Writing Secure CFML

Pete Freitag, Foundeo Inc.

Who am I?

- Over 10 years working with ColdFusion
- Owner of Foundeo Inc a ColdFusion consulting & Products company
- Author, Blogger, and Twitterer?

Today's Agenda

- Learn about web specific vulnerabilities
- Learn how to protect your code
- General Security Principals

Locking Down ColdFusion

- Today @ 5:30
- Security for ColdFusion Server Admins

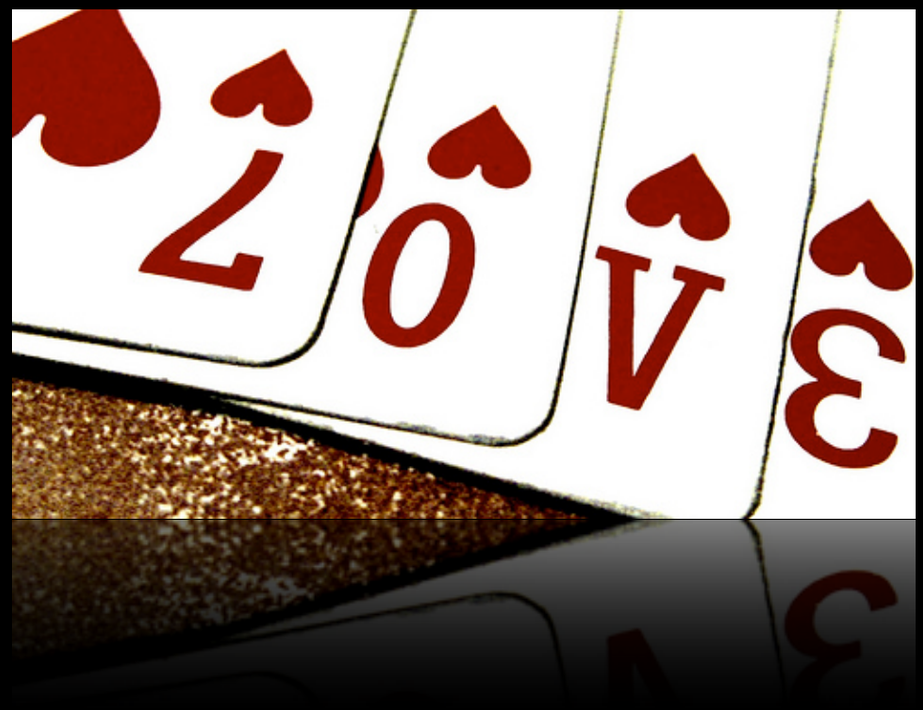
On Security

- Security is hard
- Security is not absolute
- Security requires education
- Anyone who claims they know everything about security knows nothing about security!

How many of you have been hacked?

Hackers Love Web Apps

- Easy to Exploit
- Plentiful



Who is hacking?

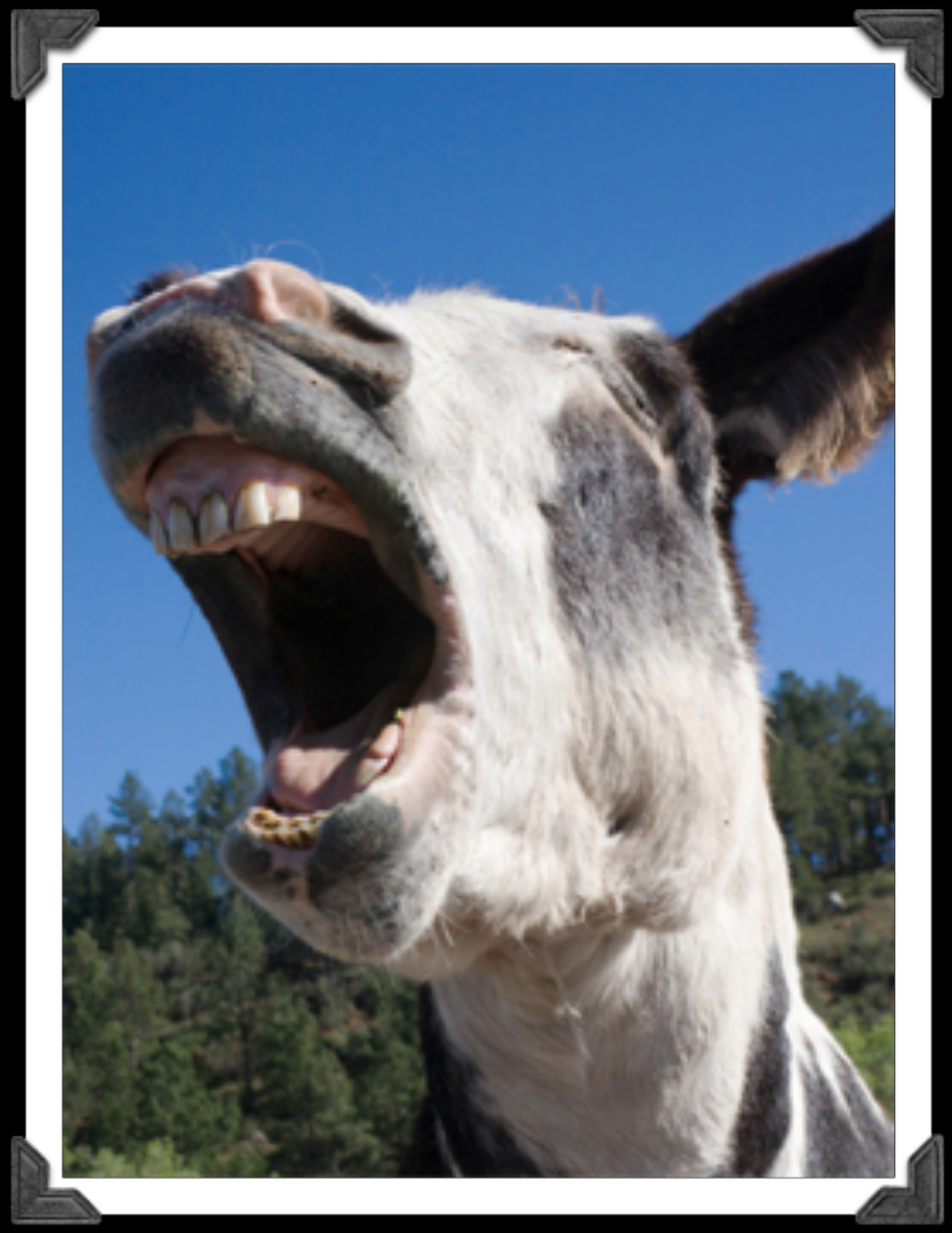
credit: thenovys on flickr

Who is hacking?

photo credit: onlp on flickr

Don't Assume

- You know who is hacking or what their interests are.
- You know everything about security.
- You know how users use your site.



Common Tradeoffs

- Security vs Usability
- Security vs Performance
- Security vs Time / Money

In General:

- Protect the inputs
  - What are the inputs?
- Guard what you output
  - Is the user allowed to see this?
  - Is the output properly encoded?

Path Traversal's

- **The Risk:**
  - Allows attacker to read any file CF has permission to read.

Vulnerable Code

```
<cfinclude template="files/#url.page#">
```

Exploiting It

- Instead of page.cfm?path=about.cfm
- Attacker runs:
  - page.cfm?path=../../any/file/on/the/server
- Demo

Fixing Path Traversals

- **Applies to:** cfinclude, cfmodule, cffile, File Functions, any code that deals with file paths
- **To fix:**
  - Avoid using unsafe variables in code that deals with files.
  - If you must use a user supplied variable validate it.

CRLF Injection

- **The Risk:**
  - Attacker can create their own HTTP Response or Email MIME headers.
    - Redirect users to hacker sites, XSS, Phishing, Cache Poisoning
    - Alter email messages to create spam

What's a CRLF?

- Carriage Return Line Feed (CRLF)
  - ASCII Characters: 13 & 10
  - \r\n
  - %0D%0A (URL Encoded)
  - Line Delimiter used by many protocols

Example HTTP Response

```
HTTP/1.1 200 OK<CRLF>
Date: Thu, 29 Jul 2010 18:55:04 GMT<CRLF>
Server: Apache<CRLF>
Content-Type: text/html<CRLF>
<CRLF>
<html>
...
```

Vulnerable Code

```
<cfheader name="Content-Type" value="text/#url.format#">
```

Exploiting It

- Instead of view.cfm?format=html
- Attacker Runs:
  - view.cfm?format=format=html%0D%0AHeader:+Value
- Demo

Vulnerable Code

```
<cfmail to="you@example.com"
  subject="#form.subject#"
  from="them@example.com">

Email Body...

</cfmail>
```

Exploiting It

- Attacker manipulates variable used in the subject field to write new MIME headers and message body.
- Demo

Fixing CRLF Injection

- Applies to: CFHeader, CFContent (type attribute), CFMail, CFMailParam, CFMailPart. Any code that writes protocol headers.
- To fix, strip CRLF from these variables

```
<cfset form.subject = ReReplace(form.subject, "[\r\n]", "", "ALL")>
```

Insecure File Upload

- **Very Common, Very Dangerous**
- **The Risk:**
  - An attacker uploads and executes a file on your server.

Vulnerable Code

```
<cffile action="upload"
        filefield="photo"
        accept="image/gif,image/jpeg,image/png"
        destination="#ExpandPath("./photos/")#">
```

Wait a Sec...

- Doesn't the **accept** attribute limit the types of files that can be uploaded?

NOPE

- The mime type used by the **accept** attribute is supplied by the client (from the web browser).

Exploiting it

```
<cfhttp url="http://example.com/upload.cfm" method="post">
    <cfhttpparam file="#ExpandPath("code.cfm")#"
                 mimetype="image/gif"
                 type="file"
                 name="photo">
</cfhttp>
```

What did we learn?

Still Vulnerable?

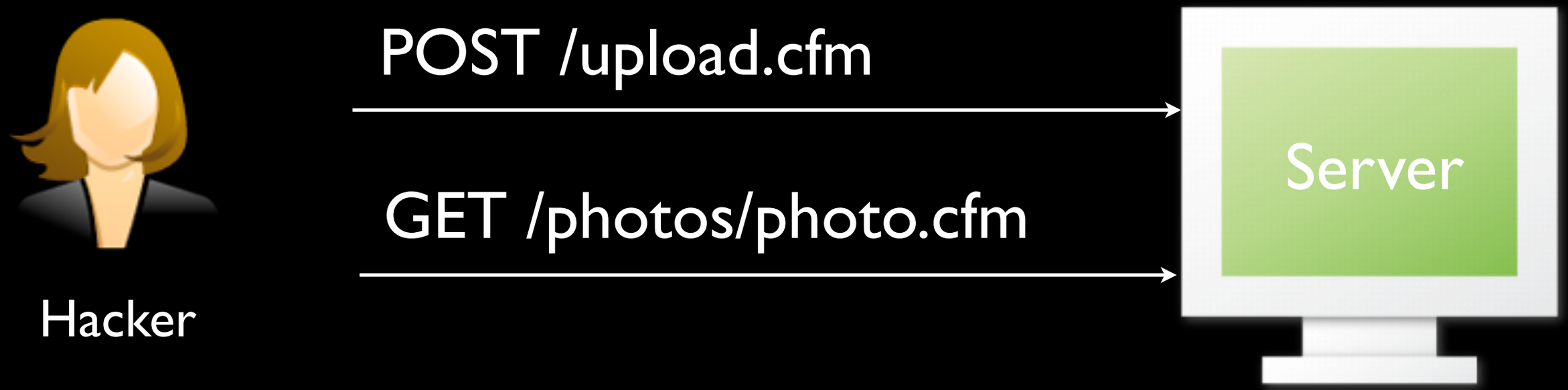
```
<cffile action="upload"
            filefield="photo"
            accept="image/gif,image/jpeg,image/png"
            destination="#ExpandPath("./photos/")#">

<cfif NOT ListFindNoCase("gif,jpg,png", cffile.ServerFileExt)>
    <cffile action="delete"
      file="#cffile.ServerDirectory#/#cffile.ServerFile#">
<cfelse>
    <p>File Was Uploaded.</p>
</cfif>
```

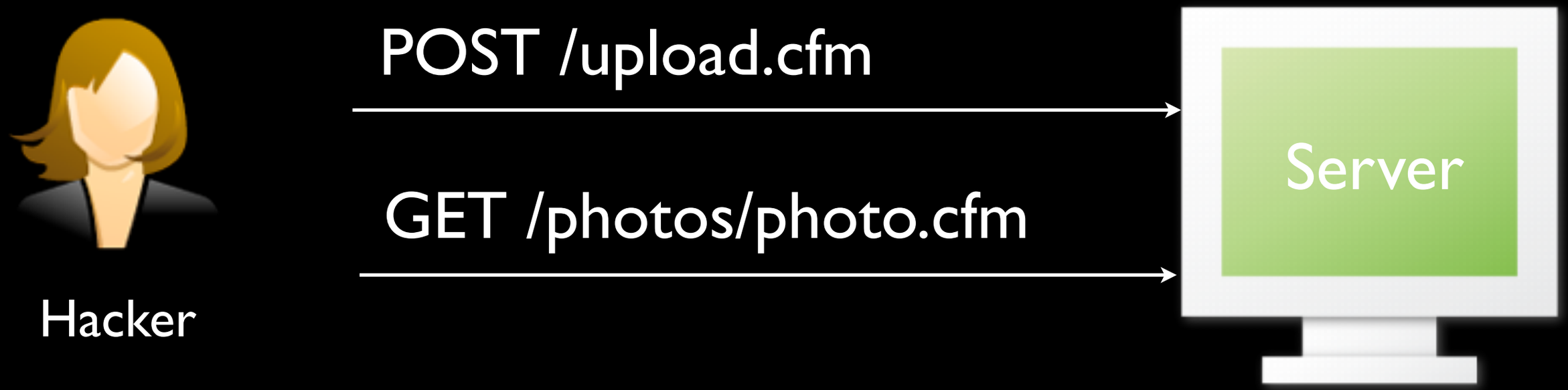
Yes, Still Vulnerable

- Notice that the destination of cffile was under the web root.
  - The file was uploaded to the web root and may be executed before it is deleted milliseconds later.

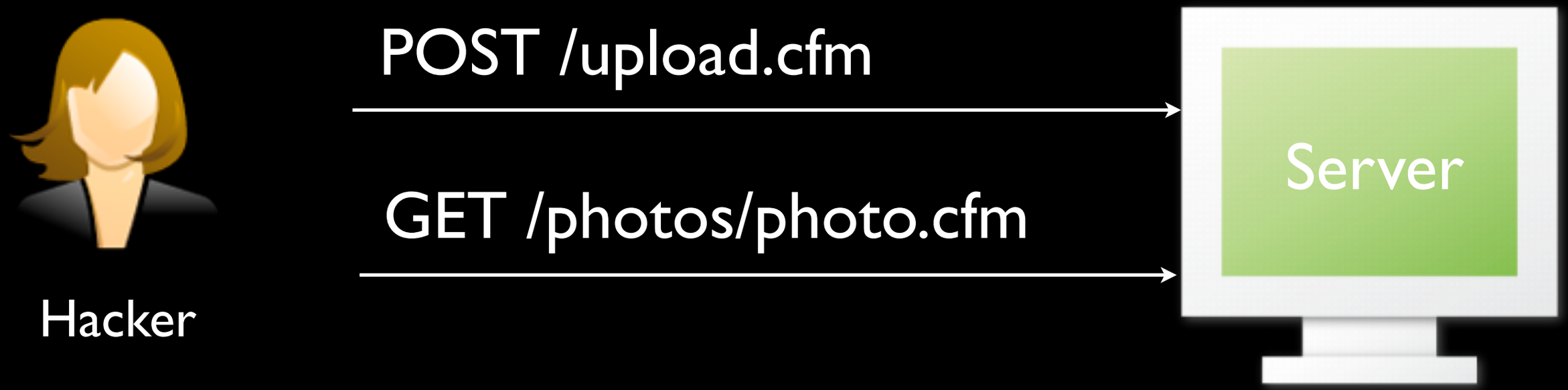
Hacker


POST /upload.cfm
Server
Hacker


POST /upload.cfm
GET /photos/photo.cfm
Server
Hacker



Hacker uses a load tool to make repeated concurrent requests.



Hacker uses a load tool to make
repeated concurrent requests.

After a while, the attacker will get lucky

Fixing File Uploads

- Use but don't rely on the accept attribute.
- Always validate file extension
- Never upload under the web root.
    - Only copy files there once validated.

Whitelist vs Blacklist

- Prefer whitelists over blacklists
  - eg: allow jpg, png, gif, pdf
- Black lists are very hard to maintain
  - eg: block cfm,cfc,jsp
  - Oops you missed: cfml, cfr, jws
  - Admin just installed php...

File Upload Tips

- Validate File Content if possible
  - IsImageFile(*path*)
  - IsPDFFile(*path*)
  - jHOVE - Java API for additional types

File Upload Tips

- Deny execution for upload destination directory.
  - On Web Server
    - In ColdFusion (with Sandbox Security)
- Serve files from a static content server
  - Build your own
  - Amazon S3, etc.

File Upload Tips

- Set **mode** attribute of cffile on unix
  - eg: 640 = rw-r-----
    - 7 = read, write, execute (rwx)
    - 6 = rw
    - 4 = r
    - 0= no privledges
    - owner group other

SQL Injection

- **The Risk:**
  - Attacker can run arbitrary SQL against your database.
  - Typically execute system commands on the database server.

Vulnerable Code

```
<cfquery datasource="#application.ds#" name="news">
    SELECT id, title, story
    FROM news
    WHERE id = #url.id#
</cfquery>
```

Exploiting It

- Instead of news.cfm?id=1
- Attacker Runs:
  - news.cfm?id=1;DROP+Users
  - news.cfm?id=1+UNION+SELECT+...
- Demo

Fixing SQL Injection

- Use the **<cfqueryparam>** tag for variables in a query wherever possible.

```
<cfquery datasource="#application.ds#" name="news">
    SELECT id, title, story
    FROM news
    WHERE id = <cfqueryparam value="#url.id#"
                cfsqltype="cf_sql_integer">
</cfquery>
```

How CFQueryParam Works

This:

```
<cfquery datasource="#application.ds#" name="news">
SELECT id, title, story FROM news
WHERE id = <cfqueryparam value="#url.id#" cfsqltype="cf_sql_integer">
AND category = <cfqueryparam value="#url.cat#" cfsqltype="cf_sql_integer">
</cfquery>
```

Is sent to the DB as:

```
SELECT id, title, story FROM news
WHERE id = ?
AND category = ?
```

data[1] = 123 ← ID

data[2] = 913 ← CAT

CFQUERYPARAM

- Works in WHERE clauses, INSERT values, and UPDATE values.
  - Some places it does not work SELECT TOP *n*
- Can be used with lists in an IN statement using list=true

When you Can't Use CFQueryParam

- Be sure you have validated the variable as a simple type.
- EG: SELECT TOP #Int(Val(url.top))#
  - Val() returns 0 when it can't convert a string to a number.
  - Int() converts a decimal value to an integer. Throws exception if not numeric.

Cross Site Scripting

- The Risk:
  - Identity Theft for Web Sites
  - Phishing
  - Session Hijacking
  - More

Cross Site Scripting

- Often abbreviated as XSS
- Occasionally abbreviated as CSS
  - But for obvious reasons that's not a smart name.

Vulnerable Code

```
<cfoutput>
  Hello #url.name#
</cfoutput>
```

Exploiting XSS

- Instead of hello.cfm?name=pete
- Attacker runs:
  - hello.cfm?name=<script>alert(‘pete’)</script>
- Demo’s

Fixing XSS

- One Solution: Strip all harmful characters
  - < > ' " ( ) ; #
- Not always a realistic solution.

Fixing XSS

- Encode variables to escape special characters. (eg < becomes <)
- The best way to do this depends on where the variable is output, in a tag attribute, inside JavaScript, etc.
- 5 Output Contexts to be aware of

Output Context's

Context	Example
HTML	`<p>Hello #url.name#</p>`
HTML Attribute	`<div id="#url.name#" />`
JavaScript	`<a onclick="hi(#url.name#)" />` `<script>#var#</script>`
CSS	`<div style="font-family: #url.name#" />` `<style>#var#</style>`
URL	`<a href="hi.cfm?name=#url.name#" />`

HTML Context

- XMLFormat() or HTMLEditFormat()
  - XMLFormat Escapes < > ' "
  - HTMLEditFormat Escapes <> "

Using ESAPI

- OWASP Enterprise Security API
- Java API that has encoder methods for each context.
- http://code.google.com/p/owasp-esapi-java/

Using ESAPI

Context	Method
HTML	`esapi.encodeForHTML(variable)`
HTML Attribute	`esapi.encodeForHTMLAttribute(variable)`
JavaScript	`esapi.encodeForJavaScript(variable)`
CSS	`esapi.encodeForCSS(variable)`
URL	`esapi.encodeForURL(variable)`

```
<cfset esapi = CreateObject("java", "org.owasp.esapi.ESAPI").encoder()>
```

What if my user must submit HTML?

- You need to make sure the html is valid
- Does not contain any script, iframe, object, style, etc tags.
- HTML attributes do not have harmful JS event handlers or exploit CSS hacks in the style attribute.
- Very difficult to write something like this

Accepting HTML

- AntiSamy for Java
  - Create a policy defining allowed HTML
    - ESAPI has integrated AntiSamy in it's Validator implemention
      - ESAPI.validator().isValidSafeHTML()
    - Ask me who's "Samy" later.

Session Hijacking

- **The Risk:**
  - Attacker learns the value of a session id (cfid & cftoken) that is currently authenticated.

Session Hijacking

- How does the attacker learn the session id?
  - XSS
  - Guessing
  - Sniffing
  - Sharing a URL

Mitigating Session Hijacking

- Use addtoken=false in cflocation tag.
- Use UUID for CFToken
- Use HTTPOnly session cookies
- Use Secure cookies for SSL
- Use SSL
- Cookie Path Attribute

Remember Cookies are Inputs too

- Does your code trust variables like cookie.user_id?

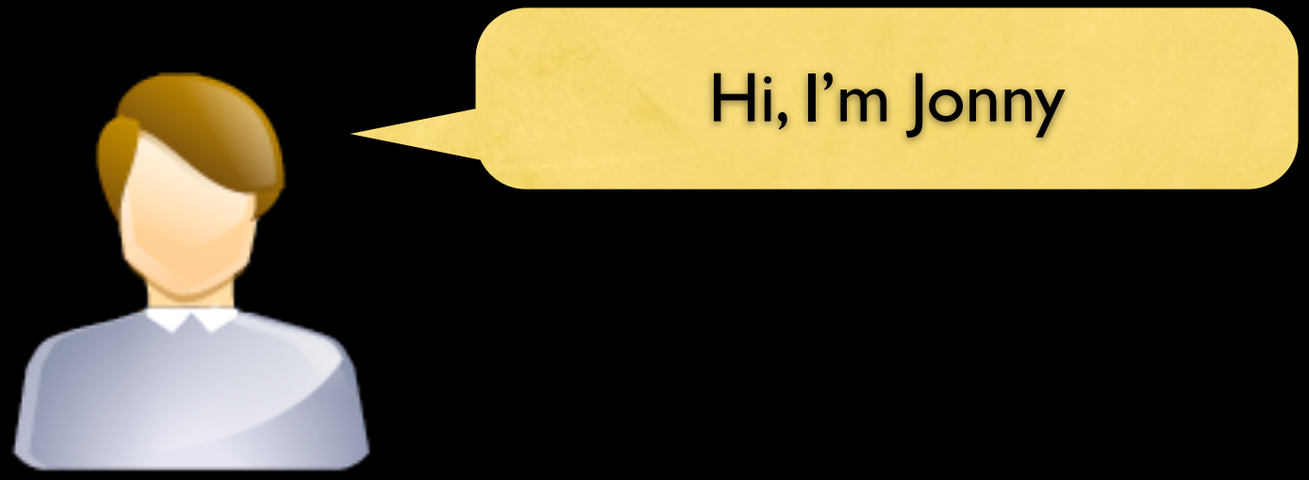
Cross Site Request Forgery

- **The Risk:**
  - Attackers can perform an authenticated action on the victim's behalf.

How it works

- Victim authenticates
- Attacker gets the victim to make a http request
  - Sends email with

CSRF Example



CSRF Example

Hi, I'm Jonny

Jonny is currently logged into auction site: hack-bay.com

CSRF Example

Hi, I'm Jonny

Jonny is currently logged into auction site: hack-bay.com

CSRF Example

Jane - is this really Eric Clapton's Strat?

CSRF Example

Jane - is this really Eric Clapton's Strat?

Hi Jonny, Yes, check out this photo:
http://bit.ly/1337

CSRF Example

Jane - is this really Eric Clapton's Strat?

Hi Jonny, Yes, check out this photo:
http://bit.ly/1337

Sweeeet!!

CSRF Example

CSRF Example

```
<img src="http://hacker.example.com/clapton.jpg" />

<img src="http://hack-bay.com/bid.cfm?item=123&amount=80000" height="1" width="1" />
```

CSRF Example

- Jonny just bid $80,000 on the guitar, by clicking on the link from Jane.

Fixing CSRF

- Use method = POST
  - CSRF still possible with POST, but more difficult.

Fixing CSRF

- Reject Foreign Referrers
  - Doesn't fix XSS + CSRF
  - Referrer might not be present / spoofed.

Fixing CSRF

- Random Token
  - Include a random token as a hidden field.
  - Store the token in a session variable
  - Compare the hidden form field with session variable on form action page.

Fixing CSRF

- Require Password or Captcha
  - Not very usable, but sometimes essential.

Secure Advice

- Validate Everything!
  - Become a Regexpert!
    - All inputs (form, url, cgi, cookie, any untrusted external data)
- Be Paranoid
- Keep Learning

FuseGuard

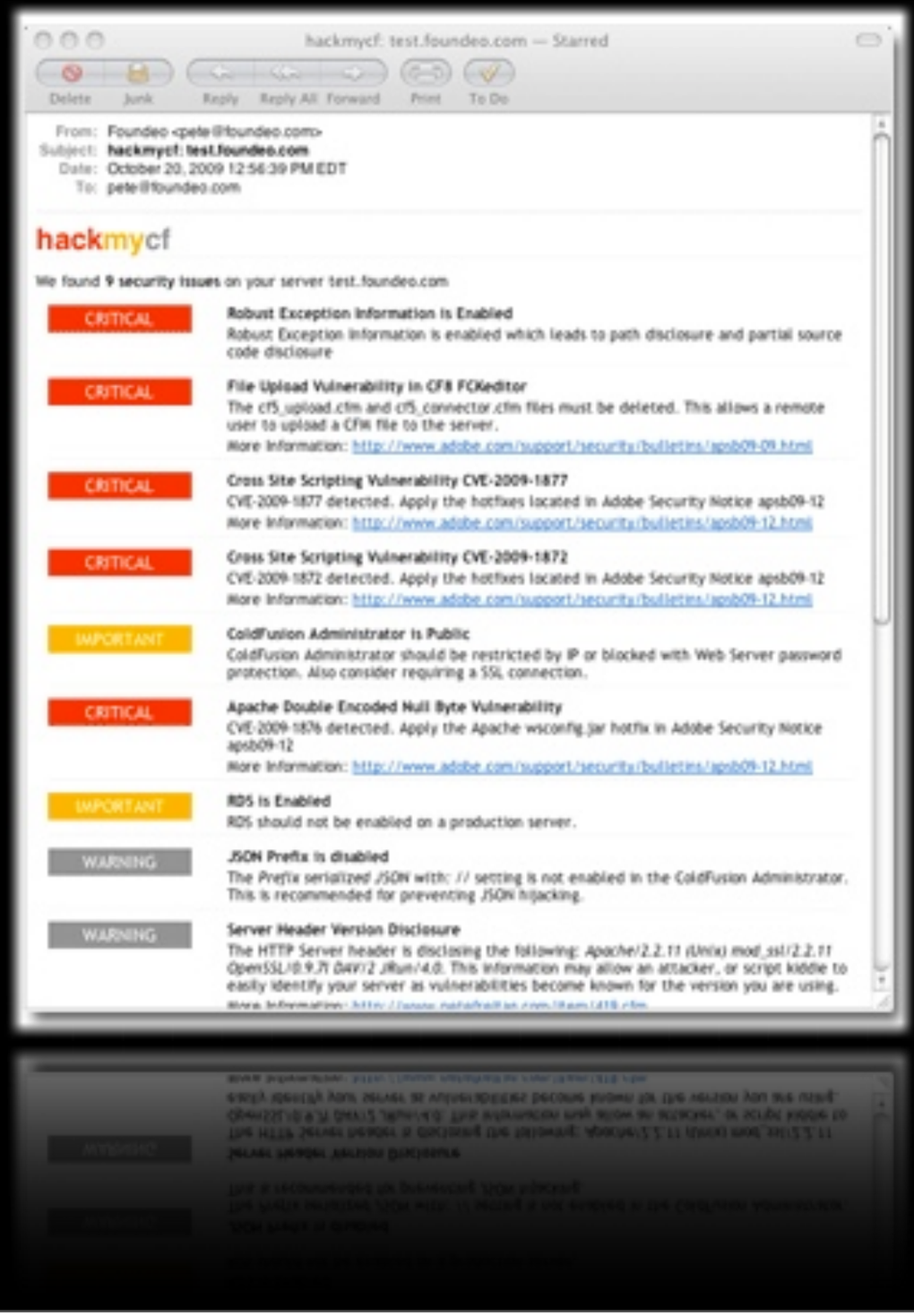
Web Application Firewall for ColdFusion

http://foundeo.com/security/

foundeo inc.

HackMyCF.com

- Free ColdFusion Security Scan
  - Enter your email & domain to get a free report
- Paid Subscription:
  - Automated Scans
  - Additional Features
- http://hackmycf.com/



Thanks. Questions?

pete@foundeo.com

www.hackmycf.com

www.foundeo.com