The Benefits of Static Compliance Testing for SCA Next

R-Check™ SCA

James Ezick
Jonathan Springer

Reservoir Labs, Inc.
New York, NY

SDR ’11
Wireless Innovation Conference & Product Exposition

2 December 2011
Static Compliance Testing with R-Check™ SCA

Outline

Introduction to Static Analysis
- What is Static Analysis?
- Capabilities of Static Analysis
- Successes from the State of the Art

Static Analysis and the SCA
- Relating the Specification to Testing
- Unique Challenges of the SCA

R-Check SCA
- Modern Static Analysis Customized to the SCA
- Looking Ahead to SCA Next
- What is Possible ...
Introduction to Static Analysis

Static Analysis seeks to find bugs through inspection of source code rather than through the execution of the program

- Analyzes all possible program paths without bias
- Can be run on code in an intermediate state
- Integrates with development environments

What can it do?
- Provide reproducible, automated tests
- Explain specifications, answer “what ifs”
- Generate counter-examples

What are the limitations?
- Depending on how specifications are written, some problems are very hard

What infrastructure is needed?
- Works best within a tool that can break code down into data-structures
Foundations of Static Analysis

G. Kildall – Dataflow Analysis (1973)
- Equations for deriving facts that hold at each program point
- Solution reduces to finding a fixed point over a lattice
- Foundation of modern compiler-driven optimization (e.g., live variable analysis, use-before-def detection)

M. Sharir & A. Pnueli – Interprocedural Analysis (1978)
- Extended the equations to support flow through procedures
- Added a level of abstraction to support calling contexts

- Logical sentences (CTL, LTL, etc.) over abstract labeled transition systems (Kripke Structures)
- 2007 ACM Turing Award

Field has a deep history with a solid mathematical foundation – not just a bag of tricks!
Successes from the State of the Art

Locks over the Linux Kernel (SATURN, Stanford)
- **Precise checking** of lock/unlock sequencing
- Use constraints to model conditional branches
- Found hundreds of previously unknown errors, low false positives

Counter-Example Guided Abstraction Refinement (CMU)
- **Automate** the abstraction process for a program
- Finds faults in programs using model checking techniques
- Big leap forward in proving properties about real systems

Proving Termination (MS Research)
- Provides **usable results** for an impossible problem!
- Applicable to *liveness* properties – what must happen
- *Proving Program Termination*, CACM, May 2011

The Halting Problem:
There is always a record that breaks the player
Static Analysis for the SCA

JTEL SCA 2.2.2 Applications Requirement List

<table>
<thead>
<tr>
<th>Requirement Tag</th>
<th>Criterion Tag</th>
<th>Requirement/Criterion Text</th>
<th>Section Number</th>
<th>Test Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP0603</td>
<td></td>
<td>Applications shall be limited to using the OS services that are designated as mandatory in the SCA Application Environment Profile (Appendix B).</td>
<td>3.2.1.1</td>
<td>Manual</td>
</tr>
</tbody>
</table>

Simple – Can be performed with search and inspect
- Benefits from a context aware parsing – preprocessor, syntax, library awareness

<table>
<thead>
<tr>
<th>Requirement Tag</th>
<th>Criterion Tag</th>
<th>Requirement/Criterion Text</th>
<th>Section Number</th>
<th>Test Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP0604</td>
<td></td>
<td>Applications shall perform file access through the CF File interfaces.</td>
<td>3.2.1.1</td>
<td>Manual</td>
</tr>
</tbody>
</table>

Deceptive – Simple statement, but non-trivial to test
- Requires an enumeration of what is not allowed – domain & language expertise

<table>
<thead>
<tr>
<th>Requirement Tag</th>
<th>Criterion Tag</th>
<th>Requirement/Criterion Text</th>
<th>Section Number</th>
<th>Test Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP0075</td>
<td></td>
<td>The releaseObject operation shall release all internal memory allocated by the component during the life of the component.</td>
<td>3.1.3.1.2.5.2.3</td>
<td>Manual</td>
</tr>
</tbody>
</table>

Holy grail – Reducible to the locking or termination problems
- Simple and intuitive statement – really hard to get right
- Balance between eliminating false-negatives, limiting false-positives, speed
- Opens the door to the deepest types of analysis available today
Why Memory Leaks are Hard

Lessons from Examples

Memory leaks cannot be found by simply inspecting the memory allocation/deallocation lines

• Context, sequencing matter
• Semantics matter

These errors occur in real code

• Linux kernel (c.2005)
  Open source – thousands of sets of eyeballs – hundreds of undiscovered lock bugs

A safe, conservative analysis requires deeper analysis tools

Example 1: Memory Leaks through Pointer Reassignment
Component::method_a() {
    p = malloc(...);
    ...
    p = malloc(...);
}
Component::releaseObject() {
    free(p);
}

Second malloc() leaks memory allocated by first malloc()

Example 2: Memory Leaks through Control Flow
Component::method_a() {
    if (A) {
        p = malloc(...);
    }
}
Component::releaseObject() {
    if (B) {
        free(p);
    }
}

If “A” evaluates to true, but “B” does not, then memory allocated by malloc() will be leaked
Static Analysis for the SCA

Static Analysis isn't limited to just C/C++ source code

SCA 2.2.2 also puts requirements on XML domain profile files ...

<table>
<thead>
<tr>
<th>AP0613</th>
<th>C174</th>
<th>The device that loaded this component refers to a specific component found in the assembly, which is used to obtain the logical CF Device that was used to load the referenced component from the CF ApplicationFactory.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>D.6.1.5.1.1.6</td>
</tr>
</tbody>
</table>

May also want to analyze CORBA IDL files ...

- These files define implementation contracts with the source code

And check consistency requirements across file types ...

- Profile matches interface description matches implementation

Or check non–SCA–specific properties

- Memory leaks, memory/pointer usage
- API usage requirements
R-Check SCA

Goal: Draw from the most successful ideas in static analysis to develop a solution customized to the SCA

Version 1.0

- Structure and context aware
- C and C++, POSIX and CORBA support
- Intermediate representation that supports advanced analysis techniques
  - Type system
  - Control–flow abstraction
- Support for XML, CORBA IDL
- Scales to enterprise code
  - Including incomplete/in–development code
- Push–button support for SCA tests
R-Check SCA Workflow

**CORBA IDL & XML Descriptors “as is”**

- IDL/XML
  - .IDL
  - .XML

**C/C++ Source Code for Analysis “as is”**

- Source .CPP
  - C

---

**Extract R-Check SPaR**

**Feature Summary**

- Summary .XML

**Project Config File**

**Configure R-Check ProCon**

**Source Code Paired with Compilation Flags**

- Flags
  - Source .CPP
  - .C

**Analyze R-Check**

**Compliance Report for Each Source File**

- Reports .XML

**Consolidate R-Check Blender**

**Merged Compliance Report**

- Report .XML

---

**With GNU Make Enabled Scripting:**

- Source Compiler Replacement
- Linker Replacement
- Build Targets

---

**Report**

- Report .CSV
- Report .TXT
- Report .HTML
Looking Ahead to SCA Next

*We expect static testing to become an even more integral component in SCA Next certification*

New Challenges making Dynamic Analysis Harder

- More flexibility in interface (e.g., CORBA vs. no-CORBA)
- More flexibility in capability supported
- Data hiding – component interfaces behind Domain Manager

Opportunities

- Static testing tool can be used to “teach” the specification with each compile operation

Providing meaningful guarantees requires an accord among

- **Specification authors**: What the specification says
- **Testers**: Tools available (time vs. precision), what can be tested
- **Developers**: How code is written
What is Possible

SCA Next
- R-Check SCA architecture extends to SCA Next
- SCA Profiles
- Support for Platform Specific Model
- Retain push-button functionality

Deeper Analyses
- Add flow and path-sensitivity, more precision
- Supported by R-Check SCA architecture

Direct Query Interface
- Write new analyses using structured natural-language syntax
- Motivated by model-checking ideas (logic sentences)
- Ask questions about what the radio might do
Privately owned, Reservoir Labs has been providing leading-edge consulting and contract R&D to the computer industry, business, end-users, and the US Government since 1990

Expertise

- Custom verification solutions
- Applied compiler research for emerging high-performance and embedded architectures
- Reasoning, constraint solving, and mathematics
- Cyber-security, deep network content inspection

Technologies

- R-Check Static Analysis Platform
- R-Stream Mapping Compiler
- R-Solve Reasoning and Planning Technology
- R-Scope Network Security Technology

Reservoir Labs’ offices in New York, NY and Portland, OR
Acknowledgements

Development funded by SPAWAR under Navy Phase II SBIR Contract “Static Analysis Tool for Interface Compliance Verification and Program Comprehension”

Thanks!

John Thom, TPOC (SPAWAR)

JTEL Test Execution Team

Jim Kulp, Parera Information Services
For More Information on R-Check SCA

Visit our website
• https://www.reservoir.com/rcheck

Contact us by email
• rccheck-support@reservoir.com

Updated version of this presentation
• https://www.reservoir.com/rcheck