

Computer Security

CS 426

Lecture 6



Building Secure Software

Readings for this lecture

- This lecture is mostly based on “Software Security: Building Security In”
 - Addison-Wesley, Gary McGraw
- Other useful reference books
 - Threat Modeling, Frank Swiderski & Window Snyder
 - Exploiting Software, Greg Hoglund & Gary McGraw

While the software security problem is growing

- The Trinity of Trouble connectivity
 - connectivity
 - extensibility
 - complexity
- Code-level Bugs vs. Design-level Flaws
 - 50/50

Three Pillars of Software Security

- Pillar 1: Applied Risk Management
- Pillar II: Software Security Touchpoints
- Pillar III: Knowledge

Risk Management Framework

- Understand the business context
- Identify the business and technical risks
- Synthesize and prioritize the risks, producing a ranked set
- Define the risk mitigation strategy
- Carry out required fixes and validate that they are correct

Tasks for Software Security Risk Management

- Create security abuse/misuse cases
- Listing normative security requirements (and security features and functions)
- Performing architectural risk analysis
- Build risk-based security test plans
- Wielding static analysis tools
- Performing security tests
- Performing penetration testing in the final environment
- Cleaning up after security breaches

Seven Software Security Touchpoints

[Software Security, Gary McGraw]

1. Code review
2. Architectural risk analysis
3. Penetration testing
4. Risk-based security tests
5. Abuse cases
6. Security requirements
7. Security operations

An Example

1. `read(fd, userEntry, sizeof(userEntry));`
2. `comparison = memcmp(userEntry,
 correctPasswd, strlen(userEntry));`
3. `if (comparison != 0)`
4. `return (BAD_PASSWORD);`

Code Review with Static Analysis Tools

- Aims at finding bugs (rather than flaws)
- Three possibilities for code review
 - manual analysis
 - static source code analysis
 - binary code analysis
 - dynamic analysis

Approaches to source code analysis

- Use grep
 - match a stream of letters against pattern
 - lo-fidelity in terms of analyzing code meaning
 - try “bug”, “XXX”, “fix”, “assume”
- Lexical analyzer
 - match a stream of tokens against pattern
- Analyze the syntax tree
 - Scope of analysis: Local analysis, Module-level analysis, Global analysis

Existing Tools: Research Prototypes

- BOON [Wagner et al. 2000]
 - uses integer range analysis to check for buffer overflow
- CQual [Foster et al.]
 - uses type qualifiers to perform taint analysis
 - can detect format-string vulnerabilities, user/kernel pointer bugs
- xg++ [Ashcraft and Engler 2002]
 - use compiler extensions to check for patterns, e.g., use data from untrusted source without checking first

Existing Tools: Research Prototypes

- The Eau Clair tool [Chess 2002]
 - uses a theorem prover to check function implementations behave as expected
- MOPS [Chen and Wagner 2002]
 - uses model checking to look for violations of temporal safety properties
- Splint [Larochelle and Evans 2001]
 - (extends lint concept to security)
 - developers can add annotations to help check for potential bugs

Commercial Tools

- Coverity
- Fortify
- Ounce Labs
- Secure Software

Architectural Risk Analysis: A Basic Risk Analysis Approach

- Aim at finding design-level flaws
- Start from some high-level model of the system (often include multiple tiers)
- Consider
 - the threats who are likely to want to attack our system
 - the risks present in each tier
 - the kind of vulnerabilities that might exist in each component, as well as the data flow
 - the business impact of such technical risks, were they to be realized
 - the probability of such a risk being realized
 - any feasible countermeasures that could be implemented

An Example: Analysis of a Voting System

- **Analysis of an Electronic Voting System**
 - Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, Dan S. Wallach
- Analyzes the Diebold electronic voting system
 - AccuVote-TS DRE system, Oct 2000 - April 2002
 - Available on open ftp server
 - Identified by activist Bev Harris
 - Some zip files, cvs repository
 - DMCA concern over zip “encryption”
 - Available on New Zealand site

Problems found

- No authentication of smartcard to voting terminal
- Votes kept in sequential order
- Several glaring errors in cryptography
 - e.g., Encrypted votes and audit logs
 - `#define DESKEY ((des_key*)"F2654hD4")`
- Default Security PINs of 1111 on administrator cards
- Windows Operating System
 - tens of millions of lines of code
 - new “critical” security bugs announced frequently

Sample comment in code

```
// LCG - Linear Congruential Generator  
// used to generate ballot serial numbers  
// A psuedo-random-sequence generator  
// (per Applied Cryptography,  
// by Bruce Schneier, Wiley, 1996)
```

*Unfortunately, linear congruential generators
cannot be used for cryptography"*

Page 369

Applied Cryptography, by Bruce Schneier

- BallotResults.cpp

Diebold Election Systems

In the News: California withdraw Certification of Voting Machines

- On August 3, 2007, California Secretary of State Bowen announced her decisions regarding which systems in the review will be permitted to be used in the 2008 elections and beyond.
 - withdraw of approval for some models of Election Systems, and allow conditional usage of some other models

http://www.sos.ca.gov/elections/elections_vsr.htm

The STRIDE Model

- Introduced by Howard and LeBlanc from Microsoft in “Threat Modeling”
- Cycling through a list of attacks
 - Spoofing
 - Tampering
 - Repudiation
 - Information disclosure
 - Denial of service
 - Elevation of privilege

Three Critical Steps for Architectural Risk Analysis

- Ambiguity analysis
 - have at least two analysts to carry out separate analysis in parallel, then unify the results
 - when differences in understanding how the system works occur, security risks probably exist
- Weakness analysis
 - understand the impact of external software dependencies
 - e.g., relying on browsers, standard protocols that have weaknesses (RMI, COM), interposition attacks

Three Critical Steps for Architectural Risk Analysis

- Attack resistance analysis
 - use information about known attacks, attack patterns, and vulnerabilities to analyze how well the system defend against them
 - examples:
 - transparent authentication token generation/management
 - misuse of cryptographic primitives

Software Penetration Testing

- Difficulties
 - security defects are often not directly related to functionalities
 - Functional testing is testing for positive
 - Testing for negative

How to Do Penetration Testing

- Thinking with a black hat
- Testing a system in its final production environment, probing configuration problems and other environmental factors
- Use tools
 - static as well as dynamic tools, e.g., fuzzing: submit malformed, malicious, and random data as input
 - attacker's toolkit: disassemblers and decompilers, control flow and coverage tools, APISPY (examine outside API/DLL calls), Debugger with breakpoint setters and monitors,
- Test more than once
- Incorporate findings into development

Risk-based Security Testing

- Pen Testing occurs when software is complete and installed in operation environment
- Security testing occurs when software is complete
- Functional security testing:
 - make sure that security mechanisms properly implemented
- Adversarial security testing
 - perform tests motivated by understanding and simulating the attacker's approach

Abuse Cases

- Use cases focuses functionality, what should happen from the users' point of view
 - e.g., “the system allows users in the HR management group to view and modify salaries of all employees”
- Misuse cases help think from the attackers' point of view
- Misuse cases decide and document a priori how the software should react to illegitimate use
 - the process help differentiate appropriate use from inappropriate use
 - forces people to ask the right question

How to Create Misuse Cases

- A practical approach to create abuse cases is through informed brainstorming
- Don't believe "but no one would ever do that"
- Start by knowing the requirements, standard use cases, and attack patterns
- Identify and document threats (who may want to attack and why)
- Create anti-requirements
 - things that you don't want your software to do
- Create an attack model
 - cycle through attack patterns to see whether they apply

48 Attack Patterns from Exploiting Software [Hoglund and McGraw 2004]

- Make the client invisible
- Target programs that write to privileged OS resources
- Use a user-supplied configuration file to run commands that elevate privilege
- Make use of configuration file search paths
- Direct access to executable files
- Embedding scripts within scripts
- Leverage executable code in Nonexecutable files
- Argument Injection
- Command delimiters
- ...

Software Security Meets Security Operations

- Bridge operational security expertise (network security, administration) with software security expertise

Software Security Knowledges

- Principles
- Guidelines
- Rules
- Vulnerabilities
- Exploits
- Attack patterns
- Historical risks

Coming Attractions ...

- September 11: Access Control for Process Confinement
- Additional readings
 - Best Practices for UNIX chroot() Operations,
 - The Linux kernel capabilities FAQ
 - wikipedia topics: Introduction to virtualization, Operating system-level virtualization, FreeBSD jail,

