# PRACTICAL APPLICATION SECURITY

ColdFusion Application Security Practices and Coding

- Bilal Soylu, CFCamp 2012

## Agenda

- Hola!
- Why it matters: The current landscape
- The Language of Security
- My Practical Methodology
- Frequent Parts (URL, FORM, SESSION, FILE)
- Examples and Code
- Random Ramblings
- 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

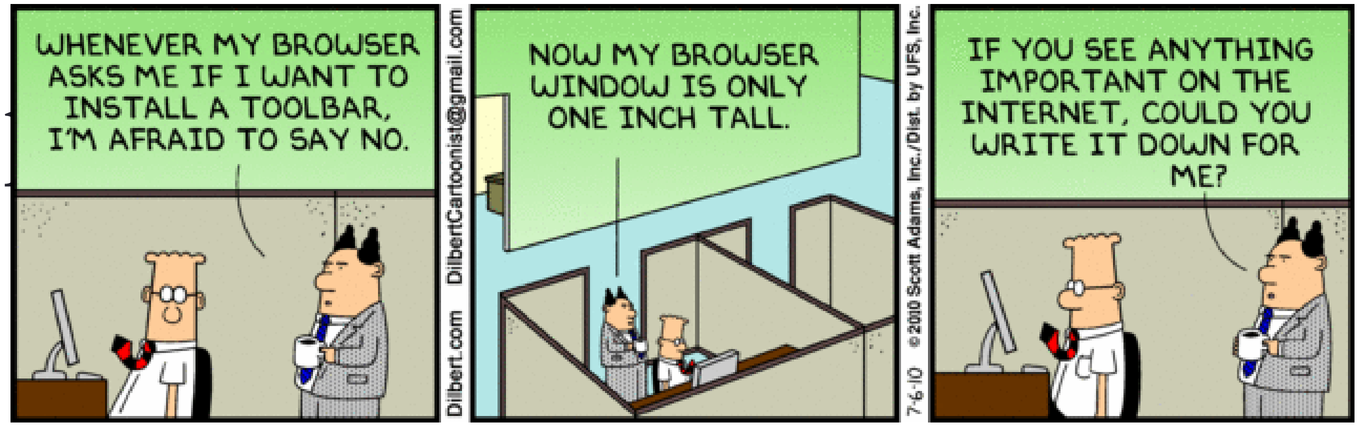
- Twitter
  - @BmanCLT

## Security is a common challenge

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

| | |
|---|---|
| Time | Budget |
| Knowledge | Lunch |
| We have a life | We are stressed enough |

# “T ce”


WHENEVER MY BROWSER ASKS ME IF I WANT TO INSTALL A TOOLBAR, I'M AFRAID TO SAY NO.
Dilbert.com DilbertCartoonist@gmail.com
NOW MY BROWSER WINDOW IS ONLY ONE INCH TALL.
7-6-10 © 2010 Scott Adams, Inc./Dist. by UFS, Inc.
IF YOU SEE ANYTHING IMPORTANT ON THE INTERNET, COULD YOU WRITE IT DOWN FOR ME?

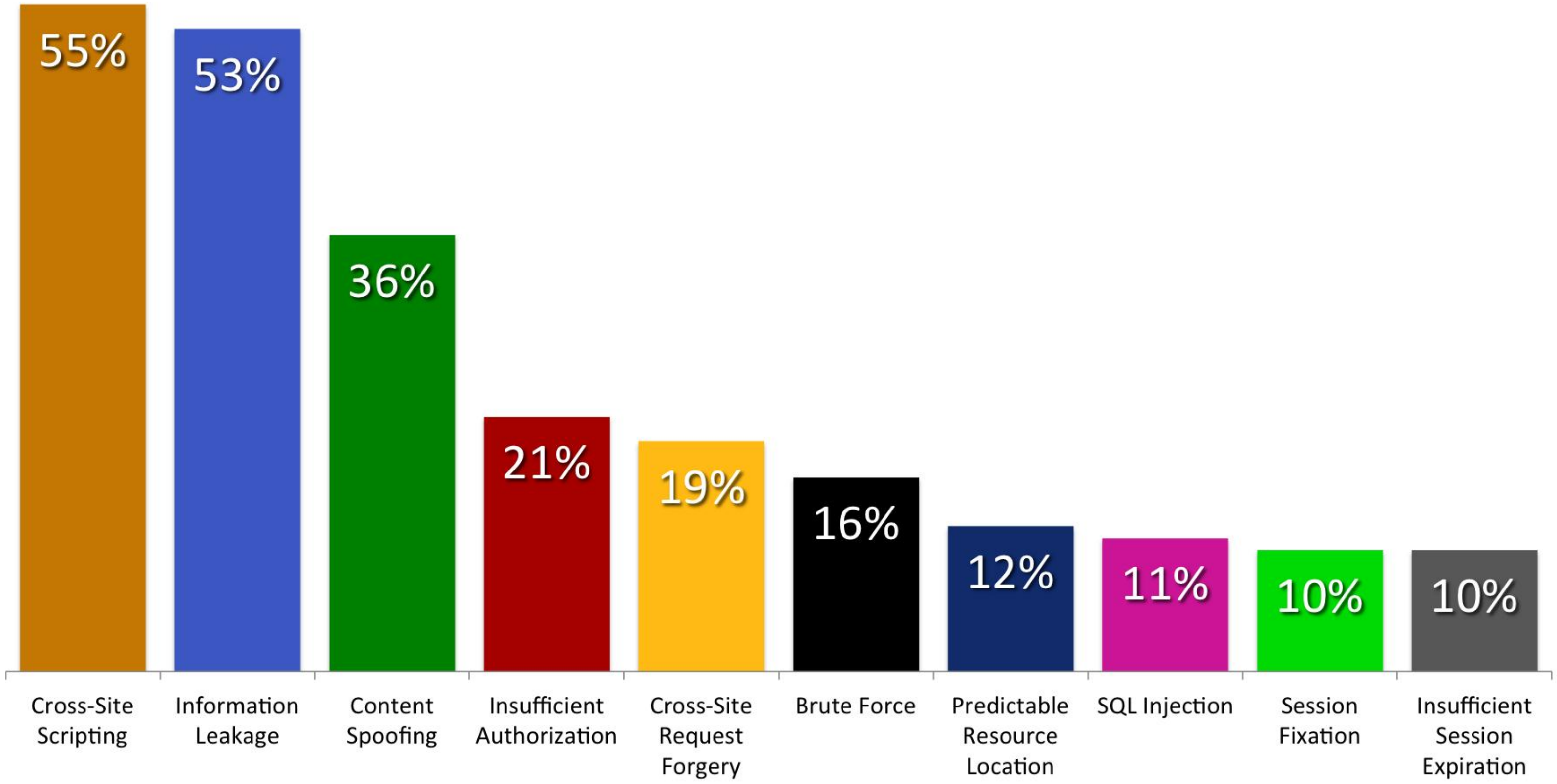
*”The Geography of HTML5 Security”,
Mike Shema, Techcrunch

## Common Framework / Language

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

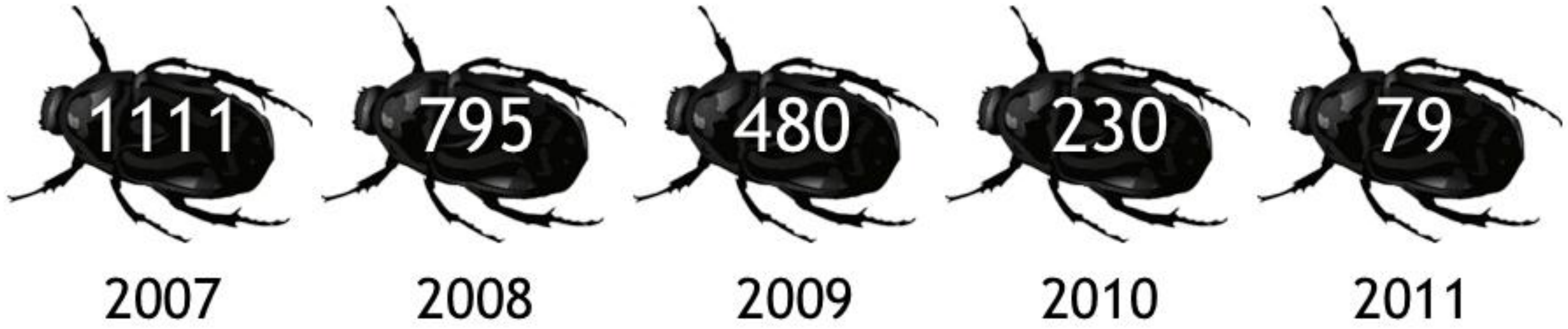
OWASP
The Open Web Application Security Project

# Overall Top Ten Vulnerability Classes of 2011



(Percentage likelihood that at least one vulnerability will appear in a website)
- White Hat Security Report Summer 2012

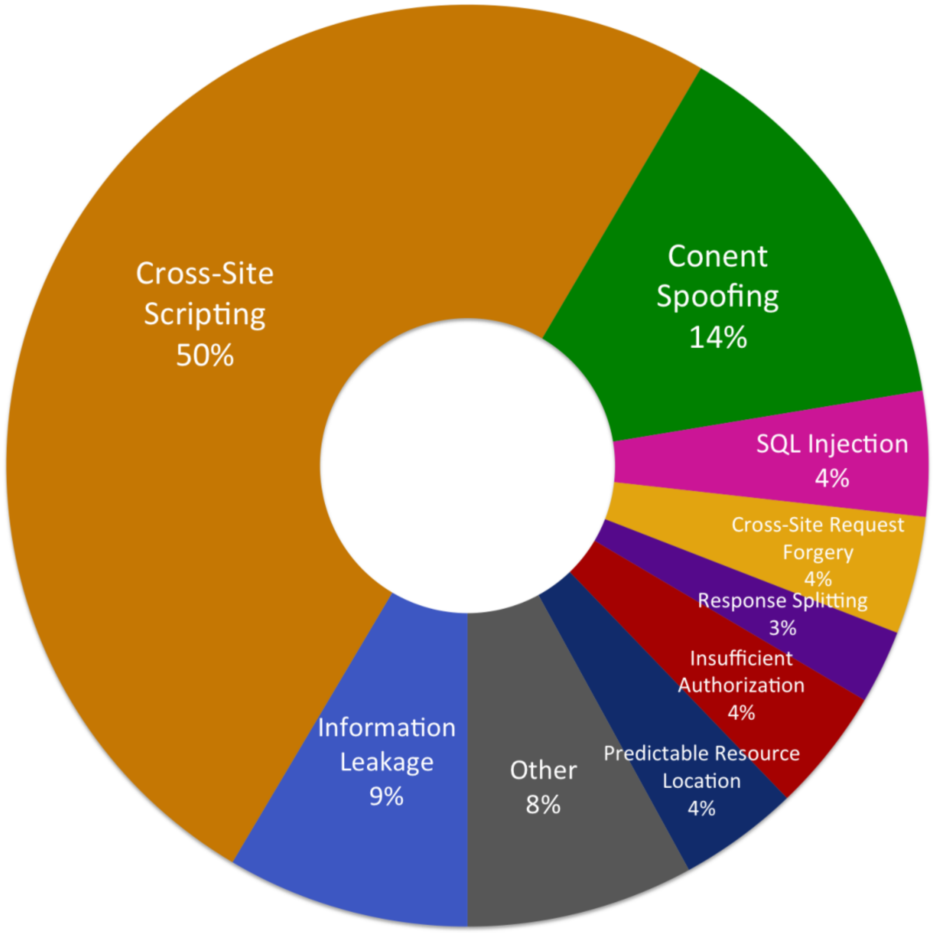
## Awareness of Security Practices Improving ?



Vulnerability Historical Trend -- The annual average number of serious vulnerabilities discovered per website per year

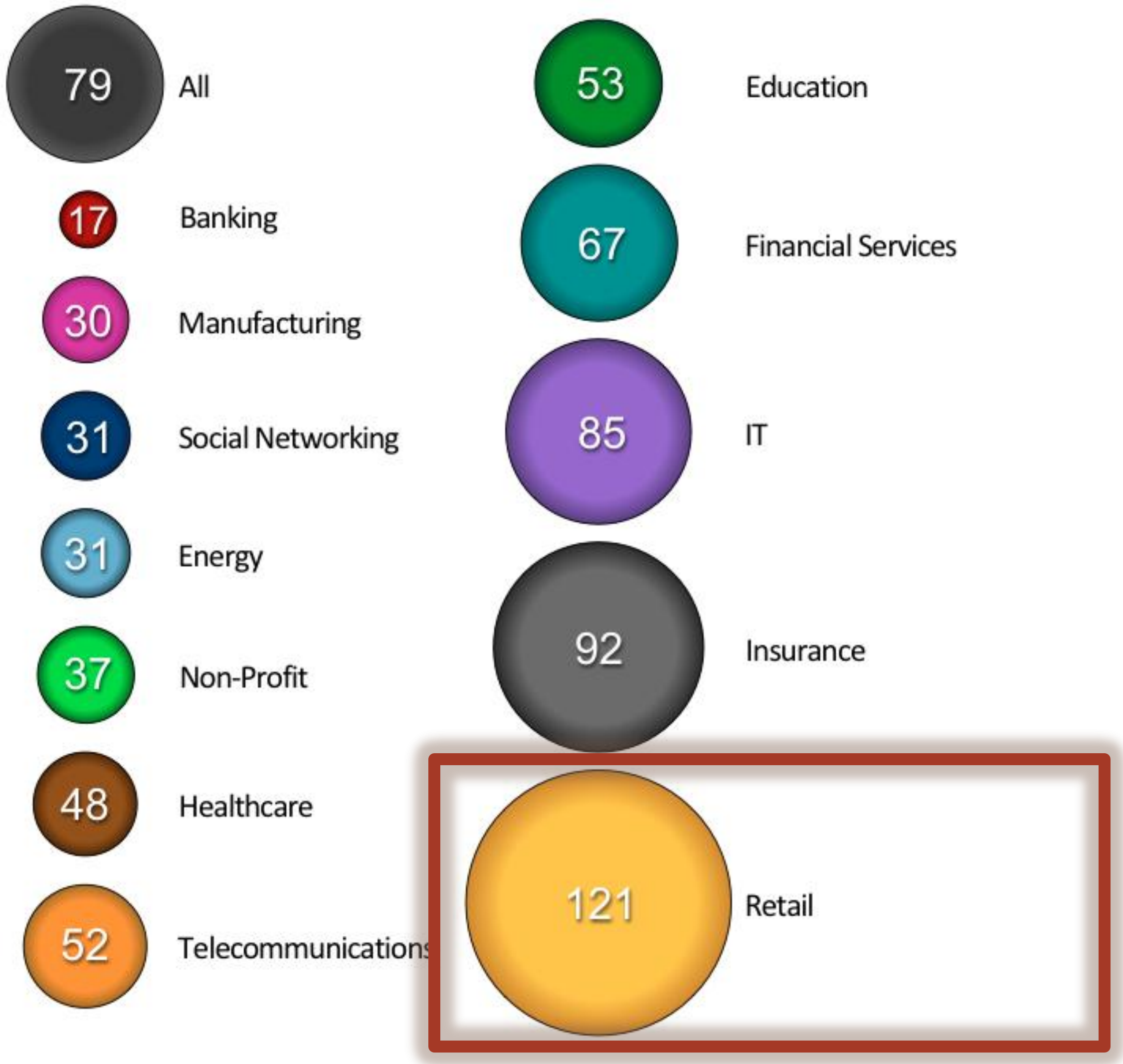
* WhiteHat Security Report Summer 2012

## Likelihood of finding a specific vulnerability

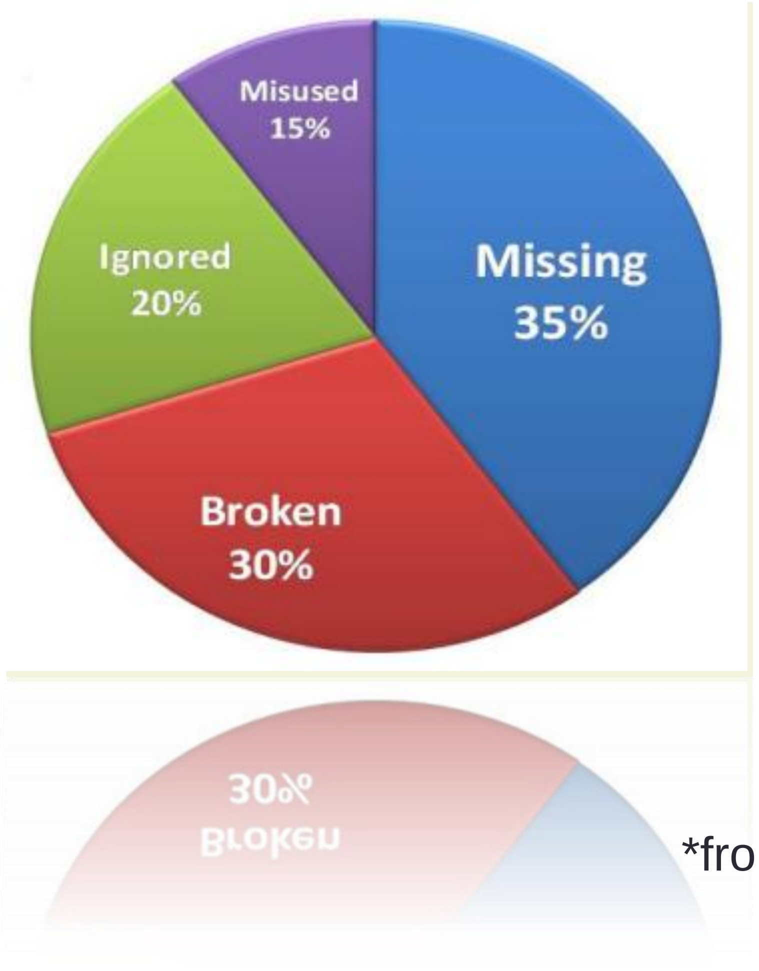


- * WhiteHat Report 2012

## By Industry: Serious Vulnerabilities

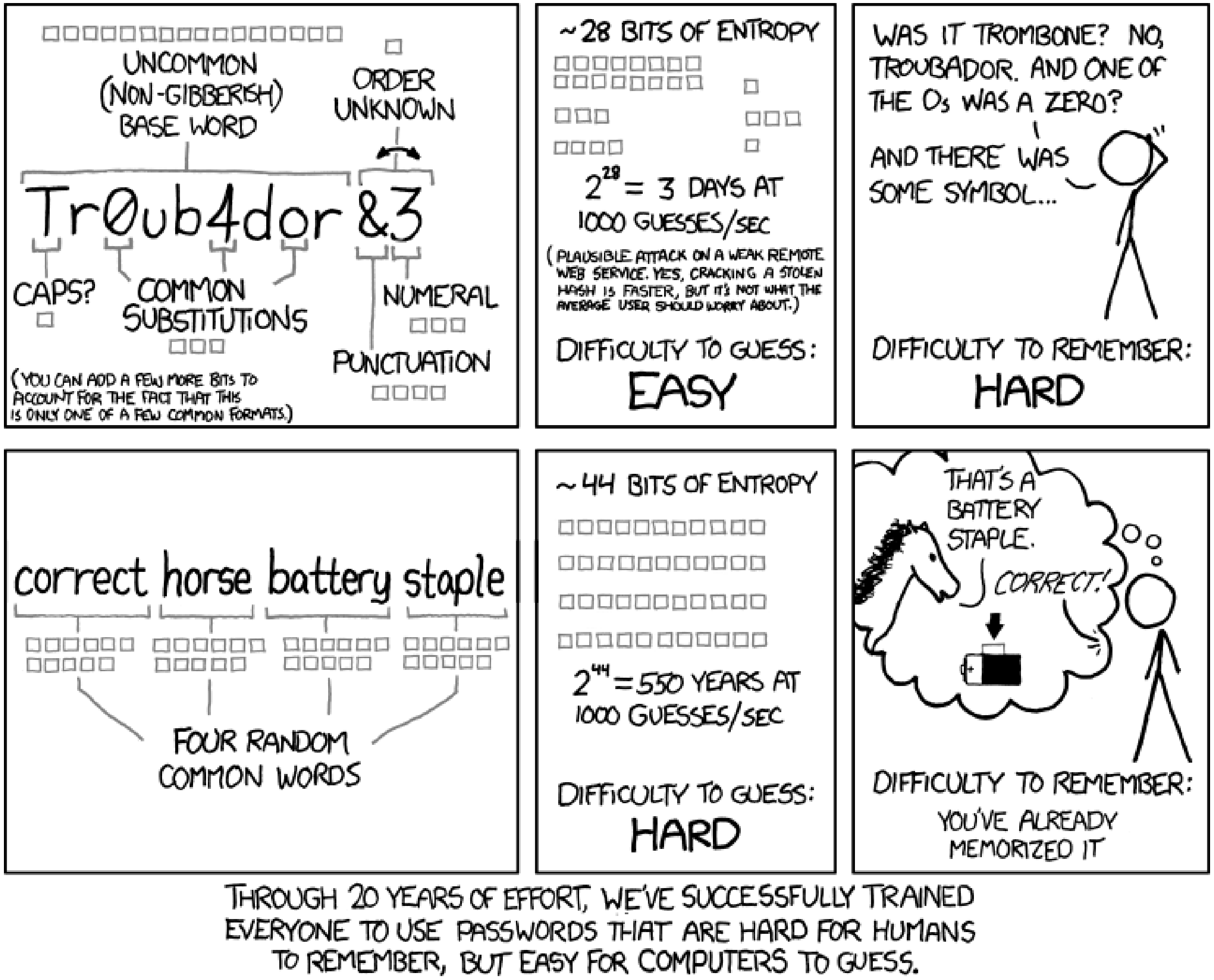


## Classes of Code Vulnerability (the how to code)

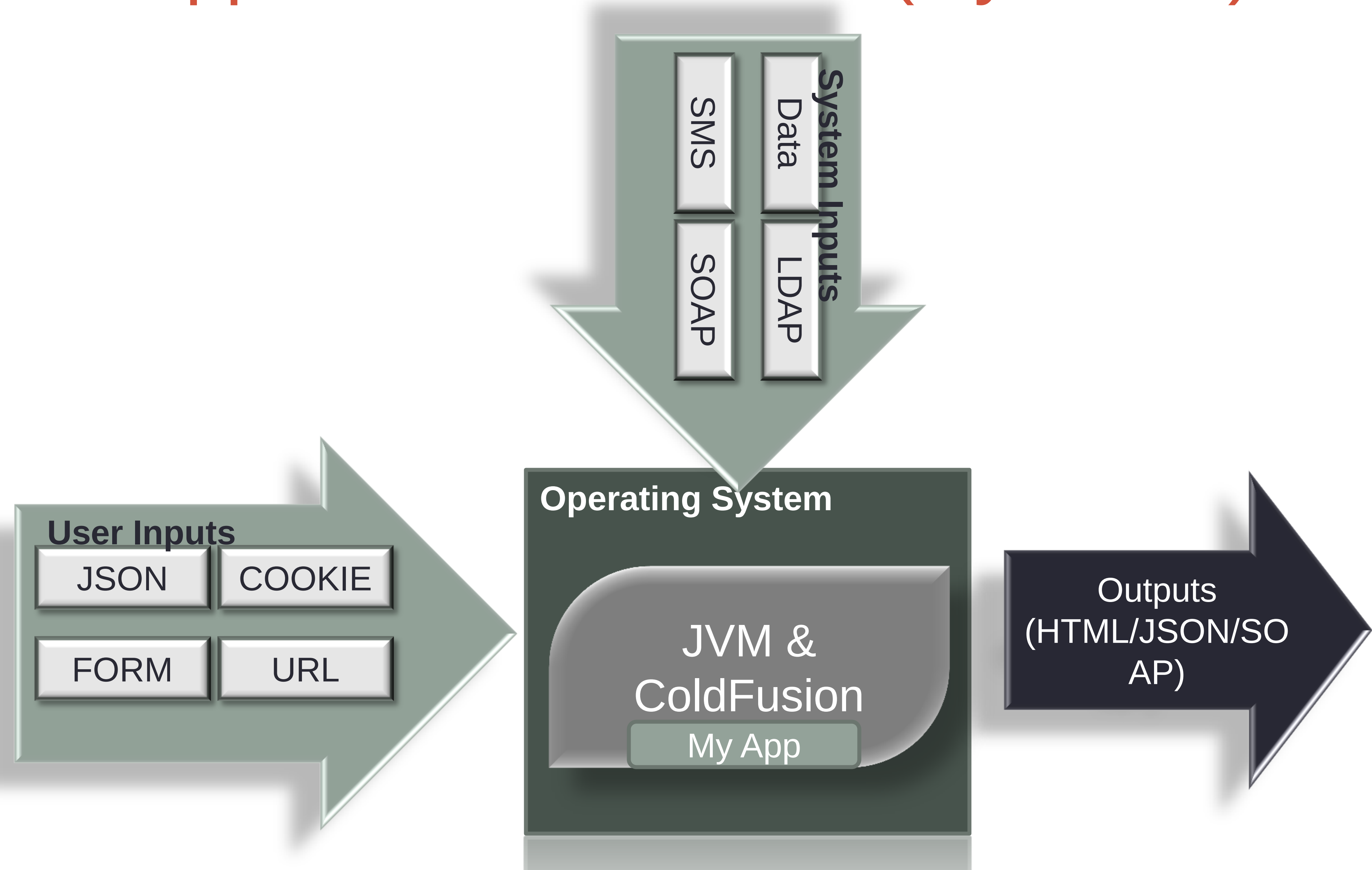


*from Aspect Security

# Example of "Misuse" implementation


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

## No Application is an Island (My Model)



## Current Top 10

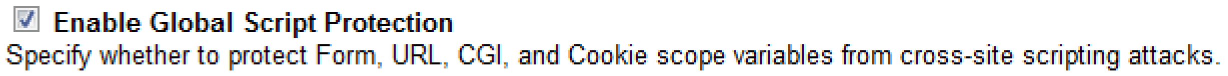
- A1: Injection (SQL) – User Input
- A2: Cross-Site Scripting (XSS) -  User Input
- A3: Broken Authentication and Session Management - Logic
- A4: Insecure Direct Object References – User Input
- A5: Cross-Site Request Forgery (CSRF) – User Input
- A6: Security Misconfiguration - Logic
- A7: Insecure 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

## Mobile Top 10

1. Insecure Data Storage
2. Weak Server Side Controls
3. Insufficient Transport Layer Protection
4. Client Side Injection
5. Poor Authorization and Authentication
6. Improper Session Handling
7. Security Decisions Via Untrusted Inputs
8. Side Channel Data Leakage
9. Broken Cryptography
10. Sensitive Information Disclosure

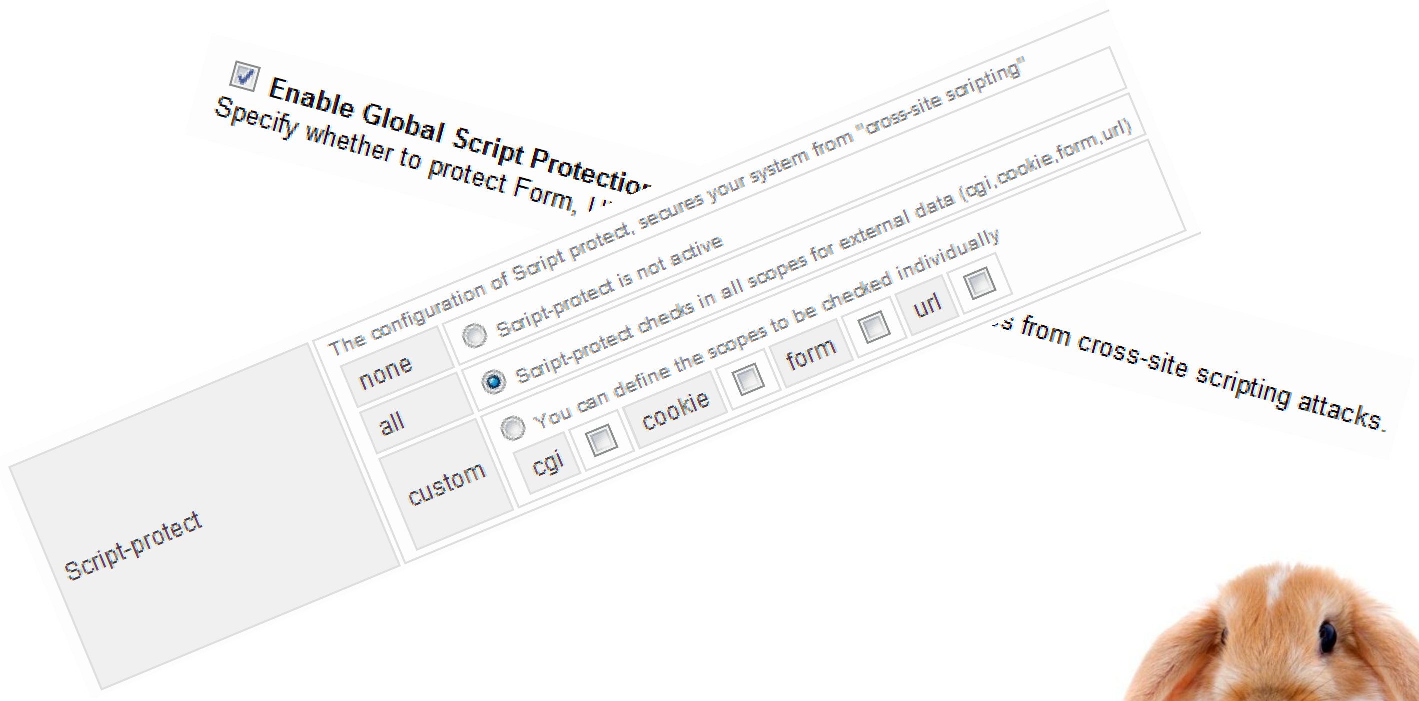
## XSS – A2

- 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.

## Platform got me covered, right?

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



✓ **Enable Global Script Protectio**
Specify whether to protect Form, U

s from cross-site scripting attacks.

| | | |
|---|---|---|
| Script-protect | | The configuration of Script protect, secures your system from "cross-site scripting" |
| | none | ○ Script-protect is not active |
| | all | ● Script-protect checks in all scopes for external data (cgi,cookie,form,url) |
| | custom | ○ You can define the scopes to be checked individually<br>cgi ☐ cookie ☐ form ☐ url ☐ |

**Demo Bunny was here !**

## Quick XSS cheat sheet

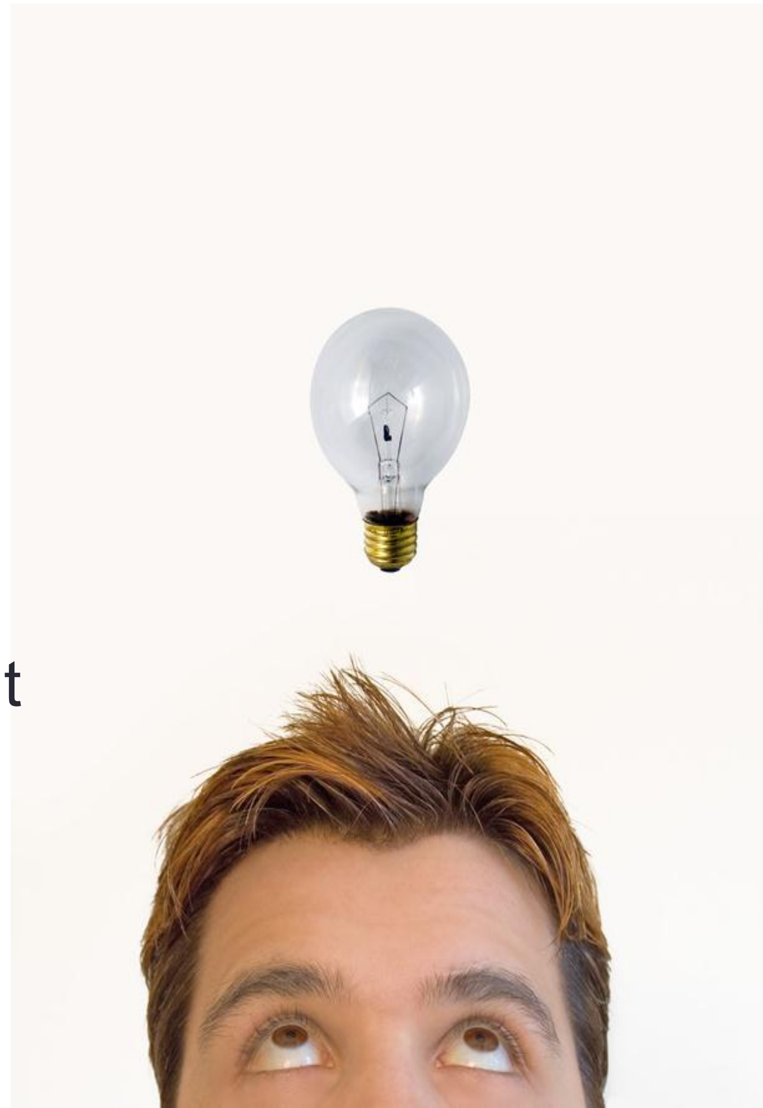
- Enter something like this

";!--"<XSS>=&{()}

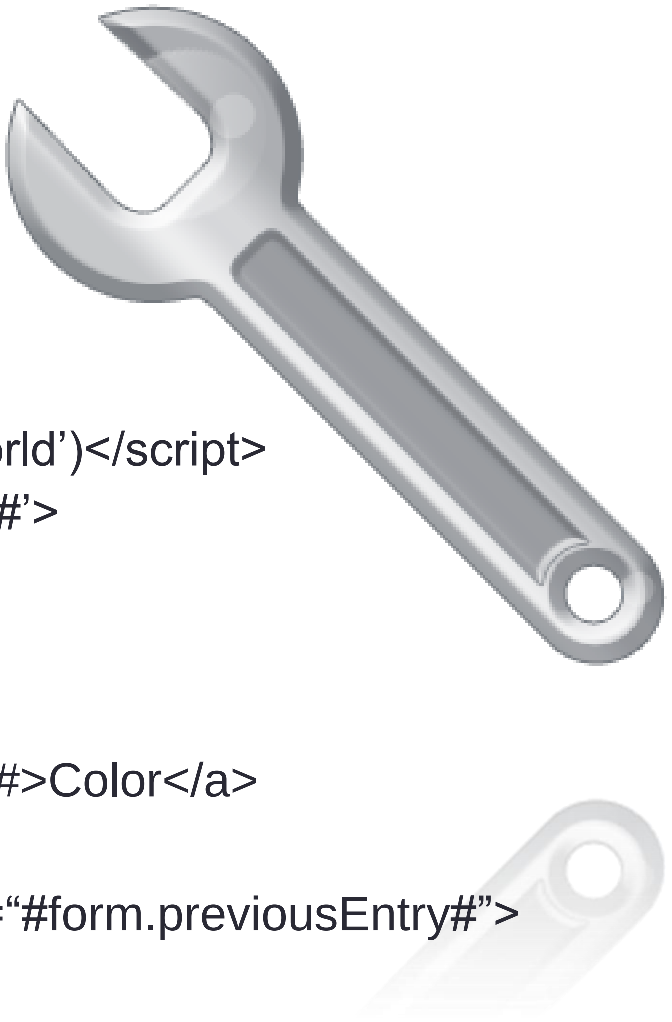
If you see

**<XSS** instead of **&lt;XSS** you are vulnerable!!!!

(don’t rely on all users using the right browser)



## 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
    - <a href="server/my.cfm?**para1**=#form.color#>Color</a>
  - FORM
    - <input type="hidden" id="prevValue" **value**="#form.previousEntry#">

## Easy Hacks : Some Common Trouble

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

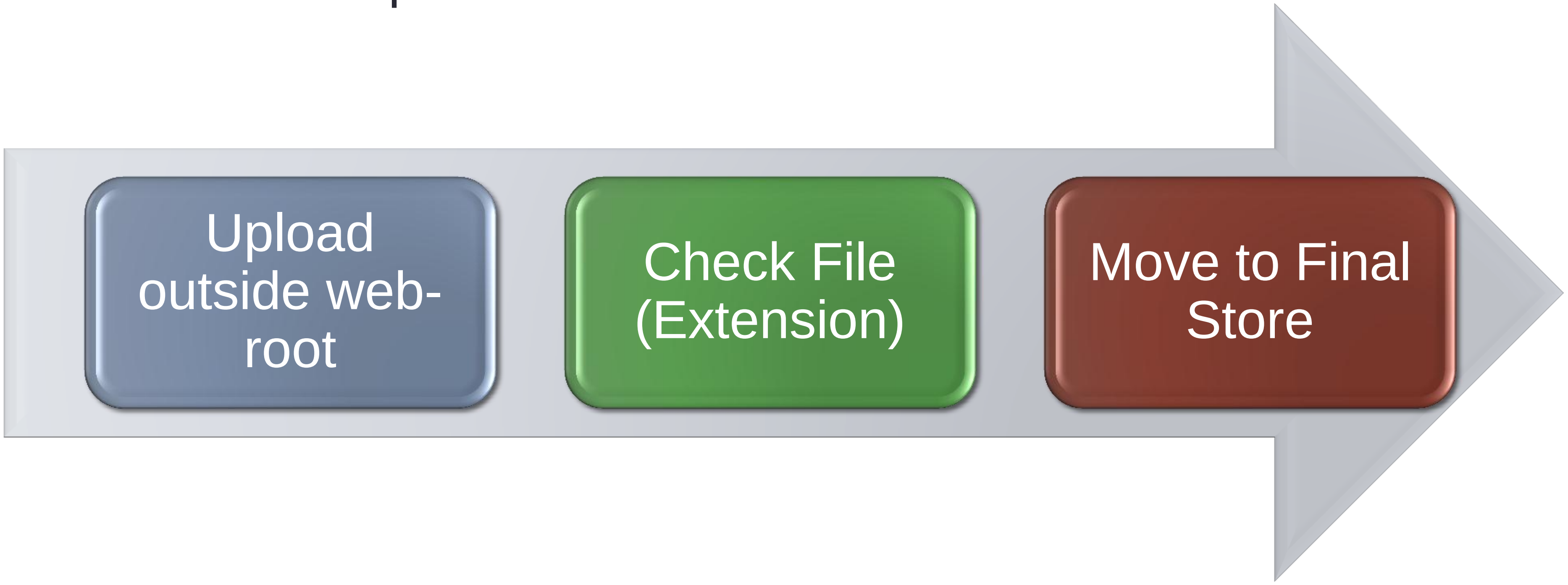
```
<cfhttp  method="post" url="http://localwheels/sec/fileProcess.cfm"
throwonerror="Yes">

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

</cfhttp>
```

## Fun with Files Part 1

- A safer file upload

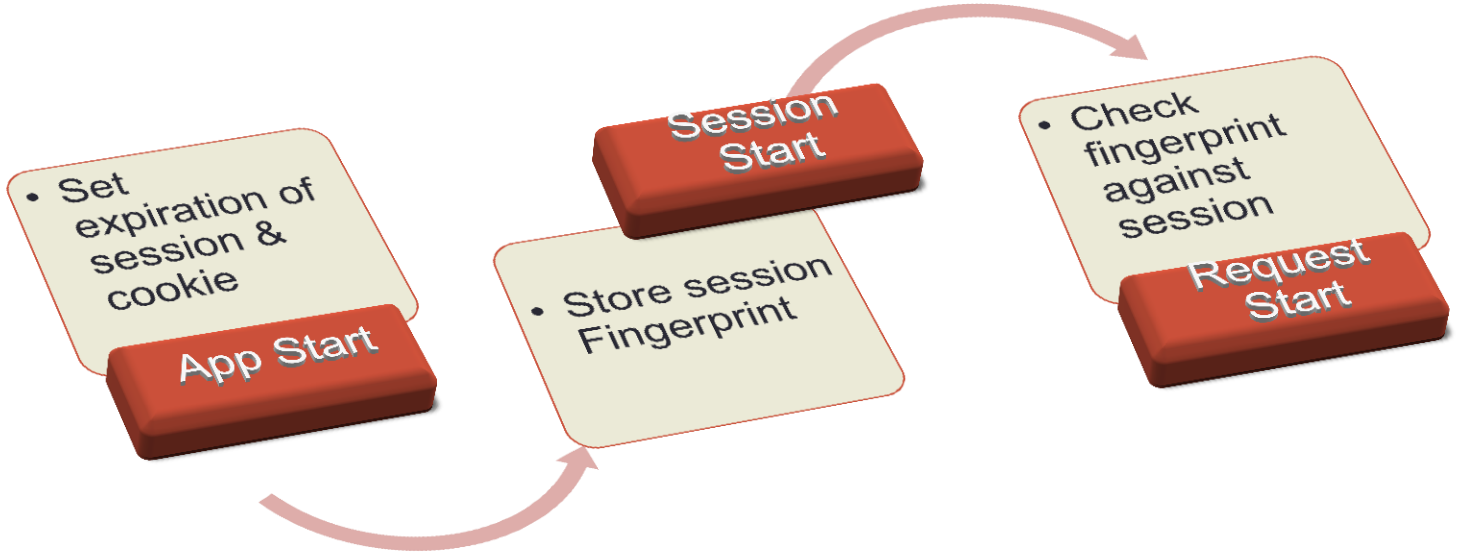


## Session Platform Measures

- 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
  - Option in CF10 Admin
- Consider using SSL when authenticated (prevent sniffers)

## Session Application Measures

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


Set expiration of session & cookie
App Start
Session Start
Store session Fingerprint
Check fingerprint against session
Request Start

- **Better**
  - Create new Session once Authenticated (destroy old): SessionRotate() in CF10
  - Use Fingerprinting to identify the client to which session was issued, e.g. http://panopticlick.eff.org/ / BonCode Connector for IIS

## On IIS Generating Automatic Fingerprint with BonCode Connector

- You can use web connector for fingerprint on Win + IIS:
  - GetHttpRequestData().headers["xajp-clientfingerprint"]
- Change connector setting file, add setting:
  - <EnableClientFingerPrint>True</EnableClientFingerPrint>
- You need connector version 1.0.11 an higher, check so:
  - http://localhost/a.cfm?BonCodeConnectorVersion=true
- For ColdFusion 10 you need to remove Adobe connector and install BonCode connector, see instruction:
  - http://boncode.blogspot.de/2012/06/cf-coldfusion-10-experimenting-with.html

## 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”);
myQuery.addParam(name = "id", value = "99", cfsqltype="cf_sql_integer");
```

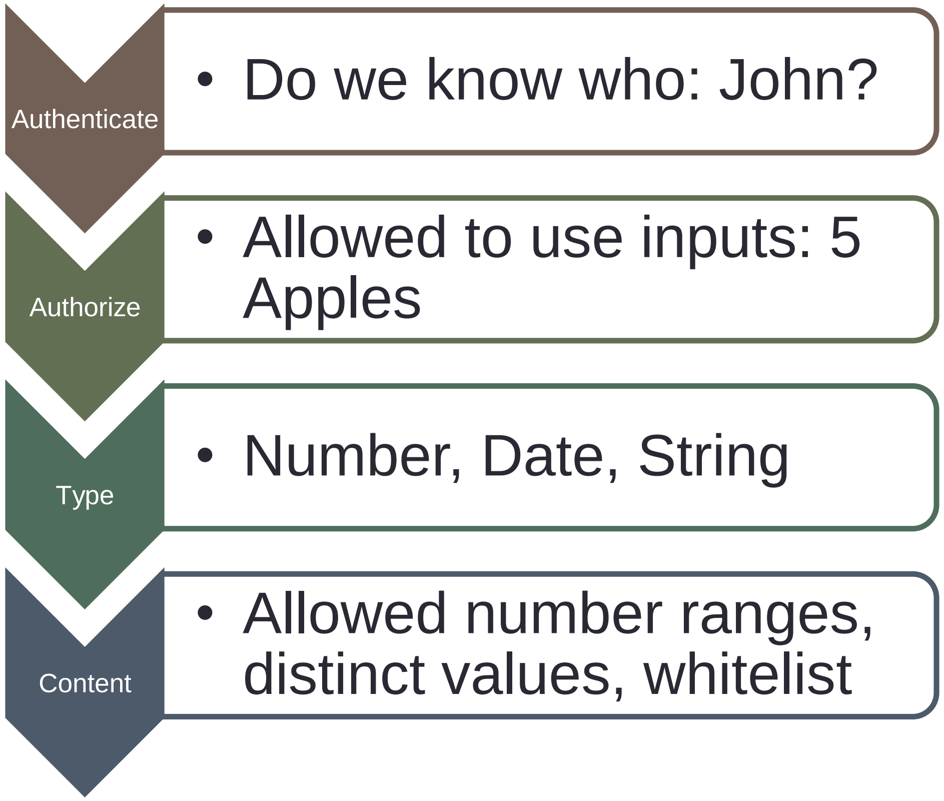
## Hmm.. Did they get me?

Detecting code compromise

Thanks Alex Skinner !

## How to Regain Control / Trust

- How to re-establish trust
  - Establish Validation Strategy
    - Decide **centralized** vs. distributed treatment of inputs
  - Decide Workflow
    - Outbound - Encryption
    - Inbound Validate Workflow



- Do we know who: John?
- Allowed to use inputs: 5 Apples
- Number, Date, String
- Allowed number ranges, distinct values, 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#">
```

## Form Scope and CSRF (A5) Adobe Proposed Solution

**On Submitting Page:**

```
<cfset csrfToken=CSRFGenerateToken() />
<cfform method="post" action="sayHello.cfm">
    <cfinput name="userName" type="text" >
    <cfinput name="token" value="#csrfToken#" type="hidden" >
    <cfinput name="submit"  value="Say Hello!!" type="submit" >
</cfform>
```

**On Processing Page:**

```
<cfset token=form.token>
<cfset validated = CSRFverifyToken(token)>
<cfif validated> ...do regular code </cfif>
```

## Indicating trust within your code

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

## Fun with Files, Part 2

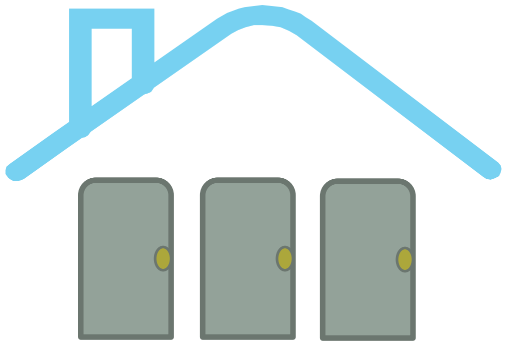
- Let's browse

- PCI and the use of file system as data storage (https://www.pcisecuritystandards.org/)
  - Sensitive Card Holder data **MUST NOT** be stored after authentication.
  - Card holder data must be protected at all times when stored.
- Health Industry Data (HIPAA, HITECH, NHS)

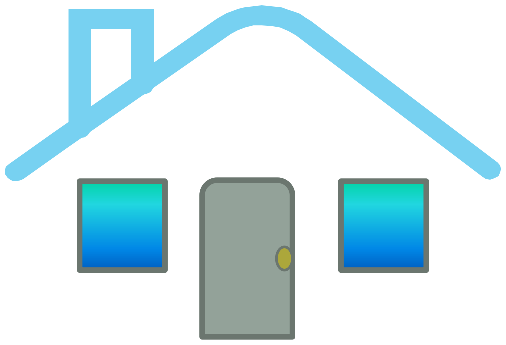
- Establish reasonable Key Management:

## A word on REST services

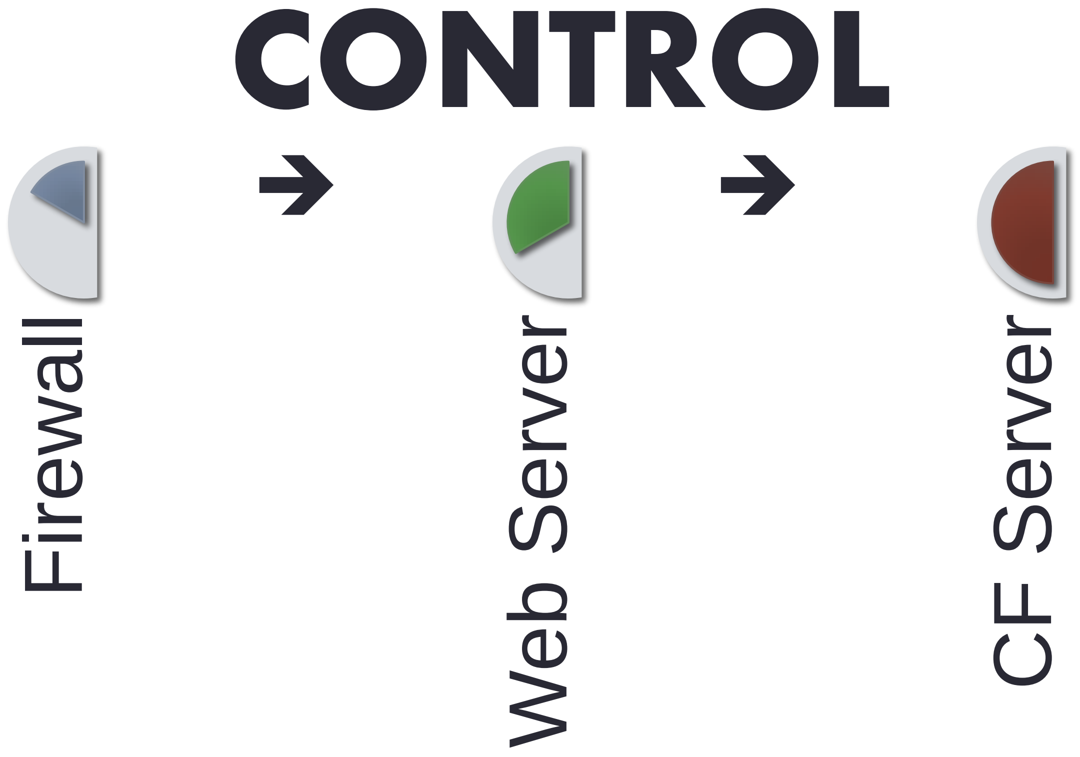
**CONTROL**



**vs.**



## Another word on Administrator Access…



## 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
    - <a href=“my.cfm?par=#URLEncodedFormat(orm.color)#>Col</a>

## Outputting: ESAPI (Enterprise Security API)

- Java library OWASP project
- Installation is not trivial
- CF10 and Railo 4 have it packaged ESAPI for you. Yeah!!
- In HTML Attributes (between double quotes):
  - <a href="#Form.Page#">myLink</a>
  - 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
  - <a href="http://targetsite.com/my.cfm?para1=#form.color#>Color</a>
  - 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**

t.com

MY SECURITY SOFT-WARE KEPT INSISTING THAT I DOWNLOAD CRITICAL UPDATES.
I DIDN'T HAVE TIME TO DO ALL THAT, AND I COULDN'T RISK USING MY COMPUTER WITHOUT CRITICAL UPDATES. IT WAS A NO-WIN SITUATION.
DID YOU ACCOMPLISH ANYTHING THIS WEEK?
WHAT PART OF "NO-WIN" IS CON-FUSING YOU?
www.dilbert.com
scottadams@aol.com
8-9-08 © 2008 Scott Adams, Inc./Dist. by UFS, Inc.

- 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.
- Keep vigilant and up-to-date !

CFAcademy users

Please check blog and download needed examples

## Resources

- OWASP (www.owasp.org)
  - Tons of info
- Adobe (http://www.adobe.com/support/security/)
- SAFECode (http://www.safecode.org)
- My Blog (http://boncode.blogspot.com)
- ESAPI
  - https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
  - https://www.owasp.org/images/7/79/ESAPI_Book.pdf
- Google!

# THANK YOU

Q&A

@BmanClt

http://BonCode.blogspot.com