The Deconstruction of Dyninst: Experiences and Future Directions

Drew Bernat, Madhavi Krishnan, Bill Williams, Bart Miller
Paradyn Project
Why components?

Share tools
Build new tools quickly
Share Tools

scalasca

Dyn inst

Open SpeedShop™

Paraver
Dyninst Components

DyninstAPI
Dyninst Component Users

StackwalkerAPI
A Dyninst Component

SymtabAPI
A Dyninst Component

CRAY ATP

HPCToolkit
Rice University
Build New Tools Quickly: Dataflow Analysis

- PowerPC jump tables and return instruction detection
- Malware return address tampering
- Behavior-preserving relocation
Build New Tools Quickly: Binary Rewriter
Build New Tools Quickly: Unstrip

Symbol Table

ParseAPI
A Dyninst Component

SymtabAPI
A Dyninst Component

ROSE compiler

targ8056f50

targ805c3bd0

targ805fd70

g getpid40

01000101...101010101...011010101...101101011

Data
Down The Memory Lane

SymtabAPI – version 1.0
DynStackwalker – coming soon
InstructionAPI – proposed
BinInst – proposed
Dyninst Components Timeline

- Design and Implementation
- Beta Release
- First Release
- Integration into Dyninst

---

**ProcControlAPI**
A Dyninst Component

---

**DataflowAPI**

**PatchAPI**

**ParseAPI**

**ProcControlAPI**

**InstructionAPI**

**StackwalkerAPI**

**SymtabAPI**

---

2006 2007 2008 2009 2010 2011
Componentization: Design Decisions

Define the scope of the component

ParseAPI CFG model
- Block
- InstPoints
- Instrumentability

Dyninst CFG model
- Edge
- Cached register liveness info
- Function
Componentization: Design Decisions

Balance internal and external user requirement
Componentization: Design Decisions

Refine requirements
Componentization: Design Decisions

Create right level of abstractions
Componentization: Design Decisions

Design extensible and adaptable interfaces

Stack frame stepper
- Standard frame
- Debug frame
- Signal frame

ParseAPI
A Dyninst Component
Componentization: Design Decisions

Plan for reintegration

StackwalkerAPI
A Dyninst Component

ProcControlAPI
A Dyninst Component
Ongoing Research
Ongoing Research

• Lightweight, Self-Propelled Instrumentation
  • Wenbin Fang

• Binary Editing
  • Andrew Bernat

• Malware Analysis and Instrumentation
  • Kevin Roundy

• Binary Provenance and Authorship
  • Nate Rosenblum

• Instrumenting Virtualized Environments
  • Emily Jacobson
Lightweight Instrumentation

• Analyze intermittent bugs and fine-grained performance problems
  • Autonomy
  • Little perturbation
  • High level of detail
• Rapid activation
• Ability to analyze black-box systems
  • User level and kernel level
Self-Propelled Instrumentation

User Mutator

Dyn\textsuperscript{inst}

PatchAPI
A Dyninst Component

Snippet

Snippet
How it Works

```c
void foo() {
    bar();
}

void bar() {
    baz();
}
```

Instrumenter.so

Process

Snippet

ProcControlAPI
A Dyninst Component
Binary Instrumentation

PatchAPI
A Dyninst Component

ParseAPI
A Dyninst Component
Binary Editing

- Predicate switching
- Insert error checking and handling
- Dynamic patching
- Code surgery
Malware Analysis and Instrumentation

- Unpacking Code
- Overwriting Code
- Self-Checksumming
- Address Space Sensitive
SR-Dyninst

Parse Reachable Code
Dynamic Code Discovery
Catch Exceptions
Overcome Sensitivity

ParseAPI
A Dyninst Component
PatchAPI
A Dyninst Component
ProcControlAPI
A Dyninst Component
DataflowAPI
A Dyninst Component

Dyninst
CFG of Conficker A
Binary Provenance and Authorship
Provenance System Overview

**TRAINING DATA**

01110101
1010101
01010111
0101001
01101
01110101
1010101
01010111
0101001
01101
01110101
1010101
01010111
0101001
01101
01110101
1010101
01010111
0101001
01101
01110101
1010101
01010111
0101001
01101
01110101
1010101
01010111
0101001
01101

**PROGRAM**

01110101
1010101
01010111
0101001
01101
01110101
1010101
01010111
0101001
01101
01110101
1010101
01010111
0101001
01101
01110101
1010101
01010111
0101001
01101
01110101
1010101
01010111
0101001
01101

**BINARY ANALYSIS TOOL**

ParseAPI
A Dyninst Component

**LEARNING FRAMEWORK**

provenance model
Provenance Evaluation

175 programs \times \quad 2,686 \text{ binaries} \quad 955\text{k functions}

<table>
<thead>
<tr>
<th>Language</th>
<th>Compiler</th>
<th>Optimization</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>.999</td>
<td>.998</td>
<td>.993</td>
<td>.910</td>
</tr>
</tbody>
</table>

Acc.
Instrumenting Virtualized Environments
Status Update
Dyninst 7.0.1

Major new features:

• New platforms for binary rewriter
  • x86 and x86_64 - statically linked binaries
  • ppc32 and BlueGene/P - dynamically linked binaries

• Improvements to parsing speed

• Reductions in memory usage

• Deprecated Solaris and IA64 platforms

• AIX pending due to support difficulties
Component Status Update

- SymtabAPI 7.0.1
  - Speed and space optimizations
- InstructionAPI 7.0.1
  - PowerPC (ppc32, ppc64) platform
  - Full integration with Dyninst
- ParseAPI 7.0.1 - Platform independent API for parsing binaries
  - Control flow graph representation
  - Interprocedural edges (call and return)
  - Built on InstructionAPI and SymtabAPI
  - Full integration with Dyninst
Component Status Update

- **StackwalkerAPI 2.1**
  - Significant reduction in memory usage

- **ProcControlAPI 1.0.1** - Platform independent interface for creating, monitoring and controlling processes
  - High level abstraction for process control, breakpoints and callbacks for process events

- **DynC API 1.0.1** - Instrumentation language for specifying snippets
  - C like instrumentation snippets for easy and more legible mutator
  - Handles creation and destruction of snippet-local variables
Dyninst 8.0

- ProcControl API - Windows and BlueGene
- Stackwalker API - Windows and VxWorks
- Stackwalker & ProcControl integration into Dyninst
- PatchAPI and integration into Dyninst
- SR Dyninst for tamper resistant and obfuscated binaries
- New platforms for binary rewriter
  - Dynamically linked binaries on ppc64 and Windows
  - Statically linked binaries on ppc32 and BlueGene/P
- Dataflow API official release
MRNet 3.0.1

- Support for loading several filters from the same library
- Lightweight MRNet back-end support for non-blocking receives
- CrayXT support for staging files using ALPS tool helper
- Improved build structure that permits configuration for multiple platforms from a single source distribution
- Numerous bug fixes and enhancements
Software and Manuals

• Dyninst 7.0.1, MRNet 3.0.1: available now!
• Downloads:
  

• Dyninst 8.0 – 4th quarter, 2011
• MRNet 3.0.2 – coming soon!
New Environments

• Virtual Machines
  • Whole-system profiling (guest + VMM) using instrumentation
  • VMM-level information to understand how and why an application's performance is affected by the virtualized environment
  • Expand performance profiling in the virtualized environment, where traditional approaches do not work or may not be sufficient

• Mobile environments – VxWorks, ARM
• GPUs
Questions
Unstrip: Semantic Descriptors

• We take a semantic approach
• Record information that is likely to be invariant across multiple versions of the function

<accept>:

mov %ebx, %edx
mov %0x66,%eax
mov $0x5,%ebx
lea 0x4(%esp),%ecx
int $0x80
mov %edx, %ebx
cmp %0xffffffff83,%eax
jae 8048300
ret
mov %esi,%esi

{<socketcall, 5>}
Identifying Functions in a Stripped Binary

For each wrapper function
{
1. Build the semantic descriptor.
2. Search the database for a match (two stages).
3. Add label to symbol table.
}

unstrip: Restoring Function Information to Stripped Binaries
Performance: Capturing Fine-grained behavior

**Introduction to the PatchAPI**

**Dyninst**

(3rd party instrumentation)

- **User Mutator**
  - DyninstAPI
  - PatchAPI
    - find point
    - insert snippet
    - delete snippet

- **Process**
  - void foo () {
    - Snippet
  }
  - void bar () {
    - Snippet
  }
  - void baz () {
    - Snippet
  }

**Self-propelled instrumentation**

(1st party instrumentation)

- **Process**
  - void foo () {
    - Snippet
    - bar()
  }
  - void bar () {
    - Snippet
    - baz()
  }
  - void baz () {
    - Snippet
  }

**Instrumenter.so**

PatchAPI
New Component: PatchAPI

Introduction to the PatchAPI
Dyninst is a toolbox for analysts

- Specifications instrumentation
- Gets callbacks for runtime events
- Builds high-level analysis

Dyninst

Control flow analyzer
Data flow analyzer
Instrumenter

loop, block, function, instruction instrumentation
library injection
function replacement
symbol table reading, writing
machine language parsing
call stack walking
binary rewriting
process control
forward & backward slices
loop analysis

program binary reading, writing
What we could do because of components?

- SymtabAPI & StackwalkerAPI
- DyninstAPI Instrumentor
- ROSE semantics engine
- Tools we developed - quickly
  - Binary rewriter, unstrip
Componentization

• Trade-offs
  • Internal requirements vs. external requirements
  • Early feedback vs. interface stability
  • Development time vs. scope
  • Structured vs. organic

• Lesson learned
  • Keep the project details where they belong
  • Change code incrementally
  • Test new interfaces
Binary rewriter

- Read binary file format from disk
- Parse binary code and build CFG
- Generate code for instrumentation
- Patch code
- Emit new binary file
Binary rewriter

Dyninst

ParseAPI
A Dyninst Component

ProcControlAPI
A Dyninst Component

StackwalkerAPI
A Dyninst Component

The GNU Project Debugger

DataflowAPI
A Dyninst Component

PatchAPI
A Dyninst Component
Componentization: Design decisions

- Define the scope of the component
- Balance internal and external user requirement
- Refine the assumptions
- Create right level of abstractions
- Build from scratch or improve existing code
- Early feedback vs. interface stability
Dyninst and the components

- DynC API
- AST
- Binary
- Symtab API
- Parse API
- Dataflow API
- Stackwalker API
- ProcControl API
- CodeGen API
- Patch API
- Process