Symbolic Crosschecking of Data-Parallel Code

Cristian Cadar
Department of Computing
Imperial College London

Joint work with Peter Collingbourne and Paul Kelly
[EuroSys 2011, HVC 2011]

Dawson Engler, Daniel Dunbar, Peter Pawlowski, Vijay Ganesh, David Dill, Junfeng Yang, Peter Boonstoppel, Can Sar, Paul Twohey, JaeSeung Song, Peter Pietzuch, Paul Marinescu
Dynamic Symbolic Execution

• Renewed interest in the last few years:
  – **Software testing**: high-coverage test generation
  – **Automatic bug-finding**
  – **Security**: automatic vulnerability signature generation, security testing

• Main enablers:
  – Recent advances in constraint solving
  – Mixed concrete and symbolic execution
Dynamic SymEx in Practice

• Many dynamic symbolic execution/concolic tools available as open-source:
  – CREST, KLEE, SYMBOLIC JPF, etc.

• Started to be adopted by the industry:
  – Microsoft (SAGE, PEX), IBM (APOLLO), Fujitsu (KLEE/KLOVER, SYMBOLIC JPF), NASA (SYMBOLIC JPF), etc.
Dynamic Symbolic Execution

- Dynamic symbolic execution can *automatically explore multiple paths* through a program
  - Determine the feasibility of a particular path by reasoning about all possible values using a constraint solver
- Before each dangerous operation, can check if there are *any* values that can cause an error
- For each path, can usually generate a *concrete input triggering the path*

Let the code generate its own (complex) test cases!
Scalability Challenges

Path exploration challenges

• Employing search heuristics
• Dynamically eliminating redundant paths
• Statically merging paths
• Using existing regression test suites to prioritize execution
• etc.

Constraint solving challenges

Exploit the characteristics of constraints generated by symex
• Eliminating irrelevant constraints
• Exploiting similarity between constraints
• etc.

[Joint work with Engler, Dunbar, Collingbourne, Kelly, Pawlowski, Sar, Twohey, Yang, Boonstoppel, Ganesh, Dill, Song, Pietzuch, Marinescu]
Three tools: EGT, EXE, KLEE

C code

Constraint Solver (STP)

x ≥ 0
x ≠ 1234
x = 3

x = -2
x = 1234
x = 3

[Joint work with Dawson Engler, Daniel Dunbar, Peter Pawlowski, Peter Boonstoppel, Vijay Ganesh, David Dill]
Successfully used our tools to:

• Automatically generate high-coverage test suites

• Find bugs and security vulnerabilities in complex software
Bug Finding with EGT, EXE, KLEE: Focus on Systems and Security Critical Code

<table>
<thead>
<tr>
<th>Applications</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX utilities</td>
<td>Coreutils, Busybox, Minix (over 450 apps)</td>
</tr>
<tr>
<td>UNIX file systems</td>
<td>ext2, ext3, JFS</td>
</tr>
<tr>
<td>Network servers</td>
<td>Bonjour, Avahi, udhcpd, WsMp3</td>
</tr>
<tr>
<td>Library code</td>
<td>libdwarf, libelf, PCRE, uClibc, Pintos</td>
</tr>
<tr>
<td>Packet filters</td>
<td>FreeBSD BPF, Linux BPF</td>
</tr>
<tr>
<td>MINIX device drivers</td>
<td>pci, lance, sb16</td>
</tr>
<tr>
<td>Kernel code</td>
<td>HiStar kernel</td>
</tr>
<tr>
<td>Computer vision code</td>
<td>OpenCV (filter, remap, resize, etc.)</td>
</tr>
<tr>
<td>OpenCL code</td>
<td>Parboil, Bullet, OP2</td>
</tr>
</tbody>
</table>

• Most bugs fixed promptly
JFS, Linux 2.6.10: Disk of death

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hex Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>08000</td>
<td>464A 3135 0000 0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>08010</td>
<td>1000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>08020</td>
<td>0000 0000 0100 0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>08030</td>
<td>E004 000F 0000 0000 0002 0000 0000 0000</td>
</tr>
<tr>
<td>08040</td>
<td>0000 0000 0000 ...</td>
</tr>
</tbody>
</table>

- 64\textsuperscript{th} sector of a 64K disk image
- Mount it and PANIC your kernel

[Joint work with Junfeng Yang, Dawson Engler, Can Sar, Paul Twohey]
### Bonjour: Packet of Death

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hex Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000 0000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>0010</td>
<td>003E 0000 4000 FF11 1BB2 7F00 0001 E000</td>
</tr>
<tr>
<td>0020</td>
<td>00FB 0000 14E9 002A 0000 0000 0000 0001</td>
</tr>
<tr>
<td>0030</td>
<td>0000 0000 0000 055F 6461 6170 045F 7463</td>
</tr>
<tr>
<td>0040</td>
<td>7005 6C6F 6361 6C00 000C 0001</td>
</tr>
</tbody>
</table>

- Causes Bonjour to abort, potential DoS attack
- Apple confirmed it and released a security update

[Joint work with JaeSeung Song and Peter Pietzuch]
Kerberized Telnet: Packet of Death

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hex Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>001E 8C97 BBD9 001B FC40 5983 0800 4500</td>
</tr>
<tr>
<td>0010</td>
<td>0040 8930 4000 4006 7E39 9BC6 7DE0 9BC6</td>
</tr>
<tr>
<td>0020</td>
<td>7DE1 AAA9 0017 7FBE B5A2 494D 6AF4 8018</td>
</tr>
<tr>
<td>0030</td>
<td>005C 4FAE 0000 0101 080A 014E 3CCD 1115</td>
</tr>
<tr>
<td>0040</td>
<td>029A FFFD 25FF FA25 03FF F0FF F800</td>
</tr>
</tbody>
</table>

- Crashes the telnet daemon
- Reported and confirmed by developers

[Joint work with JaeSeung Song and Peter Pietzuch]
Semantic Bugs

• Bugs shown before were all generic errors
• What about semantic bugs?

Option 1: Write specifications!
• Can find assert() violations
  (Can verify assert() statements on a per-path basis)
Option 2: Crosschecking!

- Successfully used in the past
- Great match for symbolic execution

Lots of available opportunities:

- **Different implementations** of the same functionality: e.g., libraries, servers, compiler
- **Optimized versions** of a reference implementation
- Refactored code
- **Reverse computations**: e.g., compress and uncompress
New Platforms, New Code

• Recent years have seen the emergence of new computing platforms which provide many opportunities for optimizations.

• Code is often adapted manually to benefit from these platforms. 

   Error-prone, as any manual process
SIMD Optimizations

Most processors offer support for SIMD instructions

• Can operate on multiple data concurrently
• Many algorithms can make use of them (e.g., computer vision algorithms)
General Purpose GPU Computing

(2006) NVIDIA CUDA

(2008) OpenCL
General Purpose GPU Computing

New programming model:
- Large number of threads
- Hierarchical execution and memory model
Crosschecking (Equivalence Checking)

We can find any mismatches in their behavior by:
1. Using symbolic execution to explore multiple paths
2. Comparing the path constraints across implementations
Crosschecking: Advantages

- No need to write any specifications
- Constraint solving queries can be solved faster
- Can support constraint types not (efficiently) handled by the underlying solver, e.g., floating-point

Many crosschecking queries can be syntactically proved to be equivalent
Crosschecking: Advantages

Many crosschecking queries can be *syntactically* proved to be equivalent
OpenCL Optimizations

- **Parboil:**
  - GPU benchmark suite, originally written in CUDA

- **OP2**
  - Library for applications on unstructured grids

- **Bullet open-source physics library**
  - Popular library used in movie studios and professional game developers
  - Analyzed soft body engine
OpenCL Benchmarks:
Bugs and Mismatches

Several bugs and mismatches:

• 2 mismatches between C and OpenCL code
  • Incorrect FP associativity and distributivity assumptions (CP in Parboil)
• 3 memory errors
  • Buffer overflows (MRI-Q&MRI-FHD in Parboil)
  • Use-after-free: incorrect synchronization between host and kernel code (MRI-Q in Parboil)
  • Uninitialized memory (MRI-FHD in Parboil)
• 1 race condition
  • Missing synchronization barrier (OP2)
• 1 compiler bug
  • NVidia compiler bug (incorrect optimization)
**SIMD Optimizations**

**OpenCV**: popular computer vision library from Intel and Willow Garage

[Corner detection algorithm]
OpenCV Results

• Crosschecked 51 SIMD-optimized versions against their reference scalar implementations
  • Proved the bounded equivalence of 41
  • Found mismatches in 10
• Most mismatches due to tricky FP-related issues:
  • Precision
  • Rounding
  • Associativity
  • Distributivity
  • NaN values
Surprising find: min/max not commutative nor associative!

\[
\text{min}(a, b) = a < b ? a : b
\]

\[a < b \text{ (ordered)} \rightarrow \text{always returns false if one of the operands is NaN}\]

\[
\begin{align*}
\text{min}(\text{NaN}, 5) & = 5 \\
\text{min}(5, \text{NaN}) & = \text{NaN} \\
\text{min}(\text{min}(5, \text{NaN}), 100) & = \text{min}(\text{NaN}, 100) = 100 \\
\text{min}(5, \text{min}(\text{NaN}, 100)) & = \text{min}(5, 100) = 5
\end{align*}
\]
Integrating Crosschecking into Development Process

Semantic mismatches not always errors
  – Underspecified behavior

Two (anecdotal) insights:

1. Provide developers the ability to add “assumptions” eg:
   – Floating-point associativity holds:
     • \( A+(B+C) = (A+B)+C \)
   – Disregard the difference between 0_ and 0_+
     • \( A+0 = A \)

2. All things being equal, developers prefer to keep the behavior of the reference implementation
   – Particularly if we can provide some guarantees
     • bounded equivalence
KLEE: Freely Available as Open-Source

http://klee.llvm.org

- Over 200 subscribers to the klee-dev mailing list
- Extended in many interesting ways by several research groups, in the areas of:
  - wireless sensor networks
  - schedule memoization in multithreaded code
  - automated debugging
  - exploit generation
  - online gaming, etc.