PRACTICAL APPLICATION SECURITY

ColdFusion Application Security Practices and Coding

- Bilal Soylu

Agenda

- Introduction
- Landscape
- Best Practices Guidelines
- Frequent Parts (URL, FORM, SESSION)
- Examples and Code
- Beyond the Platform (ESAPI)
- Closing

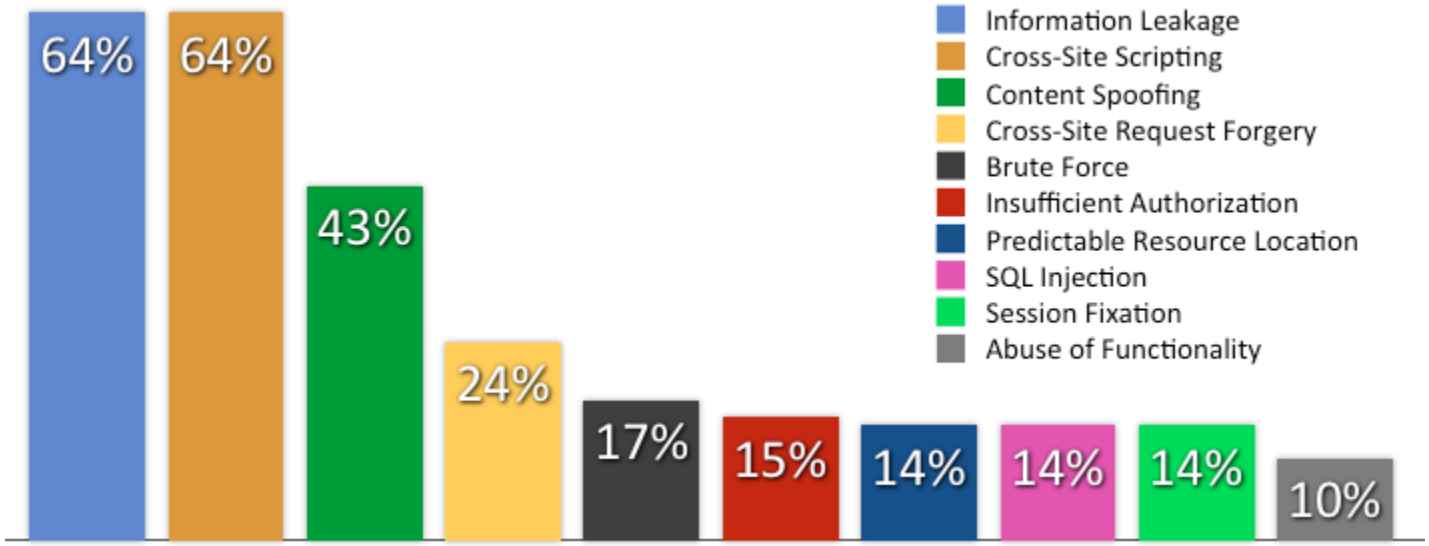
Introduction

- Bilal Soylu
  - CTO Verian Technologies LLC (www.verian.com)
  - ColdFusion since mid 90s
  - Open Source contributor
  - Enough mistakes to know better ;o)
- Email
  - bilal.soylu [at] gmail.com
- Blog
  - http://BonCode.blogspot.com
- Twitter
  - @BmanCLT

Security is a common challenge

- Many applications have security issues regardless of platform (YouTube, Blogger, LiveSearch)
- Thinking about security comprehensively is actually the best way to achieve secure applications
- Writing insecure code is easy
  - Time
  - Budget
  - Knowledge
  - Lunch

Overall Top Ten Vulnerability Classes of 2010

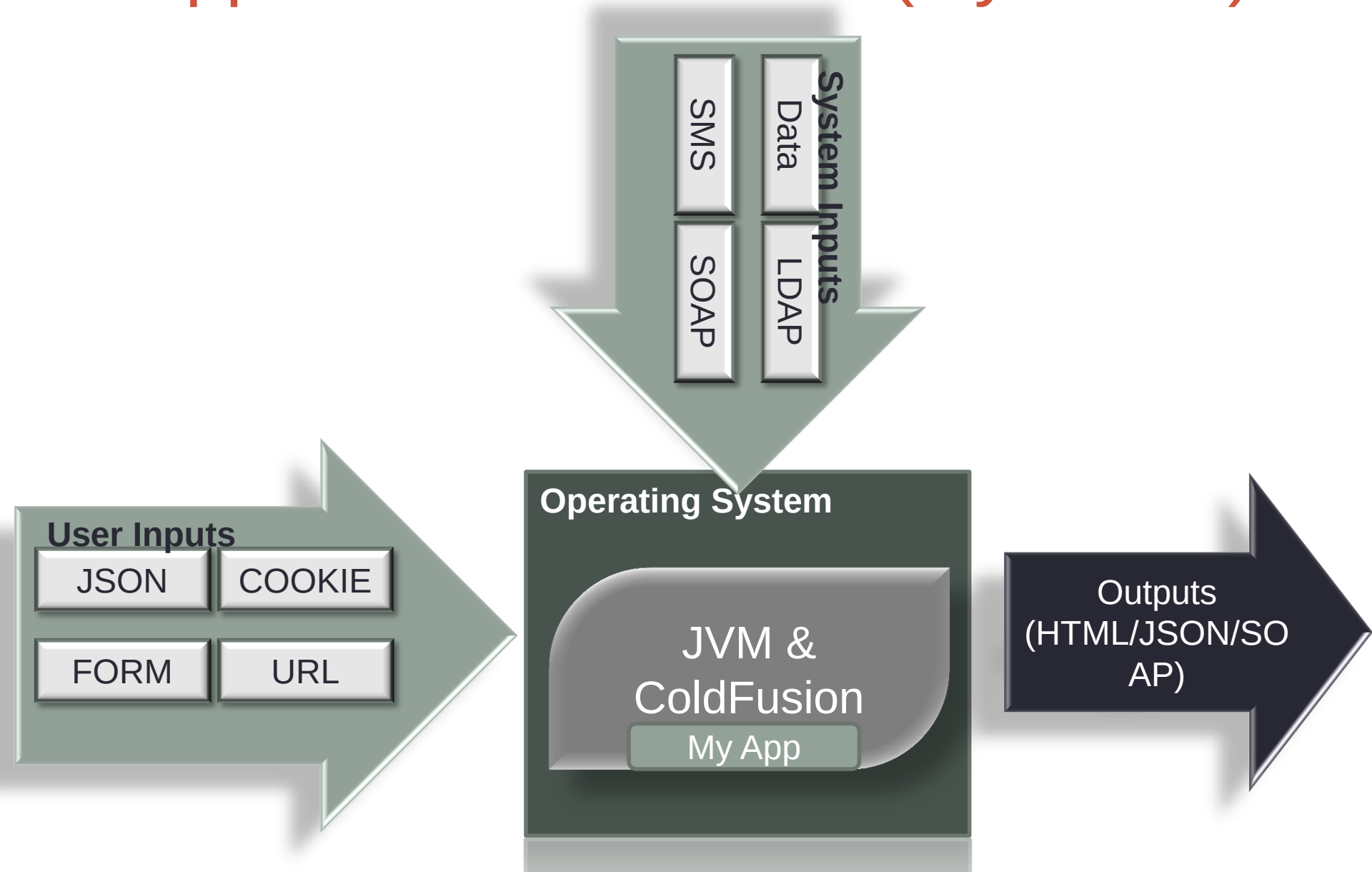


(Percentage likelihood that at least one vulnerability will appear in a website)
- White Hat Security Report Winter 2011

Classes of Code Vulnerability (the how to code)

*from Aspect Security

No Application is an Island (My Model)



Common Framework

- OWASP (www.owasp.org)
  - Open Web Application Security Project
- Using Top Ten (Ranked by Severity)

OWASP
The Open Web Application Security Project

Current Top 10

- A1: Injection (SQL) – User Input
- A2: Cross-Site Scripting (XSS) - User Input
- A3: Broken Authentication and Session Management - Logic
- A4: Insecure Direct Object References – User Input
- A5: Cross-Site Request Forgery (CSRF) – User Input
- A6: Security Misconfiguration - Logic
- A7: Insecure Cryptographic Storage - Knowledge
- A8: Failure to Restrict URL Access – System Input
- A9: Insufficient Transport Layer Protection – System Input
- A10: Unvalidated Redirects and Forwards - User Inputs
- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Platform got me covered, right

- Why do I have to worry. Look at this:

☑ **Enable Global Script Protection**
Specify whether to protect Form, URL, CGI, and Cookie scope variables from cross-site scripting attacks.

☐ **Prefix serialized JSON with** //well
Protects web services which return JSON data from cross-site scripting attacks by prefixing serialized JSON strings with a custom prefix.

DATA Context Elements in HTML

- DATA
  - Output Context
    - Between HTML tags
  - Attributes
    - <p **align**="#form.align#">some text</p>
  - JavaScript (DOM events)
    - <script type="text/javascript">alert('hello world')</script>
    - <div **onfocus**="this.style.color='#form.color#'>
  - CSS
    - .myCss { color: #form.color# }
  - URL Parameter
    - Color
  - FORM
    - <input type="hidden" id="prevValue" **value**="#form.previousEntry#">

XSS – A2

- Most common vulnerability in web-apps
- Target is other users
- Break out context into the other
  - Data : for display to user
  - Code: for execution (running your logic)
- Common example, using vulnerability in your app to distribute a script to others
  - XSS is possible without <script> tag

☑ **Enable Global Script Protection**
Specify whether to protect Form, URL, CGI, and Cookie scope variables from cross-site scripting attacks.

XSS Example

- Demo

Easy Hacks : Some Common Trouble

- Sessions are always mine (A3)
- I am good with Files (A10?)

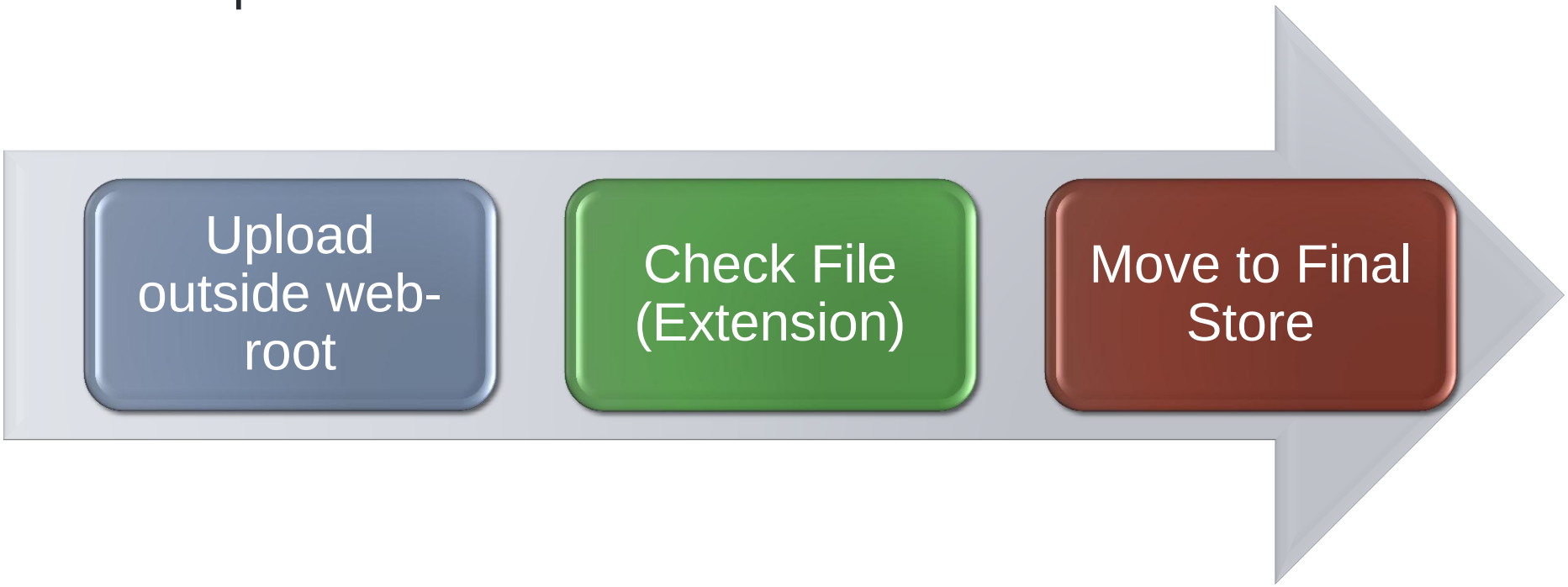
```
<cfhttp  method="post" url="http://localwheels/sec/fileProcess.cfm"
throwonerror="Yes">

    <cfhttpparam name="fileToLoad" type="file"
 file="#ExpandPath("badFile.jsp")#" mimeType="image/jpg">

</cfhttp>
```

A Safer File Upload:

- File Upload



Session Platform Measures

- Don’t pass in URL (addToken=false in CFLOCATION)
- Validate with cross checked or encrypted stored cookie (see below)
- Switch to JEE or UUID tokens
- Use HTTP only Session Cookies
  - -Dcoldfusion.sessioncookie.httponly=true on CF 9.0.1
- Consider using SSL when authenticated (prevent sniffers)

Session Application Measures

- Good
  - Use Application Logic to check against hi-jack

- Best
  - Create new Session once Authenticated (destroy old)
  - Use Fingerprinting to identify the client to which session was issued, e.g. http://panopticlick.eff.org/

Injection (SQL) – (A1)

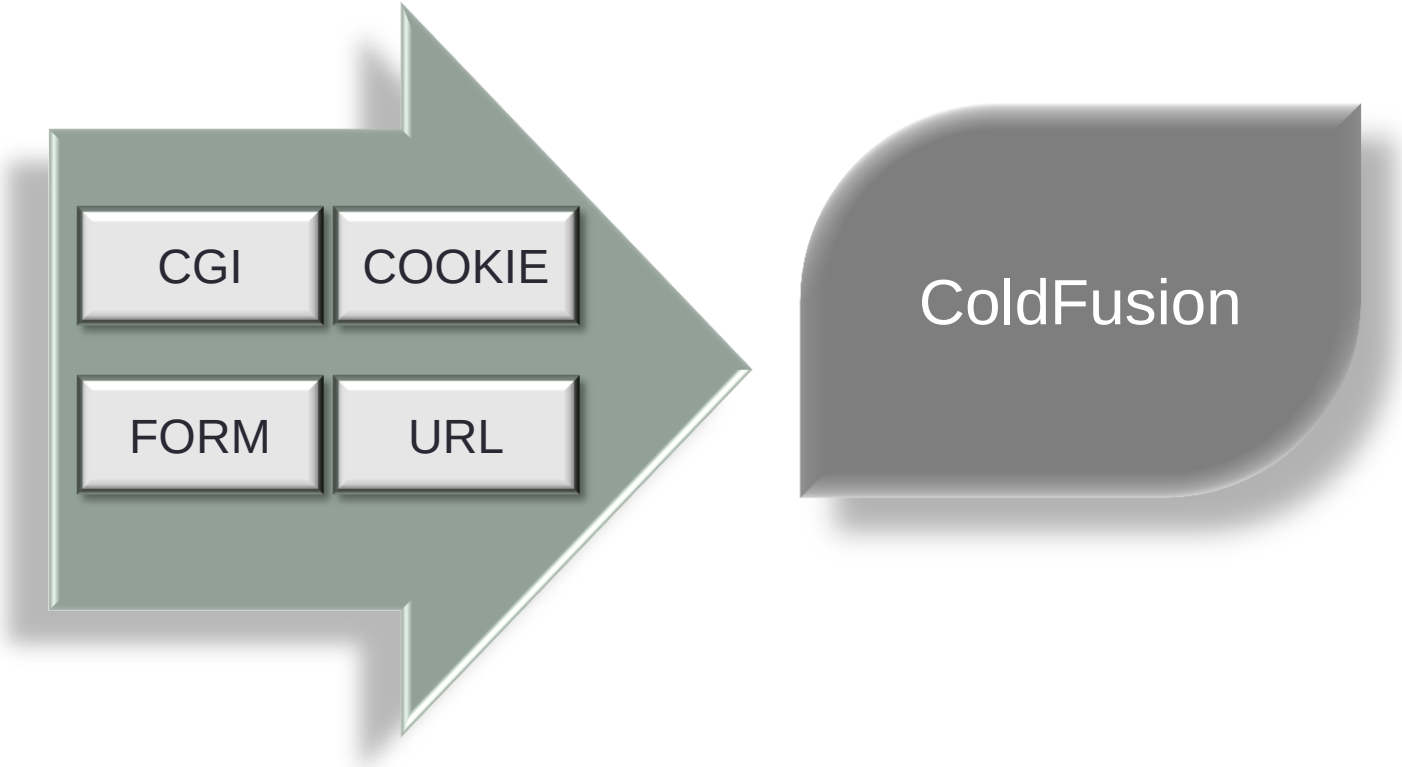
- Target is database

```
BAD:
<cfquery>
  SELECT * FROM accounts WHERE custID='#Form.custID#'
</cfquery>
```

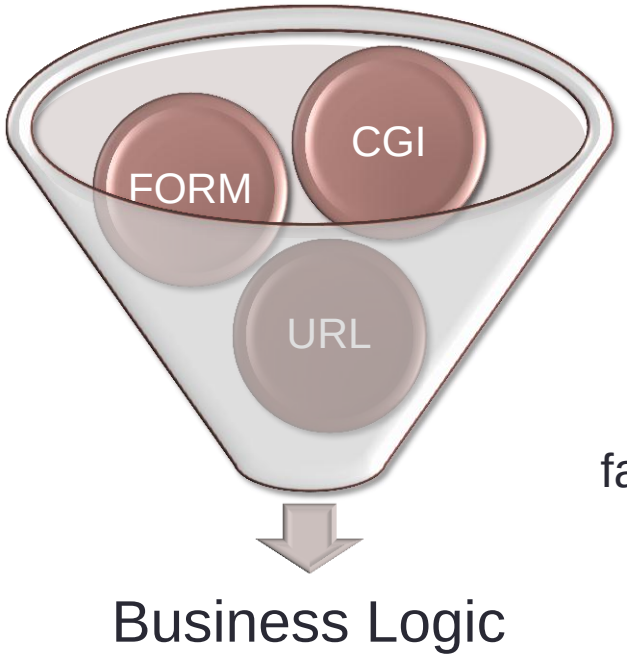
```
GOOD:
<cfquery>
  SELECT  accountName FROM accounts WHERE custID=
  <cfqueryparam value="#Form.custID#" cfsqltype="CF_SQL_INTEGER">
</cfquery>
OR (CF9 and Token Placeholder)
myQuery.setSQL("SELECT accountName FROM table WHERE ID=:myID");
```

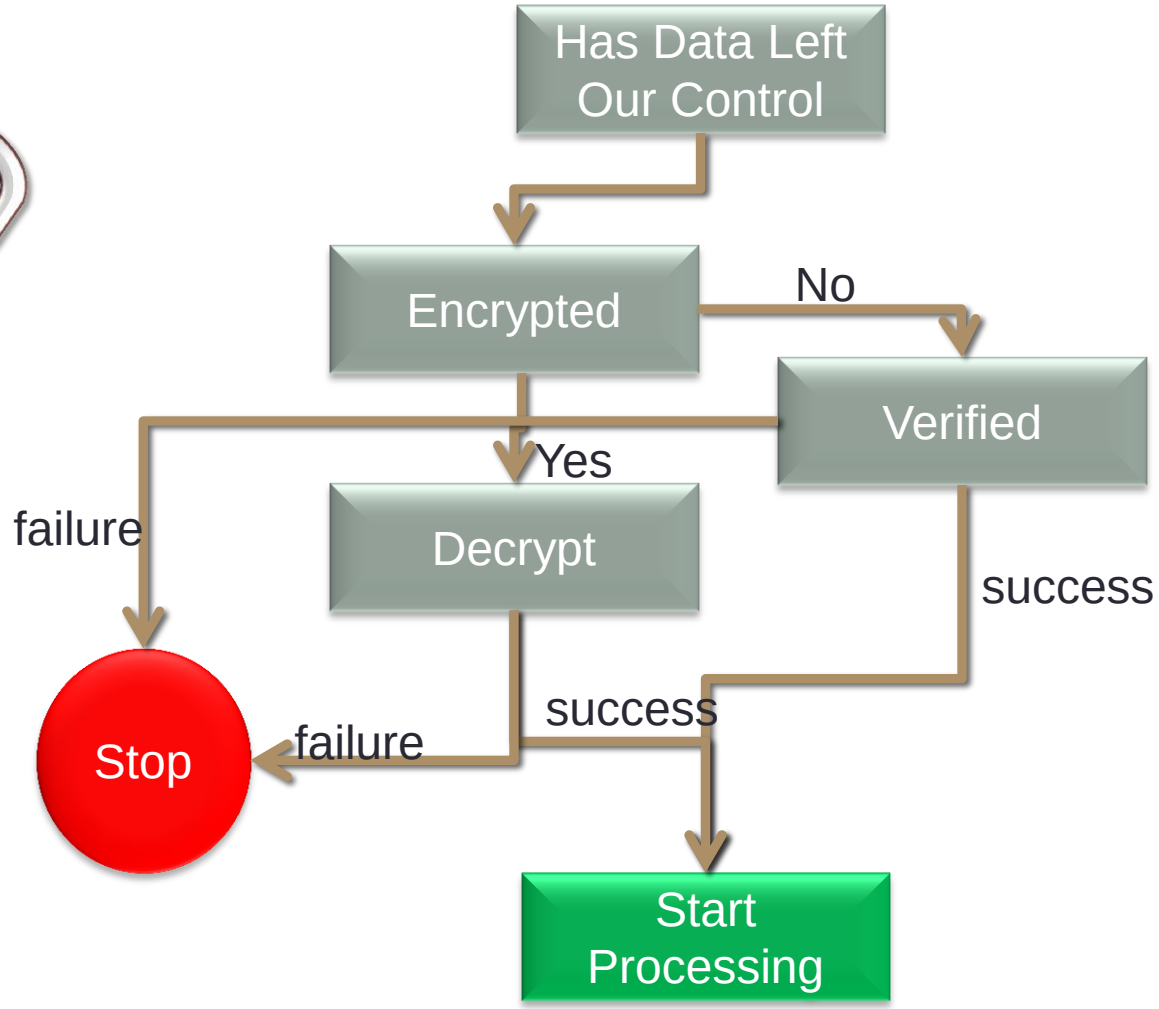
Do we really know where you've been last night?

- Stateless nature of http causes loss of insight into transferred data to client
- Common Scopes with loss of control
  - CGI
  - COOKIE
  - FORM
  - URL
  - CLIENT (?)



Establish a Chain of Trust





How to Regain Control / Trust

- How to re-establish trust
  - Outbound: Encryption ➔ secure encryption
  - Inbound: Validation
    - Type
      - Numeric, Date, String
    - Content / Scope
      - Number Range, Date Range, distinct string values, e.g. Pass valid Whitelist in encrypted form for non-sequential selections)
      - Whitelist, whitelist, whitelist

- Examples for Form and URL

URL

Common use:
http://www.myserver.com/mypage.cfm?userid=299&Pass=hello
Vulnerable to A2:XSS and A5:CSRF (cross site request forgery)

Better Encrypt all URL variables:

http://www.myserver.com/mypage.cfm?Package=383%83N%3948

Use OnRequestStart in Application.cfc to decode and place in separate scope/struct:
e.g. Request.URL

Add on:
Add a timestamp for how long this URL Package is valid from issuance
Example Implementation: http://urlencoder.riaforge.org/

Form Scope

Common use:
<input type=“hidden” name=“id” value=“22”>

Passing all Form variables into a component:

```
myCFC = CreateObject("component","processor");
myCFC.process(argumentcollection=Form);
```

Better:

<input type=“hidden” name=“id” value=“#encryptedValue#”>

Or whitelist data (small data set):

```
<cfwddx action="cfml2wddx" input="myData" output="serializedData">
<cfset formData=URLEncodedFormat(Encrypt(serializedData,"#CGI.REMOTE_ADDR#"))>
<cfset type=“hidden” name=“whiteList” value=“#formData#”>
```

Indicating trust within your code

- Use generic URL / FORM encryption function
- Once inputs have been validated or secured put them into a different scope, e.g.:
  - Request.URL
  - Request.Form

Outputting Data

- Still use Global Script protection
- Important to know where we ware using user generated data (context)
- Outputting data from an uncontrolled / un-trusted input will lead to common XSS scenarios.
  - Only output from verified scope (e.g. Reques.URL), whitelist, whitelist, whitelist
- Output data requires context awareness
  - In data context: XMLFormat()
    - Welcome #XMLFormat(Form.UserName)#
  - URL Context
    - Col

Outputting: ESAPI (Enterprise Security API)

- Java library OWASP project
- Installation is not trivial
- A port to CF (CFESAPI) is in progress (rumor CF10/Zeus)
- In HTML Attributes (between double quotes):
  - myLink
  - encoderForHTMLAttribute(formString)
- JavaScript Context (+DOM Events)
  - <div onfocus="this.style.color='#form.color#'>
  - encodeForJavaScript(form.color)
  - There are 1,677,721,600,000,000 ways to encode <script> tag
- CSS Context
  - .myCss { color: #form.color# }
  - encodeForCSS(form.color)
- URL Context
  - Color
  - encodeForURL(form.color)
- ESAPI has more stuff, e.g. command line, SQL etc.

Keep current with updates

Security experts estimate that barely 50 percent of all software security patches are applied by enterprise IT administrators.

***** eSecurityPlanet.com

- For all the elements outside our direct control
  - Operating Systems
  - Databases
  - Application Servers

Conclusions

- Most web attack vectors are based on developer logic errors
- Establishing trust in your inputs will go a long way in securing your applications
- You can have coding practices indicate to you if inputs have been secured/validated.
- Outputting data needs to be context sensitive
- OWASP is superset of guidelines that we should be familiar with.

Resources

- OWASP (www.owasp.org)
  - Nice video episodes
- Adobe (http://www.adobe.com/support/security/)
- My Blog (http://boncode.blogspot.com)
- CFESAPI
  - https://www.owasp.org/index.php/ESAPI_ColdFusion_CFML_Readme
- ESAPI
  - https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
  - https://www.owasp.org/images/7/79/ESAPI_Book.pdf

THANK YOU

Q & A

More Stuff

- Passwords
  - Use Hash (SHA-256 or SHA-512)… you need longer storage, add salt
- Encryption
  - Use strong encryption method, including salt and IV to introduce variability
    - CFMX_COMPAT is bad
    - TripleDES, AES, Blowfish are good, AES fast
    - Use CBC mode of operation
  - For stronger encryption need to change Java Policy files changed