软件漏洞挖掘方法探索
Finding Vulnerabilities with Fuzzing

Chao Zhang

Tsinghua University

http://netsec.ccert.edu.cn/chaoz/

About Me

2004-2008-2013 ➔ 2013-2016 ➔ 2016-present

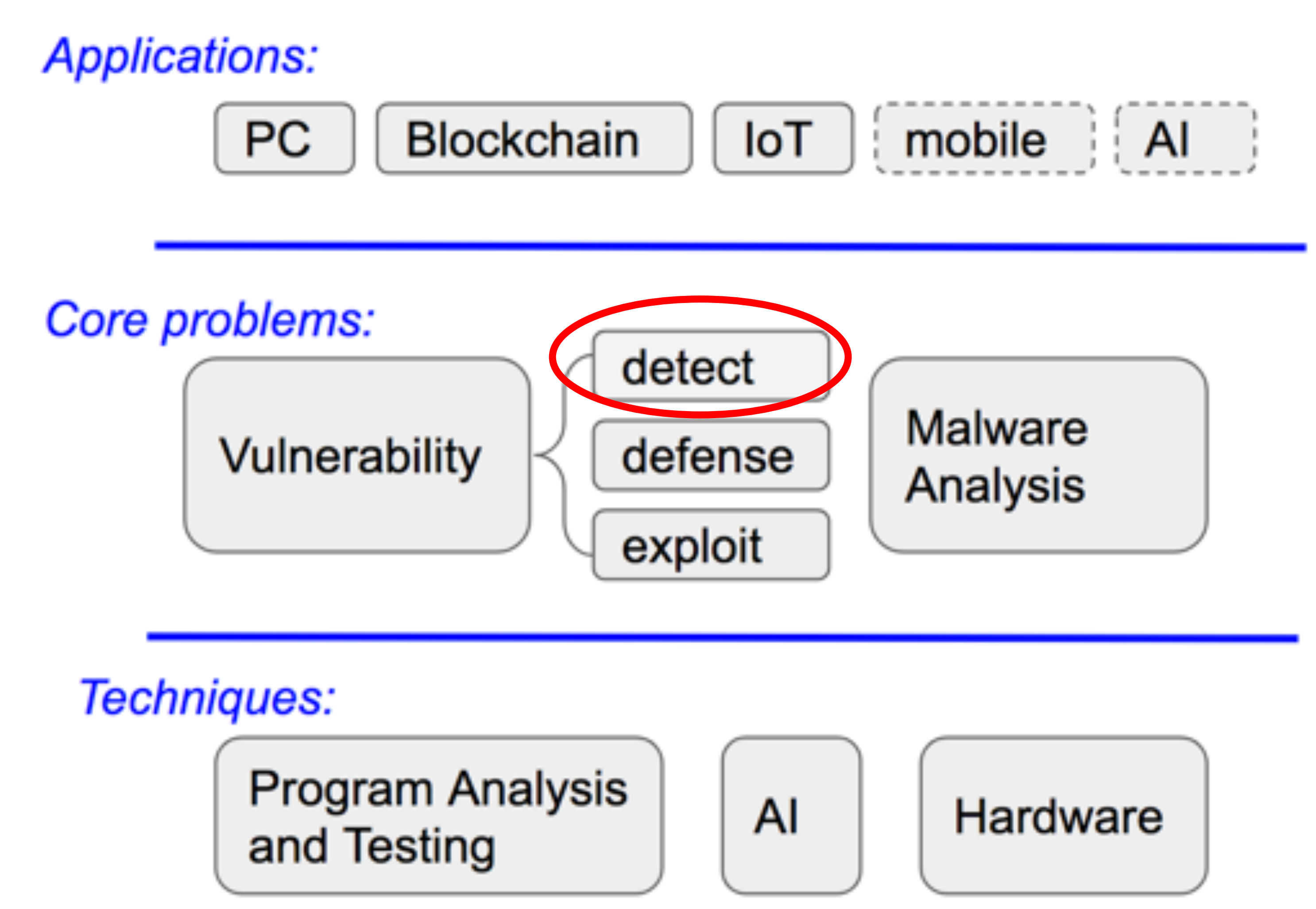
PEKING UNIVERSITY | Berkeley UNIVERSITY OF CALIFORNIA | Tsinghua University

- **Hack for fun** — *software and system security*
  - Automated vuln. discovery: Tencent CSS TSec 2nd Place, 300+ CVE
  - Automated exploit mitigation: Microsoft BlueHat Prize (Special Recognition Award)
  - Automated exploit generation: Tencent CSS TSec Breakthrough Prize (1st place)
  - Automated attack & defense: DARPA CGC (1st in defense 2015, 2nd in offense 2016)
  - Manual hacking: DEFCON CTF (2nd in 2016, 5th in 2015 and 2017)

- **Goal:** AlphaGo for software security.

To better defend yourself, know your enemy first. --- Sun Tzu

Research Interests



网络空间安全实验室

- 段海新教授，张超副教授，李琦副教授，诸葛建伟副研究员等
- 学术研究
  - 研究方向：网络、系统、应用安全（AI、物联网、区块链）
  - 学术成果：国际四大安全会议论文数量名列前茅
  - 实践应用：促进Google、微软、IETF等多次改进产品、协议标准安全性
- 组织发起
  - InForSec网络安全研究国际学术论坛
  - XCTF国际网络安全技术对抗联赛
  - “蓝莲花” “紫荆花”战队

没有什么能够阻挡

蓝莲花

BLUE-LOTUS

紫荆花

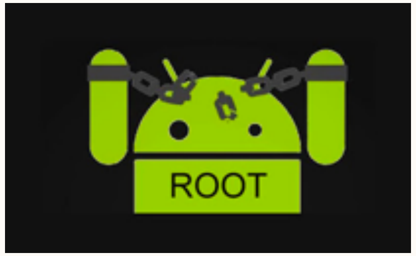
REDBUD

没有什么能够阻挡
你对自由的向往
…
…
如此的清澈高远
盛开着永不凋零
蓝莲花

欢迎热爱安全研究的同学们加入蓝莲花！（不限学校）

Vulnerability: Ghost in Cyberspace

- Valuable assets, root causes of most security incidents



Hacking Practice: DEFCON CTF

Blue-Lotus (coach)

- 2013 first time in DEFCON；
- 2014 5th place；
- 2015 5th place ；
- 2016 2nd place； (human vs. machine)
- 2017 5th place ;
- 2018 6th place
- **2019 3rd place**

Global

- 2013： ppp, men in black hats, raon_ASRT
- 2014： ppp, hitcon, dragonsector, blue-lotus
- 2015： defkor, ppp, 0daysober, hitcon, blue-lotus
- 2016： ppp, b1o0p, defkor, hitcon
- 2017： ppp, hitcon, a*0*e, defkor, tea-deliverers
- 2018： defkoroot, ppp, hitcon, a*0*e, sauercloud, tea-deliverers
- 2019: ppp, hitcon, tea-deliverers

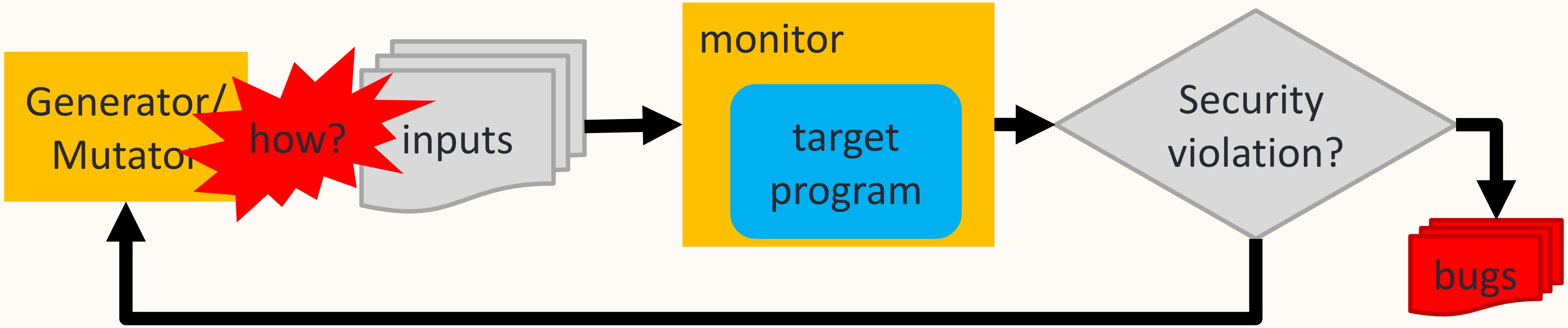
DARPA Cyber Grand Challenge
（Automated Offense and Defense）
（CodeJitsu Team Captain, CQE Defense #1，CFE Offense #2）

Vulnerability Discovery

- **Code Review** ***(10%?)***
- Static Analysis
- Dynamic Analysis
- Taint Analysis
- **Symbolic Execution**
- Model Checking
- **Fuzzing** ***(80%?)***

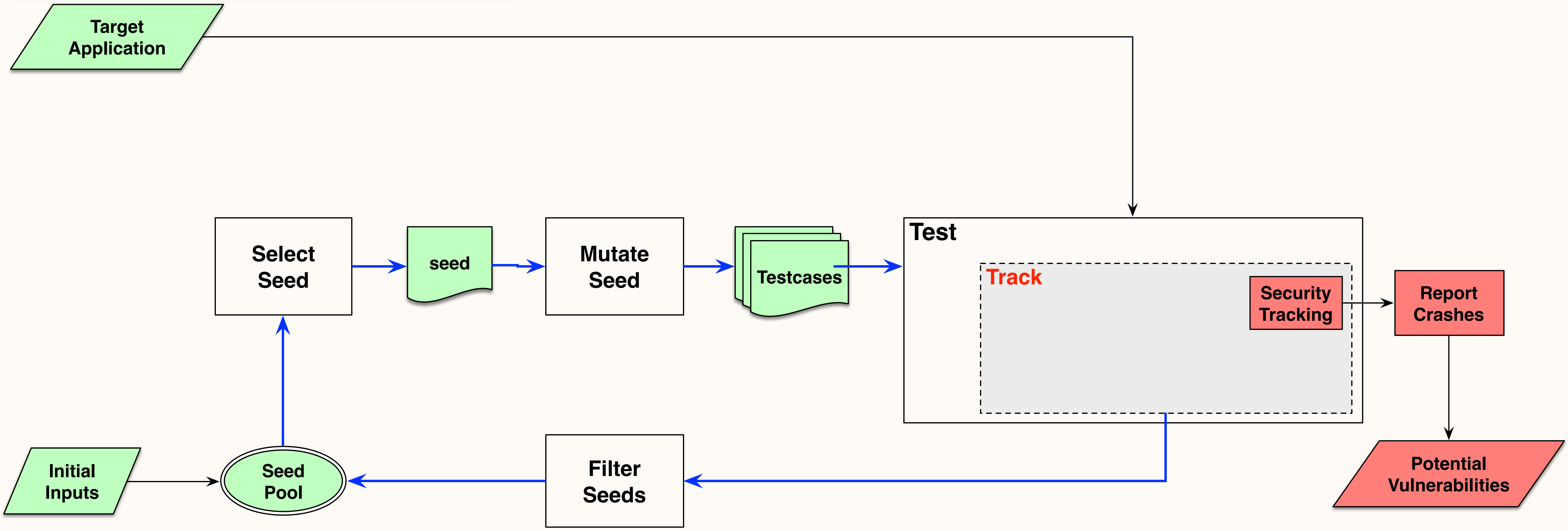
Fuzzing

- Goal:
  - Finding PoC samples that prove vulnerabilities
- Solution: testing



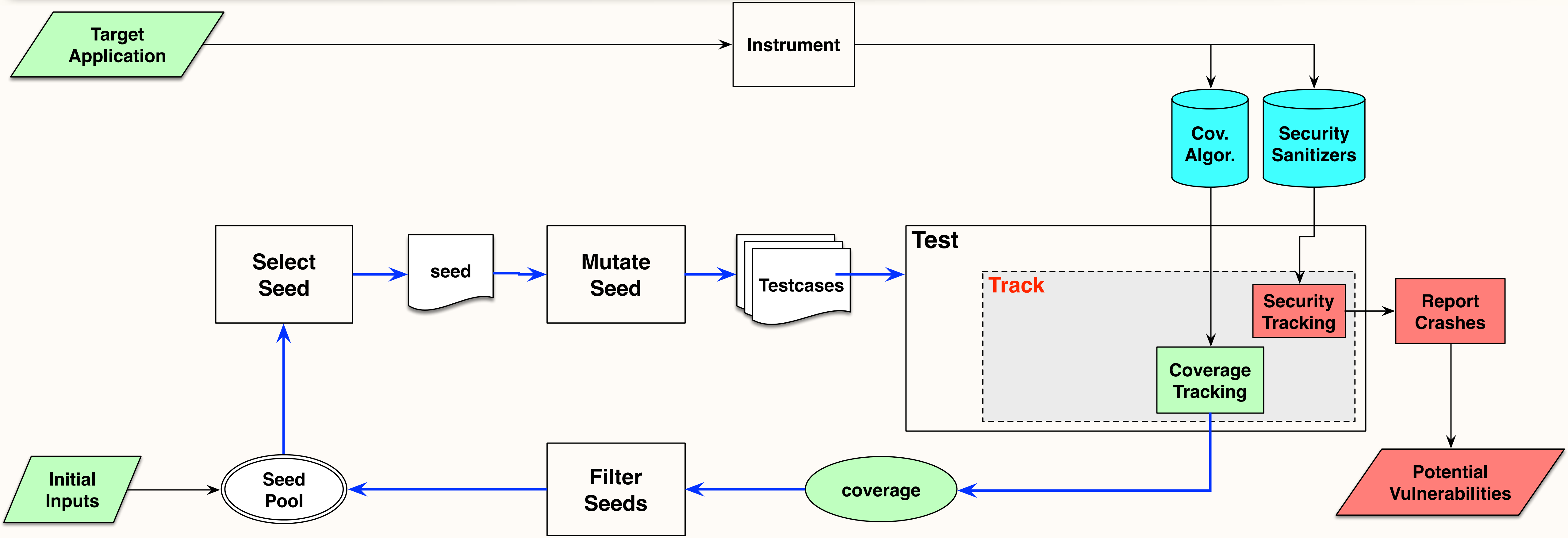
- Find needle in the haystack

A better strategy: Genetic Algorithm



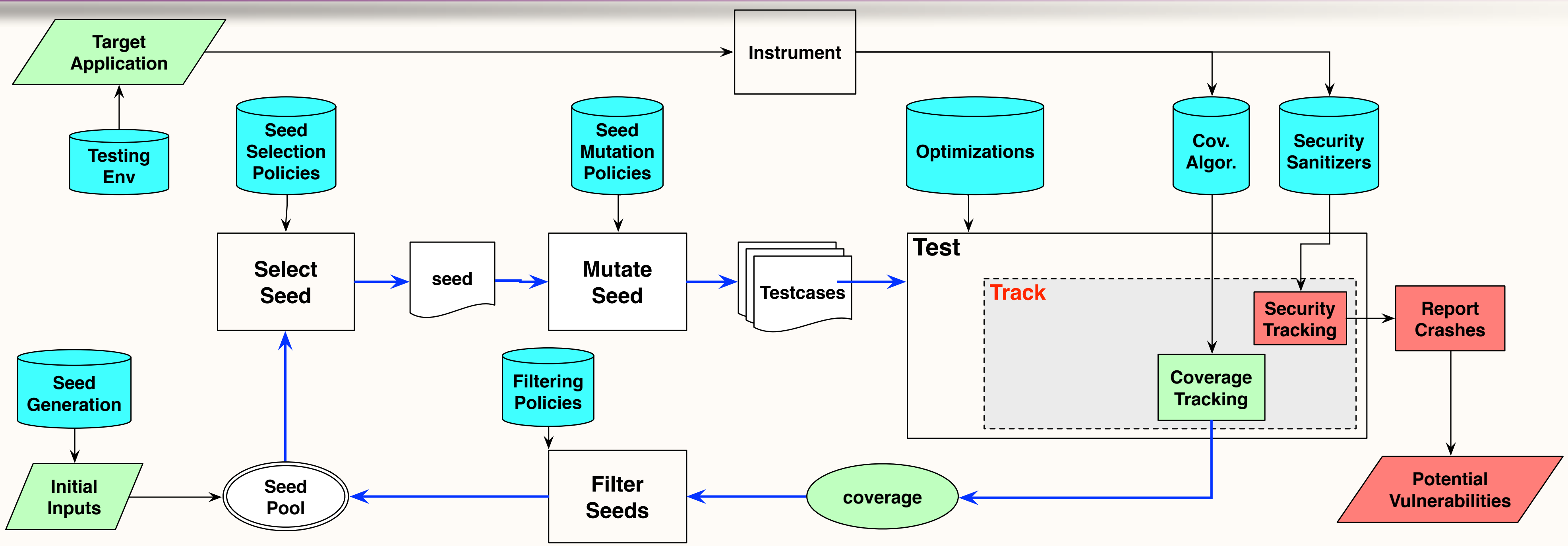
- Iterative testing, keep GOOD seeds, report bugs

A better strategy: Genetic Algorithm



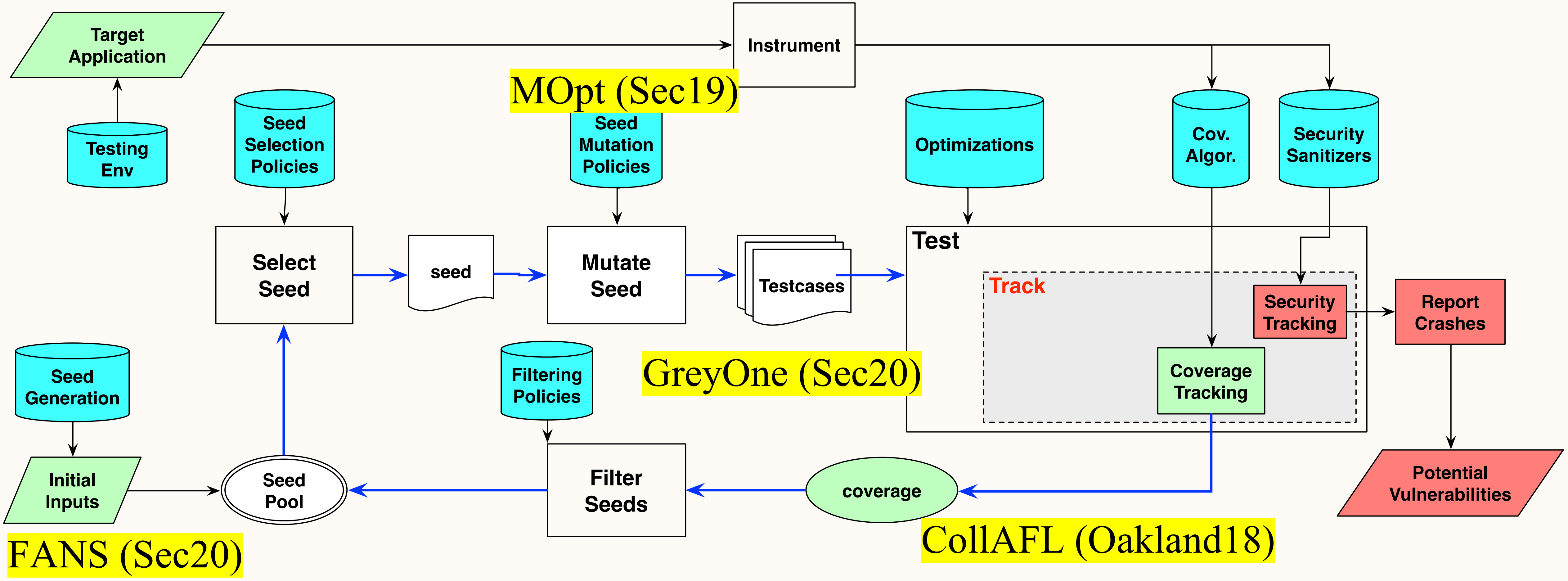
- GOOD: coverage increases
- Bugs: sanitizers

A pioneer tool: AFL



- Evolving: filter out only GOOD samples contributing to code coverage
- Scalable: mutation-based, few knowledge required
- Fast: fork-server, persistent, parallel
- Sensitive: support different sanitizers to catch security violations

Our works



HOTracer (Sec17)

Vul Dist (ICSE20)

Improvement 1: Coverage & Seed Selection

IEEE S&P 2018

CollAFL: Path Sensitive Fuzzing

Shuitao Gan[1], Chao Zhang[2]✉, Xiaojun Qin[1], Xuwen Tu[1], Kang Li[3], Zhongyu Pei[2], Zuoning Chen[4]

Observations (1)

- Collision in Coverage Tracking
  - *"The size of the map is chosen so that collisions are sporadic with almost all of the intended targets, which usually sport between 2k and 10k …"* -- *from AFL's description*

- AFL uses a 64KB bitmap to track edge coverage

```
; key: prev
Code in BB1
    ↓
; key: cur
hash = cur⊕(prev≫1)
bitmap[hash]++
Code in BB2
```

- Two edges may have a same hash
  - Discarding GOOD seeds
  - Discarding unique crashes
  - Providing inaccurate coverage info for fuzzing policies (e.g., seed selection)

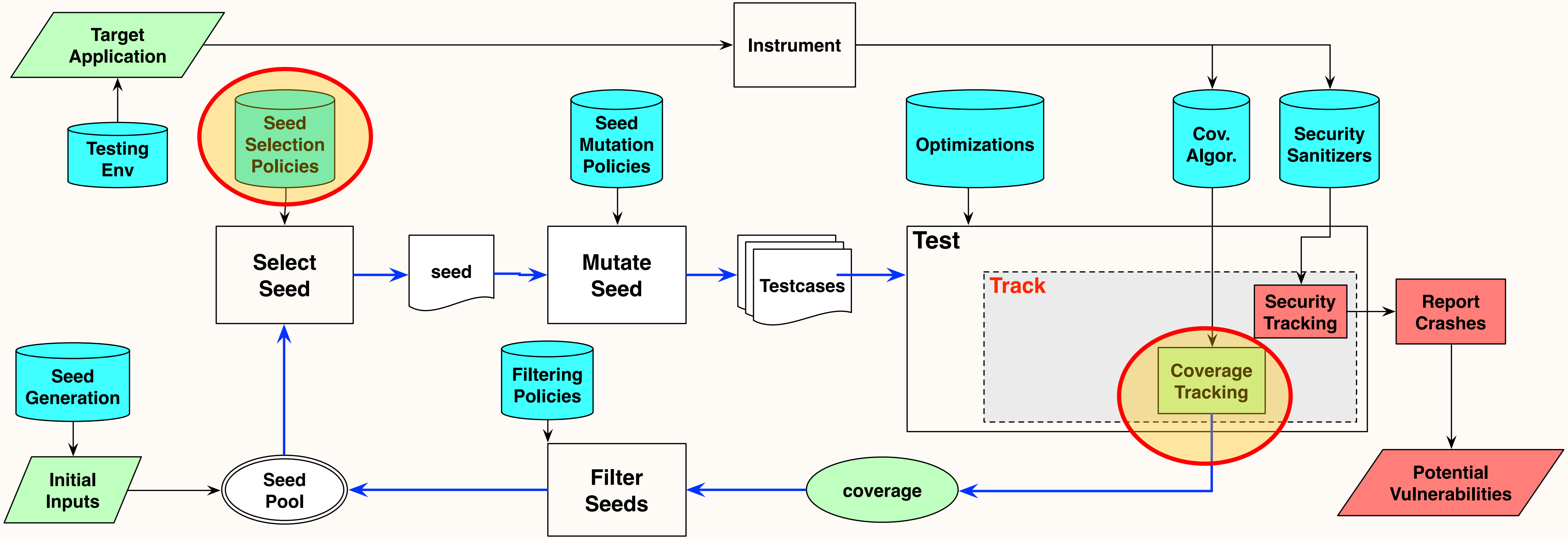
Applications	Size	#ins.	#BB	#edges	collision
LAVA(base64)	193KB	5570	822	1308	0.8%
LAVA(uniq)	208KB	5285	890	1407	0.92%
LAVA(md5sum)	234KB	7397	1013	1560	1.02%
LAVA(who)	1.52MB	84648	1831	3332	1.8%
catdoc	202KB	6448	841	1322	1.29%
libtasn1	540KB	12511	2163	3820	2.72%
cflow	688KB	24655	4286	7001	5.2%
libncurses	338KB	21486	4646	7883	5.57%
libtiff+tiffset	1.77MB	61119	8974	14826	10.4%
libtiff+tiff2ps	1.97MB	65932	9632	15927	10.84%
libtiff+tiff2pdf	2.1MB	71530	10507	17603	12.31%
libming+listswf	4.04MB	87148	11456	19154	13.61%
libdwarf	3MB	73921	11698	20260	13.7%
tcpdump	4.62MB	127082	18781	32656	21.2%
nm	8.72MB	218326	31611	53652	36.06%
bison	3.28MB	219268	42856	55658	32.8%
nasm	4.4MB	226665	41691	57411	33.38%
libpspp	5MB	259501	41323	71335	38.9%
objdump	11.88MB	305620	43935	74313	40.17%
clamav	11.35MB	347156	46140	81069	42.48%
exiv2+libexiv2	4.75MB	283284	59650	91287	45.87%
libsass+sassc	32.8MB	593570	68538	106738	50.7%
vim	14.7MB	478402	83877	153689	61.4%
libav	76.7MB	1776730	158009	255212	74.85%
libtorrent	97.5MB	1228513	164325	260485	75.29%

Observations (2)

- Few seed selection policies aim at increasing the code coverage directly
  - E.g., AFLfast, VUzzer, AFLgo, QTEP, SlowFuzz

- Coverage-first seed selection policies could reach higher code coverage faster.

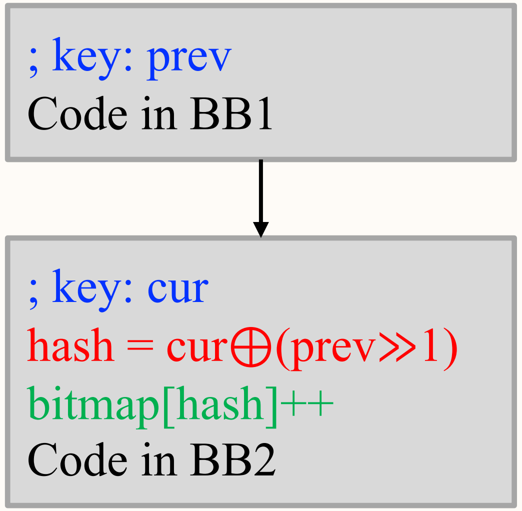
Our Solution: CollAFL



- Mitigate collision in coverage tracking
- Apply coverage-first seed selection policy

RQ1: Eliminate hash collisions

- AFL uses a 64KB bitmap to track edge coverage



Naïve solution: increase bitmap size

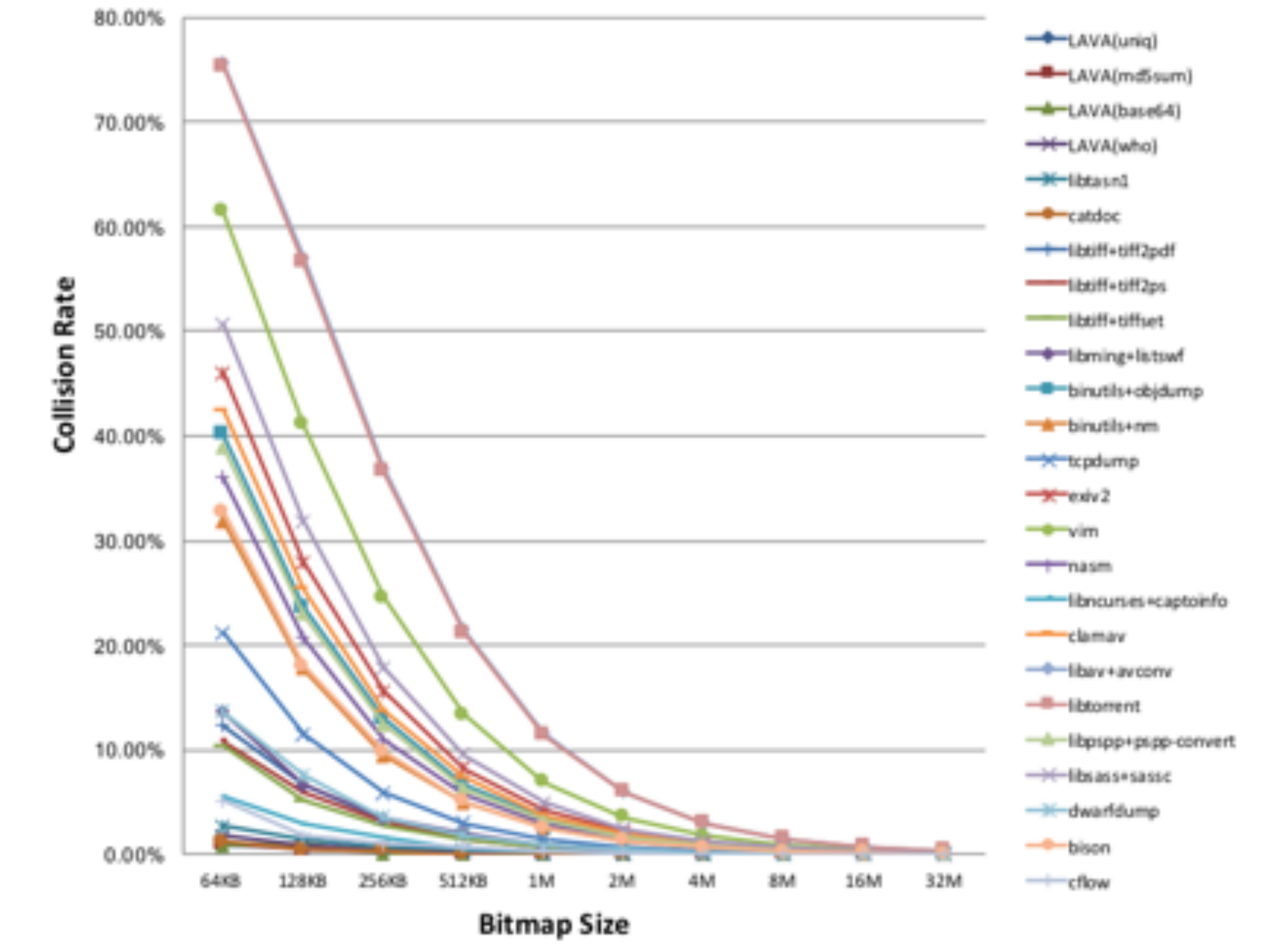


Fig. 4: Edge collision rate drops if enlarge bitmap size.

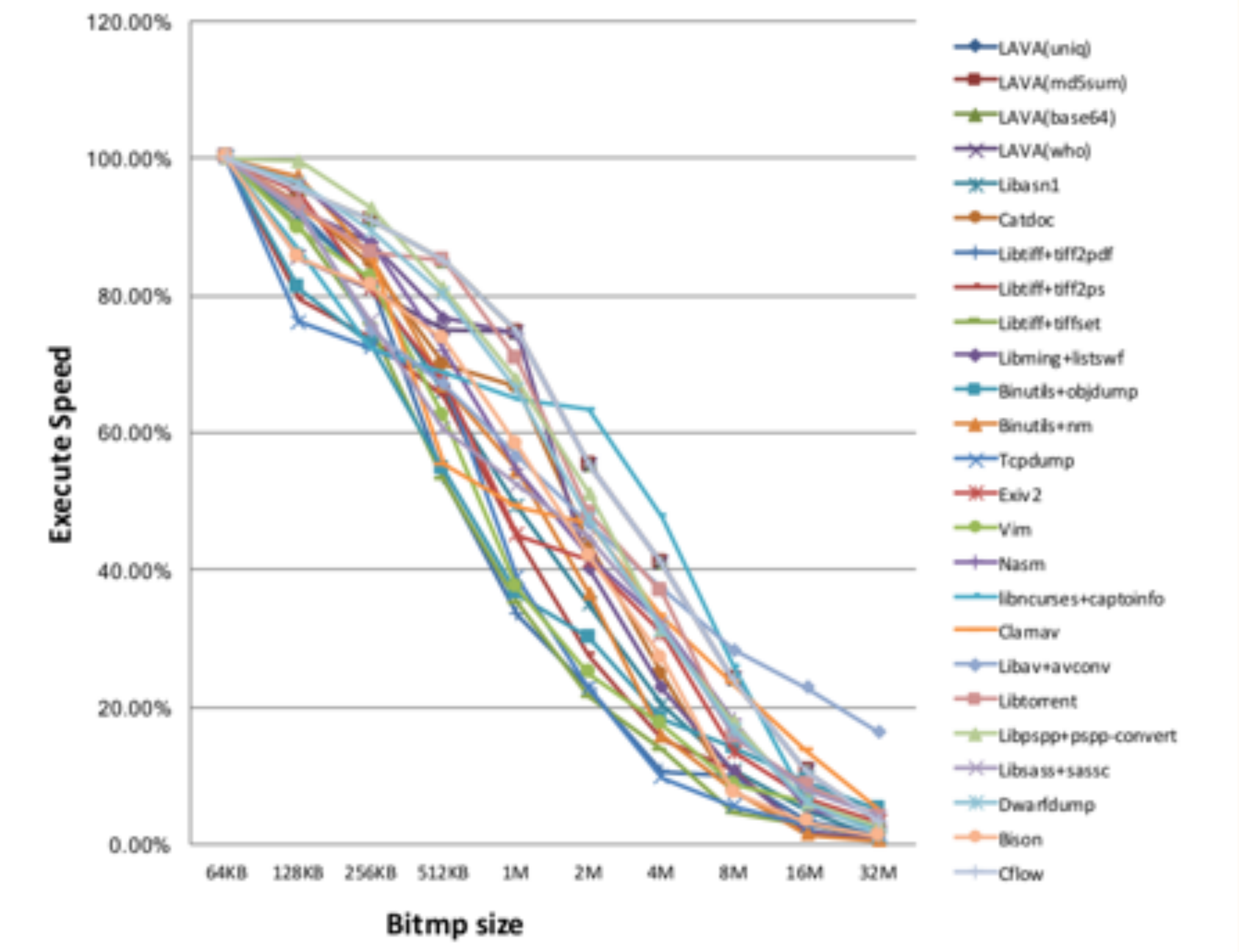
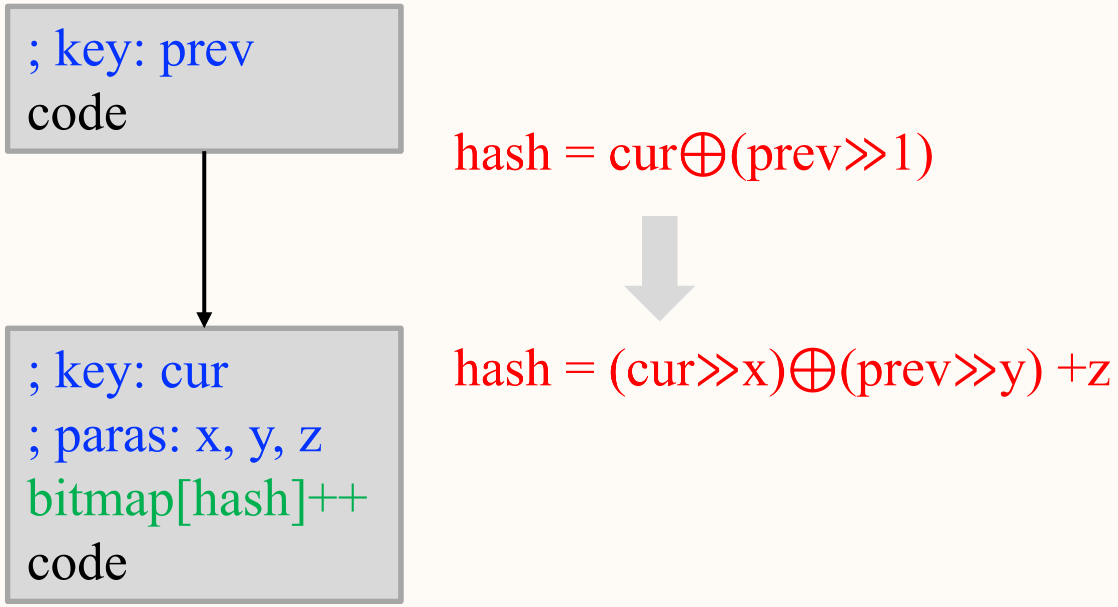


Fig. 5: Execution speed drops too if enlarge bitmap size.

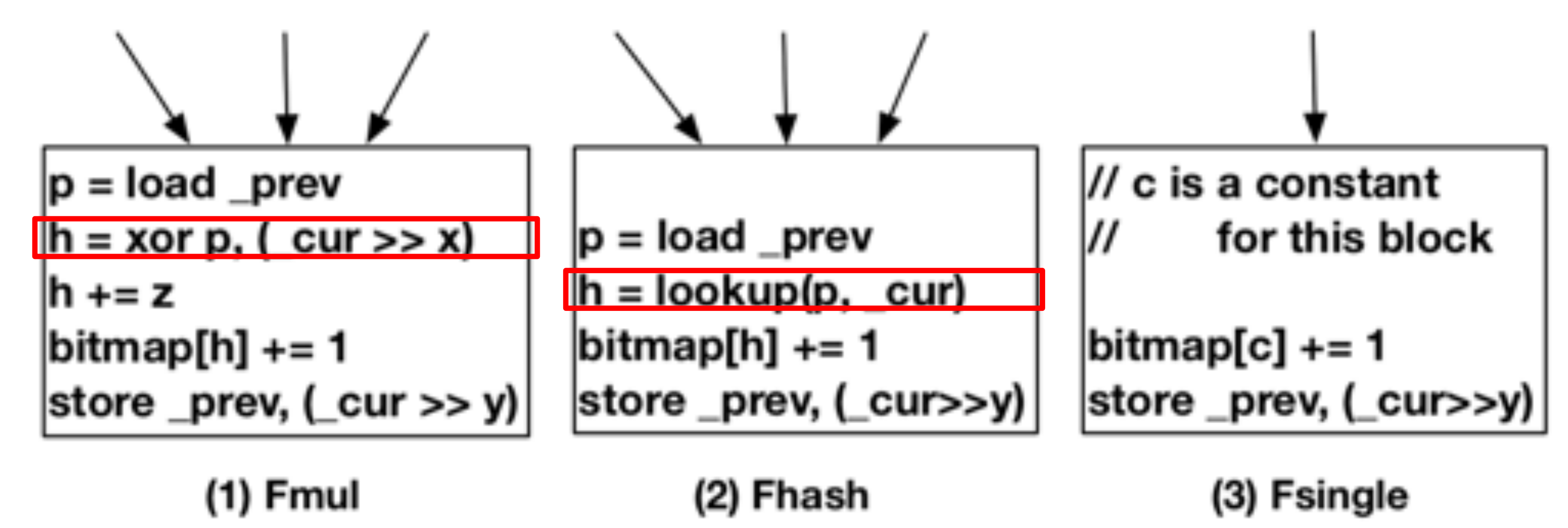
Our solution: intuition

- Replace the hash algorithm, without much performance loss



- Each block could have different combination of parameters x,y,z
- Search parameters x,y,z for all blocks one by one, to avoid collisions.
  - ***harder and harder to find parameters for remaining blocks.***

Our solution: in-a-nutshell



- Search parameters x,y,z for multi-precedent blocks
- Construct hash table for unsolvable multi-precedent blocks
- Assign un-used hashes to single-precedent blocks

Performance of Collision Mitigation

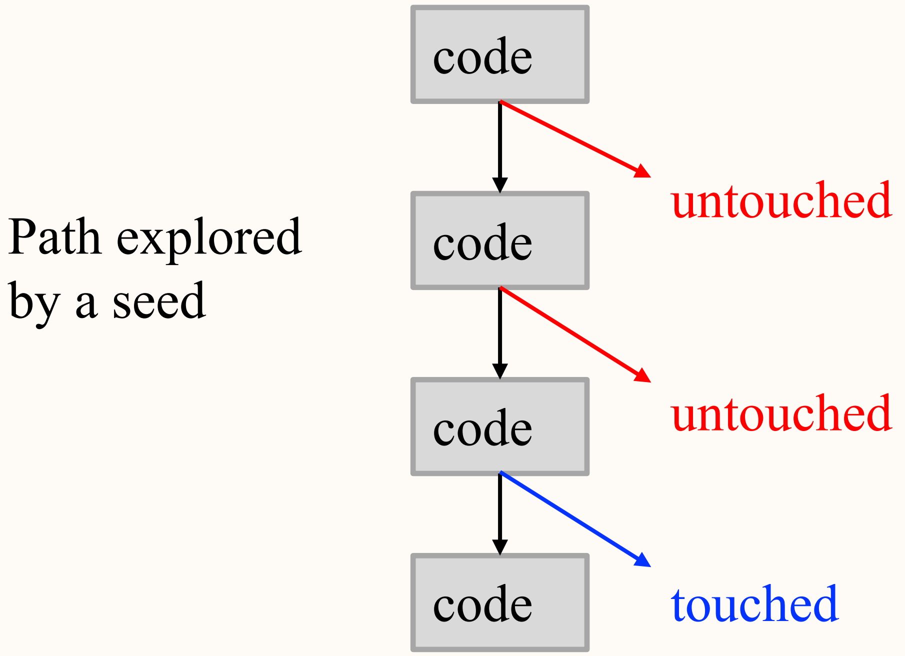
The bitmap will be enlarged when the edge count is larger than bitmap size, otherwise collision is inevitable.

Applications	bitmap size	AFL #ins.	CollAFL delta	Fmul	Fsingle	Fhash	coll. ratio
libncurses	64KB	37168	-2.93%	1779	2867	0	0
clamav	128KB	368912	-4.45%	14845	31269	26	0
	64KB	-	-	17573	28567	0	19.16%
libav	256KB	1264072	-0.6%	75068	82915	26	0
	64KB	-	-	10392	147617	0	74.32%
libtorrent	256KB	1314568	-2.91%	63012	101309	4	0
	64KB	-	-	10756	153569	0	74.84%
libpspp	128KB	330528	-3.15%	15444	25872	7	0
	64KB	-	-	16946	24377	0	8.13%
libsass	128KB	548296	-3%	26897	41640	1	0
	64KB	-	-	15785	52753	0	38.6%
libdwarf	64KB	93568	-5.03%	3494	8202	2	0
bison	64KB	342848	+1.36%	23760	19096	0	0
cflow	64KB	34288	-1.44%	1896	2390	0	0

Most BBs have only one precedent, saving hash computation and improving runtime performance.

RQ2: Coverage-first seed selection

- Prioritize seeds with more untouched branches



- Mutations on these seeds are more likely to exercise those untouched branches, contributing to coverage.

Evaluation: Code Coverage

- 20% more paths over AFL

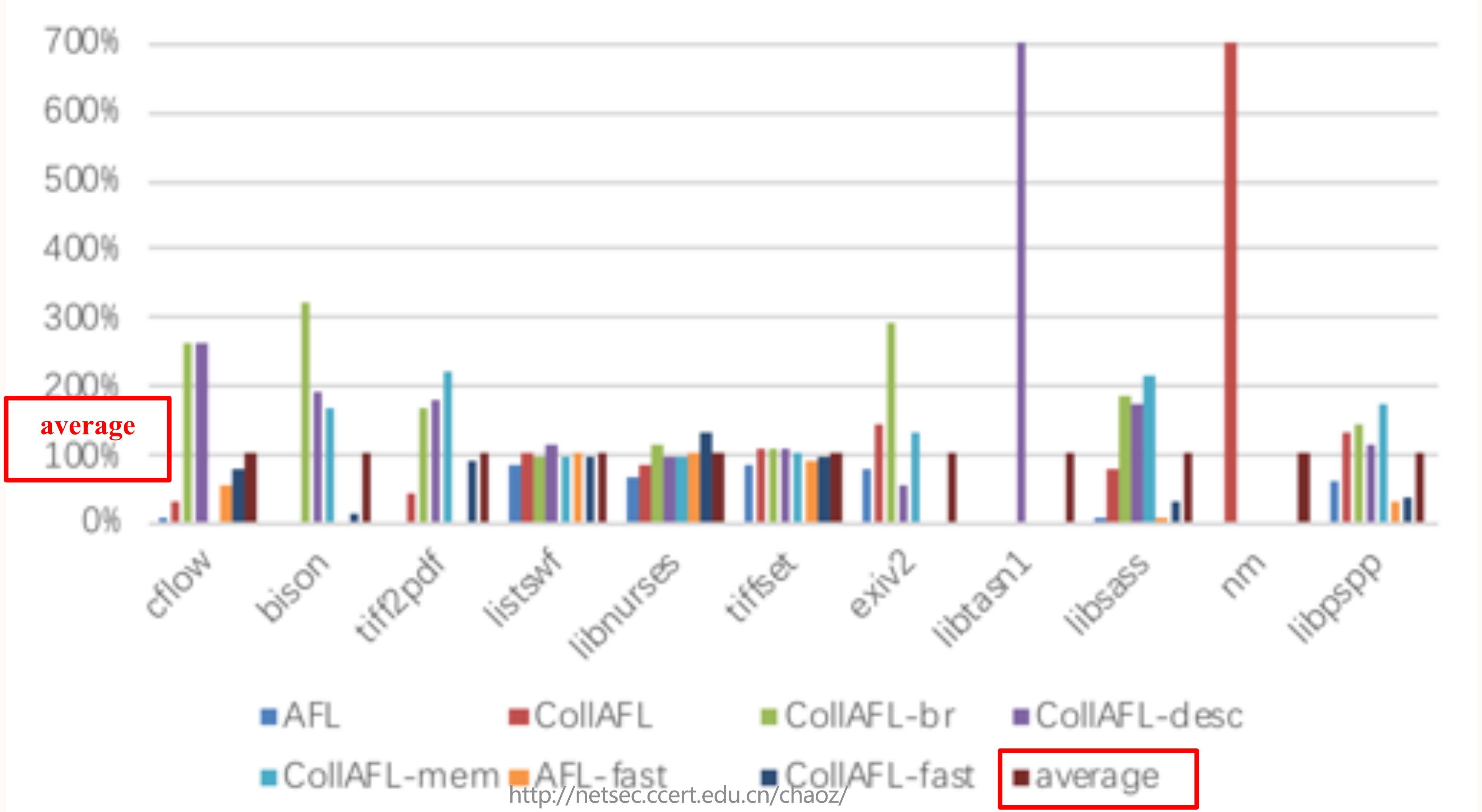
With extra untouched-branch seed selection policy

With collision mitigation only

Software	AFL	CollAFL	-br	-desc	-mem	AFL-fast	CollAFL-fast
Cflow	1080	+3.43%	+59.17%	+41.11%	+21.3%	1389	+7.27%
bison	1388	+9.51%	+50.94%	+75.36%	+63.04%	1969	+6.81%
tiff2pdf	5332	+5.46%	+11.7%	+14.12%	+10%	4979	+2.37%
listswf	4292	+1.34%	+6.85%	+3.36%	+0.07%	4104	+0.79%
libnurses	1529	+19.56%	+29.5%	+19.62%	+26.95%	1848	+0.6%
tiffset	1784	+0.73%	+5.04%	+10.82%	-4.37%	1616	-1.86%
exiv2	1209	+36.56%	+36.06%	+6.45%	+21.17%	201	+17.62%
libtasn1	465	+15.27%	+59.14%	+33.76%	+53.33%	511	+4.31%
libsass	8790	-1.37%	-0.61%	+3.69%	-1.66%	8771	-1.25%
nm	2389	+11.76%	-17.79%	-14.65%	-16.83%	1493	+47.15%
libpspp	2258	+6.64%	-11.43%	-4.07%	-0.27%	1772	+9.14%
Average	2774	+9.9%	+20.78%	+17.23%	+15.7%	2604	+8.45%

Evaluation: Crashes

- 320% more unique crashes than AFL (CollAFL-br)



Evaluation: Vulnerabilities

- 134 new bugs, 23 collided bugs, 95 CVE, 9 ACE

Applications	version	uniq crashes	vulnerabilities		AFL	CollAFL				unknown vulnerabilities	
			unknown	known		default	-br	-desc	-mem	CVE	ACE
libtiff	4.0.8	1569	10	3	1	7	10	8	6	7	2
libtasn1	4.12	1	1	0	0	0	0	1	0	1	0
libming	0.4.8	1303	2	4	2	2	3	4	4	2	0
libncurses	6.0	526	15	0	3	5	13	10	7	11	2
libexiv2	0.26	222	14	0	5	9	14	14	9	13	0
libsass	3.5.0	155	10	2	4	7	12	12	9	9	0
libpspp	0.10.5	412	10	2	4	5	10	10	12	6	0
bison	3.0.4	212	3	2	1	2	5	5	2	0	0
cflow	1.5	298	7	2	4	5	7	8	6	0	0
binutils	2.28	397	4	4	4	6	8	8	6	2	1
libav	12.1	239	2	0	1	1	2	2	1	2	0
tcpdump	4.9.0	10	3	0	1	2	2	3	2	2	0
clamav	0.99.2	12	1	0	0	1	1	1	1	1	0
libdwarf	20170416	14	1	0	1	1	1	1	0	1	0
libtorrent	1.1.3	177	1	0	0	1	1	1	1	1	0
nasm	2.14	1619	17	0	5	13	17	17	12	14	2
vim	8.0.679	28	3	0	1	2	3	3	2	1	1
catdoc	0.9.5	16	3	0	2	3	3	3	2	1	1
libgxps	0.2.5	32	1	0	1	1	1	1	1	1	0
Libmpg123	1.25.0	11	1	0	0	0	1	1	1	1	0
Libraw	0.18.2	14	1	0	0	0	1	1	0	1	0
Liblouis	3.2.0	38	10	0	4	5	8	7	6	7	0
Graphicmagick	1.3.26	88	4	0	2	3	4	4	3	2	0
jasper	2.0.12	122	10	4	5	7	14	14	6	9	0
Total	-	7501	134	23	51	88	141	139	99	95	9
Fraction of total vul.	-	-	85%	15%	32%	56%	90%	89%	63%	61%	4%

Improvement 2: Seed Mutation & Tracking

USENIX Security 2020

GreyOne: Data Flow Sensitive Fuzzing

Shuitao Gan[1], *Chao Zhang*[2,3]✉, *Peng Chen*[4], *Bodong Zhao*[2], *Xiaojun Qin*[1], *Dong Wu*[1], *Zuoning Chen*[5]

```
// magic number: direct copy of input[0:8] vs. constant
if(u64(input) == u64("MAGICHDR")){
    bug1();
}
// checksum: direct copy input[8:16] vs. computed val
if(u64(input+8) == sum(input+16, len-16)){
    bug2();
}
// length: direct copy of input[16:18] vs. constant
if( u16(input+16) > len )) { bug3(); }
// indirect copy of input[18:20]
if(foo(u16(input+18))==...){ bug4(); }
// implicit dependency: var1 depends on input[20:24]
if(u32(input+20) == ...){
    var1 = ...;
}
// var1 may change if input[20:24] changes
// FTI infers: var1 depends on input[20:24]
if(var1 == ...){    bug5(); }
```

- **Where to mutate?**
  - input[0:8]
- **How to mutate?**
  - MAGICHDR
- **Seed prioritization**
  - 1 byte match, vs.
  - 7 bytes match

Data flow information is useful for fuzzing

What types of data-flow features?

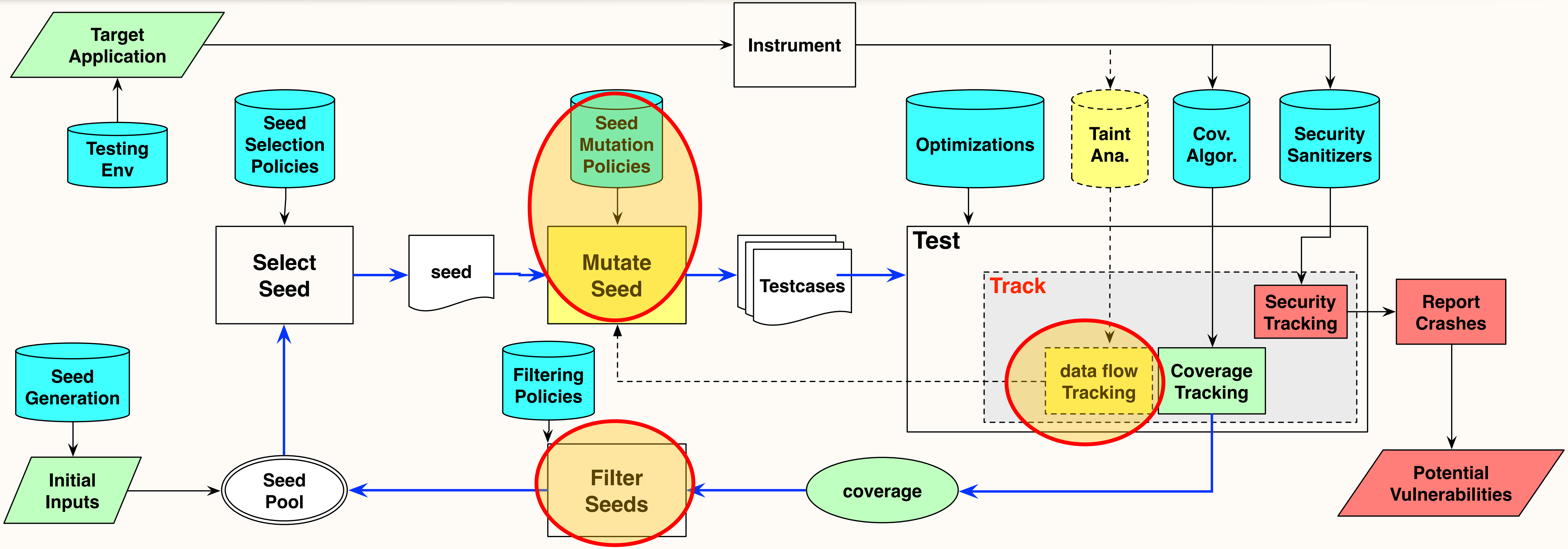
- **Taint attributes**
  - Dependency between inputs and variables

- **Branch value conformance**
  - Distance between branch condition operands

$$C_{br}(br, S) = NumEqualBits(var1, var2)$$

  - The higher conformance, the closer distance

Our Solution: GreyOne



- Data flow tracking
- Guided seed mutation
- Data sensitive evolving

RQ1: How to efficiently get data-flow features?

*** taint attributes**

*** branch value conformance**

RQ2: How to utilize data-flow features to guide mutation?

RQ3: How to utilize data-flow features to tune fuzzing direction?

RQ1-1: Taint Attributes

- Traditional dynamic taint analysis
  - Libdft/DFSan…
  - Propagate taint inst. by inst.
  - Taint rules manually/automatically
  - Under-taint and over-taint issues

```
//under-taint: missing taint model
var1 = externalCall(u32(input));
//br1 depends on [0,1,2,3]
if(var1 > ... ){
...
}
//over-taint: bit masking
var2 = var1 & 0xFFFF
//br2 depends on [0,1]
if(var2 == ...){
...
}
//under-taint: implicit control flow
while(var2--){
var3++;
}
```

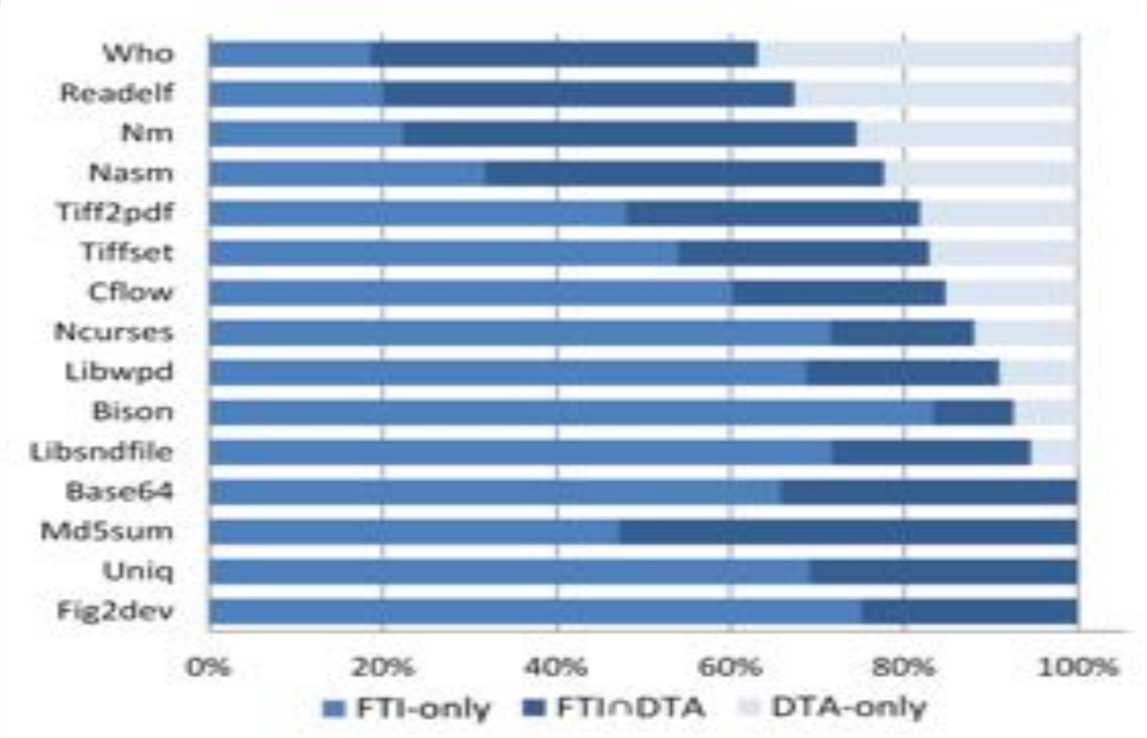
- Fuzzing-driven Taint Inference (FTI)
  - Interference rule

$$v(var, S) \neq v(var, S[i])$$

  - Taint inference
    - Byte-level mutation
    - Branch variable monitoring
    - Deterministic fuzzing stage

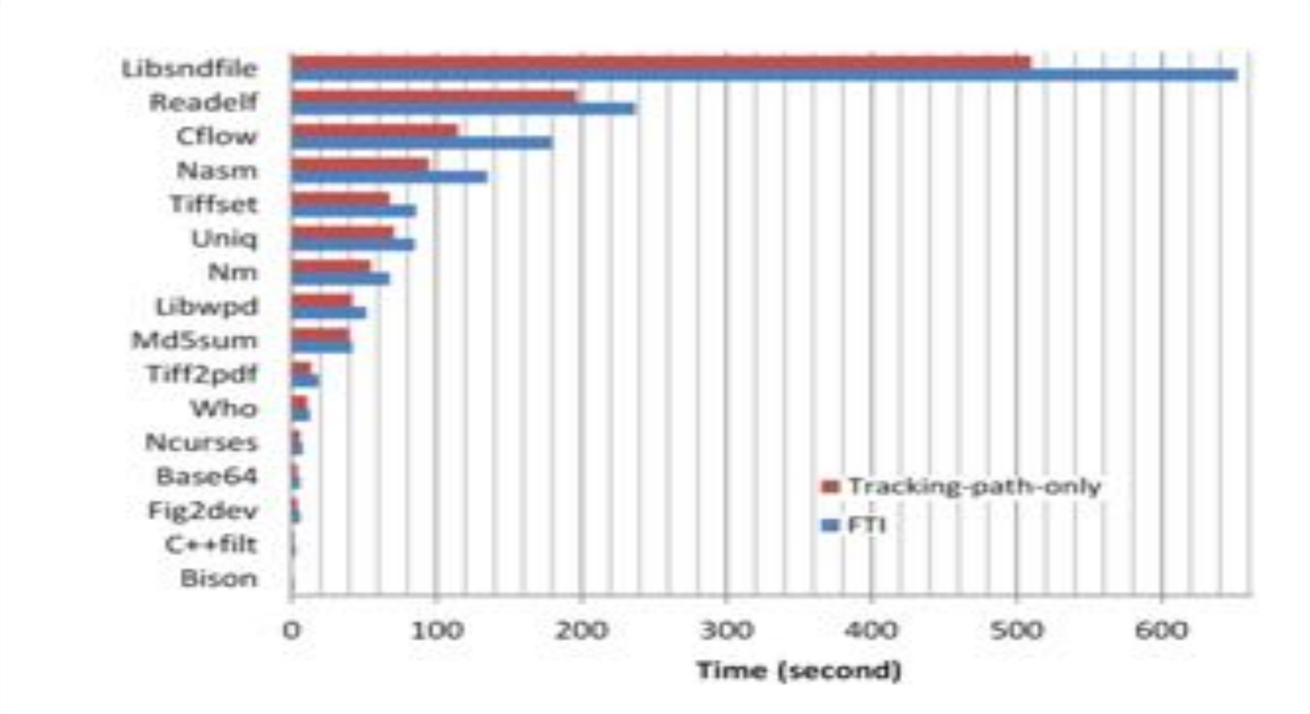
- Comparison
  - Speed: faster
  - Manual efforts: none, arch-independent
  - No over-taint
  - less under-taint

Performance of FTI



Proportion of tainted untouched branches reported

- ✓ FTI outperforms the classic taint analysis solution DFSan
- ✓ FTI finds 1.3X more untouched branches that are tainted



Average speed of analyzing one seed by FTI

- ✓ FTI brings 25% overhead on average

RQ1-2: Constraint Conformance

Conformance of constraints

- ✓ Expressing the distance of tainted variables to values expected in untouched branches
- ✓ Higher conformance means lower complexity of mutation

Q1: How to evaluate single constraint?
Q2: How to evaluate a set of constraints?

Conformance of one branch

$$C_{br}(br, S) = NumEqualBits(var1, var2)$$

Conformance of a basic block

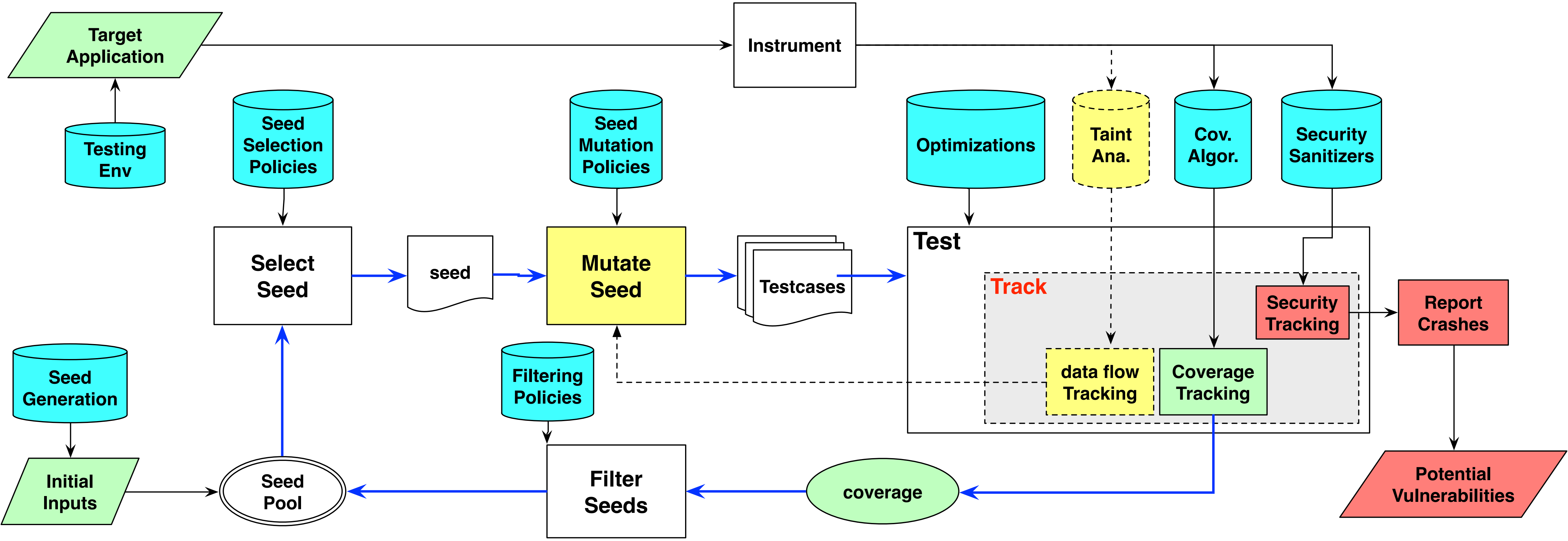
$$C_{BB}(bb, S) = \max_{br \in Edges(bb)} IsUntouched(br) * C_{br}(br, S)$$

Conformance of one path

$$C_{seed}(S) = \sum_{bb \in Path(S)} C_{BB}(bb, S)$$

Features

- ✓ Low instrumentation overhead
- ✓ Keep the original construct of program
- ✓ Able to evaluate conformance for comparisons between non-constant variables



Where and how to mutate?

RQ2: taint-guided mutation (how)

How to mutate direct copies of input?

- ✓ Direct copies
  - ◆ Magic number, Checksum…
- ✓ Execute twice
  - ◆ First round
    - ◆ FTI taint analysis: input offsets, expected value
  - ◆ Second round
    - ◆ Mutate and test

How to mutate indirect copies of input?

- ✓ Random bit flipping and arithmetic operations on each dependent byte
- ✓ Multiple dependent bytes could be mutated together

Mitigate the under-taint issue

- ✓ Randomly mutate their adjacent bytes with a small probability

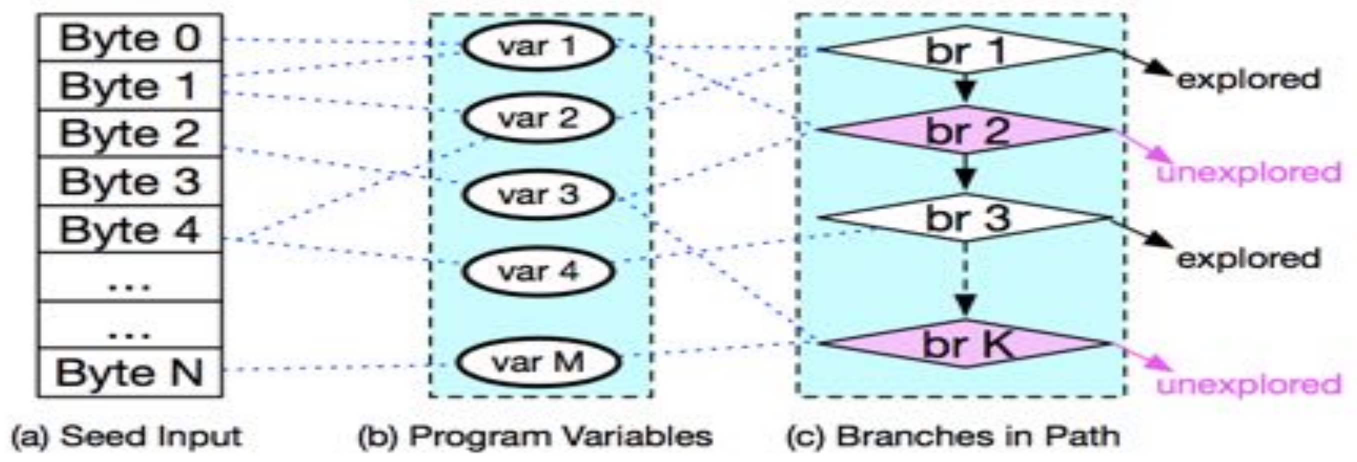
RQ2: taint-guided mutation (where)

Where to mutate?

- ✓ Explore the untouched neighbor branches along this path one by one
  - ◆ In descending order of branch weight
- ✓ For specific untouched neighbor branch
  - ◆ Mutating its dependent input bytes one by one
  - ◆ In descending order of byte weight

RQ2: taint-guided mutation (order)

- Inputs may affect program variables, which may influence branches

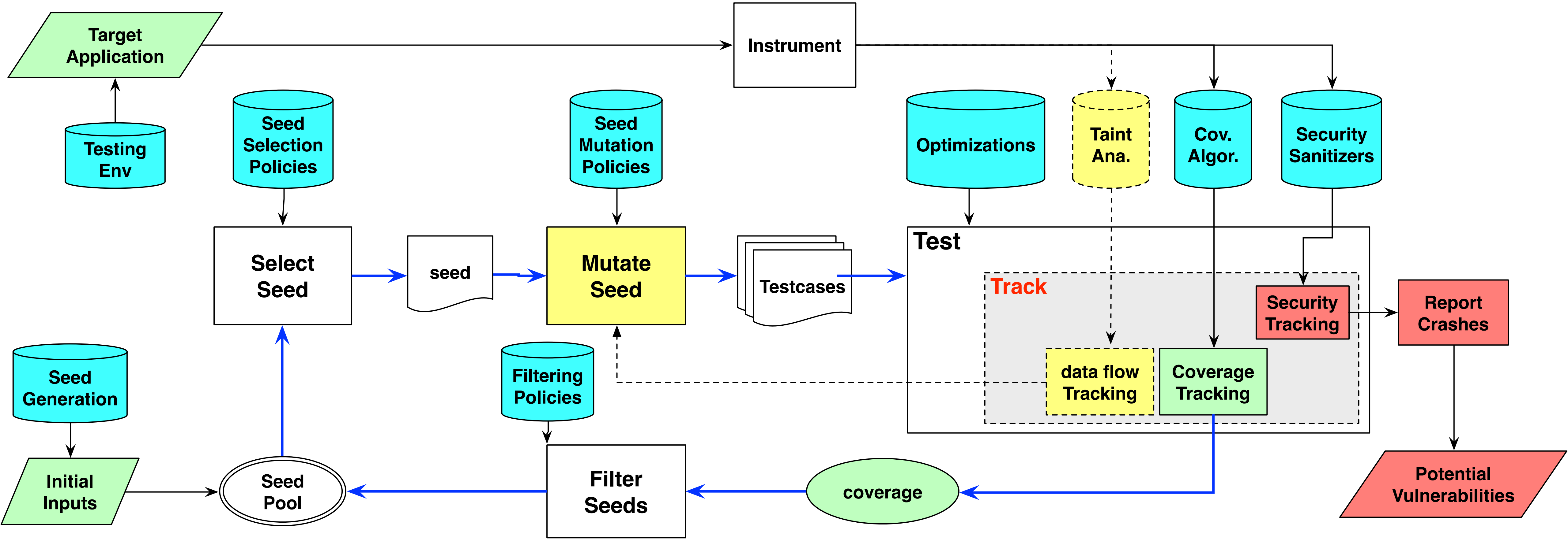


- Prioritize bytes to mutate: **affecting more untouched branches**

$$W_{byte}(S,pos) = \sum_{br \in Path(S)} IsUntouched(br) * DepOn(br,pos)$$

- Prioritize branches to explore: **depending on more high-weight bytes**

$$W_{br}(S,br) = \sum_{pos \in S} DepOn(br,pos) * W_{byte}(S,pos)$$

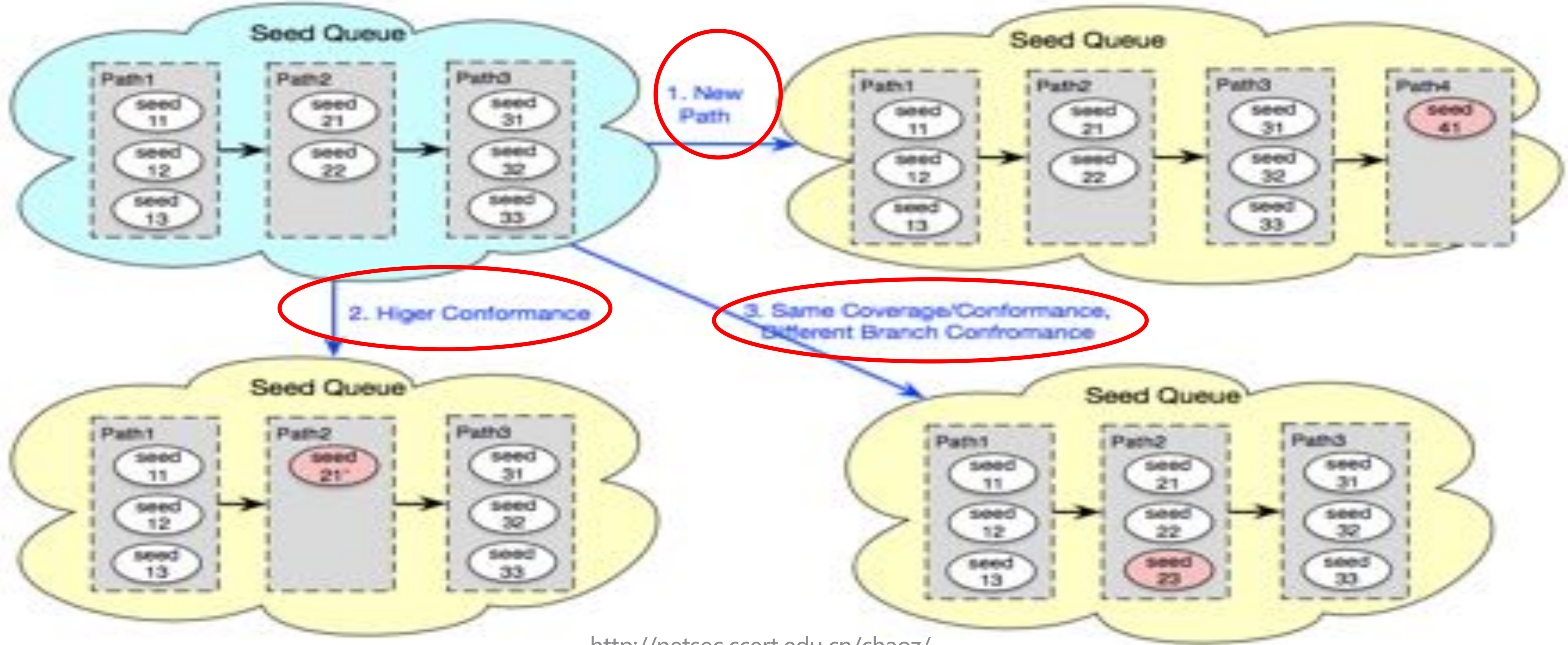


Tune evolution direction with Branch Conformance

RQ3: Conformance-guided evolution

- Updating seed queues:
  - **the higher conformance, the better**
  - together with AFL's policy: **coverage-guided**

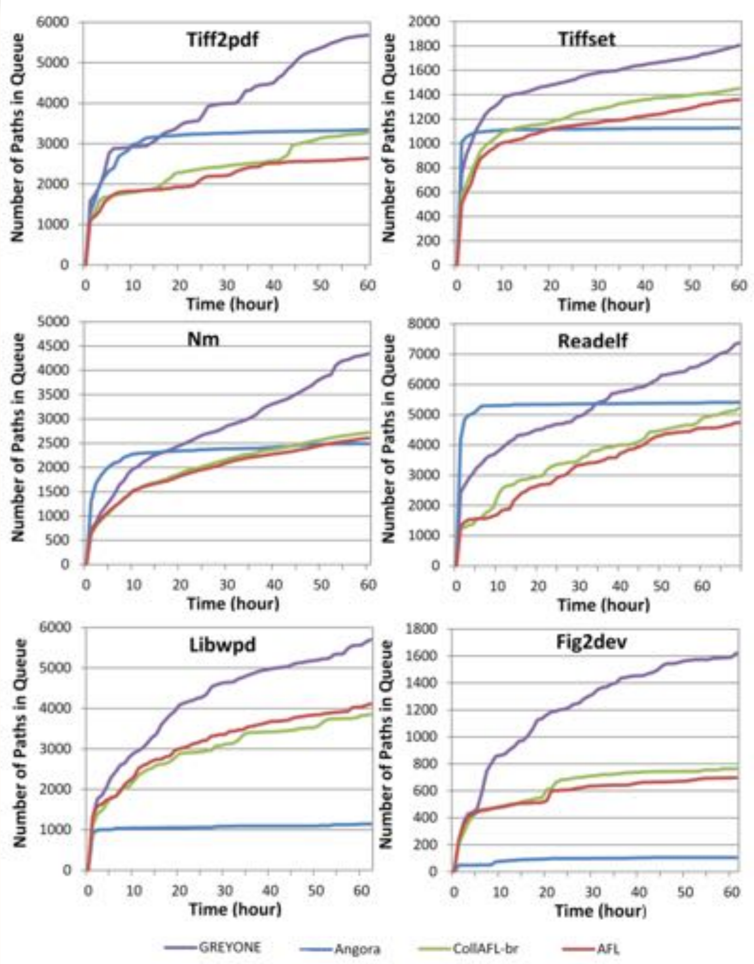
- New coverage
- Same coverage, higher path conformance
- Same coverage, same path conformance, different branch conformance



Evaluation: Code Coverage

Applications	Path Coverage				Edge Coverage			
	AFL	CollAFL-br	Angora	GREYONE (INC)	AFL	CollAFL-br	Angora	GREYONE (INC)
tiff2pdf	2638	3278	3344	5681(+69.9%)	6261	6776	6820	8250(+20.9%)
readelf	4519	4782	5212	6834(+32%)	6729	6955	7395	8618(+14.5%)
fig2dev	697	764	105	1622(+112%)	934	1754	489	2460(+40.2%)
ncurses	1985	2241	1024	2926(+30.6%)	2082	2151	1736	2787(+28.2%)
libwpd	4113	3856	1145	5644(+37.2%)	5906	5839	4034	7978(+35.1%)
c++filt	9791	9746	1157	10523(+8%)	6387	6578	3684	7101(+8%)
nasm	7506	7354	3364	9443(+25.8%)	6553	6616	4766	8108(+22.5%)
tiffset	1373	1390	1126	1757(+26%)	3856	3900	3760	4361(+11.8%)
nm	2605	2725	2493	4342(+59%)	5387	5526	5235	8482(+53.5%)
libsndfile	911	848	942	1185(+25.8%)	2486	2392	2525	2975(+17.8%)

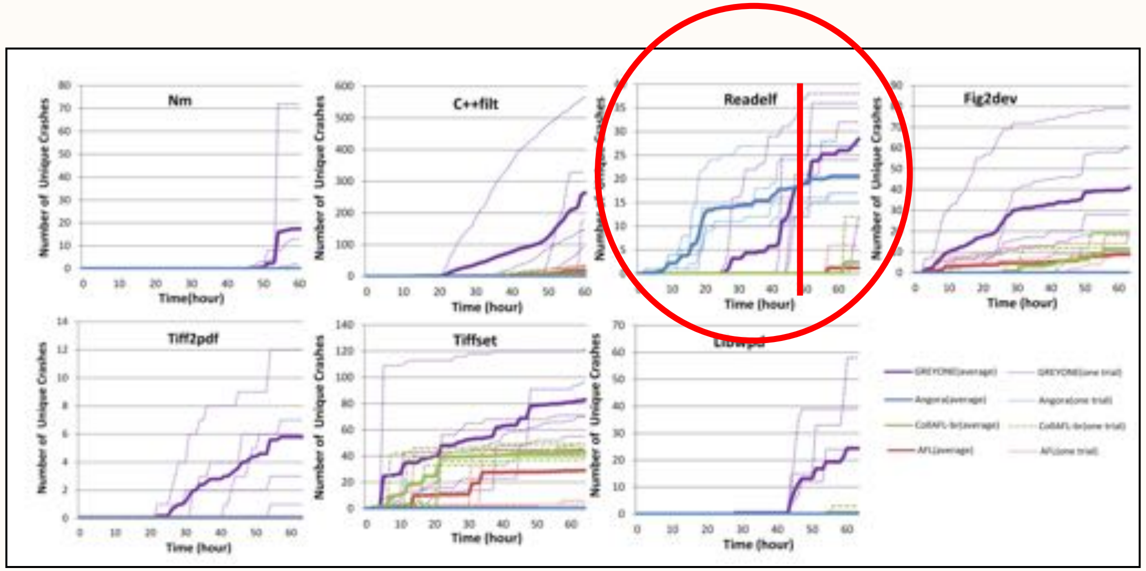
Number of unique crashes (average and maximum count in 5 runs) found in real world programs by various fuzzers



The growth trend of number of unique paths (average in 5 runs) detected by AFL, CollAFL-br, Angora and GREYONE

Unique Crashes Evaluation

Applications	AFL		CollAFL-br		Angora		GREYONE	
	Average	Max	Average	Max	Average	Max	Average	Max
tiff2pdf	0	0	0	0	0	0	6	12
libwpd	0	0	1	3	0	0	21	58
fig2dev	8	12	11	20	0	0	40	79
readelf	0	0	0	0	21	27	28	38
nm	0	0	0	0	0	0	16	72
c++filt	18	30	7	32	0	0	268	575
ncurses	7	18	12	23	0	0	28	37
libsndfile	4	13	8	20	0	0	23	33
libbson	0	0	0	0	0	0	6	12
tiffset	22	46	43	49	0	0	83	122
libsass	0	0	0	0	0	0	8	12
cflow	9	47	17	35	0	0	32	185
nasm	5	15	20	42	6	12	157	212
Total	73	181	119	229	27	39	716 (+501%)	447 (+631%)



Number of unique crashes (average and maximum count in 5 runs) found in real world programs by various fuzzers

The growth trend of number of unique crashes (average and each of 5 runs) detected by AFL, CollAFL-br, Angora and GREYONE

Evaluation: Vulnerabilities

Applications	Version	AFL	CollAFL- br	Honggfuzz	VUzzer	Angora	GREYONE	Vulnerabilities Unknown	Vulnerabilities Known	CVE
readelf	2.31	1	1	0	0	3	4	2	2	-
nm	2.31	0	0	0	0	0	2	1	1	*
c++filt	2.31	1	1	1	0	0	4	2	2	*
tiff2pdf	v4.0.9	0	0	0	0	0	2	1	1	0
tiffset	v4.0.9	1	2	0	0	0	2	1	1	1
fig2dev	3.2.7a	1	3	2	0	0	10	8	2	0
libwpd	0.1	0	1	0	0	0	2	2	0	2
ncurses	6.1	1	1	0	0	0	4	2	2	2
nasm	2.14rc15	1	2	2	1	2	12	11	1	8
bison	3.05	0	0	1	0	2	4	2	2	0
cflow	1.5	2	3	1	0	0	8	4	4	0
libsass	3.5-stable	0	0	0	0	0	3	2	1	2
libbson	1.8.0	1	1	1	0	0	2	1	1	1
libsndfile	1.0.28	1	2	2	1	0	2	2	0	1
libconfuse	3.2.2	1	2	0	0	0	3	2	1	1
libwebm	1.0.0.27	1	1	0	0	0	1	1	0	1
libsolv	2.4	0	0	3	2	2	3	3	0	3
libcaca	0.99beta19	2	4	1	0	0	10	8	2	6
liblas	2.4	1	2	0	0	0	6	6	0	4
libslax	20180901	3	5	0	0	0	10	9	1	*
libsixl	v1.8.2	2	2	2	2	3	6	6	0	6
libxsmm	release-1.10	1	1	2	0	0	5	4	1	3
Total	-	21	34	18	6	12	105 (+209%)	80	25	41

19 popular applications

2X more vulnerabilities (41 CVEs)

Number of vulnerabilities (accumulated in **5 runs**) detected by 6 fuzzers, including AFL, CollAFL-br, VUzzer, Honggfuzz,Angora, and GREYONE, after testing each application for **60 hours**

CVEs

libwpd	CVE-2017-14226, CVE-2018-19208
libtiff	CVE-2018-19210
libbson	CVE-2017-14227,
libncurses	CVE-2018-19217, CVE-2018-19211
libsass	CVE-2018-19218, CVE-2018-19218
libsndfile	CVE-2018-19758
nasm	CVE-2018-19213, CVE-2018-19215, CVE-2018-19216, CVE-2018-20535, CVE-2018-20538, CVE-2018-19755
libwebm	CVE-2018-19212
libconfuse	CVE-2018-19760
libsixel	CVE-2018-19757, CVE-2018-19756, CVE-2018-19762, CVE-2018-19761, CVE-2018-19763, CVE-2018-19763
libsolv	CVE-2018-20533, CVE-2018-20534, CVE-2018-20532
libLAS	CVE-2018-20539, CVE-2018-20536, CVE-2018-20537, CVE-2018-20540
libxsmm	CVE-2018-20541, CVE-2018-20542, CVE-2018-20543
libcaca	CVE-2018-20545, CVE-2018-20546, CVE-2018-20547, CVE-2018-20548, CVE-2018-20544, CVE-2018-20544

Libxsmm: CVE-2018-20541

```
There is a  heap-buffer-overflow in libxsmm_sparse_csc_reader  at src/generator_spgemm_csc_reader.c:174
src/generator_spgemm_csc_reader.c:122) in libxsmm.

Description:

The asan debug is as follows:

$./libxsmm_gemm_generator sparse b a 10 10 10 1 1 1 1 1 1 0 wsm nopf SP POC0

=================================================================
==51909==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60200000eff0 at pc 0x000000444875 b
WRITE of size 4 at 0x60200000eff0 thread T0
    #0 0x444874 in libxsmm_sparse_csc_reader src/generator_spgemm_csc_reader.c:174
    #1 0x405751 in libxsmm_generator_spgemm src/generator_spgemm.c:279
    #2 0x40225a in main src/libxsmm_generator_gemm_driver.c:318
    #3 0x7f73105a0a3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x20a3f)
    #4 0x402ea8 in _start (/home/company/real_sanitize/poc_check/libxsmm/libxsmm_gemm_generator_asan+0x

0x60200000eff1 is located 0 bytes to the right of 1-byte region [0x60200000eff0,0x60200000eff1)
allocated by thread T0 here:
    #0 0x7f7310c009aa in malloc (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x989aa)
    #1 0x443f78 in libxsmm_sparse_csc_reader src/generator_spgemm_csc_reader.c:122
    #2 0x7ffc367e92bf  (<unknown module>)
    #3 0x439  (<unknown module>)
```

Libsixel:CVE-2018-19757

```
$./img2sixel   POC2
=================================================================
==624==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60200000a7b1 at pc 0x7fcd853aa04c bp 0x7ffd2dcd54d0 sp
0x7ffd2dcd4c78
WRITE of size 67108863 at 0x60200000a7b1 thread T0
    #0 0x7fcd853aa04b in __asan_memset (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x8d04b)
    #1 0x7fcd8508bf10 in memset /usr/include/x86_64-linux-gnu/bits/string3.h:90
    #2 0x7fcd8508bf10 in image_buffer_resize /home/company/real_sanitize/libsixel-master/src/fromsixel.c:311
    #3 0x7fcd8508d5d4 in sixel_decode_raw_impl /home/company/real_sanitize/libsixel-master/src/fromsixel.c:565
    #4 0x7fcd8508e8b1 in sixel_decode_raw /home/company/real_sanitize/libsixel-master/src/fromsixel.c:881
    #5 0x7fcd850c042c in load_sixel /home/company/real_sanitize/libsixel-master/src/loader.c:613
    #6 0x7fcd850c042c in load_with_builtin /home/company/real_sanitize/libsixel-master/src/loader.c:782
    #7 0x7fcd850c43d9 in sixel_helper_load_image_file /home/company/real_sanitize/libsixel-master/src/loader.c:1352
    #8 0x7fcd850cf283 in sixel_encoder_encode /home/company/real_sanitize/libsixel-master/src/encoder.c:1737
    #9 0x4017f8 in main /home/company/real_sanitize/libsixel-master/converters/img2sixel.c:457
    #10 0x7fcd84a88a3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x20a3f)
    #11 0x401918 in _start (/home/company/real_sanitize/poc_check/libsixel/img2sixel+0x401918)

0x60200000a7b1 is located 0 bytes to the right of 1-byte region [0x60200000a7b0,0x60200000a7b1)
allocated by thread T0 here:
    #0 0x7fcd853b59aa in malloc (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x989aa)
    #1 0x7fcd8508be1f in image_buffer_resize /home/company/real_sanitize/libsixel-master/src/fromsixel.c:292
```

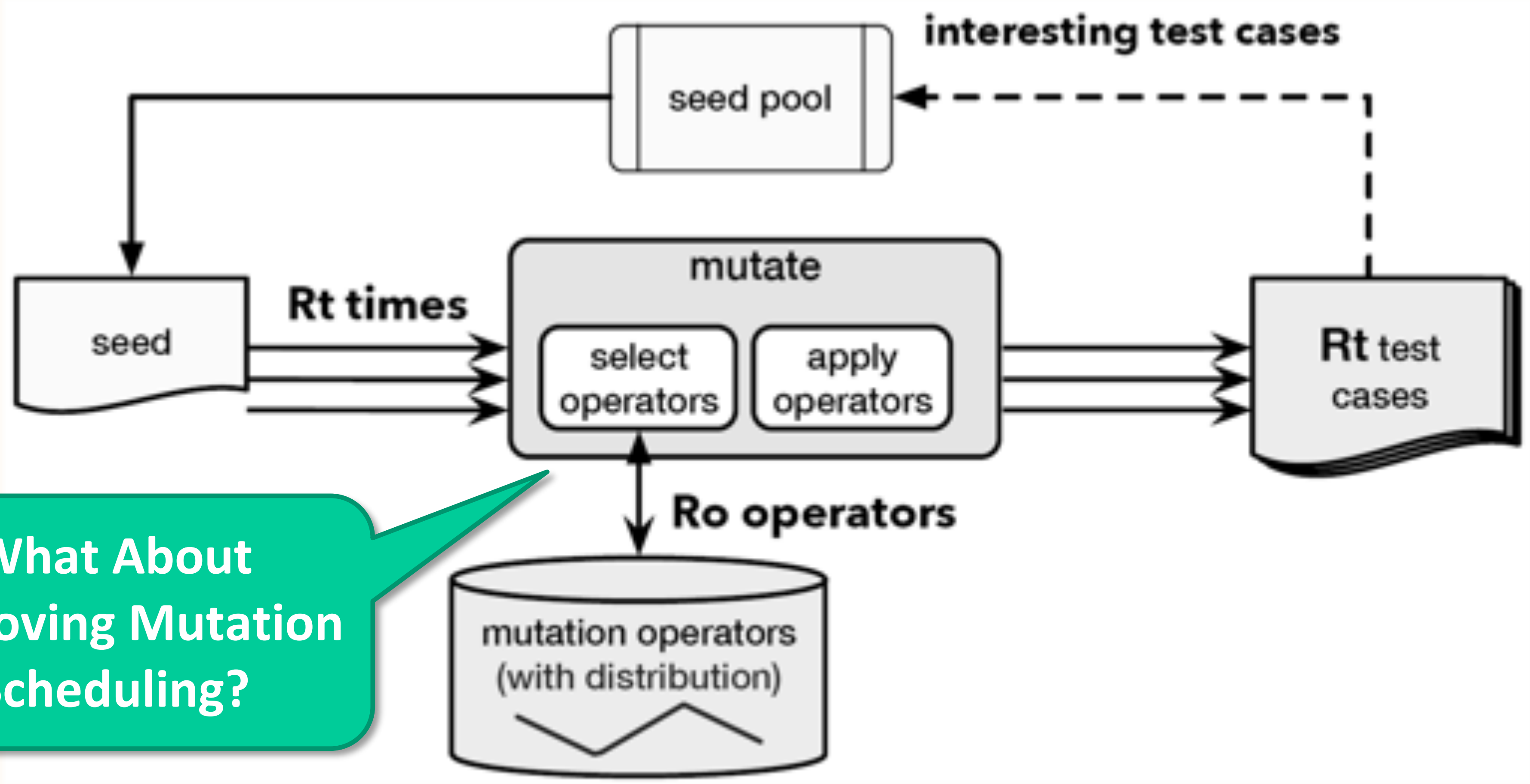
Improvement 3: Seed Mutation Scheduling

USENIX Security 2019

MOpt: Optimized Mutation Scheduling for Fuzzers

Chenyang Lyu[†], Shouling Ji[†,+,(✉)], Chao Zhang[¶,(✉)], Yuwei Li[†], Wei-Han Lee[§], Yu Song[†], and Raheem Beyah[‡]

How to improve (mutation-based) fuzzing?



What About Improving Mutation Scheduling?

Mutation operators of AFL

- Mutation operators characterize where and how to mutate the seed.

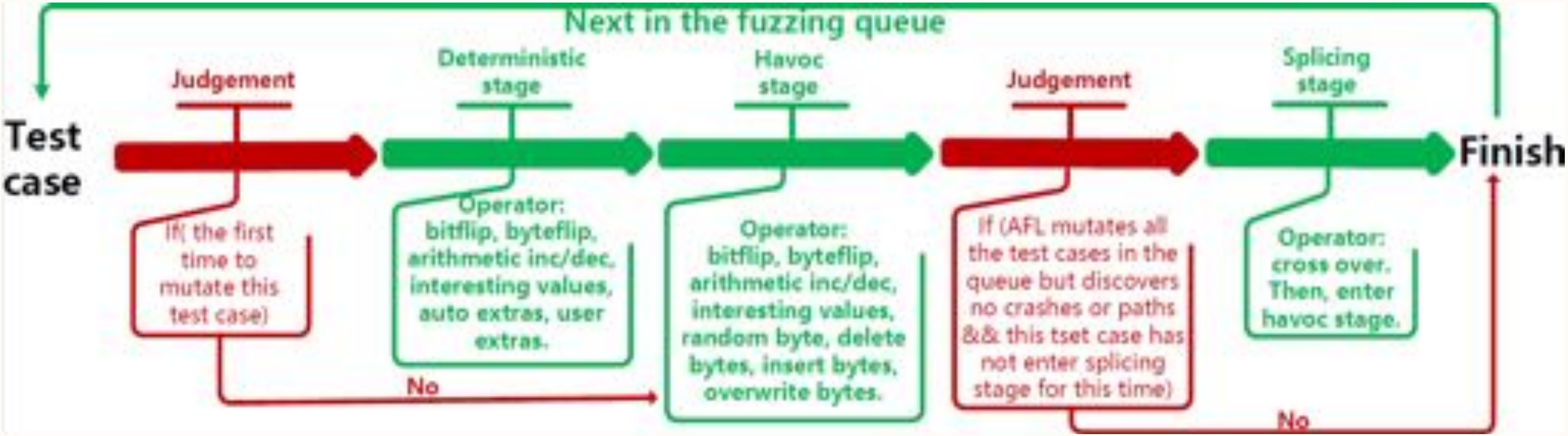
Type	Meaning	Operators
bitflip	Invert one or several consecutive bits in a test case, where the stepover is 1 bit.	bitflip 1/1, bitflip 2/1, bitflip 4/1
byteflip	Invert one or several consecutive bytes in a test case, where the stepover is 8 bits.	bitflip 8/8, bitflip 16/8, bitflip 32/8
arithmetic inc/dec	Perform addition and subtraction operations on one byte or several consecutive bytes.	arith 8/8, arith 16/8, arith 32/8

Some of the mutation operators in AFL.

The mutation operator *bitflip 2/1* represents flipping 2 consecutive bits, where the stepover is 1 bit

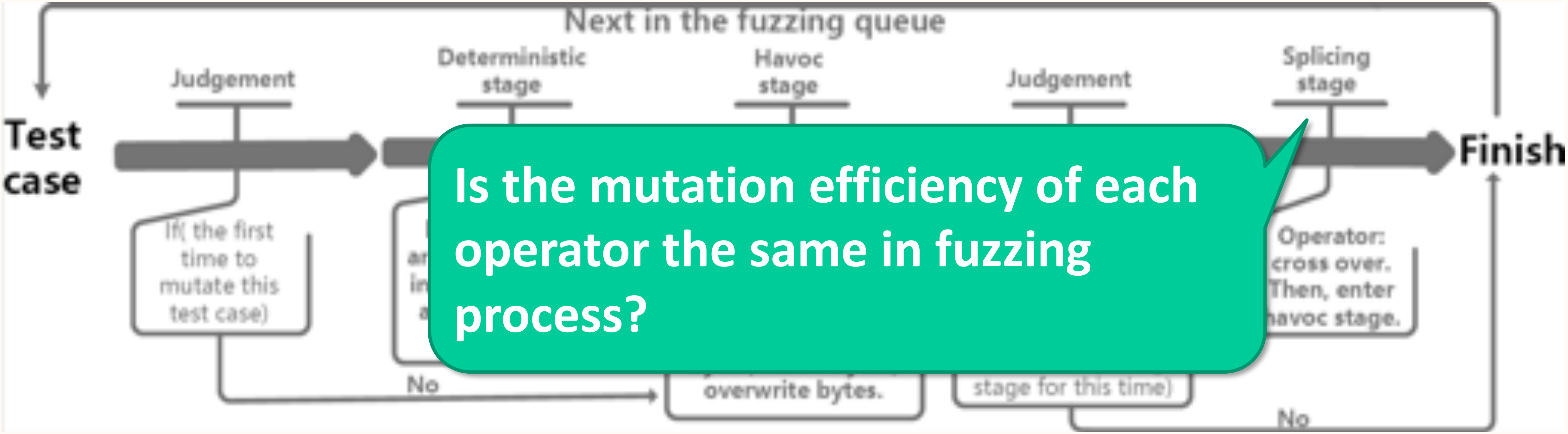
Mutation scheduling of AFL

- Three mutation stages:
  - Deterministic, havoc, and splicing

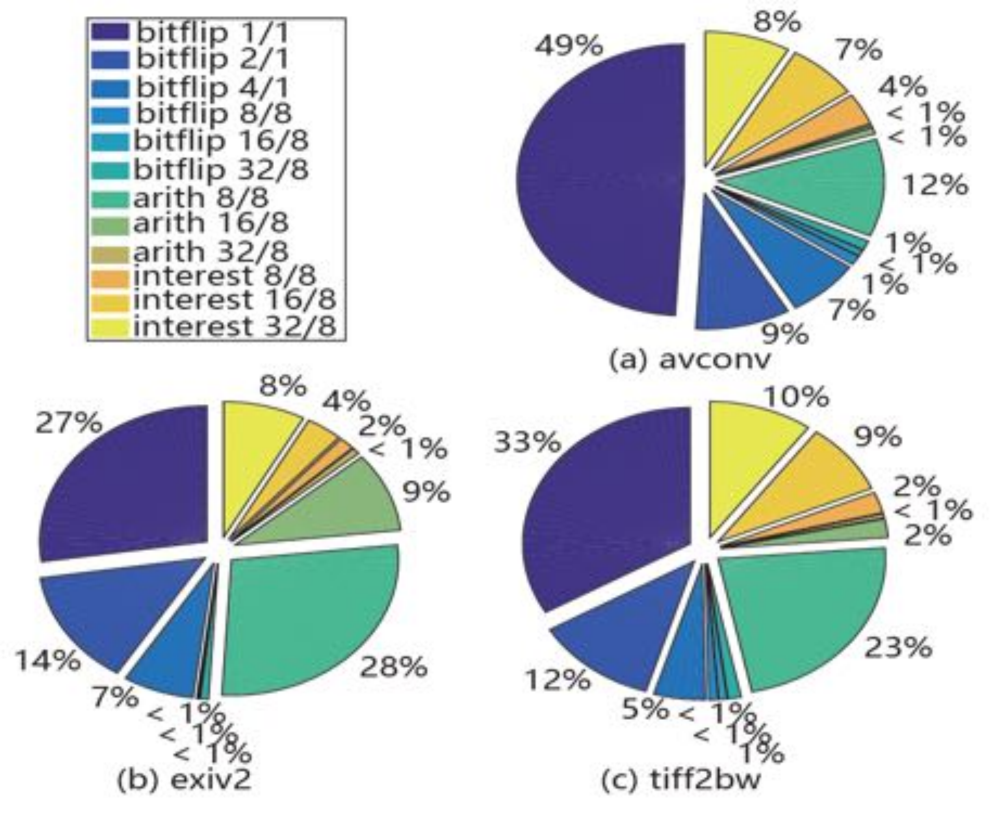


Mutation scheduling scheme of AFL

- Three mutation stages:
  - Deterministic, havoc, and splicing



Mutation efficiency study on AFL



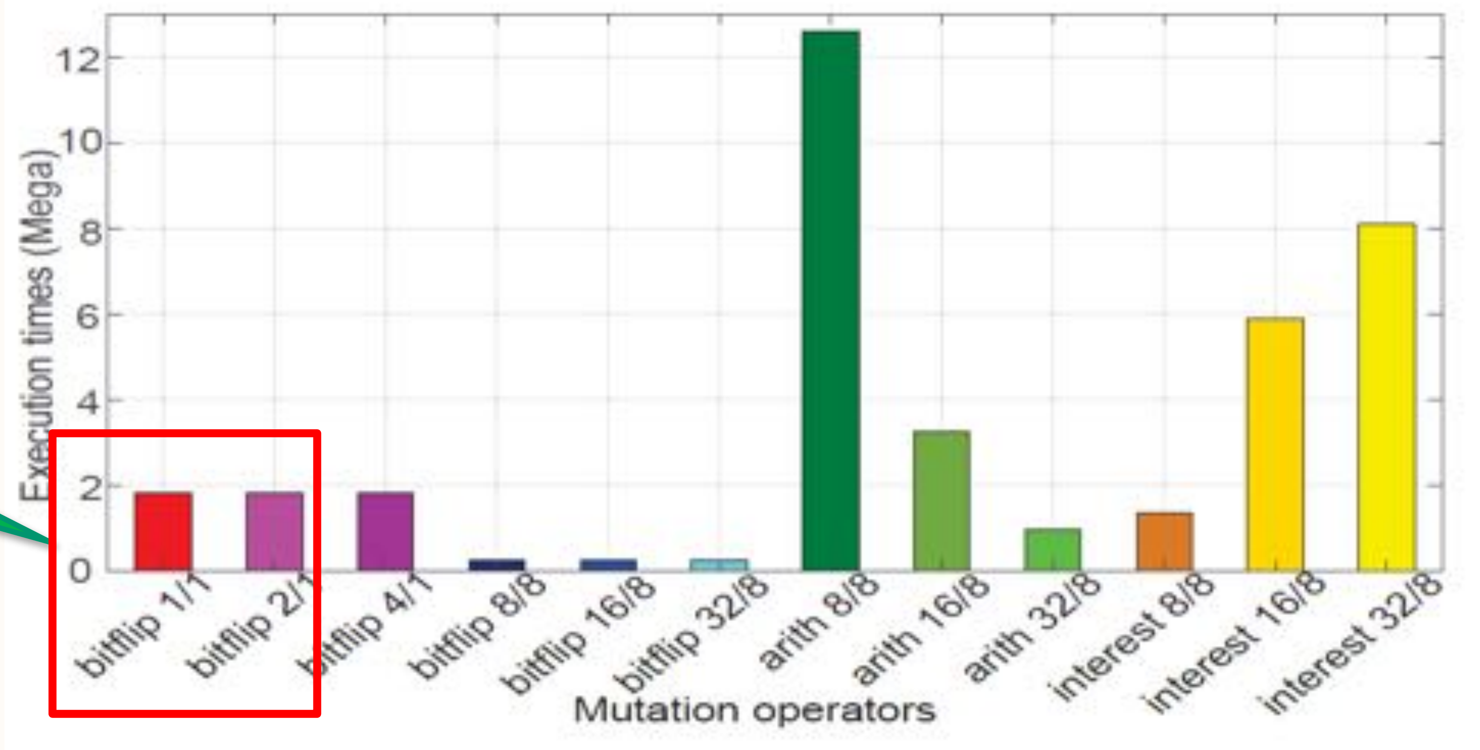
Different mutation operators' efficiencies are different.

For these programs, the mutation operators *bitflip 1/1, bitflip 2/1* and *arith 8/8* could yield more interesting test cases than other mutation operators.

Percentages of interesting test cases produced by different operators in the deterministic stage of AFL

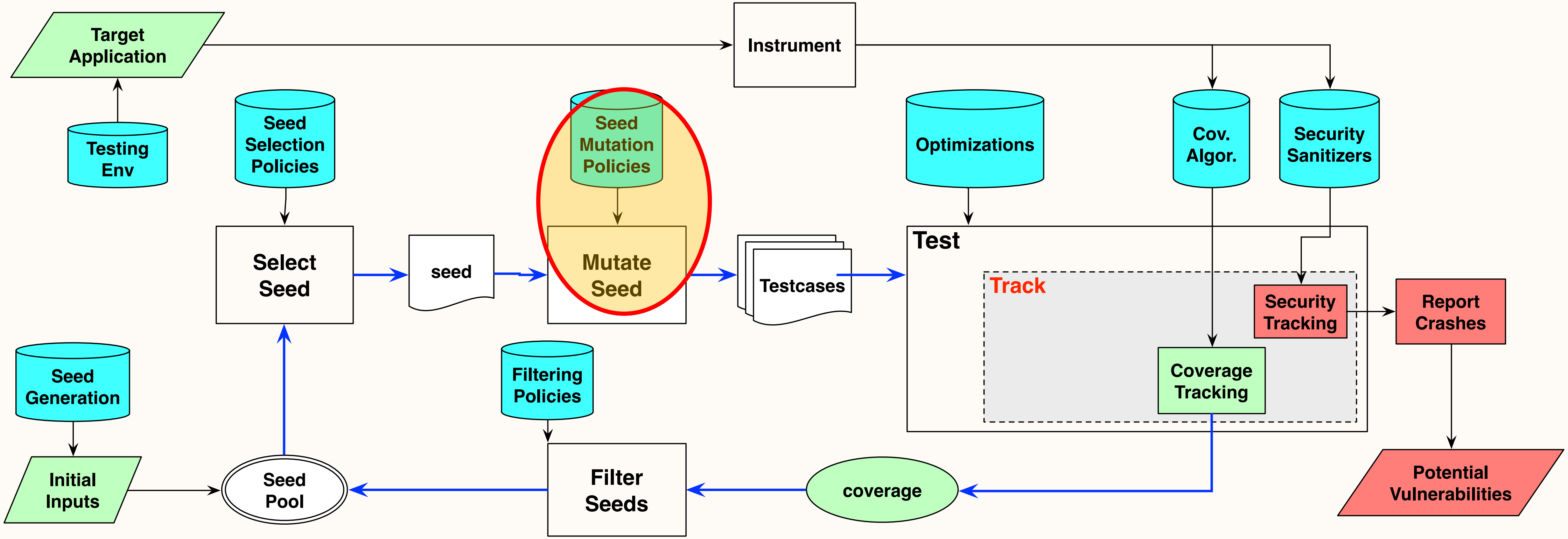
How does AFL select these mutation operators?

The two efficient operators are selected for a small number of times.



The times that mutation operators are selected when AFL fuzzes the target program avconv.

Our Solution: MOPT

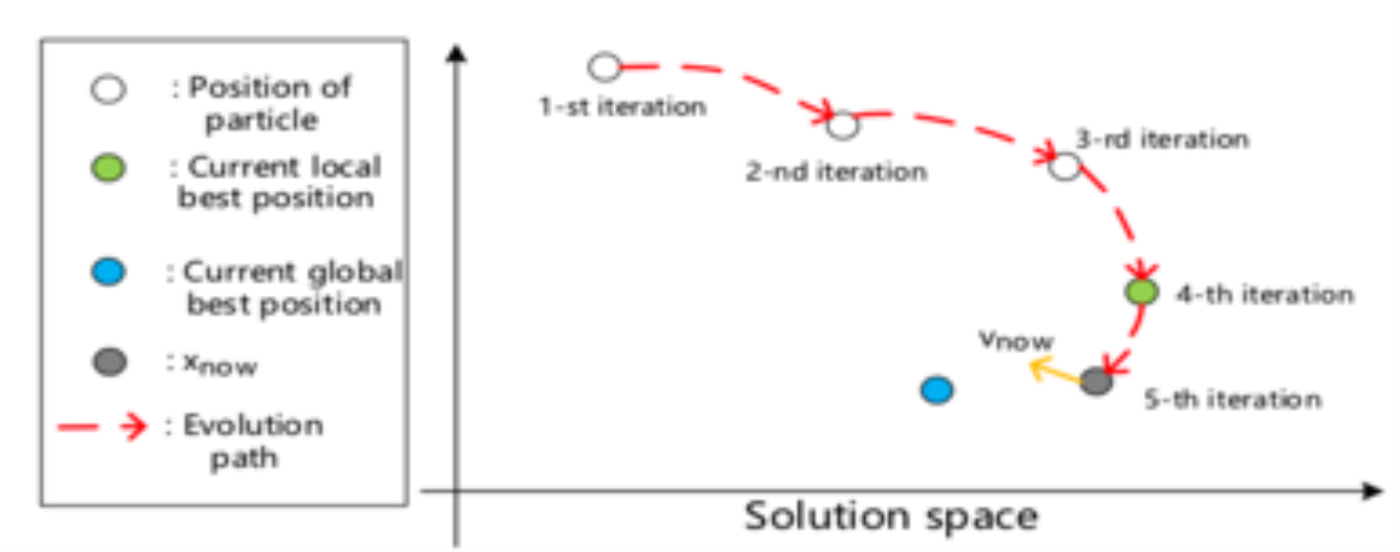


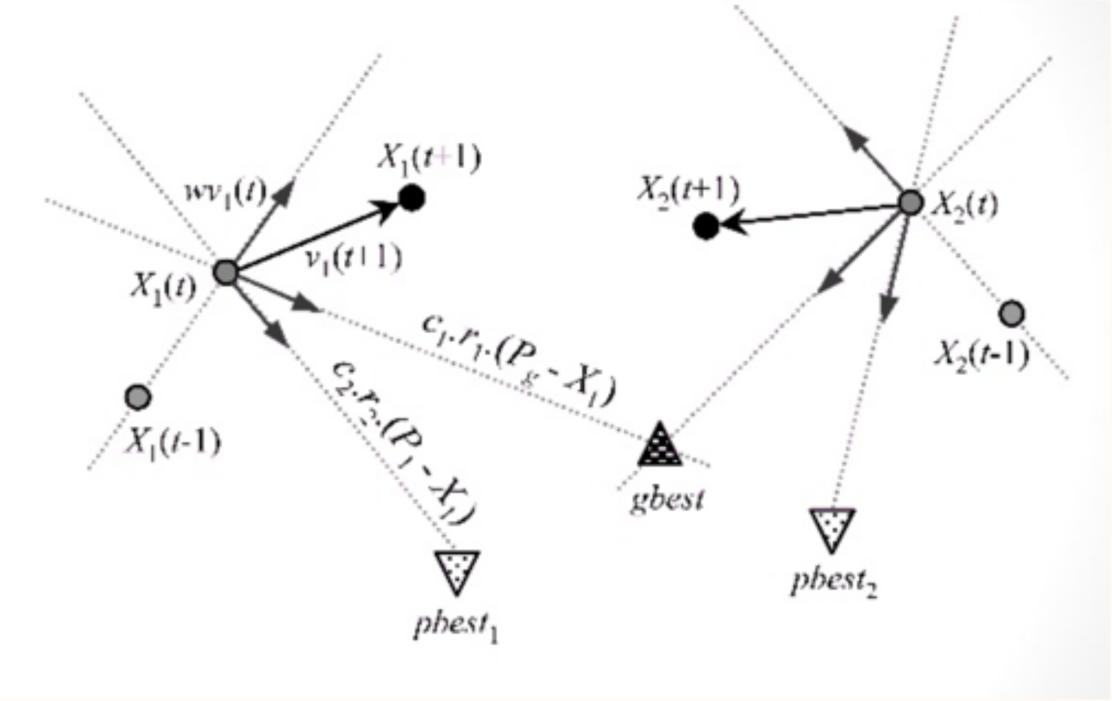
- Schedule seed mutation operators in a smarter way

Intuition

- Idea: select the "best" mutation operator based on
  - each operator's historic performance

- Solution: Particle Swarm Optimization





Particle Swarm Optimization

- For each iteration, the movement of a particle p is updated as follows:

$$V_{now}(\mathrm{p}) \leftarrow w \times V_{now}(\mathrm{p}) + r \times \left(L_{best}(p) - x_{now}(p)\right) + r \times \left(G_{best} - x_{now}(p)\right)$$

$$X_{now}(\mathrm{p}) \leftarrow X_{now}(\mathrm{p}) + V_{now}(\mathrm{p})$$

- $V_{now}(\mathrm{p})$ is the velocity of a particle p.
- $X_{now}(\mathrm{p})$ is the position of a particle p.
- $L_{best}(p)$ is the local best position of a particle p.
- G_{best} is the global best position.
- w is the inertia weight.
- $r \in (0,1)$ is a random displacement weight

The customized PSO algorithm of MOPT

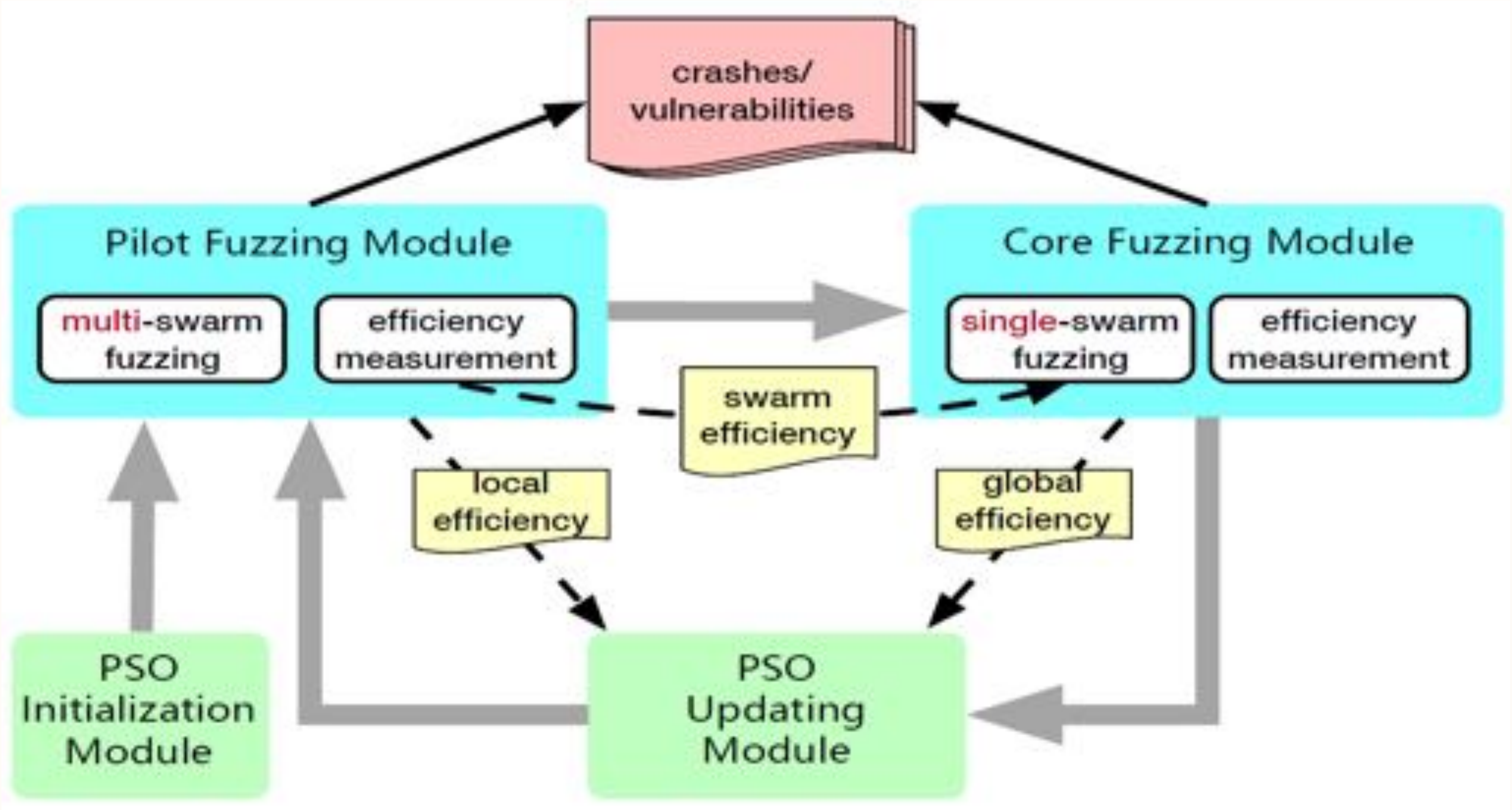
- For each iteration, the movement of a particle P_j (mutation operator) in a swarm S_i (a set of mutation operators), its position $X_{now}[S_i]\,[P_j]$ (the probability that it will be selected) is updated by these formula:

$$V_{now}[S_i][P_j] \leftarrow \quad w \times V_{now}[S_i][P_j] + r \times \left(L_{best}[S_i][P_j] - x_{now}[S_i][P_j]\right) + r \times \left(G_{best}[P_j] - x_{now}[S_i]\,[P_j]\right)$$

$$X_{now}[S_i]\,[P_j] \leftarrow X_{now}[S_i]\,[P_j] + V_{now}[S_i]\,[P_j]$$

- w is the inertia weight.
- $r \in (0,1)$ is a random displacement weigh

MOPT main framework



PSO Initialization Module

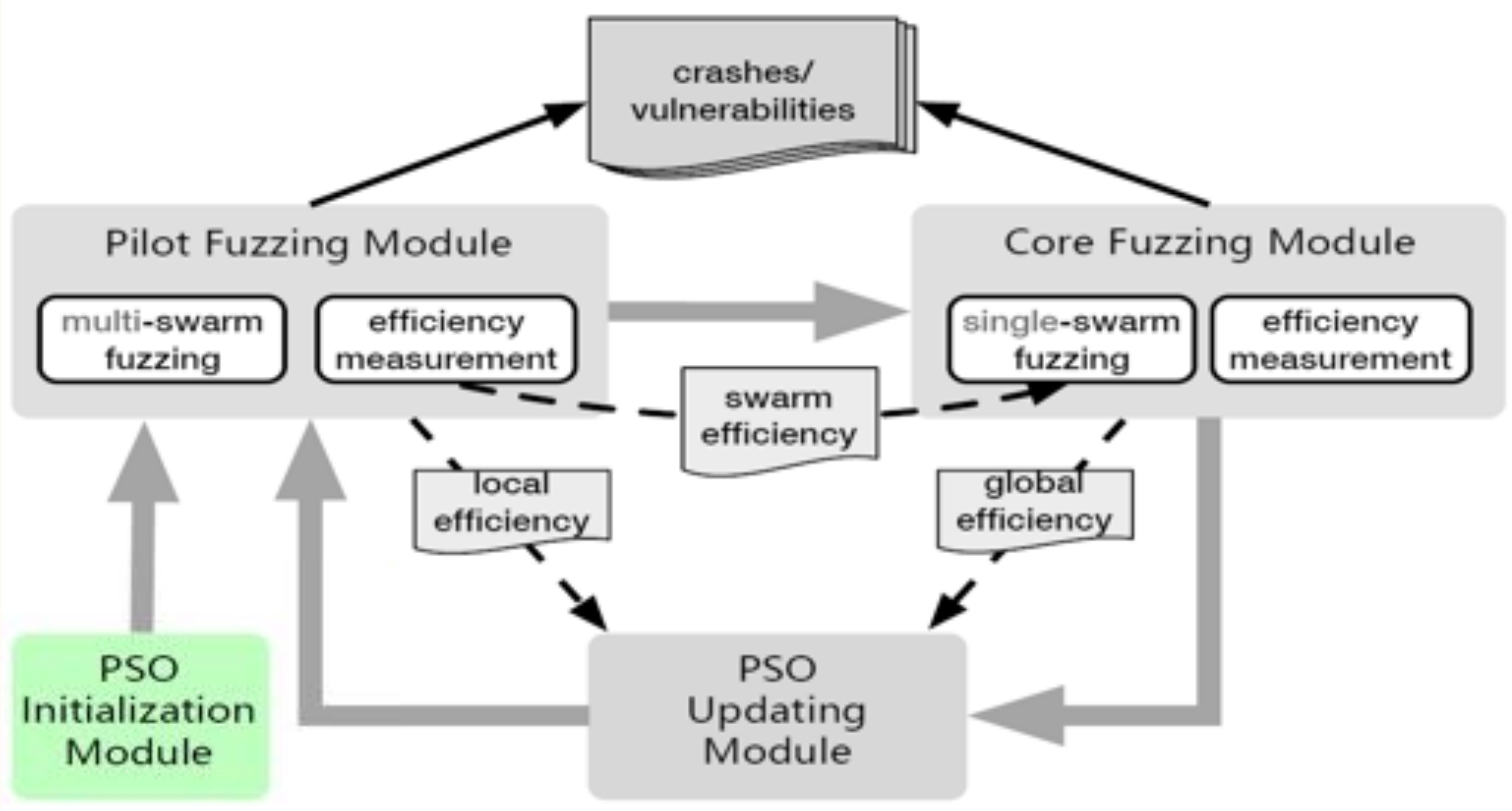
Pilot Fuzzing Module

Core Fuzzing Module

PSO Updating Module

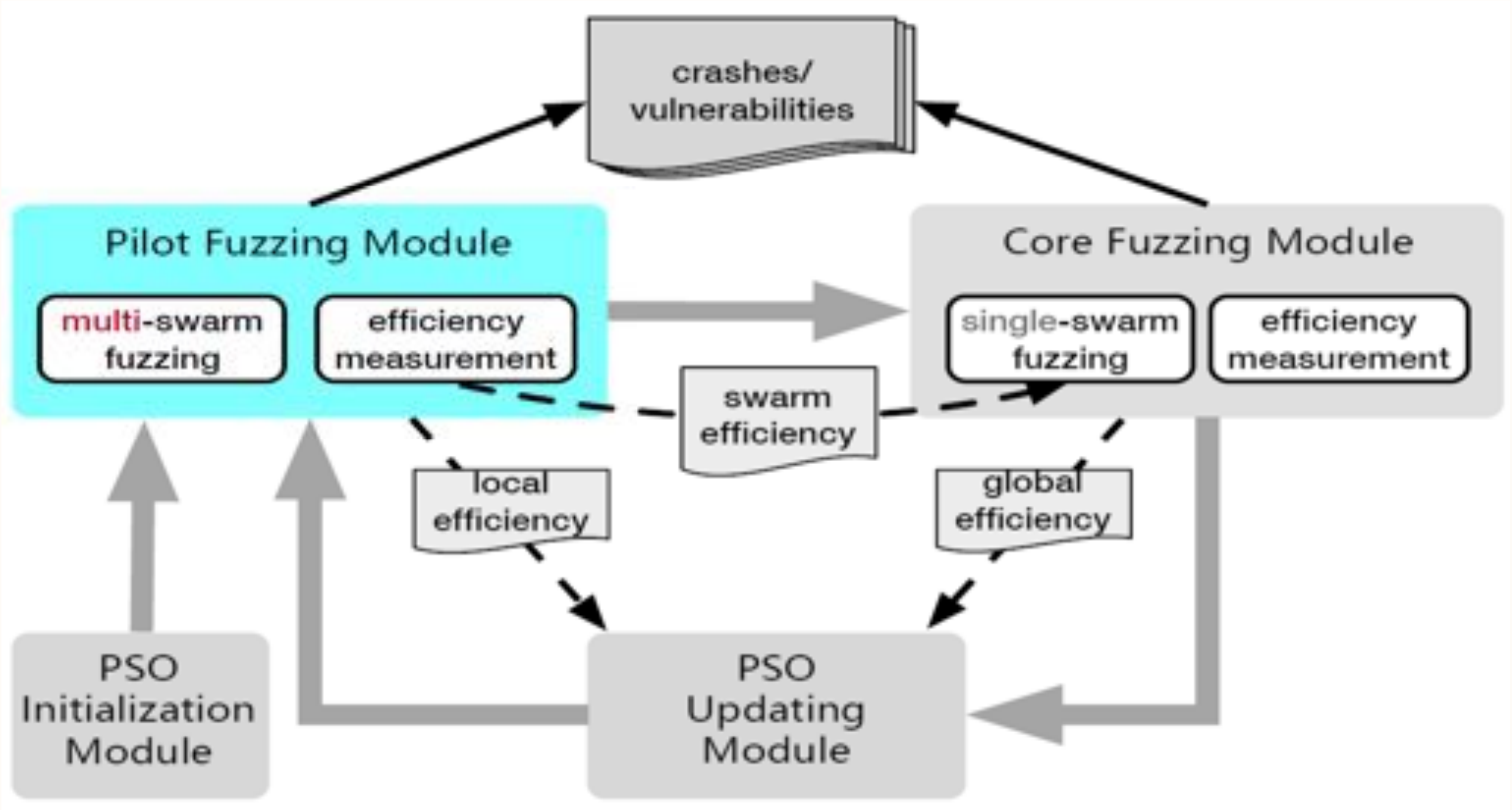
Source: https://github.com/vul337/MOpt-AFL

MOPT main framework



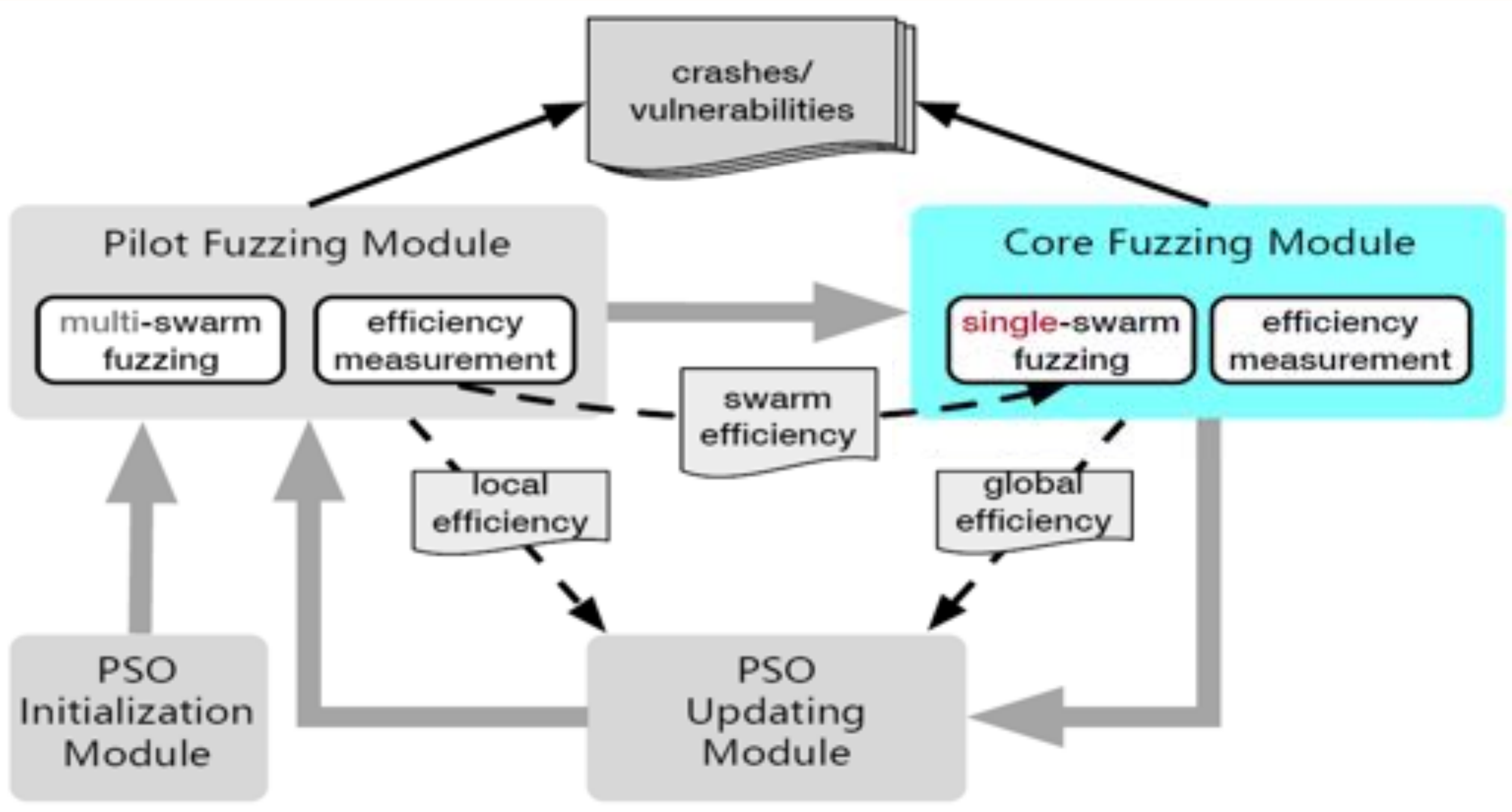
PSO Initialization Module initializes parameters for the customized PSO algorithm.

MOPT main framework



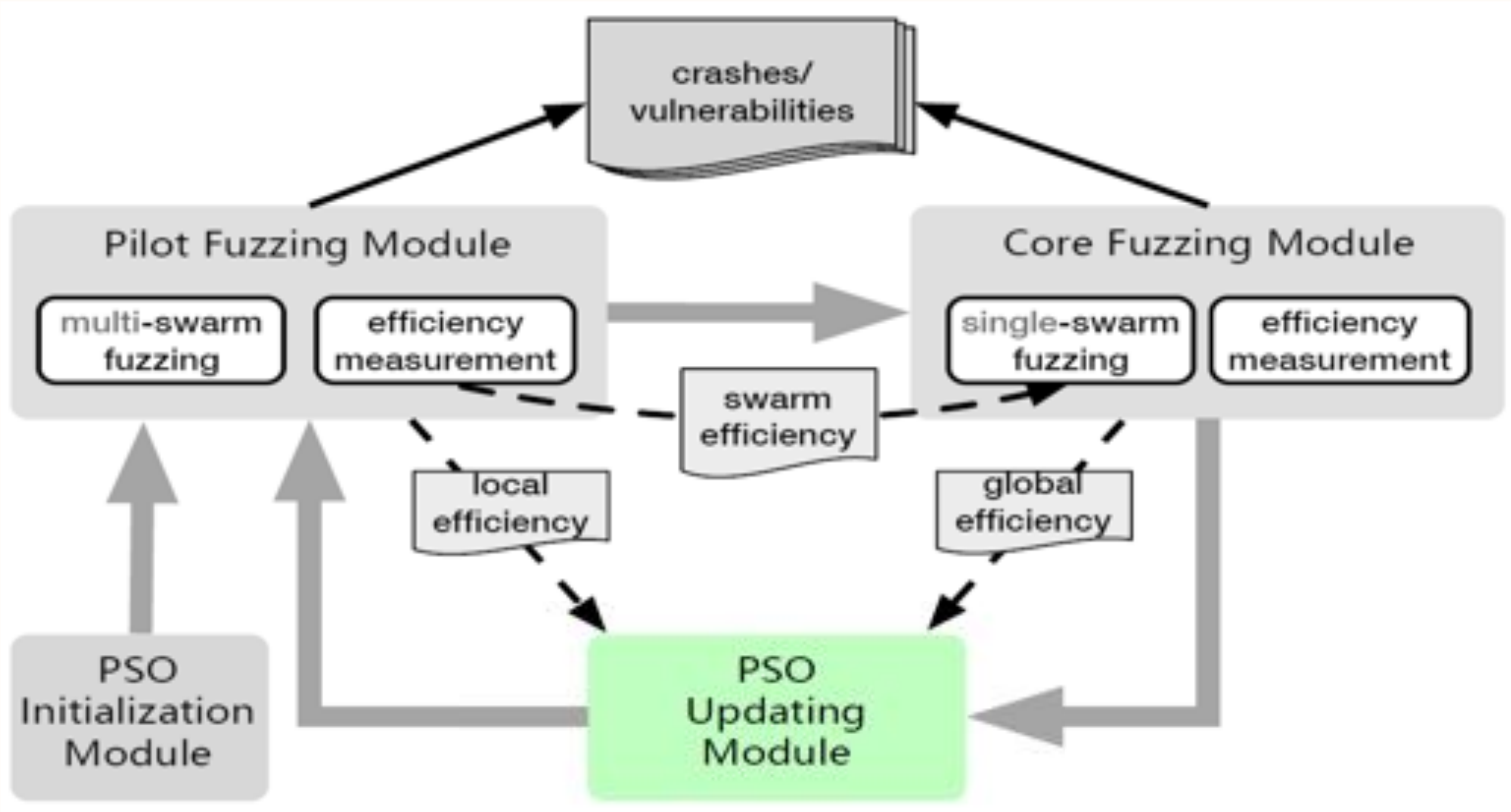
Pilot Fuzzing Module employs the distributions from multiple swarms to perform fuzzing and records the measurements for updating.

MOPT main framework



Core Fuzzing Module employs the best swarm evaluated by *Pilot Fuzzing Module* to perform fuzzing and records the measurements.

MOPT main framework



PSO Updating Module updates the distribution of each swarm with the measurements from Pilot Fuzzing and Core Fuzzing Modules.

Evaluation: unique crashes and paths

Program	AFL		MOpt-AFL-tmp				MOpt-AFL-ever			
	Unique crashes	Unique paths	Unique crashes	Increase	Unique paths	Increase	Unique crashes	Increase	Unique paths	Increase
mp42aac	135	815	**209**	+54.8%	1,660	+103.7%	199	+47.4%	**1,730**	+112.3%
exiv2	34	2,195	54	+58.8%	2,980	+35.8%	**66**	+94.1%	**4,642**	+111.5%
mp3gain	178	1,430	**262**	+47.2%	**2,211**	+54.6%	**262**	+47.2%	2,206	+54.3%
tiff2bw	4	4,738	**85**	+2,025.0%	**7,354**	+55.2%	43	+975.0%	7,295	+54.0%
pdfimages	23	12,915	357	+1,452.2%	22,661	+75.5%	**471**	+1,947.8%	**26,669**	+106.5%
sam2p	36	531	105	+191.7%	1,967	+270.4%	**329**	+813.9%	**3,418**	+543.7%
avconv	0	2,478	**4**	+4	**17,359**	+600.5%	1	+1	16,812	+578.5%
w3m	0	3,243	**506**	+506	5,313	+63.8%	182	+182	**5,326**	+64.2%
objdump	0	11,565	**470**	+470	19,309	+67.0%	287	+287	**22,648**	+95.8%
jhead	19	478	55	+189.5%	**489**	+2.3%	**69**	+263.2%	483	+1.0%
mpg321	10	123	**236**	+2,260.0%	1,054	+756.9%	229	+2,190.0%	**1,162**	+844.7%
infotocap	92	3,710	340	+269.6%	6,157	+66.0%	**692**	+652.2%	**7,048**	+90.0%
podofopdfinfo	79	3,397	**122**	+54.4%	**4,704**	+38.5%	114	+44.3%	4,694	+38.2%
total	610	47,618	2,805	+359.8%	93,218	+95.8%	**2,944**	+382.6%	**104,133**	+118.7%

Both MOpt-AFL-tmp and –ever found more unique crashes and paths than AFL.

Evaluation: Vulnerability discovery

Program	AFL				MOPT-AFL-tmp				MOPT-AFL-ever			
	Unknown vulnerabilities		Known vulnerabilities	Sum	Unknown vulnerabilities		Known vulnerabilities	Sum	Unknown vulnerabilities		Known vulnerabilities	Sum
	Not CVE	CVE	CVE		Not CVE	CVE	CVE		Not CVE	CVE	CVE	
mp42aac	/	1	1	2	/	2	1	3	/	5	1	6
exiv2	/	5	3	8	/	5	4	9	/	4	4	8
mp3gain	/	4	2	6	/	9	3	12	/	5	2	7
pdfimages	/	1	0	1	/	12	3	15	/	9	2	11
avconv	/	0	0	0	/	2	0	2	/	1	0	1
w3m	/	0	0	0	/	14	0	14	/	5	0	5
objdump	/	0	0	0	/	1	2	3	/	0	2	2
jhead	/	1	0	1	/	4	0	4	/	5	0	5
mpg321	/	0	1	[illegible]	/	0	1	[illegible]	/	0	1	[illegible]
infotocap	/	3	0	[illegible]	/	3	0	[illegible]	/	3	0	[illegible]
podofopdfinfo	/	5	0	[illegible]	/	6	0	[illegible]	/	6	0	[illegible]
tiff2bw	1	/	/	[illegible]	2	/	/	[illegible]	2	/	/	[illegible]
sam2p	5	/	/	5	14	/	/	14	28	/	/	28
Total	6	20	7	33	16	58	14	88	30	43	12	85

33

88

85

Vulnerabilities discovered by AFL, MOPT-AFL-tmp, MOPT-AFL-ever

Both MOPT-AFL-tmp and –ever found much more vulnerabilities than AFL.

CVE discovery

Target	Types	AFL	MOpt-AFL-tmp	MOpt-AFL-ever	Severity
mp42aac	buffer overflow	CVE-2018-10785	CVE-2018-10785; CVE-2018-18037	CVE-2018-10785; CVE-2018-18037; CVE-2018-17814	4.3
	memory leaks	CVE-2018-17813	CVE-2018-17813	CVE-2018-17813; CVE-2018-18050; CVE-2018-18051	4.3
exiv2	heap overflow	CVE-2017-11339; CVE-2017-17723; CVE-2018-18036	CVE-2017-11339; CVE-2017-17723; CVE-2018-10780	CVE-2017-11339; CVE-2017-17723; CVE-2018-18036	5.8
	stack overflow	CVE-2017-14861	CVE-2017-14861	CVE-2017-14861	4.3
	buffer overflow	CVE-2018-18047	CVE-2018-17808; CVE-2018-18047	CVE-2018-18047	4.3
	segmentation violation	CVE-2018-18046	CVE-2018-18046	CVE-2018-18046	4.3
	memory access violation	CVE-2018-17809; CVE-2018-17807	CVE-2018-17809; CVE-2018-17823	CVE-2017-11337; CVE-2018-17809	4.3
mp3gain	stack buffer overflow	CVE-2017-14407	CVE-2017-14407; CVE-2018-17801; CVE-2018-17799	CVE-2017-14407	4.3
	global buffer overflow	CVE-2018-17800; CVE-2018-17802; CVE-2018-18045; CVE-2018-18043	CVE-2017-14409; CVE-2018-17800; CVE-2018-17803; CVE-2018-17802; CVE-2018-18045; CVE-2018-18043; CVE-2018-18044	CVE-2018-17800; CVE-2018-17803; CVE-2018-17802; CVE-2018-18045; CVE-2018-18043	6.8
	segmentation violation	CVE-2017-14406	CVE-2017-14412	CVE-2017-14412	6.8
	memcpy param overlap		CVE-2018-17824		5.8
pdfimages	heap buffer overflow		CVE-2018-8105; CVE-2018-18054		4.3
	stack overflow	CVE-2018-17114	CVE-2018-16369; CVE-2018-17114; CVE-2018-17115; CVE-2018-17116; CVE-2018-17117; CVE-2018-17119; CVE-2018-17120; CVE-2018-17121; CVE-2018-17122; CVE-2018-18053; CVE-2018-18055	CVE-2018-16369; CVE-2018-17115; CVE-2018-17116; CVE-2018-17119; CVE-2018-17121; CVE-2018-17122; CVE-2018-18053	6.1
	global buffer overflow		CVE-2018-8102	CVE-2018-8102	4.3
	alloc dealloc mismatch		CVE-2018-17118	CVE-2018-17118	4.3
	segmentation violation			CVE-2018-17123; CVE-2018-17124	4.3
avconv	segmentation violation		CVE-2018-17804	CVE-2018-17804	4.3
	memory leaks		CVE-2018-17805		4.3
w3m	segmentation violation		CVE-2018-17815; CVE-2018-17816; CVE-2018-17817; CVE-2018-17818; CVE-2018-17819; CVE-2018-17821; CVE-2018-17822; CVE-2018-18038; CVE-2018-18039; CVE-2018-18040; CVE-2018-18041; CVE-2018-18042; CVE-2018-18052	CVE-2018-17816; CVE-2018-18040; CVE-2018-18041; CVE-2018-18042	5.3
	memory leaks		CVE-2018-17820	CVE-2018-17820	4.3
objdump	stack exhaustion		CVE-2018-12700	CVE-2018-12641	5.0
	stack overflow		CVE-2018-9138; CVE-2018-16617	CVE-2018-9138	4.3
jhead	heap buffer overflow	CVE-2018-17810	CVE-2018-17810; CVE-2018-17811; CVE-2018-18048; CVE-2018-18049	CVE-2018-17810; CVE-2018-17811; CVE-2018-17812; CVE-2018-18048; CVE-2018-18049	4.3
mpg321	heap buffer overflow	CVE-2017-12063	CVE-2017-12063	CVE-2017-12063	4.3
infotocap	memory leaks	CVE-2018-16614	CVE-2018-16614	CVE-2018-16614	4.3
	segmentation violation	CVE-2018-16615; CVE-2018-16616	CVE-2018-16615; CVE-2018-16616	CVE-2018-16615; CVE-2018-16616	4.3
podofopdfinfo	stack overflow	CVE-2018-18216; CVE-2018-18221; CVE-2018-18222	CVE-2018-18216; CVE-2018-18217; CVE-2018-18221; CVE-2018-18222	CVE-2018-18216; CVE-2018-18217; CVE-2018-18218; CVE-2018-18221	4.7
	heap buffer overflow	CVE-2018-18219	CVE-2018-18219	CVE-2018-18219	4.3
	segmentation violation	CVE-2018-18220	CVE-2018-18220	CVE-2018-18220	4.3

Both MOpt-AFL-tmp and –ever found more CVEs with a variety of types than AFL.

Improvement 4: Seed Generation

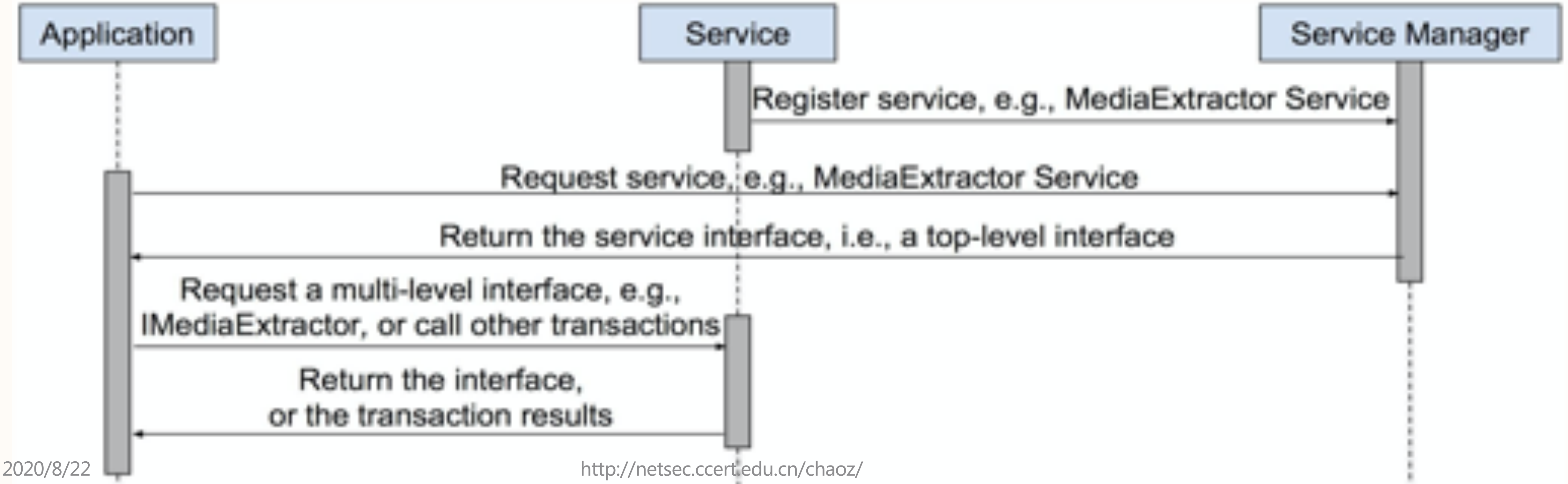
USENIX Security 2020

FANS: Fuzzing Android Native System Services via Automated Interface Analysis

Baozheng Liu[1,2], Chao Zhang[1,2], Guang Gong[3],
Yishun Zeng[1,2], Haifeng Ruan[4], Jianwei Zhuge[1,2]

Android Application-Service Communication

- Android native system services provide fundamental functionalities, thus attractive to attackers
- A specific binder IPC mechanism is implemented to support native services
- Locate service interface (IBinder obj), launch transactions (transact method) with serialized data



Fuzzing Android Native Services

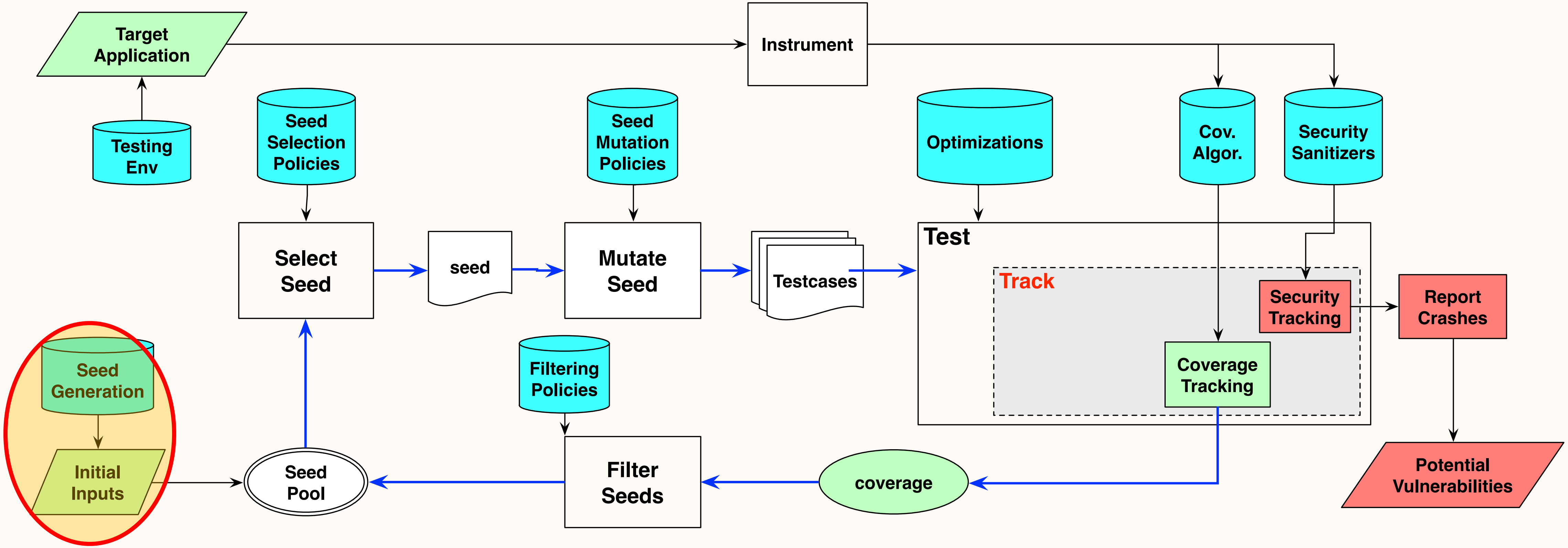
- Locate service interface (IBinder proxy obj)
  - some interfaces are deeply nested (not registered in Service Manager)
- launch transactions (transact method), with
  - many transactions are available, and
  - some are inter-dependent
- serialized data
  - data type
  - data dependency
- Simple random fuzzing is inefficient.

Client:
IBinder::transact(code,data,reply,flags)

IPC

Service:
Binder::onTransact(code, data, reply, flags)

Our Solution: FANS

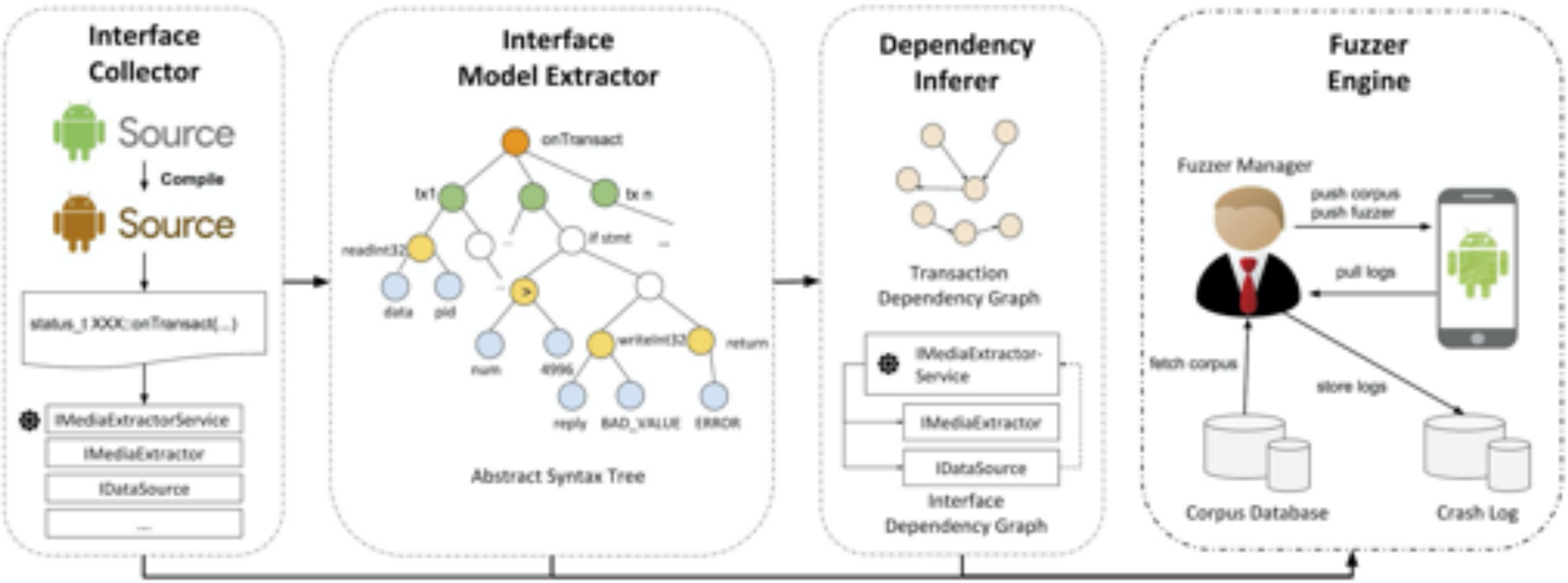


- Recognize testcase format
- Generate valid testcases

Challenges

- **C1. Multi-Level Interface Recognition**
  - Collect all Interfaces
  - Identify multi-level interfaces
- **C2. Interface Model Extraction**
  - Collect all of the possible transactions
  - Extract the input and output variables in the transactions
- **C3. Semantically-correct Input Generation**
  - Variable name and variable type
  - Variable dependency
  - Interface dependency

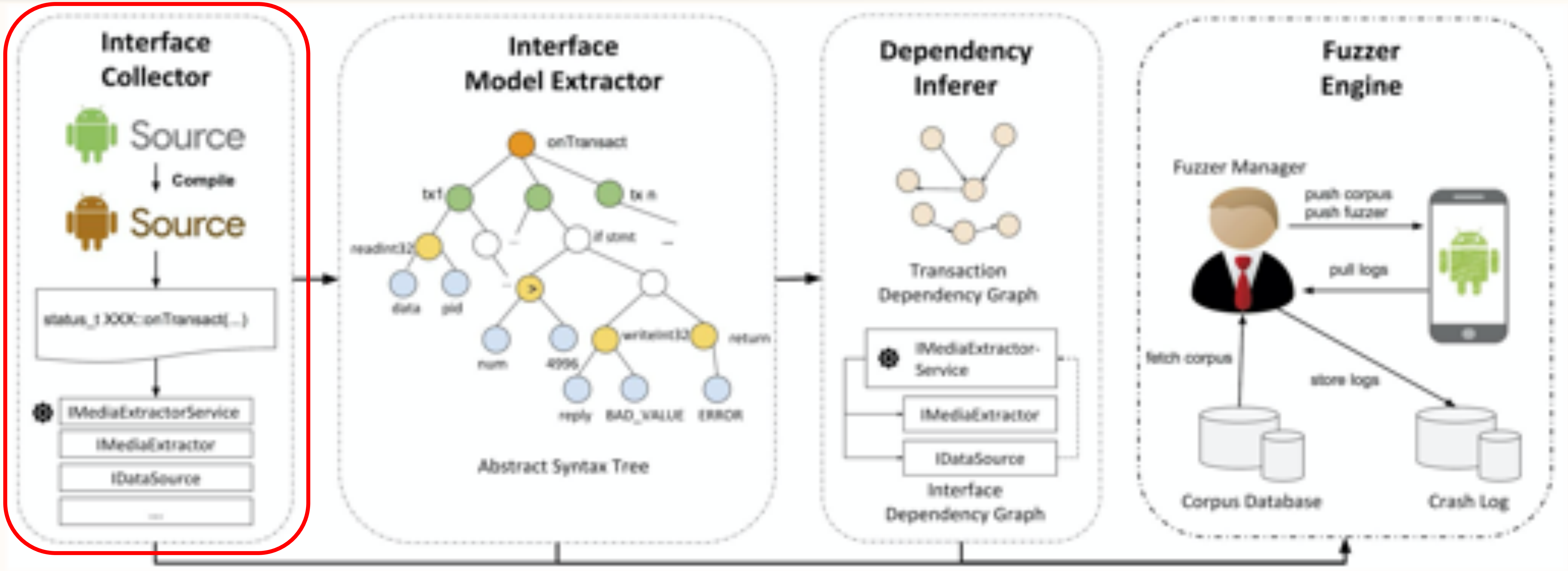
Overview



Interface Collector
Source
Compile
Source
status_t XXX::onTransact(...)
IMediaExtractorService
IMediaExtractor
IDataSource
...
Interface Model Extractor
onTransact
tx1
tx n
readInt32
if stmt
>
data
pid
num
4996
writeInt32
return
reply
BAD_VALUE
ERROR
Abstract Syntax Tree
Dependency Inferer
Transaction Dependency Graph
IMediaExtractor-Service
IMediaExtractor
IDataSource
Interface Dependency Graph
Fuzzer Engine
Fuzzer Manager
push corpus
push fuzzer
pull logs
fetch corpus
store logs
Corpus Database
Crash Log

Interface Collector

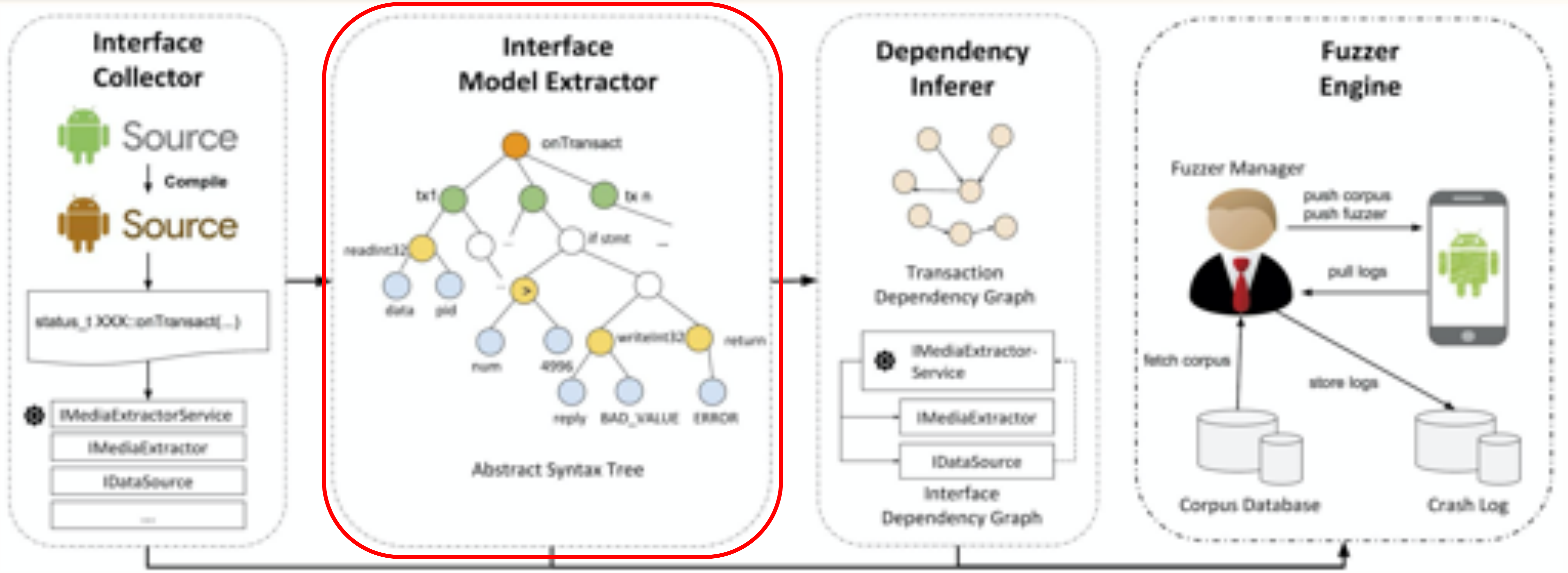
Binder::onTransact(code, data, reply, flags)



- Compile source code (including AIDL files)
- Recognize candidate service interfaces (with onTransact dispatcher)

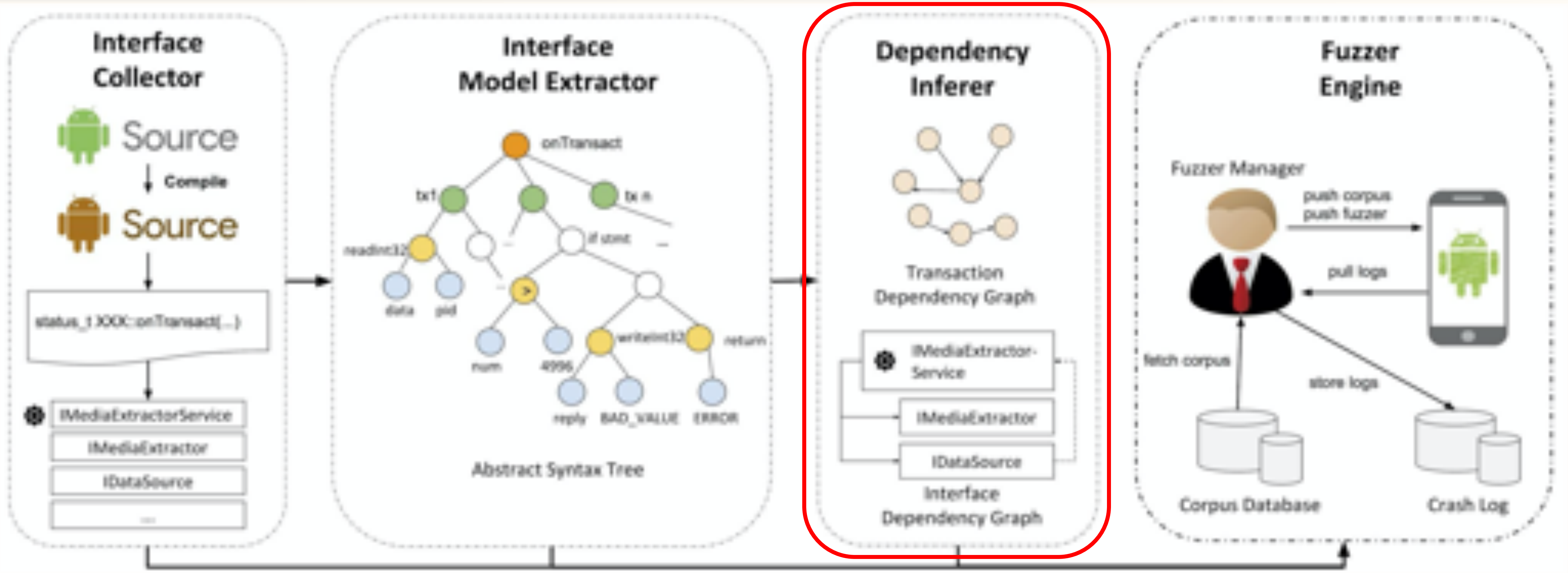
Interface Model Extractor

Binder::onTransact(code, data, reply, flags)



- Transactions supported by the interface: switch conditions in onTransact
- I/O variables (data) used in the interface: readInt32, writeInt32 (name, type, size)
- Other information: aggerated type definition (e.g., structure)

Dependency Analysis



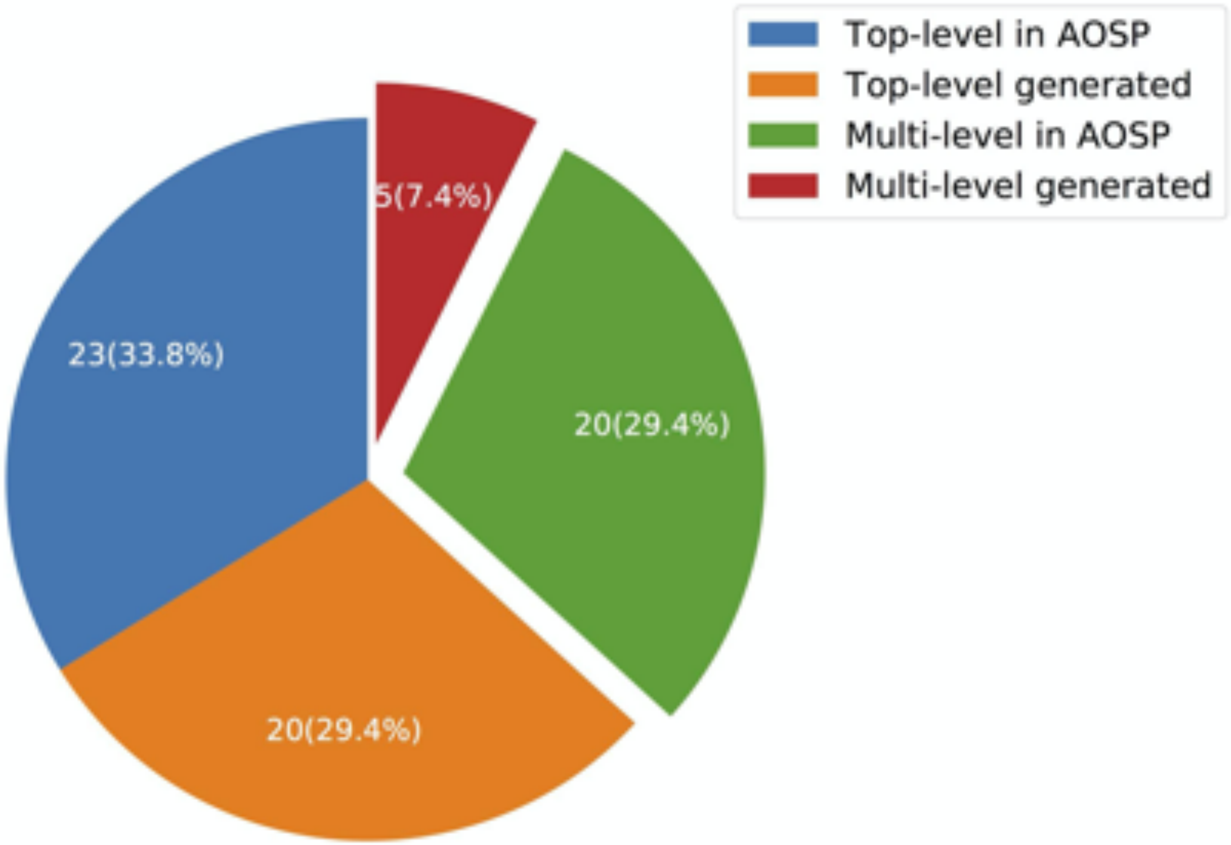
- Interface dependency: writeStrongBinder and readStrongBinder
- intra-transaction value dependency (conditional statement)
- inter-transaction value dependency (input/output variables with matching type and name)

Fuzzer


Interface Collector
Source
Compile
Source
status_t XXX::onTransact(...)
IMediaExtractorService
IMediaExtractor
IDataSource
...
Graph
Fuzzer Engine
Fuzzer Manager
push corpus
push fuzzer
pull logs
fetch corpus
store logs
Corpus Database
Crash Log

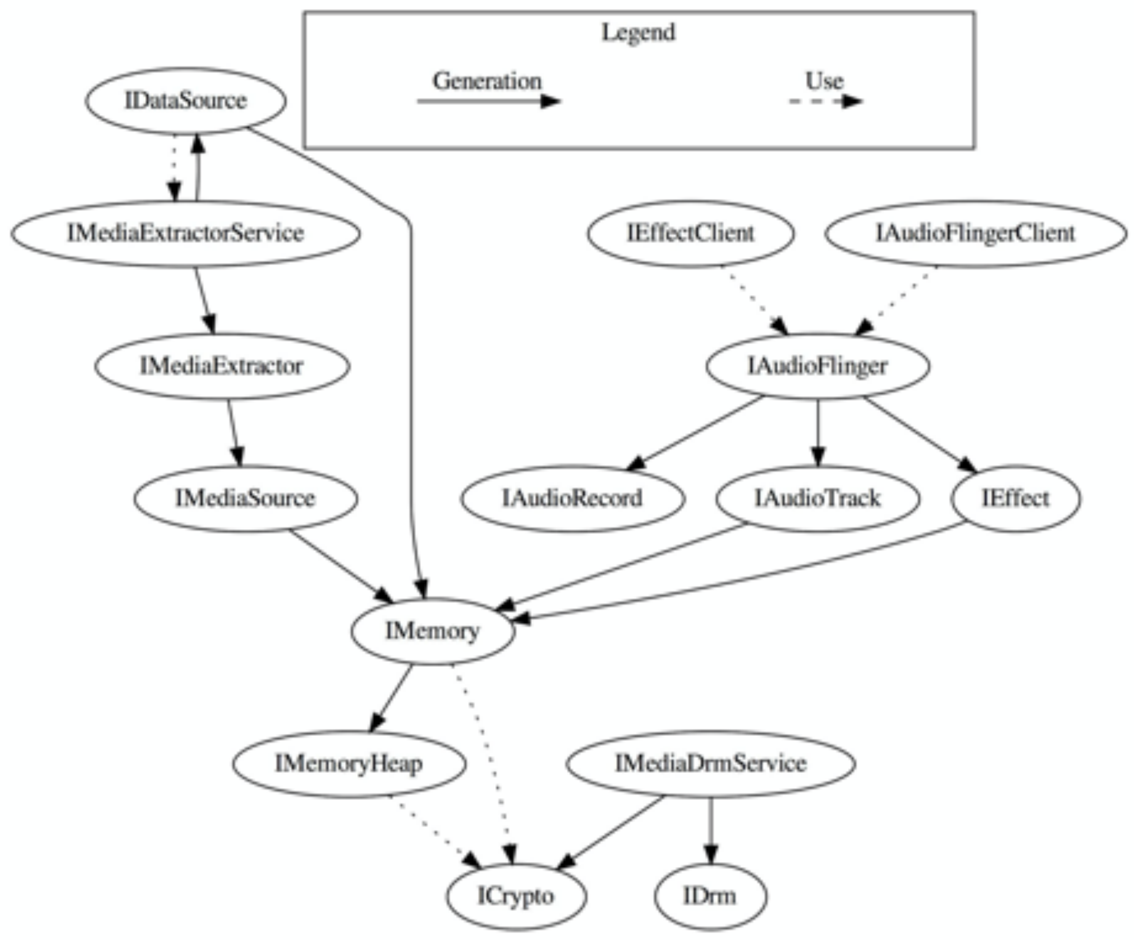
Q1 - Interface Statistics

- 43 top-level interfaces
- 25 multi-level interfaces
- **Most interfaces are written manually**



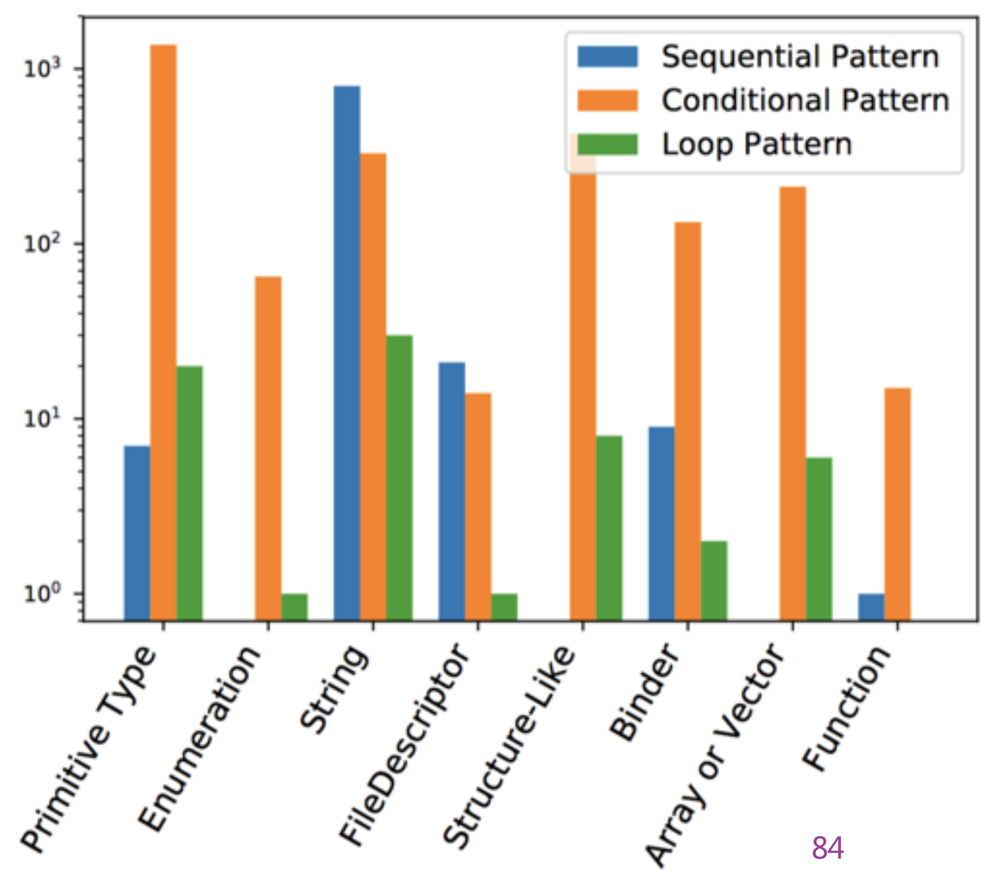
Q1 - Interface Dependency

- Interface generation
  - e.g., IMemory
- Deepest interface
  - IMemoryHeap (five-level)
- Customized interface
  - e.g., IEffectClient



Q2 - Extracted Interface Model Statistics

- Transaction
  - 530 transactions in top-level interfaces
  - 281 transactions in multi-level interfaces
- Variable
  - **Most variables are under constraint(s)**

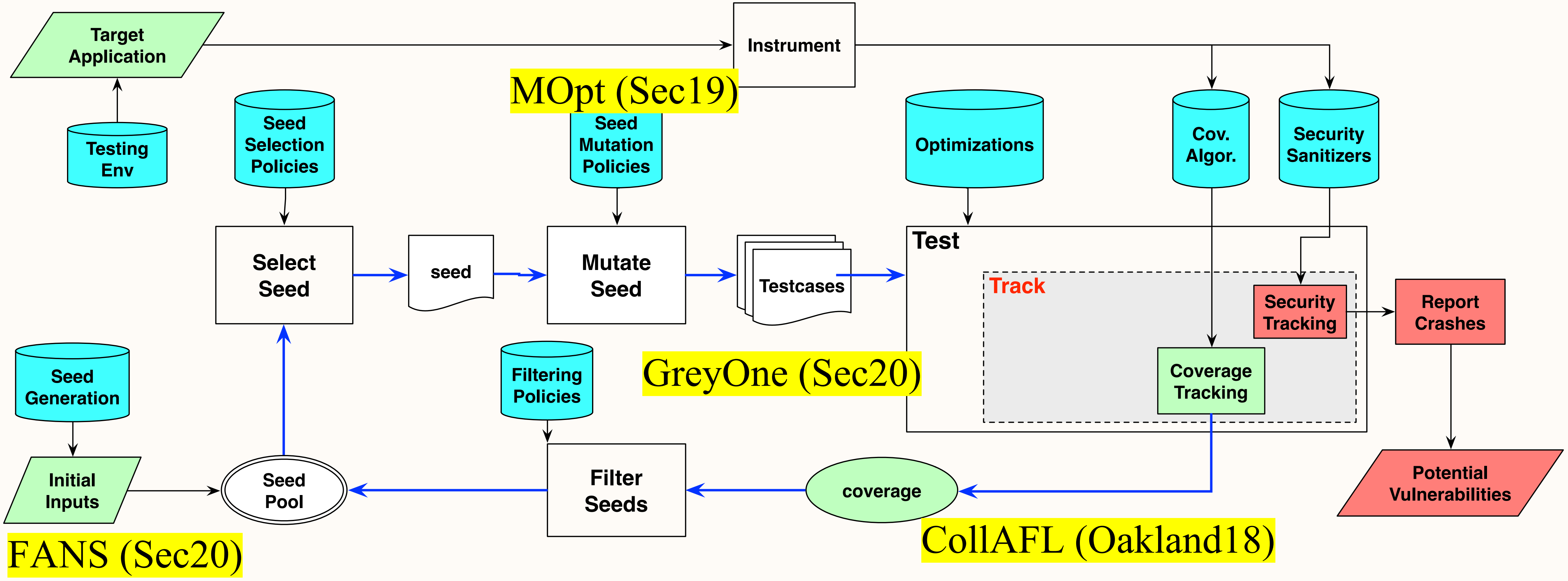


Q3 - Vulnerability Discovery

- We intermittently ran FANS for around 30 days
- FANS triggered thousands of crashes
  - **30 vulnerabilities in native programs**
    - **Google has confirmed 20 vulnerabilities**
  - **138 Java exceptions**
- Comparison with BinderCracker
  - BinderCracker found 89 vulnerabilities on Android 5.1 and Android 6.0
  - FANS discovered 168 vulnerabilities on android-9.0.0_r46

Source: https://github.com/vul337/fans

Recap

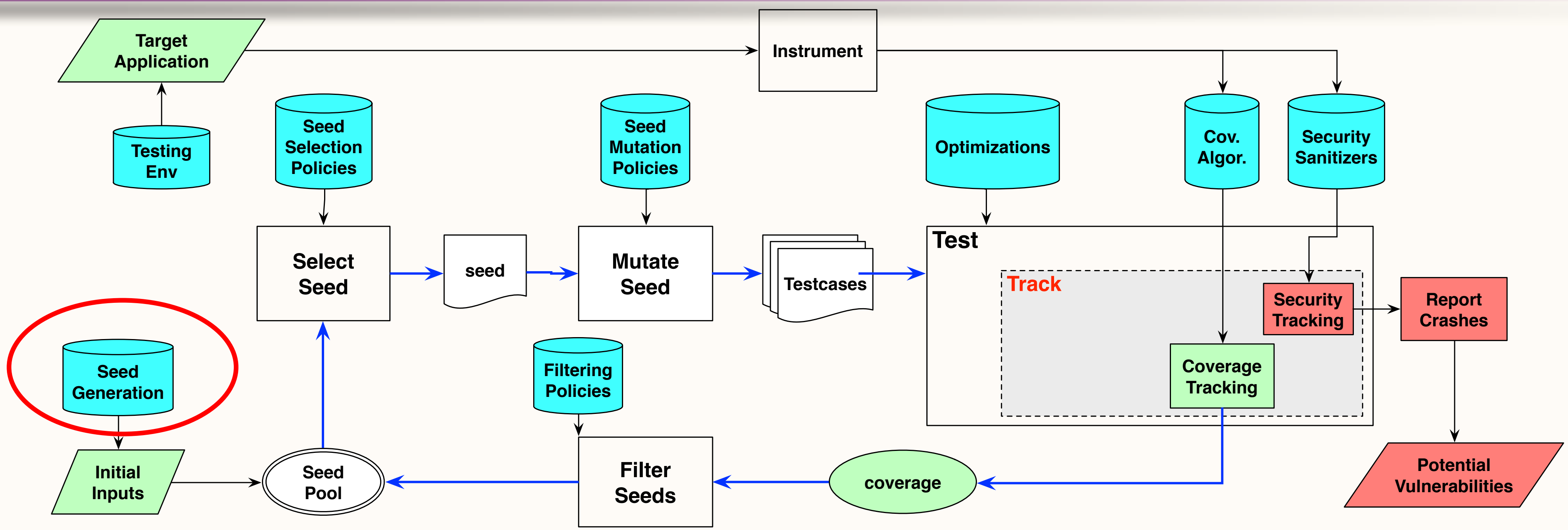


HOTracer (Sec17)

Vul Dist (ICSE20)

Improvements to Fuzzing

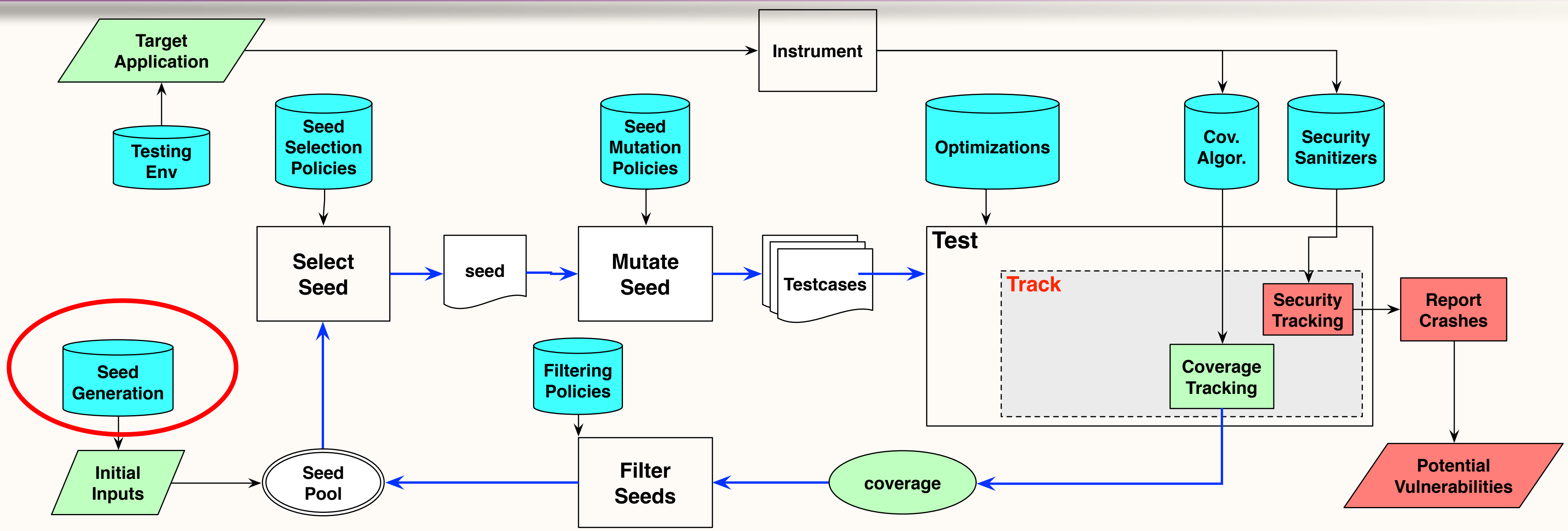
Seed Generation



How to get/generate seeds?

Skyfire (Oakland17):	learn a probabilistic CFG grammar
Learn&Fuzz (ASE17):	learn a RNN model of valid inputs
GAN (2017/11)	learn a GAN to generate legitimate seeds
Neuzz (Oakland19):	learn a NN to model input→coverage

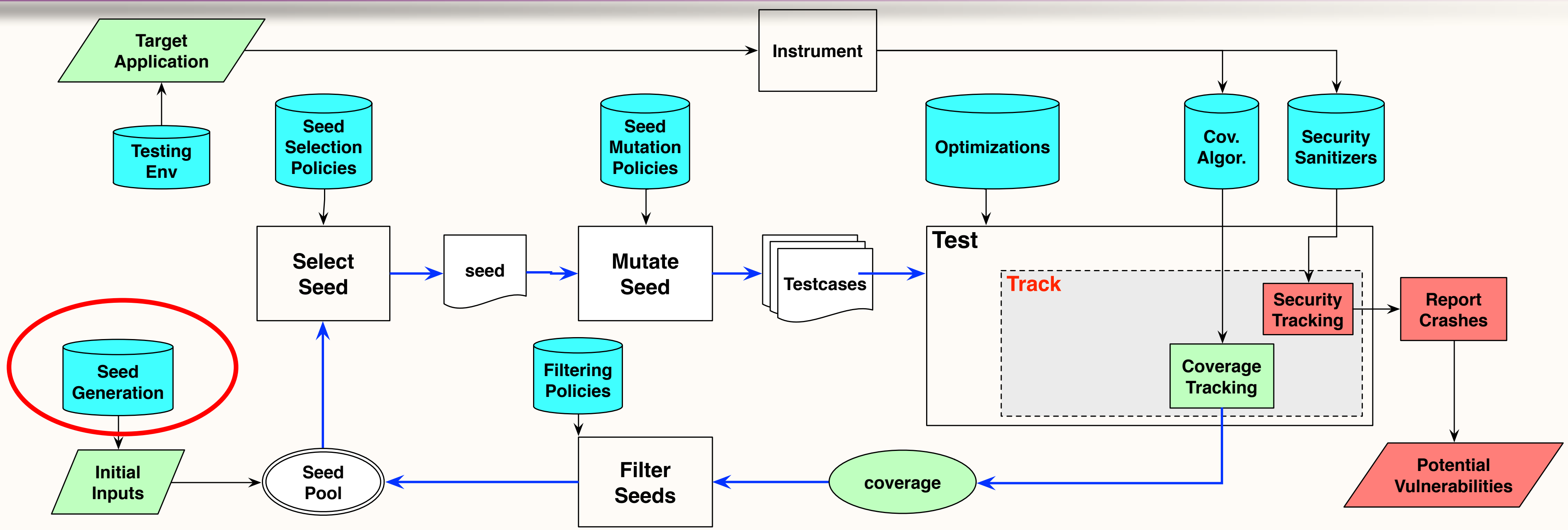
Seed Generation (2)



How to get/generate seeds?

Driller (NDSS16):	hybrid fuzzing (symbex)	DigFuzz (NDSS19)	schedule hybrid fuzzing
QSYM (CC18)	efficient symbex or binary	HFL (NDSS20)	hybrid fuzzing for kernel
Intriguer (CCS19)	field-level symbex	SAVIOR (Oakland20)	symbex
Matryoshka (CCS19)	symbex for nested branches		

Seed Generation (3)



How to get/generate seeds?

DIFUZE (CCS17): static analysis, input format of ioctrl()

FANS (USENIX Sec20): static analysis, interface of Android

IMF (CCS17): dynamic analysis, dependency of macOS

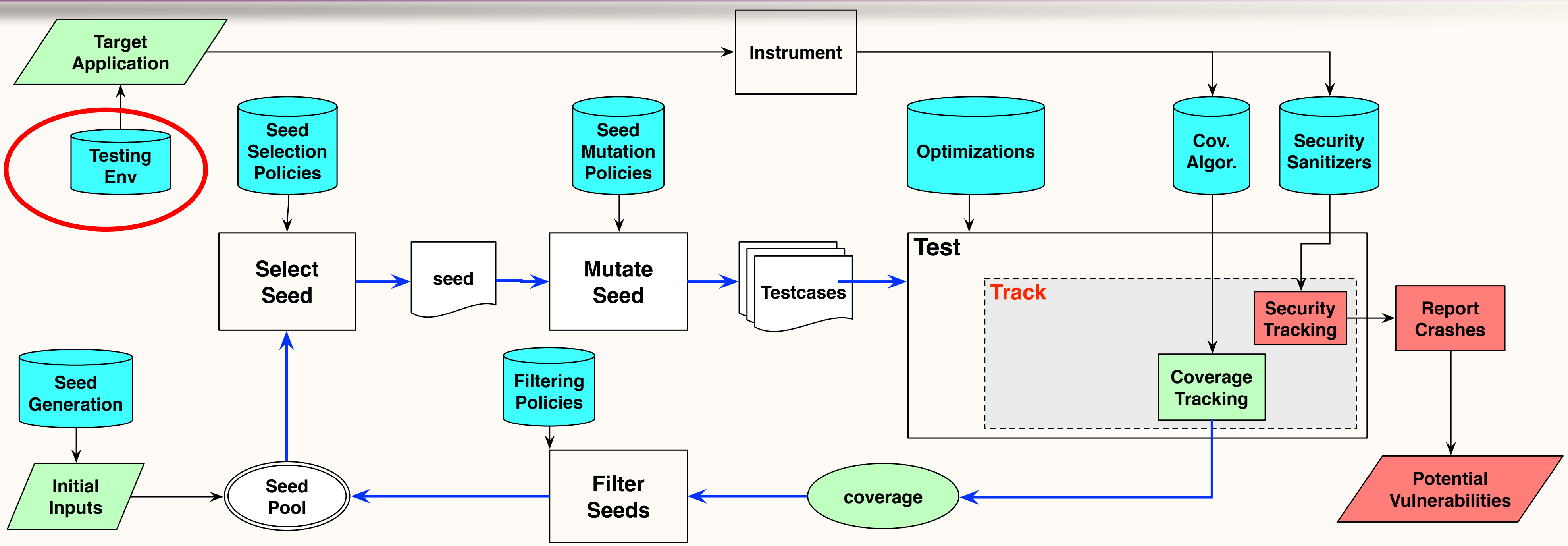
Moonshine (Sec18): static analysis, dependency of Linux

NAUTILUS (NDSS19): Context-Free Grammar by users

CodeAlchemist (NDSS19) JavaScript semantics

Grimoire (Sec19) Learn grammar during fuzzing

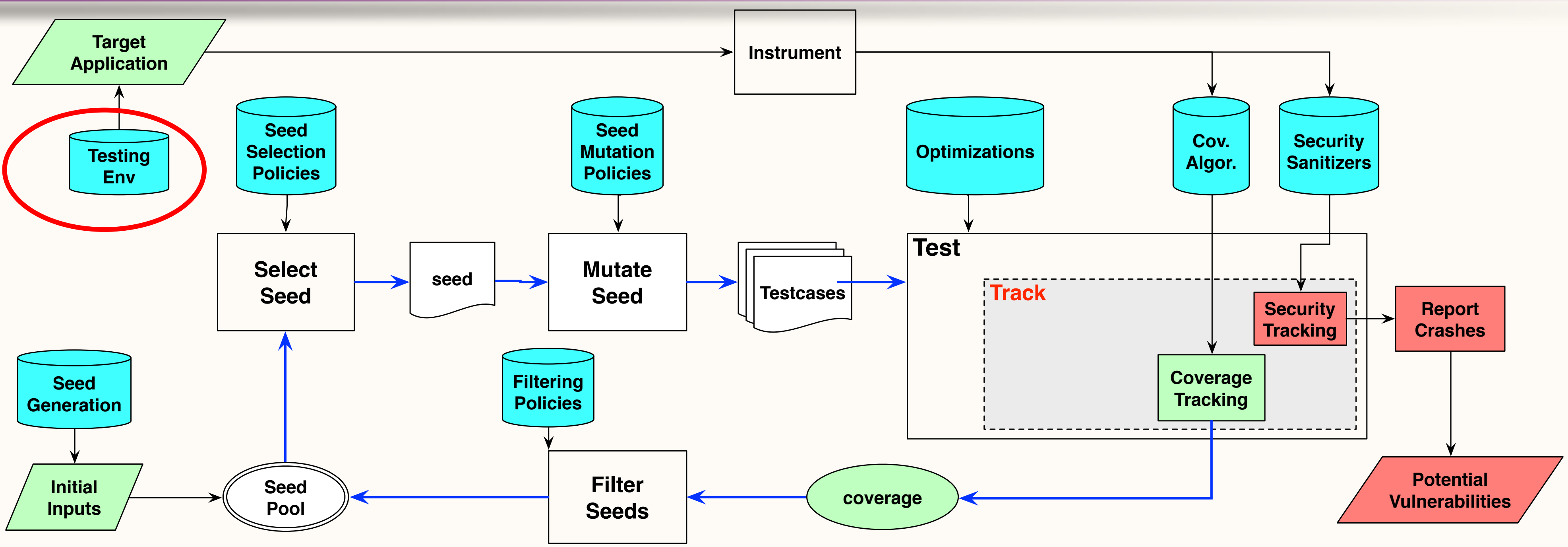
Testing Environments



How to test targets?

- T-Fuzz (Oakland18): bottleneck in binary
- Kelinci (CC17): Java applications
- TLS-Attacker (CCS17): TLS
- EFuzz (CCS17): smart grid
- Dachshund (NDSS17): JIT constant opt.
- DELTA (NDSS17): SDN applications
- IoTFuzzer (NDSS18): IoT devices.
- FirmAFL (Sec19): IoT firmware effic.

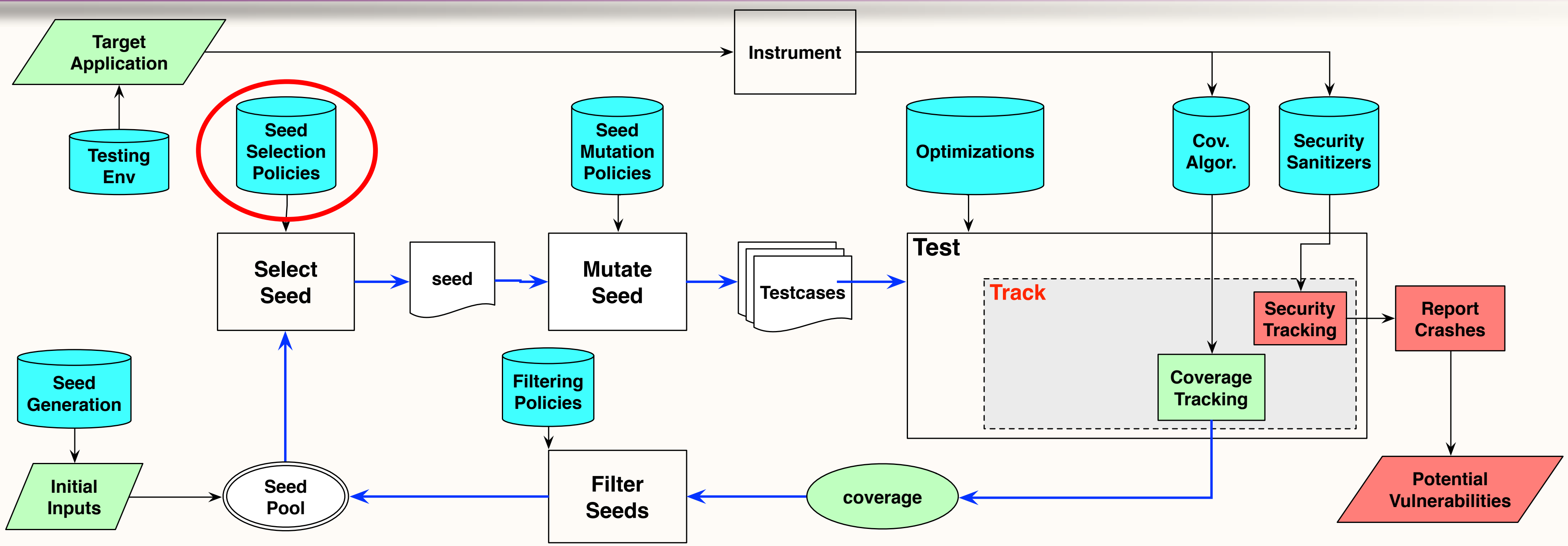
Testing Environments (2)



How to test targets?

LipFuzzer (NDSS19):	voice assistant	PeriScope (NDSS19):	driver (hardware).
HyperCube (NDSS20):	hypervisor	RVFUZZER (Sec19):	Robotic Vehicles
kAFL (USENIX Sec17):	kernel & PT	JANUS (Sec19):	File System
Charm (USENIX Sec18):	mobile device driver	SQUIRREL (CCS20):	Database

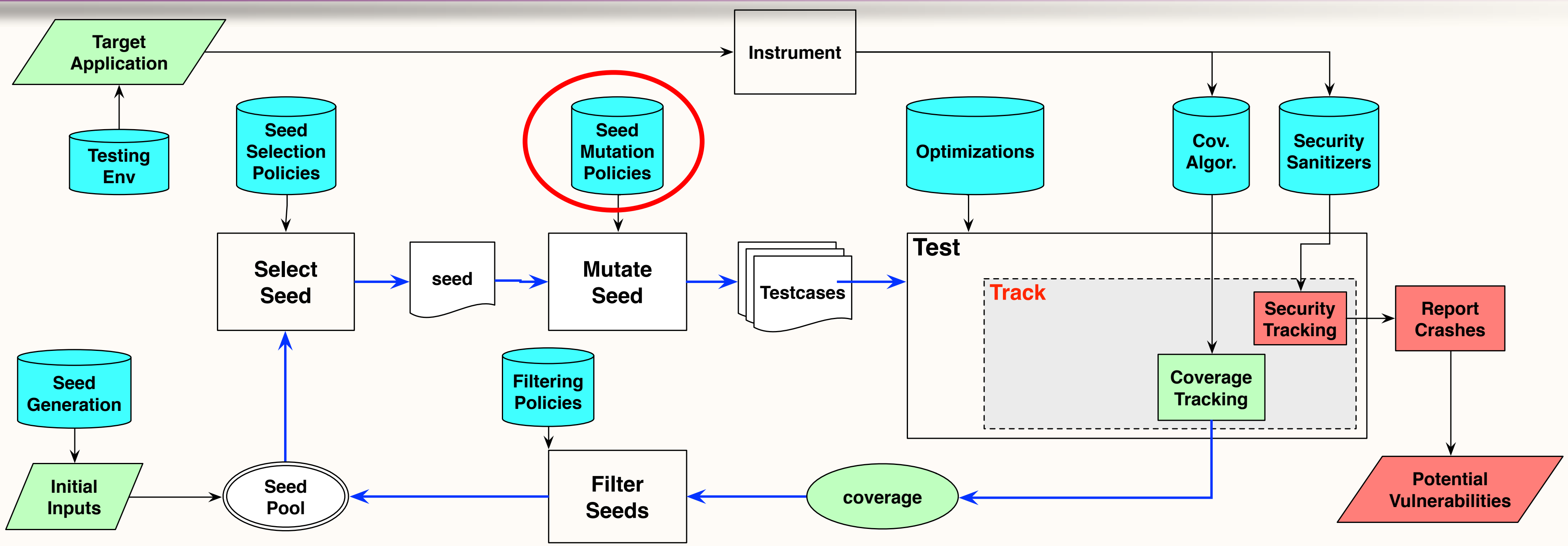
Seed Selection



How to select seed from the pool?

AFLfast (CCS16),	cold paths/seeds	QTEP(FSE17),	more vul candidates
VUzzer (NDSS17),	deeper paths	SlowFuzz (CCS17)	more comp. resources
AFLgo(CCS17),	closer paths	FairFuzz (ASE18)	rare branches
EcoFuzz(Sec17),	closer paths	CollAFL (Oakland18)	more unvisited children

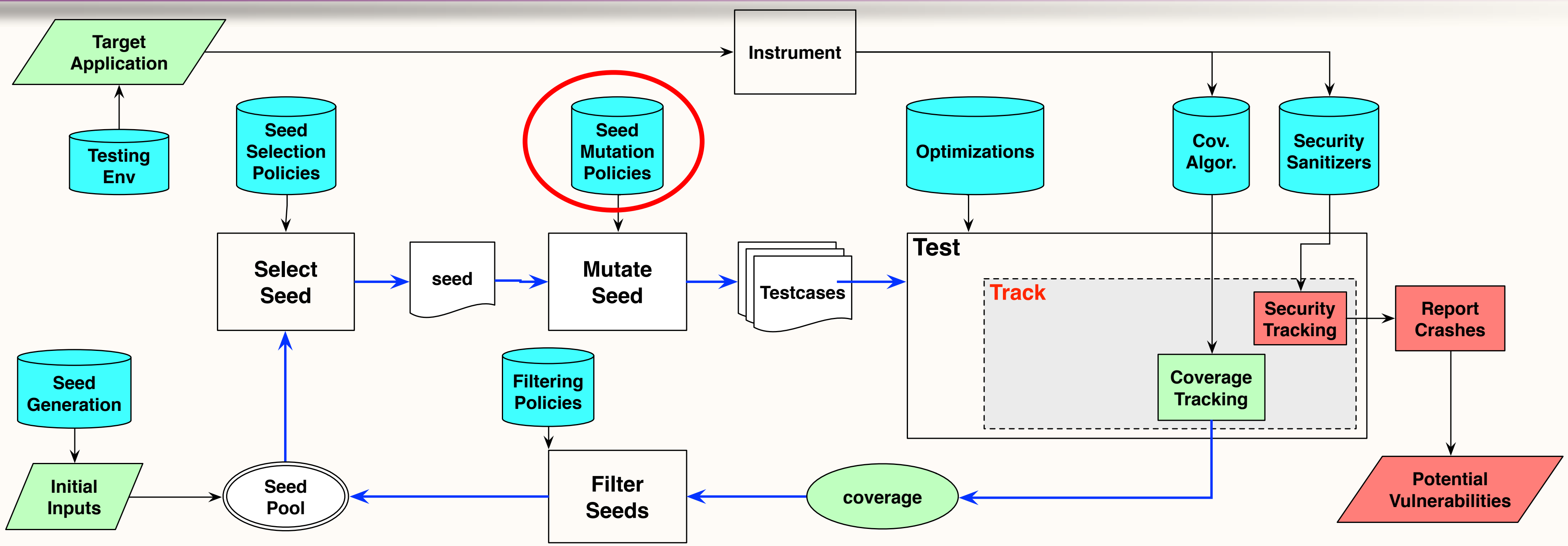
Seed Mutation



How to generate/mutate new testcases?

LSTM (Microsoft, 2017/11)	predicate which bytes to mutate first
Reinforcement Learning (2018/1)	predicate which mutation op. is better
Mopt (USENIX Sec 2019)	select the best mutation algorithm using Particle Swarm Optimization
ILF (CCS19)	learn an AI model from symbex to produce fuzzing policy

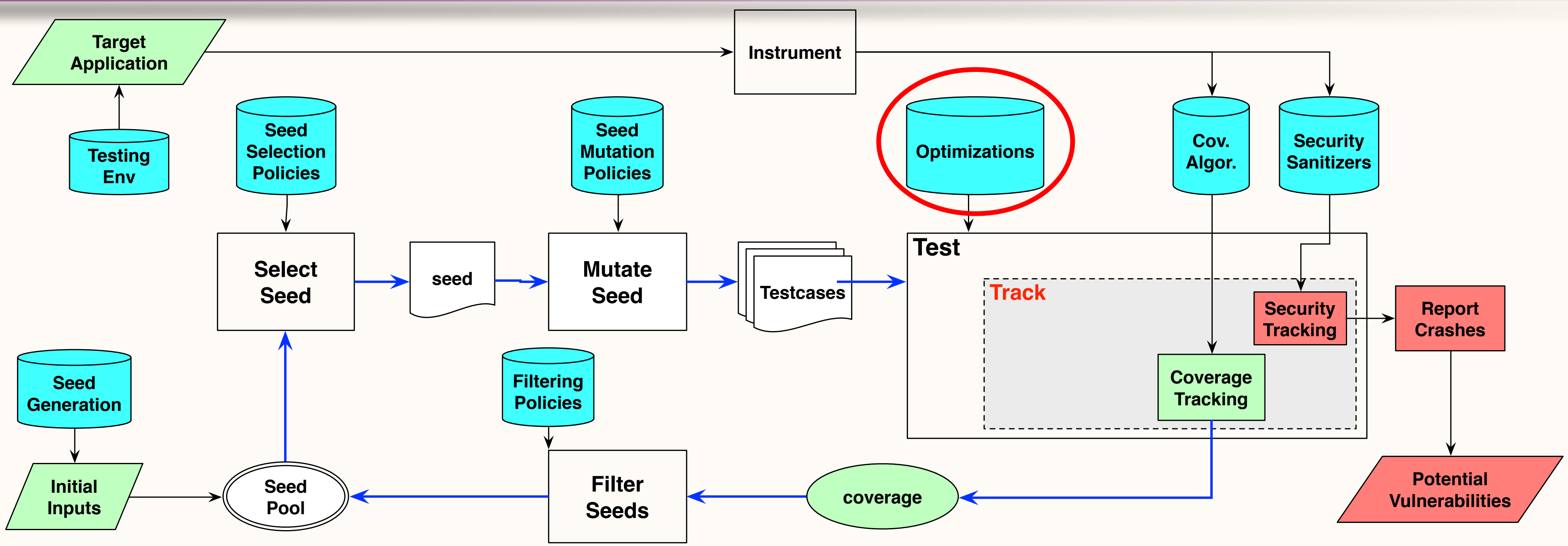
Seed Mutation (2)



How to generate/mutate new testcases?

VUzzer (NDSS17)	taint analysis: which bytes/how to mutate
REDQUEEN (NDSS19)	identify direct copy of inputs
Angora(Oakland18)	gradient descent
ProFuzzer (Oakland19)	recognize input shape by monitoring input-cov casuality
GreyOne (USENIX SEC20)	lightweight taint analysis, branch conformance

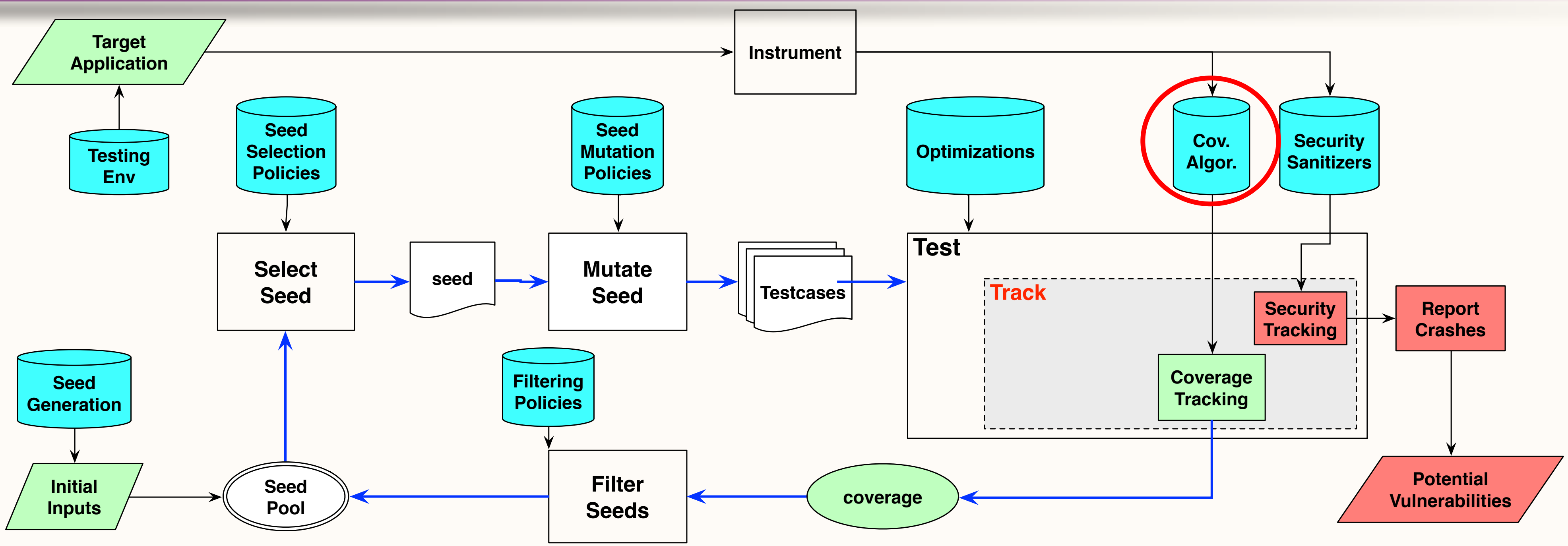
Efficient Testing



How to efficiently test target application?

perf-fuzz (CCS17)	enable efficient parallel fuzzing
PAFL (FSE18)	each fuzzer node focuses on partial code (bitmap)
Untracer (Oakland19)	remove cov tracking after a while
EnFuzz (USENIX SEC19)	combine multiple strategies with parallel fuzzing
FuzzGuard (USENIX SEC20)	remove inputs that cannot reach targets via AI

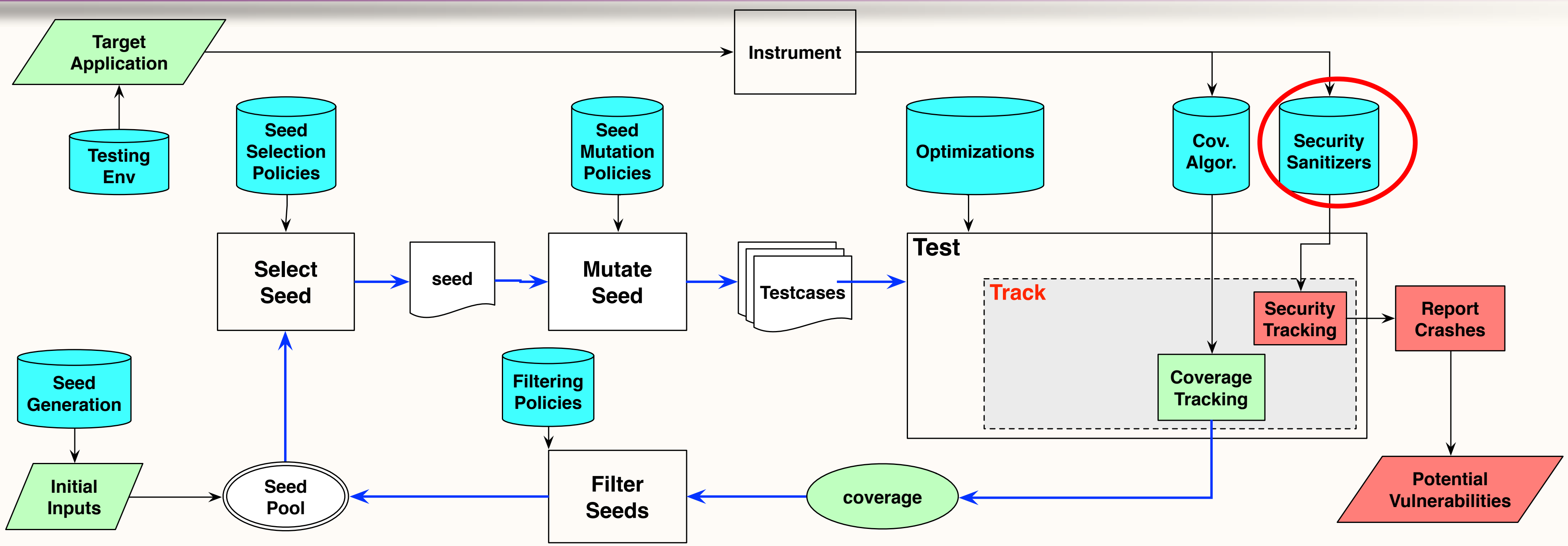
Coverage Metrics



A better/alternative coverage algorithm?

CollAFL (Oakland18)	mitigate coverage collision issue
IJON (Oakland20)	customize coverage metrics, e.g., position in the maze
AFLgo (CCS17)	directed fuzzing targeting specific code
HawkEye (CCS18)	refined directed fuzzing

Security Tracking



How to catch security violations during testing?

AddressSanitizer (ATC12):	detect spatial and temporal mem violation
Meds (NDSS18)	fix minor defects of AddressSanitizer
Razar (S&P19)	race condition bugs

Conclusions

- Fuzzing is the most popular vulnerability discovery solution.
- Genetic-algorithm-based fuzzers achieve great success, and
- Many improvements have been proposed and deployed in practice
  - Including our works
- Many more topics to explore in fuzzing

Join us

- highly motivated students
  - undergraduate intern students
  - visiting master/phd students
- Research assistants, engineers
- postdocs
- tenure-track faculty

http://netsec.ccert.edu.cn/contact/

Thanks!

Q&A