ASE 2026 · INDUSTRY SHOWCASE · MUNICH

Understanding the Performance-Effectiveness Trade-Offs of AddressSanitizer in Real-World Programs: An Empirical Study

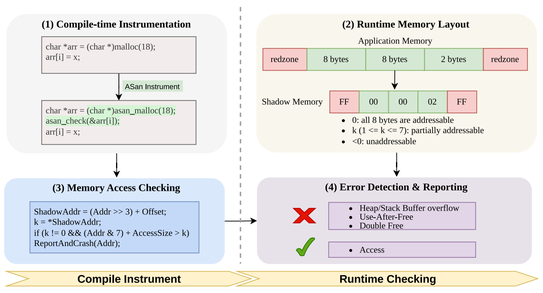
Jiaxin Xi[1], Cheng Wen[1]*, Zhiwei Lin[1], Yuandao Cai[2],
Jie Su[1], Bin Yu[3], Zhuohua Li[1], Shengchao Qin[3]

[1] Guangzhou Institute of Technology, Xidian University
[2] Hong Kong University of Science and Technology
[3] School of Computer Science and Technology, Xidian University

XIDIAN UNIVERSITY

PAPER DETAILS

Why Study ASan's Trade-Offs?



Memory safety matters. In C/C++, out-of-bounds accesses and use-after-free can crash programs, corrupt data, or enable attacks.

ASan exposes these errors. AddressSanitizer inserts checks at compile time and tracks memory at runtime. CPU and memory costs can limit continuous integration and production-like tests.

The research gap. Different workloads, toolchains, and settings obscure the detection and deployment trade-offs of optimized sanitizers.

Four questions: cost sources (RQ1), parameter tuning (RQ2), build optimization (RQ3), and practical cross-tool trade-offs (RQ4).

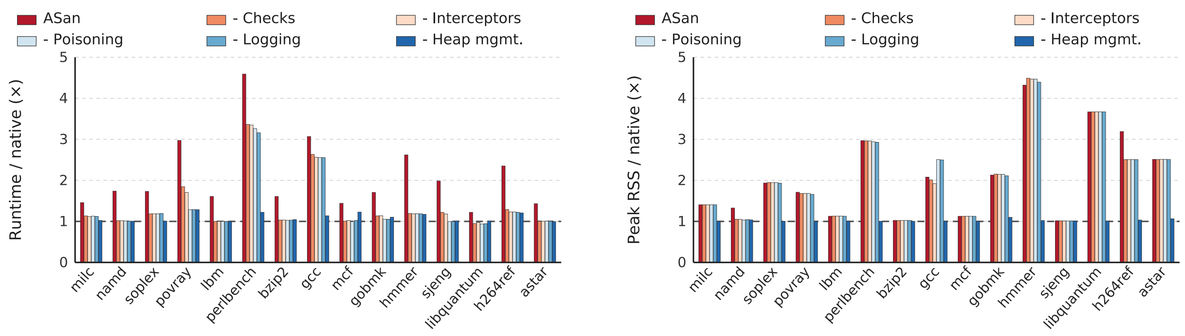
Checks and memory metadata expose invalid accesses.

Study: default/tuned ASan and seven optimized sanitizers; repeated runs on x86-64 Linux, Intel i9-13900K, 128 GB RAM.

Workloads: SPEC CPU2006 (17 programs: runtime and peak RSS); Juliet (9,301 buggy + 9,301 non-buggy cases); MSET (232 error patterns); 21 CVE PoCs; MySQL 5.7.19 (build, initialization, OLTP read/write, 1-50 clients).

RQ1: Component-Wise Impact

Cumulative Ablation Order: checks → interceptors → poisoning → logging/symbolization → heap management; removals accumulate.



Left: runtime. Right: peak resident memory (RSS). Both use 15 shared SPEC programs and a common native baseline.

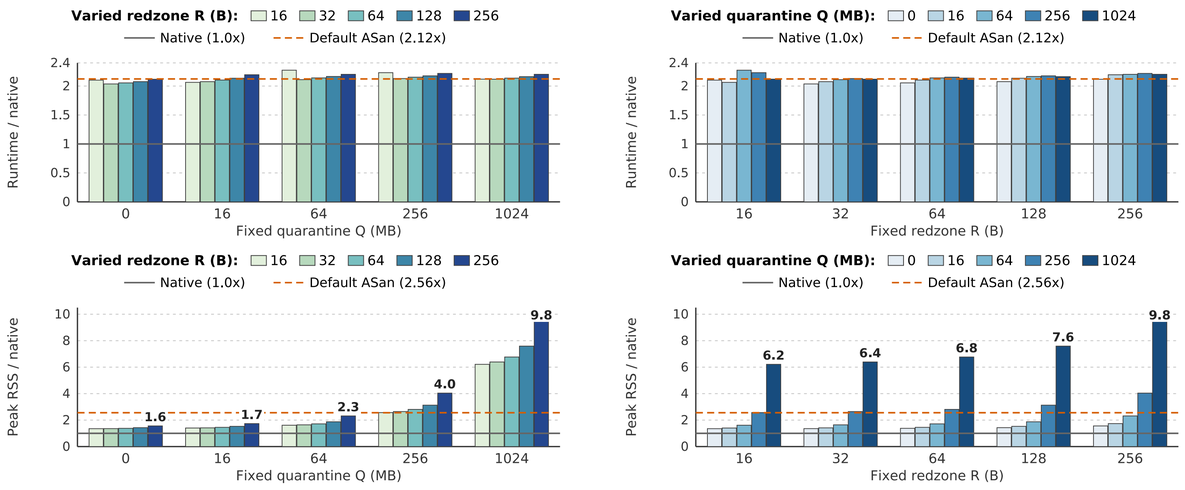
Runtime: removing checks lowers perlbench by 26.8% (4.59× to 3.36× native). **Memory:** final heap-management removal lowers mean RSS from 2.06× to 1.02× native across the same 15 completed programs.

Juliet	Default	1. Checks	2. Interceptors	3. Poisoning	4. Logging	5. Heap
Found (TP)	9,245	7,306	7,059	3,871	3,871	2,296
Missed (FN)	56	1,995	2,242	5,430	5,430	7,005

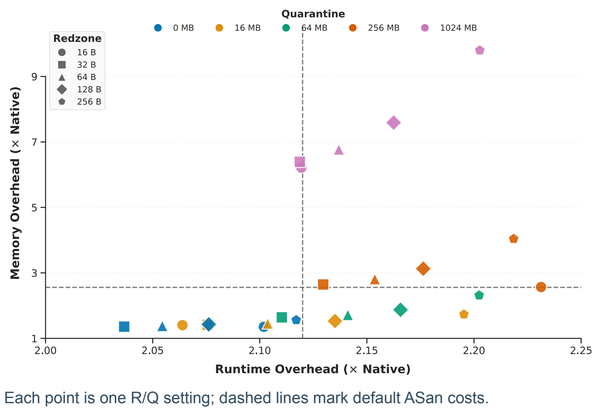
RQ1 answer: checks cost runtime; late heap removal cuts RSS; detection declines. These are order-dependent, not independent effects.

RQ2: Parameter Sensitivity

25 settings: redzone R = 16/32/64/128/256 B; quarantine Q = 0/16/64/256/1024 MB. Redzones guard memory boundaries; quarantine retains freed blocks to delay reuse and expose use-after-free.



Columns: redzone / quarantine; rows: runtime / peak RSS. Across Q, mean runtime is within ±0.2× of default (2.12×); peak RSS reaches ≈9.8× native.



Each point is one R/Q setting; dashed lines mark default ASan costs.

Why R16 / Q16?

16 B minimum redzone
16 MB quarantine

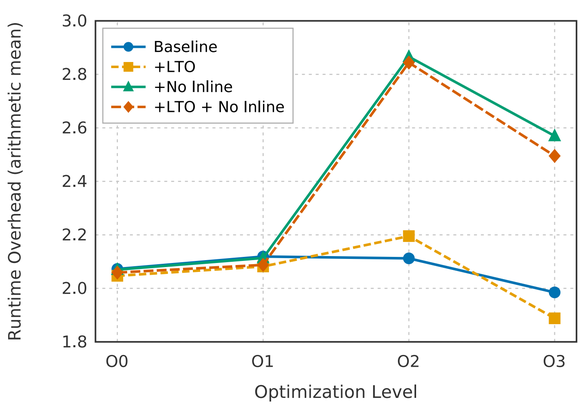
R32/Q0 and R16/Q16 lower peak RSS with **no extra Juliet misses**. R16/Q16: 9,245 reports, 56 misses.

We retain R16/Q16 because it preserves a **nonzero quarantine**, delaying reuse of freed memory.

The scatter measures cost, not detection. Juliet does not establish preservation of delayed use-after-free detection.

RQ2 answer: include tuned ASan as a baseline. Re-test R16/Q16 on target-project and delayed-use-after-free workloads.

RQ3: Compiler-Level Optimization



Native references match each optimization level. Baseline denotes default ASan.

FP: frame pointer.
LTO: Link-Time Optimization across source files.
NoInline: restricted function inlining.

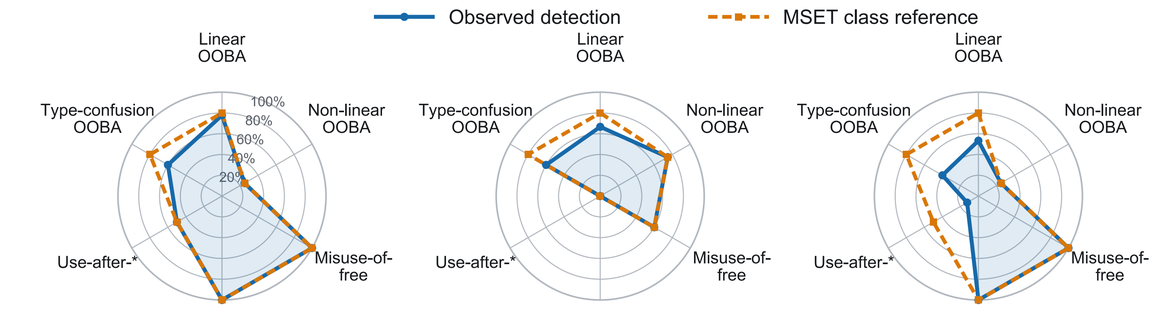
Juliet build	Found	Missed
O0 No FP / no debug	9,245	56
O1 + FP Line-table debug	7,999	1,302
O2 + FP + LTO Line-table debug	7,166	2,135
O3 + FP + LTO Line-table debug	7,166	2,135

Default ASan remains near 2×: 2.07× (O0) to 1.98× (O3). NoInline raises O2/O3 ratios; LTO effects vary by optimization level.

RQ3 answer: validate both cost and detection for the full build. The Juliet configurations change multiple flags, so they do not isolate LTO.

RQ4: Comparative Utility · Detection

MSET: 232 patterns; error type changes the comparison



Solid: observed detection. Dashed: technique-class reference, not guaranteed coverage. OOBA = out-of-bounds access.

LFP (LowFat) detects **75% of non-linear OOBA** vs 25% for ASan/ASan--/ASAP/MEDs, but no tested double-free or use-after-* patterns.

Juliet: 9,301 buggy + 9,301 non-buggy cases

Tool / configuration	TP	FN	FP
Default ASan	9,245	56	0
Tuned ASan (R16/Q16)	9,245	56	0
ASan--	9,245	56	0
LFP	4,710	4,591	0
MEDs	9,240	61	0
ASAP	7,696	1,605	43
SanRazor	9,137	164	4
RSan	9,301	0	0
GiantSan	8,332	969	0

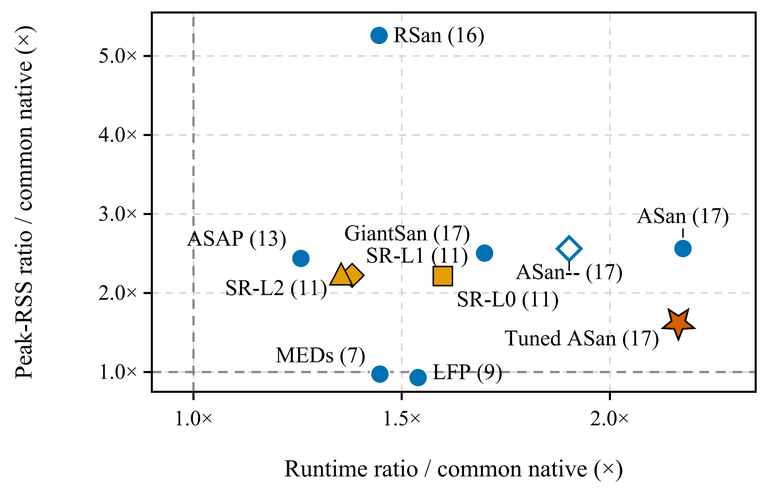
TP: reported bugs; FN: missed bugs; FP: reports on non-buggy cases. RSan finds all buggy Juliet cases; this is not a universal coverage result.

Real vulnerabilities: 21 selected CVE PoCs

Sanitizer	Reports / 21
ASan / ASan-- / GiantSan	21 each
LFP	19
SanRazor	15
MEDs	14

Seven projects: libzip (1), autotrace (9), imageworsener (4), zziplib (2), potrace (1), mp3gain (3), binutils (1). ASAP/RSan results are unreported, not zero. Selected PoCs do not establish general vulnerability coverage.

RQ4: Comparative Utility · Efficiency



Completed / 17

ASan, tuned ASan, ASan--, GiantSan:
17 each

RSan: 16
ASAP: 13
SanRazor L0/L1/L2:
11 each
LFP: 9
MEDs: 7

Reading the plot

Left: lower runtime.
Down: lower peak RSS.
Each point averages its own completed subset.

ASan uses a 25-point tuning search; alternatives use published or supported settings, not exhaustive tuning.

Arithmetic-mean ratios use each completed subset and one native reference. SR-L0, SR-L1, and SR-L2 denote the three SanRazor optimization levels.

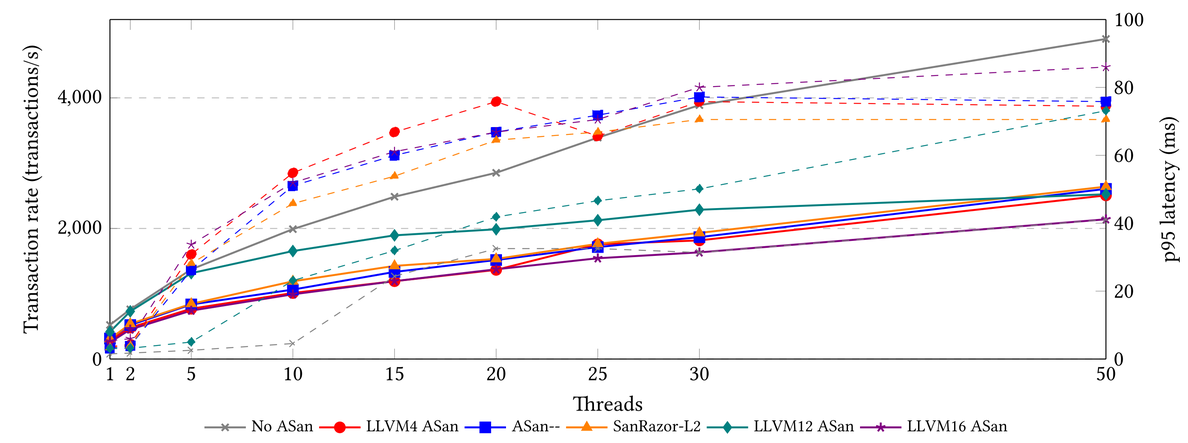
Low cost is not a deployment ranking: completion sets, toolchains, and tuning budgets differ. Include a tuned-ASan baseline.

RQ4: Comparative Utility · MySQL

MySQL 5.7.19: can each tool build, initialize, and run an OLTP read/write workload?

Tool / LLVM	Outcome
ASan / 4, 12, 16	All three complete the workload
ASan-- / 4; SanRazor-L2 / 9	Both complete the workload
GiantSan / 12	Compilation fails in yaSSL/TaoCrypt
MEDs / 4	gen_lex_hash source-generation crash
ASAP / 4	Compiler crashes on srv0start.cc
LFP / 4	Initialization aborts: LowFat OOB escape
RSan / 16	Link >4 h, then InnoDB initialization crash

Five instrumented configurations reach the workload. Earlier failures are deployment outcomes, not detection misses.



Solid: throughput (left axis). Dashed: p95 latency (right). Repeated runs; LLVM configurations share one native reference.

At 50 clients, instrumented configurations retain **44-54% of native throughput**: 2,140-2,641 versus 4,902 tx/s. The native p95 latency is 41.1 ms; instrumented configurations take 70.6-86.0 ms.

RQ4 answer: weigh cost, detection, and deployability. Halved MySQL throughput resembles ≈2× SPEC runtime in scale, not metric.

Scope: tested x86-64 Linux workloads, versions, and configurations. Legacy MySQL results do not represent all current database systems.

From Measurement to Deployment: Choose for Your Constraints

Broad deployment evidence

Start from default ASan: the most extensively exercised option in this environment, with Juliet, MSET, CVE, and MySQL evidence.

Memory-constrained testing

Try R16/Q16 as an ASan-based candidate. Re-test the target project, especially delayed use-after-free; lower RSS is not a coverage guarantee.

Specialized low-cost tools

ASAP, LFP, and MEDs show low costs on their completed subsets. Check relevant error types and application feasibility before adopting them.

Paper DOI: 10.1145/3832783.3834469
Reproduction: doi.org/10.5281/zenodo.21366704

* Cheng Wen · wencheng@xidian.edu.cn