Finanziato dall'Unione europea
NextGenerationEU

Ministero dell'Università e della Ricerca

Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

SERI
SECURITY AND RIGHTS IN T

Predictive Context-sensitive Fuzzing

Pietro Borrello (Sapienza)
Andrea Fioraldi (EURECOM)
Daniele Cono D'Elia (Sapienza)
Davide Balzarotti (EURECOM)
Leonardo Querzoni (Sapienza)
Cristiano Giuffrida (VU Amsterdam)

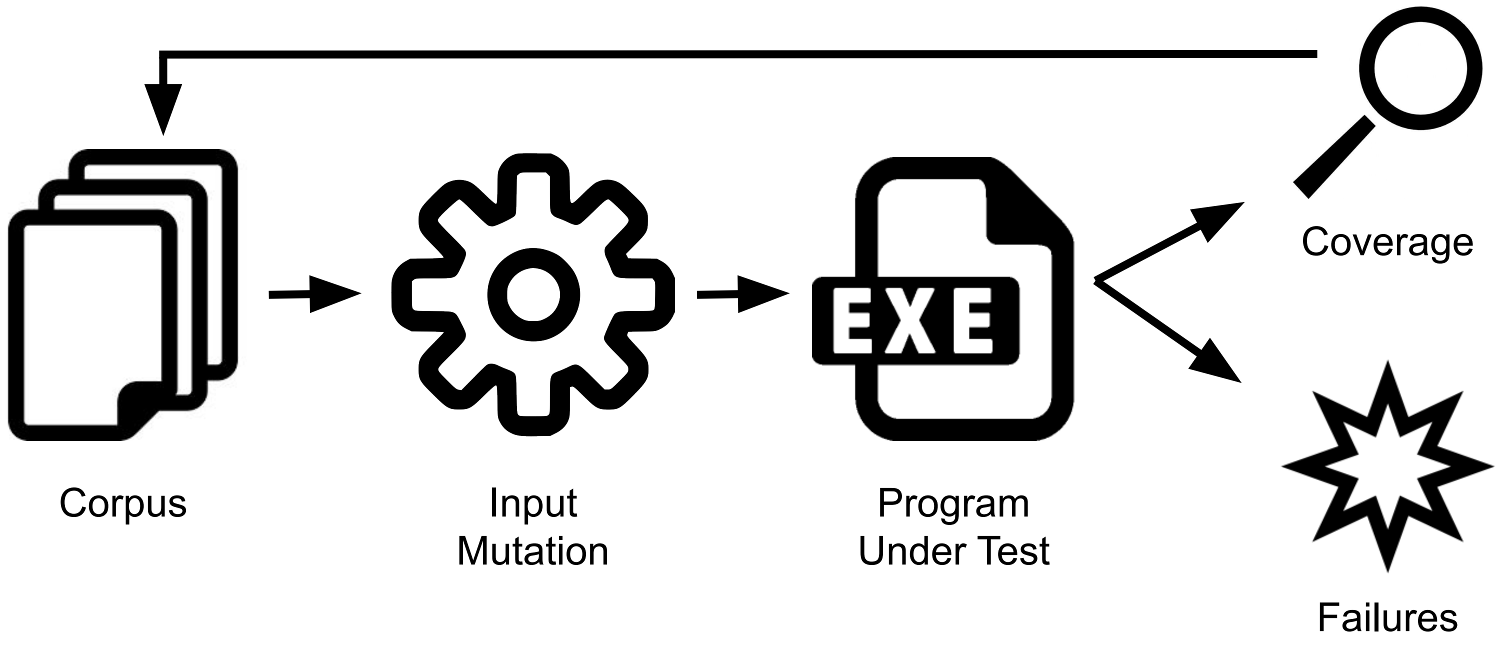
Feb 29, 2024. **NDSS'24 Symposium**

SAPIENZA
UNIVERSITÀ DI ROMA

EURECOM
Sophia Antipolis

VUSec

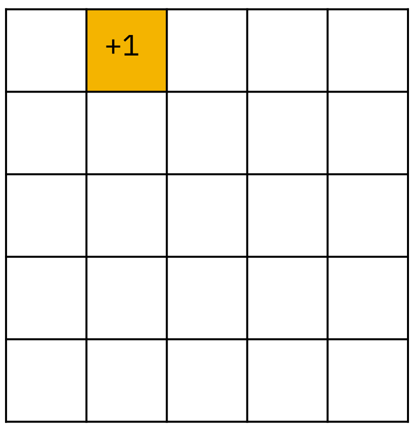
Coverage-guided fuzzing

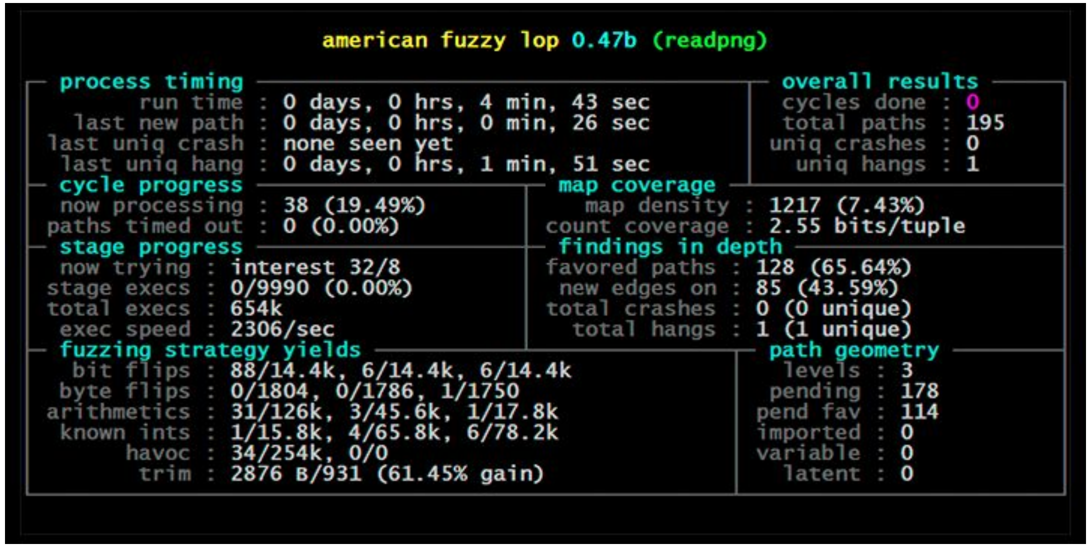


Edge-coverage guided fuzzing

```
hash_edge(0x41414141, 0x42424242)

0x41414141: jmp 0x42424242
```





Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERI SECURITY AND RIGHTS IN T

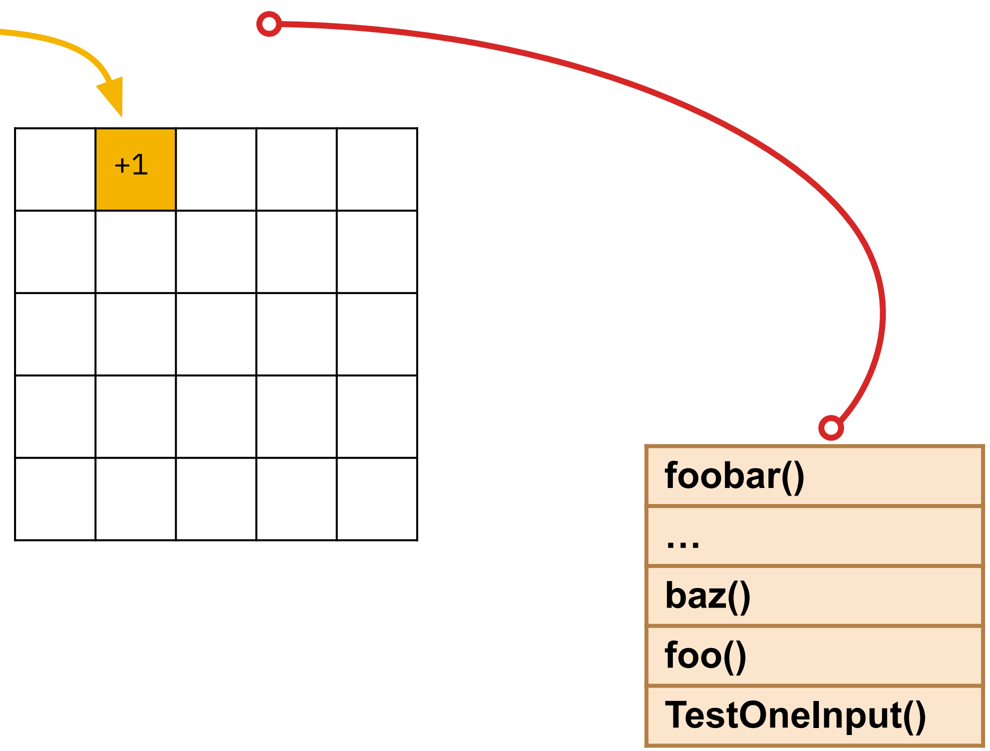
Context-sensitive, edge-coverage guided fuzzing

```
hash_edge(0x41414141, 0x42424242, ctx)
```

```
foobar:
<ctx = ctx ^ foobar>
...

0x41414141: jmp 0x42424242

...
<ctx = ctx ^ foobar)>
ret
```



Why helpful?

```
void cmd_handler_foo(int a, size_t b) { memset(buf, a, b); }
void cmd_handler_bar(int a, size_t b) { cmd_handler_foo(...
void cmd_handler_baz(int a, size_t b) { cmd_handler_bar(...

typedef void (* dispatch_t)(int, size_t);

dispatch_t handlers[UCHAR_MAX] = {
    cmd_handler_foo,
    cmd_handler_bar,
    cmd_handler_baz,
};

int main(int argc, char **argv)
{
    int cmd;

    while ((cmd = getchar()) != EOF) {
        if (handlers[cmd]) {
            handlers[cmd](getchar(), getchar());
```

Project Zero

News and updates from the Project Zero team at Google

Wednesday, December 1, 2021

This shouldn't have happened: A vulnerability postmortem

Posted by Tavis Ormandy, Project Zero

Introduction

This is an unusual blog post. I normally write posts to highlight some hidden attack surface or interesting complex vulnerability class. This time, I want to talk about a vulnerability that is neither of those things. The striking thing about this vulnerability is just how simple it is. This should have been caught earlier, and I want to explore why that didn't happen.

In 2021, all good bugs need a catchy name, so I'm calling this one "BigSig".

First, let's take a look at the bug, I'll explain how I found it and then try to understand why we missed it for so long.

🚨 coverage of cmd **bar** is a superset of cmd **foo**

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERI SECURITY AND RIGHTS IN T

Challenges to efficiency

1. coverage map explosion
2. queue explosion

Fuzzer	Queue size	Exec/sec	Collisions (map use)
classic edge (2^16 map)	9 911	609	9.8% (19.86%)
pcguard edge (coll.-free)	11 093	572	-
ctx-sensitive (2^16 map)	(2) **33 675**	530	(1) **50.7%** (79.54%)
ctx-sensitive (2^20 map)	(2) **21 157**	(1) **84**	1.2% (7.21%)
predictive ctx-sensitive	15 455	490	-

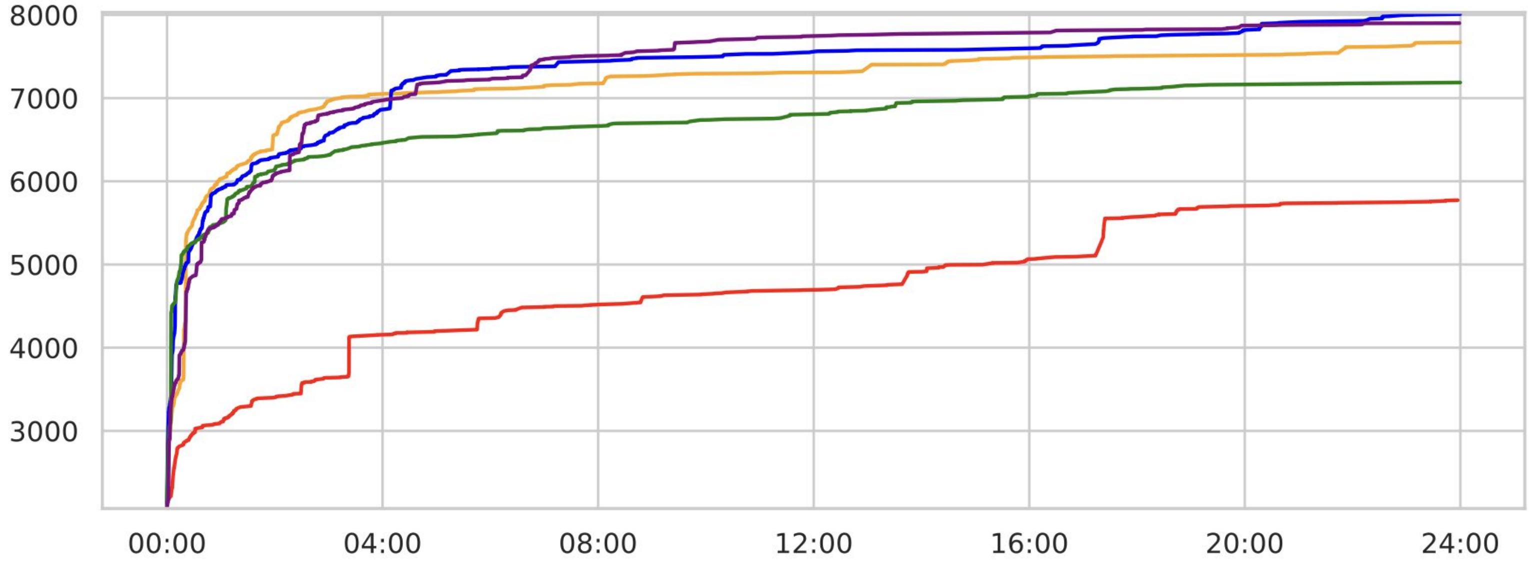
24h on **libxml2** - AFL++ 3.15a
Drivers & seeds: FuzzBench

Challenges to efficiency

1. coverage map explosion
2. queue explosion

Fuzzer
classic edge (2^16 map)
pcguard edge (coll.-free)
ctx-sensitive (2^16 map)
ctx-sensitive (2^20 map)
predictive ctx-sensitive

Impact on edge coverage



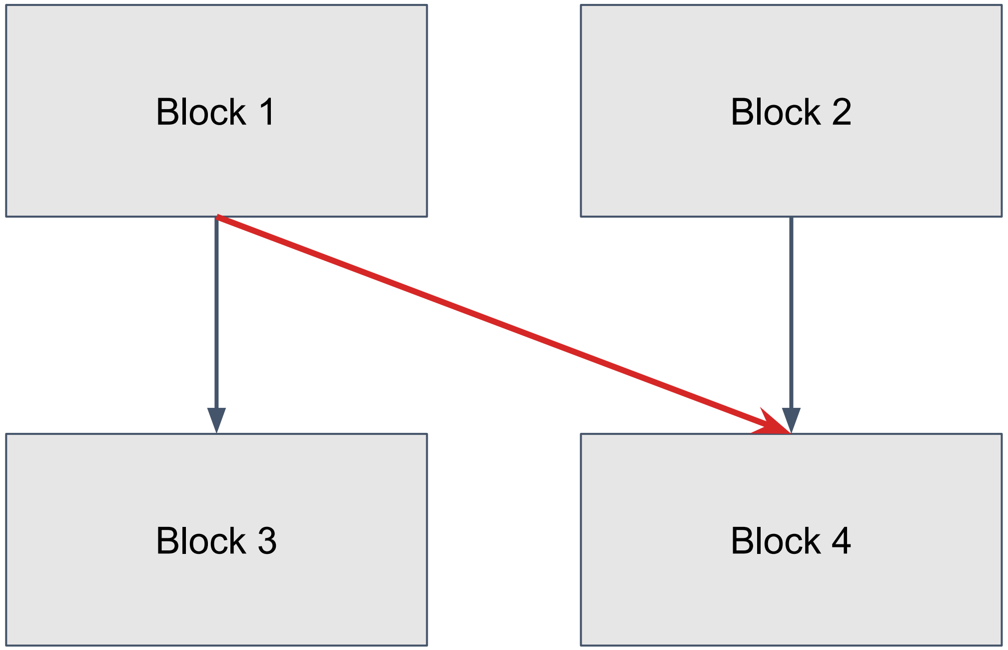
Making it efficient…

Key ideas

1. encode context without collisions
2. track context only at selected regions
3. predict profitable regions

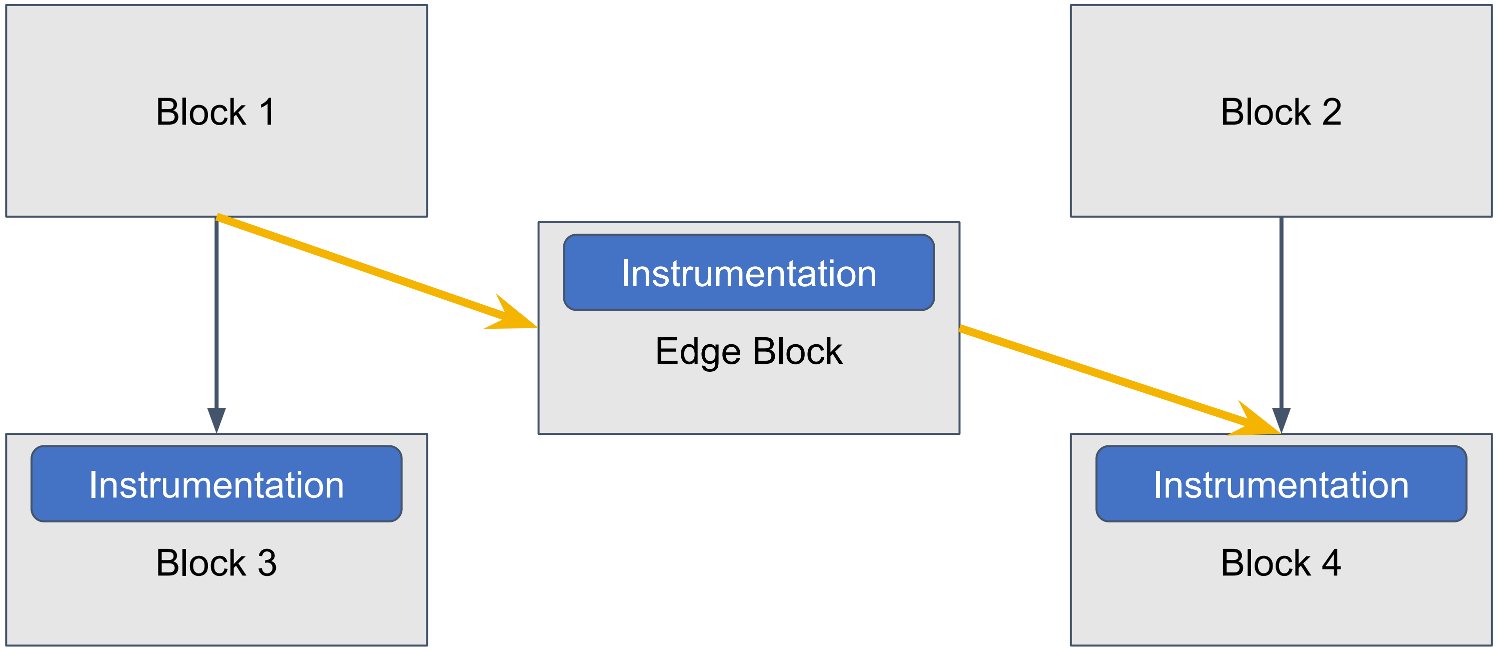
Finanziato dall'Unione europea
NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERI
SECURITY AND RIGHTS IN T

Collision-free edge coverage



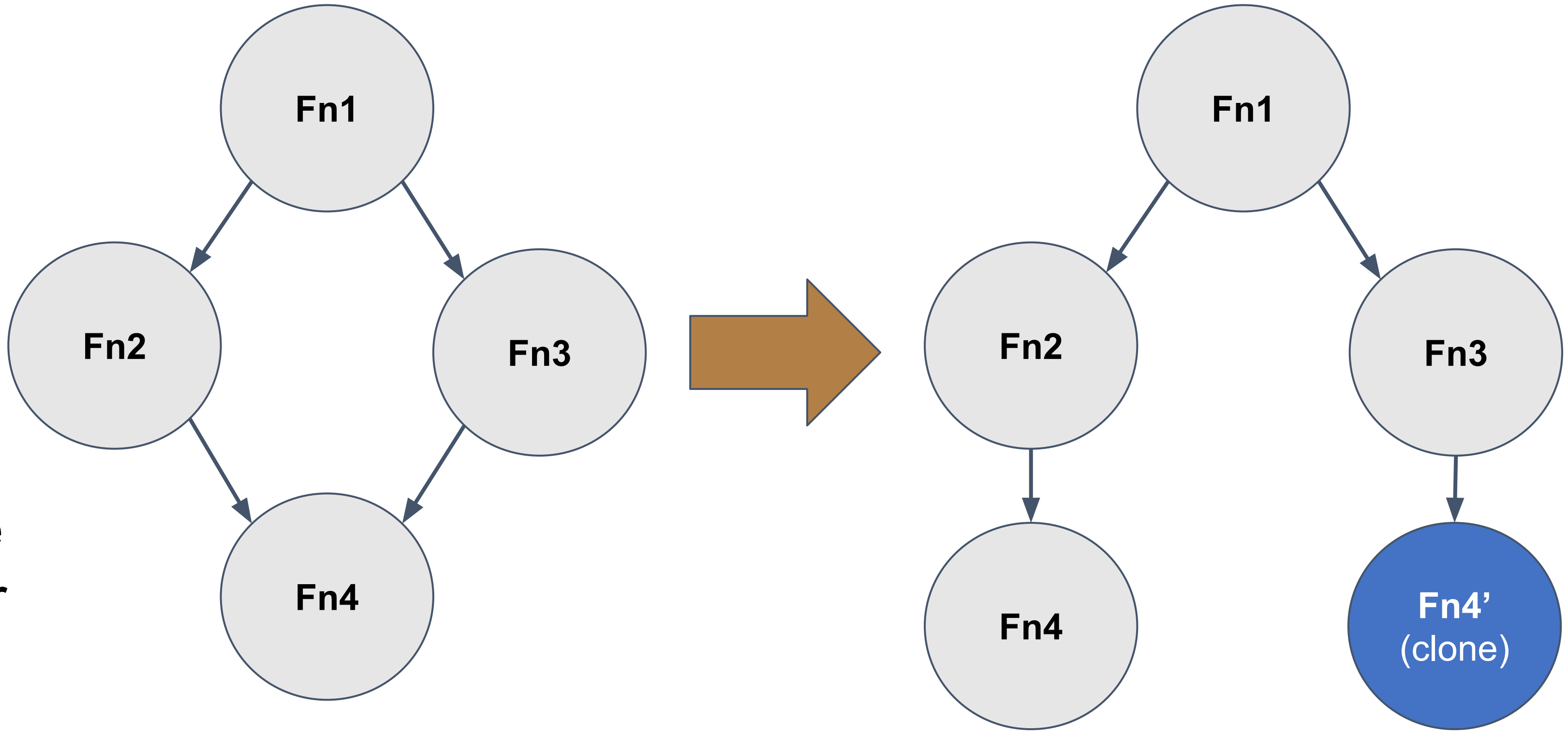
Splitting CFG critical edges removes collisions in edge-coverage tracking

Collision-free edge coverage



As basic-block IDs now suffice to uniquely identify edges

Collision-free context-sensitivity?



By **cloning** a function, we get unique CFG edges for a (caller, callee) pair

Selective sensitivity

Cloning is fuzzer-friendly, but the path explosion problem stays!

Benchmark (FuzzBench)	Edges	Functions	Call sites	Calling contexts
ffmpeg	716 K	5 K	44 K	8 M
libarchive	67 K	<1 K	4 K	27 M
libhevc	120 K	2k	< 1K	125 M
libxml2	104 K	1k	7 K	44 B
njs	57 K	490	4 K	13 M

Initial strategies

Prioritize within cloning **budget**

- favor call sites from nodes closer to call-graph root (harness)
- favor call sites closer to leaves
- treat every call site with the same priority

vs. random selection

📣 No explosion, but none > pcguard 👎

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA | SERI SECURITY AND RIGHTS IN T

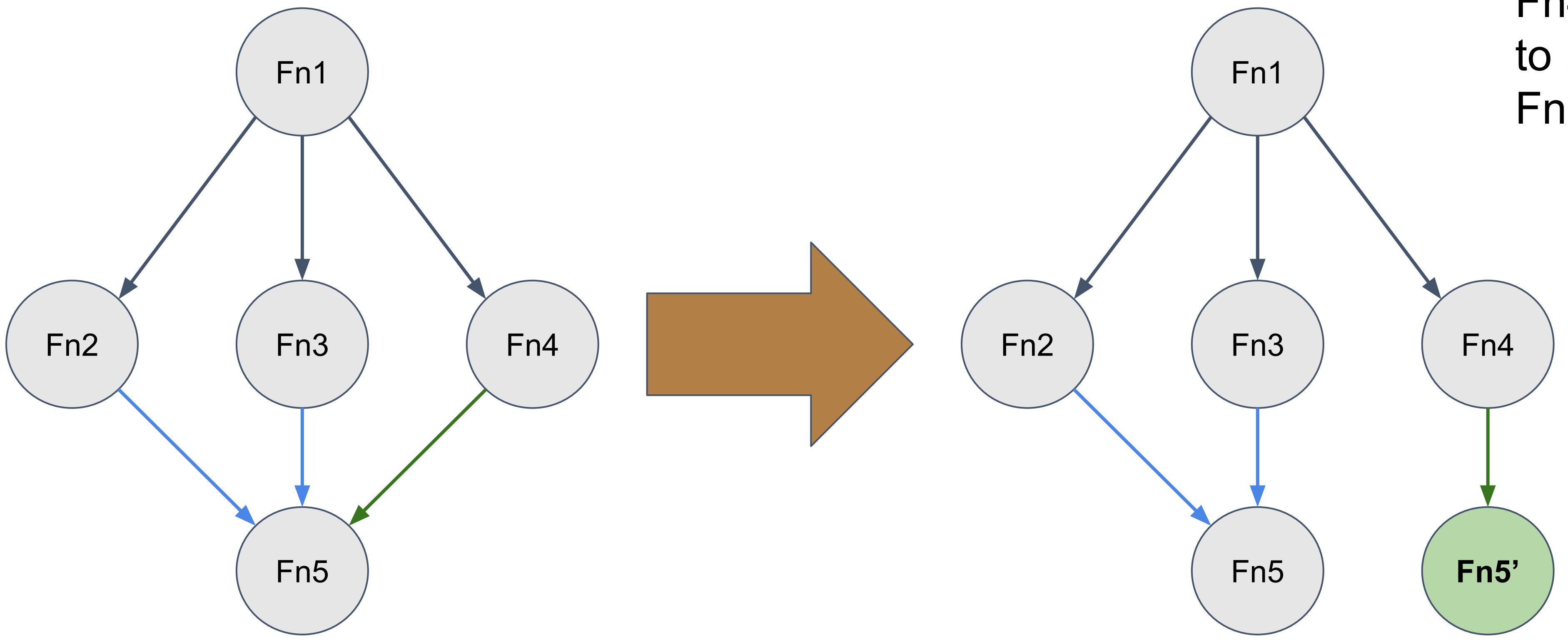
Data flow-based prediction

Prioritize call sites with distinctive incoming data

- clone one if it passes data "seen less often" at other callers
- **diversity** as a proxy for interesting
- needs only call-site sensitivity (i.e., no full contexts)
- focus on pointer-type arguments

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

Data flow-based prediction



Fn4 passes objects to Fn5 not seen at Fn2 or Fn3? Clone it

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERI SECURITY AND RIGHTS IN T

Implementation

- gllvm
- AFL++ 3.15a
- LLVM 10
- SVF framework

https://github.com/eurecom-s3/predictive-cs-fuzzing

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
SERI SECURITY AND RIGHTS IN T

Evaluation

RQ: Can we find bugs that existing approaches overlook?

RQ: How is fuzzing performance affected?

Who: pcguard LTO, Angora-style CS fuzzing, Predictive CS fuzzing

How: 16 programs from FuzzBench bug benchmarks
budget of 2^18 entries (L2 size)

Bug counts

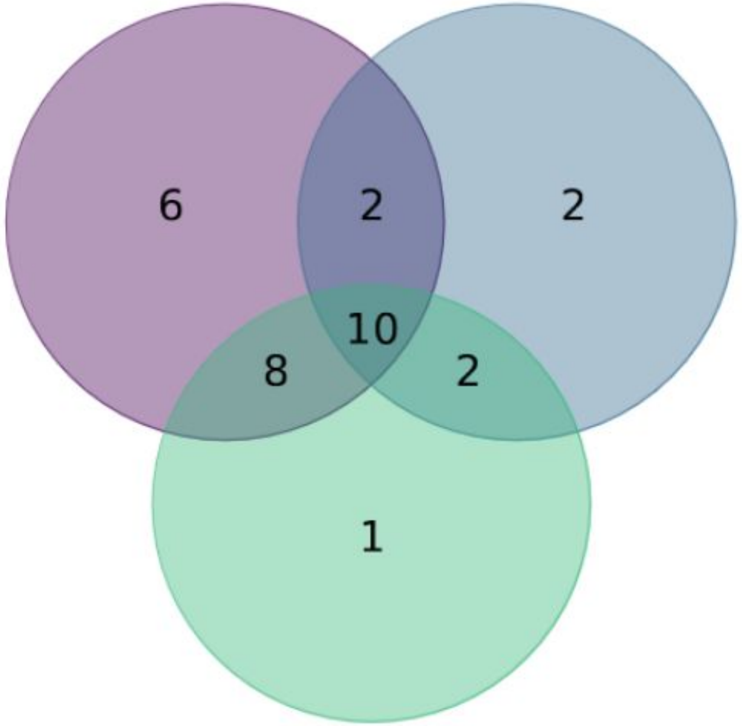
- predictive (125)
- lto (112)
- context (102)

Highlights

- +11.6% than lto, +22.5% than context
- 23 of our bugs (19.2%) were **missed** by lto (7 from new coverage, 16 from exploitation)

And new bugs!

■	predictive	(26)
■	lto	(16)
■	context	(21)



Highlights

- 26 out of 31 were exposed by predictive
- 8 security issues (1 each in ffmpeg, njs, libhevc, and matio; 4 in stb) - 6 CVEs assigned

Fuzzing performance

Trends

- **queue size:** +26.4% vs lto (context: +81.7%)
- **throughput:** 6.5% slower than lto (context: 20.3%)
- **coverage:** close to lto on 12/16 subjects, better on 8
- tenable compilation costs, 3.6x binary size increase

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA | SERI SECURITY AND RIGHTS IN T

Final remarks

Existing approaches face an **impossible trade-off** between collisions and trashing from queue/map explosion. We show a profitable avenue as we proactively select the most promising contexts based on data-flow diversity

Future opportunities: non-pointer arguments, cloning for indirect calls