29TH USENIX SECURITY SYMPOSIUM

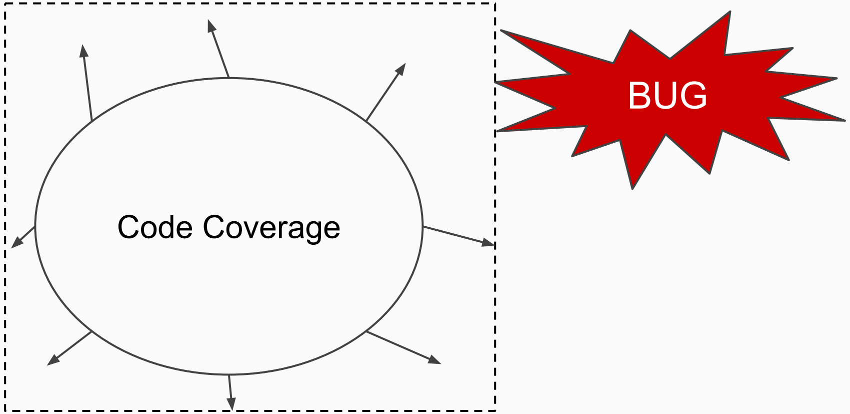
ParmeSan: Sanitizer-guided Greybox Fuzzing

Sebastian Österlund, Kaveh Razavi, Herbert Bos, Cristiano Giuffrida
Vrije Universiteit Amsterdam

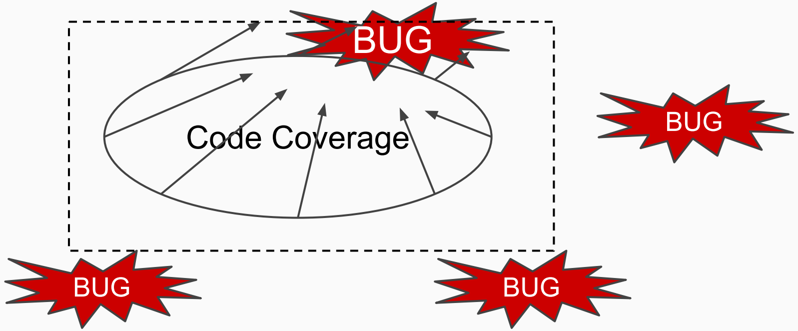
Slides made by Manh-Dung Nguyen

Research Context

Coverage-guided Greybox Fuzzing

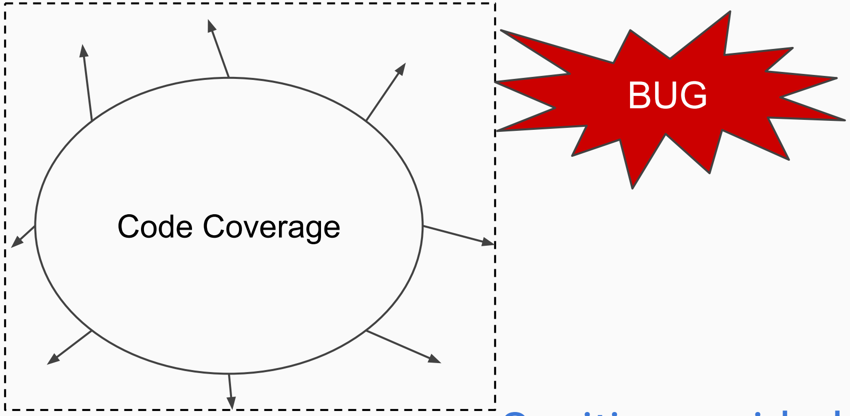


Directed Greybox Fuzzing

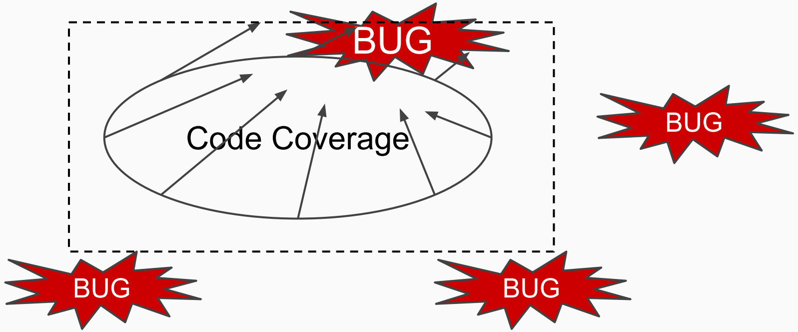


Research Context

Coverage-guided Greybox Fuzzing

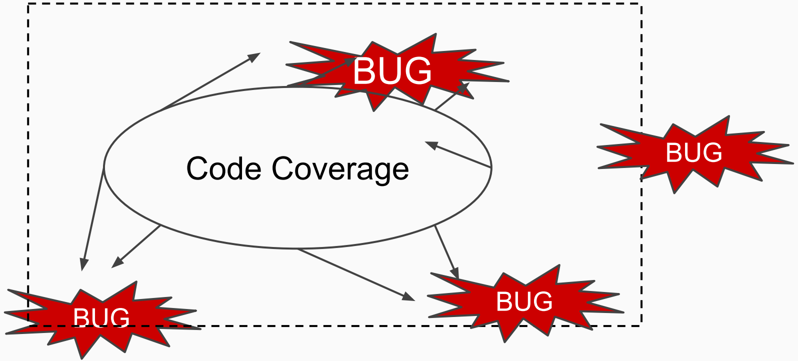


Directed Greybox Fuzzing



Sanitizer-guided Greybox Fuzzing

Better view of bugs



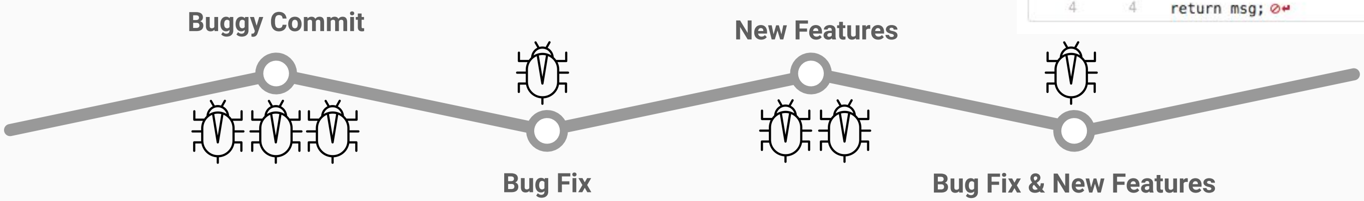
Better guidance towards bugs

Directed Greybox Fuzzing (DGF)

CVE-2018-20623 Detail

(1) Bug Reproduction
(lack of PoC, # env)

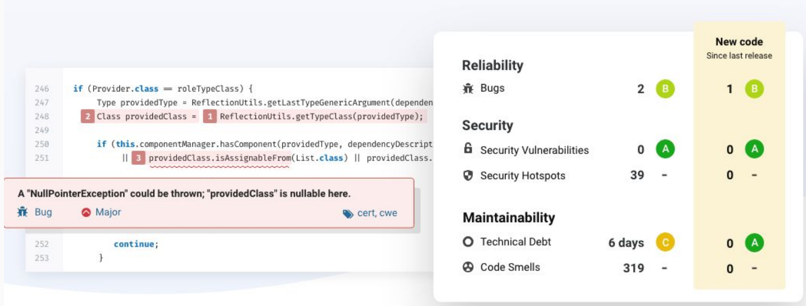
(2) Patch Testing
(vulnerable code)



(2) Patch Testing
(incomplete patch)

(3) Static Report Verification
(provide PoC inputs)

Focus on (3) Static Report Verification



Goal & Challenges

Sanitizer-guided Greybox Fuzzing

- Efficient interesting targets identification
- Efficient bug detection
- Low instrumentation-/runtime- overhead

Challenges

- No ideal sanitizers
  - Specific sanitizer: focus on specific bug & fewer targets
  - General sanitizer: more bugs & more targets
- More targets (compared to bug reproduction & patch testing)

Contributions

Techniques

- ★ A generic way of *finding interesting fuzzing targets* by relying on existing compiler sanitizer passes.
- ★ A *dynamic* approach to build a *precise* control-flow graph used to steer the input towards our targets.

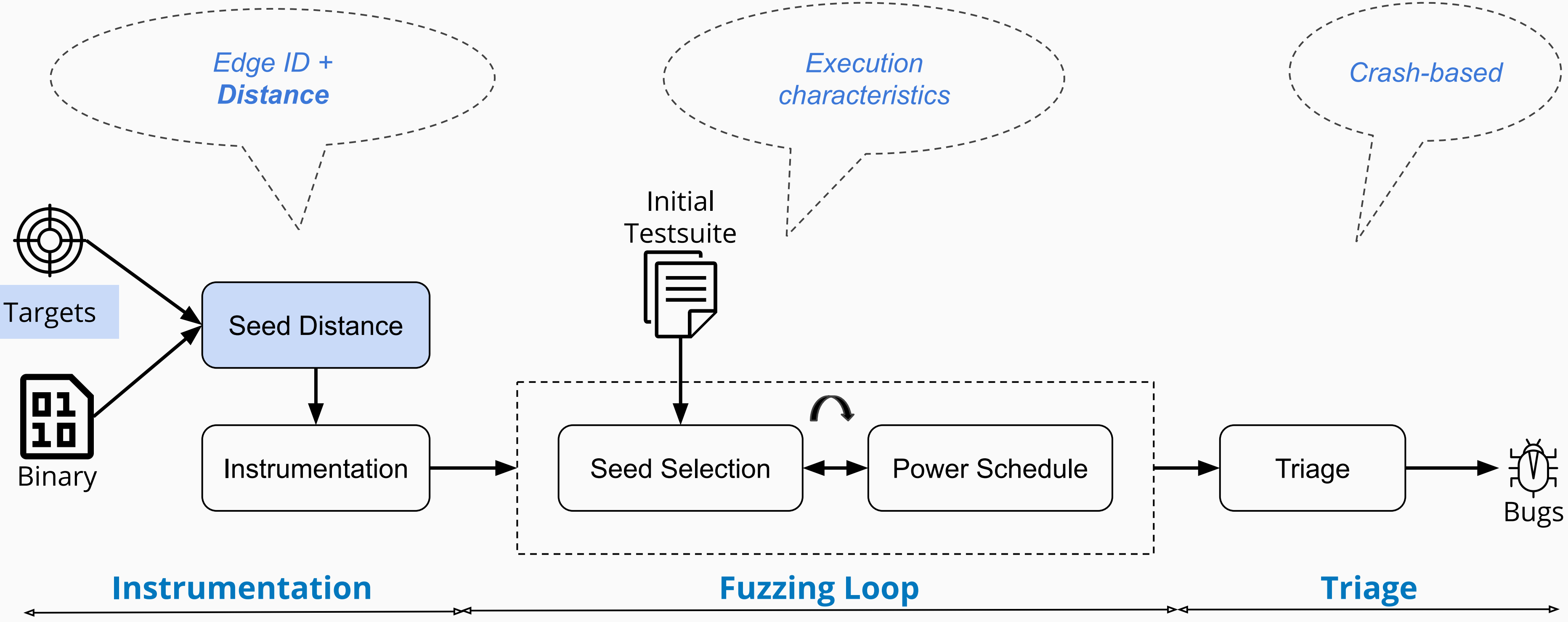
Implementation

- ★ ParmeSan, the *first sanitizer-guided fuzzer* using a two-stage directed fuzzing strategy to efficiently reach all the interesting targets.

Evaluation

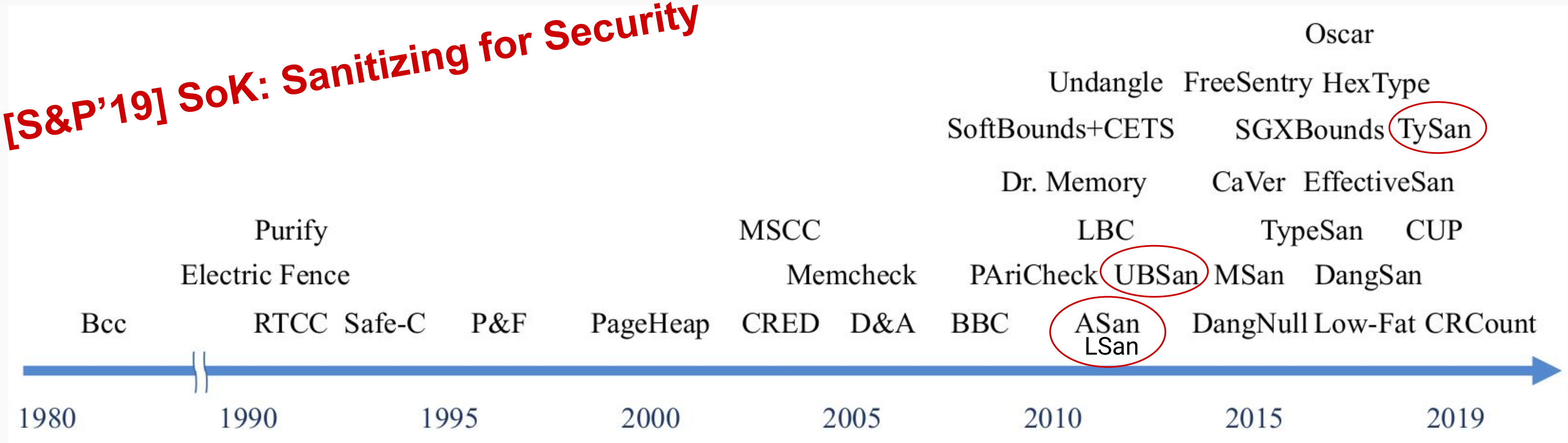
- ★ ParmeSan finds the *same bugs* as state-of-the-art coverage-guided and directed fuzzers *in less time*.

Background: Directed Greybox Fuzzing



Background: Sanitizers - Dynamic Analysis Tools (1)

[S&P'19] SoK: Sanitizing for Security



C/C++ Bugs

Null pointer
Integer Overflow

UndefinedBehaviorSanitizer (UBSan)

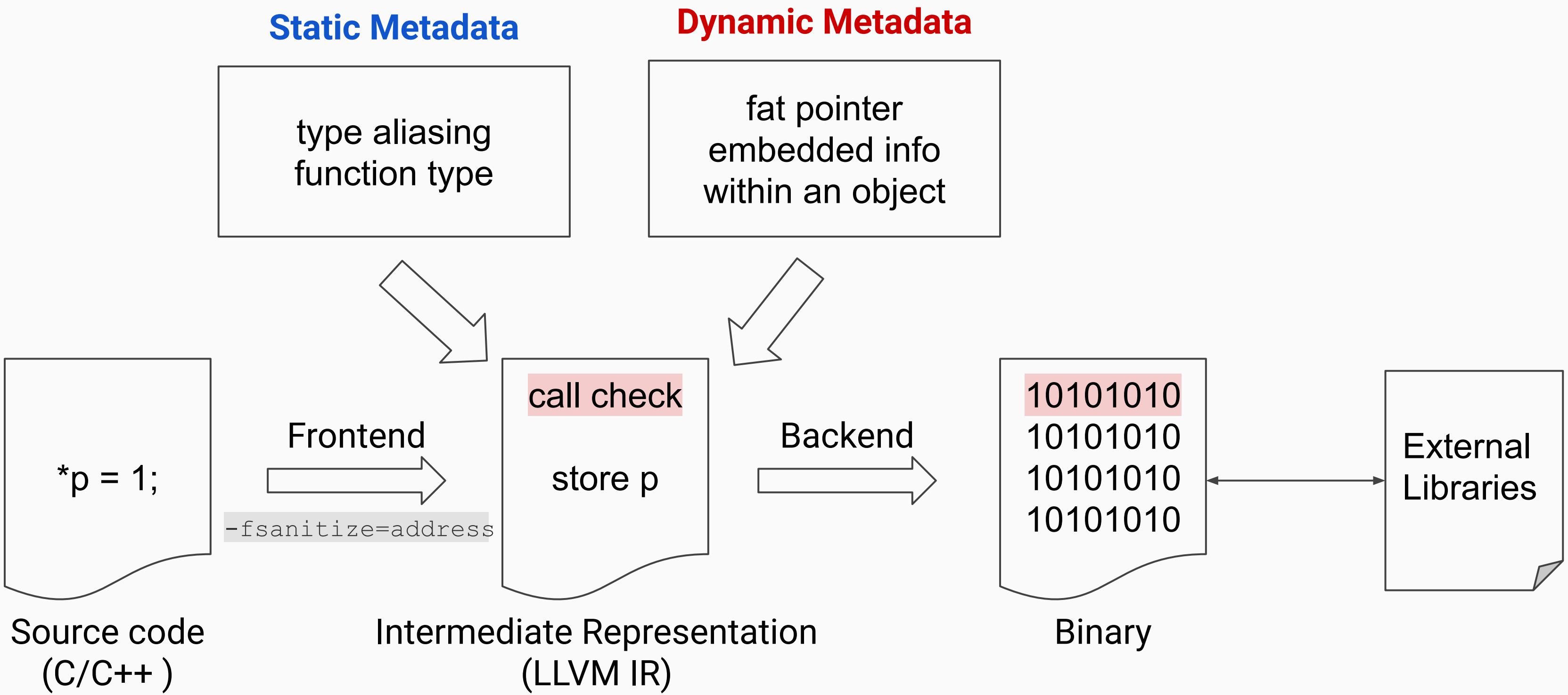
Out-of-bounds
Double Free
Use-After-Free
Memory Leak

AddressSanitizer (ASan)
LeakSanitizer (LSan)

Type Confusion

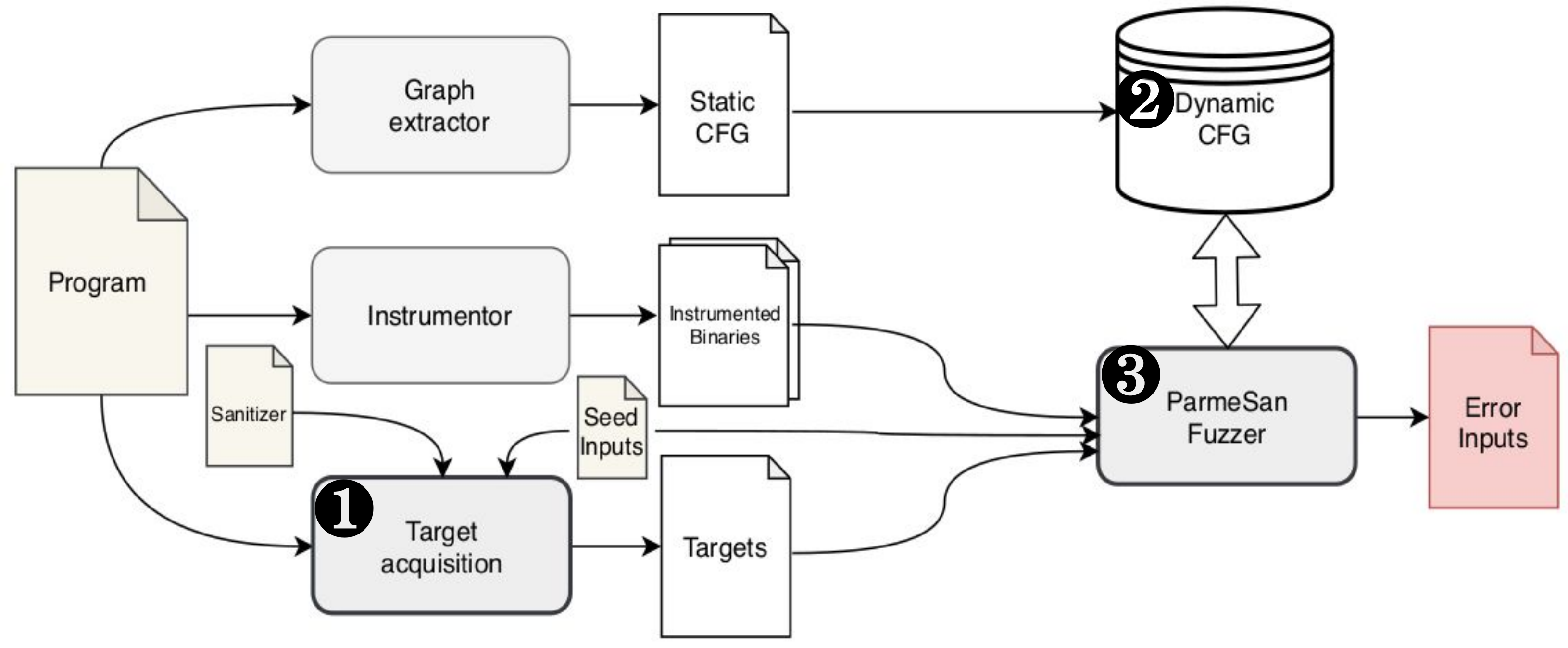
TypeSanitizer (TySan)

Background: Sanitizers - Instrumentation & Metadata (2)



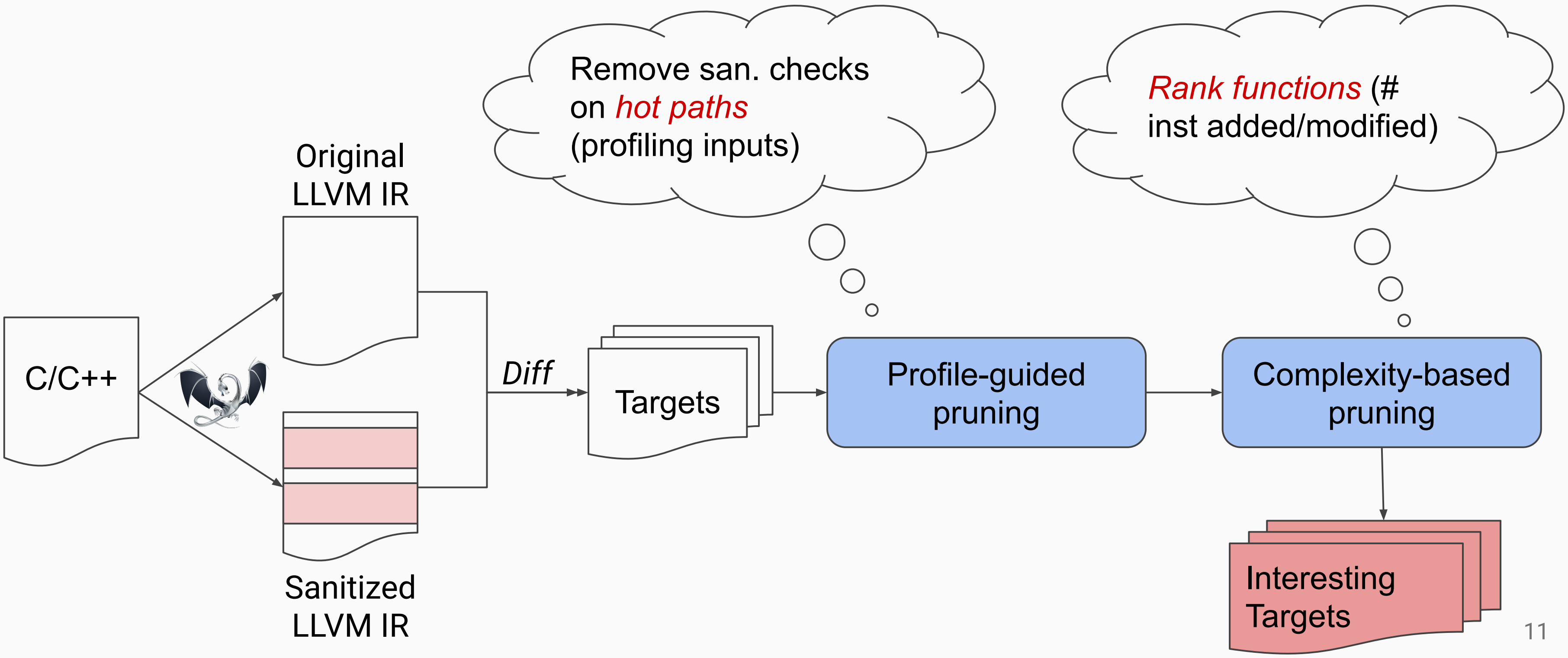
- Update internal data structures (shadow mem)
- Add branch conditions (san. checks)

ParmeSan Overview



Target Acquisition

<u>*Goal*</u>: Find *interesting (likely-buggy)* targets (BBs) for directed fuzzing



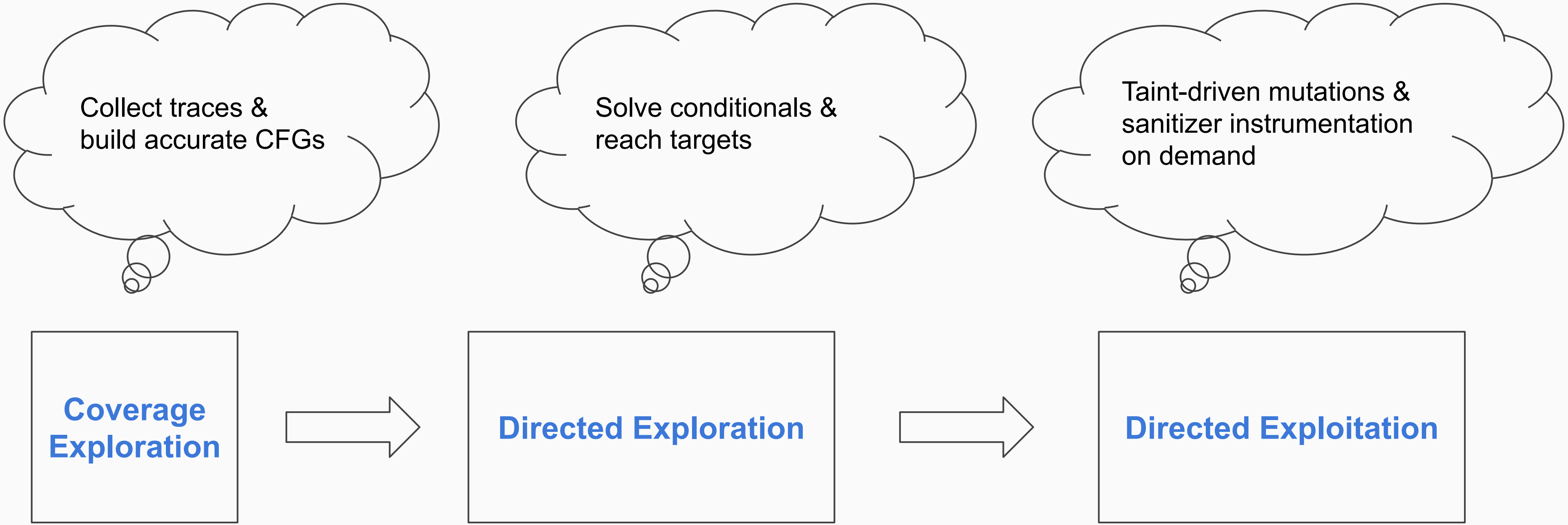
Dynamic CFG

- SoA directed fuzzers rely on *statically-generated CFGs*
  → scale at runtime but *imprecise*

Goal: *Dynamically* construct *precise* CFGs

- Add edges *on the fly* → resolve indirect calls
- Construct Conditional Graph: compacted CFG that *only contains the conditionals* →use it to perform *scalable/precise* distance calculations
- Augment CFG with *taint analysis* → record a taint label at uncovered edges to decide mutate input bytes or not

Sanitizer-guided Fuzzer



Evaluation Configurations

- Research Questions
  - RQ1: ParmeSan vs. Directed Fuzzers (AFLGo, Hawkeye)
  - RQ2: ParmeSan vs. Coverage-guided Fuzzers (NEUZZ, QSYM, Angora)
  - RQ3: Sanitizer Impact (ASan, UBSan, LSan, TySan)
  - RQ4: New Bugs
- Implementation
  - ParmeSan = LLVM passes (static analysis) + Angora (fuzzer)
- Benchmark
  - Binutils (vs. directed fuzzers) + Google Fuzzing Testsuite (vs. coverage-guided fuzzers)

RQ1: ParmeSan vs. Directed Fuzzers

- Bench: 6 Binutils (AFLGo's paper) + 1 OpenSSL
- Rerun AFLGo
- Reclaim results of Hawkeye in paper

ParmeSan *outperforms directed fuzzers* against AFLGo's benchmark

CVE	Fuzzer	Runs	p-val	Mean TTE
OpenSSL				
2014-0160	ParmeSan	30		5m10s
	HawkEye	—		—
	AFLGo	30	0.006	20m15s
Binutils				
2016-4487 2016-4488	ParmeSan	30		35s
	HawkEye	20		2m57s
	AFLGo	30	0.005	6m20s
2016-4489	ParmeSan	30		1m5s
	HawkEye	20		3m26s
	AFLGo	30	0.03	2m54s
2016-4490	ParmeSan	30		55s
	HawkEye	20		1m43s
	AFLGo	30	0.01	1m24s
2016-4491	ParmeSan	10		1h10m
	HawkEye	9		5h12m
	AFLGo	5	0.003	6h21m
2016-4492 2016-4493	ParmeSan	30		2m10s
	HawkEye	20		7m57s
	AFLGo	20	0.003	8m40s
2016-6131	ParmeSan	10		1h10m
	HawkEye	9		4h49m
	AFLGo	5	0.04	5h50m

RQ2: ParmeSan vs. Coverage-guided Fuzzers

- Bench: 15 bugs of Google Fuzzing Testsuite
- All fuzzers run *with sanitizers enabled*
- AFLGo uses the targets obtained using the ParmeSan analysis stage

	Branch Cov	Time-to-Exposure
AFLGo	+16%	+288%
NEUZZ	+40%	+81%
QSYM	+95%	+867%
Angora	+33%	+37%
ParmeSan	-	-

- ParmeSan *outperforms coverage-guided fuzzers* (1) branch coverage (2) TTE
- Directed fuzzers require *less coverage*

RQ3: Sanitizer Impact

- Bench: 10 bugs
- ParmeSan using *different sanitizers* in the analysis stage.
- Example: UAF bug in pcre2
  - # targets: TySan < ASan (supports more bug classes)
  - TTE : TySan is 20% faster than ASan

Different sanitizers have *different impacts* on # targets and TTE

RQ4: New Bugs

- Bench: 12 targets from OSS-Fuzz to fuzz the most recent commits
- Found 47 unique bugs
  - 37 bugs in outdated library *pbc*
  - 10 bugs in well-fuzzed libraries

ParmeSan *found new bugs* in well-fuzzed programs

Limitations

- ParmeSan relies on LLVM IR and relevant analysis techniques
- Raw binaries: using binary hardening ?

- Composing sanitizers
- Apply heuristics in SoA directed fuzzers

Backup

Additional Results

Impact of different components

	Branch Cov	Time-to-Exposure
No lazysan	+0%	+25%
No pruning	+19%	+28%
No dyncfg	+17%	+34%
ParmeSan	-	-

Run-time and compile-time overhead

The overhead is negligible in most cases: *less than 3%* of the total execution time