Implementing SoftBound on Binary Executables

Ruturaj Kiran Vaidya

Graduate Advising committee:

Dr. Prasad Kulkarni (Adviser)
Dr. Drew Davidson
Dr. Alex Bardas

Contents

Contents

Goal / Contribution

- Provide spatial safety
- By implementing SoftBound technique on binaries
- With no source code support

Contents

Background

- Memory safety
- The SoftBound Technique
- Static analysis and Dynamic Instrumentation

Background - Memory Safety

- What is memory safety?

Memory safety is the property which ensures all the memory accesses are valid, i.e. they stick to the semantics defined by the language.

- Spatial errors

  Eg. Buffer overrun

- Temporal errors

  Eg. Use after free

Background - Memory Safety

Spatial memory safety violation

```
char *p = malloc(10);
for (int i = 0; i < 15; ++i)
{
  *(p + i) = i + 65;
  printf("%c\n", p[i]);
}
```

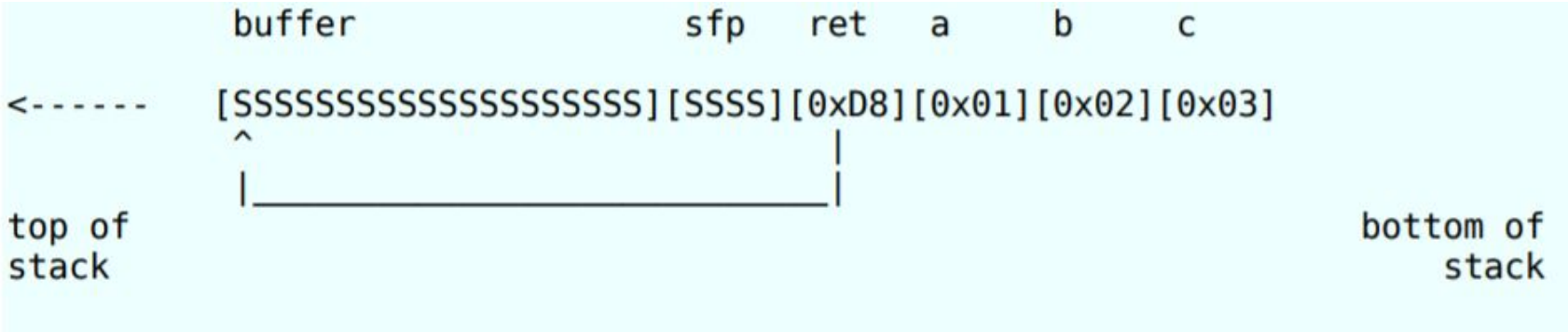
Background - Memory Safety

Temporal memory safety violation

```
char *p = malloc(10);
free(p);
for (int i = 0; i < 10; ++i)
{
  *(p + i) = i + 65;
  printf("%c\n", p[i]);
}
```

Background - Memory Safety

Leveraging Spatial memory safety violation



Ref: One, Aleph. "Smashing the stack for fun and profit." *Phrack magazine* 7, no. 49 (1996): 14-16.

Background - The SoftBound Technique

- What is SoftBound?
- Compiler transformation approach
- Complete Safety against spatial errors
- Stores base and bound metadata per pointer in a disjoint data structure

Goal of this work

- Provide spatial safety
- By implementing SoftBound technique on binaries
- Using static analysis and dynamic instrumentation frameworks

Background - Static Binary Analysis

Automatically analyzing binary properties

- Program testing, verification, Reverse engineering, etc.

Background - Static Binary Analysis

Tools: Ghidra, Radare2, Binary Ninja, IDA pro, etc.

- We use Ghidra
- GUI or CLI
- disassembly, decompilation, scripting, etc.
- API for scripting

Background - Dynamic Instrumentation

Analyse the binary and add instrumentation at run time

- Taint analysis, Control flow modifications, leak detection, etc.

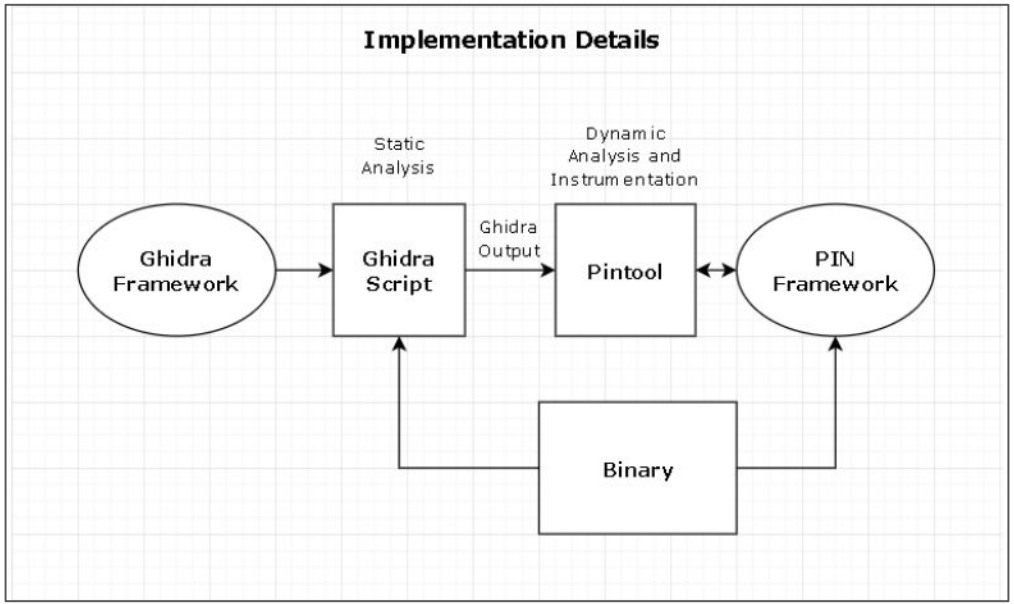
Background - Dynamic Instrumentation

Tools: Intel PIN, valgrind, DynamoRIO, etc.

- We use Intel PIN
- JIT based instrumentation
- Provides extensive API

Contents

Implementation Details



Implementation Details - Ghidra Script

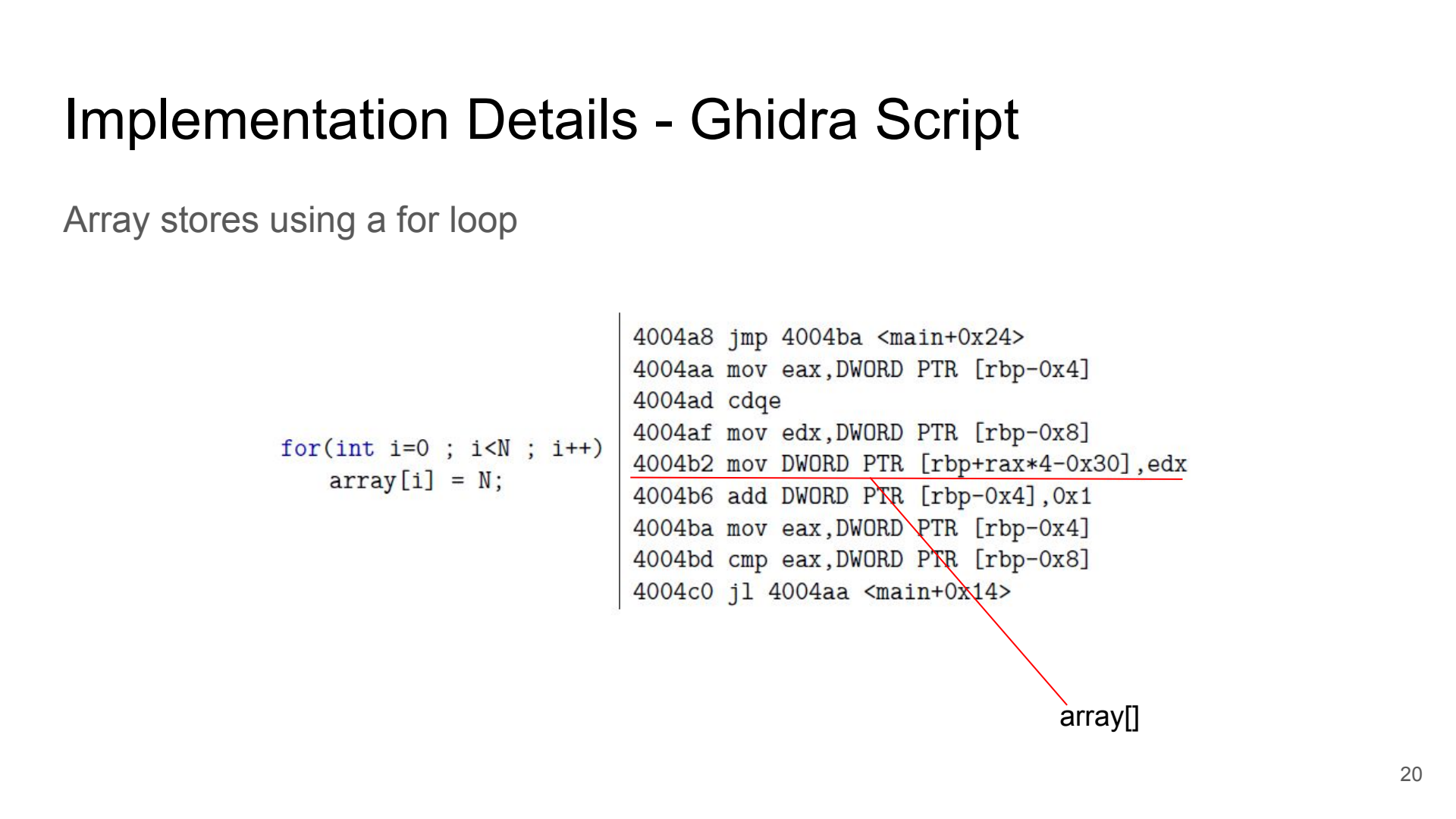
- Python 2.7 and using ghidra API
- One condition - Binary must be compiled using gcc -g flag
- Fetches variable information and their instruction access information

Implementation Details - Ghidra Script

Array stores using a for loop

```
for(int i=0 ; i<N ; i++)
    array[i] = N;
```

```
4004a8 jmp 4004ba <main+0x24>
4004aa mov eax,DWORD PTR [rbp-0x4]
4004ad cdqe
4004af mov edx,DWORD PTR [rbp-0x8]
4004b2 mov DWORD PTR [rbp+rax*4-0x30],edx
4004b6 add DWORD PTR [rbp-0x4],0x1
4004ba mov eax,DWORD PTR [rbp-0x4]
4004bd cmp eax,DWORD PTR [rbp-0x8]
4004c0 jl 4004aa <main+0x14>
```



Implementation Details - Ghidra Script

Invalid Pointer Access

```
int main()
{
  int b[5];
  b[0] = 1;
  int *ptr = b;
  int c = *(ptr + 10);
  return 0;
}
```

```
400496 push rbp
400497 mov rbp,rsp
40049a mov DWORD PTR [rbp-0x20],0x1 ←
4004a1 lea rax,[rbp-0x20]
4004a5 mov QWORD PTR [rbp-0x8],rax ←
4004a9 mov rax,QWORD PTR [rbp-0x8]
4004ad mov eax,DWORD PTR [rax+0x28] ←
4004b0 mov DWORD PTR [rbp-0xc],eax
4004b3 mov eax,0x0
4004b8 pop rbp
4004b9 ret
```

Implementation Details - Ghidra Script

Ghidra Output

```
int main()
{
  int b[5];
  b[0] = 1;
  int *ptr = b;
  int c = *(ptr + 10);
  return 0;
}
```

```
400496 push rbp
400497 mov rbp,rsp
40049a mov DWORD PTR [rbp-0x20],0x1
4004a1 lea rax,[rbp-0x20]
4004a5 mov QWORD PTR [rbp-0x8],rax
4004a9 mov rax,QWORD PTR [rbp-0x8]
4004ad mov eax,DWORD PTR [rax+0x28]
4004b0 mov DWORD PTR [rbp-0xc],eax
4004b3 mov eax,0x0
4004b8 pop rbp
4004b9 ret
```

Code

```
1          // Number of functions
main       // Function name

addresses  // Function addresses and instruction owner information
40049a main_b
4004a1 main_b
4004b0 main_c
4004a5 main_ptr
4004ad main_ptr

locals     // Function local variables
-32 array main_b 20
-12 scalar main_c 4
-8 pointer main_ptr 8

namespace  // Static variables local to the function

.global    // Global variables
6295544 pointer .global_PTR___gmon_start___00600ff8 8
```

Generated Output

Implementation Details - Ghidra Script

Edge cases

```
int main()
{
  {
    int c;
    c = 8;
  }
  int c = 5;
  return 0;
}
```

Implementation Details - Pintool

- Written in C++ using Pin API
- Takes binary and information generated by ghidra as input
- Fetches and Stores the variable metadata and instruction access information
- Bounds information per variable is stored in a global structure
- Detection mechanism
- Add checks for every load and store

Implementation Details - Pintool

Invalid Pointer Access

```
int main()
{
  int b[5];
  b[0] = 1;
  int *ptr = b;
  int c = *(ptr + 10);
  return 0;
}
```

```
400496 push rbp
400497 mov rbp,rsp
40049a mov DWORD PTR [rbp-0x20],0x1
4004a1 lea rax,[rbp-0x20]
4004a5 mov QWORD PTR [rbp-0x8],rax
4004a9 mov rax,QWORD PTR [rbp-0x8]
4004ad mov eax,DWORD PTR [rax+0x28]
4004b0 mov DWORD PTR [rbp-0xc],eax
4004b3 mov eax,0x0
4004b8 pop rbp
4004b9 ret
```

Implementation Details - Pintool

Invalid Pointer Access

```
int main()
{
  int b[5];
  // Check-1
  b[0] = 1;
  // Check-2
  int *ptr = b;
  // Check-3
  int c = *(ptr + 10);
  return 0;
}
```

```
400496 push rbp
400497 mov rbp,rsp
// Check-1 //
40049a mov DWORD PTR [rbp-0x20],0x1
4004a1 lea rax,[rbp-0x20]
// Check-2 //
4004a5 mov QWORD PTR [rbp-0x8],rax
4004a9 mov rax,QWORD PTR [rbp-0x8]
// Check-3 //
4004ad mov eax,DWORD PTR [rax+0x28]
4004b0 mov DWORD PTR [rbp-0xc],eax
4004b3 mov eax,0x0
4004b8 pop rbp
4004b9 ret
```

Implementation Details - Pintool

Our Approach

- Add analysis routine before each instruction
- Check if the owner bound information is present in the global structure
- Add bound information into global structure

Implementation Details - Pintool

Invalid Pointer Access

```
int main()
{
  int b[5];
  // Check-1
  b[0] = 1;
  // Check-2
  int *ptr = b;
  // Check-3
  int c = *(ptr + 10);
  return 0;
}
```

```
400496 push rbp
400497 mov rbp,rsp
// Check-1 //
40049a mov DWORD PTR [rbp-0x20],0x1
4004a1 lea rax,[rbp-0x20]
// Check-2 //
4004a5 mov QWORD PTR [rbp-0x8],rax
4004a9 mov rax,QWORD PTR [rbp-0x8]
// Check-3 //
4004ad mov eax,DWORD PTR [rax+0x28]
4004b0 mov DWORD PTR [rbp-0xc],eax
4004b3 mov eax,0x0
4004b8 pop rbp
4004b9 ret
```

Implementation Details - Pintool

Calculate bounds information

```
        |----> lower_bound = rbp - 32
main_b  |
        |----> upper_bound = rbp - 32 + size = rbp - 32 + 19
```

Implementation Details - Pintool

What if variable is in data section?

```
           |----> lower_bound = address
main_owner |
           |----> upper_bound = address + size
```

Implementation Details - Pintool

Our Approach

- Add analysis routine before each instruction
- Check if the owner bound information is present in the global structure
- Add bound information into global structure
- Check if access is within the bounds

Implementation Details - Pintool

Check if access is within the bounds

```
if (access < lower_bound || access > upper_bound)
    abort;
```

Implementation Details - Pintool

Bounds propagation

- Our implementation supports pointer metadata propagation

Implementation Details - Pintool

Bounds propagation

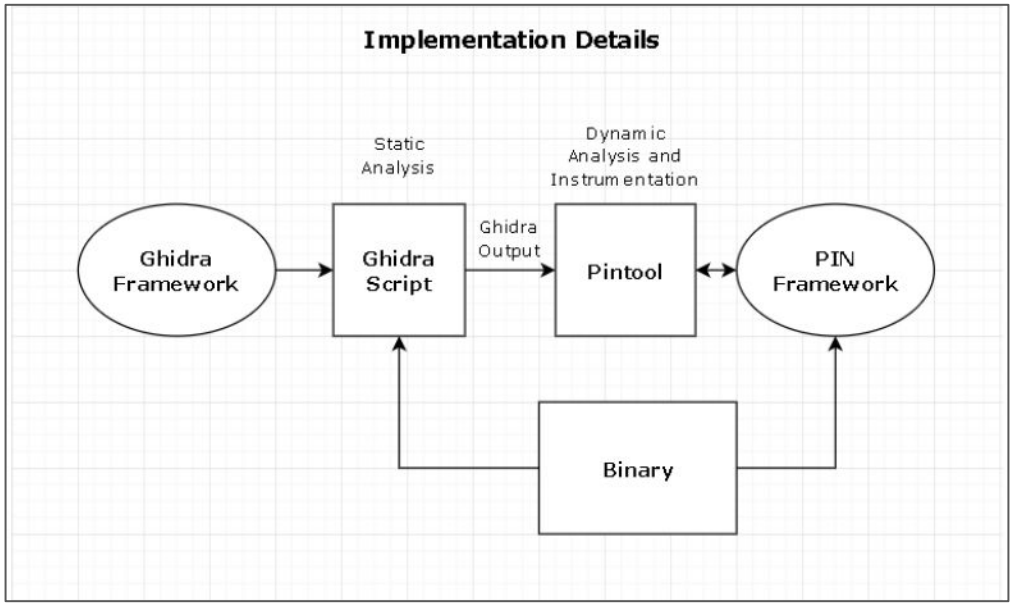
```
void foo(int *ptr)
{
  int* x = ptr;
  int y = *(x+5);
}

int main()
{
  int b[5];
  b[0] = 1;
  foo(b);
  return 0;
}
```

```
<foo>:
400496 push rbp
400497 mov rbp,rsp
40049a mov QWORD PTR [rbp-0x18],rdi
40049e mov rax,QWORD PTR [rbp-0x18]
4004a2 mov QWORD PTR [rbp-0x8],rax
4004a6 mov rax,QWORD PTR [rbp-0x8]
4004aa mov eax,DWORD PTR [rax+0x14]
4004ad mov DWORD PTR [rbp-0xc],eax
4004b0 nop
4004b1 pop rbp
4004b2 ret

<main>:
4004b3 push rbp
4004b4 mov rbp,rsp
4004b7 sub rsp,0x20
4004bb mov DWORD PTR [rbp-0x20],0x1
4004c2 lea rax,[rbp-0x20]
4004c6 mov rdi,rax
4004c9 call 400496 <foo>
4004ce mov eax,0x0
4004d3 leave
4004d4 ret
```

Implementation Details



Contents

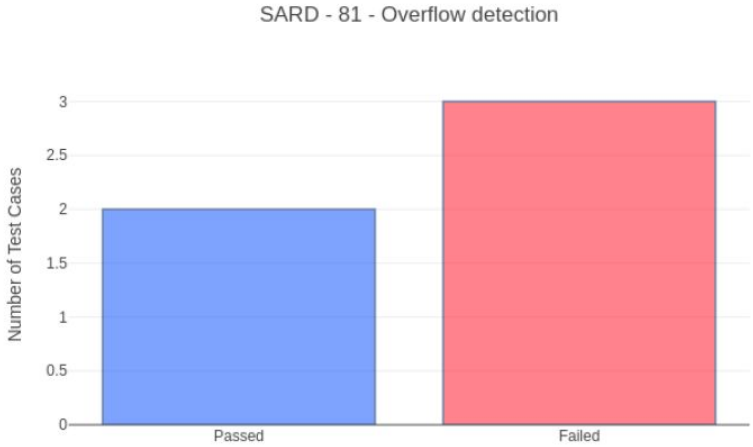
- Goal
- Background
- Implementation Details
- Results
- Future Work
- Conclusion

Results

- Overflow detection
- Run time overhead
- We use SARD benchmarks for testing

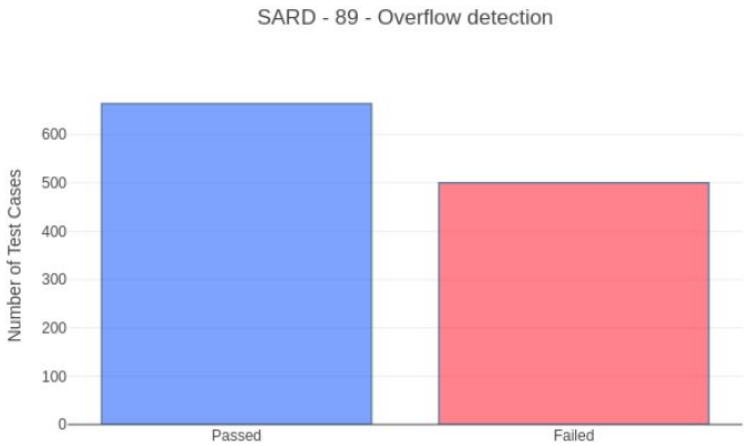
Results - Error Detection

2 out of 5 test cases are passed



Results - Error Detection

664 out of 1164 cases are passed



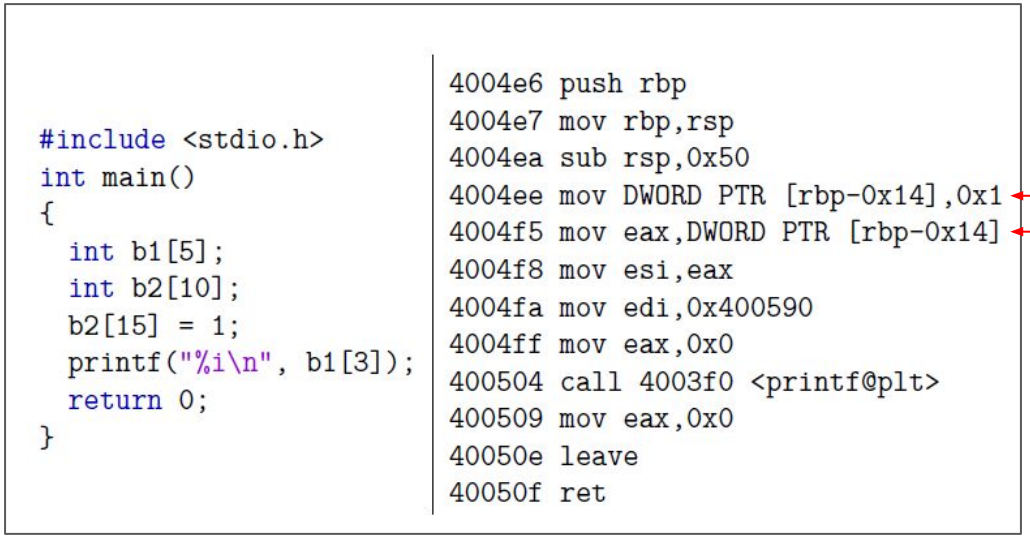
Results - Error Detection

Case failure details

Reason	Number of Test Cases
Ghidra script failed to detect	446
Pintool failed to detect	9
String copy (strcpy and strncpy) functions	25
Shared memory Operation (shmat) function	4
Get current working directory (getcwd) function	3
Copy memory area (memcpy) function	12
Threading	4

Results - Error Detection

Ghidra script detection failure

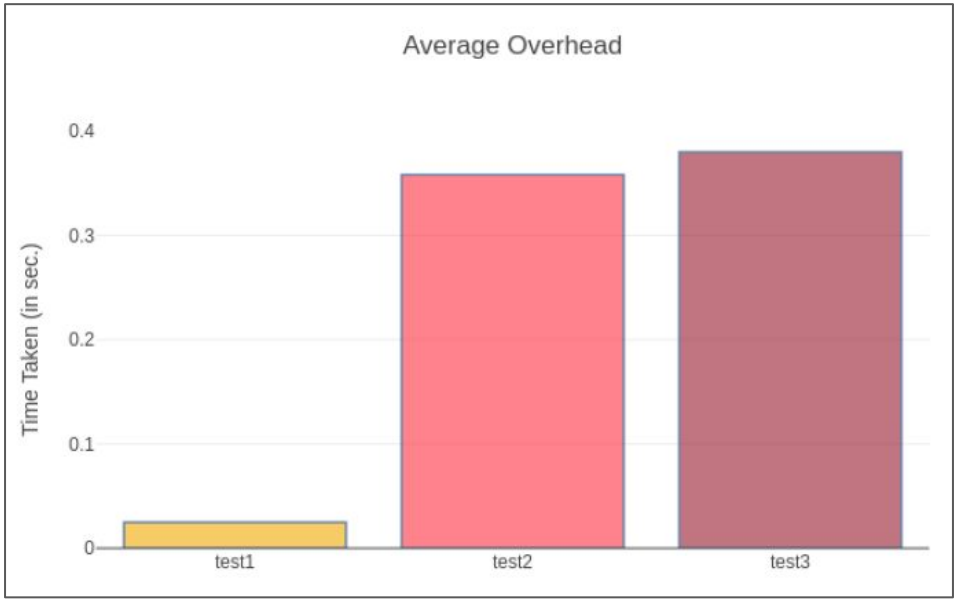


```
#include <stdio.h>
int main()
{
  int b1[5];
  int b2[10];
  b2[15] = 1;
  printf("%i\n", b1[3]);
  return 0;
}
```

```
4004e6 push rbp
4004e7 mov rbp,rsp
4004ea sub rsp,0x50
4004ee mov DWORD PTR [rbp-0x14],0x1
4004f5 mov eax,DWORD PTR [rbp-0x14]
4004f8 mov esi,eax
4004fa mov edi,0x400590
4004ff mov eax,0x0
400504 call 4003f0 <printf@plt>
400509 mov eax,0x0
40050e leave
40050f ret
```

Results - Run time overhead

Overhead is about 6% over minimal pintool



Contents

- Goal
- Background
- Implementation Details
- Results
- Future Work
- Conclusion

Future Work

- Temporal Safety
- Completeness
- Optional Debugging Information
- Overhead Reduction and SPEC benchmark checks

Contents

- Goal
- Background
- Implementation Details
- Results
- Future Work
- Conclusion

Conclusion

- Spatial safety in binaries
- Based on SoftBound approach
- Using static analysis and dynamic instrumentation
- No source code support
- No compiler transformation
- No hardware changes

Thanks so much for listening!

Questions?

Ghidra script failed to detect

```
int main(int argc, char *argv[])
{
  char buf[10];
  /* BAD */
  buf[10] = 'A';
  return 0;
}
```

```
int main(int argc, char *argv[])
{
  char buf[10];
  /* BAD */
  buf[4105] = 'A';
  return 0;
}
```

Pintool detection failure

```
int main(int argc, char *argv[])
{
  int i;
  char buf[10];
  i = 9;
  /* OK */
  (buf + i)[0] = 'A';
  return 0;
}
```

String copy example

```
#include <string.h>
int main(int argc, char *argv[])
{
  char buf[10];
  /* BAD */
  strcpy(buf, "AAAAAAAAAAAAAAAA");
  return 0;
}
```

Shared memory operations

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <assert.h>
#include <stdlib.h>
int getSharedMem()
{
  return (shmget(IPC_PRIVATE, 10, 0xffffffff));
}
void relSharedMem(int memID)
{
  struct shmid_ds temp;
  shmctl(memID, IPC_RMID, &temp);
}
int main(int argc, char *argv[])
{
  int memIdent;
  char * buf;
  memIdent = getSharedMem();
  assert(memIdent != -1);
  buf = ((char *) shmat(memIdent, NULL, 0));
  assert(((int)buf) != -1);
  /* BAD */
  buf[17] = 'A';
  shmdt((void *)buf);
  relSharedMem(memIdent);
  return 0;
}
```

Current working directory

```
#include <unistd.h>
int main(int argc, char *argv[])
{
  char buf[10];
  /* BAD */
  getcwd(buf, 18);
  return 0;
}
```

Copy memory area

```c
#include <string.h>
int main(int argc, char *argv[])
{
  int copy_size;
  char src[4106];
  char buf[10];
  memset(src, 'A', 4106);
  src[4106 - 1] = '\0';
  copy_size = -1;
  if (copy_size <= (int)(sizeof buf))
  {
    /* BAD */
    memcpy(buf, src, copy_size);
  }
  return 0;
}
```

Threading

```c
#include <pthread.h>
void * thread_function1(void * arg1)
{
  char buf[10];
  /* BAD */
  buf[4105] = 'A';
  pthread_exit((void *)NULL);
  return 0;
}
int main(int argc, char *argv[])
{
  pthread_t thread1;
  pthread_create(&thread1, NULL, &thread_function1, (void *)NULL);
  pthread_exit((void *)NULL);
  return 0;
}
```