Introduction
SepAppDev 2007
Contents of the Course

- Not so much in chronological order, but
  - Security objectives
  - Development process
  - Mechanisms in current technologies
  - Design
  - Coding
  - Quality assurance
The Problem
Software vulnerability growth

The Trinity Of Trouble: Connectivity

- The Internet is everywhere and most of our software is on it.
- When was the last time that you did business with a major vendor who had no Internet connectivity?
- Tried VoIP on your mobile phone in a coffee shop WiFi hotspot yet?
A simple user interface can be enormously complex “under the hood”
Consider what happens behind the scenes in one of today’s AJAX web applications
But it sure does make for a compelling “user experience”
The Trinity Of Trouble:

- **Extensibility**

  - Systems evolve in unexpected ways and are changed on the fly.
  - After all, who would want a computing device that can't be functionally extended?
  - From J2ME to desktop PC users (running with administrative privileges). .NET
The classic security tradeoff

Security - Functionality

Windows Complexity

© 2007, Cigital & KRvW Associates
Learning from history

- We don’t pay enough attention to our failures
- Consider other engineering disciplines
  - Transportation
  - Construction
  - Medical
Focus on function

- Too much attention is paid to functional spec
- Consider what can go wrong as well
NO SIGNAL
Security problems are complicated

CODE

- Buffer overflow
  - String format
  - One-stage attacks
- Race conditions
  - TOCTOU (time of check to time of use)
- Unsafe environment variables
- Unsafe system calls
  - System()
- Untrusted input problems

DESIGN

- Misuse of software “feature”
- Flawed cryptographic key management
- Compartmentalization problems in design
- Catastrophic security failure (fragility)
- Insecure or insufficient auditing
- Broken or illogical access control (RBAC over tiers)
- Signing too much code
Code example: The dreaded buffer overflow

- Overwriting the bounds of data objects
- Allocate some bytes, but the language doesn’t care if you try to use more
  - char x[12];
  - x[12] = ‘\0’;
- Why was this done? Efficiency!
- Two main flavors of buffers
  - Heap allocated buffers
  - Stack allocated buffers
  - Smashing the stack is the most common attack
- The second most pervasive security problem today in terms of reported bugs
- Any guesses what problem has overtaken it recently?
void main() {
    char buf[1024];
    gets(buf);
}

- How not to get input
  - Attacker can send an infinite string!
  - Chapter 7 of K&R (page 164)

- Calls to watch out for

<table>
<thead>
<tr>
<th>Instead of:</th>
</tr>
</thead>
<tbody>
<tr>
<td>gets(buf)</td>
</tr>
<tr>
<td>strcpy(dst, src)</td>
</tr>
<tr>
<td>strcat(dst, src)</td>
</tr>
<tr>
<td>sprintf(buf, fmt, a1, Ė )</td>
</tr>
<tr>
<td>*scanf(Ē )</td>
</tr>
</tbody>
</table>

- Hundreds of such calls
- Use static analysis to find these problems
  - ITS4, Fortify
- Careful code review is necessary
Design example: Microsoft WMF

- Windows Metafile Format -- used for interchange of data between programs
  - Design feature included ability to include arbitrary executable data along with a WMF file
  - Feature was included to allow cancellation of print files
  - Attacker could send a WMF file with embedded arbitrary executable code
Breaking stuff is important

- Learning how to think like an attacker is essential
- Do not shy away from carrying out attacks on your own stuff
  - Engineers learn from stories of failure
- Attacking is fun! Fun is good!
Software security: state of the practice

- **Software security** still in infancy
  - Lacking standards
  - Many “best practices” to choose from
  - Most have yet to really prove themselves
- Information/guidance resources are appearing quickly
  - Study and adopt to your needs

- Tools are getting better, but only cover coding defects
  - Leave much to be done manually

Software security is not security software!
Software security is about building things properly.
What can be done?

Strive for the following criteria

- Repeatable
- Predictable
- Businesslike
- High quality
- Measurable

Must be firmly embedded into entire existing dev process without breaking it.
Solution sets abound

Several “best practices” options to choose from, including

- OWASP’s CLASP
- Microsoft’s SDL
- Cigital’s “touchpoints”

Each has strengths and weaknesses

- Best bet is to learn each and adapt the aspects that work best in your organization
- Alignment with extant build process is vital
Three pillars of software security

- Risk management framework
- Secure SDLC practices or “touchpoints”
- Knowledge catalog
Why risk management?

- Business understands the idea of risk, even software risk
- Technical perfection is impossible
  - There is no such thing as 100% security
  - Perfect quality is a myth
- Technical problems do not always spur action
  - Answer the “So what?” question explicitly
- Help customers understand what they should do about software risk
- Build better software

So what?
The Cigital risk management framework

1. Understand the Business Context
   - Artifact Analysis
   - Business Context
   - Identify the Technical Risks
   - Artifact Analysis

2. Identify the Business Risks
   - Technical Expertise

3. Identify the Technical Risks

4. Synthesize & Prioritize the Risks

5. Define the Risk Mitigation Strategy

6. Fix the Artifacts

7. Validate the Artifacts

Measure & Report

Initiate Process Improvement
Software security touchpoints

- Security requirements
- Abuse cases
- Risk analysis
- External review
- Risk-based security tests
- Code review (tools)
- Risk analysis
- Penetration testing
- Security operations

- Requirements and Use Cases
- Architecture and Design
- Test Plans
- Code
- Tests and Test Results
- Feedback from the Field
Knowledge catalogs

- Principles
- Guidelines
- Rules
- Attack patterns
- Vulnerabilities
- Historical Risks
Managing knowledge

- Perhaps the toughest hurdle
  - Combines people, skills, experience, etc.
  - Training helps, but there is no substitute for experience
- Start with clear targets in mind
  - Train to get started
  - Hire qualified people
- Mentoring is vital
  - Apprenticeship still plays its roll
Will this stuff work?

When applied thoughtfully, there is no reason that you can’t produce measurable improvements in your software

- Don’t get too hung up on process
- Take small steps towards your goal
- Start measuring immediately

*If you can’t measure it, how can you manage it?*