CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

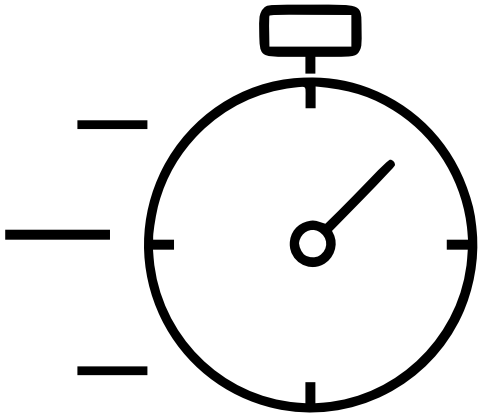
FAU

A Binary-level Thread Sanitizer or Why Sanitizing on the Binary Level is Hard

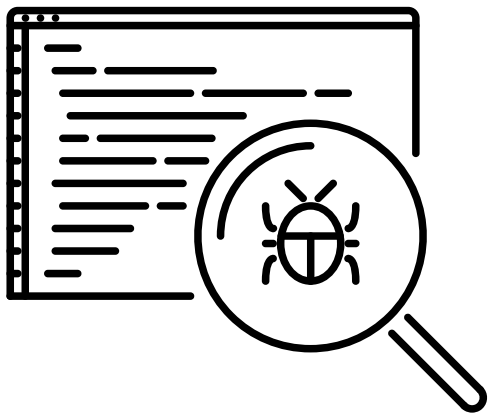
Joschua Schilling[1], Andreas Wendler[2], Philipp Görz[1], Nils Bars[1], Moritz Schloegel[1], Thorsten Holz[1]

1: CISPA Helmholtz Center for Information Security 2: Friedrich-Alexander-Universität Erlangen-Nürnberg

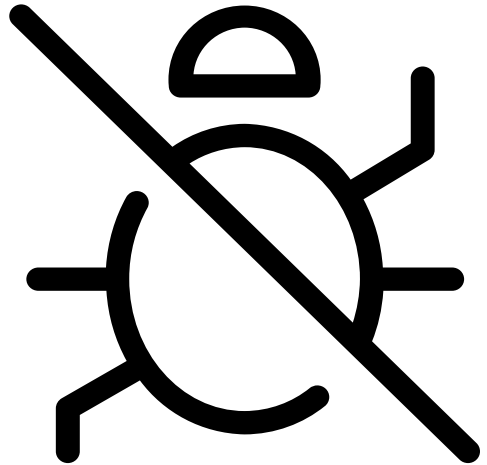
Sanitizers in a nutshell



Runtime checks



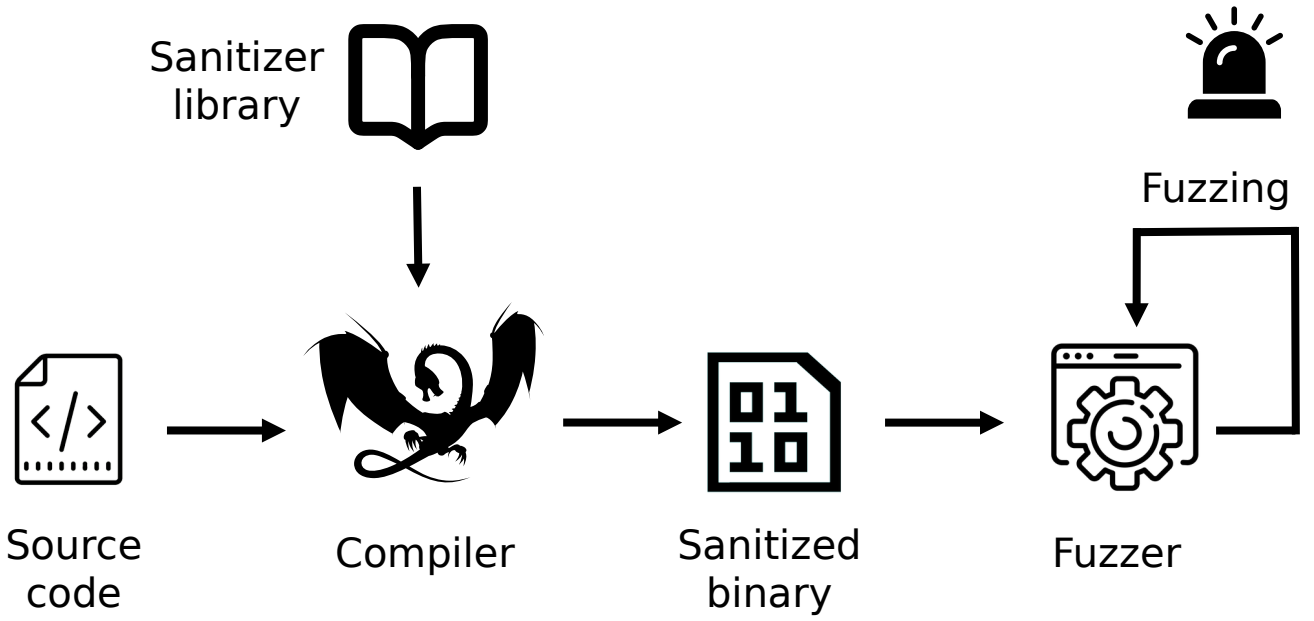
Detect errors



Facilitate debugging

How to use sanitizers?

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




```
WARNING: ThreadSanitizer: data race (pid=115060)
  Read of size 4 at 0x000000407314 by thread T1:
    #0 thrdFn() /home/joschua/Desktop/CISPA/Projekte/binary-
    #1 void std::__invoke_impl<void, void (*)()>(std::__invo
    #2 std::__invoke_result<void (*)()>::type std::__invoke<
    #3 void std::thread::_Invoker<std::tuple<void (*)()> >::
    #4 std::thread::_Invoker<std::tuple<void (*)()> >::opera
    #5 std::thread::_State_impl<std::thread::_Invoker<std::t
    #6 <null> <null> (libstdc++.so.6+0xe62b2)

  Previous write of size 4 at 0x000000407314 by main thread:
    #0 main /home/joschua/Desktop/CISPA/Projekte/binary-tsan/

  Location is global 'a' of size 4 at 0x000000407314 (test.ts

  Thread T1 (tid=115062, running) created by main thread at:
    #0 pthread_create ../../../../src/libsanitizer/tsan/tsan_
    #1 std::thread::_M_start_thread(std::unique_ptr<std::thre
    #2 main /home/joschua/Desktop/CISPA/Projekte/binary-tsan/

SUMMARY: ThreadSanitizer: data race /home/joschua/Desktop/CI
```

However: Most sanitizers are not available for binaries

Challenges in binary sanitizers

Information loss

- Control flow
- Type information
- Signedness
- Debug information
- Memory order
- Atomicity

Conceptual differences

- Different representations (source code, IR, assembly)
- Number of registers (limited vs. unlimited)
- SSA vs non SSA
- RISC vs CISC
- Memory model

Assessment of existing sanitizers

Address Sanitizer (ASAN)

✅

Detects: memory corruption

Runtime library
Minimal invasive instrumentation
Existing prototype: ASan-RetroWrite

Memory Sanitizer (MSAN)

❌

Detects: uninitialized memory

Runtime library (unfit for binaries)
Complex instrumentation

Undefined Behavior Sanitizer (UBSAN)

Detects: undefined behavior

Concept of UB does not apply to binaries
11/28 checks (partly) possible

❌

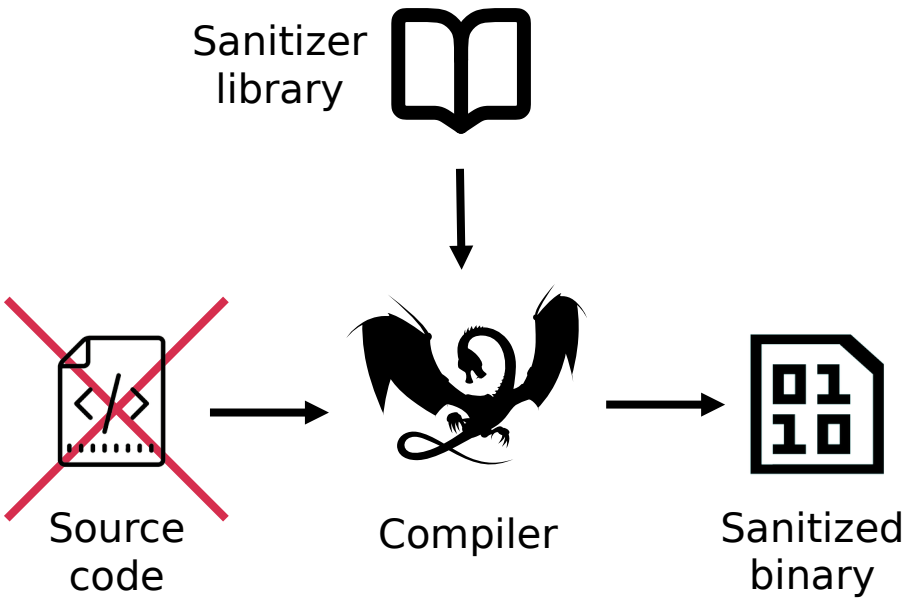
Thread Sanitizer (TSAN)

Detects: data races

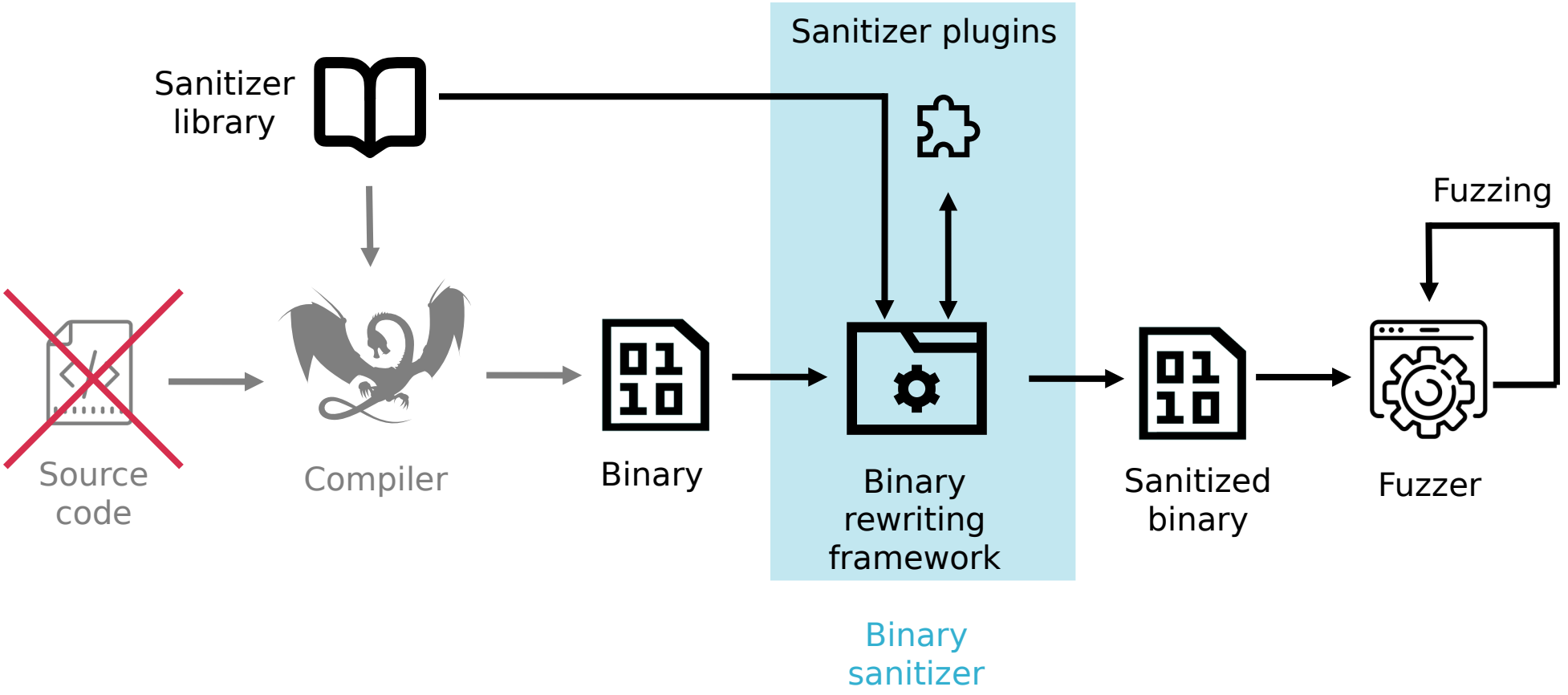
Runtime library
Minimal invasive instrumentation
Memory order/atomicity information partly lost

❌

Binary sanitizers architecture

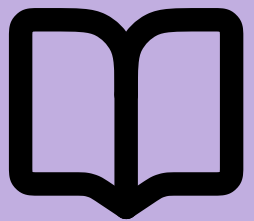


Binary sanitizers architecture



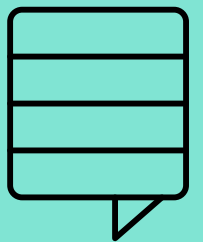
BinTSAN

TSAN library



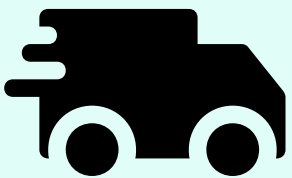
Instrumentation of memory accesses

Call stack recording

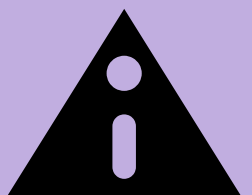


State saving/restoring

Optimizations



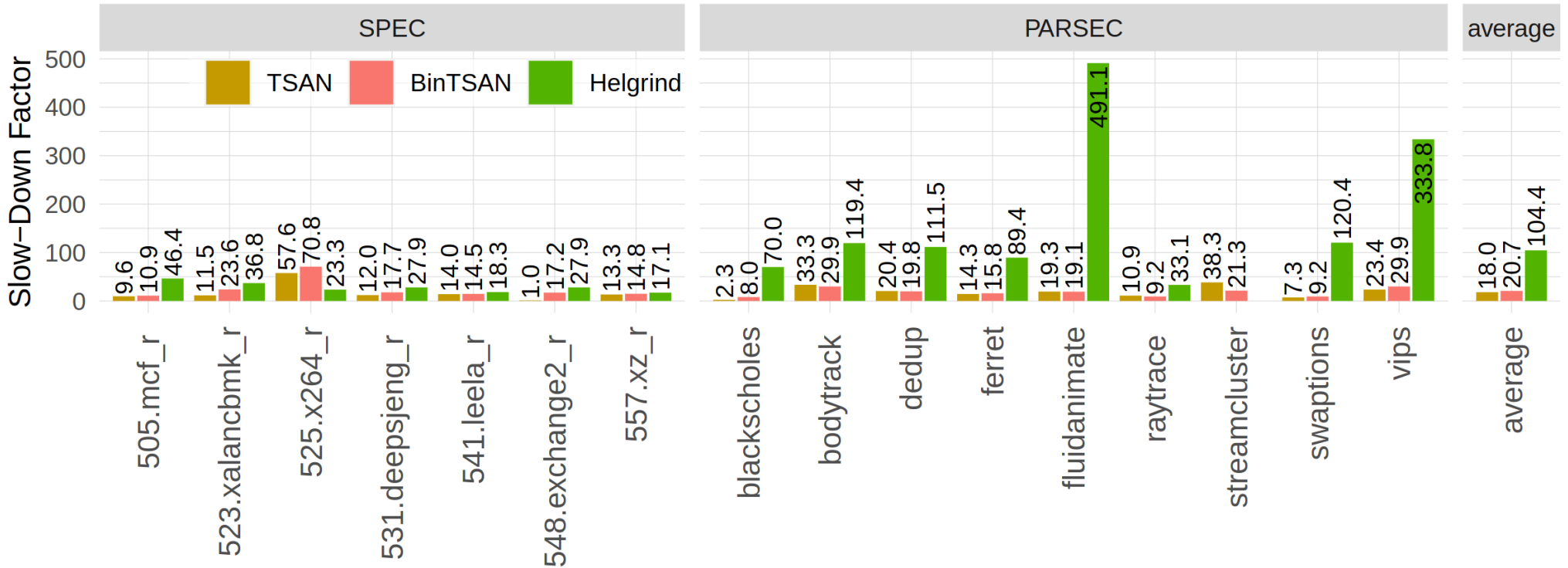
Heuristics to overcome information loss



Evaluation: Error detection

Target	TSAN	BinTSAN (FP)	Helgrind (FP)
ffmpeg	290	95 (830)	74 (2,120)
bodytrack	6	1 (6)	1 (9)
QtNotepad	3	3 (1)	3 (3)
streamcluster	4	4 (0)	4 (1)
axel	4	3 (0)	3 (3)
bifrost	3	2 (4)	2 (21)
vips	3	3 (0)	2 (2)
lrzip	2	2 (0)	2 (0)
rar	N/A	1 (0)	0 (0)
unrar	1	1 (0)	0 (0)
ferret	1	1 (0)	0 (2)
fluidanimate	1	1 (0)	1 (1)

Evaluation: Performance



Code

- Code is publicly available!
- Interested? Talk to us!

github.com/CISPA-SysSec/binary-tsan