Instrumenting Executables for Dynamic Analysis

Jeff Perkins and Michael Ernst

MIT CSAIL

Compiled Instrumentation Approaches

- Source to Source
- Binary
- Other
  - Scripting a debugger
  - Linking with modified libraries

Source to Source

- Easy to create instrumentation
- Easy to debug
- Takes advantage of compiler optimizations
- Portable to different architectures
- Languages (such as C++) can be very complex. It is difficult to handle all constructs correctly.
- System libraries can’t be instrumented
- Difficult for users -- multiple source and object files.

Binary instrumentation

- Instructions are simple
- Portable to different languages
- Libraries can be instrumented
- Easier for users
- Tied to machine architecture
- Instrumentation is tedious to produce (assemby or intermediate language)

Memory Safety

- Analysis needs to access variables
- Variables and pointers may be uninitialized
- Heap space may be deallocated
- Array lengths may not be known
- Analysis tools should never crash program or change its behavior

Memory Safety Solutions

- Smart pointers
  - Safe-C (http://www.cs.wisc.edu/~austin/scc.html)
  - Xu et al. FSE November 2004
- Binary instrumentation
  - Purify (http://www-306.ibm.com/software/awdtools/purifyplus/)
  - Valgrind (http://valgrind.org/)

Fjalar

- Mixed level instrumentation toolkit
  - Binary instrumentation
  - Source level information via DWARF2 debugging information
- Based on Valgrind
- Access information on memory, registers etc
- Valgrind provides bit level information on memory initialization and allocation.
- Code insertion is handled automatically.
- Available soon

Instrumentation Example

- Define a helper function that takes a string name and the address of the basic block:

```
di = unsafeIRDirty_0_N(2/*regparms*/, "enter_function", &enter_function,     mkIRExprVec_2(IRExpr_Const(IRConst_U32((Addr)curFuncPtr->daikon_name)),                IRExpr_Const(IRConst_U32(curren...
```

- Make the stack pointer available to that function

```
di->nFxState = 1;  di->fxState[0].fx     = Ifx_Read;  di->fxState[0].offset = mce->layout->offset_SP;  di->fxState[0].size   = mce->layout->sizeof_SP;
```

- Insert the code into the intermediate representation

```
stmt( mce->bb, IRStmt_Dirty(di) );
```