

Additional project suggestions
Finish lecturing on BDDs
Survey some symbolic analysis papers

Ganesh, Week 8
CS 6110, Spring 2011

Project suggestions

- We have the “Inspect” tool
 - Somewhat maintained
 - Used in projects at NEC Laboratories America
 - Multiple extensions made by them, as well
 - Used in benchmarking studies at UC Berkeley
 - You can contrast Inspect and Murphi/SPIN
 - One project: study ways to analyze the safety of a protocol using Murphi/SPIN; then translate to Pthreads/C and re-verify using Inspect
- Coverage analysis of an MPI Library
 - Take OpenMPI
 - Obtain coverage info for various modules when running MPI applications
 - Generate raw data for deeper testing later

Overview of Symbolic Methods for Program Analysis

Ganesh, Week 8
CS 6110, Spring 2011

Papers Surveyed

- These papers are at the site <http://klee.llvm.org/Publications.html>
- I'll be surveying these papers
 - KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs
 - AEG: Automatic Exploit Generation
 - KleeNet: Discovering Insidious Interaction Bugs in Wireless Sensor Networks Before Deployment
 - Execution Synthesis: A Technique for Automated Software Debugging
 - Server-side Verification of Client Behavior in Online Games
 - Stable Deterministic Multithreading through Schedule Memoization

KLEE: Unassisted and Automatic ...

- Authors: Cristian Cadar, Daniel Dunbar, Dawson Engler, Stanford
 - Dunbar is the main author of KLEE
- Paper in OSDI 2008
- Checked all 89 stand-alone programs in GNU CoreUtils
- KLEE-generated tests achieve line coverage – on average 90%
- Significantly beat hand-generated test coverage
- In BusyBox, achieved 100% coverage over 75 equivalent tools

Challenges

- Exponential number of code paths
 - Demo via “islower1”
 - Other demos ‘under construction’ (Greg resolving some issues wrt symbols not found..)
- Interaction with the environment (how to model the environment?)
- KLEE vs their previous EXE tool
 - Constr solving optimizations
 - Represents program states compactly
 - Heuristics for high code coverage

Challenges

- Exponential number of code paths
 - Demo via “islower1”
 - Other demos ‘under construction’ (Greg resolving some issues wrt symbols not found..)
- Interaction with the environment (how to model the environment?)
- KLEE vs their previous EXE tool
 - Constr solving optimizations
 - Represents program states compactly
 - Heuristics for high code coverage

Example: The MINIX tr routine

- Finds buffer overflow at line 18 wrt input tr ["" "")
- How does KLEE work?
 - Constructs symbolic command-line string argument
 - Constraints to 3 args and arg sizes to 1 10 and 10
 - When line 33 hit, calls STP to see what directions the branch can go
 - Forks executions in lines marked *
 - Checks at “dangerous operation” (pointer dereference) if any values allowed by current path condition would cause an error
 - Bug: arg incremented twice without checking if the line has ended
 - For this example, KLEE generates 37 tests covering all executable statements

Paper: AEG

Automatic Exploit Generation

- Thanassis Avgerinos, Sang Kil Cha, Brent Lim Tze Hao, David Brumley, NDSS 2011 Workshop
- Analyzed 14 open-source projects
- Successfully generated 16 control flow hijacking exploits, including two zero-day exploits (attacks unknown to developer)
- Their approach is based on combining source code and binary analysis
- Main idea: which paths to prioritize?
- Answer: prioritize exploitable paths!
 - Paths on which http get calls exist
 - Paths on which non-exploitable mistakes exist; but probing further may lead to exploitable mistakes
- Works by generating symbolic program arguments
- Creates tests to create buffer overflows
- Generates constraints for the return address of a vulnerable function to contain the shell code (to attain root)
- Youtube video demo of the tool is informative

KleeNet: Discovering Insidious Interaction Bugs in Wireless Sensor Networks Before Deployment

- Authors: Sasnauskas et al – Aachen University
- KleeNet discovers bugs before sensor network deployment
- Executes unmodified sensor network apps on symbolic inputs, automatically injecting nondet failures
- Found critical bugs in the microIP TCP/IP protocol stack

KleeNet insights

- In addition to symbolic inputs generated for the environment, KleeNet injects symbolic non-det events such as loss, duplication, and corruption of packets
- It also simulates node failures
- Allows the formulation of intuitive assertions about the distributed state of a sensor network
- Allows seamless repro of bugs discovered on the distributed system
 - KleeNet creates situations (packet valid/invalid) automatically
 - e.g. when node A sends to node B
 - Node B does two cases too (packet to forward; local delivery)
 - KleeNet automatically covers these scenarios which are difficult to hit with manual testing

Execution Synthesis

- Zamfir and Candea, EuroSys'10
- Given a program and a bug report, synthesize an execution that manifests the bug
 - Also synthesizes thread schedule and various program inputs for the bug to manifest
- In essence, automatic “explanations” for hard to find bugs
- Takes core dump + program being debugged
- Outputs a trace that can be then played back in the debugger with the ESD runtime environment
- May even generate a shorter bug trace
- Example: deadlock trace misses crucial pieces of info
 - Return values of external calls
 - Thread interleavings
- ESD fills in the blanks to synthesize the concrete execution

Server-side debugging

- Validates online games!
- Game developers can enable game operators to validate the behavior of game clients as being consistent with valid executions of the sanctioned client software!
- Approach demoed on Xpilot and Pac-Mac of their own design

Stable deterministic multithreading

- Memoization of schedules to help determinize and debug