A Hybrid Approach to Memory Safety of C Programs

Chenyi Zhang

Oracle Labs Australia
24 November 2014

Outline

1. Introduction
2. Prototype Architecture
3. Performance
4. Future Work

Introduction

```
char *ptr = malloc(6);
strcpy(ptr, “Hello!”);          // overflows heap allocation
```

```
float f = 3.14;
char *ptr = *(char **)&f;     // weak type control
ptr[0] = ‘c’;                 // illegal write to memory
```

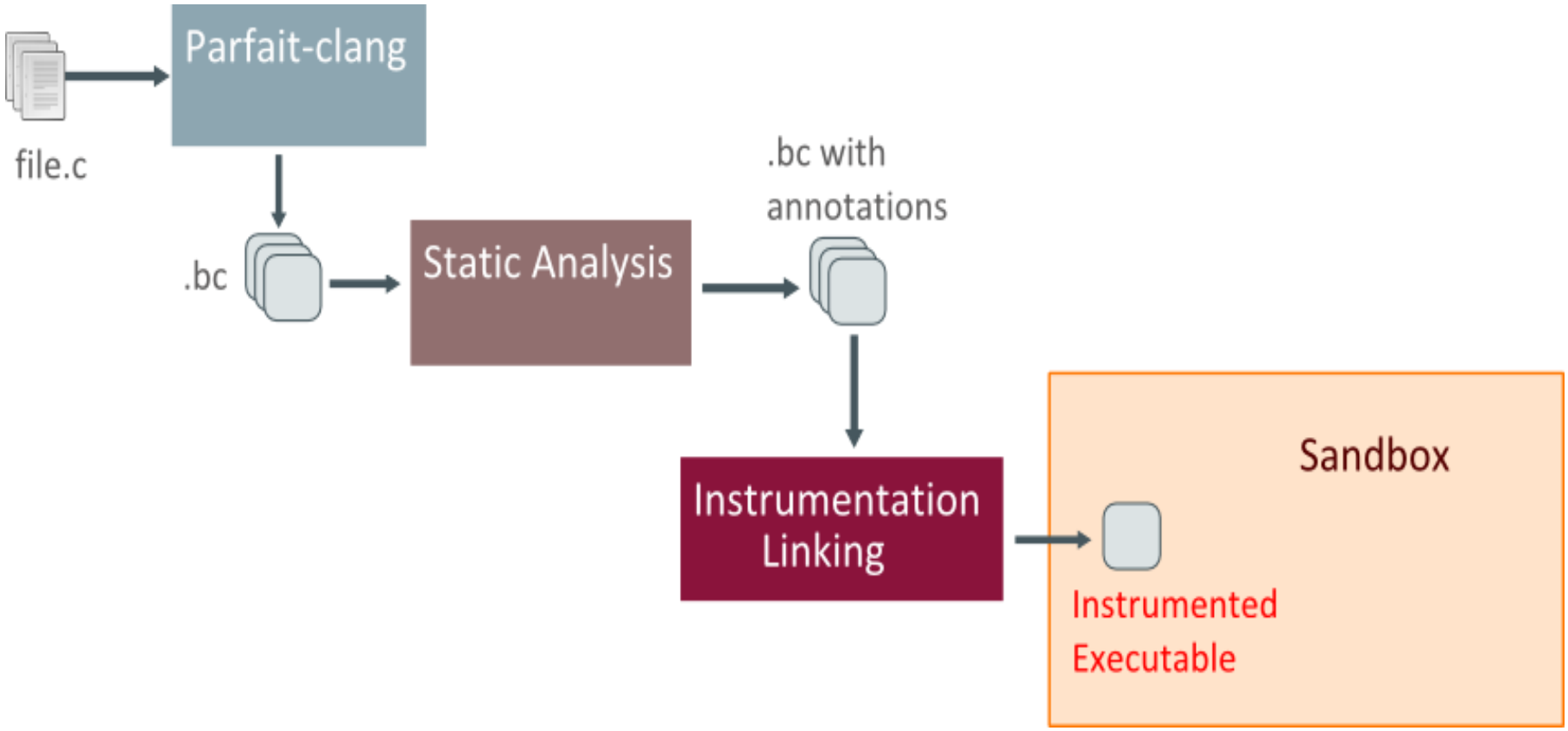
Introduction

```
char *ptr = malloc(5);
char *ptr2 = ptr;
…
free(ptr2);       // heap allocation “malloc(5)” is deallocated
…
ptr[0] = 'c';
```

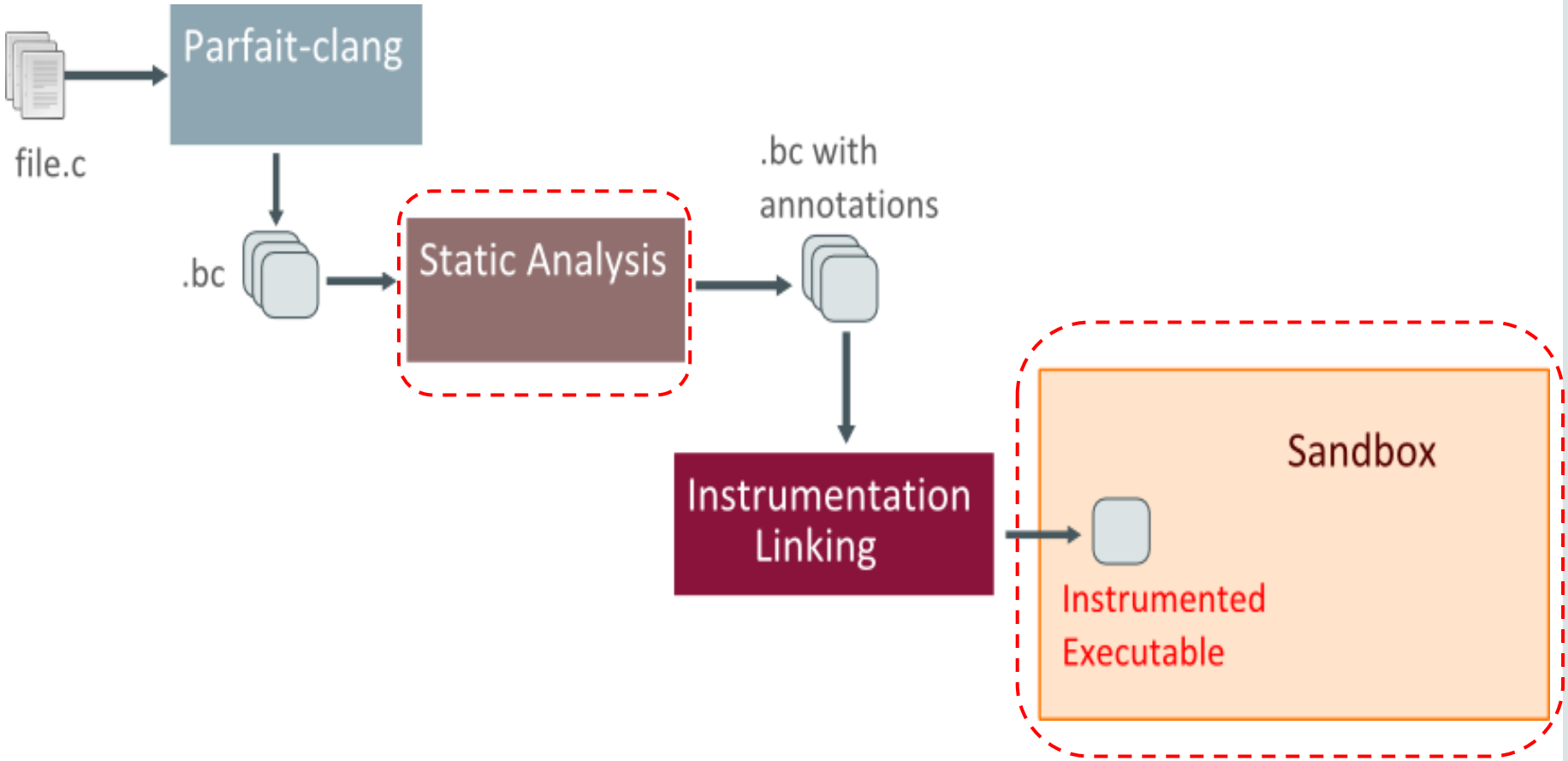
Introduction

- Types of safety violations in C
  - Use after free (Stale pointers, temporal violation)
  - Buffer overflow (spatial violation)
  - Illegal access to system memory (spatial violation)
- Not including
  - Memory leak
  - Integer overflow
  - Use of unitialized memory

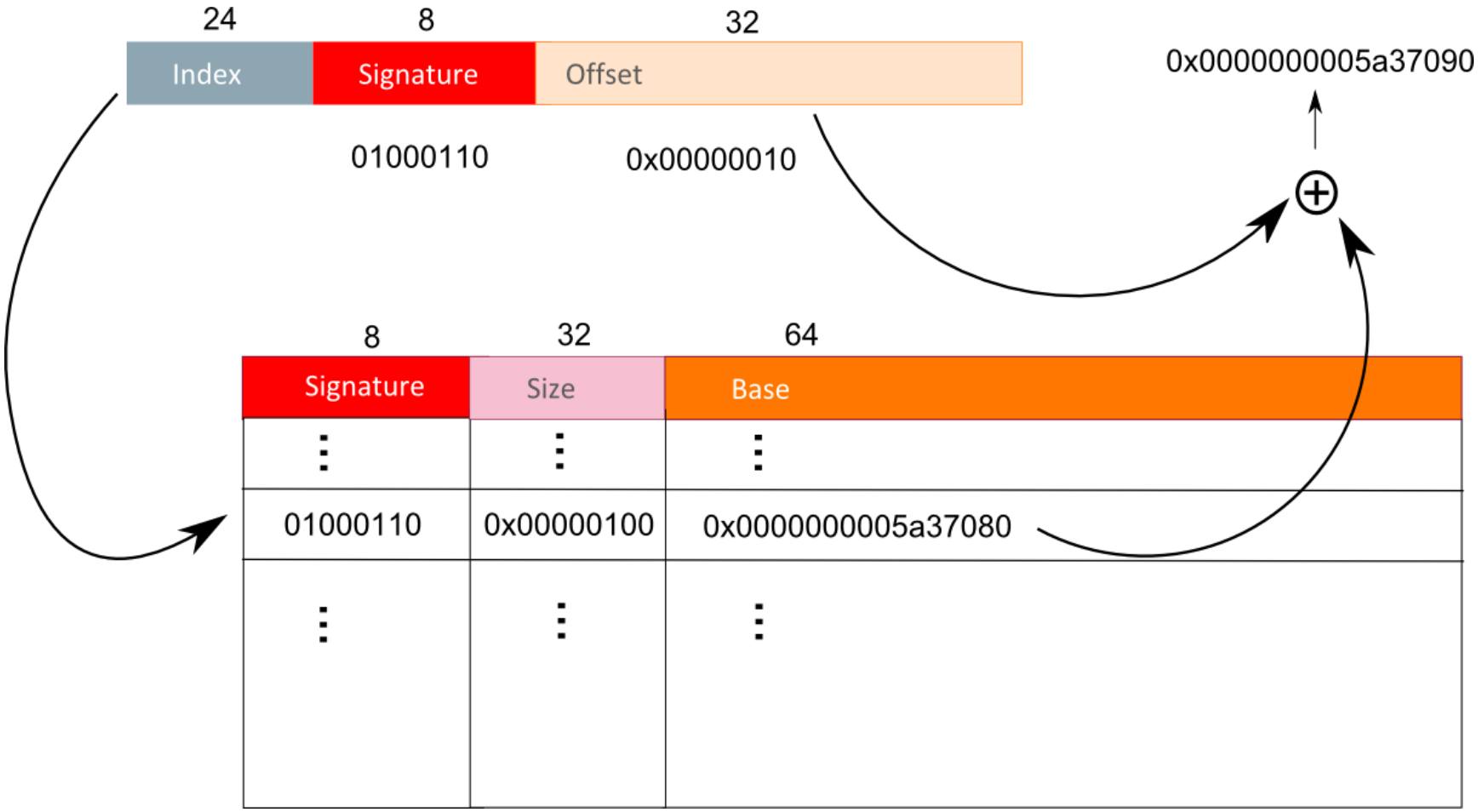
Prototype Architecture



Prototype Architecture



Long Pointer

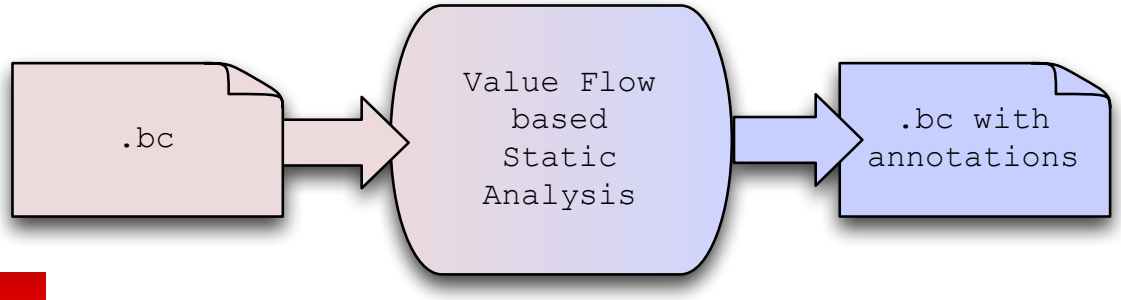


Sandbox API functions

- Allocations and frees: update metadata (swizzle)
- Pointer uses as dereference and as data: check against metadata (unswizzle)
- Pointer stores: escape pointer from Sandbox while retaining associated metadata (escape)
- Pointer loads: sanitize pointer for Sandbox by reestablishing metadata association (sanitize)

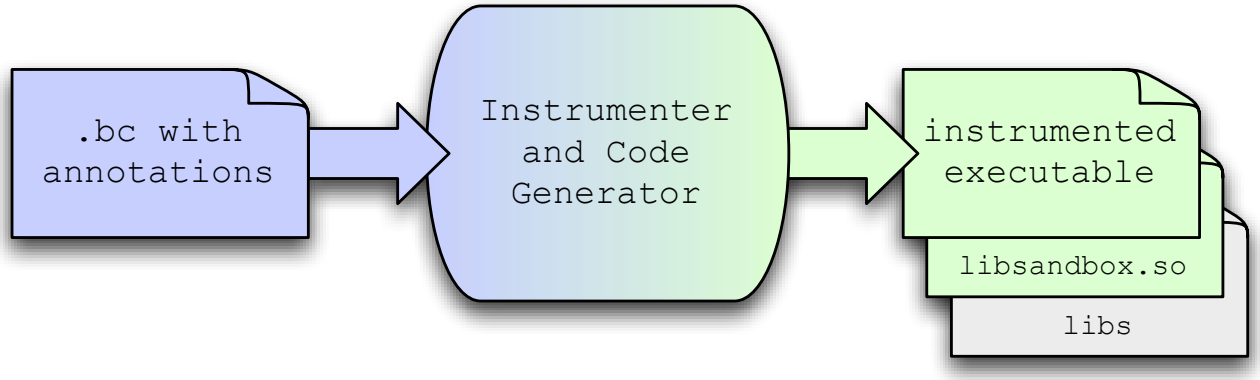
Prototype Architecture: Static Analysis

- LLVM based “link-time,” inter-procedural, flow-insensitive value-flow analysis of interesting values:
  - Memory allocations & free
  - Memory accesses, i.e. pointer dereferences (unswizzle)
  - Other uses of pointers as data (escape)
  - Pointer stores (escape) and loads (sanitize)
- Additional passes to remove annotations
  - Do not check safe accesses (removing swizzle and unswizzle)
  - Store and load long pointers (removing escape and sanitize)



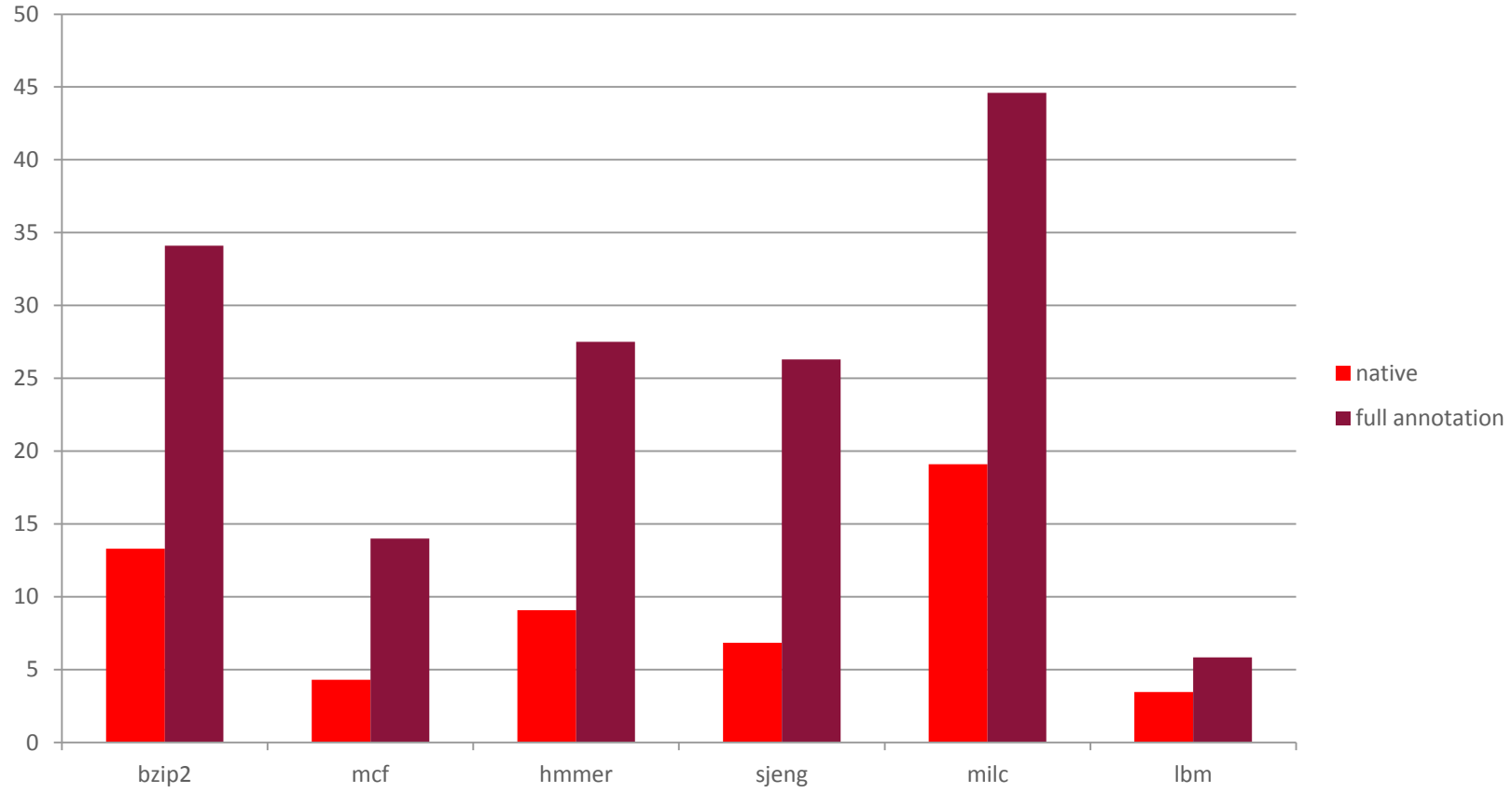
Prototype Architecture: Instrumentation

- Instrumentation
  - Inject Sandbox API calls based on annotations from static analysis
  - Gather global variables information and hijack the main() function



Sandbox Performance

71% to 280% overhead



Literature

- Fat pointers (PLDI'94)
  - Source level transformation
  - typedef {

        <type> *value;

        <type> *base;

        unsigned size;

        int capability;

    } SafePtr<type>;
  - Need to update pointers with realloc()
  - Runtime overhead 130% to 540%

Literature

- Softbound/CETS (PLDI'09, ISMM'10)
  - Compiler time instrumentation
  - Disjoint pointer meta-data value
  - No problem with realloc() due to SSA
  - Cumbersome at callsites
  - 116% total runtime overhead reported
- Intel Memory Protection Extension (MPX) – released 2013
  - Hardware implementation of Softbound
  - bound registers and new instructions
  - Low overhead (~10%)

Future Challenges

- Multi-thread support
  - Atomic load and store of long pointer values
  - Thread-safe access on shared meta-data entries

- Closed world
  - Libc included in our analysis
  - Remove most sandbox API calls to escape and sanitize long pointers