Valgrind
A Framework for Heavyweight Dynamic Binary Instrumentation

Nicholas Nethercote – National ICT Australia
Julian Seward – OpenWorks LLP
FAQ #1

• How do you pronounce “Valgrind”?

• “Val-grinned”, not “Val-grined”

• Don’t feel bad: almost everyone gets it wrong at first
DBA tools

• Program analysis tools are useful
  – Bug detectors
  – Profilers
  – Visualizers

• Dynamic binary analysis (DBA) tools
  – Analyse a program’s machine code at run-time
  – Augment original code with analysis code
Building DBA tools

• Dynamic binary instrumentation (DBI)
  – Add analysis code to the original machine code at run-time
  – No preparation, 100% coverage

• DBI frameworks
  – Pin, DynamoRIO, Valgrind, etc.
Prior work

<table>
<thead>
<tr>
<th>Well-studied</th>
<th>Not well-studied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework performance</td>
<td>Instrumentation capabilities</td>
</tr>
<tr>
<td>Simple tools</td>
<td>Complex tools</td>
</tr>
</tbody>
</table>

• Potential of DBI has not been fully exploited
  – Tools get less attention than frameworks
  – Complex tools are more interesting than simple tools
Shadow value tools
Shadow value tools (I)

- Shadow every value with another value that describes it
  - Tool stores and propagates shadow values in parallel

<table>
<thead>
<tr>
<th>Tool(s)</th>
<th>Shadow values help find...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcheck</td>
<td>Uses of undefined values</td>
</tr>
<tr>
<td>Annelid</td>
<td>Array bounds violations</td>
</tr>
<tr>
<td>Hobbes</td>
<td>Run-time type errors</td>
</tr>
<tr>
<td>TaintCheck, LIFT, TaintTrace</td>
<td>Uses of untrusted values</td>
</tr>
<tr>
<td>“Secret tracker”</td>
<td>Leaked secrets</td>
</tr>
<tr>
<td>DynCompB</td>
<td>Invariants</td>
</tr>
<tr>
<td>Redux</td>
<td>Dynamic dataflow graphs</td>
</tr>
</tbody>
</table>

- Leaked secrets
- Secret tracker
- Array bounds violations
- Run-time type errors
- Uses of untrusted values
- Leaked secrets
- Invariants
- Dynamic dataflow graphs
Memcheck

• Shadow values: defined or undefined

<table>
<thead>
<tr>
<th>Original operation</th>
<th>Shadow operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>int * p = malloc(4)</td>
<td>sh(p) = undefined</td>
</tr>
<tr>
<td>R1 = 0x12345678</td>
<td>sh(R1) = defined</td>
</tr>
<tr>
<td>R1 = R2</td>
<td>sh(R1) = sh(R2)</td>
</tr>
<tr>
<td>R1 = R2 + R3</td>
<td>sh(R1) = add_{sh}(R2, R3)</td>
</tr>
<tr>
<td>if R1==0 then goto L</td>
<td>complain if sh(R1) is undefined</td>
</tr>
</tbody>
</table>

• 30 undefined value bugs found in OpenOffice
All shadow value tools work in the same basic way

Shadow value tools are **heavyweight** tools
- Tool’s data + ops are as complex as the original programs’s

Shadow value tools are hard to implement
- Multiplex real and shadow registers onto register file
- Squeeze real and shadow memory into address space
- Instrument most instructions and system calls
Valgrind basics
Valgrind

• Software
  – Free software (GPL)
  – {x86, x86-64, PPC}/Linux, PPC/AIX

• Users
  – Development: Firefox, OpenOffice, KDE, GNOME, MySQL, Perl, Python, PHP, Samba, RenderMan, Unreal Tournament, NASA, CERN
  – Research: Cambridge, MIT, Berkeley, CMU, Cornell, UNM, ANU, Melbourne, TU Muenchen, TU Graz

• Design
  – Heavyweight tools are well supported
  – Lightweight tools are slow
Two unusual features of Valgrind
#1: Code representation

**D&R**
Disassemble-and-resynthesize (Valgrind)

- $\text{asm}_{\text{in}}$ → disassemble → IR
- IR → resynthesize → $\text{asm}_{\text{out}}$

**C&A**
Copy-and-annotate

- $\text{asm}_{\text{in}}$ → annotate → descriptions
- descriptions → instrument → analysis code
- analysis code → interleave → $\text{asm}_{\text{out}}$
- $\text{asm}_{\text{out}}$ → copy → $\text{asm}_{\text{in}}$
Pros and cons of D&R

• Cons: Lightweight tools
  – Framework design and implementation effort
  – Code translation cost, code quality

• Pros: Heavyweight tools
  – Analysis code as expressive as original code
  – Tight interleaving of original code and analysis code
  – Obvious when things go wrong!

D&R
  bad IR → wrong behaviour
  C&A
  bad descriptions → correct behaviour → wrong analysis
Other IR features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>First-class shadow registers</td>
<td>As expressive as normal registers</td>
</tr>
<tr>
<td>Typed, SSA</td>
<td>Catches instrumentation errors</td>
</tr>
<tr>
<td>RISC-like</td>
<td>Fewer cases to handle</td>
</tr>
<tr>
<td>Infinitely many temporaries</td>
<td>Never have to find a spare register</td>
</tr>
</tbody>
</table>

- Writing complex inline analysis code is easy
#2: Thread serialisation

• Shadow memory: memory accesses no longer atomic
  – Uni-processors: thread switches may intervene
  – Multi-processors: real/shadow accesses may be reordered

• Simple solution: serialise thread execution!
  – Tools can ignore the issue
  – Great for uni-processors, slow for multi-processors…
Performance
## SPEC2000 Performance

<table>
<thead>
<tr>
<th>Tool</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valgrind, no-instrumentation</td>
<td>4.3x</td>
</tr>
<tr>
<td>Pin/DynRIO, no-instrumentation</td>
<td>~1.5x</td>
</tr>
<tr>
<td>Memcheck</td>
<td>22.1x (7--58x)</td>
</tr>
<tr>
<td>Most other shadow value tools</td>
<td>10--180x</td>
</tr>
<tr>
<td>LIFT</td>
<td>3.6x (*)</td>
</tr>
</tbody>
</table>

(* ) LIFT limitations:
- No FP or SIMD programs
- No multi-threaded programs
- 32-bit x86 code on 64-bit x86 machines only
Post-performance

• Only Valgrind allows robust shadow value tools
  – All robust ones built with Valgrind or from scratch

• Perception: “Valgrind is slow”
  – Too simplistic
  – Beware apples-to-oranges comparisons
  – Different frameworks have different strengths
Future of DBI
The future

- Interesting tools!
  - Memcheck changed many C/C++ programmer’s lives
  - Tools don’t arise in a vacuum

- What do you want to know about program execution?
  - Think big!
  - Don’t worry about being practical at first
If you remember nothing else...
Take-home messages

- Heavyweight tools are interesting
- Each DBI framework has its pros and cons
- Valgrind supports heavyweight tools well

www.valgrind.org
(Extra slides)
The past: performance

- Influenced by Dynamo: dynamic binary optimizer

- Everyone in research focuses on performance
  - No PLDI paper ever got rejected for focusing on performance
    
    “The *subjective* issues are important — ease of use and robustness, but performance is the item which would be most interesting for the audience.” (my italics)

- Slow tools are ok, if sufficiently useful
Shadow value requirements

• Requirements:
  – (1) Shadow all state
  – (2) Instrument operations that involve state
  – (3) Produce extra output without disturbing execution
Robustness

• Q. How many programs can Valgrind run?
  – A. A lot

• Valgrind is robust, Valgrind tools can be

• SPEC2000 is not a good stress test!
  – Reviewer: “If the authors want to claim that their tool is to be used in real projects, then they would need to evaluate their tools using the reference inputs for the SPEC CPU2K benchmarks.”