Security is a big problem

In the rush to clean up the Debian-OpenSSL fiasco, a number of other major security holes have been uncovered:

<table>
<thead>
<tr>
<th>Affected System</th>
<th>Security Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fedora Core</td>
<td>Vulnerable to certain decoder rings</td>
</tr>
<tr>
<td>Xandros (see PC)</td>
<td>Gives root access if asked in stern voice</td>
</tr>
<tr>
<td>Gentoo</td>
<td>Vulnerable to flattery</td>
</tr>
<tr>
<td>OLPC OS</td>
<td>Vulnerable to Jeff Goldblum's powerbook</td>
</tr>
<tr>
<td>Slackware</td>
<td>Gives root access if user says elvish word for &quot;friend&quot;</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>Turns out distro is actually just Windows Vista with a few custom themes</td>
</tr>
</tbody>
</table>

```c
int getRandomeNumber()
{
    return 4;  // chosen by fair dice roll.
    // guaranteed to be random.
}
```
Security and Privacy

• Many web sites are developed initially without taking these into account
  • What are the consequences?
  • We hear about security/privacy issues daily
  • Most exploits use well-understood techniques
  • Most exploits could be avoided with a little care

• Need to think about security and privacy
  • From the start
  • Design the web site with this in mind
  • Design the code with this in mind
Homework

• What news stories did you find
  • Did they tell you what went wrong?
Security & Privacy Problems

• Security Week
  • securityweek.com
• SC Magazine
  • scmagazine.com
• CNET on Security
  • www.cnet.com/topics/security
• ThreatPost on Web Security
  • https://threatpost.com/category/web-security
Security is Fun

• Creative thinking – outside the box
• Think of all the ways of breaking software
• Think of all the ways of preventing such breakage

"Information security is a major priority at this company. We've done a lot of stupid things we'd like to keep secret."
Obvious Problems

• Not considering security in your application
  • Not requiring user authentication
  • Allowing weak authentication
  • Not encrypting sensitive communication
  • Sessions that don’t time out
  • Session ids that are guessable
• Not putting in the resources needed
  • Yahoo
Obvious Problems

• Having a vulnerable server
  • Not being at the latest patch level
• Having bugs in the software
  • Disclosing information inadvertently
  • Exposing SQL and other errors
• Sending private information publically
Obvious Problems

• Applications sharing a common back end
  • PHP, Tomcat, ... have common code
  • Applications can interfere with each other
Non-Obvious Problems

• Assume we have secured our server, validated our code, used best practices for data encoding, etc.
  • Are there still things that can go wrong?
Security Issues

• Application security
  • SQL injection attacks
  • Cross-site scripting attacks
  • Cross-site request forgery
  • Code insertion attacks
  • File name attacks
  • Buffer overflow attacks
  • Timing attacks

• Keeping information secret
  • Passwords, Credit Cards, …
  • From whom and when?

• Principle of minimum access
Security and Web Applications

• “All we can teach people about server-side programming in a few hours is how to create security holes, even if we use modern frameworks.”

• **Think about security throughout the design**
Question

A SQL injection attack involves

A. A malicious user inputting text that is used in a prepared SQL query to do malicious things.
B. A malicious user inputting text that is concatenated to form an unexpected SQL operation.
C. A malicious user adding JavaScript code to the web page to create new SQL operations in the back end.
D. A malicious user generating a XMLHttpRequest that cause the back end to add information to the database.
E. A malicious user adding code to the web server to execute their own SQL commands.
SQL Injection Attacks

• Let user = user_id from the web page
  • Authentication or just for information
• Code
  
  var q = "SELECT firstname 
          FROM users WHERE name = " + user + ";"
  var rslt = db.query(q);
  var msg = "Hello " + rslt[0].username;
SQL Injection Attacks

• Input: \texttt{user\_id = spr}
  • \texttt{SELECT firstname FROM users WHERE name = 'spr'}
• Input: \texttt{user\_id = x' or '1' = '1}
  • \texttt{SELECT firstname}
  • \texttt{FROM users WHERE name = 'x' or '1' = '1'}
SQL Injection Attacks

• What if the user passes in
  
  `x'; DROP TABLE 'users'; SELECT * from passwords WHERE '1' = '1`
  
  `SELECT firstname FROM users where User = 'x';`
  
  `DROP TABLE 'users';`
  
  `SELECT * FROM passwords WHERE '1' = '1'`

• Pass in `x'/`*

• Pass in a query that takes a long time
SQL Injection Attacks

• The attacker needs to know
  • What the queries look like (the code)
  • The structure of the underlying database

• Can you determine this?
  • Yes – might take some time, but easy to do
  • Recipes are on the web
  • Tools are available
SQL Injection Attacks

• Can do different malicious things depending on query and underlying database system

• Used to
  • email passwords to a different user
  • Get field and table names for the database
  • Find user names
  • Password guessing
  • Adding new users (super users)
Avoiding SQL Attacks

• Validate all requests before sending to server
  • Understand what data should look like
    • User ids, product ids, email, ... have a given form
  • Use JavaScript in the client to do the validation
• Is this sufficient?
Avoiding SQL Attacks

• Validate the strings in the server
  • Most back end languages have a function for this
  • Check there are no funny characters
    • E.g. check that there are no quotes in the string
    • Is this sufficient?
      • Value is a number
      • Name is O’Reilly
      • Unicode
  • Check each string is in a standard form
    • Is this sufficient?
Avoiding SQL Attacks

• **Use prepared statements**
  - SELECT firstname FROM USERS WHERE name = $;
    - Pass in array of values separately
  - The substitution is handled by the database
    - Done correctly even with quotes, odd characters, etc.

• **This is basically sufficient**
  - Done automatically in RUBY, DJANGO
  - This is what you should use
Avoiding SQL Attacks

• Make sure your database is relatively secure
  • Grant permissions to tables only as needed
  • Have separate database users for different roles
  • Limit access to the web application itself
• Principle of least access
Code Insertion Attacks

• SQL isn’t the only place where web apps run arbitrary code
  • Php, JavaScript, Python have eval(...) statements
  • Back end might run system commands (ls on a directory)
• These have the same vulnerabilities
  • Solutions are similar (but no prepared statements)
  • These should be avoided if at all possible
File Naming Attacks

• Suppose your back end opens a particular file
  • Based on the user name
    • Image for user is /web/html/site/user_images/$user
  • What happens if I use the user name “..../..../..../etc/passwd”

• Solutions
  • Validate the form of the name
  • Don’t use names directly (look up image name in database)
  • Restrict access to the file system
    • chroot provides a virtual root (node.js accessible)
    • php/java/tomcat/ruby security policies
Question

A cross-site scripting attack (XSS) can involve:

A. One web site using its cookies to infect another.
B. A malicious user inserting text into a blog causing another user to run arbitrary JavaScript.
C. An IFRAME from one page accessing data from another page being displayed in the browser.
D. A malicious user redirecting traffic from one site to a look-alike,
E. None of the above,
Cross Site Scripting Attacks

• The attacker inserts arbitrary HTML on your web page
  • How can this ever happen?
  • XSS
• What can go wrong
  • Disrupt the page or the portion where inserted
Cross Site Scripting

- What if the HTML include `<script>` tags?
  - Replace the page with a new one
    - Fake instance of a page to get passwords, accounts, etc.
  - Pass information from the page to foreign page
    - Cookies, passwords, credit card numbers, session Ids
  - Download user’s cookies (passwords) for other sites
Cross Site Scripting: How

• Suppose you allow user comments
  • Guest book, ratings, wiki, postings, ...
  • Text is gotten from user, then inserted into HTML
• Suppose instead of typing “I love this page”
  “I love this page<script
     language='javascript'>document.location='http://bad/';
   </script>”
• What would happen?
Cross Site Scripting

• What can go wrong
  • Reading data from URL (session id)
  • Replace data in the URL
  • Accessing/Replacing hidden form variables
  • Loading foreign web page into a frame inside your page
    • Using JavaScript to read and manipulate that frame
    • Using code in the frame to monitor your activities

• Taking control of the browser
Cross Site Scripting: Prevention

- Don’t allow any HTML to be inserted
  - Back end libraries to strip out HTML tags
- Don’t allow malicious HTML to be inserted
  - Back end libraries to sanitize HTML
    - Limited set of allowed tags for formatting
  - Use something other than HTML
    - Map to html on output
Next Time

• Security II
• Pre-Lecture Homework
  • List the security requirements for handling user registration and login