

ASAN - Address Sanitizer

Unknown macro: 'html'

Excuse the ads! We need some help to keep our site up.

Unknown macro: 'html'

List

- [Description of Address Sanitizer](#)
 - [Detection](#)
 - [The supported operating systems and architectures.](#)
- [Others Sanitizer](#)
 - [Leak Sanitizer:](#)
 - [Thread Sanitizer](#)
 - [Memory Sanitizer](#)
 - [UndefinedBehaviorSanitizer\(UBSan\)](#)
- [Using AddressSanitizer](#)
 - [Option](#)
 - [Build](#)
 - [Build for Android](#)
 - [Build for ios](#)
- [Example](#)
 - [UAF\(User after free\)](#)
 - [Heap Buffer Overflow](#)
 - [Stack Buffer Overflow](#)
- [Related information](#)

Description of Address Sanitizer

- [Address Sanitizer Google](#)
 - [Dangling pointer](#)
 - [Address Sanitizer compiler instrumentation directly-mapped shadow memory](#)
 - [AddressSanitizer Clang \(3.1 \) GCC \(4.8 \)](#)
 - [Android clang-3.5](#)
 -

 "Dangling Pointer"?

-

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 leak .

Thread Sanitizer

- ThreadSanitizer (data races) .

Memory Sanitizer

- MemorySanitizer .

UndefinedBehaviorSanitizer(UBSan)

- .

Using AddressSanitizer

Option

Option	Description
-fsanitize=address	AddressSanitizer
-fsanitize=kernel-address	Linux kernel AddressSanitizer
-fsanitize=thread	ThreadSanitizer ,
-fsanitize=leak	LeakSanitizer , memory leak
-fno-omit-frame-pointer	trace
-fsanitize=undefined	UndefinedBehaviorSanitizer



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



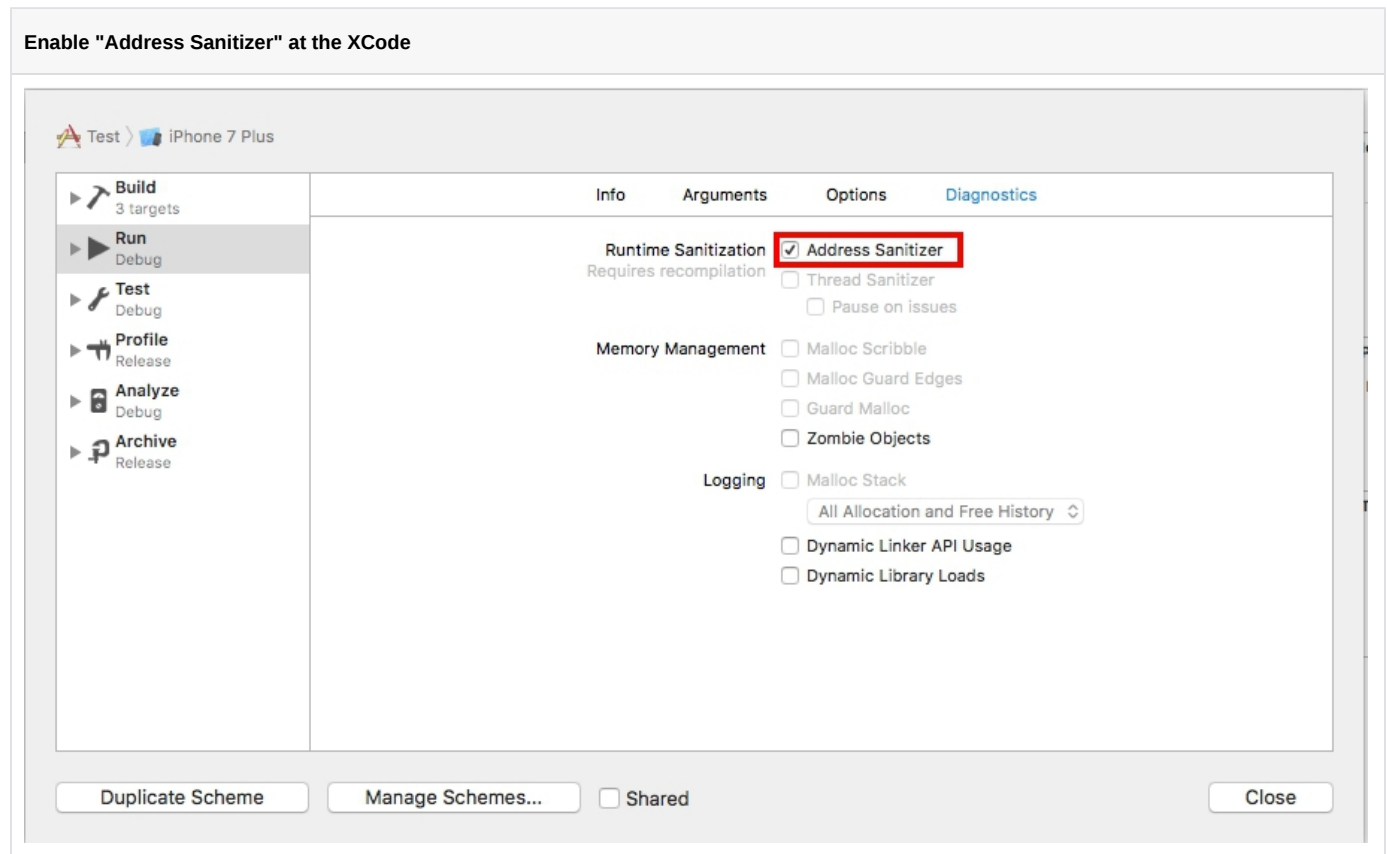
AddressSanitizerOnAndroid

- <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)

- Heap return .(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();
}
```

- .

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 fd
  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

- Heap .

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;
}
```

- .

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 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

- Stack .

Example code - StackBufferOverflow.cpp

```
#include <stdio.h>

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

- .

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 00 f1 f1 f1 f1
=>0x10001d6113b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  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>



Unknown macro: 'html'