A Framework for Binary Code Analysis, and Static and Dynamic Patching

Barton P. Miller
University of Wisconsin
bart@cs.wisc.edu
Motivation

- Binary code analysis is a basic tool of security analysts, application developers, system designers and tool developers.

- Existing binary analysis tools have significant limitations.

- We are designing and building a new foundation to support such analysis.
  - Multi-platform
  - Open architecture
  - Extensible
  - Open source
  - Testable
  - Suitable for batch processing
  - Accurate
  - Efficient
Why Binary Code?

Access to the source code often is not possible:
- Proprietary software packages.
- Stripped executables.
- Proprietary libraries: communication (MPI, PVM), linear algebra (NGA), database query (SQL libraries).

Binary code is the only authoritative version of the program.
- Changes occurring in the compile, optimize and link steps can create non-trivial semantic differences from the source and binary.

Worms and viruses are rarely provided with source code.
Our Starting Point: Dyninst

- A machine-independent library for machine level code patching.
  - Functions for binary code analysis
  - Functions for binary code patching

- Clean abstractions to encapsulate the tool complexity.

- Originally designed as part of the Paradyn performance profiling tool, but now widely used in many areas, including cyber-security.
Dynamic Instrumentation

- Does not require recompiling or relinking
  - Saves time: compile and link times are significant in real systems.
  - Can instrument without the source code (e.g., proprietary libraries).
  - Can instrument without linking (relinking is not always possible).

- Instrument optimized code.
Dynamic Instrumentation (con’d)

- Only instrument what you need, when you need
  - No hidden cost of latent instrumentation.
  - Enables “one pass” tools.

- Can instrument running programs (such as Web or database servers)
  - Production systems.
  - Embedded systems.
  - Systems with complex start-up procedures.
The Basic Mechanism

Application Program

Function foo

Trampoline

Instrumentation

Relocated Instruction(s)
The DynInst Interface

- Machine independent representation
- Write-once, analyze/instrument-many (portable)
- Object-based interface to insert new code: Abstract Syntax Trees (AST's)
- Hides most of the complexity in the API
  - Easy to build tools: e.g., an MPI tracer: 250 lines of C++ code.
Basic DynInst Operations

- **Code query routines:**
  - Find control-flow elements: modules, procedures, loops, basic blocks, instructions
    - For functions, find entry, exit, call sites.
    - For loops, find entry, exit, body.
  - Find data elements: variables and parameters
  - Call graph (parent/child) queries
  - Intra-procedural control-flow graph
Basic DynInst Operations

- Code modification routines:
  - Remove Function Call
    - Disable an existing function call in the application
  - Replace Function Call
    - Redirect a function call to a new function
  - Replace Function
    - Redirect all calls (current and future) to a function to a new function.
  - Replace Instruction
    - Code snippet executes instead of specified instruction.
  - Wrap Function
    - Allow the new function to call the replaced one (potentially with all its original parameters).
Basic DynInst Operations

- Process control:
  - Attach/create process
  - Monitor process status changes
  - Callbacks for fork/exec/exit

- Inferior (application processor) operations:
  - Malloc/free
    - Allocate heap space in application process
  - Inferior RPC
    - Asynchronously execute a function in the application.
  - Load module
    - Cause a new .so/.dll to be loaded into the application.
Basic DynInst Operations

- Building AST code sequences:
  - Control structures: if and goto
  - Arithmetic and Boolean expressions
  - Get effective address
  - Generate instruction with calculated address.
  - Get PID/TID operations
  - Read/write registers and global variables
  - Read/write parameters and return value
  - Function call
Machine Independent Code

Abstract Syntax Trees:

SPARC Code

sethi %hi(ctr)
ld [...],%o1
add %o1,%o1,1
st %o1,[...]

IA32 Code

incl ctr

Power Code

cau r3,r0,hi%ctr
l r4,lo%ctr(r3)
addi r4,1(r4)
st r4,lo%ctr(r3)
SymtabAPI

  - Supports ELF, XCoff, PE (Linux, Solaris, AIX, Windows).
- Debug information available in next release: line numbers, local variables, types.
- Unstrip - SymtabAPI demo tool that regenerates a stripped binary’s symbol table
  - Uses code parser to find function entry points
  - Uses SymtabAPI to write new symbol table into binary.
DynStackwalker

- Available soon on all Dyninst platforms.
- Cross-platform API for collecting first and third party stackwalks.
- Callback interface allows users to plug in their own stack walking mechanisms, e.g:
  - Walking through non-standard stack frames created by optimized functions.
  - Use stackwalking debug information provided by another library.
InstructionAPI

- Decodes machine code into abstract instruction representation
- Interface allows straightforward data flow and control flow analysis
  - Query interface is designed for analysis, e.g.:
    - Control flow targets
    - Registers read/written
    - Memory addresses accessed
  - Instructions can be annotated with analysis results
- Provides disassembly interface
  - Pluggable formatters
BinInst Design Goals

- Tool-kit component architecture for binary analysis and editing
- Open source
- Open data structure definitions
- Machine-independent abstract interfaces
- Batch-enabled analyses
- Static and dynamic code patching
- All major analysis products are exportable
- Enhanced testability and accompanying test suites
Static Editing Scenario (Binary Rewriting)

- Binary Code
- Binary Decode and Parsing
- Symbol Table Dump
- Call Graph
- Intra-Proc CFG
- Idiom Signatures
- Disassembly
- Code Queries and Instrumentation Requests
- AST
- Instr Control
- Code Gen

Para

© 2007 Barton P. Miller

Binary Code Analysis and Editing
Interactive Editing Scenario (Static or Dynamic)

Binary Code

Binary Decode and Parsing

Symbol Table Dump

Call Graph

Intra-Proc CFG

Idiom Signatures

Disassembly

Instr Control

Code Gen

© 2007 Barton P. Miller

Binary Code Analysis and Editing
Dynamic Editing Scenario (*Dynamic Instrumentation*)

Binary Decode and Parsing

- Symbol Table Dump
- Call Graph
- Intra-Proc CFG
- Idiom Signatures
- Disassembly

Code Queries and Instrumentation Requests

- AST
- Instr Control
- Code Gen
- Process Control
- Stack Walker

User Process

© 2007 Barton P. Miller