Path Projection for User-Centered Static Analysis Tools

Jeff Foster
University of Maryland, College Park

Joint work with YitPhangKhoo and Mike Hicks

(Not yet published, so don’t steal these ideas!)
Introduction

• Many recent successes in static analysis tools for defect detection / prevention
  ■ Lots of progress in the research community
  ■ Coverity, Fortify, and others sell static analysis tools
  ■ Microsoft has had great success with static analysis

• Major research focus: Building tools that are...
  ■ “Precise enough”
Motivation

• Static analysis tools are not perfect
  ■ Users must *tria*ge bug reports
    - Decide whether true or false positives, and how important
  ■ Users must *remediate* true bugs

• Conclusion: successful static analysis requires cooperation between the user and the tool

• How do we build more user-centered static analyses?
Path Projection

• A new interface to help users visualize code paths
  ■ E.g., call stacks, control flow paths, data flow paths

• Core principles
  ■ Remain true to original source code
  ■ Fit as much on one screen at a time as possible
Contributions

• Prototype implementation in WebKit

• Controlled user study
  ■ Task: Triaging Locksmith error reports
  ■ Compared to “standard viewer” (similar to IDEs)

• Experimental results
  ■ Improved performance (completion time)
  ■ Same accuracy
  ■ Qualitatively better
Sample Locksmith Error Report

Warning: Possible data race of g_conn_open (knot.c:<global>:61)
at:
1. <in knot.c>
   main():601  -> dereference
   locks: -

2. <in knot.c>
   main():558  -> pthread_create()
   thread_main_autospawn():458
   accept_loop():395  -> dereference
   locks: -

3. <in knot.c>
   main():577  -> pthread_create()
   thread_main():476
   accept_loop():395  -> dereference
   locks: -

Shared variable
No locks held at deref
Thread creation site
Three possibly-racing derefs (paths)
Triaging a Locksmith Report

• Three things to check:
  1. Both accesses refer to same location
     - Locksmith’s alias analysis may be imprecise
  2. Locksmith has not missed a held lock
     - Could happen at join points in cfg
  3. Potentially-racing accesses can occur, simultaneously
     - Both stack traces given must be simultaneously realizable

• This work: Focus on task 3
  ■ Leave 1 and 2 as future work
Splittable file viewer

Basic searching

Hyperlinked error report

Triaging checklist
Standard Interface Demo
Challenges with Standard Interface

• Requires lots of scrolling through code
  ■ Hyperlinks help a little
  ■ Still hard to keep track of the *context* along the path

• Need to compare multiple paths together
  ■ Are both realizable?

• Need to visually switch between error report and program source code
  ■ Adds cognitive burden
Multiple simultaneous searches
Path Projection Demo
Informational Visualization Strategies

• *Increase user’s memory and processing resources, and reduce the search for information*
  - Make important lines of code visible on screen
  - Place related lines of code close together

• *Use visual representation to enhance pattern matching*
  - Put function definition in colored boxes
  - Format/color code to reveal program structure

• *Encode information in a manipulable medium*
  - Allow users to search/highlight
User Study: Overview

• Standard Viewer (SV) vs. Path Projection (PP)

• Task: Triage a Locksmith error report
  ■ Decide whether it is a false positive or not

• Measurements
  ■ Completion time for the task
  ■ Qualitative feedback from users
  ■ Observations of user behavior
Locksmith Task Details

• All trials from Locksmith benchmarks
  ■ E.g., web server, ftp client, etc.
  ■ Roughly 1,500 lines each
  ■ Unfamiliar to participants

• One warning per trial
  ■ No need to manage warnings
Locksmith Task Details (cont’d)

• No potential aliasing issues
  ■ All shared variables and locks are global
  ■ (Just a property of these particular warnings)

• Semantics-preserving simplifications:
  ■ Made local static variables global
  ■ Changed wait()/signal() to join()
  ■ Deleted #if 0 or other conditional macros
  ■ Converted some goto/switch statements to if
Within-subjects Design

• Two possible schedules for a participant
  ■ Same problems in same order

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Session 1</th>
<th>Session 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 2</td>
<td>PP (1.1, 1.2,</td>
<td>SV (2.1, 2.2,</td>
</tr>
<tr>
<td></td>
<td>1.3)</td>
<td>2.3)</td>
</tr>
<tr>
<td></td>
<td>SV (1.1, 1.2,</td>
<td>PP (2.1, 2.2,</td>
</tr>
<tr>
<td></td>
<td>1.3)</td>
<td>2.3)</td>
</tr>
</tbody>
</table>

• Pros:
  ■ Participants can directly compare both interfaces
  ■ Fewer problems due to individual variances

• Cons:
  ■ Order effect: may prefer first interface
  ■ Learning effect: may become better at task over time
Experimental Procedure

• Pre-questionnaire (background/demographic)

• Each session
  ■ Tutorials on data races, Locksmith, the interface
  ■ One practice trial
  ■ Three measured trials
    - First, complete task; measure time
    - Then, repeat same task and explain aloud
      - Helps users learn faster, and gives us some insights
      - *We do not* tell users whether their reasoning is correct
The Learning Effect

• Triaging a possible data race is hard!

• Participants
  ■ Did not have a lot of experience with this
  ■ Were unfamiliar with Locksmith (except one user)
  ■ Tended to get side-tracked during the task

• Two solutions
  ■ Extensive pre-experiment tutorials
  ■ A *checklist* to guide the user
Warm-up task

One tab per task component

Locksmith Checklist

Please verify that the locks reported at the end of each path in the path report, if any, are correctly acquired in the source code (list the function:line, or none):

<table>
<thead>
<tr>
<th>Locks</th>
<th>Path 1</th>
<th>Path 2</th>
<th>Path 3</th>
<th>1-2</th>
<th>1-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Submit completes checklist (and ends trial)

Anecdotally, checklist improves time by 41%
Tab for a Unprotected Access

- Deference with no locks held
  - May race with itself if called from multiple threads
    - `while(1) { fork { *p++ } }`
  - One special case to look for (only case in trials)
- Questions only enabled as appropriate
# Tab for Two Accesses

For threads leading to dereferences in Paths $i$ and $j$:

Are they parent-child (or child-parent), or child-child?

<table>
<thead>
<tr>
<th>Parent-child / Child-child</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

**Parent-child (or child-parent) threads.**

Does the parent’s dereference occur after the child is spawned?

- Before its dereference, does the parent wait (via `pthread_join()`) for the child?
  - If no, there is likely a race. Are there reasons to show otherwise?

  Explain:

<table>
<thead>
<tr>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Child-child threads.**

Are the children mutually exclusive (i.e., only one can be spawned by their common parent/ancestor)?

- If no, there is likely a race. Are there reasons to show otherwise?

  Explain:

<table>
<thead>
<tr>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Participants and Equipment

- 8 student participants
  - 3 undergraduates, 5 graduates
  - Prior experience in C, multithreading (not necc. C)
  - Self-rated experience: 3 to 4
    - Scale of 1: no experience to 5: very experienced
  - 2 participants had experience in Locksmith and Eraser

- Apparatus
  - 24" 1920-by-1200 LCD
  - Mac OS X 10.5.2
    - All shortcuts disabled except for cut/copy/paste/find/find-next
Mean Time for All Participants

Completion time (sec)

- Standard Viewer
- Path Projection

PP-SV: 55s
SV-PP: 188s

Still a learning effect

Both improvements statistically significant
PP improvement large
SV improvement small-med
Accuracy and Detailed Times

<table>
<thead>
<tr>
<th>Completion times and accuracy for each trial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trial</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>User</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>User</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Tabs</th>
<th>3</th>
<th>2</th>
<th>6</th>
<th>6</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
</table>

* one incorrectly answered tab in the checklist

- Similar # of mistakes: 10 PP (10.9%), 9 SV (9.8%)
- Mistakes in 2.2, 2.3 due to common, unrealizable sub-path
Mouse Hover Time in Error Report

- 20s on average for PP, compared to 1:34 for SV
  - Little need to use hyperlinks under PP
Overall Impression (Qualitative)

- Boxplot
  - Centerline = median
  - Whiskers = min/max
  - Box extent = quartiles
  - Dots = outliers

Strongly Disagree

Strongly Agree
Overall Impression (Qualitative)

- No statistically significant differences in answers
  - Small sample? Limited exposure?
- All but one preferred PP
PP Feature Ratings (Qualitative)

- All statistically significant vs. neutral response
- Generally favorable towards PP features
• Surprisingly, liked code folding/function inlining
  ■ Code folding was “the best feature” or “my favorite feature”
PP Feature Ratings (Qualitative)

- Checklist: “saved me from having to memorize rules”
- Two participants did not favor multi-query
  - But forgot multi-query had 4 default items
Threats to Validity

• Results may not generalize
  ■ Small population, students, not data race experts
  ■ Small set of programs
  ■ Learning effect still present

• Changes to programs to make task easier
  ■ Task in experiments is very focused
  ■ Understanding error reports generally requires wider range of activities

• SV interface is not production quality
  ■ Deliberate choice, to avoid giving any advantages
Summary

• Introduced Path Projection, a new interface
  ■ Side-by-side display of paths
  ■ Function call inlining
  ■ Code folding
  ■ In general, tries to follow InfoViz principles

• Experimental results suggest PP
  ■ Improves completion times
  ■ Is liked by users

• Lots more to do on this topic!