Enable MSan & TSan on Go for Arm platform

Fangming.Fang@arm.com

Linaro connect
Bangkok 2019

Agenda

- MSan
  - Msan overview
  - Shadow memory
  - Shadow propagation
  - Instrumentation in Go
  - Demo
- TSan
  - TSan overview
  - Algorithm
  - Instrumentation in Go
  - Demo

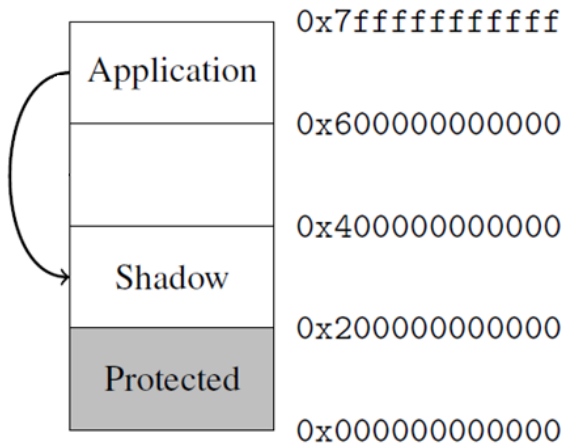
MemorySanitizer (Msan) overview

Use of uninitialized memory (UUM) is such an issue that is hard to reproduce and debug. So MemorySanitizer, MSan for short, is introduced to address that issue.

- MSan is a dynamic tool that detects uses of uninitialized memory in C. The tool is based on compile time instrumentation and relies on bitprecise shadow memory at run-time. Shadow propagation technique is used to avoid false positive reports on copying of uninitialized memory.

- MSan Is part of LLVM trunk and implemented as an LLVM optimization pass.

Shadow memory



1 application bit = 1 shadow bit

- 1 = poisoned (uninitialized)
- 0 = clean (initialized)

Shadow = Addr & ShadowMask

- The ShadowMask constant is platform-specific

Shadow propagation

Basic shadow propagation rules

$A = \texttt{load}\ P$	check P', $A' = \texttt{load}\ (P\ \&\ \text{ShadowMask})$
$\texttt{store}\ P,\ A$	check P', $\texttt{store}\ (P\ \&\ \text{ShadowMask}),\ A'$
$A = const$	$A' = 0$
$A = undef$	$A' = 0xff$
$A = B\ \&\ C$	$A' = (B'\ \&\ C')\ \|\ (B\ \&\ C')\ \|\ (B'\ \&\ C)$
$A = B\ \|\ C$	$A' = (B'\ \&\ C')\ \|\ (\sim B\ \&\ C')\ \|\ (B'\ \&\ \sim C)$
$A = B\ \text{xor}\ C$	$A' = B'\ \|\ C'$
$A = B \ll C$	$A' = (\text{sign-extend}(C' \neq 0))\ \|\ (B' \ll C)$

Approximate propagation

A = B + C $\Longrightarrow$ A' = B' | C'

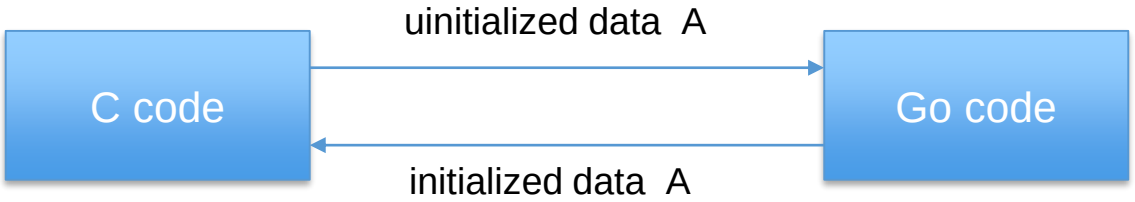
Exact propagation logic is way too complex.
Bitwise OR is common propagation logic.

- Never makes a value "less poisoned".
- Never makes a poisoned value clean.

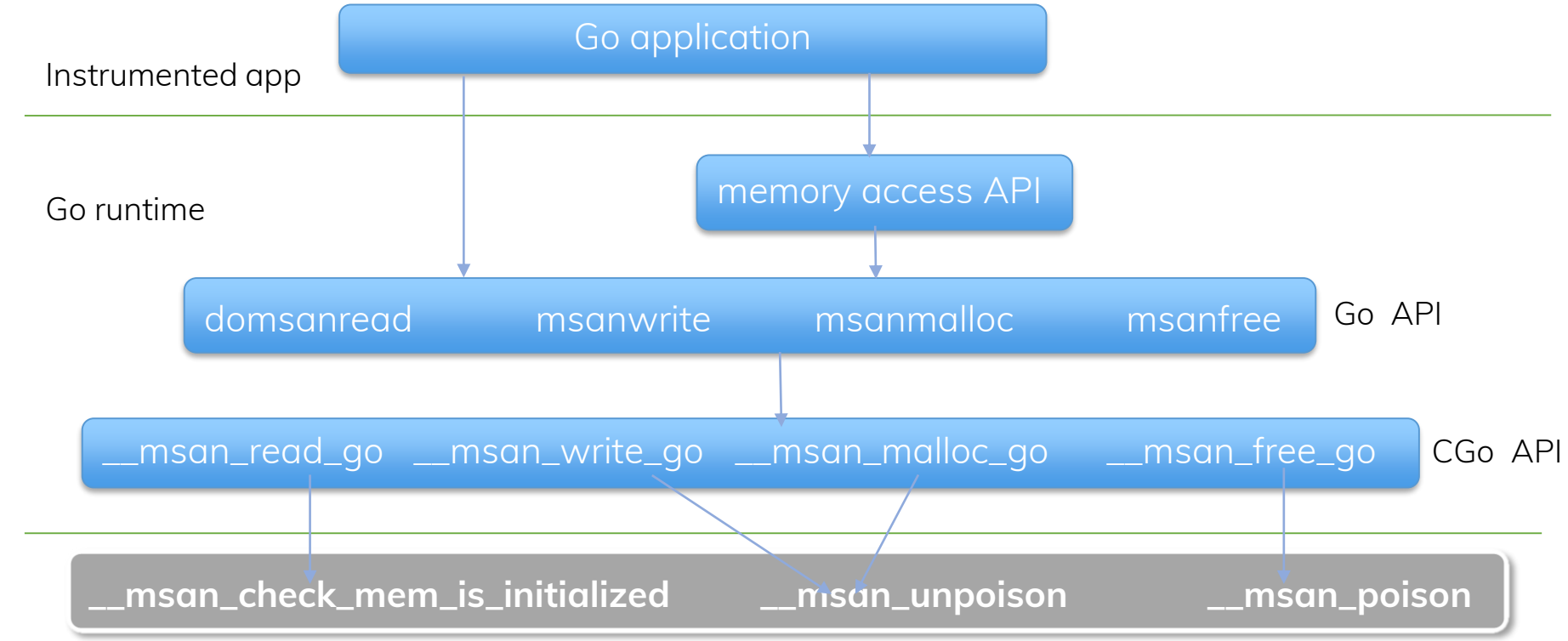
CGo

MSan does not apply to pure Go application where all memory is initialized. But Go can call C code by the way called CGo. People can use CGo to pass memory back and forth between C and Go.

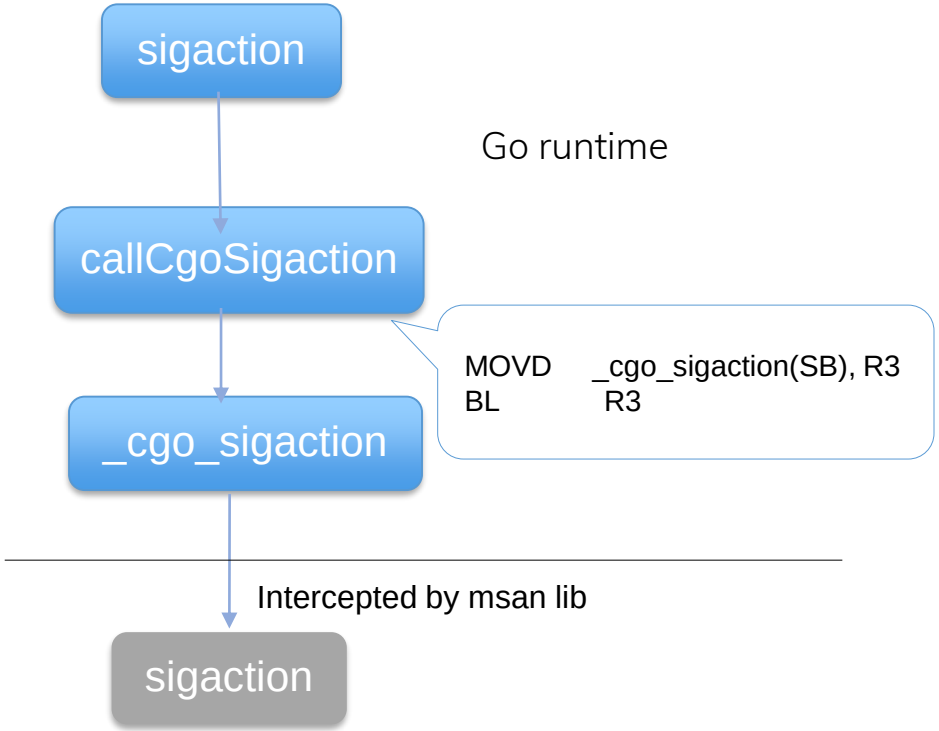
False positive will happen if C creates some uninitialized memory, passes that memory to Go, Go fills it in and returns to C.



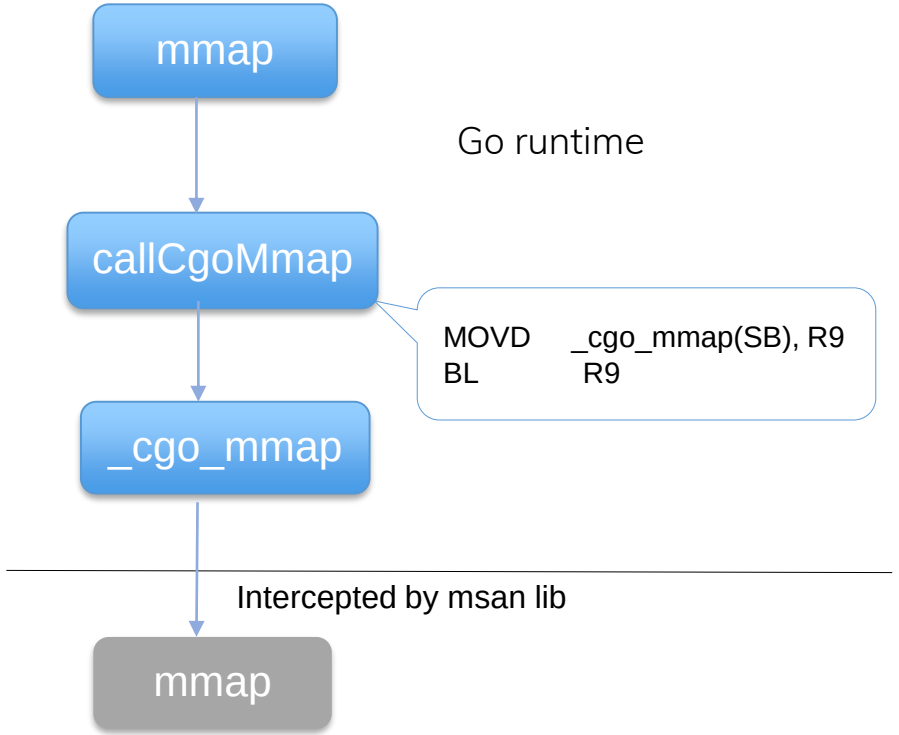
MSan framework in Go



Sigaction



Mmap



Demo

Result in all elements in array '**a**' uninitialized

```
// C code
void f(int32_t *p, int n) {
  int32_t * volatile q = (int32_t *)malloc(sizeof(int32_t) * n);
  memcpy(p, q, n * sizeof(*p));
  free(q);
}
void g(int32_t *p, int n) {
  if (p[4] != 1) {    // take code branch depending on the fifth element uninitialized
    abort();
  }
}
```

```
// Go code
func main() {
        a := make([]int32, 10)
        C.f((*C.int32_t)(unsafe.Pointer(&a[0])), C.int(len(a)))
        a[3] = 1    // Initialize the fourth element
        C.g((*C.int32_t)(unsafe.Pointer(&a[0])), C.int(len(a)))
}
```

CC=clang go test **-msan** –c msan_fail.go

```
fanfan01@fanfan01-01-arm-vm:~/ci-scripts/golang/misc/cgo/testsanitizers/src$ ./msan_fail
==14953==WARNING: MemorySanitizer: use-of-uninitialized-value
    #0 0xaaaaab85fdaf  (/home/fanfan01/ci-scripts/golang/misc/cgo/testsanitizers/src/msan_fail+0x11cdaf)
    #1 0xaaaaab859a97  (/home/fanfan01/ci-scripts/golang/misc/cgo/testsanitizers/src/msan_fail+0x116a97)

SUMMARY: MemorySanitizer: use-of-uninitialized-value (/home/fanfan01/ci-scripts/golang/misc/cgo/testsanitizers/src/msan_fail+0x11cdaf)
Exiting
```

ThreadSanitizer (TSan)

Data race is that two concurrent accesses to a shared location, at least one of them for writing. It is kind of threading bugs and hard to reproduce and debug. So ThreadSanitizer, TSan for short, is developed to address this issue.

TSan is a dynamic detector of data race and a subproject of LLVM project. It observes following events generated by the running program.

- Memory access envents : READ / WRITE
- Synchronization events:
  - Lock events : WrLock, RdLock, WrUnlock, RdUnlock
  - Happens-before events : SIGNAL, WAIT

Algorithm – Hybrid state machine

Global state: the information about the synchronization events that have been observed so far (lock sets, happens-before arcs).

Per-ID state: the information about each memory location of the running program. Per-ID state consists of two segment sets: the writer segment set SSWr and the reader segment set SSRd. SSWr of a given ID is a set of segments where the writes to this ID appeared. SSRd is a set of all segments where the reads from the given ID appeared.

Handle read/write events and Check for race

```
HANDLE-READ-OR-WRITE-EVENT(IsWrite, Tid, ID)
 1  ▷ Handle event READ Tid(ID) or WRITE Tid(ID)
 2  (SS_Wr, SS_Rd) ← GET-PER-ID-STATE(ID)
 3  Seg ← GET-CURRENT-SEGMENT(Tid)
 4  if IsWrite
 5     then ▷ WRITE event: update SS_Wr and SS_Rd
 6          SS_Rd ← {s : s ∈ SS_Rd ∧ s ⋠ Seg}
 7          SS_Wr ← {s : s ∈ SS_Wr ∧ s ⋠ Seg} ∪ {Seg}
 8     else ▷ READ event: update SS_Rd
 9          SS_Rd ← {s : s ∈ SS_Rd ∧ s ⋠ Seg} ∪ {Seg}
10  SET-PER-ID-STATE(ID, SS_Wr, SS_Rd)
11  if IS-RACE(SS_Wr, SS_Rd)
12     then ▷ Report a data race on ID
13          REPORT-RACE(IsWrite, Tid, Seg, ID)
```

```
IS-RACE(SS_Wr, SS_Rd)
 1  ▷ Check if we have a race.
 2  N_W ← SEGMENT-SET-SIZE(SS_Wr)
 3  for i ← 1 to N_W
 4    do W_1 ← SS_Wr[i]
 5       LS_1 ← GET-WRITER-LOCK-SET(W_1)
 6       ▷ Check all write-write pairs.
 7       for j ← i + 1 to N_W
 8         do W_2 ← SS_Wr[j]
 9            LS_2 ← GET-WRITER-LOCK-SET(W_2)
10            ASSERT(W_1 ⋠ W_2 and W_2 ⋠ W_1)
11            if LS_1 ∩ LS_2 = ∅
12              then return true
13       ▷ Check all write-read pairs.
14       for R ∈ SS_Rd
15         do LS_R ← GET-READER-LOCK-SET(R)
16            if W_1 ⋠ R and LS_1 ∩ LS_R = ∅
17              then return true
18  return false
```

TSan runtime library

TSan, as a runtime library, is implemented in compiler-rt which is a part of LLVM project.

TSan requires specific memory layout to implement TSan algorithm. And the memory layout is also arch-related. In order to support TSan on arm, we have to add memory mapping for arm in Tsan library.

TSan runtime library provides many APIs to record memory access events and synchronization envents.

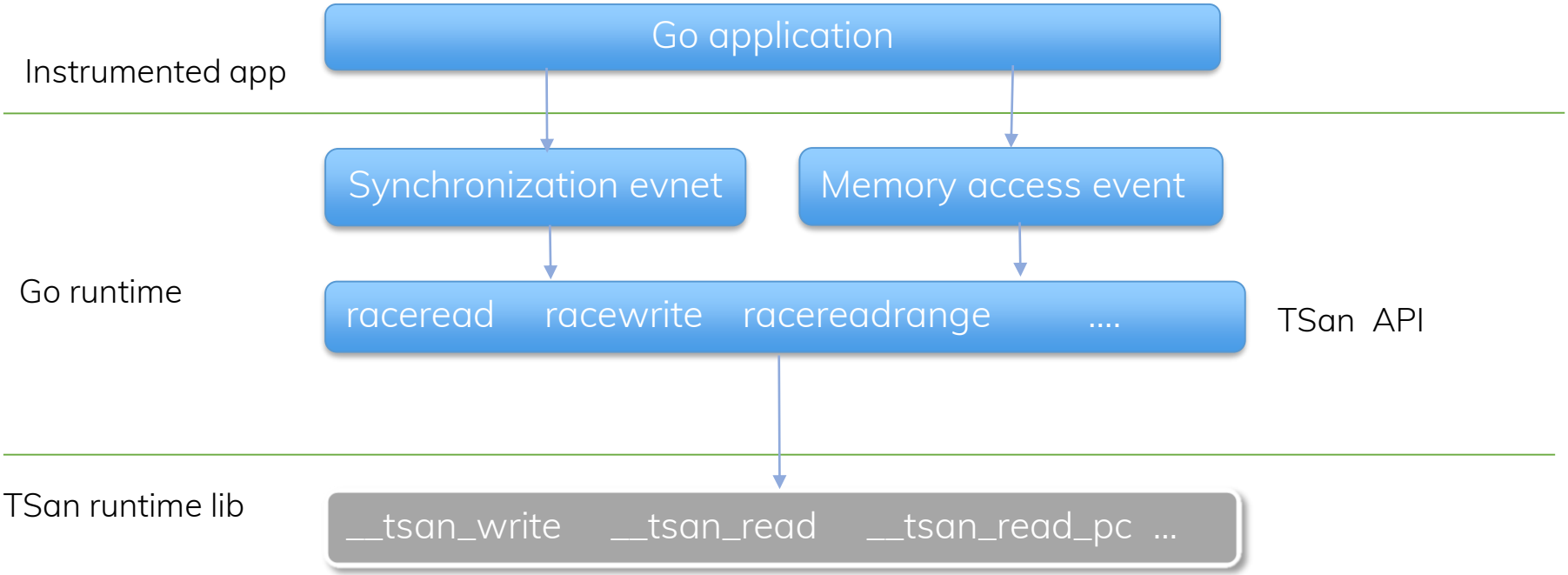
TSAN API - Memory access

```
void __tsan_read(ThreadState *thr, void *addr, void *pc)
void __tsan_write(ThreadState *thr, void *addr, void *pc)
void __tsan_read_pc(ThreadState *thr, void *addr, void *callpc, void *pc)
void __tsan_write_pc(ThreadState *thr, void *addr, void *callpc, void *pc)
void __tsan_read_range(ThreadState *thr, void *addr, uintptr size, void *pc)
void __tsan_write_range(ThreadState *thr, void *addr, uintptr size, void *pc)
```

TSAN API - Atomic operation

void __tsan_go_atomic32_load(ThreadState *thr, uptr cpc, uptr pc, u8 *a)
void __tsan_go_atomic64_load(ThreadState *thr, uptr cpc, uptr pc, u8 *a)
void __tsan_go_atomic32_store(ThreadState *thr, uptr cpc, uptr pc, u8 *a)
void __tsan_go_atomic64_store(ThreadState *thr, uptr cpc, uptr pc, u8 *a)
void __tsan_go_atomic32_exchange(ThreadState *thr, uptr cpc, uptr pc, u8 *a)
void __tsan_go_atomic64_exchange(ThreadState *thr, uptr cpc, uptr pc, u8 *a)
void __tsan_go_atomic32_fetch_add(ThreadState *thr, uptr cpc, uptr pc, u8 *a)
void __tsan_go_atomic64_fetch_add(ThreadState *thr, uptr cpc, uptr pc, u8 *a)
void __tsan_go_atomic32_compare_exchange(ThreadState *thr, uptr cpc, uptr pc, u8 *a)
void __tsan_go_atomic64_compare_exchange(ThreadState *thr, uptr cpc, uptr pc, u8 *a)

TSan framework in Go



Demo

Instrument application with TSan

```
func main() {
	done := make(chan bool)
	x := 0
	startRacer(&x, done)
	store(&x, 43)  // write x
	<-done
}

func store(x *int, v int) {
	*x = v
}

func startRacer(x *int, done chan bool) {
	go racer(x, done)
}
                // start a go routine
func racer(x *int, done chan bool) {
	time.Sleep(10*time.Millisecond)
	store(x, 42)  // write x
	done <- true
}
```

```
fanfan01@fanfan01-01-arm-vm:~/test$ ~/ci-scripts/golang/bin/go build -race main_go.go
fanfan01@fanfan01-01-arm-vm:~/test$ ./main_go
==================
WARNING: DATA RACE
Write at 0x004000016080 by goroutine 5:
  main.racer()
      /home/fanfan01/test/main_go.go:11 +0x30

Previous write at 0x004000016080 by main goroutine:
  main.main()
      /home/fanfan01/test/main_go.go:11 +0x74

Goroutine 5 (running) created at:
  main.startRacer()
      /home/fanfan01/test/main_go.go:14 +0x3c
  main.main()
      /home/fanfan01/test/main_go.go:6 +0x68
==================
Found 1 data race(s)
fanfan01@fanfan01-01-arm-vm:~/test$
```

Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org

Linaro
connect
Bangkok 2019