Sanitizers

Address sanitizer

AddressSanitizer (aka ASan) is a memory error detector for C/C++. It finds:

- 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

Each bullet point links to an example (in C++) of what it looks like when ASan detects an error. They all start similar (`ERROR: AddressSanitizer`...) except for Memory leaks (`ERROR: LeakSanitizer...`). Two examples are provided below:

use-after-free.c

```
#include <stdlib.h>
int main() {
  char *x = (char*)malloc(10 * sizeof(char*));
  free(x);
  return x[5];
}
```

Address Sanitizer: heap use after free

```
==162==ERROR: AddressSanitizer: heap-use-after-free on address 0x607000000025 at pc 0x0000004c317b bp
0x7ffccaf1d220 sp 0x7ffccaf1d218
READ of size 1 at 0x607000000025 thread T0
    #0 0x4c317a in main (/mnt/c/Users/Pascal/netdef/a.out+0x4c317a)
    #1 0x7f60f59e00b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #2 0x41b2dd in _start (/mnt/c/Users/Pascal/netdef/a.out+0x41b2dd)

0x607000000025 is located 5 bytes inside of 80-byte region [0x607000000020,0x607000000070)
freed by thread T0 here:
    #0 0x49379d in free (/mnt/c/Users/Pascal/netdef/a.out+0x49379d)
    #1 0x4c3135 in main (/mnt/c/Users/Pascal/netdef/a.out+0x4c3135)
    #2 0x7f60f59e00b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)

previously allocated by thread T0 here:
    #0 0x493a1d in malloc (/mnt/c/Users/Pascal/netdef/a.out+0x493a1d)
    #1 0x4c3128 in main (/mnt/c/Users/Pascal/netdef/a.out+0x4c3128)
    #2 0x7f60f59e00b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)

SUMMARY: AddressSanitizer: heap-use-after-free (/mnt/c/Users/Pascal/netdef/a.out+0x4c317a) in main
```

memory-leak.c

```
#include <stdlib.h>
int main(void) {
   int *data = malloc(sizeof(int));
   return 0;
}
```

Address Sanitizer: memory leaks

```
==188==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 4 byte(s) in 1 object(s) allocated from:
    #0 0x493a1d in malloc (/mnt/c/Users/Pascal/netdef/a.out+0x493a1d)
    #1 0x4c3128 in main (/mnt/c/Users/Pascal/netdef/a.out+0x4c3128)
    #2 0x7fe30a1950b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)

SUMMARY: AddressSanitizer: 4 byte(s) leaked in 1 allocation(s).
```

UndefinedBehavior Sanitizer

UndefinedBehaviorSanitizer (UBSan) is a fast undefined behavior detector. UBSan modifies the program at compile-time to catch various kinds of undefined behavior during program execution.

A list of available checks can be found here. The flag -fsanitize=undefined will perform all checks except for `float-divide-by-zero`, `unsigned-integer-overflow`, `implicit-conversion`, `local-bounds` and the `nullability-*` group of checks.

Depending on the severity, error detection look like the following:

division-by-zero.c

```
int main() {
    int test = 5;
    int zero = 0;
    return test/zero;
}
```

UndefinedBehavior: division by zero

```
test.c:4:13: runtime error: division by zero
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior test.c:4:13 in
UndefinedBehaviorSanitizer:DEADLYSIGNAL
==120==ERROR: UndefinedBehaviorSanitizer: FPE on unknown address 0x00000042370f (pc 0x00000042370f bp
0x7fff8e7e9860 sp 0x7fff8e7e9850 T120)
    #0 0x42370f in main (/mnt/c/Users/Pascal/netdef/a.out+0x42370f)
    #1 0x7f941dd3c0b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #2 0x40330d in _start (/mnt/c/Users/Pascal/netdef/a.out+0x40330d)

UndefinedBehaviorSanitizer can not provide additional info.
SUMMARY: UndefinedBehaviorSanitizer: FPE (/mnt/c/Users/Pascal/netdef/a.out+0x42370f) in main
==120==ABORTING
```

overflow.c

```
int main() {
    int one = 578765432;
    int two = 567654365;
    return one*two;
}
```

UndefinedBehavior: signed integer overflow

```
test.c:3:5: runtime error: signed integer overflow: 2147483647 + 1 cannot be represented in type 'int'
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior test.c:3:5 in
```

Thread Sanitizer

`ThreadSanitizer` (aka TSan) is a data race detector for C/C++. Data races are one of the most common and hardest to debug types of bugs in concurrent systems. A data race occurs when two threads access the same variable concurrently and at least one of the accesses is write. An example is provided below:

data-race.c

```
#include <pthread.h>
int global = 0;

void *threadfunc(void *x) {
  global = global + 1;
  return x;
}

int main() {
  pthread_t t1;
  pthread_t t2;
  pthread_create(&t1, NULL, threadfunc, NULL);
  pthread_create(&t2, NULL, threadfunc, NULL);
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);
  return global;
}
```

Thread Sanitizer: data race

```
WARNING: ThreadSanitizer: data race (pid=371)
  Write of size 4 at 0x000000f12358 by thread T2:
    #0 threadfunc <null> (a.out+0x4b18ee)

  Previous write of size 4 at 0x000000f12358 by thread T1:
    #0 threadfunc <null> (a.out+0x4b18ee)

  Location is global 'global' of size 4 at 0x000000f12358 (a.out+0x000000f12358)

  Thread T2 (tid=374, running) created by main thread at:
    #0 pthread_create <null> (a.out+0x424a2b)
    #1 main <null> (a.out+0x4b1960)

  Thread T1 (tid=373, finished) created by main thread at:
    #0 pthread_create <null> (a.out+0x424a2b)
    #1 main <null> (a.out+0x4b1941)

SUMMARY: ThreadSanitizer: data race (/mnt/c/Users/Pascal/netdef/a.out+0x4b18ee) in threadfunc
==================
ThreadSanitizer: reported 1 warnings
```

Memory Sanitizer

MemorySanitizer (MSan) is a detector of uninitialized memory reads in C/C++ programs.

uninit-mem.c

```
int main() {
    int x;
    return x;
}
```

Memory Sanitizer

```
==247==WARNING: MemorySanitizer: use-of-uninitialized-value
    #0 0x495056 in main (/mnt/c/Users/Pascal/netdef/a.out+0x495056)
    #1 0x7f43b7ed10b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #2 0x41c26d in _start (/mnt/c/Users/Pascal/netdef/a.out+0x41c26d)

SUMMARY: MemorySanitizer: use-of-uninitialized-value (/mnt/c/Users/Pascal/netdef/a.out+0x495056) in main
```