SDFUZZ: Target States Driven Directed Fuzzing

Penghui Li
The Chinese University of Hong Kong
Zhongguancun Laboratory

Wei Meng
The Chinese University of Hong Kong

Chao Zhang
Tsinghua University
Zhongguancun Laboratory

USENIX Security' 24

Background and Motivation

模糊测试涉及向计算机程序提供任意或随机输入，目的是发现如crash的意外行为。

定向灰盒模糊测试（Directed Grey-Box Fuzzing，**DGF**)

- 有别于一般的以覆盖率为导向的灰盒模糊测试，它专门测试特定的代码位置。
- 用例：Reproducing crashes, Validating vulnerabilities

探索阶段（exploration stage）：

- 大多数遵循 AFLGo 的方案，利用**覆盖率反馈**来扩展对不同代码区域的探索。

利用阶段（exploitation stage）：

- 在过去：将距离度量用于种子选择和能量调度，以引导测试方向
  - AFLGo 将基本块的距离定义为其到所有目标点的最短路径长度的调和平均数

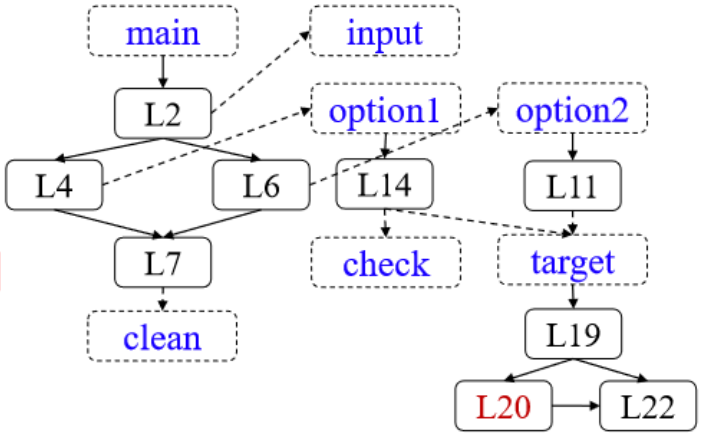
Background and Motivation

```
1  void main() {
2      int x = input();
3      if(x < 10)
4          option1(x); //(1)
5      else
6          option2(x); //(2)
7      clean();
8  }
9
10 void option2(int opcode) {
11     target(opcode);
12 }
```

```
13 void option1(int opcode) {
14     check();
15     target(opcode + 5);
16 }
17
18 void target(int arg) {
19     if (arg <= 20) {
20         assert(arg < 5);
21     }
22     ...
23 }
```

target sites

(a) Code example.



(b) ICFG.

Figure 1: A motivating example. The starting line number of a basic block is used as its name in Figure 1(b).

现有方法（基于AFLGO）的局限性：

1. 在探索阶段，基于覆盖率反馈，对不需要的代码进行不必要的测试。

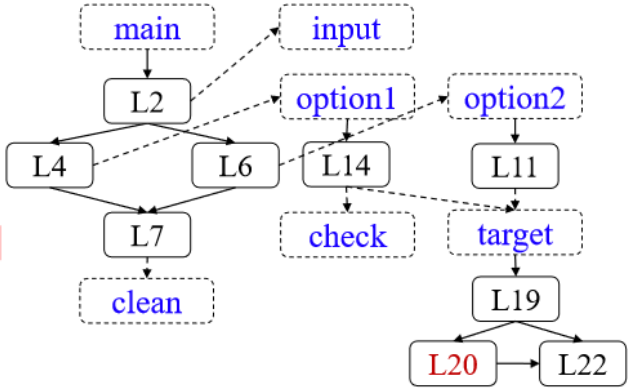
2.在利用阶段，对不可达/可达的**执行**（execution）进行不必要的测试。

Background and Motivation

```
1  void main() {
2      int x = input();
3      if(x < 10)
4          option1(x); //(1)
5      else
6          option2(x); //(2)
7      clean();
8  }
9
10 void option2(int opcode) {
11     target(opcode);
12 }
```

(a) Code example.

```
13 void option1(int opcode) {
14     check();
15     target(opcode + 5);
16 }
17
18 void target(int arg) {
19     if (arg <= 20) {
20         assert(arg < 5);
21     }
22     ...
23 }
```



(b) ICFG.

Figure 1: A motivating example. The starting line number of a basic block is used as its name in Figure 1(b).

先前的缓解措施：

- Beacon插入断言检查，通过反向区间分析（Backward Interval Analysis），如果执行不满足到达target sites的前置条件，它将提前终止。
  - 产生复杂的前置条件，运行时开销大
- SieveFuzz进一步集成了动态分析功能，以识别不需要的代码，然后相应地终止执行。
- SelectFuzz 可识别target sites的data和control相关代码。

它们**只考虑了target sites的可达性**，仍存在了许多不必要的执行（如 *(2)*）

Target States

target sites：感兴趣的代码位置。

- **漏洞报告（Vulnerability Reports）**
  - 过去：只是将**crash point**设置为target site
  - **call trace**：驱动directed fuzzer调用符合crash dumps的函数列表
  - **多目标漏洞**：为每个涉及的target site找到相关的call trace，并推断出它们的**预期到达顺序**。

```
1 0x... in target     file.c:20
2 0x... in option1    file.c:15
3 0x... in main       file.c:4
4 0x... in in __libc_start_main
```

Figure 2: Crash dump.

target states：**预期的call traces** 和 **预期的target sites到达顺序**

- **静态分析结果（Static Analysis Results）：**
  - 报告满足启发式分析条件（**the conditions of the heuristics**）的the vulnerable flows
    - 过去：将the vulnerable flows中的内存操作定为target site：探索所有可能到达这些target sites的路径
      - 许多flows不符合**the conditions of the heuristics**

Target States

Program State (PS)形式化为函数调用栈，每一项都是**函数名**和**调用位置**组成的元组

$$PS = [(Func_1, Loc_1), (Func_2, Loc_2), ..., (Func_n, Loc_n)]$$

Target State (TS)：漏洞或bug被触发时的Program State。

$$TSs = [TS_1, TS_2, ..., TS_m]$$

多目标漏洞：为每个target sites推导出一个Target State，按照target sites的到达顺序对**TS**进行排序

Overview

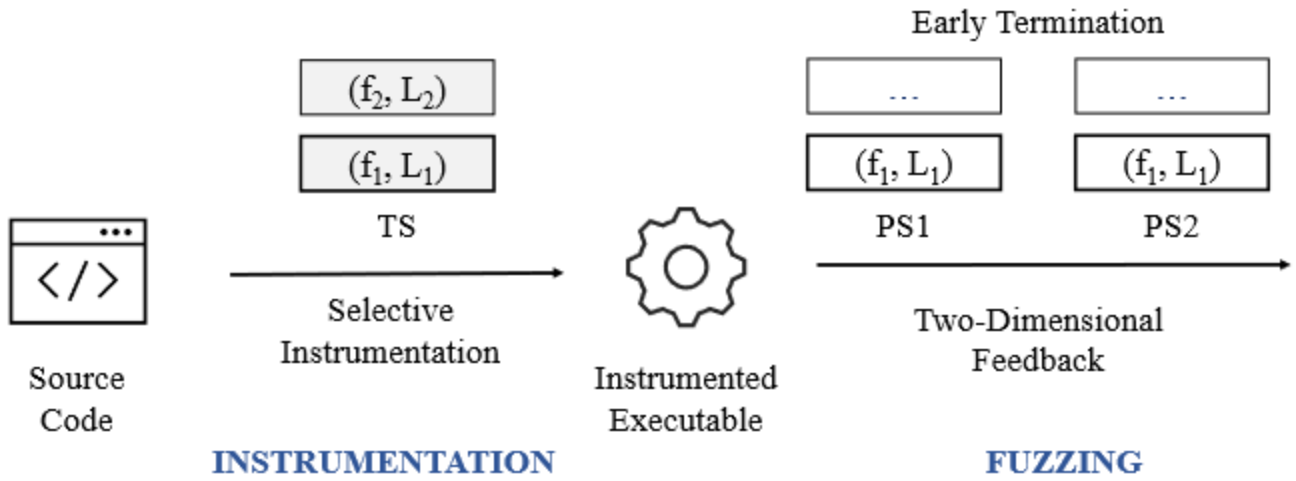


Figure 3: The workflow of SDFuzz.

提取Target States

漏洞报告（Vulnerability Reports）

1) 函数名称　　2) 调用位置

```
1 0x... in target     file.c:20
2 0x... in option1    file.c:15
3 0x... in main       file.c:4
4 0x... in in __libc_start_main
```

Figure 2: Crash dump.

- 正则表达式提取
- 确定是否符合格式
- 对Target States进行排序（例：use-after-free）

静态分析结果（Static Analysis Results）

- 自动提取须针对每种静态分析工具进行专门设计

对所需代码进行插桩

Algorithm 1: Required code identification.

```
input  : TSs, ICFG
output : requiredFuncs
initRequiredFuncs ← [ ]
requiredFuncs ← [ ]
for TS ∈ TSs do
    for f ∈ TS do
        initRequiredFuncs.insert(f)
        funcs ← backwardAnalysis(f, ICFG) // get
          functions with intra-procedural dependencies
        initRequiredFuncs.insert(funcs)
    end
end
while ! initRequiredFuncs.empty() do
    f ← initRequiredFuncs.remove()
    if f ∉ requiredFuncs then
        requiredFuncs.insert(f)
        callees ← getCallees(f, ICFG) // get callees
          of f
        initRequiredFuncs.insert(callees)
    end
end
return requiredFuncs
```

- 选取了到达target states所需的代码
  - 到达target sites的代码的子集
- 相比于SieveFuzz和Beacon：SDFuzz**不直接删除**代码，而是**基于插桩去排除**不需要的代码
  - 在缩小模糊测试范围的同时，具有容错能力。

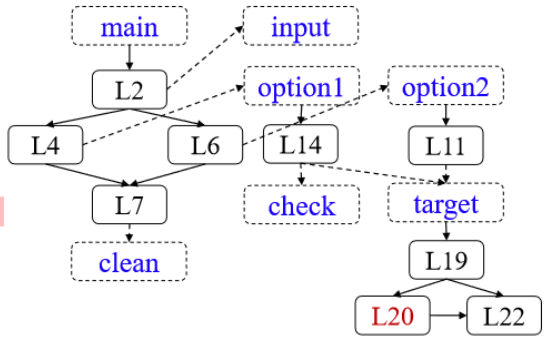
提前终止执行（Early Termination of Executions）

监控运行时的函数调用，并记录函数调用的堆栈。

- 为缓解状态爆炸和巨大开销，只跟踪Target States相关的函数的状态

```
1  void main() {
2      int x = input();
3      if(x < 10)
4          option1(x); //(1)
5      else
6          option2(x); //(2)
7      clean();
8  }
9
10 void option2(int opcode) {
11     target(opcode);
12 }
```

```
13 void option1(int opcode) {
14     check();
15     target(opcode + 5);
16 }
17
18 void target(int arg) {
19     if (arg <= 20) {
20         assert(arg < 5);
21     }
22     ...
23 }
```



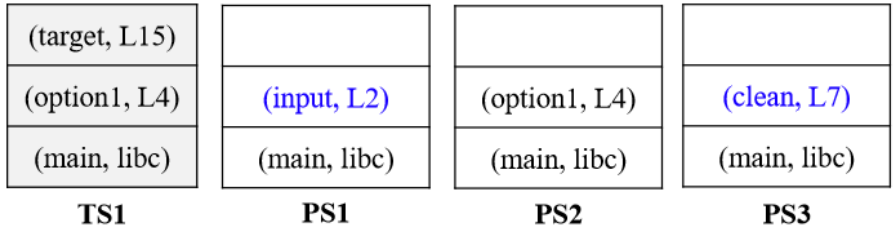
(a) Code example.

(b) ICFG.

Figure 1: A motivating example. The starting line number of a basic block is used as its name in Figure 1(b).

```
TS1=[(main, libc), (option1, L4), (target, L15)]

PS1=[(main, libc), (input, L2)]
PS2=[(main, libc), (option1, L4)]
PS3=[(main, libc), (clean, L7)]
```



(a) A target state and selected program states.

(b) Target state and selected program states, with the root deviations are in blue.

Figure 4: A target state and selected program states.

Beacon做不到这一点

两个维度的反馈（Two-Dimensional Feedback）

Algorithm 2: Execution termination and target state similarity.

```
   input  : PS, TSs, reachedTSs, ICFG
   output : score, termination, reachedTSs
1  nextTS ← null
2  termination ← false
3  for TS ∈ TSs\reachedTSs do
4  |  nextTS ← TS // find next target state
5  |  break
6  end
7  if nextTS = null then
8  |  return 1, false, reachedTSs
9  end
10 deviationIdx ← rootDeviation(PS, nextTS)
11 if deviationIdx ≠ nextTS.size then
12 |  termination ← ! ICFG.path.exists(PS[deviationIdx],
   |    nextTS[deviationIdx]) // check if recoverable
13 |  score ← (deviationIdx / nextTS.size + reachedTSs.size)
   |    / TSs.size
14 else
15 |  reachedTSs.insert(nextTS) // no deviation
16 |  score ← reachedTSs.size / TSs.size
17 end
18 return score, termination, reachedTSs
19
20 function rootDeviation(PS, TS):
21 |  index ← 0
22 |  for index<min(PS.size, TS.size) & PS[index]=TS[index]
   |    do
23 |  |  index ← index + 1
24 |  end
25 |  return index
26 end
```

Target State Feedback

将运行时的程序状态与目标状态进行比较：

相似度得分=

已经执行的Target States的函数个数 /

Target States中函数的总数

两个维度的反馈（Two-Dimensional Feedback）

Distance Feedback

过去：对CG的所有边一视同仁，不够精确

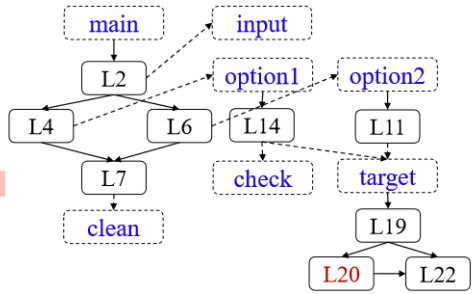
- 调用链很长 不完全等于 到达目标函数的几率很低

CG边的权重应反映caller调用callee的机会

- SDFUZZ 基于**调用点（call-site）**权重计算边权重=
- 从**caller function的起始点**到**callee的调用点**的**过程内**距离
  - 在一个caller中，同一个callee有多个调用点，则取最小权重

```
1  void main() {
2      int x = input();
3      if(x < 10)
4          option1(x); //(1)
5      else
6          option2(x); //(2)
7      clean();
8  }
9
10 void option2(int opcode) {
11     target(opcode);
12 }
```

```
13 void option1(int opcode) {
14     check();
15     target(opcode + 5);
16 }
17
18 void target(int arg) {
19     if (arg <= 20) {
20         assert(arg < 5);
21     }
22     ...
23 }
```



(a) Code example.　　(b) ICFG.

Figure 1: A motivating example. The starting line number of a basic block is used as its name in Figure 1(b).

caller和callee之间的边权重构成加权 CG

$$interDistance(f_s, f_e) = min\left(\sum_{(f_i,f_j)\in path} weight(f_i, f_j)\right)$$

种子选择和能量调度

- 主要排序属性：target state feedback
- 次要属性： seed distance feedback

target state feedback捕获运行时上下文，更加精确

Implementation

主要模糊测试组件：基于AFLGo

静态分析：利用Andersen’s points-to analysis，复用SVF的管道 -> CG + CFG

插桩：修改AFL 的 LLVM 编译器

运行时状态跟踪：实现了一个运行时库，用于程序状态维护和选择性执行终止，
并提供目标状态反馈；修改了AFL相关函数实现种子选择策略

评估：Target State Generation Capability

Magma：a widely-used fuzzing benchmark

在 **138** 个bug相应报告中，SDFUZZ 成功提取了 **127** 个正确的目标状态

- 在没有crash dumps的情况下，SDFUZZ 无法生成目标状态

Targets：1~3个

函数调用：2~6个

评估结果

Target State Generation Capability

Magma：a widely-used fuzzing benchmark

在 **138** 个bug相应报告中，SDFUZZ 成功提取了 **127** 个正确的目标状态

- 在没有crash dumps的情况下，SDFUZZ 无法生成目标状态

Targets：1~3个

函数调用：2~6个

SDFUZZ 平均消除了 48.18% 的非必要函数，并将模糊范围缩小到其他 51.82% 的必需函数

- SieveFuzz 平均消除了31.53% 的非必要代码

采用执行提前终止技术的fuzzer具有更高的吞吐量

评估结果

45 个独特的漏洞；

5次，每次限时24h

严格按照比较工具提供的说明，使用相同的硬件环境和初始种子，尽最大努力进行公平比较

SDFUZZ触发漏洞的数量最多；

大多数case中SDFUZZ耗时最少；

2.83×, 2.65×, 1.29×, 1.81×

使用Mann-Whitney U test说明显著性水平（<0.05）

Table 1: Vulnerability exposure results. Factor is the ratio of time used by a tool compared to that of SDFuzz. CE denotes compilation error. TO denotes that a tool reaches the time limit (timeout) before triggering a vulnerability. The best result of a case is underlined.

ID	Program	Location	AFLGo			WindRanger			Beacon			SieveFuzz			SDFuzz
			Time	Factor	p-val	Time	Factor	p-val	Time	Factor	p-val	Time	Factor	p-val	Time
1	libming	decompile.c:349	216	2.45	0.003	195	2.22	0.002	147	1.67	0.001	199	2.26	0.007	88
2	libming	decompile.c:398	268	1.71	0.008	348	2.22	0.003	194	1.24	0.050	282	1.80	0.030	157
3	LMS	service.c:227	5	1.67	0.009	8	2.67	0.006	3	1.00	0.001	3	1.00	0.001	3
4	mjs	mjs.c:13732	272	1.36	0.132	204	1.02	0.012	128	0.64	0.003	228	1.14	0.023	200
5	mjs	mjs.c:4908	8	2.67	0.007	5	1.67	0.004	5	1.67	0.006	3	1.00	0.001	3
6	tcpdump	print-ppp.c:729	608	4.68	0.004	708	5.45	0.003	CE	-	-	512	3.94	0.003	130
7	lrzip	stream.c:1747	372	18.60	0.005	251	12.55	0.003	38	1.90	0.001	176	8.80	0.003	20
8	lrzip	stream.c:1756	329	7.48	0.002	224	5.09	0.001	158	3.59	0.003	137	3.11	0.009	44
9	objdump	objdump.c:10875	785	5.38	0.002	752	5.15	0.008	235	1.61	0.003	327	2.24	0.003	146
10	objdump	dwarf2.c:3176	TO	-	-	618	7.92	0.001	CE	-	-	154	1.97	0.019	78
11	libssh	messages.c:1001	TO	-	-	TO	-	-	TO	-	-	TO	-	-	1,112
12	libxml2	valid.c:952	151	2.44	0.009	42	0.68	0.004	52	0.84	0.003	70	1.13	0.001	62
13	libxml2	messages.c:1001	217	1.43	0.003	209	1.38	0.002	78	0.51	0.003	192	1.26	0.018	152
14	libxml2	parser.c:10666	134	3.35	0.012	211	5.28	0.007	TO	-	-	78	1.95	0.009	40
15	libarchive	format_warc.c:537	TO	-	-	TO	-	-	TO	-	-	TO	-	-	1,039
16	Little-CMS	cmsintrp.c:642	382	2.98	0.003	565	4.41	0.003	229	1.79	0.001	258	2.02	0.004	128
17	boringssl	asn1_lib.c:459	511	4.26	0.006	368	3.07	0.004	263	2.19	0.003	346	2.88	0.006	120
18	c-ares	ares_create_query.c:196	3	3.00	0.019	3	3.00	0.122	1	1.00	0.151	1	1.00	0.132	1
19	guetzli	output_image.cc:398	42	10.50	0.030	51	12.75	0.003	17	4.25	0.004	25	6.25	0.012	4
20	harfbuzz	hb-buffer.cc:419	TO	-	-	TO	-	-	TO	-	-	1,350	2.13	0.001	633
21	json	fuzzer-parse_json.cpp:50	8	4.00	0.013	19	9.50	0.003	3	1.50	0.001	5	2.50	0.003	2
22	woff	buffer.h:86	519	1.46	0.019	638	1.79	0.003	389	1.09	0.001	443	1.24	0.003	356
23	vorbis	codebook.c:479	TO	-	-	TO	-	-	198	0.78	0.001	TO	-	-	254
24	re2	nfa.cc:532	1,121	12.18	0.005	654	7.11	0.003	157	1.71	0.001	465	5.05	0.005	92
25	pcre	pcre2_match.c:5968	55	4.23	0.001	30	2.31	0.001	8	0.62	0.005	27	2.08	0.005	13
26	tcpdump	in_cksum.c:108	369	1.16	0.104	420	1.32	0.040	CE	-	-	CE	-	-	319
27	tcpdump	print-isakmp.c:2502	615	1.21	0.009	502	0.98	0.053	TO	-	-	419	0.82	0.008	510
28	tiffcp	tiffcp.c:1596	551	3.01	0.012	580	3.17	0.064	319	1.74	0.071	611	3.34	0.050	183
29	tiffcp	tiffcp.c:1423	TO	-	-	TO	-	-	TO	-	-	1,284	1.29	0.091	994
30	imginfo	jpc_cs.c:316	93	2.21	0.022	172	4.10	0.029	25	0.60	0.032	39	0.93	0.012	42
31	imginfo	bmp_dec.c:474	182	1.52	0.010	TO	-	-	116	0.97	0.012	209	1.74	0.019	120
32	imginfo	jas_image.c:378	382	5.23	0.068	273	3.74	0.044	CE	-	-	193	2.64	0.091	73
33	lame	gain_analysis.c:224	91	2.60	0.023	30	0.86	0.033	39	1.11	0.029	58	1.66	0.0.43	35
34	lame	mpglib_interface.c:142	911	2.34	0.023	1,029	2.64	0.012	592	1.52	0.084	671	1.72	0.004	390
35	lame	get_audio.c:1452	488	1.74	0.043	391	1.39	0.045	276	0.98	0.019	401	1.91	0.029	281
36	mujs	jsrun.c:1024	347	1.96	0.003	287	1.62	0.012	591	3.34	0.004	638	3.60	0.004	177
37	mujs	jsdump.c:892	694	2.62	0.009	392	1.48	0.018	TO	-	-	192	0.72	0.008	265
38	mujs	jsdump.c:867	605	3.83	0.018	482	3.06	0.014	291	1.84	0.007	263	1.66	0.007	158
39	mujs	jsvalue.c:396	TO	-	-	TO	-	-	1,109	-	-	1,284	-	-	TO
40	libming	parser.c:3232	118	1.59	0.059	231	3.12	0.091	45	0.61	0.038	83	1.12	0.036	74
41	libming	outputtxt.c:143	372	2.13	0.007	413	2.36	0.009	283	1.62	0.023	309	1.77	0.016	175
42	libming	parser.c:3089	439	2.02	0.034	364	1.68	0.019	492	2.27	0.043	284	1.31	0.024	217
43	libtiff	tif_dirwrite.c:1901	TO	-	-	893	1.52	0.041	482	0.82	0.018	693	1.18	0.075	588
44	libtiff	tif_read.c:346	TO	-	-	TO	-	-	1,034	1.39	0.012	920	1.24	0.056	744
45	libtiff	tiffcp.c:1386	643	1.45	0.043	325	0.74	0.027	339	0.77	0.019	458	1.03	0.032	443

评估结果

si: selective instrumentation

et: execution termination

sf: target state feedback

df: distance feedback

bl: bug locations (target sites)

Table 2: Speedup of vulnerability exposure time above AFLGo. We use ✓ to denote the situation that the variant (or SDFuzz) triggers the vulnerability, but the speedup factor is not available because AFLGo does not trigger it.

ID	AFLGo$_{+si}$	AFLGo$_{+et}$	AFLGo$_{+sf}$	AFLGo$_{+df}$	SDFuzz$_{-si}$	SDFuzz$_{bl}$	SDFuzz
1	1.16	1.95	1.27	1.08	2.05	1.34	2.45
2	1.00	2.11	2.11	1.34	2.31	1.42	1.71
3	1.25	1.67	1.00	1.00	1.64	TO	1.67
4	1.30	1.17	1.17	1.08	1.19	1.25	1.36
5	2.21	2.00	0.80	1.60	2.55	1.31	2.67
6	1.24	3.22	1.12	1.16	3.78	2.49	4.68
7	2.10	9.79	3.19	2.49	12.18	TO	18.60
8	2.19	3.58	1.39	1.36	2.39	5.18	7.48
9	2.02	2.42	1.54	1.32	4.29	2.12	5.38
10	✓	✓	TO	TO	✓	TO	✓
11	TO	TO	TO	TO	✓	✓	✓
12	1.96	1.94	1.26	1.16	1.94	1.19	2.44
13	1.09	1.15	1.09	1.14	1.35	TO	1.43
14	1.72	1.97	1.12	1.54	1.78	1.28	3.35
15	TO	TO	TO	TO	TO	TO	✓
16	1.28	1.49	1.07	1.06	2.54	1.24	2.98
17	1.38	4.02	1.94	1.13	3.38	3.71	4.26
18	1.50	1.50	1.39	1.69	2.32	1.98	3.00
19	1.08	1.16	0.78	1.14	3.19	4.29	10.50
20	TO	✓	TO	TO	✓	TO	✓
21	1.60	1.33	1.12	1.04	1.64	2.91	4.00
22	1.06	1.30	1.60	1.00	1.13	1.28	1.46
23	TO	✓	TO	TO	✓	✓	✓
24	6.96	5.66	2.38	2.12	7.49	9.98	12.18
25	1.28	1.90	1.47	1.08	2.58	2.49	4.23
26	1.09	0.95	1.11	1.03	1.13	0.87	1.16
27	1.14	1.19	1.04	1.10	1.03	0.91	1.21
28	2.14	2.73	1.31	1.25	2.41	2.13	3.01
29	TO	TO	TO	TO	TO	✓	✓
30	1.41	1.33	1.31	1.26	1.54	1.92	2.21
31	1.43	1.11	1.24	1.34	1.09	1.21	1.52
32	2.81	4.21	1.71	1.32	4.43	3.37	5.23
33	2.07	1.83	1.29	1.48	1.95	1.87	2.60
34	1.54	1.96	1.20	1.29	2.24	2.18	2.34
35	1.34	1.44	1.32	1.15	1.23	1.72	1.74
36	1.23	1.67	1.29	1.16	1.06	1.27	1.96
37	1.86	2.23	1.21	1.42	2.47	2.17	2.62
38	1.78	2.81	1.49	1.19	2.94	1.28	3.83
39	TO	TO	TO	TO	TO	TO	TO
40	1.32	1.41	1.18	1.09	1.52	1.43	1.59
41	1.54	1.98	1.26	1.05	1.24	1.74	2.13
42	1.76	1.53	1.32	1.15	1.92	1.39	2.02
43	✓	✓	✓	TO	✓	TO	✓
44	TO	✓	TO	TO	✓	✓	✓
45	1.42	1.36	1.29	1.13	1.19	TO	1.45
Avg.	1.56	1.94	1.32	1.24	2.11	1.87	2.83

评估结果

New Vulnerability Discovery

使用SDFUZZ验证SVF静态分析的结果

Table 3: Vulnerability discovery results.

Program	Statically Reported	SDFuzz Validated
libjpeg	46	2
tinyexr	22	1
pugixml	59	1
ffmpeg	32	0
Total	159	4

讨论（潜在改进）

Requirement of target states：

- 某些应用场景（如补丁测试）可能无法提供可用的目标状态。
  - 符号执行等技术 为补丁合成可行的目标状态。

States other than target states

Incomplete call graph：

- 一些call targets无法由静态推断
  - 动态跟踪来监控函数调用，并相应地在调用图中添加额外的调用边
  - 利用其他先进的类型推断方法

结论

定向灰盒模糊测试往往会不必要地探索无法触发漏洞的代码和路径。

在本文中，我们提出了 SDFUZZ，一种由**目标状态驱动**的高效定向Fuzzer，以缓解这一问题。

SDFUZZ 通过**消除不需要的代码，提早终止**无法达到目标状态的执行，排除了不必要的探索。

SDFUZZ 还采用了二维反馈机制来主动引导测试方向。

评估结果表明，SDFUZZ 能更快地触发漏洞，性能优于之前的工作。

SDFUZZ 还发现了四个以前未知的漏洞，证明了其在自动漏洞验证方面的实用价值。

Thank you for listening