Manual vs. Automated Vulnerability Assessment: A Case Study

James A. Kupsch and Barton P. Miller
University of Wisconsin

MIST 2009, Purdue University
June 16th, 2009
Study Motivation

How should we view static analysis tools?

- Literature on static analysis tools, papers are almost self limiting:
  - missing is comparison against security as a whole
  - Very hard to look beyond the immediate techniques and say what they are not finding
  - tool writers write about what they have found
- Every valid new thing tools find is progress, but it’s easy to lose perspective on what these tools are not able to do
- Good science and engineering quantifies and verifies limitations instead of just stating them
Our Work

• We have a strong manual assessment protocol
• Applied it to several significant projects:

**Condor**
University of Wisconsin
Batch queuing workload management system

**SRB**
SDSC
Storage Resource Broker - data grid

**MyProxy**
NCSA
Credential Management System

**glExec** (in progress)
NIKHEF
Identity mapping service
Our Work (cont.)

Technique has been extremely successful
- found critical problems
- helped groups redesign software
- changed their development practices and release cycle management
First Principles Vulnerability Assessment (FPVA)

- **Insider** - have access to
  - developers
  - source code
  - documentation
- **Independent** from development team
  - no agenda
  - no blinders
- **First Principles** - let the process guide where to look
FPVA
Understanding the System

Architectural Analysis - functionality and structure of the system, major components, communication channels

Resource Analysis - objects in the system and operations allowed

Trust & Privilege Analysis - trust boundaries of components, privilege presented to users, and external privilege model used components
Condor Data Flow Diagram

Real UIDs
- root
- condor
- user

Submit Host
- User
  1. Job Description File
- submit
  2. Job ClassAd

- startd
  7. Fork Shadow

- schedd
  8. Establish Communication Path

- shadow
  4. Negotiation Cycle

Central Manager
- negotiator
  3. Job ClassAd

Execute Host
- startd
  7. Starter

- schedd
  6. Claim Host

- starter
  9. Set policy and fork User Job

- User Job

1. Machine ClassAd

4. Negotiation Cycle
5. Report Match

Match

7. Fork Shadow
FPVA - Component Analysis
Search for Vulnerabilities

• Connect user supplied data to security violation
• Audit source code
• Guide search using
  – previous analyses results and diagrams
  – knowledge of vulnerabilities
FPVA Condor Results

15 significant vulnerabilities discovered
http://www.cs.wisc.edu/condor/security/vulnerabilities

- 7 implementation bugs
  - easy to discover - localized in code
  - use of troublesome functions:
    exec, popen, system, strcpy, tmpnam

- 8 design flaws
  - hard to discover in code - higher order problems
  - defects include:
    - injections, directory traversals, file permissions, authorization & authentication, and a vulnerability in third party library
Case Study

• Condor 6.7.12
  – 15 discovered vulnerabilities present
  – small changes to build with newer compiler

• Talked to academics, military, and industry people about what they thought were the best tools: Coverity and Fortify

• Analyze Condor 6.7.12 with tools
  – Coverity Prevent 4.1.0
  – Fortify SCA 5.1.0016

• Review tool output
  – defect with matching location of known vulnerability is a positive result
  – sample tool output to understand results
Using the Tools

Three step process:

1. **Collect Build Information**
   - easy, no modification to build process
   - tool monitors complete build of project

2. **Run Analysis Phase**
   - uses build information

3. **View Results**
   - categorized by type, location, ...
   - presents evidence of defect - what, where, how
   - source code browser
   - defect management capabilities
## Tool Result Summary

<table>
<thead>
<tr>
<th></th>
<th>Coverity</th>
<th>Fortify</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defects Found</td>
<td>2,986</td>
<td>15,466</td>
<td>total</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2301</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8101</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5061</td>
</tr>
<tr>
<td>Defect Categories</td>
<td>70</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>FPVA Defects Found</td>
<td>1</td>
<td>6</td>
<td>total</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>6</td>
<td>impl. bug</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>design flaw</td>
</tr>
</tbody>
</table>
FPVA Tool Discovered Defects

• Simple implementation bugs found
  – Coverity found 1
    • errs on the side of false negatives
    • only flags certain functions when input can be proven to come from untrusted sources
  – Fortify found 6
    • errs on the side of false positives
    • will always flag certain functions
  – One defect not present on platform, neither found

• No design flaw defects found
Other Tool Discovered Defects

- **Security Issues**
  - potential null pointer dereferences
  - minor resource leaks
  - no other major security issues in sample

- **Correctness Issues**
  - buffer overflows (small number of bytes)
  - uninitialized variables
  - failure to check errors

- **Code Quality Issues**
  - maintenance problems
  - future problems

- **False Positives**
  - also can be a code quality issue
  - can be tool misinterpreting code
Defect Comments

• Number of defects can be overwhelming for mature projects
• Easier with new project
  – require 0 new defects
  – learn to program in tool’s style
• Requires manual inspection to determine defect’s real severity (may be faster to fix)
• Defect can easily change from one category to another as code is modified (even far away code)
Tuning the Tools

• Both tools allow some types of user defined checks
• Make Coverity find all FPVA defects that Fortify found:
  – flag uses of functions: `exec`, `popen`, `system`
  – even without proving tainted arguments
Contributions

• Showed limitations of current tools
• Presented manual vulnerability assessment as a required part of a comprehensive security assessment
• Creating a reference set of vulnerabilities to perform apples-to-apples comparisons
Questions

For more information see:
http://pages.cs.wisc.edu/~kupsch/vuln_assessment