Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation

Nicholas Nethercote
National ICT Australia, Melbourne, Australia

Julian Seward
OpenWorks LLP, Cambridge, UK

By: Dan McCrevan & Daniel Ackermans
Summary
Forecast

- Shadow values
- Valgrind’s JIT
- Why Valgrind is different
- Lightweight vs heavyweight tools
Background

- Dynamic Binary Analysis (DBA)
- Dynamic Binary Instrumentation (DBI)
- Unexploited potential of DBI
- Frameworks provide a simple base system
- Comparisons with Pin and DynamoRIO
How Valgrind Works

- Shadow Values
- Architecture
- Valgrind’s JIT
- Handling special cases
Shadow Values

- Tools that shadow every value with another value that describes it

- Uses:
  - Tracking which bit values are undefined
  - Tracking tainted byte values
  - Type checking
  - Checking if secret information is being exposed
Shadow Value Requirements

- Shadow registers
- Shadow Memory
- Instrument Read/Write instructions
- Instrument Read/Write System Calls
- Instrument start up allocations
- Instrument system allocation/deallocation
- Instrument stack allocation/deallocation
- Instrument heap allocation/deallocation
- Provide extra output in a side channel
Architecture

- Valgrind Core + Tool plugins (Memcheck, TaintCheck, etc...)
- Valgrind runs on the host machine’s CPU
- The client program will run under Valgrind’s control in a simulated CPU with simulated registers.
  - Shadow registers are shadows of the simulated registers
Valgrind’s JIT

● Disassemble and re-synthesize
● Translates a block of code at a time
● Architecture neutral, was originally designed for x86
● Runs through an 8 phase D&R process
8 Phase Process

1. Disassemble: Client Machine code -> Tree IR
2. Optimization: Tree IR -> Flat IR
3. Instrumentation: Flat IR -> Flat IR
4. Optimization: Flat IR -> Flat IR
5. Tree Building: Flat IR -> Tree IR
6. Instruction selection: Tree IR -> instruction list
7. Register allocation: IL -> IL
8. Assemble: IL -> Machine code
Handling special cases

- System calls handled by massive Event System
- Threads are serialized
- Signals are intercepted
- Self modifying code
Our Critique
Limitations

- Valgrind not ideal for lightweight analysis (DBA)
- Only usable on UNIX machines
- Executes client executables serially
- Uses a FIFO cache instead of LRU
- Valgrind’s JIT doesn’t implement chaining
Future work

Three main areas:

- Avoid serial thread execution
- Improve performance to allow for lightweight tools
- Further development of heavyweight DBI tools
Valgrind is a standard C/C++ development tool on Linux

Used in numerous major software projects

Memcheck’s popularity
Overall thoughts

- Interesting approach with the JIT
- More research should go towards this approach
- Would like to see more experimental builds
Thoughts (Cnt’d)

- Questions?

- Thank you Professor Milanova