CMSC 714
Lecture 16
Valgrind and DynInst

Alan Sussman
Notes

• Midterm exam scheduled for Tuesday, Nov. 13
  • sample exam questions posted
• Research project interim report due Nov. 9
Valgrind

• Framework for building dynamic binary analysis tools
  • works on program binaries
  • instrumentation inserted before the program runs
  • provides basic services that a tool writer can use to perform dynamic analyses
  • basic mechanism is *shadow values*

• Shadow values – heavyweight instrumentation
  • basic idea is to maintain a copy of all program state for an analysis tool to use (and tool can add more state needed for its analysis)
  • 9 requirements, 3 classes
    • shadow state – registers and memory
    • read/write operations – instrument instructions (loads and stores) and system calls – arguments and return values to/from registers/memory, and via pointers
    • allocation/deallocation operations – start-up (registers, static data), system calls (*brk, mmap*), stack pointer movement (function call/return), heap (esp. bookkeeping data)
  • transparent execution, but extra output – only effect on instrumented program is extra side-channel output
Valgrind

• **Tool-specific code plugs into Valgrind core**
  • to instrument code fragments that the core passes to it

• **Dynamic binary recompilation**
  • a tool loads client program, recompiles it a block at a time as the client program executes within Valgrind
  • core disassembles code block into IR, then tool plug-in instruments it, then core converts IR back to machine code to execute
    • can deal with dlls, shared libraries, and dynamically generated code – only problem is self-modifying code
    • disassemble/resynthesize (D&R), vs. copy/annotate (C&A) – claim is that D&R better for heavyweight analyses
  • key issue, and reason for difficulty of implementation, is having the tool/core sharing memory with the (instrumented) client program

• Events system used to inform tools about system call activities not directly visible from IR
  • i.e. what state gets changed in the system call

• One big problem is that thread execution is serialized, to keep updates to main and shadow memory consistent always
  • not clear how to fix this and allow concurrent thread execution

• Tool performance (e.g., Memcheck) similar to that of other equivalent tools
DynInst

• C++ class library for binary static and dynamic instrumentation
  • lightweight infrastructure for building dynamic analysis tools
  • differs from earlier instrumentation tools because can work on executing program, and uses machine independent description of inserted code

• Insert *snippets* into one or more client processes
  • at instrumentation *points*
  • *mutator* process inserts snippets into the application program, which was linked with the Dyninst runtime library, either before or at runtime

• Implementation for runtime patching uses similar OS services as a debugger, for controlling activity of another process
  • control process execution
  • read/write address space
DynInst

• Generate code from snippet calls into machine language of host machine in the mutator, then copy into space allocated in application address space
  • use trampoline code – base tramp with pointers to pre and post code surrounding one relocated instruction from the point of insertion
  • mini-tramp for pre or post code snippet, to save/restore registers and set up arguments for snippet function code
    • multiple snippets can be chained at one point

• Conditional breakpoint example shows power of the method, and how it can reduce execution cost for expensive operations by directly inserting code into the application at runtime