ASAN - Address Sanitizer

List

Description of Address Sanitizer

- Address Sanitizer는 Google에서 제공하는 취약점 탐지 도구 입니다.
  - 버퍼 오버플로와 Dangling pointer에 접근 할 수 있는 메모리 손상을 발견 할 수 있습니다.
  - Address Sanitizer는 compiler instrumentation과 directly-mapped shadow memory를 기반으로 동작합니다.
  - AddressSanitizer는 현재 Clang (버전 3.1 이상) 및 GCC (버전 4.8 이상) 에서 구현됩니다 .
    - Android의 경우 clang-3.5 이상의 최신버전에서 구현됩니다.
  - 개발자들은 해당 옵션을 적용하면 프로그램 동작이 느려진다고 합니다.

"Dangling Pointer"란?

- 포인터가 해제된 메모리 영역을 가리키고 있는 포인터입니다.

Detection

- Use after free (dangling pointer dereference)
- Heap buffer overflow
- Stack buffer overflow
- Global buffer overflow
- Use after return
- Use after scope
- Initialization order bugs
- Memory leaks

The supported operating systems and architectures.

OS	x86	x86_64	ARM	ARM64	MIPS	MIPS64	PowerPC64
Linux	O	O			O	O	O
OS X	O	O					
iOS Simulator	O						
FreeBSD	O	O					

Android			O	O			

Others Sanitizer

Leak Sanitizer:

- Heap leak을 실행 중에 탐지합니다.

Thread Sanitizer

- ThreadSanitizer는 데이터 경쟁(data races)을 실행 중에 탐지합니다.

Memory Sanitizer

- MemorySanitizer는 초기화되지 않은 메모리의 참조를 실행 중에 탐지합니다.

UndefinedBehaviorSanitizer(UBSan)

- 정의되지 않은 행동을 실행 중에 탐지 합니다.

Using AddressSanitizer

Option

Option	Description
-fsanitize=address	AddressSanitizer 활성화
-fsanitize=kernel-address	Linux kernel용 AddressSanitizer 활성화
-fsanitize=thread	ThreadSanitizer 활성화,
-fsanitize=leak	LeakSanitizer 활성화, memory leak탐지
-fno-omit-frame-pointer	오류 메시지에서 조금더 상세한 스택 trace 결과를 출력
-fsanitize=undefined	UndefinedBehaviorSanitizer 활성화

Other options of -fsanitize

- https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html

Build

clang

```
clang -o test test.c -fsanitize=address
```

gcc

```
gcc -o test test.c -fsanitize=address
```

Build for Android

Android.mk

```
LOCAL_CFLAGS    := -fsanitize=address -fno-omit-frame-pointer
LOCAL_LDFLAGS   := -fsanitize=address
LOCAL_ARM_MODE := arm
```

Application.mk

```
NDK_TOOLCHAIN_VERSION=clang3.5
```

Application.mk

```
APP_ABI := armeabi armeabi-v7a x86
```

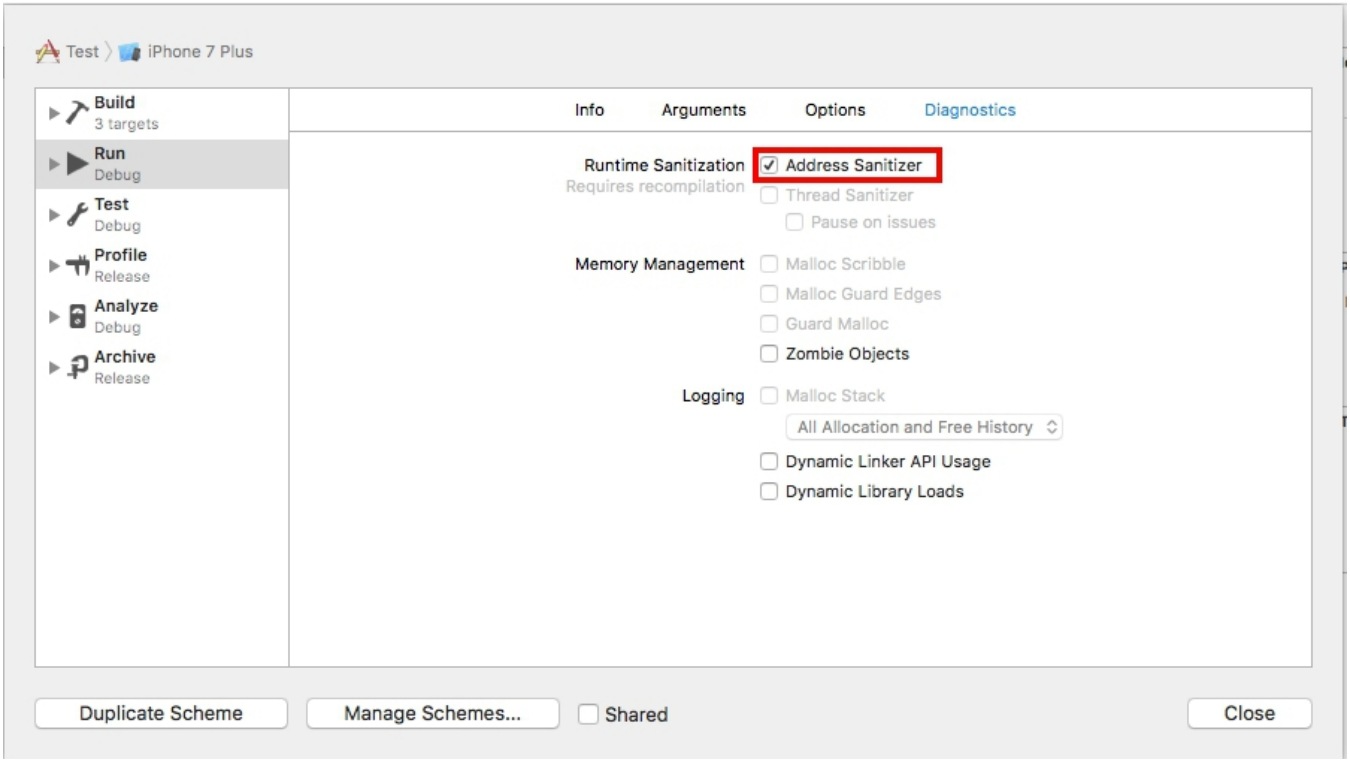
AddressSanitizerOnAndroid

- https://github.com/google/sanitizers/wiki/AddressSanitizerOnAndroid

Build for ios

- Menu → Product → Scheme → Edit Scheme → Diagnostics → Enable "Address Sanitizer"

Enable "Address Sanitizer" at the XCode



Example

UAF(User after free)

- 다음 코드는 해제된 Heap 영역의 값을 return 하고 있습니다.(User after free)

Example code - UAF.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char UAF(){
        char *a = new char[100];
        delete [] a;
        return a[1];
}

int main()
{
        UAF();
}
```

- 다음과 같이 프로그램 실행 후 취약성이 탐지되었습니다.

Detected a bug.

```
lazenca0x0@ubuntu:~/Documents/Definition/addressSanitizer$ g++ -o UAF UAF.cpp -fsanitize=address
lazenca0x0@ubuntu:~/Documents/Definition/addressSanitizer$ ./UAF
=================================================================
==21446==ERROR: AddressSanitizer: heap-use-after-free on address 0x60b00000af91 at pc 0x0000004007e7 bp
0x7fffea6837d0 sp 0x7fffea6837c0
READ of size 1 at 0x60b00000af91 thread T0
    #0 0x4007e6 in UAF() (/home/lazenca0x0/Documents/Definition/addressSanitizer/UAF+0x4007e6)
    #1 0x4007f9 in main (/home/lazenca0x0/Documents/Definition/addressSanitizer/UAF+0x4007f9)
    #2 0x7fe4206f082f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
    #3 0x4006b8 in _start (/home/lazenca0x0/Documents/Definition/addressSanitizer/UAF+0x4006b8)


0x60b00000af91 is located 1 bytes inside of 100-byte region [0x60b00000af90,0x60b00000aff4)
freed by thread T0 here:
    #0 0x7fe420b33caa in operator delete[](void*) (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x99caa)
    #1 0x4007ae in UAF() (/home/lazenca0x0/Documents/Definition/addressSanitizer/UAF+0x4007ae)
    #2 0x4007f9 in main (/home/lazenca0x0/Documents/Definition/addressSanitizer/UAF+0x4007f9)
    #3 0x7fe4206f082f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)

previously allocated by thread T0 here:
    #0 0x7fe420b336b2 in operator new[](unsigned long) (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x996b2)
    #1 0x400797 in UAF() (/home/lazenca0x0/Documents/Definition/addressSanitizer/UAF+0x400797)
    #2 0x4007f9 in main (/home/lazenca0x0/Documents/Definition/addressSanitizer/UAF+0x4007f9)
    #3 0x7fe4206f082f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)


SUMMARY: AddressSanitizer: heap-use-after-free ??:0 UAF()
Shadow bytes around the buggy address:
  0x0c167fff95a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff95b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff95c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff95d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff95e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c167fff95f0: fa fa[fd]fd fd fd fd fd fd fd fd fd fd fd fd fa
  0x0c167fff9600: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff9610: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff9620: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff9630: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff9640: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
==21446==ABORTING
lazenca0x0@ubuntu:~/Documents/Definition/addressSanitizer$
```

Heap Buffer Overflow

- 다음 코드는 할당된 Heap 영역을 벗어나는 곳으로 부터 값을 가져옵니다.

Example code - HeapBufferOverflow.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
        char *a = new char[100];
        printf("%c\n",a[101]);
        delete [] a;
}
```

- 다음과 같이 프로그램 실행 후 취약성이 탐지되었습니다.

Detected a bug.

```
lazenca0x0@ubuntu:~/Documents/Definition/addressSanitizer$ g++ -o HeapBufferOverflow HeapBufferOverflow.cpp -
fsanitize=address
lazenca0x0@ubuntu:~/Documents/Definition/addressSanitizer$ ./HeapBufferOverflow
=================================================================
==21644==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60b00000aff5 at pc 0x0000004008c8 bp
0x7ffd007ae090 sp 0x7ffd007ae080
READ of size 1 at 0x60b00000aff5 thread T0
    #0 0x4008c7 in main (/home/lazenca0x0/Documents/Definition/addressSanitizer/HeapBufferOverflow+0x4008c7)
    #1 0x7f8e9afb182f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
    #2 0x4007a8 in _start (/home/lazenca0x0/Documents/Definition/addressSanitizer/HeapBufferOverflow+0x4007a8)

0x60b00000aff5 is located 1 bytes to the right of 100-byte region [0x60b00000af90,0x60b00000aff4)
allocated by thread T0 here:
    #0 0x7f8e9b3f46b2 in operator new[](unsigned long) (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x996b2)
    #1 0x400887 in main (/home/lazenca0x0/Documents/Definition/addressSanitizer/HeapBufferOverflow+0x400887)
    #2 0x7f8e9afb182f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)


SUMMARY: AddressSanitizer: heap-buffer-overflow ??:0 main
Shadow bytes around the buggy address:
  0x0c167fff95a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff95b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff95c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff95d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff95e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c167fff95f0: fa fa 00 00 00 00 00 00 00 00 00 00 00[04]fa
  0x0c167fff9600: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff9610: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff9620: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff9630: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c167fff9640: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
==21644==ABORTING
lazenca0x0@ubuntu:~/Documents/Definition/addressSanitizer$
```

Stack Buffer Overflow

- 다음 코드는 할당된 Stack 영역을 벗어나는 곳으로 부터 값을 가져옵니다.

Example code - StackBufferOverflow.cpp

```
#include <stdio.h>

int main(){
        char stack[100];
        printf("%c\n",stack[101]);
        return 0;
}
```

- 다음과 같이 프로그램 실행 후 취약성이 탐지되었습니다.

Detected a bug.

```
lazenca0x0@ubuntu:~/Documents/Definition/addressSanitizer$ g++ -o StackBufferOverflow StackBufferOverflow.cpp -
fsanitize=address
lazenca0x0@ubuntu:~/Documents/Definition/addressSanitizer$ ./StackBufferOverflow
=================================================================
==21732==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffceb0c9de5 at pc 0x0000004009e0 bp
0x7ffceb0c9d50 sp 0x7ffceb0c9d40
READ of size 1 at 0x7ffceb0c9de5 thread T0
    #0 0x4009df in main (/home/lazenca0x0/Documents/Definition/addressSanitizer/StackBufferOverflow+0x4009df)
    #1 0x7f7e27cf882f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
    #2 0x400848 in _start (/home/lazenca0x0/Documents/Definition/addressSanitizer/StackBufferOverflow+0x400848)

Address 0x7ffceb0c9de5 is located in stack of thread T0 at offset 133 in frame
    #0 0x400925 in main (/home/lazenca0x0/Documents/Definition/addressSanitizer/StackBufferOverflow+0x400925)


  This frame has 1 object(s):
    [32, 132) 'stack' <== Memory access at offset 133 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow ??:0 main
Shadow bytes around the buggy address:
  0x10001d611360: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10001d611370: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10001d611380: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10001d611390: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10001d6113a0: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
=>0x10001d6113b0: 00 00 00 00 00 00 00 00 00 00 00 00[04]f4 f4 f4
  0x10001d6113c0: f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00
  0x10001d6113d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10001d6113e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10001d6113f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10001d611400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
==21732==ABORTING
lazenca0x0@ubuntu:~/Documents/Definition/addressSanitizer$
```

Related information

- https://github.com/google/sanitizers
- https://github.com/google/sanitizers/wiki/AddressSanitizer
- https://en.wikipedia.org/wiki/AddressSanitizer
- https://clang.llvm.org/docs/ThreadSanitizer.html
- https://source.android.com/devices/tech/debug/asan
- https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html
- https://clang.llvm.org/docs/LeakSanitizer.html
- https://clang.llvm.org/docs/AddressSanitizer.html
- https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html