Auditing Software

TDT60
06.09.2005
Introduction

• Open source vs. closed source

• The many-eyeballs phenomena

• When to audit, how to audit, what tools to use…
Architectural Security Analysis

- Performing a risk analysis on a product design

- Three phases (may overlap – but focus differs):
  - Information gathering
    - Requirements – that address security
    - High-level architecture
    - Identify security-critical areas of design
    - System + environment
  - Analysis
    - Exploring attacks that one could launch against a system
    - Assess risks – rank in order of severity
    - Identify countermeasures
  - Reporting
    - Ranked risks – based on analysis phase
    - For each risk:
      - Overview of attack
      - Discussion of potential consequences
      - Mitigation techniques – how to reduce risk

- The results of an architectural risk analysis can be used to guide and focus an implementation analysis.
Attack trees

• Used in the analysis phase
• A concept derived from fault trees used in software safety
• => build a graph to represent the decision-making process of attackers
  – Root: goal
  – Leaves: ways of achieving the goal
  – Lower in the tree -> more specific attack
  – Pruning node: specifies what conditions must be true for its child nodes to be relevant
• It is hard to construct a complete attack tree
  – Should be confident that it is reasonably complete
**Attack tree example – SSL**

**Goal: intercept a network connection for a particular user**

1. Break the encryption  
   1.1 Break the public key encryption  
      1.1.1 Using RSA?  
         1.1.1.1 Factor the modulus  
         1.1.1.2 Find a weakness in the implementation  
         1.1.1.3 Find a new attack on the cryptography system  
      1.1.2 Using El Gamal?  
      ....  
   1.2 Break the symmetric key encryption  
      ....  
   1.3 Break the use of cryptography in the protocol  
      ...

2. Obtain a key  
   2.1 Use uses public key authentication  
      2.1.1 Obtain private key of user  
      2.1.1.1 Obtain encrypted private key (AND)  
      ....  
      2.1.1.2 Obtain pass phrase  

3. Obtain a password  
   ....

4. Attempt man-in-th-middle attack  
   ....
Using attack trees

- Get data organized in a way that is easy to analyze
- Assigning value to nodes: perceived risk
  - Time/effort
  - Cost
  - Risk to attacker
- Identify cheapest and/or most likely attack
- Pruning criteria: determine at each node whether it violates criteria
  - Example: insider attacks
- Building attack tree
  - Identify assets (data and resources) of a system (target)
  - Identify modules, communication points between modules, classes of users (most likely failure points)
  - NB: include also
    - Any off-the-shelf components
    - Computers the software is running on
    - Network etc
- Requires expert knowledge
Implementation Security Analysis

1. Validate whether the implementation actually meets the design
   • Involves picking through the code by hand – trying to ensure that the implementation matches the design (time-consuming)
   • Shortcut: ask developers questions about the implementation (less time-consuming, lower quality of test)

2. Looking for implementation-specific vulnerabilities
   • Search for flaws that do not show up in design (e.g., buffer overflows)
   • Language specific vulnerabilities

⇒ Generally more complex and time-consuming than architectural security analysis
   – Complex code
   – Follow data flow
   – Subtle problems
Auditing source code

*Complexity of code vs. complexity the human brain is able to manage*

- **Strategy:**
  - Identify location of user input
  - Identify location of input from other programs or any other potentially untrusted source
  - Examine how internal API handles input
  - Look for symptoms of problems
    - Often: function calls to standard libraries that are frequently misused
  - Manual analysis
    - Hard: often rewrite better
  - Supplement with code review (as thorough as you can afford)
Source-level security auditing tools

• Search for language elements in source code that are commonly involved in security-related implementation flaws

• Examples
  – C/C++:
    • RATS, Flawfinder, ITS4
  – Java:
    • JLint, BugFinder, PMD, CheckStyle
The effectiveness of security scanning of software

• They still require a significant level of expert knowledge
  – To evaluate potential impact of discovered flaws
• Even for experts, analysis is still time-consuming.
  – Use of automatic scanners only eliminates ¼ to 1/3 of total time spent on code inspection
• Every little bit helps
• They can help find real bugs