Fuzzing AOSP for non-crash bugs

Elphet & Guang Gong
@360 Alpha Team

About us

Elphet

- Security Researcher (Intern) of 360 Alpha Team
- Focus on Android system and application security

Guang Gong

- Senior Security Researcher and Team Leader of 360 Alpha Team
- Focus on Android/Chrome

Introduction

Fuzzers catch memory corruption bugs via crashes.

When a bug doesn’ t result in a crash, then it will be ignored.

Memory debugging tool also known as a runtime debugger is a tool for finding software memory problem during runtime.

They increase the crash rate of a program by reporting errors positively. (e.g. ASAN, MSAN, TSAN)

- Some kinds of bugs cannot be uncovered even with the help of sanitizers.
  - e.g. Intra-object-overflow
  - We built a tool based on LLVM to help our fuzzers find ~30 vulnerabilities in AOSP

Agenda

Fuzzers and Sanitizers

Intra-object-overflow Bugs

LLVM and IOODetector

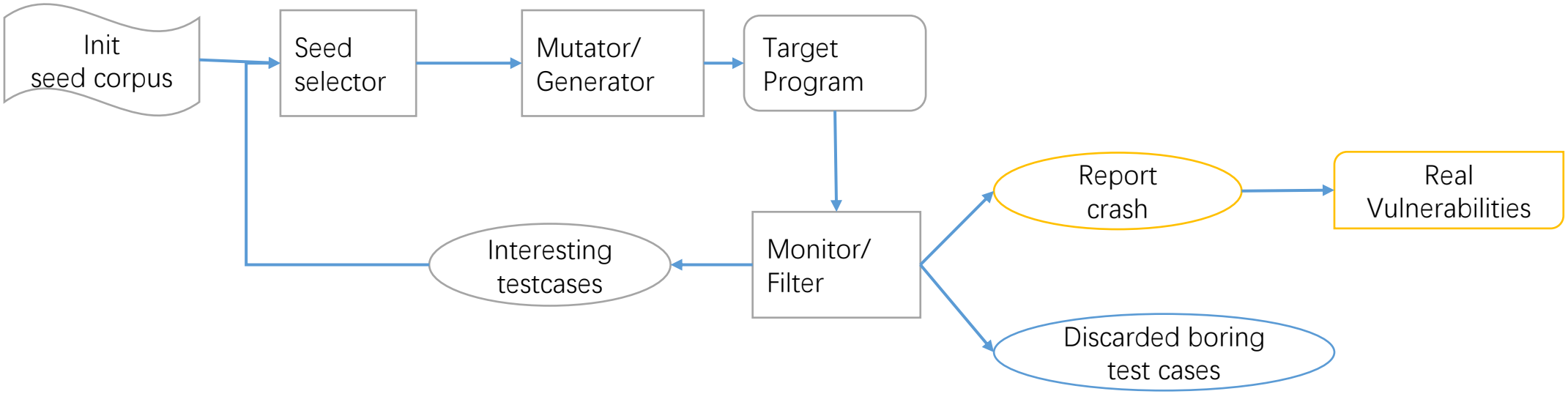
Case study

Related Work and Discussion

Fuzzers and Sanitizers

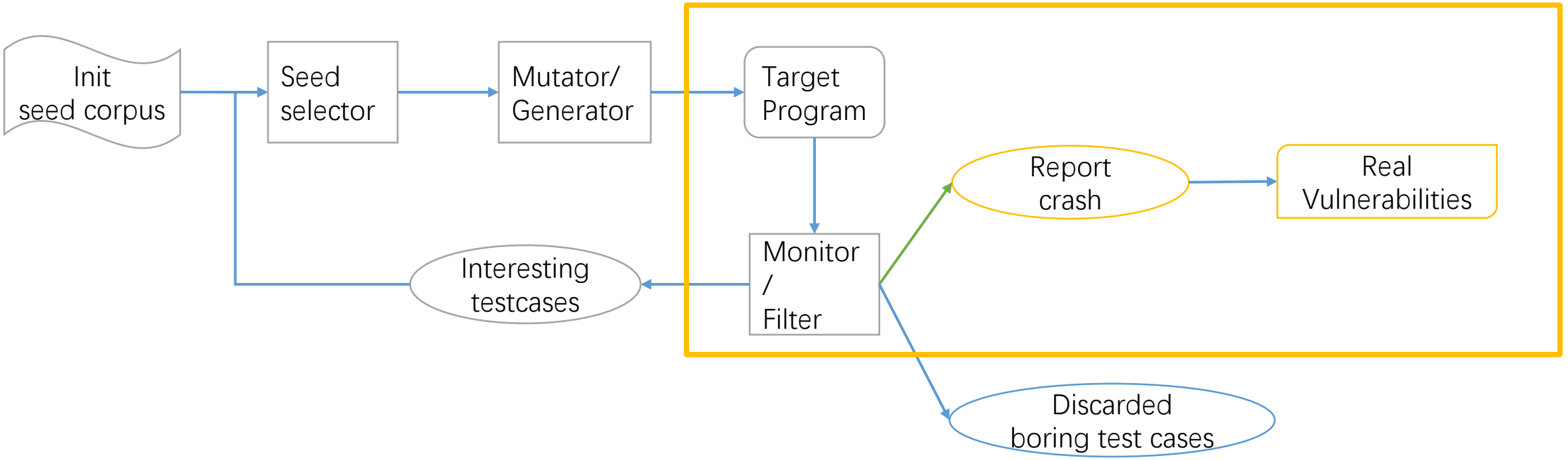
Fuzzing is proved to be an effective way to find memory corruption bugs.

- A general workflow of a fuzzer



Fuzzers

Crash is a necessary signal for identifying a vulnerability in general purpose fuzzers



Crash handlers in popular fuzzers

Crash handler in AFL

```
tatic u8 run_target(char** argv, u32 timeout) {
///......
if (dumb_mode == 1 || no_forkserver) {
  if (waitpid(child_pid, &status, 0) <= 0) PFATAL("waitpid() failed");
} else {
  s32 res;
  if ((res = read(fsrv_st_fd, &status, 4)) != 4) {
    if (stop_soon) return 0;
    RPFATAL(res, "Unable to communicate with fork server (OOM?)");
  }
}
//......
/* Report outcome to caller. */
if (WIFSIGNALED(status) && !stop_soon) {
  kill_signal = WTERMSIG(status);
  if (child_timed_out && kill_signal == SIGKILL) return FAULT_TMOUT;
  return FAULT_CRASH;
}
//......
```

Crash handler in honggFuzz

```
static struct {
    const char* descr;
    bool important;
} arch_sigs[_NSIG + 1] = {
    [0 ...(_NSIG)].important = false,
    [0 ...(_NSIG)].descr = "UNKNOWN",

    [SIGTRAP].important = false,
    [SIGTRAP].descr = "SIGTRAP",

    [SIGILL].important = true,
    [SIGILL].descr = "SIGILL",

    [SIGFPE].important = true,
    [SIGFPE].descr = "SIGFPE",

    [SIGSEGV].important = true,
    [SIGSEGV].descr = "SIGSEGV",

    [SIGBUS].important = true,
    [SIGBUS].descr = "SIGBUS",

    /* Is affected from monitorSIGABRT flag */
    [SIGABRT].important = false,
    [SIGABRT].descr = "SIGABRT",

    /* Is affected from tmoutVTALRM flag */
    [SIGVTALRM].important = false,
    [SIGVTALRM].descr = "SIGVTALRM-TMOUT",

    /* seccomp-bpf kill */
    [SIGSYS].important = true,
    [SIGSYS].descr = "SIGSYS",
};
```

Crash handler in libFuzzer

```
- The fuzzing engine will execute the fuzz
target many times with different inputs in the
same process.
- It must tolerate any kind of input (empty,
huge, malformed, etc).
- It must not exit() on any input.
- It may use threads but ideally all threads
should be joined at the end of the function.
- It must be as deterministic as possible.
Non-determinism (e.g. random decisions not
based on the input bytes) will make fuzzing -
inefficient.
- It must be fast. Try avoiding cubic or
greater complexity, logging, or excessive
memory consumption.
- Ideally, it should not modify any global
state (although that's not strict).
- Usually, the narrower the target the better.
E.g. if your target can parse several data
formats, split it into several targets, one per
format.
```

Memory debugging tools (Sanitizers)

Address Sanitizer

- Use after free
- Heap Buffer overflow
- Stack Buffer overflow
- Global buffer overflow
- Use after return
- Use after scope
- Initialization order bugs

- Leak Sanitizer
  - Memory leaks
- Memory Sanitizer
  - use of uninitialized memory
- Thread Sanitizer
  - Data races and dead locks
- Undefined Behavior Sanitizer
  - .....

Address Sanitizer Founded Bugs

Sanitizers help fuzzers find more bugs by raise interesting signals in the program

It founds many bugs in open source projects

Chromium, WebKit Safari, iTunes php parrot RocksDB

Mozilla ffmpeg webrtc libreoffice PostgreSQL

vim Opera FreeType perl Phusion Passenger

Hypertable vlc bash libcurl MySQL

Algorithm of ASAN

Core part: Shadow memory and poisoned red zone

After Instrumentation

Heap buffer overflow check

Stack buffer overflow check

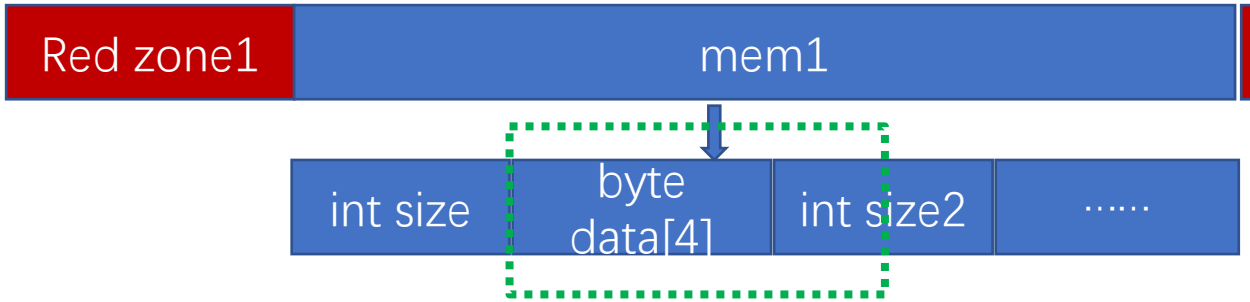
Before Instrumentation

```
void foo() {
  char a[8];
  ...
  return;
}
```

```
void foo() {
  char redzone1[32];  // 32-byte aligned
  char a[8];          // 32-byte aligned
  char redzone2[24];
  char redzone3[32];  // 32-byte aligned
  int  *shadow_base = MemToShadow(redzone1);
  shadow_base[0] = 0xffffffff;  // poison redzone1
  shadow_base[1] = 0xffffff00;  // poison redzone2, unpoison 'a'
  shadow_base[2] = 0xffffffff;  // poison redzone3
  ...
  shadow_base[0] = shadow_base[1] = shadow_base[2] = 0; // unpoison all
  return;
}
```

Bugs that ASAN cannot detect

Overflow an inner field in an object




```
struct s1{
    int size;
    uint8_t data[4];
    int size2;
}obj;
obj.data[5] = input();
```

Overflow a buffer with a large array index


```
index = 4 + sizeof(redzone2);
data[index] = input();
```

Intra Object Overflow

AddressSanitizerIntraObjectOverflow

Alexander Potapenko edited this page on 31 Aug 2015 · 2 revisions

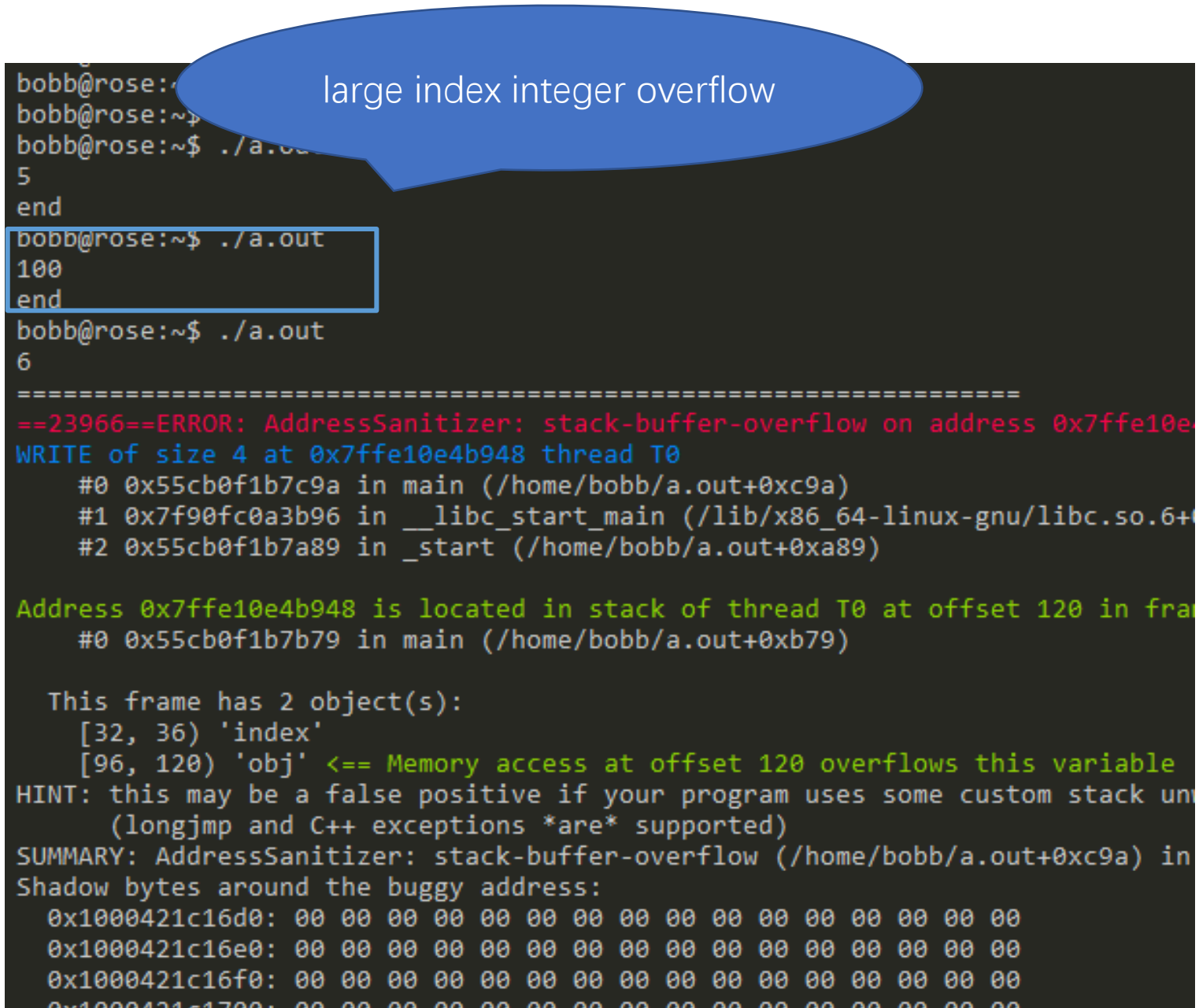
One class of bugs currently not properly detected by AddressSanitizer is **intra-object-overflow**:

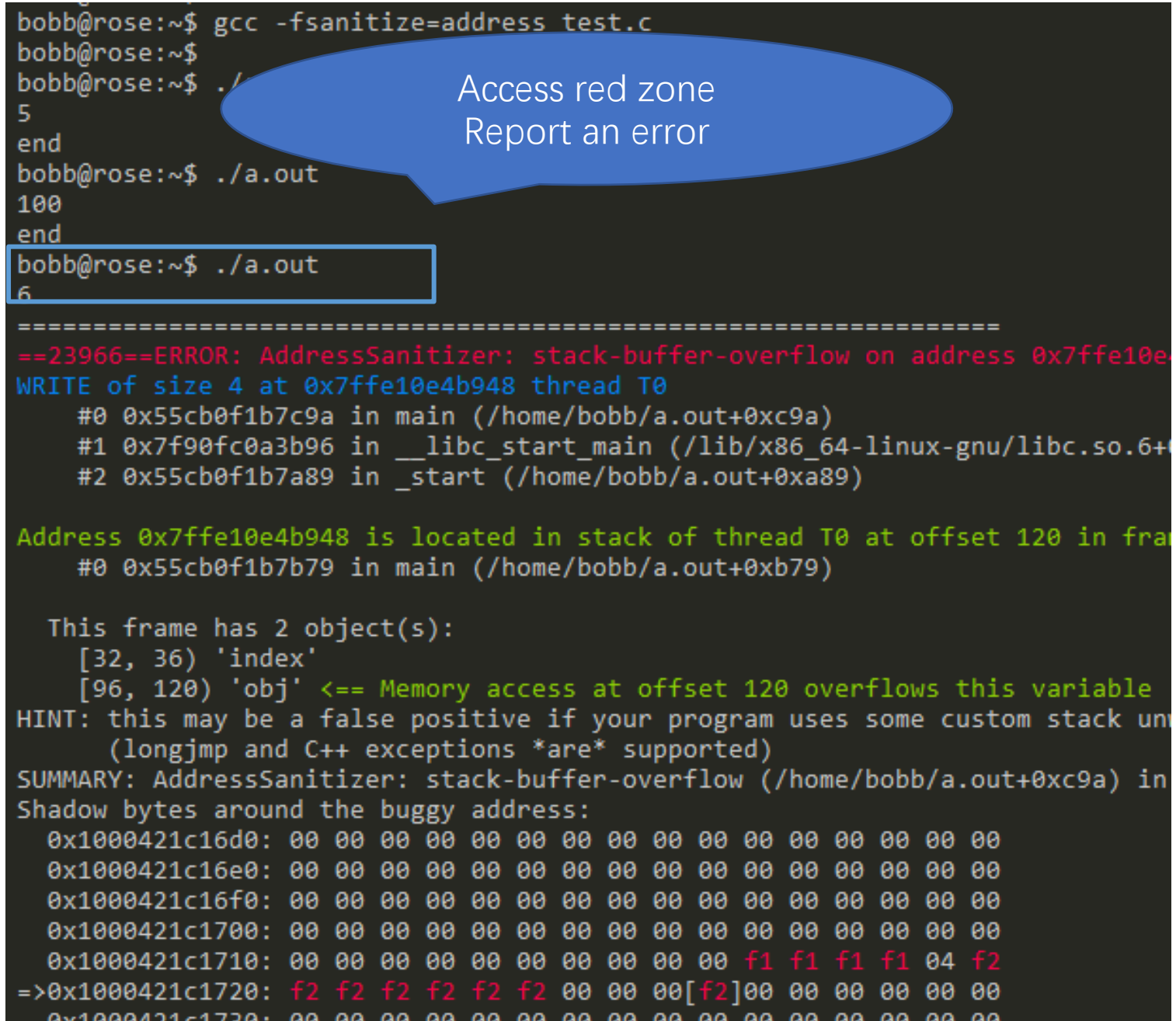
```
struct S {
  int buffer[5];
  int other_field;
};

void Foo(S *s, int idx) {
  s->buffer[idx] = 0;  // if idx == 5, asan will not complain
}
```

```
bobb@rose:~$ cat test.c
#include<stdio.h>

struct s1{
    int buffer[5];
    int other_field;
};

int main(){
    struct s1 obj;
    int index;
    scanf("%d", &index);
    obj.buffer[index] = 0;
    printf("end\n");
    return 0;
}
```

intra-object-overflow
other_field overwriten

```
bobb@rose:~$ gcc -fsani
bobb@rose:~$
bobb@rose:~$ ./a.out
5
end
bobb@rose:~$ ./a.out
100
end
bobb@rose:~$ ./a.out
6
=================================================================
==23966==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffe10e
WRITE of size 4 at 0x7ffe10e4b948 thread T0
    #0 0x55cb0f1b7c9a in main (/home/bobb/a.out+0xc9a)
    #1 0x7f90fc0a3b96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+
    #2 0x55cb0f1b7a89 in _start (/home/bobb/a.out+0xa89)

Address 0x7ffe10e4b948 is located in stack of thread T0 at offset 120 in fra
    #0 0x55cb0f1b7b79 in main (/home/bobb/a.out+0xb79)

  This frame has 2 object(s):
    [32, 36) 'index'
    [96, 120) 'obj' <== Memory access at offset 120 overflows this variable
HINT: this may be a false positive if your program uses some custom stack un
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/home/bobb/a.out+0xc9a) in
Shadow bytes around the buggy address:
  0x1000421c16d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000421c16e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000421c16f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000421c1700: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```


large index integer overflow
bobb@rose:~$ ./a.out
5
end
bobb@rose:~$ ./a.out
100
end
bobb@rose:~$ ./a.out
6
==23966==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffe10e
WRITE of size 4 at 0x7ffe10e4b948 thread T0
#0 0x55cb0f1b7c9a in main (/home/bobb/a.out+0xc9a)
#1 0x7f90fc0a3b96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+
#2 0x55cb0f1b7a89 in _start (/home/bobb/a.out+0xa89)
Address 0x7ffe10e4b948 is located in stack of thread T0 at offset 120 in fra
#0 0x55cb0f1b7b79 in main (/home/bobb/a.out+0xb79)
This frame has 2 object(s):
[32, 36) 'index'
[96, 120) 'obj' <== Memory access at offset 120 overflows this variable
HINT: this may be a false positive if your program uses some custom stack un
(longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/home/bobb/a.out+0xc9a) in
Shadow bytes around the buggy address:
0x1000421c16d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1000421c16e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1000421c16f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


bobb@rose:~$ gcc -fsanitize=address test.c
bobb@rose:~$
bobb@rose:~$ ./
5
end
bobb@rose:~$ ./a.out
100
end
bobb@rose:~$ ./a.out
6
==23966==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffe10e
WRITE of size 4 at 0x7ffe10e4b948 thread T0
#0 0x55cb0f1b7c9a in main (/home/bobb/a.out+0xc9a)
#1 0x7f90fc0a3b96 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+
#2 0x55cb0f1b7a89 in _start (/home/bobb/a.out+0xa89)
Address 0x7ffe10e4b948 is located in stack of thread T0 at offset 120 in fra
#0 0x55cb0f1b7b79 in main (/home/bobb/a.out+0xb79)
This frame has 2 object(s):
[32, 36) 'index'
[96, 120) 'obj' <== Memory access at offset 120 overflows this variable
HINT: this may be a false positive if your program uses some custom stack un
(longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/home/bobb/a.out+0xc9a) in
Shadow bytes around the buggy address:
0x1000421c16d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1000421c16e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1000421c16f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1000421c1700: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x1000421c1710: 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1 04 f2
=>0x1000421c1720: f2 f2 f2 f2 f2 f2 00 00 00[f2]00 00 00 00 00 00
Access red zone
Report an error

Intra Object Overflow Detector

Manually?

Manually sanitize the index to avoid buffer overflow after reviewing the project

A lot of boring work. Some bugs would be missed.

```
bobb@rose:~$ cat test.c
#include<stdio.h>
#define BUFFER_SIZE 5

struct s1{
    int buffer[BUFFER_SIZE];
    int other_field;
};

void crashme(){
    unsigned int badaddr = 0xdeadbeaf;
    *(unsigned int*)badaddr = badaddr;
}

int main(){
    struct s1 obj;
    int index;
    scanf("%d", &index);
    if( index >= BUFFER_SIZE ){
        crashme();
    }
    obj.buffer[index] = 0;
    printf("end\n");
    return 0;
}
bobb@rose:~$
bobb@rose:~$ ./a.out
5
Segmentation fault (core dumped)
```

Intra Object Overflow Detector

Automatically?

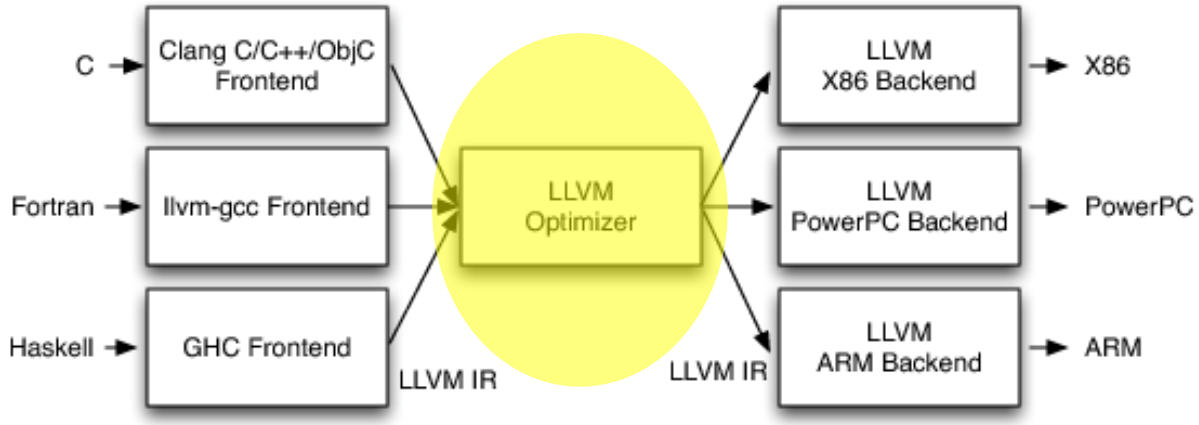
- Static Analysis
  - LLVM Passes
  - Data Flow Analysis
- Dynamic Analysis
  - LLVM Instrumentation
  - Data Flow tracing
  - Fuzzing

IOODetector

LLVM

- Clang Frontend translate source code into IR
- LLVM optimizer performs a sequence of optimization on IR
- Backend then translate the optimized IR into machine code

- We can customize our own LLVM passes on the IR



LLVM IR

```
bobb@rose:~$ cat test.c
#include<stdio.h>
#define BUFFER_SIZE 5

struct s1{
    int size;
    char buffer[BUFFER_SIZE];
    int other_field;
};

int main(){
    struct s1 obj;
    int index;
    scanf("%d", &index);
    obj.buffer[index] = 0;
    printf("end\n");
    return 0;
}
```

```
bobb@rose:~$ cat test.ll
; ModuleID = 'test.c'
source_filename = "test.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

%struct.s1 = type { i32, [5 x i8], i32 }

@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str.1 = private unnamed_addr constant [5 x i8] c"end\0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca %struct.s1, align 4
  %3 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %4 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i32 0, i32 0), i32* %3)
  %5 = getelementptr inbounds %struct.s1, %struct.s1* %2, i32 0, i32 1
  %6 = load i32, i32* %3, align 4
  %7 = sext i32 %6 to i64
  %8 = getelementptr inbounds [5 x i8], [5 x i8]* %5, i64 0, i64 %7
  store i8 0, i8* %8, align 1
  %9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x i8]* @.str.1, i32 0, i32 0))
  ret i32 0
}

declare i32 @__isoc99_scanf(i8*, ...) #1

declare i32 @printf(i8*, ...) #1
```

LLVM IR

```
bobb@rose:~$ cat test.c
#include<stdio.h>
#define BUFFER_SIZE 5

struct s1{
    int size;
    char buffer[BUFFER_SIZE];
    int other_field;
};

int main(){
    struct s1 obj;
    int index;
    scanf("%d", &index);
    obj.buffer[index] = 0;
    printf("end\n");
    return 0;
}
```

```
bobb@rose:~$ cat test.ll
; ModuleID = 'test.c'
source_filename = "test.c"
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

%struct.s1 = type { i32, [5 x i8], i32 }

@.str = private unnamed_addr constant [3 x i8] c"%d\00", align 1
@.str.1 = private unnamed_addr constant [5 x i8] c"end\0A\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca %struct.s1, align 4
  %3 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  %4 = call i32 (i8*, ...) @__isoc99_scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @.str, i32 0, i32 0), i32* %3)
  %5 = getelementptr inbounds %struct.s1, %struct.s1* %2, i32 0, i32 1
  %6 = load i32, i32* %3, align 4
  %7 = sext i32 %6 to i64
  %8 = getelementptr inbounds [5 x i8], [5 x i8]* %5, i64 0, i64 %7
  store i8 0, i8* %8, align 1
  %9 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([5 x i8], [5 x i8]* @.str.1, i32 0, i32 0))
  ret i32 0
}

declare i32 @__isoc99_scanf(i8*, ...) #1

declare i32 @printf(i8*, ...) #1
```

LLVM IR GEP

‘getelementptr’ Instruction

Syntax:

```
<result> = getelementptr <ty>, <ty>* <ptrval>{, [inrange] <ty> <idx>}*
<result> = getelementptr inbounds <ty>, <ty>* <ptrval>{, [inrange] <ty> <idx>}*
<result> = getelementptr <ty>, <ptr vector> <ptrval>, [inrange] <vector index type> <idx>
```

The ‘getelementptr’ instruction is used to get the address of a subelement of an aggregate data structure. It performs address calculation only and does not access memory.

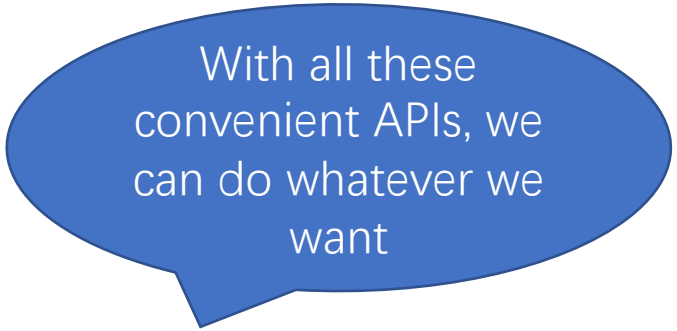
arg1: a type used as the basis for the calculations

arg2: a pointer or a vector of pointers, and is the base address to start from

arg3..n : indices that indicate which of the elements of the aggregate object are

LLVM IR GEP Instruction API

- getSourceElementType()
- getResultElementType()
- getNumIndices()
- hasIndices()
- Indices()
- getPointerOperand()
- getPointerOperandType()
- getOperand()
- ……



IOODetector simplest solution

- A check function call will be inserted before every GEP instruction whose source Type is an array
- Detector sanitize two things

```
void check_ioo(int size, int index){
        if ( index < 0 || index >= size ){
                printf("[ERROR DETECTED] size=%d, index=%d\n", size, index);
                raise(SIGSEGV);
        }
}
```

IOODetector’ s simplest solution

- Checking Results:

```
bobb@rose:~$ cat test.c
#include<stdio.h>

struct s1{
    int buffer[5];
    int other_field;
};

int main(){
    struct s1 obj;
    int index;
    scanf("%d", &index);
    obj.buffer[index] = 0;
    printf("end\n");
    return 0;
}
```

```
bobb@rose:~$ ./a.out
5
[ERROR DETECTED] size=5, index=5
Segmentation fault (core dumped)
```

IOODetector’ s simplest solution

- Checking Results:

```
bobb@rose:~$ cat test.c
#include<stdio.h>

struct s1{
    int buffer[5];
    int other_field;
};

int main(){
    struct s1 obj;
    int index;
    scanf("%d", &index);
    obj.buffer[index] = 0;
    printf("end\n");
    return 0;
}
```

```
bobb@rose:~$ ./a.out
5
[ERROR DETECTED] size=5, index=5
Segmentation fault (core dumped)
```

However, it’ s not enough

GEP without numOfElements

1. SourceType is not ArrayType.
2. Array NumElement is missing

```c
#include <stdio.h>

struct s1{
        int size;
        char data[10];
        int other_field;
};

int assign_data(char* data, int index){
        int i = data[index];
        int j = *(data+index);
        data[index] = 1;
        *(data+index) = 2;

        return 0;
}

int main(){
        struct s1 obj;
        int size;

        scanf("%d", &size);
        assign_data(&obj.data[0], size);
        printf("END\n");
        return 0;
}
```

```
define i32 @assign_data(i8*, i32) #0 {
  %3 = alloca i8*, align 8
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  store i8* %0, i8** %3, align 8
  store i32 %1, i32* %4, align 4
  %7 = load i8*, i8** %3, align 8
  %8 = load i32, i32* %4, align 4
  %9 = sext i32 %8 to i64
  %10 = getelementptr inbounds i8, i8* %7, i64 %9
  %11 = load i8, i8* %10, align 1
  %12 = sext i8 %11 to i32
  store i32 %12, i32* %5, align 4
  %13 = load i8*, i8** %3, align 8
  %14 = load i32, i32* %4, align 4
  %15 = sext i32 %14 to i64
  %16 = getelementptr inbounds i8, i8* %13, i64 %15
  %17 = load i8, i8* %16, align 1
  %18 = sext i8 %17 to i32
  store i32 %18, i32* %6, align 4
  %19 = load i8*, i8** %3, align 8
  %20 = load i32, i32* %4, align 4
  %21 = sext i32 %20 to i64
  %22 = getelementptr inbounds i8, i8* %19, i64 %21
  store i8 1, i8* %22, align 1
  %23 = load i8*, i8** %3, align 8
  %24 = load i32, i32* %4, align 4
  %25 = sext i32 %24 to i64
  %26 = getelementptr inbounds i8, i8* %23, i64 %25
  store i8 2, i8* %26, align 1
  ret i32 0
}
```

Store and Load Instructions(Memory Access)

```
4  struct s{
5      char array1[10];
6      char array2[15];
7      int size;
8  };
9
10 int main(){
11     int a;
12     scanf("%d", &a);
13
14     struct s obj;
15
16     char* ptr;
17     if ( a % 2 == 0 ){
18         ptr = obj.array1;
19     }else{
20         ptr = obj.array2;
21     }
22
23     for(int i = 0; i <= 20; i++){
24         ptr[i] = 0;
25     }
26
27     return 0;
```

```
1406 ; <label>:10:
1407   %11 = getelementptr inbounds %struct
1408   %12 = getelementptr inbounds [10 x
1409   store i8* %12, i8** %4, align 8
1410   br label %16
1411
1412 ; <label>:13:
1413   %14 = getelementptr inbounds %struct.
1414   %15 = getelementptr inbounds [15 x i8], [15
1415   store i8* %15, i8** %4, align 8
1416   br label %16
1417
1418 ; <label>:16:                                     ; preds = %13, %10
1419   store i32 0, i32* %5, align 4
1420   br label %17
1421
1422 ; <label>:17:                                     ; preds = %25, %16
1423   %18 = load i32, i32* %5, align 4
1424   %19 = icmp sle i32 %18, 20
1425   br i1 %19, label %20, label %28
1426
1427 ; <label>:20:                                     ; preds = %17
1428   %21 = load i8*, i8** %4, align 8
1429   %22 = load i32, i32* %5, align 4
1430   %23 = sext i32 %22 to i64
1431   %24 = getelementptr inbounds i8, i8* %21, i64 %23
1432   store i8 0, i8* %24, align 1
```

The real memory access behavior happens in load&store instruction. Checking on GEP instruction will result in false positive

Store and Load Instructions(Pointer Ref && Deref)

```
 4 struct s{
 5     char array1[10];
 6     char array2[15];
 7     int size;
 8 };
 9
10 int main(){
11     int a;
12     scanf("%d", &a);
13
14     struct s obj;
15
16     char* ptr;
17     if ( a % 2 == 0 ){
18         ptr = obj.array1;
19     }else{
20         ptr = obj.array2;
21     }
22
23     for(int i = 0; i <= 20; i++){
24         ptr[i] = 0;
25     }
26
27     return 0;
```

```
1406 ; <label>:10:
1407   %11 = getelementptr inbounds %struct.s
1408   %12 = getelementptr inbounds [10 x
1409   store i8* %12, i8** %4, align 8
1410   br label %16
1411
1412 ; <label>:13:
1413   %14 = getelementptr inbounds %struct
1414   %15 = getelementptr inbounds [15 x i8],
1415   store i8* %15, i8** %4, align 8
1416   br label %16
1417
1418 ; <label>:16:                                     ; preds = %13, %10
1419   store i32 0, i32* %5, align 4
1420   br label %17
1421
1422 ; <label>:17:                                     ; preds = %25, %16
1423   %18 = load i32, i32* %5, align 4
1424   %19 = icmp sle i32 %18, 20
1425   br i1 %19, label %20, label %28
1426
1427 ; <label>:20:                                     ; preds = %17
1428   %21 = load i8*, i8** %4, align 8
1429   %22 = load i32, i32* %5, align 4
1430   %23 = sext i32 %22 to i64
1431   %24 = getelementptr inbounds i8, i8* %21, i64 %23
1432   store i8 0, i8* %24, align 1
```

Load and Store instructions are also used to propagate tainted nodes.
If the src of StoreInst/loadInst is tainted

callInst and retInst


```
17   %1 = alloca i32*, align 8
18   %2 = call i32* @_Z6getPtrv()
19   store i32* %2, i32** %1, align 8
20   %3 = load i32*, i32** %1, align 8
21   %4 = getelementptr inbounds i32, i32* %3, i64 10
22   %5 = load i32, i32* %4, align 4
23   %6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @.str, i32 0, i32 0), i32 %5)
24   ret i32 0
25 }
```

```
1582
1583 ; <label>:29:                                     ; preds = %0
1584   %30 = getelementptr inbounds %struct.s1, %struct.s1* %2, i32 0, i32 1
1585   %31 = getelementptr inbounds [5 x i8], [5 x i8]* %30, i64 0, i64 0
1586   %32 = load i32, i32* %3, align 4
1587   %33 = call i32 @_Z5case4Pci(i8* %31, i32 %32)
1588   br label %41
1589
```

IOODetector’ s Solution

- Step1: Find all explicit GEP instructions. Allocate a tag for them.
  - new_node(int unique_tag, int current_index, int number_element);
- Step2: Traverse its user list
  - propagate(int src_tag, int uniq_dest_tag, int offset)
  - check(int tag)
- Step3: Recursively traverse the user list of its user
  - propagate(int src_tag, int uniq_dest_tag, int offset)
  - check(int tag)

IOODetector’ s Solution

- Taint Source
  - GEP Instruction
    - %arrayidx14.i = getelementptr inbounds [16 x i32], [16 x i32]* %16, i32 0, i32 0
- Taint Propagation
  - GEP Instruction
    - %arrayidx23.i = getelementptr inbounds i32, i32* %279, i32 8
  - Load && Restore Instruction
    - store i32* %2, i32** %1, align 8
    - %3 = load i32*, i32** %1, align 8
  - Call && Return Instruction
    - %33 = call i32 @func(i8* %31, i32 %32)
    - ret i32* %25
- Check Point
  - Load & Store Instruction
    - %37 = load i32, i32* %3, align 4
    - store i32 %24, i32* %4, align 4

Detection Result

```c
#include<stdio.h>
#define BUFFER_SIZE 5

struct s1{
    int buffer[BUFFER_SIZE];
    int other_field;
};

int main(){
    struct s1 obj;
    int index;
    scanf("%d", &index);
    obj.buffer[index] = 0;
    printf("end\n");
    return 0;
}
```

```c
#include <stdio.h>

struct s1{
        int size;
        char data[10];
        int other_field;
};

int assign_data(char* data, int index){
        int i = data[index];
        int j = *(data+index);
        data[index] = 1;
        *(data+index) = 2;

        return 0;
}

int main(){
        struct s1 obj;
        int size;

        scanf("%d", &size);
        assign_data(&obj.data[0], size);
        printf("END\n");
        return 0;
}
```

```c
#include <stdio.h>

struct s1{
        int size;
        char data[10];
        int other_field;
};

int assign_data(char* data, int size){
        int i;
        char* tmp = data;
        for( int i=0; i < size; i++){
                *tmp = 1;
                tmp++;
        }

        return 0;
}

int main(){
        struct s1 obj;
        int size;

        scanf("%d", &size);
        assign_data(&obj.data[0], size);
        printf("END\n");
        return 0;
}
```

Detection Result

All the three examples are Successfully Detected

```c
#include<stdio.h>
#define BUFFER_SIZE 5

struct s1{
    int buffer[BUFFER_SIZE];
    int other_field;
};

int main(){
    struct s1 obj;
    int index;
    scanf("%d", &index);
    obj.buffer[index] = 0;
    printf("end\n");
    return 0;
}
```

```c
#include <stdio.h>

struct s1{
        int size;
        char data[10];
        int other_field;
};

int assign_data(char* data, int index){
        int i = data[index];
        int j = *(data+index);
        data[index] = 1;
        *(data+index) = 2;

        return 0;
}

int main(){
        struct s1 obj;
        int size;

        scanf("%d", &size);
        assign_data(&obj.data[0], size);
        printf("END\n");
        return 0;
}
```

```c
#include <stdio.h>

struct s1{
        int size;
        char data[10];
        int other_field;
};

int assign_data(char* data, int size){
        int i;
        char* tmp = data;
        for( int i=0; i < size; i++){
                *tmp = 1;
                tmp++;
        }

        return 0;
}

int main(){
        struct s1 obj;
        int size;

        scanf("%d", &size);
        assign_data(&obj.data[0], size);
        printf("END\n");
        return 0;
}
```

Other Challenges and Solutions

- Recursion -> Tags in the the same function are constants
  - fun_a->fun_b->fun_a
  - Solution: we introduce node->call_layer field to simulate the call stack.
  - For efficiency: only instrumented functions will be record.

- Multiple modules in a big project
  - Global Tag Generator:
    - Unique tags are required in every module
    - Consistent tags for the same function are required

Find real target project in AOSP

Code pattern of Intra object buffer in C

```
struct sname{
        ///......
        type name[digit_or_macro];
        ///......
};
```

Search [in .h files for a coarse result

Or

Search with regular expression in the whole AOSP

libxaac, the treasure

- libxaac is a new OMX component introduced in Android P
- XAAC stands for xHE-AAC (Extended High Efficiency Advanced Audio Coding)
- Bit rate as low as 6kbps for mono and 12kbps for stereo when network is congested. Request a higher bit rate version and seamlessly switch over once more bandwidth available
- “Adding xHE-AAC to our patent pool ensures that broadcasters and service providers can deliver the next generation of audio to consumers efficiently and affordably.”
- libxaac in Android is the first implementation of xHE-AAC

libxaac, the treasure

- Memory Management in libxaac
  - libxaac itself doesn’ t allocate or deallocate any buffers. It provides a GET_SIZE_API. API caller is responsible for memory management
  - Big chunks are allocated for efficiency. ~64MB
  - Sort of anti-fuzzing or anti-crash. Unfriendly to Fuzzer+ASAN
- Lots of intra-object arrays in the allocated big chunks

```
case IA_API_CMD_GET_API_SIZE: {
  *pui_value = sizeof(ia_drc_api_struct) +
               (sizeof(ia_drc_state_struct) + 8) + 8080 * 1024;

  break;
}
```

```
386  typedef struct {
387    WORD32 drc_set_id;
388    WORD32 drc_set_complexity_level;
389    WORD32 requires_eq;
390    WORD32 drc_apply_to_dwnmix;
391    WORD32 drc_location;
392    WORD32 dwnmix_id_count;
393    WORD32 downmix_id[DOWNMIX_ID_COUNT_MAX];
394    WORD32 depends_on_drc_set_present;
395    WORD32 depends_on_drc_set;
396    WORD32 no_independent_use;
397    WORD32 drc_set_effect;
398    WORD32 gain_set_index[MAX_CHANNEL_COUNT];
399    ia_gain_modifiers_struct
400        str_gain_modifiers_of_ch_group[CHANNEL_G
401    ia_ducking_modifiers_struct
402        str_ducking_modifiers_for_channel[MAX_CH
```

libxaac – configuration and decoding

Two attack Surfaces

configuration

decoding

Both the functions receive a buffer as input

Both the two attack surfaces uses two different big data structures with lots of intra object buffers

```
82      void initPorts();
83      status_t initDecoder();
84      bool isConfigured() const;
85      int drainDecoder();
86      int initXAACDecoder();
87      int deInitXAACDecoder();
88
89      int configXAACDecoder(uint8_t* inBuffer, uint32_t inBufferLength);
90      int configMPEGDDrc();
91      int decodeXAACStream(uint8_t* inBuffer, uint32_t inBufferLength, int32_t* bytesConsumed,
92                           int32_t* outBytes);
93
94      int configflushDecode();
95      IA_ERRORCODE getXAACStreamInfo();
96      IA_ERRORCODE setXAACDRCInfo(int32_t drcCut, int32_t drcBoost, int32_t drcRefLevel,
97                                  int32_t drcHeavyCompression
98  #ifdef ENABLE_MPEG_D_DRC
99                                  ,
100                                 int32_t drEffectType
101 #endif
102     );
```

Fuzzing libxaac

Two fuzzers for libxaac – lots of crashes raised by IOODetector

- configfuzzer
  - Testing the configuration process of libxaac
  - We also use this fuzzer to generate `good config bufs` for decoding stage. (After a config-buf is processed, if the err_code is OK, then the buf is considered as a `good config-buf`)
  - 1000+ good config bufs were generated in 1 week
- decodefuzzer
  - The second stage of decoding an audio stream. a) choose a random gcb b) feed gcb to configxaacdecoder. c) generate test cases for decodexaacstream interface

Vulnerabilities found in AOSP

Confirmed bugs

- CVE-2018-9569, CVE-2018-9570, CVE-2018-9571, CVE-2018-9572, CVE-2018-9573, CVE-2018-9574, CVE-2018-9575, CVE-2018-9576, CVE-2018-9577, CVE-2019-2063, CVE-2019-2064, CVE-2019-2065, CVE-2019-2066, CVE-2019-2067, CVE-2019-2068, CVE-2019-2069, CVE-2019-2070, CVE-2019-2071, CVE-2019-2072, CVE-2019-2073, CVE-2019-2074, CVE-2019-2075, CVE-2019-2076, CVE-2019-2077, CVE-2019-2078, CVE-2019-2079, CVE-2019-2086.

Duplicated issues

- AndroidID-119054381, AndroidID-119054381, AndroidID-117992588 , AndroidID-117789761, AndroidID-117789797, AndroidID-116772652, AndroidID-116746433, AndroidID-117079549, AndroidID-117064603, AndroidID-117105233, AndroidID-117204086, AndroidID-115919654 ……

And some other issues…

Case1 CVE-2019-2065

str_node lies in the deep layer of multiple nested structures

```
typedef struct {
    WORD32 drc_gain_coding_mode;
    WORD32 num_nodes;
    ia_node_struct str_node[NODE_COUNT_MAX];   /// <=== 256
} ia_spline_nodes_struct;

typedef struct {
    ia_spline_nodes_struct str_spline_nodes [1];
} ia_drc_gain_sequence_struct;

typedef struct {
    WORD32 uni_drc_gain_ext_type[EXT_COUNT_MAX];
    WORD32 ext_bit_size[EXT_COUNT_MAX-1];
} ia_uni_drc_gain_ext_struct;

typedef struct ia_drc_gain_struct{
    WORD32 num_drc_gain_sequences;
    ia_drc_gain_sequence_struct drc_gain_sequence[SEQUENCE_COUNT_MAX];
    WORD32 uni_drc_gain_ext_flag;
    ia_uni_drc_gain_ext_struct uni_drc_gain_ext;
} ia_drc_gain_struct;
```

str_node is a pointer that points to a field of ia_spline_nodes_struct

impd_read_bits_buf is controllable, k could be larger than 256 when end_marker never meet

out-of-bounds write here

```
xref: /external/libxaac/decoder
124WORD32 impd_dec_slopes(ia_bit_buf_str      _bit_buff, WORD32* no_nodes,
125                       WORD32 gain_interpolation
126                       ia_node_struct* str_node)
////////////////
139  k = 0;
140  while (end_marker != 1) {
141    k++;
142    end_marker = impd_read_bits_buf(it_bit_buff, 1);
143    if (it_bit_buff->error) return it_bit_buff->error;
144  }
145  *no_nodes = k;
146
147  if (gain_interpolation_type == GAIN_INTERPOLATION_TYPE_SPLINE) {
148    for (k = 0; k < *no_nodes; k++) {
////////////////
169      str_node[k].slope = slope_value;
170    }
171  } else {
172    for (k = 0; k < *no_nodes; k++) {
173      str_node[k].slope = 0.0f;
174    }
175  }
176  return (0);
177}
```

Case2 CVE-2018-9575

```
Ref#4
xref: /external/libxaac/decoder/drc_src/impd_drc_struct.h
588typedef struct ia_drc_config {
589  WORD32 sample_rate_present;
590  WORD32 sampling_rate;
591  WORD32 dwnmix_instructions_count;
592  WORD32 drc_coefficients_drc_count;
593  WORD32 drc_instructions_uni_drc_count;
594  WORD32 drc_instructions_count_plus;
595  WORD32 drc_description_basic_present;
596  WORD32 drc_coefficients_basic_count;
597  WORD32 drc_instructions_basic_count;
598  WORD32 drc_config_ext_present;
599  WORD32 apply_drc;
600  ia_drc_config_ext str_drc_config_ext;
601  ia_drc_coefficients_basic_struct
602      str_drc_coefficients_basic[DRC_COEFF_COUNT_MAX];
603  ia_drc_instructions_basic_struct
604      str_drc_instructions_basic[DRC_IN
605  ia_uni_drc_coeffs_struct
606      str_p_loc_drc_coefficients_uni_dr
607  ia_drc_instructions_struct
608      str_drc_instruction_str[DRC_INSTR
609  ia_channel_layout_struct channel_layout;
610  ia_downmix_instructions_struct
611      dwnmix_instructions[DOWNMIX_INSTRUCTION_COUNT_MAX];
    /// <=== number of elements in array dwnmix_instructions is DOWNMIX_INSTRUCTION_COUNT_MAX(16)
612} ia_drc_config;
```

DOWNMIX_INSTRUCTION_COUNT_MAX is 16

```
/external/                                                    _payload.c
1102WORD32
1103impd_p                                                      it_buff,
1104                                                          ruct* ia_drc_params_struct,
1105                                                          ig) {
///////////////
1119  temp = impd_read_bits_buf(it_bit_buff, 8);
1120  if (it_bit_buff->error) return it_bit_buff-
1121
1122  drc_config->dwnmix_instructions_count = (temp >> 1) & 0x7f;
1123  drc_config->drc_description_basic_pre
///////////////
1146
1147  for (i = 0; i < drc_config->dwnmix_ins
1148    err = impd_parse_dwnmix_instructions(
1149        it_bit_buff, version, ia_drc_params_struct, &drc_config->channel_layout,
1150        &(drc_config->dwnmix_instructions[i]));
1151    if (err) return (err);
1152  }
```

Impd_read_bits_buf is controllable. temp:[0..0xff]

dwnmix_instructions_count' s range is [0..0x7f]

dwnmix_instructions[] out-out-of-bounds access

Case2 CVE-2018-9575

```
/external/libxaac/decoder/drc_src/impd_drc_static_payload.c
748WORD32
749impd_parse_dwnmix_instructions(
750    ia_bit_buf_struct* it_bit_buff, WORD32 version,
751    ia_drc_params_bs_dec_struct* ia_drc_params_struct,
752    ia_channel_layout_struct* channel_layout,
753    ia_downmix_instructions_struct* dwnmix_instructions) {
754  // WORD32 err = 0;
755  WORD32 i, j, k, temp;
756
757  temp = impd_read_bits_buf(it_bit_buff, 23);
758  if (it_bit_buff->error) return it_bit_buff->error;
759
760  dwnmix_instructions->downmix_id = (temp >> 16) & 0x7f;
761  dwnmix_instructions->target_channel_count = (temp >> 9) & 0x7f;
762  dwnmix_instructions->target_layout = (temp >> 1) & 0xff;
763  dwnmix_instructions->downmix_coefficients_present = temp & 1;
764
```

Write controllable values out-of-bounds

Case3 CVE-2019-2064

```
[details]
Ref#1
/external/libxaac/decoder/drc_src/impd_drc_dynamic_payload.c
618WORD32 impd_parse_filt_block(ia_bit_buf_struct* it_bit_buff,
619                             ia_filt_block_struct* str_filter_block,
620                             WORD32 block_count) {
621  //    WORD32 err = 0;
622  WORD32 k, j, temp;
623  ia_filt_ele_struct* str_filter_element;
624
625  for (j = 0; j < block_count; j++) {
626    str_filter_block->filter_element_count = impd_read_bits_buf(it_bit_buff, 6);
627    if (it_bit_buff->error) return it_bit_buff->error;
628
629    str_filter_element = &str_filter_block->str_filter_element[0];
630    for (k = 0; k < str_filter_block->filter_element_count; k++) {
631      temp = impd_read_bits_buf(it_bit_buff, 7);
632      if (it_bit_buff->error) return it_bit_buff->error;
633
634      str_filter_element->filt_ele_idx = (temp & 0x7E) >> 1;  ///  <=== oobw here
635      str_filter_element->filt_ele_gain_flag = temp & 1;  /// <=== oobw here
636      ;
637
638      if (str_filter_element->filt_ele_gain_flag) {
639        WORD32 bs_filter_element_gain;
640        bs_filter_element_gain = impd_read_bits_buf(it_bit_buff, 10);
641        if (it_bit_buff->error) return it_bit_buff->error;
642        str_filter_element->filt_ele_gain =   /// <=== oobw here
643            bs_filter_element_gain * 0.125f - 96.0f;
644      }
645
646      str_filter_element++;
647    }
648    str_filter_block++;
649  }
650  return (0);
651}
```

Number of elements in array str_filter_element is FILTER_ELEMENT_COUNT_MAX(16)

Filter_element_count is controllable, its range is [0..63]

str_filter_elements is a temp pointer pointed to the start address of member array str_filter_element

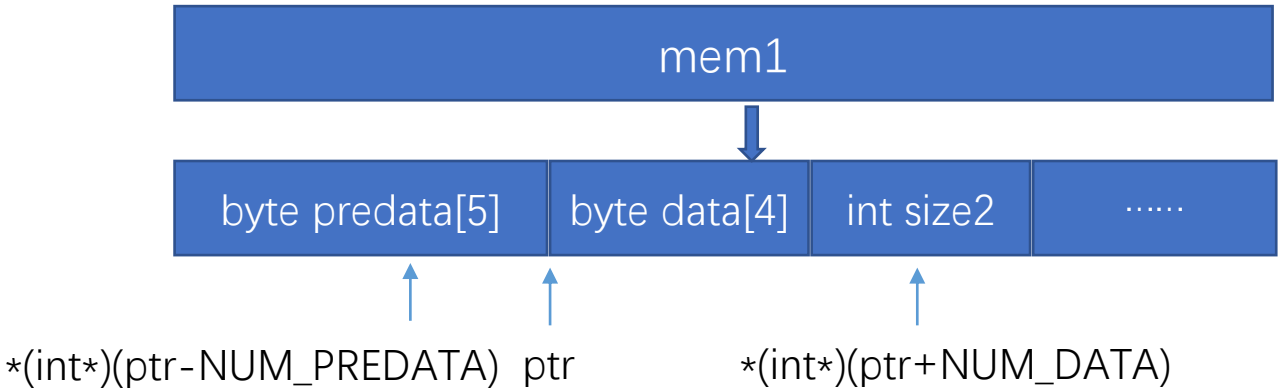
The for-loop goes through at most 64 cycles, which is larger than FILTER_ELEMENT_COUNT_MAX(16)

Many possible oobw issues

Pointer str_filter_elements will point to the next element in this array

False positive and performance

Strange accessing approaches will result in false positive



IOODetector will be sensitive and report an error

The overhead of IOODetector is about ~2.6x after we optimize our code to instrument as less code as possible

Further stories

The library has been marked as experimental and is no longer included in any production Android builds since Nov. 2018

Google introduced a Sanitizer named BoundSan to automatically instrument arrays to prevent overflows and fail safely.

BoundSan is enabled in 11 media codecs and throughout the Bluetooth stack for Android Q. By optimizing away a number of unnecessary checks the performance overhead was reduced to less than 1%.

https://security.googleblog.com/2019/05/queue-hardening-enhancements.html

THANKS

Q&A