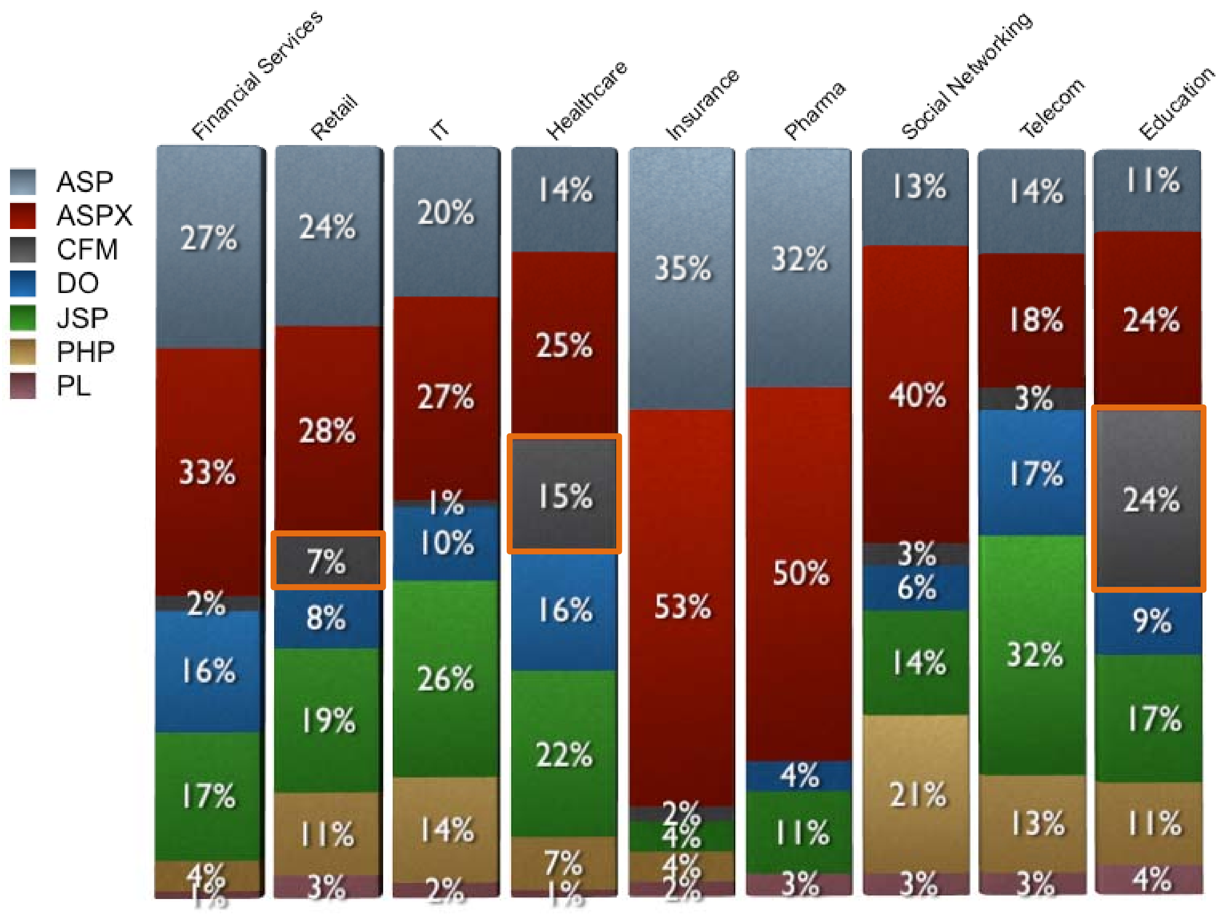
Source: Google, May 14, 2010

Who Uses ColdFusion Anyway?

- “More than ***770,000 developers*** at over ***12,000 companies*** worldwide rely on Adobe® ColdFusion® software to rapidly build and deploy Internet applications. And with more than 125,000 ColdFusion servers deployed, ColdFusion is one of the most widely adopted web technologies in the industry.”

Source: http://www.adobe.com/products/coldfusion/customers/

ColdFusion Prevalence by Vertical



Source: WhiteHat Website Security Statistics Report, 9th Edition, May 2010

Our Motivations

- We were developing ColdFusion support for our binary analysis service, so we were doing the research anyway
- Few resources available on securing or testing ColdFusion apps
  - ColdFusion 8 developer security guidelines from 2007 http://www.adobe.com/devnet/coldfusion/articles/dev_security/coldfusion_security_cf8.pdf
  - "Securing Applications" section of ColdFusion 9 developer guide is similar, almost entirely about authentication methods http://help.adobe.com/en_US/ColdFusion/9.0/Developing/coldfusion_9_dev.pdf
  - OWASP ColdFusion ESAPI started May 2009, abandoned (?) June 2009 http://code.google.com/p/owasp-esapi-coldfusion/source/list
  - EUSec presentation from 2006 focused mostly on the infrastructure footprint and deployment issues (admin interfaces, privilege levels, etc.) http://eusecwest.com/esw06/esw06-davis.pdf
  - Other presentations dedicated to ColdFusion security

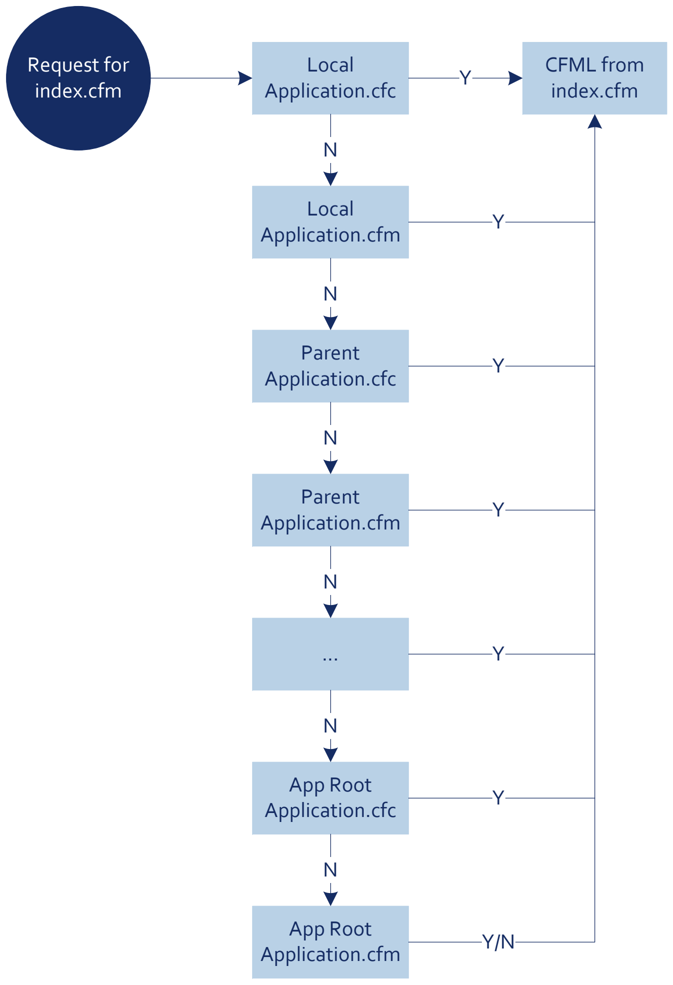
Platform Architecture and CFML

CFML Building Blocks

- Pages
  - Main entry points of a CF application
  - Similar to an HTML page (or PHP, JSP, etc.) except using CFML tags
  - .cfm extension
- Components
  - Contain reusable functions / variables for use by other code
  - Written entirely in CFML
  - .cfc extension
- Functions (UDFs)
  - Defined inside components or pages
  - Called using CFINVOKE or inside a CFSCRIPT block/expression
  - Can be exposed as an entry point inside components

CFML Page Lifecycle, Part 1

- When a page is requested, search for (and execute) Application.cfc or Application.cfm first
- Application.cfm is a plain old CFML file, while Application.cfc defines hooks into application events
- Common uses for this mechanism:
  - Login management
  - Centralized data validation
  - Messing with session variables
  - Error handling

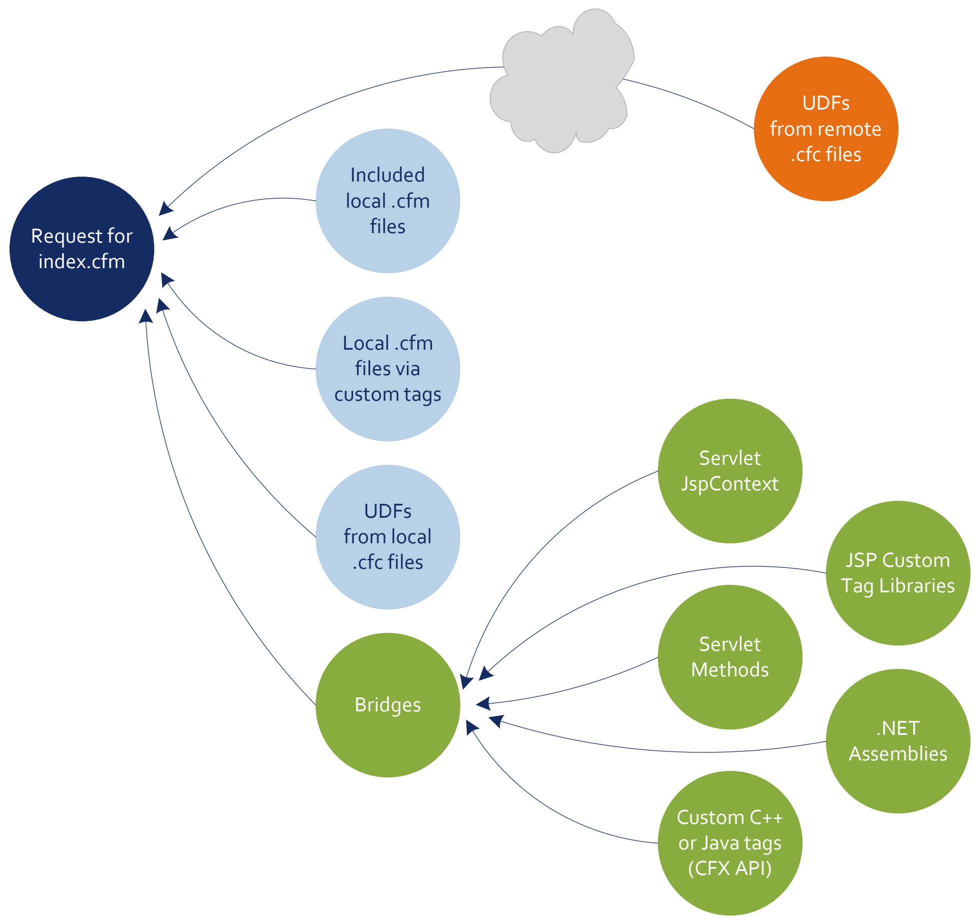


Inside Application.cfc

- **onApplicationStart**: application start (can access request variables)
- **onApplicationEnd**: application timeout/server shutdown
- **onSessionStart**: new session (can access request variables)
- **onSessionEnd**: session ends
- **onRequestStart**: called before every request (can access request variables)
- **onRequest**: called after onRequestStart code ends (can access request variables)
- **onRequestEnd**: called after request has been processed (can access request variables)
- **onMissingTemplate**: called when an unknown page has been requested (can access request variables)
- **onError**: when an uncaught exception occurs (can access request variables sometimes; check Event value)

CFML Page Lifecycle, Part 2

- A single page can include code from many different locations
- Custom tags are similar to local includes, but with different dataflow behavior
  - `<cf_foo>` is kind of like `<cfinclude template="foo.cfm">` except that changes made to variables are not visible in the calling page
- There are also built-in tags for interacting with remote HTTP, FTP, LDAP, SMTP, and POP servers



Request for index.cfm
UDFs from remote .cfc files
Included local .cfm files
Local .cfm files via custom tags
UDFs from local .cfc files
Bridges
Servlet JspContext
JSP Custom Tag Libraries
Servlet Methods
.NET Assemblies
Custom C++ or Java tags (CFX API)

Variables are Dynamically Scoped

- Silos of global variables named “scopes” can be confusing
- Variable accesses can be fully-qualified (prefixed with scope name) or not qualified at all

```
<cfoutput>#foo#</cfoutput>
<cfoutput>#URL.foo#</cfoutput>
```

- The unqualified scope can be temporarily “enhanced” with the results of a query row or loop iteration, e.g.

```
<cfquery name="qry" datasource="myDataSource">
  SELECT col1, col2, col3 FROM myTable
</cfquery>
<cfoutput query="qry">#col1#, #col2#, #col3#</cfoutput>
<cfoutput query="qry">#qry.col1#, #qry.col2#,
#qry.col3#</cfoutput>
```

- Output without iteration is also possible:

```
<cfoutput> #qry.col1#, #qry.col2#, #qry.col3# </cfoutput>
```

Variable Scopes

Scope	Description
Variables	the variable binding stack local to the current page
Application	global to every page in an app; set in application.cfc
Arguments	arguments to a function (may be tainted if called by a remote UDF)
Attributes	used to pass data to .cfm custom tag pages/threads
Caller	used within custom tags; reference to the calling page’s Variables scope
Request	persistent across all code for the lifetime of the request; useful within custom tags and cfincluded pages
This	struct/component “member variables”
ThisTag	analogous to Request scope for custom tag pages
URL	parameters present in HTTP query string
Form	parameters present in HTTP POST body
Cookie	HTTP request cookies
CGI	CGI variables, some server-defined and some tainted
Session	persistent across a single site visit
Client	client-specific persistent storage; outlasts session variables

Variable “types” in CF

- The CF type system hasn’t changed significantly since the 90s
- Implicit conversions to/from strings are the norm
- Instead of type checks, validation often done with pattern matches:
  - CFPARAM and CFARGUMENT “type” attributes
    - `<cfparam name="phoneno" type="telephone">` will throw an exception if “phoneno” is set and is not formatted as a standard US/NANPA phone number
    - Types “boolean”, “creditcard”, “date”, “time”, “eurodate”, “eurotime”, “email”, “float”, “numeric”, “guid”, “integer”, “range”, “regex”, “ssn”, “telephone”, “URL”, “uuid”, “usdate”, “variablename”, “xml”, “zipcode” all check the string representation of the variable against regexes
    - Limited type checks are possible: “array”, “query”, “struct”, and “string”
- Numerous opaque types reused among contexts
  - Example: queries are used for database queries, directory iteration, ldap queries, http/ftp requests, and others

CF Expressions

- Automatic interpolation with #-expressions inside cfoutput and attributes:
  - `<cfoutput>#URL.foo#</cfoutput>`

  -
    ```
    <cfloop query = "MyQuery" startRow = "#Start#" endRow =
    "#End#">
      <cfoutput>#MyQuery.MyColName#</cfoutput><br>
    </cfloop>
    ```
- Dynamic scoping can hinder analysis
  - `<cfset foo="bar">` vs. `<cfset "#foo#"="#bar#">`
  - `SetVariable("foo", "bar")` vs. `SetVariable(foo, bar)`
- Dynamic evaluation functions
  - Evaluate() and PrecisionEvaluate()
  - IIF()
  - DE() – used in conjunction with the other two

Practical Tips for Developers and Code Reviewers

Where to Look for Untrusted Data

- URL.any_variable
- FORM.any_variable
- COOKIE.any_variable
- FLASH.any_variable
- CGI.some_variables
  - e.g. PATH_INFO, QUERY_STRING, CONTENT_TYPE, CONTENT_LENGTH, HTTP_REFERER, HTTP_USER_AGENT, etc.
  - More on this later
- SESSION.some_variables
  - Depends on application logic
- CLIENT.any_variable
  - Only when client variables are enabled and storage is cookie-based
- CFFUNCTION arguments, when access="remote"

XSS Defense Misconceptions

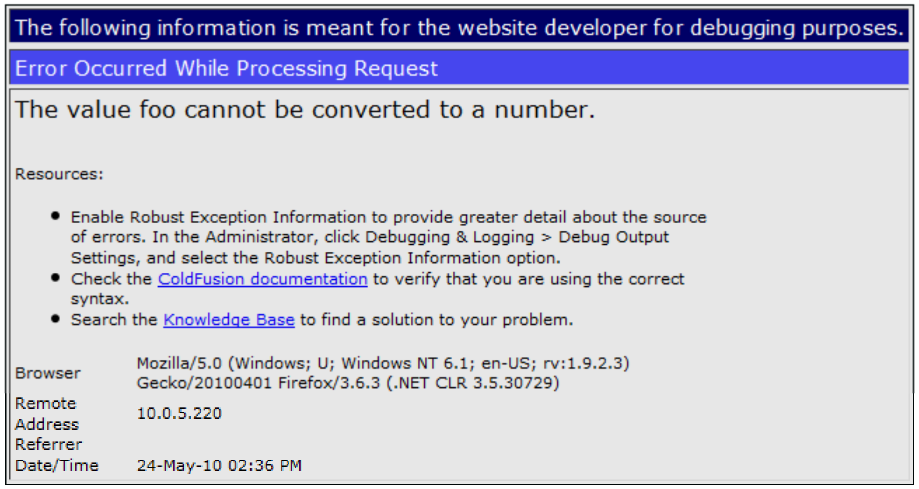
- Using scriptProtect attribute
  - Replaces blacklisted tags such as <script>, <object>, etc. with <InvalidTag> when rendering user-supplied input
  - Doesn't block injection, aside from the most basic attack strings
- Example
  - `<cfapplication scriptProtect="all">`
    `<cfoutput>You typed #URL.foo#</cfoutput>`
  - Requesting page with ?foo=<script>alert("foo")</script> will return You typed <InvalidTag>alert("foo")</script>
- Trivial to circumvent
  - One of many possibilities: requesting page with ?foo=<img src="http://i.imgur.com/4Vp9N.jpg" onload="alert('foo')"> will happily execute the alert() call
- Other regexes can be added to the blacklist, but it's still a blacklist (look for neo-security.xml if you insist)

XSS Defense Misconceptions

- Assuming HTMLEditFormat() and HTMLCodeFormat() perform sufficient HTML encoding
  - They only encode <, >, ", and &
  - Ineffective for unquoted or single-quoted tag attributes, or within script blocks
    - `<img #HTMLEditFormat(URL.foo)#>`
    - `<img alt='#HTMLEditFormat(URL.foo)#'>`
    - `<script>#HTMLEditFormat(URL.foo)#</script>`
    - `<script>var x='#HTMLEditFormat(URL.foo)#';</script>`
    - etc.
  - XMLFormat() encodes single quotes, but still won't prevent XSS in all situations
    - Inside Javascript blocks

XSS Risks in Default Error Pages

- This is effective whitelist-style input validation, right?
  - `<cfoutput>#int(URL.count)#</cfoutput>`
  - `<cfset safenum=NumberFormat(FORM.bar)>`
  - `<cfoutput>#JavaCast("boolean", URL.booly)#</cfoutput>`
- Default error page
  - scriptProtect is enabled on the default error page, but we already saw how (in)effective that is



```
The following information is meant for the website developer for debugging purposes.
Error Occurred While Processing Request
The value foo cannot be converted to a number.

Resources:

  • Enable Robust Exception Information to provide greater detail about the source
    of errors. In the Administrator, click Debugging & Logging > Debug Output
    Settings, and select the Robust Exception Information option.
  • Check the ColdFusion documentation to verify that you are using the correct
    syntax.
  • Search the Knowledge Base to find a solution to your problem.

Browser      Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.3)
             Gecko/20100401 Firefox/3.6.3 (.NET CLR 3.5.30729)
Remote       10.0.5.220
Address
Referrer
Date/Time    24-May-10 02:36 PM
```

XSS Risks in Custom Error Handling

- Using casts to sanitize input can be ineffective unless the application also defines an error page

  ```
  <cferror template="errorhandler.cfm" type="request">
  ```

  Don’t use #error.diagnostics# or #error.message# in your error page!

- Exception handling also works

  ```
  <cftry>
    <cfoutput>#int(URL.count)#</cfoutput>
    <cfcatch>Exception caught!</cfcatch>
  </cftry>
  ```

  Don’t output #cfcatch.message# in your catch block without properly encoding it first!

Common SQL Injection Mistakes

- Using CFQUERY without CFQUERYPARAM (also CFSTOREDPROC without CFPROCPARAM)

```
<cfquery name="getContent" dataSource="myData">
  SELECT * FROM pages WHERE pageID = #Page_ID# OR
  title = '#Title_Search#'</cfquery>
```

- #Title_Search# is not injectable; CF will automatically escape single quotes for expressions inside the CFQUERY tag
- #Page_ID# is still injectable because it's not quoted
- Using CFQUERYPARAM

```
<cfquery name="getContent" dataSource="myData">
  SELECT * FROM pages WHERE pageID =
  <cfqueryparam value="#Page_ID"
cfsqltype="cf_sql_integer"></cfquery>
```

(For unknown reasons, cfsqltype is an ***optional*** attribute)

Other OWASP Top Ten Vulnerabilities

- We won't waste time rehashing all of the common web vulnerabilities
  - Of course you can have CSRF, insecure cryptographic storage, broken authentication, etc. in a ColdFusion app
  - Nothing unique enough to warrant discussion here
- Here are some dangerous tags; it should be obvious why they are dangerous if not properly restricted
  - <cffile>
  - <cfdirectory>
  - <cfexecute>
  - <cfregistry>
  - <cfobject>

Directly Invoking UDFs

- Every method in a .cfc file is a potential entry point, e.g. http://example.com/foo.cfc?method=xyzzy&arga=vala&argb=valb
- This URL will invoke method xyzzy on an anonymous instance of component foo.cfc, with arguments arga="vala" and argb="valb" (also valid with POST variables, although method must be passed in the query string)
  - If method doesn't exist, onMissingMethod is called
  - If method isn't specified, then the request gets redirected to CFIDE/componentutils/cfcexplorer.cfc
  - Rules for application.cfc and application.cfm still apply
- In a source code review, look for sensitive functionality implemented as UDFs, with the access attribute set to "remote"

```
<cffunction name="ListCategories" access="remote"
returntype="query">
```

Evaluating Unscoped Variables

- If you use a variable name without a scope prefix, ColdFusion checks the scopes in the following order to find the variable:
  1. Variables (local scope)
  2. CGI
  3. URL
  4. Form
  5. Cookie
  6. Client
- For example, in applications with sloppy variable naming, you can almost always override POST (Form) parameters with GET (URL) parameters
- Compare reads/writes of variables to identify scoping inconsistencies

Source: http://www.adobe.com/devnet/server_archive/articles/using_cf_variables2.html#scopetable

Exploiting Unscoped Variables

- A generic case:

```
<cfif IsDefined("importantVar")>
  DoImportantStuff()
<cfelse>
  Sorry, you are not permitted to access this functionality.
</cfif>
```

- Putting ?importantVar=anything in the URL (or the POST body, or a cookie) will bypass this check if importantVar has not already been defined in the Variables scope

Exploiting Unscoped Variables

- Consider this logic to process a user login (yes, it's contrived)

```
<cfif AuthenticateUser(FORM.username, FORM.password) and
          IsAdministrator(FORM.username)>
  <cfset Client.admin = "true">
<cfelse>
  <cfset Client.admin = "false">
</cfif>
```

- Other pages check whether the admin variable is true before performing restricted actions

```
<cfif admin eq "true">
  Put privileged functionality here!
<cfelse>
  Sorry, only admins can access this!
</cfif>
```

- Putting ?admin=true in the URL will bypass this check because URL variables precede Client variables in the search order

Undefined Variables

- Similarly, ensure that variables are always initialized properly
- CFPARAM's "default" attribute only sets a variable if it's not set already; use CFSET or an assignment inside cfscript
- Assume undefined, unqualified valuables are filled with request data!
- It's common to see code like:

```
<cfparam name="pagenum" default="1">
<cfoutput>
  Now showing page #pagenum#.
</cfoutput>
```

- This is exploitable; GET and POST variables will override pagenum

Environment Variables

- Legitimate variables in the CGI scope can be manipulated and in some cases overridden via HTTP headers
- For example:

```
GET /index.cfm HTTP/1.0
Host: example.com
```

  The CF expression #CGI.HTTP_HOST# will contain “example.com”

```
GET /index.cfm HTTP/1.0
HTTP_HOST: evil.com
Host: example.com
```

  The CF expression #CGI.HTTP_HOST# will contain “evil.com”
- You can also override #CGI.SERVER_SOFTWARE#, #CGI.PATH_INFO#, #CGI.WEB_SERVER_API#, and many others
- Be particularly careful with #CGI.AUTH_USER#

Persistence Issues

- Client scope variables can be configured in Application.cfm in the CFAPPLICATION tag (attribute “clientmanagement”) or this.clientmanagement in Application.cfc
  - Keyed to browser via CFTOKEN/CFID cookies; actual variable storage may be client-side (other cookies) or server-side (in a database or the Windows registry)
  - All of these cookies persist by default, so watch for cookie theft/stuffing attacks
- When client scope is enabled, tampering is possible if cookie storage is enabled (“clientStorage” attribute/variable)
  - No encryption or MAC; plain text

ColdFusion Behind the Curtain

Proprietary Classfile Format

- CF can compile pages/components to sets of Java classes using the cfcompile utility
- One class per page plus one for every UDF
- All class generated for a single CFM/CFC file are placed in one file, concatenated; a custom ClassLoader is used by CF to load them up
- Names of the resulting concatenated files are identical to those of the source files
- Separately, ColdFusion Administrator can be used to bundle a directory as an EAR/WAR

A Way to Slice Them: cfexplode

- Free, open-source Java utility, on Google Code as of now: http://code.google.com/p/cfexplode/
- Splits concatenated classfiles into many; can accept individual compiled CFC/CFM files or full WAR/EAR/JAR zip archives

```
% java -jar cfexplode.jar outdir index.cfm
% ls -l outdir
total 40
-rw-r--r-- 1 cstone cstone  3534 2010-07-16 15:23
index.cfm.0.class
-rw-r--r-- 1 cstone cstone  2095 2010-07-16 15:23
index.cfm.3534.class
-rw-r--r-- 1 cstone cstone 31234 2010-07-16 15:23
index.cfm.5629.class
```

- Individual classes easily analyzable (even with the free JAD and JD-GUI)

Page/Component/Function Java Classes

- CFM/CFC: main point of entry is CFPage.runPage()
  - Other methods called beforehand set up data: variable bindings (bindPageVariables()), function names (registerUDFs()), data sources
- <cffunction>: main point of entry is UDFMethod.runFunction()
  - Argument validation is done by the runtime; any types specified in <cfargument> tags are translated into a static Map instance named "metaData"
- CfJspPage (base class).pageContext is a plain old JspContext, so pageContext.getOut() returns a JspWriter; this is used to do the bulk of the output
  - getOut() also used for things that aren't actually output to the screen, such as database queries
- Occasionally, parts of the body are factored out of runPage into separate private methods named factor0(), factor1(), factor2()..

CF Variables in Java: Static References

- Static references, usually used for local bindings

```
<cfset vfoo="value 1">
<cfparam name="pbar"
  default="value2">
<html>
  <cfoutput>
    vfoo: #vfoo#   pbar:
#pbar#
  </cfoutput>
</html>
```

- When compiled:

```
protected final Object
runPage()
  {
    // …
    VFOO.set("value 1");
    _whitespace(out, "\n");

checkSimpleParameter(PBAR,
"value2");
    out.write("\n\n<html>\n
");
    // …
    out.write("\n
vfoo: ");
    out.write(Cast._String(
_autoscalarize(VFOO)));
    out.write("   pbar: ");
    out.write(Cast._String(
_autoscalarize(PBAR)));
    _whitespace(out, "\n
");
```

CF Variables in Java: Static References

- How variables are bound to the page

```
private Variable PBAR;
private Variable VFOO;
protected final void bindPageVariables(VariableScope varscope,
                                       LocalScope locscope)
  {
    super.bindPageVariables(varscope, locscope);
    PBAR = bindPageVariable("PBAR", varscope, locscope);
    VFOO = bindPageVariable("VFOO", varscope, locscope);
  }
```

CF Variables in Java: Dynamic References

- Dynamic references, explicitly-scoped variables

```
<html>
<cfoutput>
  #url.quux#
</cfoutput>
</html>
```

- When compiled:

```
protected final Object runPage()
  {
    // …
    out.write("<html>\n    ");
    _whitespace(out, "\n        ");
    out.write(Cast._String(
              _resolveAndAutoscalarize("URL", _new String[] {
"QUUX” }))    );
    // …
  }
```

Other Ways to Set/Access Variables

- Bind the name "scope" to a variable that represents the results of the query
  - `<cfquery name="scope">`
- Looping over query results
  - `<cfoutput query="resultset">`
  - `<cfloop query>`
- Structure member accesses
  - `<cfset x=StructNew()>`
  - `<cfset x.member="val1">`
- <cfdump> tag for dumping variable contents
- Other I/O: files, HTTP requests, LDAP requests, mail messages

WAR/Application Structure

- CFMs/CFCs handled by different Servlets (CfmServlet and CFCServlet, respectively)
- These locate the class(es) necessary based on URL and parameters, then invoke their runPage()/runFunction() methods
- Chain of coldfusion.filter.FusionFilter classes (not related to J2EE Servlet filters); these handle client-scope propagation
- Even if the “Include CF Administrator” option is unchecked, many pages/components inside the CFIDE/ directory are included inside every WAR
  - Mapped by default
  - Access may not be password-protected; easily disabled by a change to neo-security.xml (see http://kb2.adobe.com/cps/404/kb404799.html)

WAR Structure: Other Servlets

- *.jsp: JSPLicenseServlet; passthrough for jrun.jsp.JSPServlet
- /flex2gateway/*, /flashservices/gateway/*, /CFFormGateway/*: FLEX/plain Flash Remoting gateways for CFC methods
  - /flashservices/gateway/path1.path2.component ⇒ path1/path2/component.cfc
  - Gateways can be used in ActionScript NetServices.createGatewayConnection()
  - Used internally by <cfgrid> and other built-in cf tags that generate Flash-based UI automatically
- GraphServlet: handles /CFIDE/GraphData.cfm (not actually a cfm file); used by the cfchart tag.
- CFFileServlet: handles /CFFileServlet/*, and serves up files from a cache directory; used by <cfimage>
- /cfform-internal/*: FLEX FileManagerServlet; serves a handful of dynamically-generated images and js files
- /WSRPProducer/*: WSRP portlet management Axis service

Final Thoughts

Conclusions

- ColdFusion designed to be simple for "developers" to use, but it's actually very complicated underneath
- It's easy to make coding mistakes (or overlook vulnerabilities during code review) if you don't understand ColdFusion internals
  - Request lifecycle
  - Error handling
  - Variable scopes and precedence
- Like many web application platforms, ColdFusion has a bunch of "features" that are useful for debugging but also open up holes
- ColdFusion-generated Java classes are pretty ugly; use **cfexplode** to help reverse engineer them
- The attack surface is huge by default; strip out unnecessary components before deploying

More Resources

- Whitepapers, webcasts, and other educational resources
  - http://veracode.com/resources
- Veracode ZeroDay Labs Blog
  - http://veracode.com/blog
- Download the cfexplode tool
  - http://code.google.com/p/cfexplode/
- Contact info
  - Email: ceng@veracode.com, bcreighton@veracode.com
  - Twitter: @chriseng, @unsynchronized
  - Phone: 781.425.6040