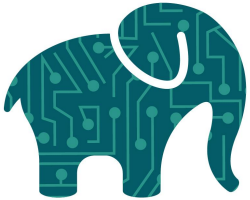


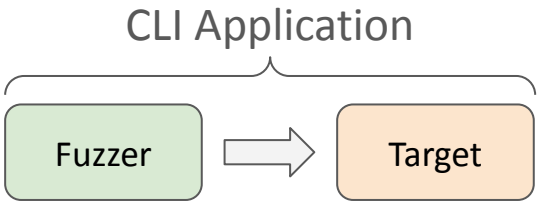

Liberating Libraries Through Automated Fuzz Driver Generation: Striking a Balance Without Consumer Code

Flavio Toffalini[*+], Nicolas Badoux[*], Zurab Tsinadze[*], Mathias Payer[*]

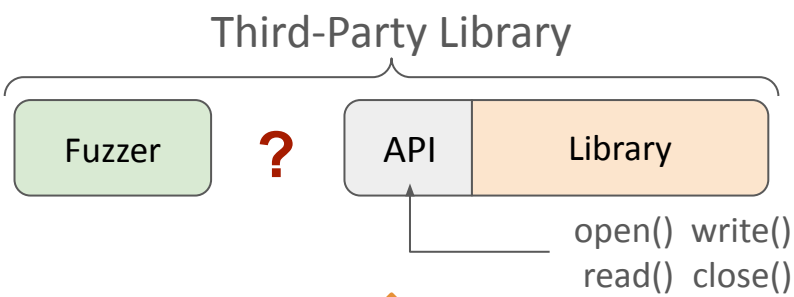
[+] Ruhr University Bochum, Germany

[*] EPFL, Switzerland

Testing Third-Party Libraries: What Semantics?



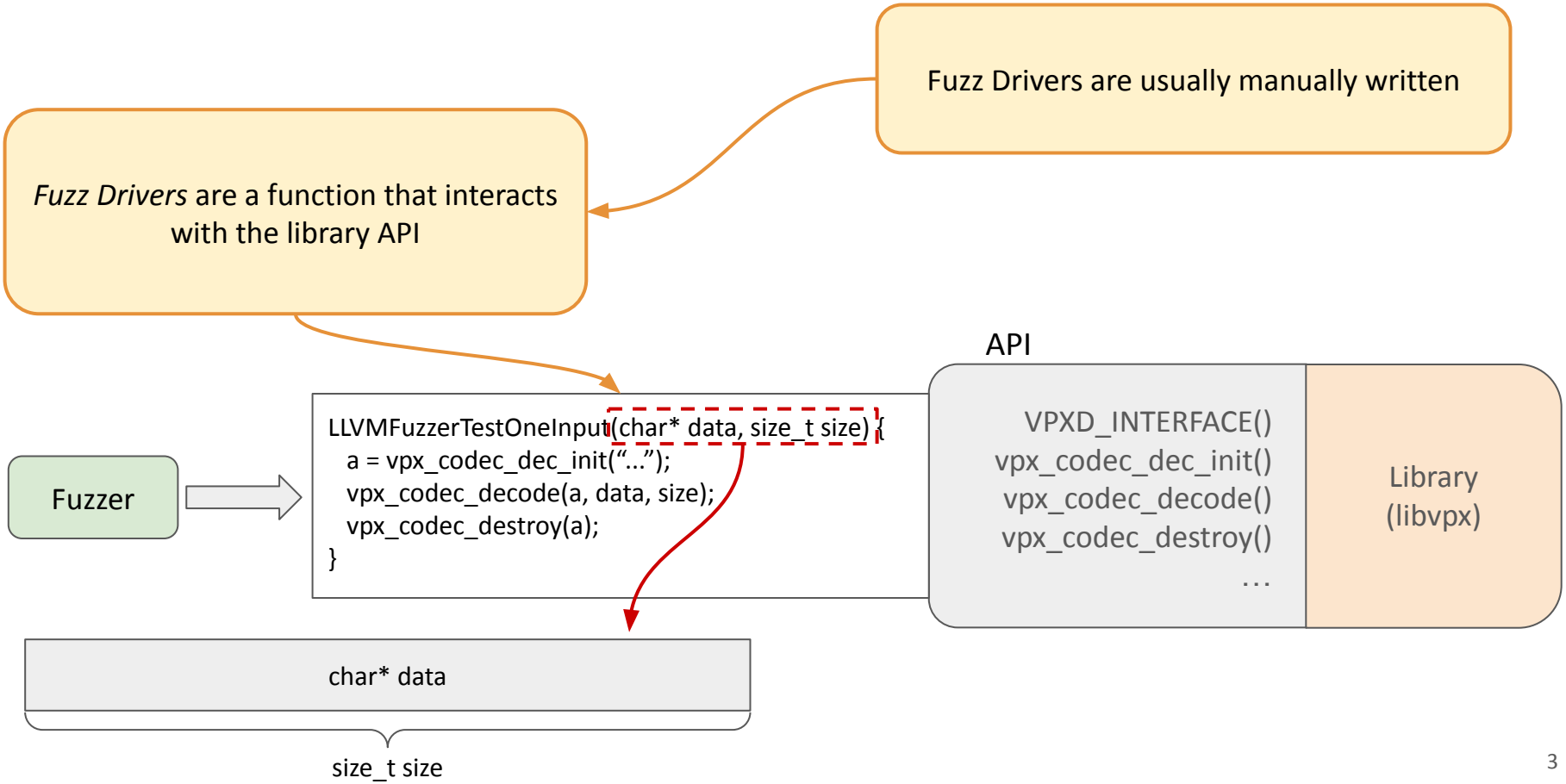
VS



Regular fuzzing targets standalone applications through main()

Libraries expose a complex Application Programming Interface (API) without clear dependencies.

Testing Third-Party Libraries



Problems in Driver Generation

Automatic generating drivers is a two-sided task:

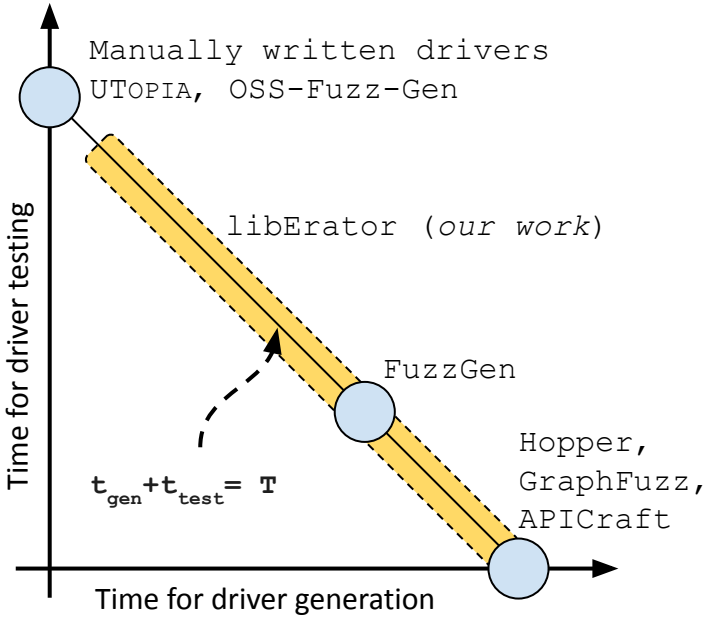
- Generating code

- Test the generated code

This leads to two questions

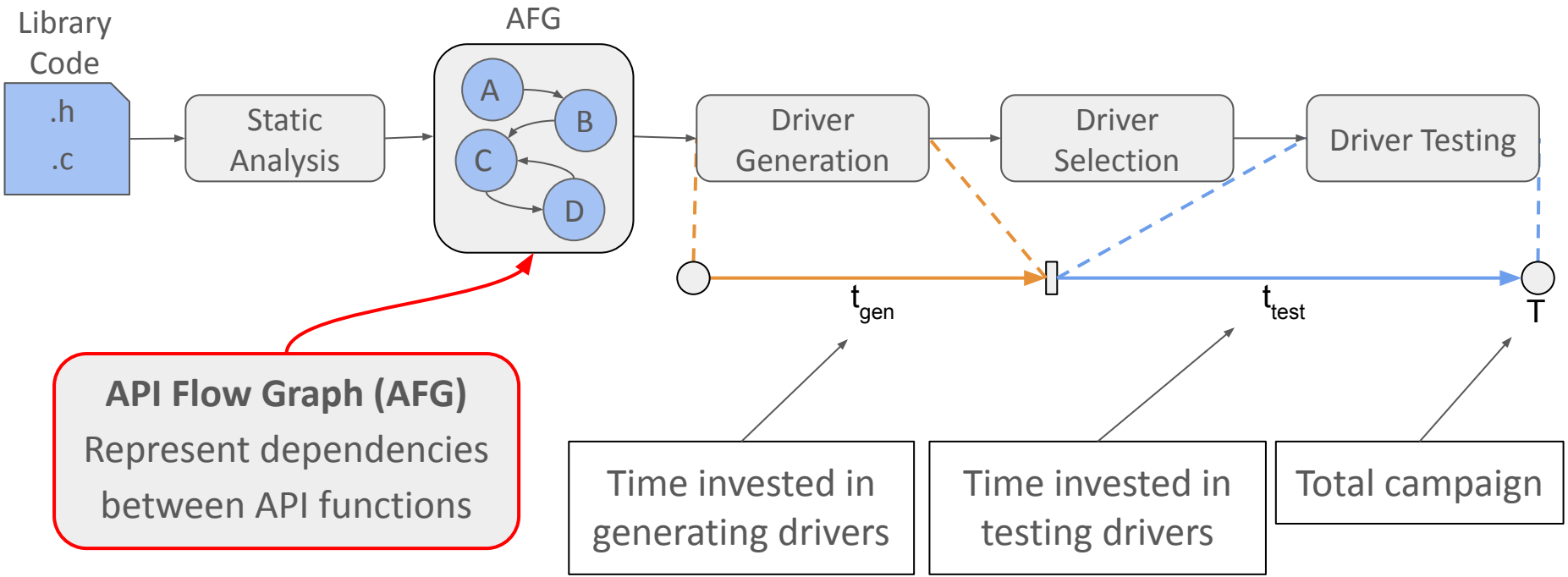
Q1: How much time should I spend to generate new drivers?

Q2: How much time should I spend to test the drivers?



libErator's (Simplified) Design

Fuzzing campaigns have limited time, how do I allocate it efficiently?



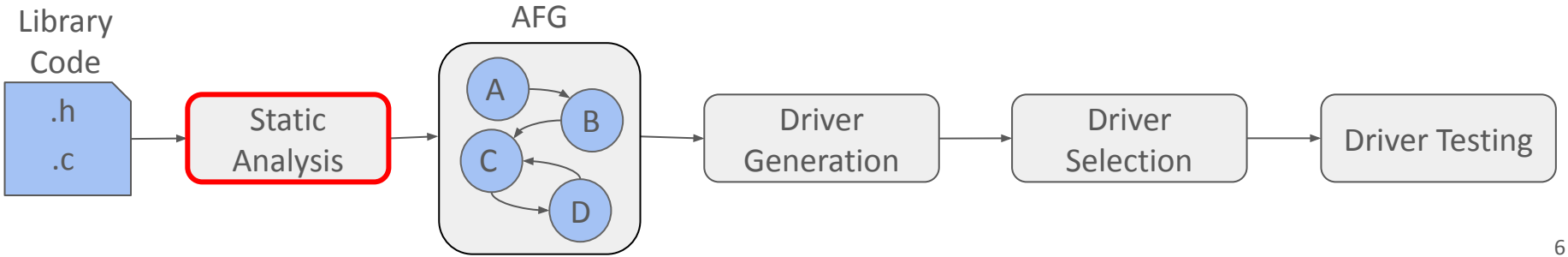
libErator's Design: Static Analysis

Populate the API Flow Graph (AFG)[1]

- Dependencies between API calls

Infers API function arguments dependencies (e.g., buffer and its length)

The type system determines the variables' initialization procedure



[1] More technical info in the paper

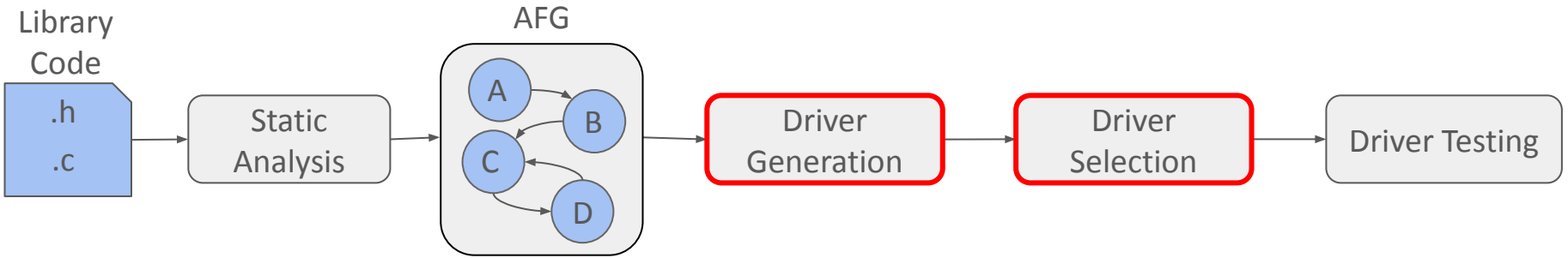
libErator's Design: Driver Generation and Selection

Driver generation through iterative AFG traversal[1]

- Instantiate necessary variable and their cleanup code
- Bias towards function manipulating more argument fields
- Deemed successful if the driver produce seeds

Selection of diverse drivers for deep testing[1]

- Pick one driver per cluster of drivers using similar API functions



[1] More technical info in the paper

Evaluation

We compare against consumer-aware and consumer-agnostic works

- Hopper, CCS'23
- UTopia, S&P'23
- FuzzGen, Usenix SEC'20
- OSS-Fuzz-Gen, Google '23
- Manually written drivers

A benchmark of **15 C libraries**, taken across the above tools

Coverage[1]

libErator vs Consumer-agnostic:
Hopper, 8 / 13 better

libErator vs Consumer-aware:
UTopia, 3 / 6 better
FuzzGen, 1 / 2 better
OSS-Fuzz-Gen, 5 / 6 better

libErator vs Manually Written:
misc. 6 / 12 better

Lesson Learned:
(i) better than SotA consumer-agnostic works
(ii) similar or better against consumer-aware works and manually written drivers
(iii) overall, fit-for-all solution seems missing
Most importantly….

[1] Not all the libraries were compatible with all the competitors

Bugs Found

24 unique bugs identified, including a CVE in libpcap

25% true positive crashes (vs 0.7% for Hopper)

False positives caused by:

- Incoherent arguments (e.g., 2D array)
- Incorrect memory tracking (e.g., UAF)

Bugs were reported and fixed

We also contribute drivers and derived test cases

Tool	Bugs
Manual drivers	0
FuzzGen	0
UTopia	0
OSS-Fuzz-Gen	0
Hopper	**6**
libErator	**24**

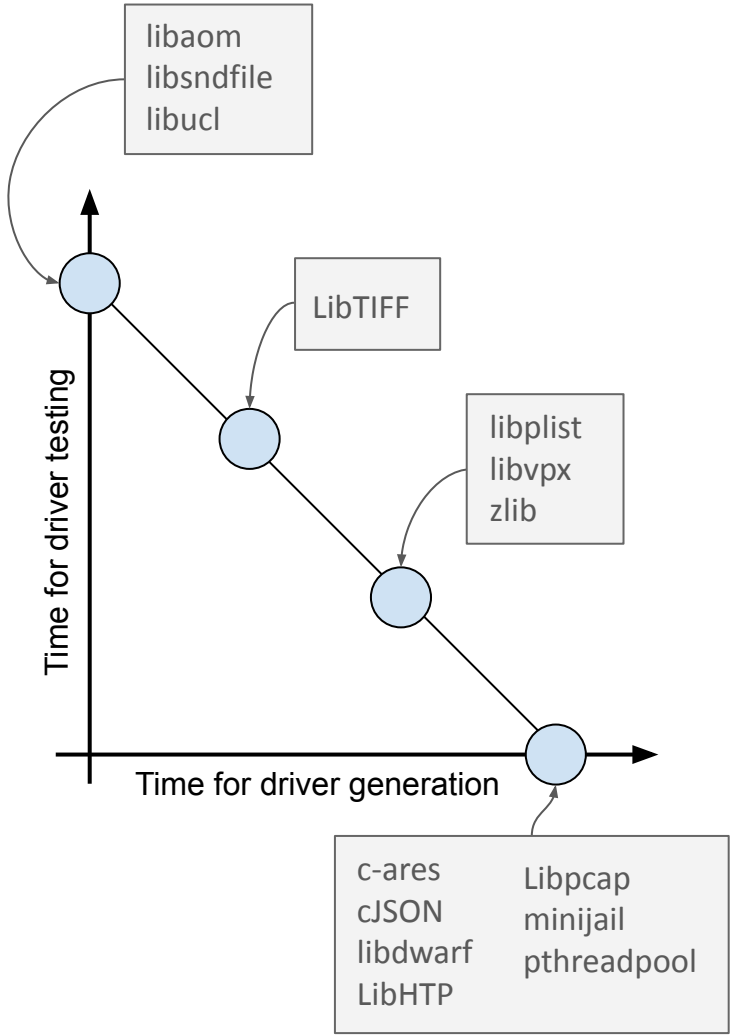
Lesson Learned:
(i) libErator reaches untested library regions and find bugs where
(ii) Consumer-aware and manually written drivers are exhausted.

Lesson Learned

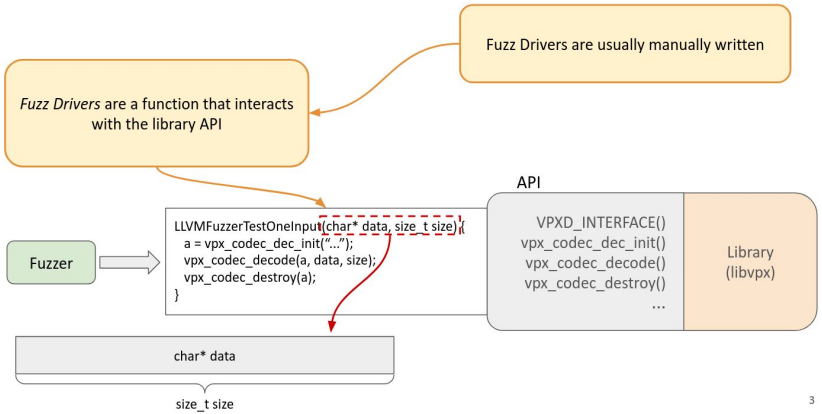
We measured how different values of t_{gen} and t_{test} affect performances in terms of coverage

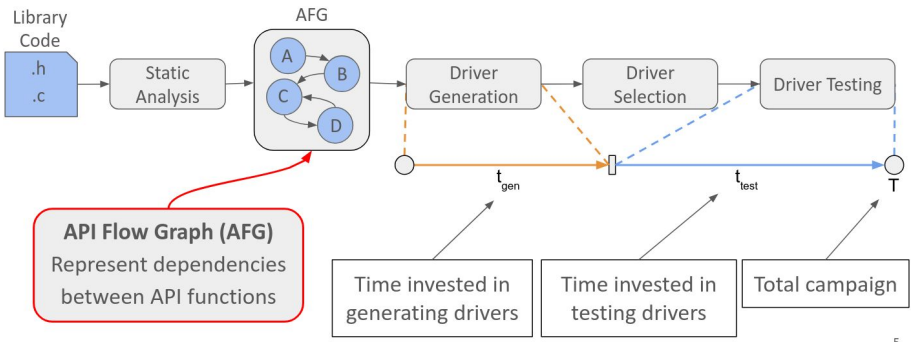
We learned that two aspects affect this tradeoff:

- **Input complexity**: libraries that expect complex inputs requires more testing time for single driver
- **API complexity**: we may need to spend more time in finding the correct library interaction



libErator Summary





Third-party library testing

24 unique bugs identified, including a CVE in libpcap

25% true positive crashes (vs 0.7% for Hopper)

False positives caused by:

- Incoherent arguments (e.g., 2D array)
- Incorrect memory tracking (e.g., UAF)

Bugs were reported and fixed

We also contribute drivers and derived test cases

Tool	Bugs
Manual drivers	0
FuzzGen	0
UTopia	0
OSS-Fuzz-Gen	0
Hopper	**6**
libErator	**24**

Real impacts

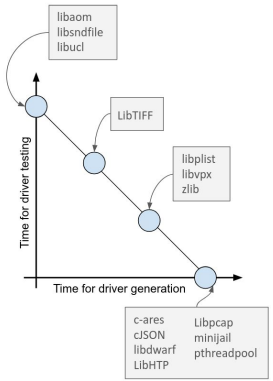
New design to generate drivers

Lesson Learned

We measured how different values of t_{gen} and t_{test} affect performances in terms of coverage

We learned that two aspects affect this tradeoff:

- **Input complexity**: libraries that expect complex inputs requires more testing time for single driver
- **API complexity**: we may need to spend more time in finding the correct library interaction



Different trade-off