New Developments in the Dyninst and MRNet Toolkits

Bill Williams

Paradyn Project

9th Petascale Tools Workshop
Tahoe, CA
August 3, 2015

Paradyn

Dyninst

Dyninst 9.0 Overview

- New features:
  - Memory optimizations
  - Initial ARM64 support
  - Improved TLS support
- Research areas:
  - Improved parsing & dataflow analysis
  - Stack frame modification interface
  - SD-Dyninst integration
- Git-head is near final, official release coming soon

MRNet 5.0 Overview

- LIBI integration
- Verified ARM64 support
- Bug fixes
- Officially released 7/30/15

Symtab memory optimization

- Lazy demangling
- Lazy line information parsing
- Have observed ~75% reduction in Symtab overhead from these changes
- Tradeoff: higher CPU cost at initial startup

Symtab optimization breakdown

Area	Pre-opt. MB	Pre-opt %	Opt. MB	Opt. %
Line info indexes	1600	31%	0	0%
Libdwarf leaks	950	18%	0	0%
String copies	300	6%	0	0%
Demangled names	1000	19%	0	0%
Mangled names	240	5%	240	18%
Exception blocks	280	6%	280	21%
Symbol indexes	150	3%	150	11%
Other	670	13%	670	50%
Total	5190	100%	1340	100%

Per-CU line info

Lazy demangling

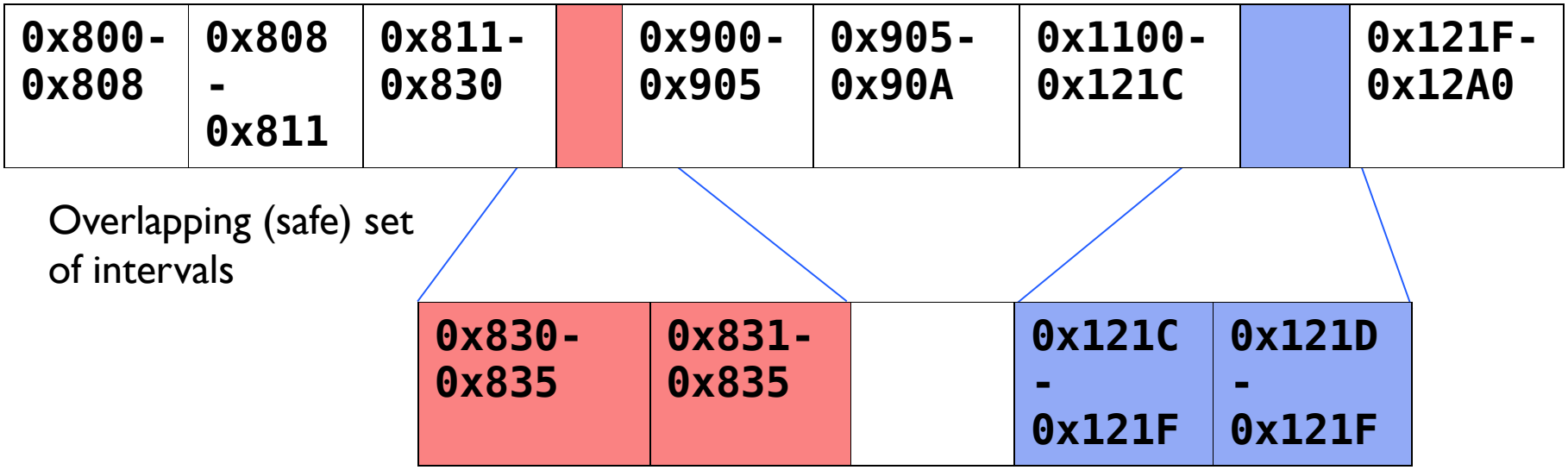
Results obtained from `openFile` and a request for line information at a single address

ParseAPI memory optimization

- Blocks, functions, etc. stored in interval trees
- Can be overlapping
- Overlap is rare
- Two types of interval tree: fast and safe
  - Fast assumes non-overlapping intervals, O(n) space
  - Safe assumes most/all intervals overlap, O(n log n) space

ParseAPI memory optimization

Non-overlapping
(fast) set of intervals



ARM64-enabled components

- **SymtabAPI**
  - Build system support
  - Generally smooth port
- ProcControl
- Stackwalker

ARM64-enabled components

- SymtabAPI
- **ProcControl**
  - Most functionality was easy
  - Kernel bug
  - Lack of ptrace backwards compatibility
- Stackwalker

ARM64-enabled components

- SymtabAPI
- Proccontrol
- **Stackwalker**
  - 3rd party support works
  - 1st party support coming later

ARM64-enabled components

- SymtabAPI
- Proccontrol
- **Stackwalker**
  - ARM stack layout is unusual
  - Calls don't save RA to stack

Normal stack	
Slot	**Contents**
0	RA
1	FP
2…N	Locals

ARM stack	
Slot	**Contents**
0…N-2	Locals
N-1	RA
N	FP

New thread local storage (TLS) features

- ProcControl: read & write TLS variables in a process

- Dyninst: trampguards moved to TLS
  - No hard limits on # of threads
  - Faster instrumentation in cases where trampguards are enabled

Instruction representation challenges

- Maintain accurate map of bytes to opcodes
  - Instruction sets grow & change rapidly
  - Syntax is easy, semantics are harder
- Maintain accurate understanding of operands
  - Register sets grow and change rapidly, too
- Documentation is highly variable
  - Good: standardized XML (ARM)
  - Medium: scrapeable HTML (PPC)
  - Bad: dead tree/PDF (Intel)

Jump table improvements

- Principled slicing-based approach
- Improves performance of instrumented binary
- Handles arbitrary number of table levels

Normal jump table

Source-level construct

```
switch(x)
{
  case 0:
  case 2:
    // …
    break;
  case 3:
    // …
    break;
  default:
    // …
}
```

Table entries

Address	Contents
0x405100	0x401102
0x405104	0x401F00
0x405108	0x401102
0x40510C	0x401107

Binary implementation

```
CMP %RAX, 0x03
JA 0x401F00
JMP *(0x405100+4*
%RAX)
```

Two-level Jump table example

Source-level construct

```
switch(x)
{
  case 0:
    // …
    break;
  case 29:
    // …
    break;
  case 100:
    // …
    break;
  case 169:
    // …
    break;
  default:
    // …
}
```

Binary implementation

```
CMP     0xa9,%EAX
JA      0x41677e
MOVZBL  *(0x416bd4+%EAX)
        ,%ECX
JMP     *(0x416bc0+4*
%ECX)
```

JMP *(0x416bc0 + 4 * *(0x416bd4 + %EAX))

First level table

Address	Contents
0x416bd4	0x0
0x416bd5	0x4
…	…
0x416c7c	0x4
0x416c7d	0x3

Second level table

Address	Contents
0x416bc0	0x4156ac
0x416bc4	0x4157d0
0x416bc8	0x41596a
0x416bcc	0x41599e
0x416bd0	0x41677e

Non-jump table example

Source-level construct

```
switch(i % 8)
{
  case 0:
    x[i]-=y[i];
    ++i;
  case 1:
    x[i]-=y[i];
    ++i;
  // …
  case 7:
    x[i]-=y[i];
    ++i;
}
```

Binary implementation

```
AND     0x7,%EAX
JE      0x80d93c8
LEA     0x80d93c5+9*%EAX
        ,%EAX
JMP     %EAX

80d93c8: //case 0
mov     (%esi),%eax
sbb     (%edx),%eax
mov     %eax,(%edi)
80d93ce: // case 1
mov     0x4(%esi),%eax
sbb     0x4(%edx),%eax
mov     %eax,0x4(%edi)
// …
80d9404: // case 7
mov     0x1c(%esi),%eax
sbb     0x1c(%edx),%eax
mov     %eax,0x1c(%edi)
```

Jump table principles

- Tables are contiguous
- Tables depend on a single bounded input value
- Tables live in read-only data or code

Jump table results

- Glibc: ~30% decrease in uninstrumentable functions, 20% increase in parse overhead
- Newly instrumentable libc functions include:
  - strncmp
  - strcmp
  - memcmp
  - memset
- Normal binaries: ~5% increase in parse overhead, 7% decrease in uninstrumentable functions

Gap parsing improvements

- Machine learning based model updated for current compilers
- …and finally integrated into Dyninst
- No longer need to apply compiler-specific models

Gap parsing results

Version	Platform	Avg. Precision	Avg. Recall
Dyninst 8.2.1	64-bit x86	98.1%	37.4%
Dyninst 8.2.1	32-bit x86	95.6%	53.9%
Dyninst 9.0	64-bit x86	94.7%	83.2%
Dyninst 9.0	32-bit x86	97.1%	93.8%

Test binaries are from binutils, coreutils, and findutils, built with icc and gcc, at –O0 through –O3.

Stack frame modifications

- Can add, remove, swap, randomize space on stack
- Operates at function scope
- Mostly a security-oriented feature
- Important prerequisite: understand the stack frame with stack analysis

Stack analysis improvements

- Stack analysis: for each register, what stack location does it point to?
  - TOP: does not point to the stack
  - Numeric height: relative to SP at function entry
  - BOTTOM: may point to anywhere on the stack
- More instructions analyzed precisely
  - Added support for sign extend, zero extend, more general math (including more LEA math)
  - Improved stack modification from covering 30% of SPEC 2006 functions to 60% at –O2

SD-Dyninst integration

- Maintain instrumentation capability through:
  - Dynamically generated code
  - Obfuscated control flow
- Designed for malware
- "Any sufficiently advanced optimizer is indistinguishable from malware"
- Can capture control flow through exception handlers

Slicing improvements

- Better handling of control flow cycles
  - Data flow around a cycle may involve different instructions on each iteration
  - Need to distinguish between visited instructions and visited assignments
- Many bug fixes, improving slice precision and accuracy

Range-based interfaces

- Lesson from Symtab optimizations: exposing containers is inflexible
  - Whole container must exist, even if user wants one element
  - Hard to change types or relocate data
- Instead, prefer ranges
  - Begin/end interfaces like STL containers
  - Typedefs for readability
  - Key to enabling, e.g., lazy demangling

LIBI

- Single interface for launching processes
- Does not replace RSH or XT launch frameworks, but augments them
- Contact Dorian Arnold for details

MRNet ARM64 support

- MRNet now supports ARM64/Linux
- Full set of features should work
- Has not been tested at large scale
- Uneventful port

MRNet bugs fixed

- Build system fixes to support ARM
- Low port numbers (<10000) now work
- Better XPLAT_RSH_ARGS support
- Filter load failures are reported to front end

Ongoing and future work

- Windows binary rewriter
- Exception table rewriting
- Further memory and CPU improvements
- Completing ARM64 port
- New instruction foundation for x86