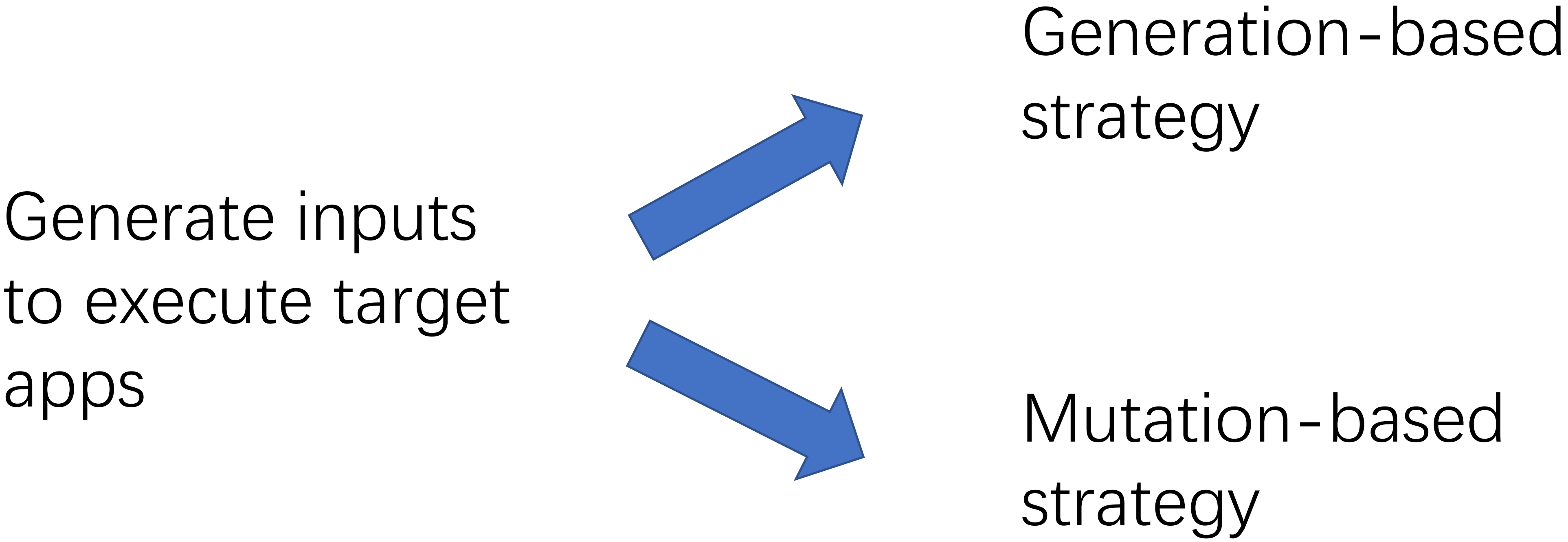
EnFuzz: Ensemble Fuzzing with Seed Synchronization among Diverse Fuzzers

28th USENIX Security Symposium (Security), USA , 2019

Idea of designing fuzzer



Generate inputs to execute target apps

Generation-based strategy

Mutation-based strategy

Current optimizations

AFLFast: Select seeds that exercise low-frequency paths for additional mutations

FairFuzz: Optimize AFL’s mutation algorithm to prioritize seeds that hit rare branches

QSYM: Use a practical concolic execution engine to solve complex branches of AFL

Key problem

Fuzzing strategies are inconsistent in the performance to real-world applications!

↓

No fuzzer can ganrantee that it can be best on every real-world applications.

Goal

For industrial practice, more robust fuzzing strategies are desired when applied across different applications.

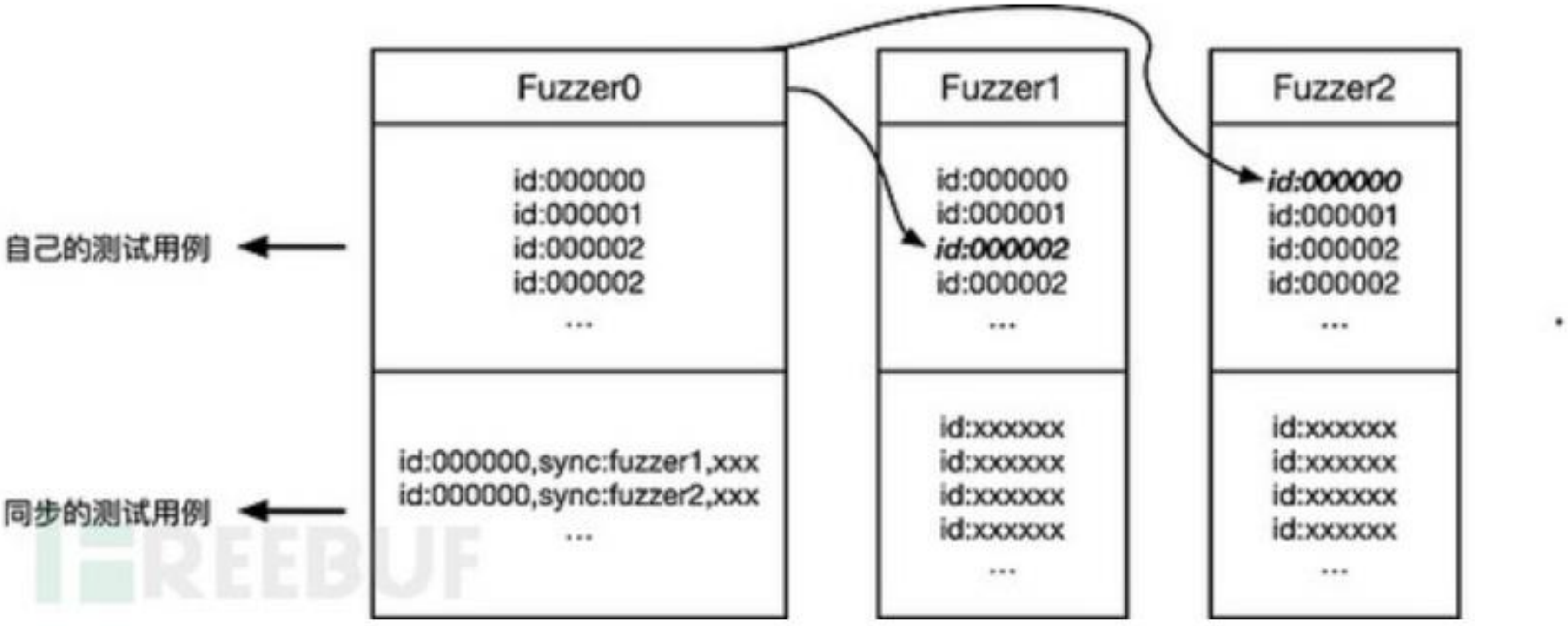
Cluster and Parallel Fuzzing

Cluster fuzzing means running multiple identical instances of fuzzers on distributed system for one target app.

e.g. ClusterFuzz

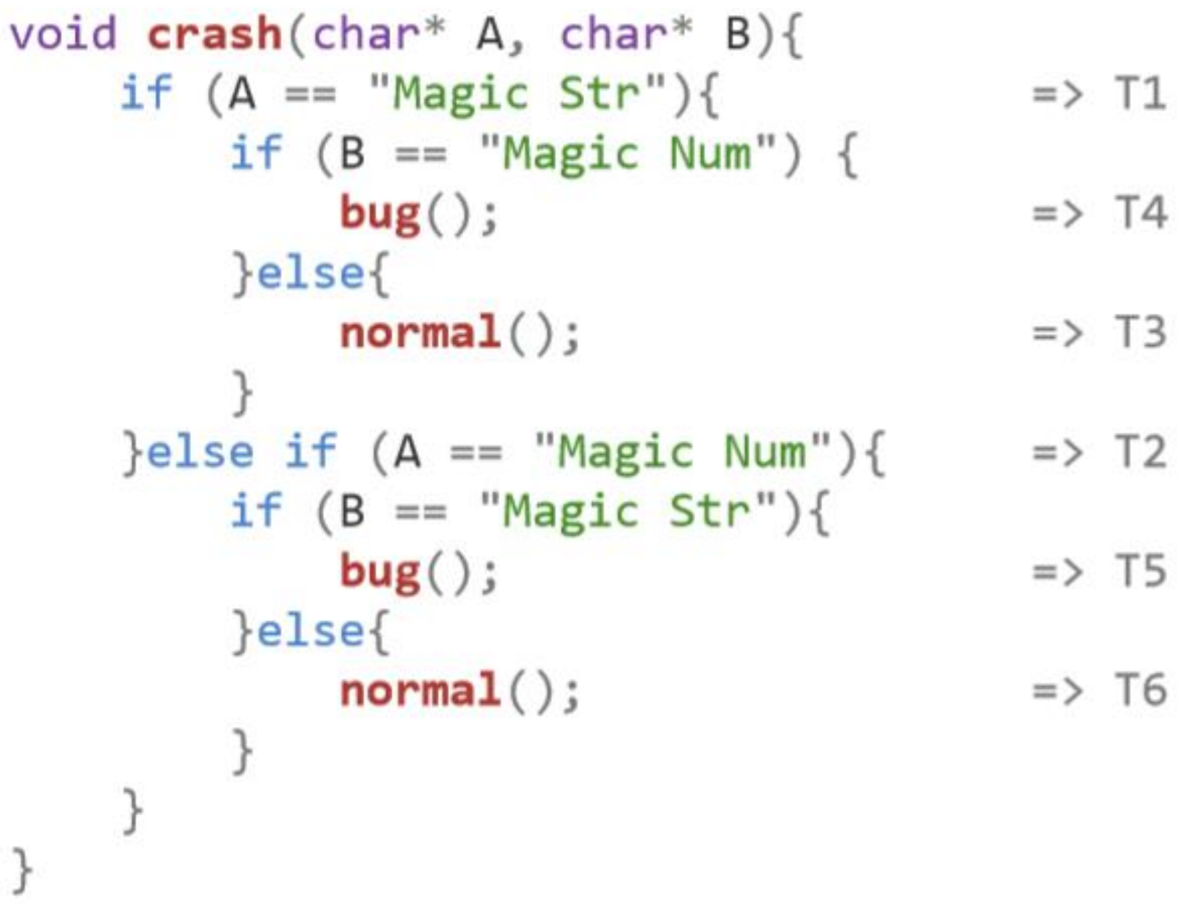
Parallel fuzzing means working with some synchronization mechisms like re-scan the top-level sync directory for any testcases found by other fuzzers.

Cluster and Parallel Fuzzing



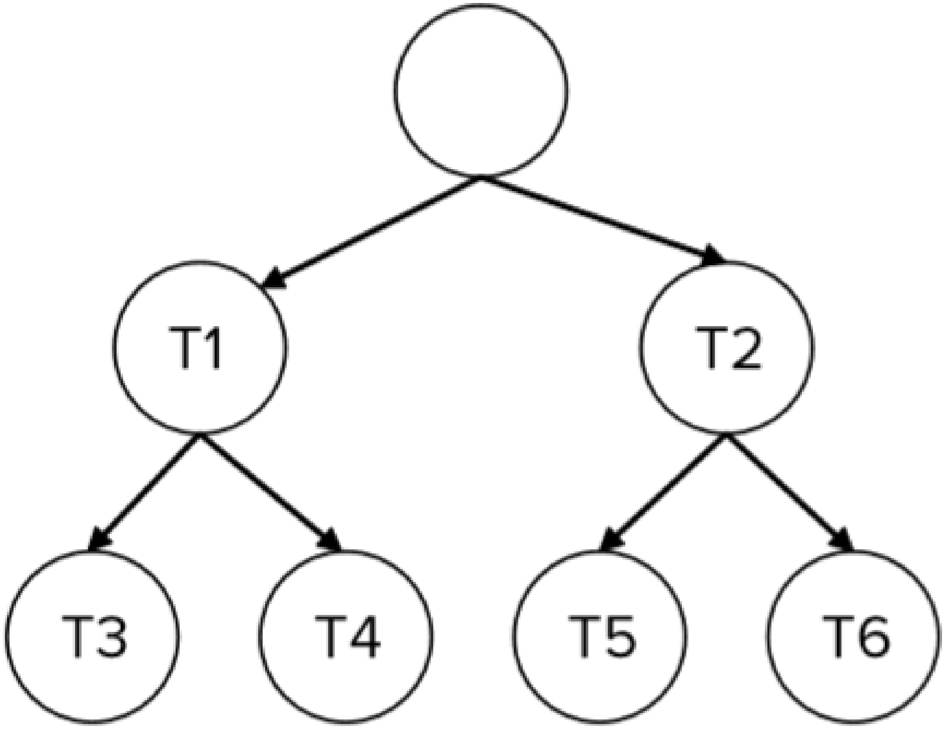
Fuzzer0 is a master fuzzer, fuzzer1 and fuzzer2 is slave fuzzers. Here fuzzer1 and fuzzer2 is the identical fuzzer.

Motivation



```
void crash(char* A, char* B){
    if (A == "Magic Str"){                  => T1
        if (B == "Magic Num") {
            bug();                          => T4
        }else{
            normal();                       => T3
        }
    }else if (A == "Magic Num"){            => T2
        if (B == "Magic Str"){
            bug();                          => T5
        }else{
            normal();                       => T6
        }
    }
}
```

Motivation



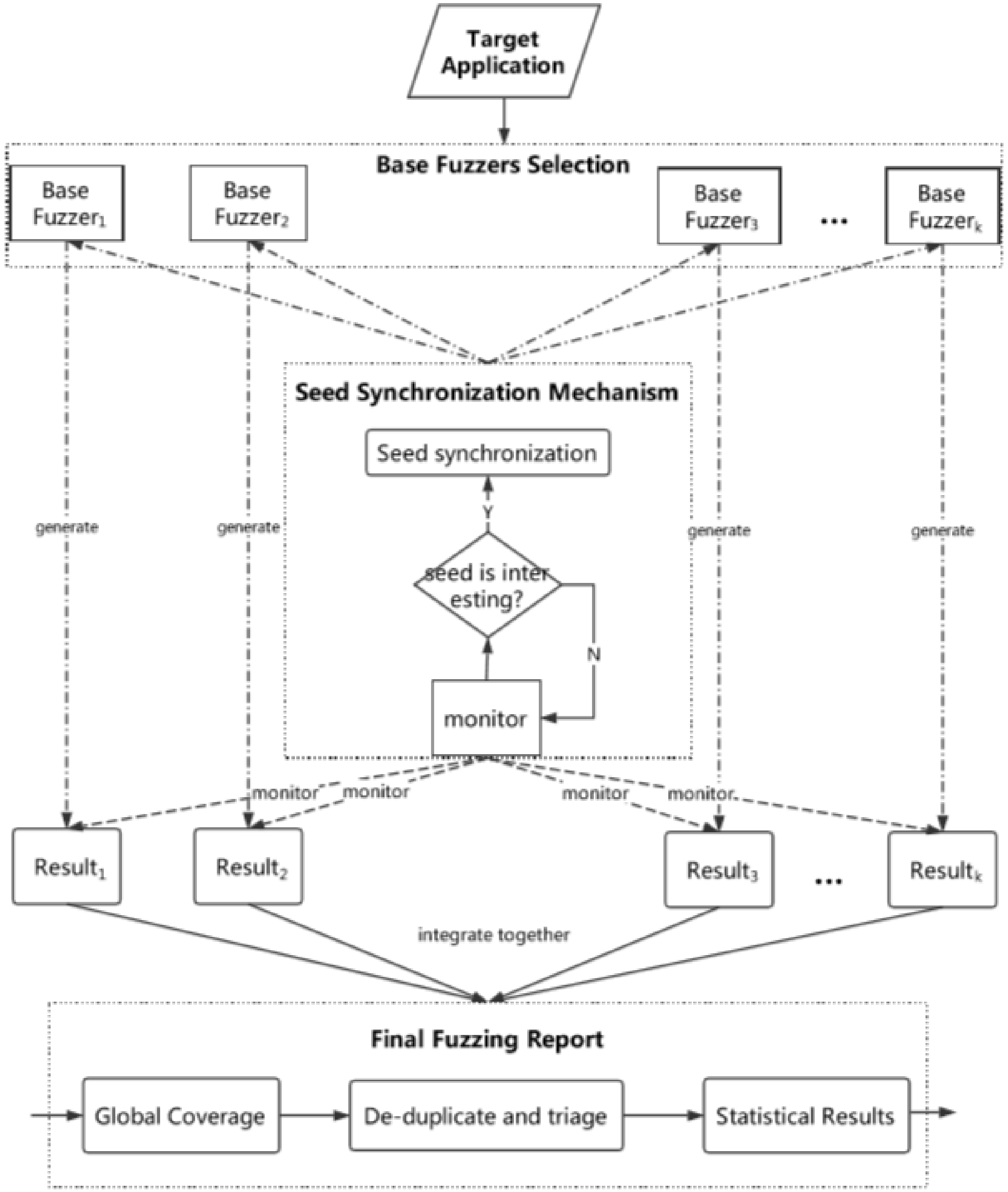
Whether there is a method to cover all the code path?

Ensemble Fuzzing

- Base fuzzers

- Globally asynchronous and locally synchronous based seed synchronization mechanism

- Crash and coverage information

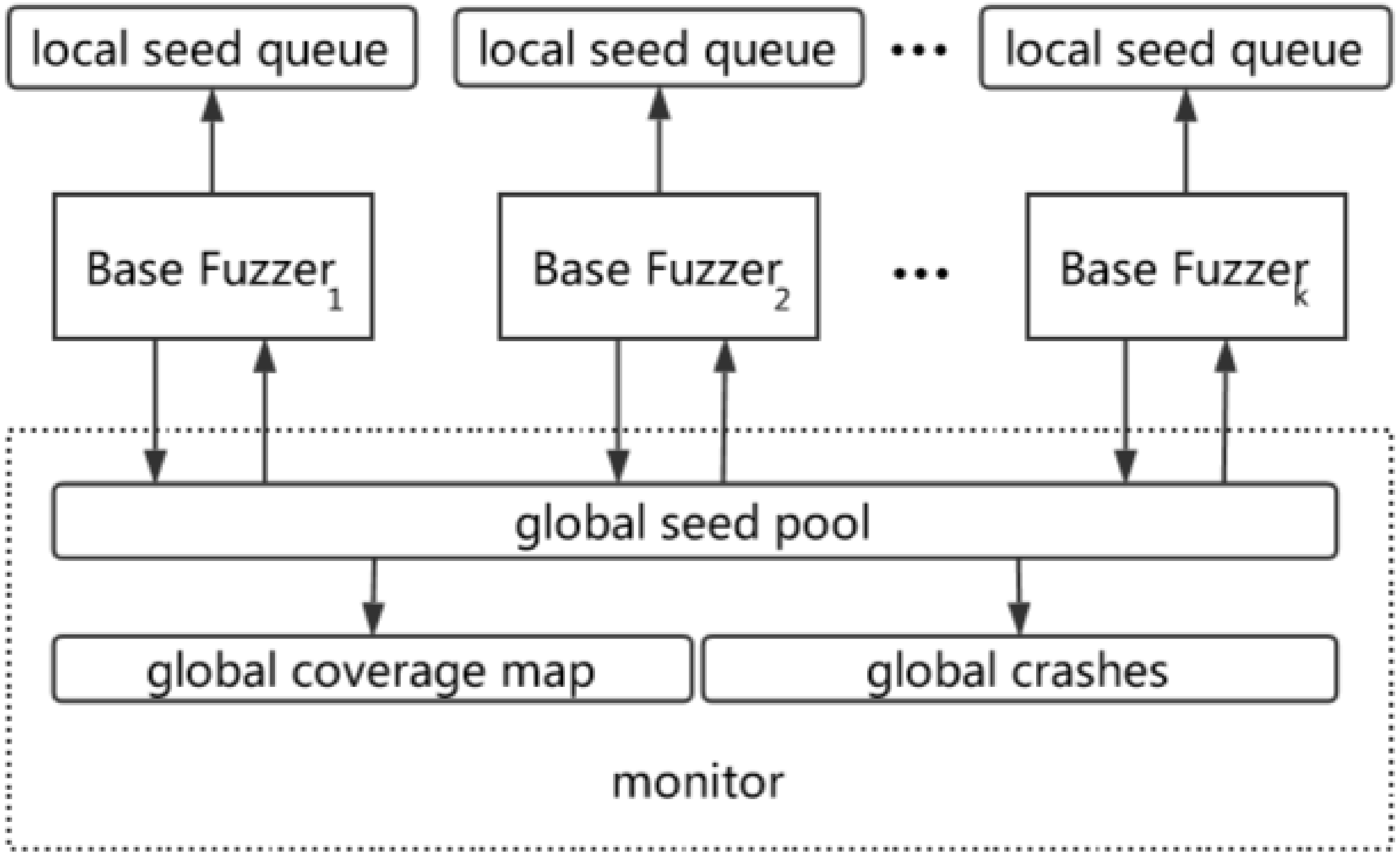


Target Application
Base Fuzzers Selection
Base Fuzzer$_1$
Base Fuzzer$_2$
Base Fuzzer$_3$
...
Base Fuzzer$_k$
generate
generate
generate
generate
Seed Synchronization Mechanism
Seed synchronization
Y
seed is interesting?
N
monitor
monitor
monitor
monitor
monitor
Result$_1$
Result$_2$
Result$_3$
...
Result$_k$
integrate together
Final Fuzzing Report
Global Coverage
De-duplicate and triage
Statistical Results

Base Fuzzer Selection

- Seed mutation and selection strategy diversity
- Coverage information granularity diversity
- Inputs generation strategy diversity

Architecture



GALS

The main idea is to identify the interesting seeds from different base fuzzers asynchronously and share those interesting seeds synchronously among all fuzzing processes.

(seeds that can cover new paths or new branches or can detect new unique crashes)

What does a base fuzzer do?

1. Select input seeds from the queue
2. mutate the selected input seeds to generate new candidate seeds
3. run the target program with the candidate seeds, track the coverage and report vulnerabilities
4. The candidate seeds have new coverage or cause unique crashes, they will be regarded as interesting seeds and be pushed asynchronously into the global seed pool

What does the monitor do?

- It initializes the global coverage information and creates the global seed pool with the initial seeds

- Each base fuzzer will be synchronously dispatched with the interesting seeds from the global seed pool. (Seeds contribute to the coverage or crash and has not been generated by the local fuzzer)

Implementation

- Three edge-coverage-guided mutationbased fuzzers – AFL, AFLFast and FairFuzz
- One block-coverage-guided mutation-based fuzzer – libFuzzer
- One generation-based fuzzer – Radamsa
- One most recently hybrid fuzzer – QSYM.

Challenges

- Standard Interface Encapsulating
- LibFuzzer Continuously Fuzzing
- Bugs De-duplicating and Triaging
- Seeds effectively Synchronizing

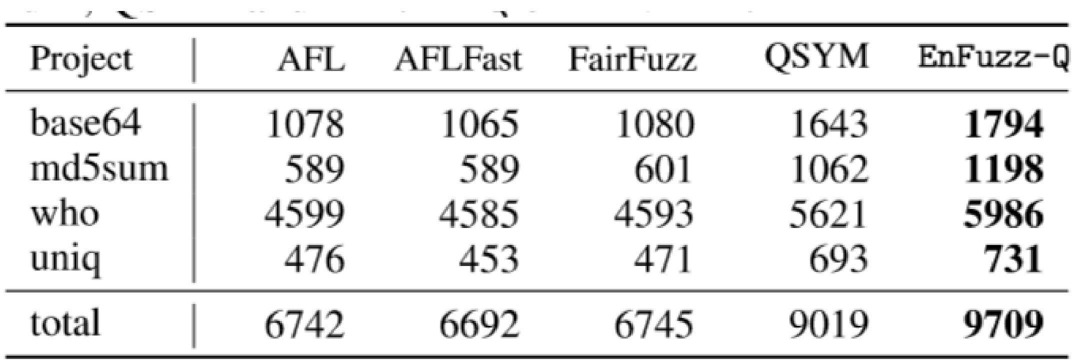
Data and Environment Setup

LAVA-M --- injects hard-to-find bugs in Linux utilities

Google fuzzer-test-suite

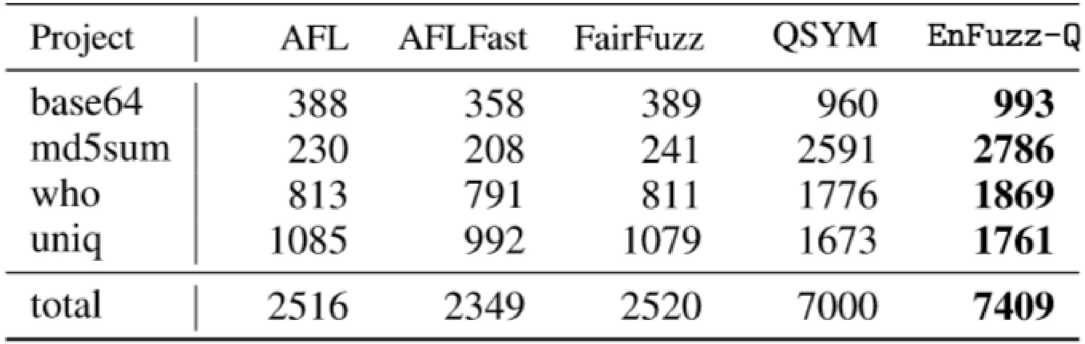
Real-world applications

Evaluation



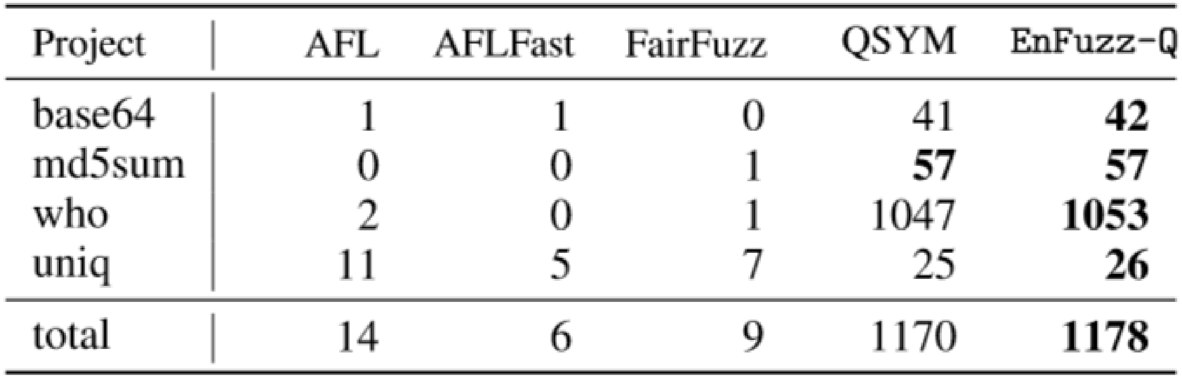
Project	AFL	AFLFast	FairFuzz	QSYM	EnFuzz-Q
base64	1078	1065	1080	1643	**1794**
md5sum	589	589	601	1062	**1198**
who	4599	4585	4593	5621	**5986**
uniq	476	453	471	693	**731**
total	6742	6692	6745	9019	**9709**

Path coverage



Project	AFL	AFLFast	FairFuzz	QSYM	EnFuzz-Q
base64	388	358	389	960	**993**
md5sum	230	208	241	2591	**2786**
who	813	791	811	1776	**1869**
uniq	1085	992	1079	1673	**1761**
total	2516	2349	2520	7000	**7409**

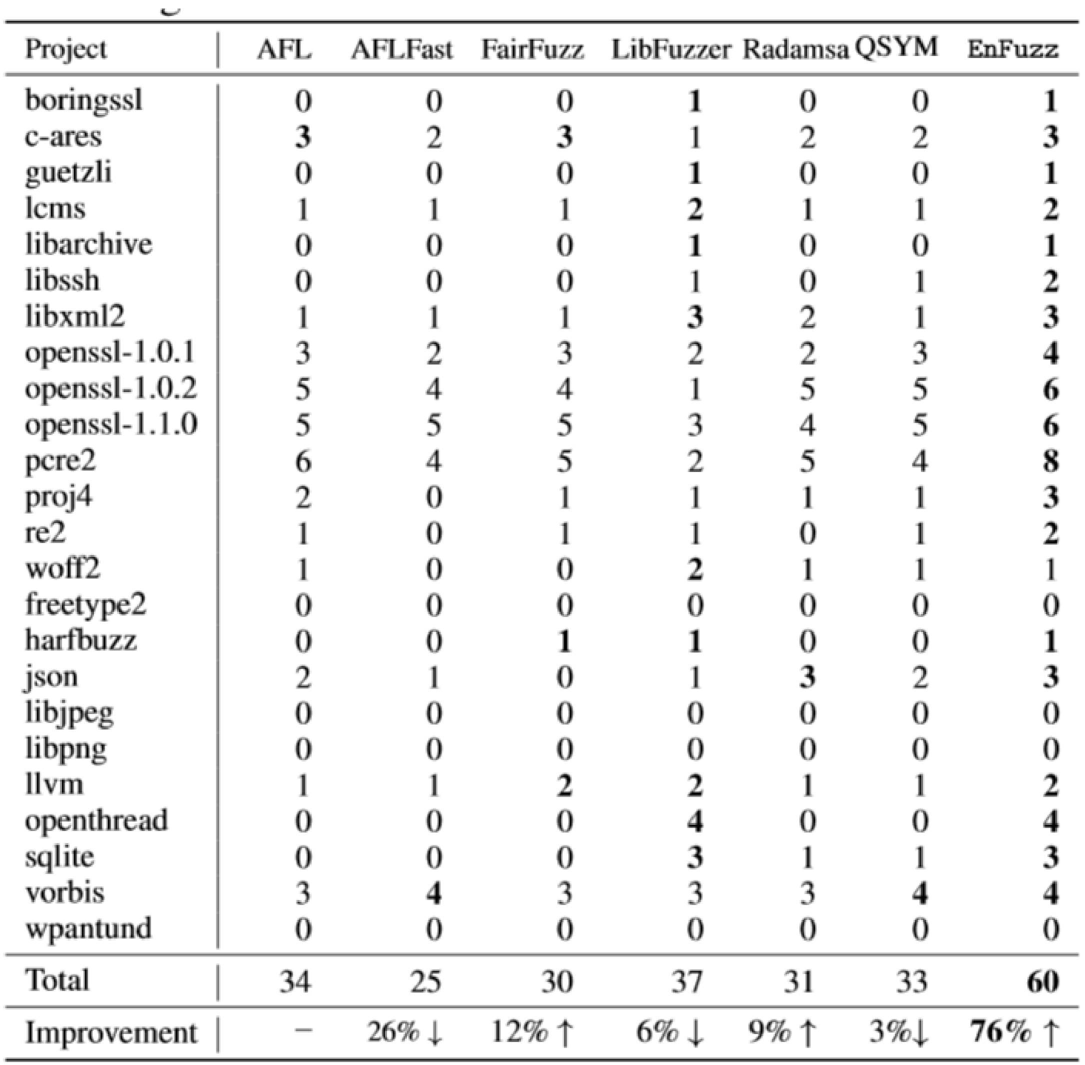
Branch coverage



Project	AFL	AFLFast	FairFuzz	QSYM	EnFuzz-Q
base64	1	1	0	41	**42**
md5sum	0	0	1	**57**	**57**
who	2	0	1	1047	**1053**
uniq	11	5	7	25	**26**
total	14	6	9	1170	**1178**

Bugs

Evaluation

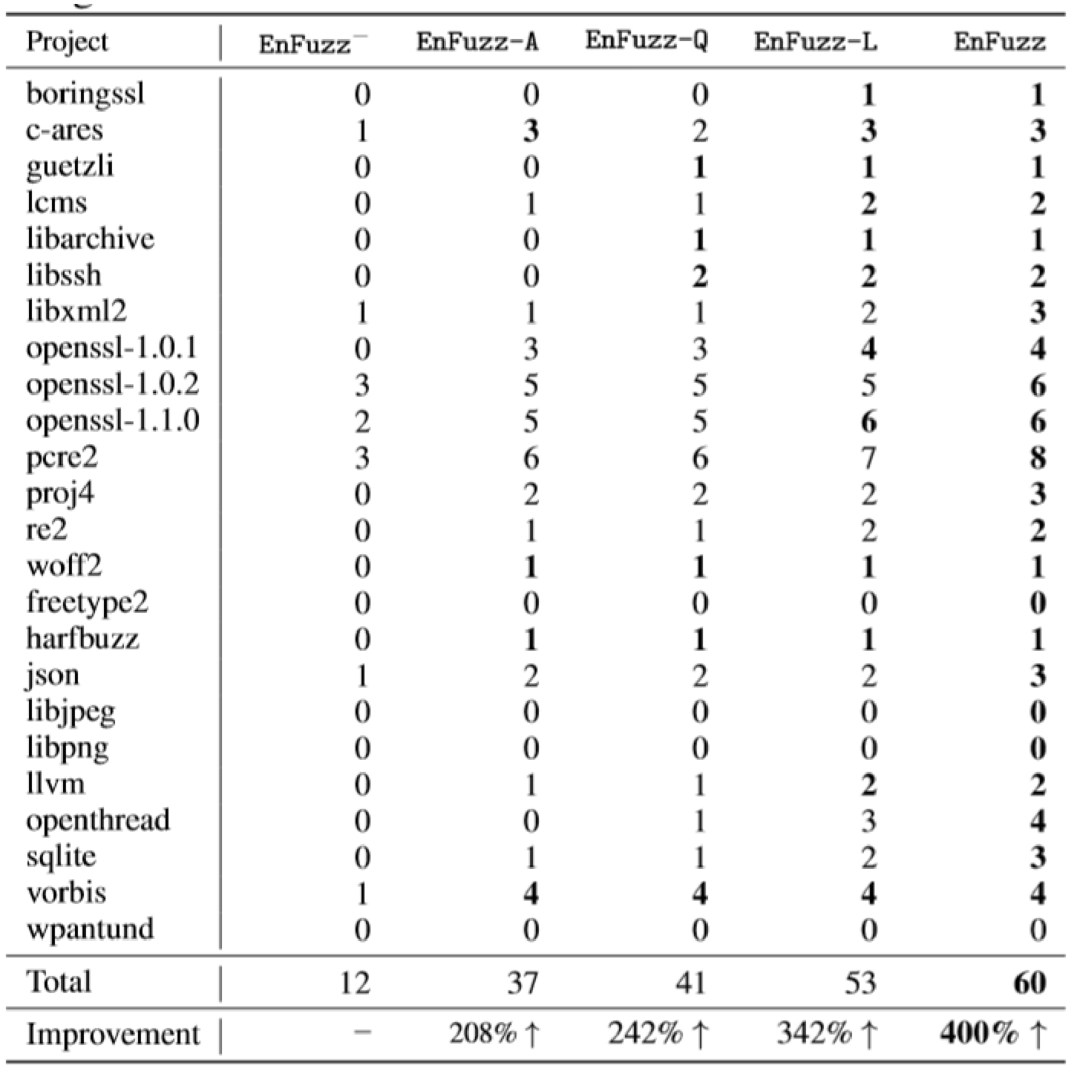


Project	AFL	AFLFast	FairFuzz	LibFuzzer	Radamsa	QSYM	EnFuzz
boringssl	0	0	0	**1**	0	0	**1**
c-ares	**3**	2	**3**	1	2	2	**3**
guetzli	0	0	0	**1**	0	0	**1**
lcms	1	1	1	**2**	1	1	**2**
libarchive	0	0	0	**1**	0	0	**1**
libssh	0	0	0	1	0	1	**2**
libxml2	1	1	1	**3**	2	1	**3**
openssl-1.0.1	3	2	3	2	2	3	**4**
openssl-1.0.2	5	4	4	1	5	5	**6**
openssl-1.1.0	5	5	5	3	4	5	**6**
pcre2	6	4	5	2	5	4	**8**
proj4	2	0	1	1	1	1	**3**
re2	1	0	1	1	0	1	**2**
woff2	1	0	0	**2**	1	1	1
freetype2	0	0	0	0	0	0	0
harfbuzz	0	0	**1**	**1**	0	0	**1**
json	2	1	0	1	**3**	2	**3**
libjpeg	0	0	0	0	0	0	0
libpng	0	0	0	0	0	0	0
llvm	1	1	**2**	**2**	1	1	**2**
openthread	0	0	0	**4**	0	0	**4**
sqlite	0	0	0	**3**	1	1	**3**
vorbis	3	**4**	3	3	3	**4**	**4**
wpantund	0	0	0	0	0	0	0
Total	34	25	30	37	31	33	**60**
Improvement	–	26% ↓	12% ↑	6% ↓	9% ↑	3%↓	**76%** ↑

Evaluation

- EnFuzz-A, an ensemble fuzzer only based on AFL, AFLFast and FairFuzz.
- EnFuzz-Q, an ensemble fuzzer based on AFL, AFLFast, FairFuzz and QSYM, a practical concolic execution engine is included.
- EnFuzz-L, an ensemble fuzzer based on AFL, AFLFast, FairFuzz and libFuzzer, a block-coverage guided fuzzer is included.
- EnFuzz, an ensemble fuzzer based on AFL, AFLFast, libFuzzer and Radamsa, a generation-based fuzzer is further added.
- EnFuzz−, with the ensemble of same base fuzzers (AFL, AFLFast and FairFuzz), but without the seed synchronization, to demonstrate the effectiveness of the global asynchronous and local synchronous based seed synchronization mechanism.

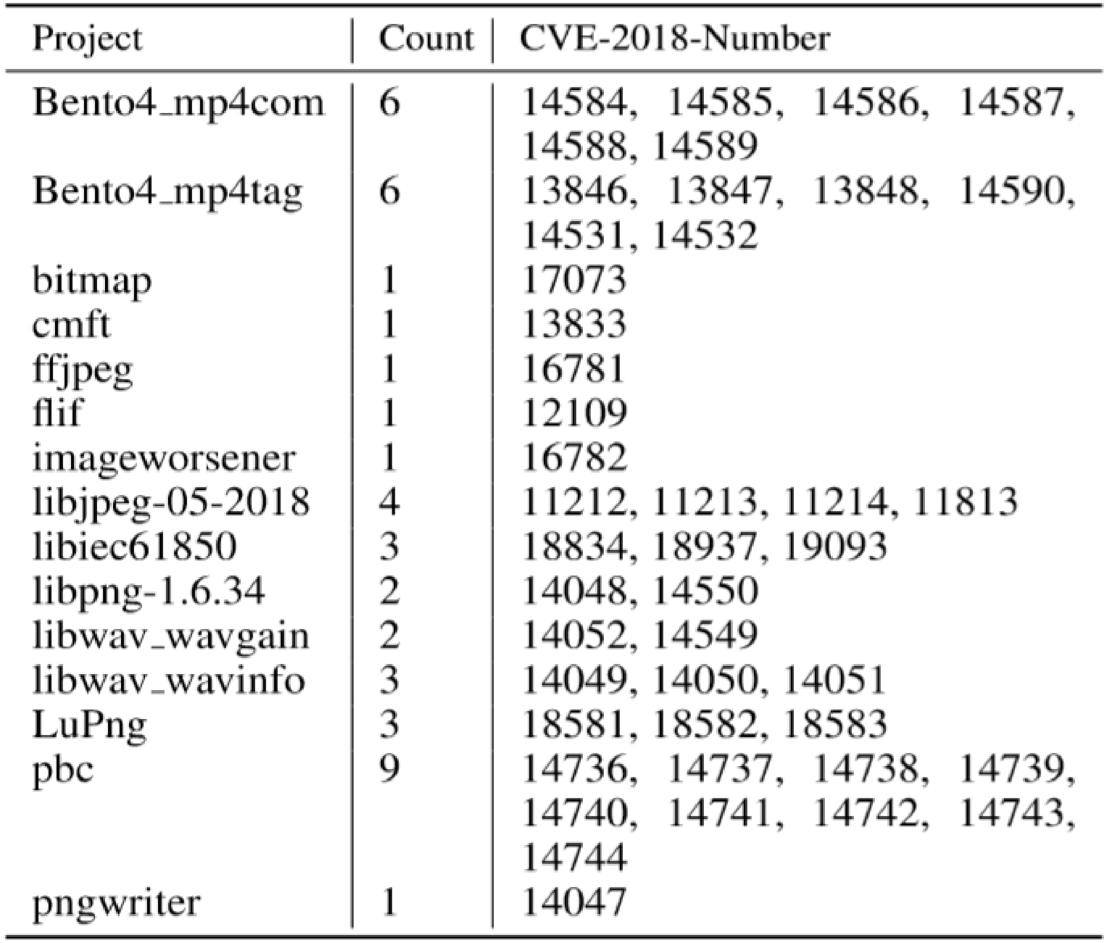
Evaluation



Project	EnFuzz$^-$	EnFuzz-A	EnFuzz-Q	EnFuzz-L	EnFuzz
boringssl	0	0	0	**1**	**1**
c-ares	1	**3**	2	**3**	**3**
guetzli	0	0	**1**	**1**	**1**
lcms	0	1	1	**2**	**2**
libarchive	0	0	**1**	**1**	**1**
libssh	0	0	**2**	**2**	**2**
libxml2	1	1	1	2	**3**
openssl-1.0.1	0	3	3	**4**	**4**
openssl-1.0.2	3	5	5	5	**6**
openssl-1.1.0	2	5	5	**6**	**6**
pcre2	3	6	6	7	**8**
proj4	0	2	2	2	**3**
re2	0	1	1	2	**2**
woff2	0	**1**	**1**	**1**	**1**
freetype2	0	0	0	0	**0**
harfbuzz	0	**1**	**1**	**1**	**1**
json	1	2	2	2	**3**
libjpeg	0	0	0	0	**0**
libpng	0	0	0	0	**0**
llvm	0	1	1	**2**	**2**
openthread	0	0	1	3	**4**
sqlite	0	1	1	2	**3**
vorbis	1	**4**	**4**	**4**	**4**
wpantund	0	0	0	0	0
Total	12	37	41	53	**60**
Improvement	–	208% ↑	242% ↑	342% ↑	**400%** ↑

Evaluation

Within 24 hours, besides the coverage improvements, EnFuzz finds 60 more unknown real bugs including 44 successfully registered as CVEs



Project	Count	CVE-2018-Number
Bento4_mp4com	6	14584, 14585, 14586, 14587, 14588, 14589
Bento4_mp4tag	6	13846, 13847, 13848, 14590, 14531, 14532
bitmap	1	17073
cmft	1	13833
ffjpeg	1	16781
flif	1	12109
imageworsener	1	16782
libjpeg-05-2018	4	11212, 11213, 11214, 11813
libiec61850	3	18834, 18937, 19093
libpng-1.6.34	2	14048, 14550
libwav_wavgain	2	14052, 14549
libwav_wavinfo	3	14049, 14050, 14051
LuPng	3	18581, 18582, 18583
pbc	9	14736, 14737, 14738, 14739, 14740, 14741, 14742, 14743, 14744
pngwriter	1	14047

discussion

- the Insuffcient and Imprecise diversity of base fuzzers
- mechanism scalability of the ensemble architecture