From “What The Fuzz?” To “All The Fuzz!”

EPFL

Mathias Payer

hexhive

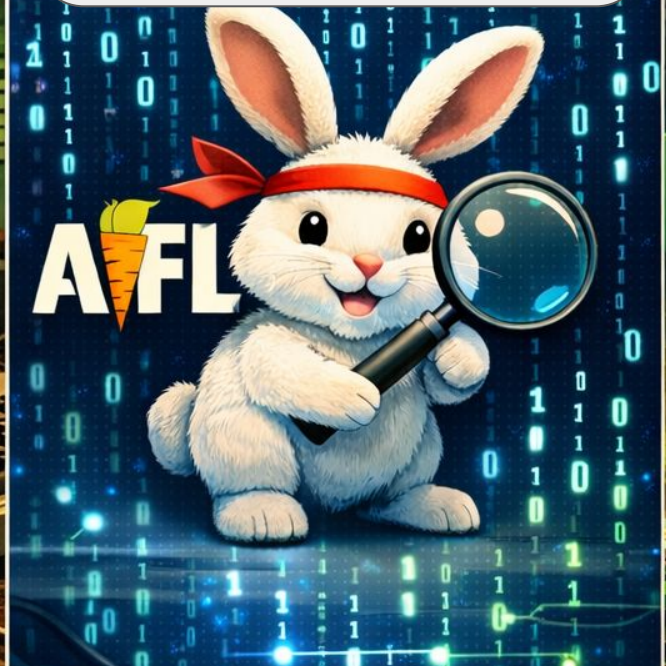
The Origins of Fuzzing

```
10 INPUT A$,
20 POKE 12345, RND(256)
30 PRINT A$
40 GOTO 10

FUZZING.....
CRASH!
```

Fuzzing started as random input mutation

The Greybox Revolution

It became coverage-guided optimization



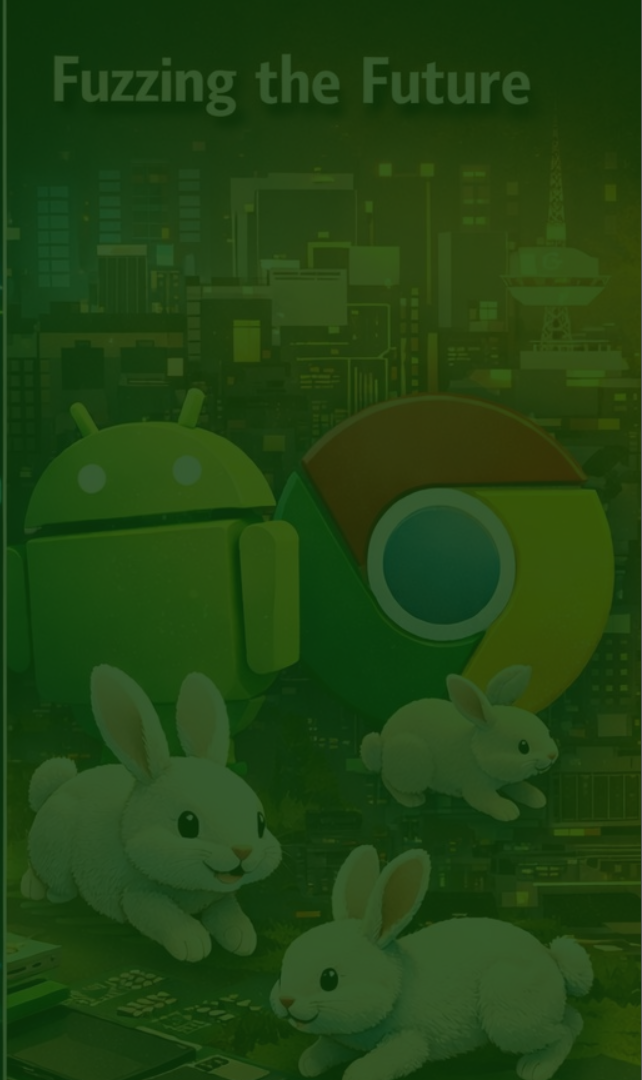
Fuzzing the Future

It is now about modeling *state, semantics, and environment*

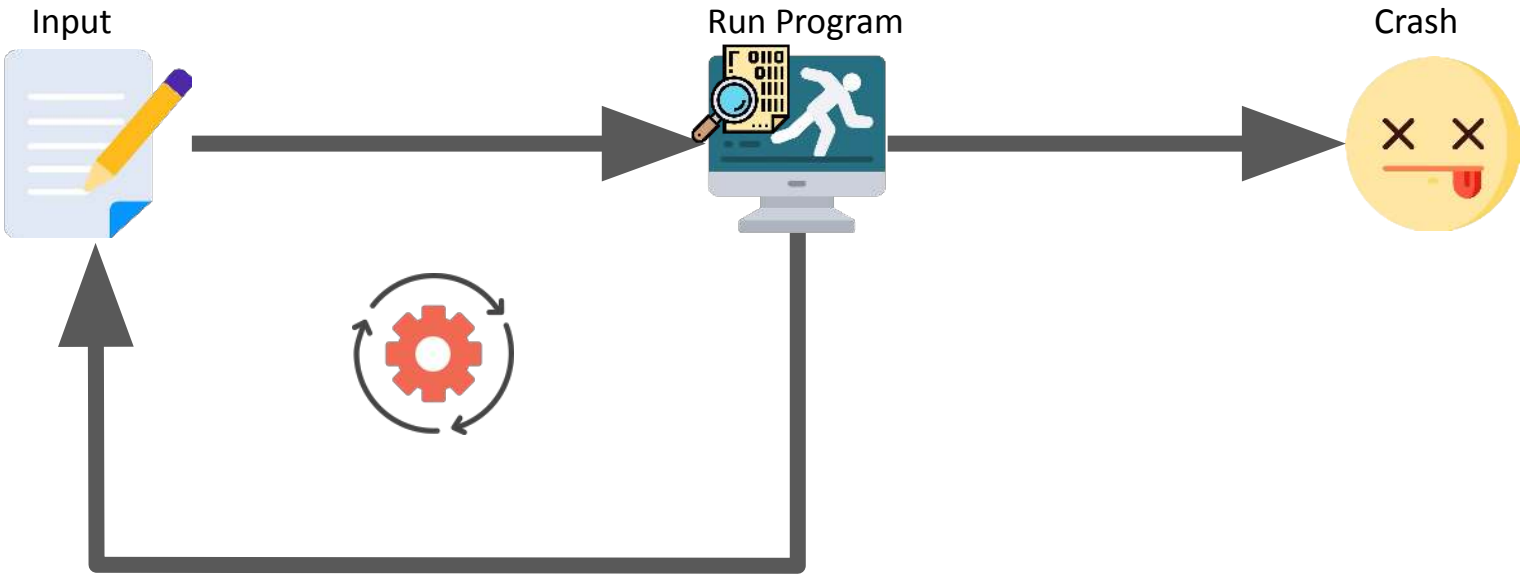
The Origins of Fuzzing

The Greybox Revolution

Fuzzing the Future



Fuzzing: Automated (Fuzz) Testing



The 90s: Birth of Blackbox Fuzzing

Barton Miller’s classic UNIX fuzzing experiments

- Random inputs, surprising crash rates
- No instrumentation, no feedback

Strength: simplicity and generality

Weakness: inefficiency and blind exploration

Lessons from Blackbox Fuzzing

Software robustness overestimated

Diminishing returns after initial low-hanging fruit

Heavy use of domain knowledge

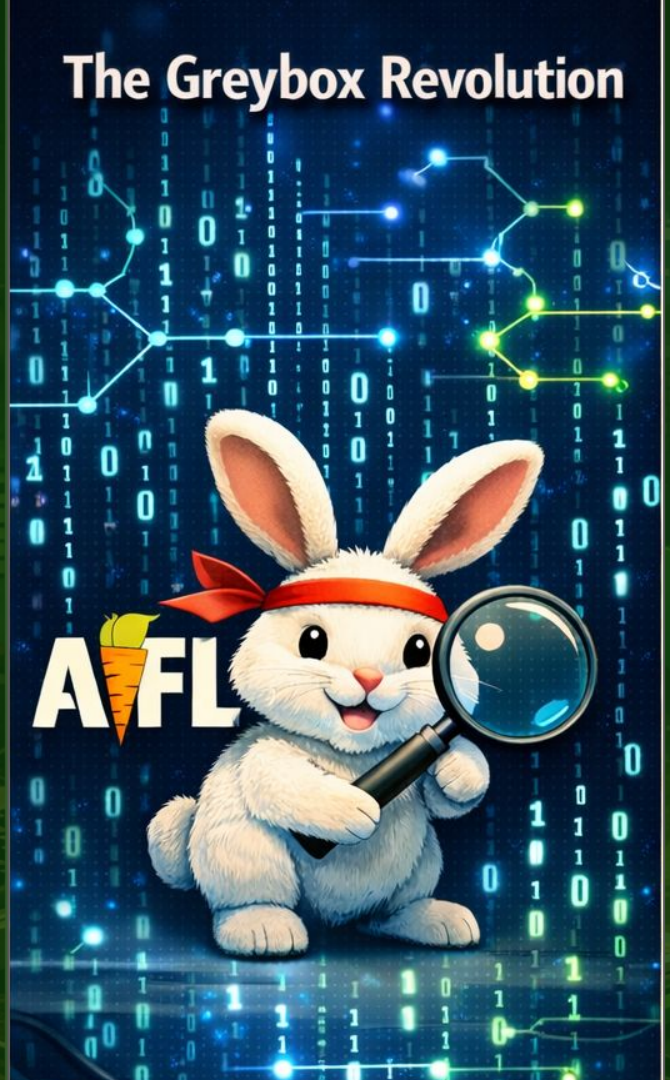
High precision, low generality

We learned that *structure matters*.

The Origins of Fuzzing

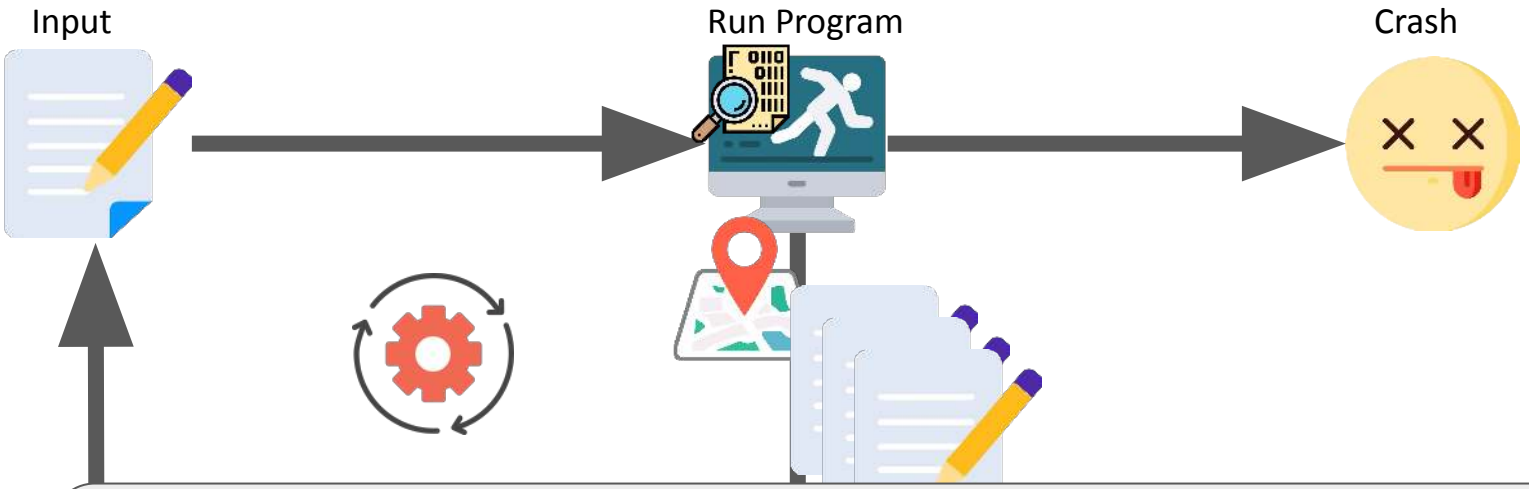
```
10 INPUT A$,
20 POKE 12345, RND(256)
30 PRINT A$
40 GOTO 10

FUZZING.....
CRASH!
```

The Greybox Revolution



Fuzzing the Future

Fuzzing: Automated (Fuzz) Testing



Balance between blackbox and whitebox
Scalability without symbolic execution

AFL: A Transformational Moment

American Fuzzy Lop (AFL)

Edge coverage as feedback signal

Mutation-based input generation

AFL made fuzzing an engineering discipline

Greybox Fuzzers: A Genealogy

2013
2015
2016
2017
2018
2019 (I)
2019 (II)
2020 (I)
2020 (II)
2021
2022

CollAFL
PathAFL
LTL-Fuzz

honggfuzz
syzkaller
Steelix
FOT
Cerebro
pFuzzer
FreeDom
Zeror
Sivo
SNAP
zz

LibFuzzer, AFL++, libAFL, …

LibAFL: the SE community steps up

AFL++/libAFL: modular fuzzing frameworks

Explosion of derivatives and extensions

Fuzzing becomes infrastructure

Solved an enormous class of problems extremely well

What Does Really Matter?

Coverage ≠ bugs

Many bugs hide behind

- State (e.g., in protocols)
- Environment (e.g., for JIT miscompilation)
- Semantics (e.g., for TEE trust-boundaries)

Feedback design and mutation strategy as core challenges

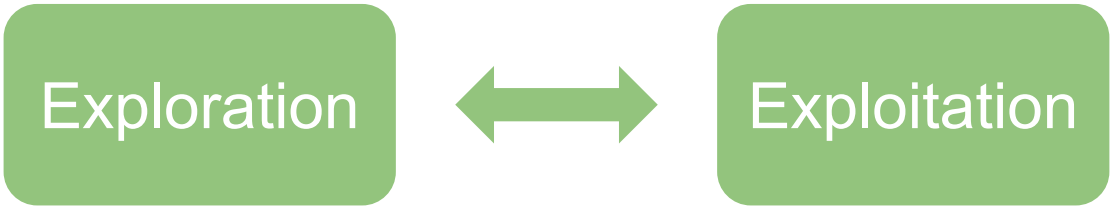
FishFuzz: Separating Exploration and Exploitation

Coverage is ***necessary*** but not ***sufficient*** for bug discovery

- Buggy code must be executed to trigger vulnerability
- Bug requires correct input to trigger

Instead of fuzzing all code equally, focus on likely buggy code?

Goal: direct fuzzer towards “risky” code areas



FishFuzz: Catch Deeper Bugs by Throwing Larger Nets. *Han Zheng, Jiayuan Zhang, Yuhang Huang, Zezhong Ren, He Wang, Chunjie Cao, Yuqing Zhang, Flavio Toffalini, and Mathias Payer.* In SEC'23

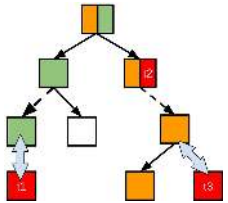
FishFuzz: But is it Worth it?

Queue culling algorithm orchestrates between exploration and exploitation

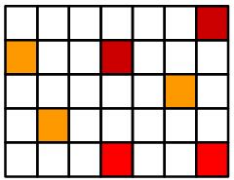
56 unknown bugs discovered and 38 CVEs assigned

Takeaway: coverage is no longer a flat space

Exploration: seed-target distance metric that is independent of target size

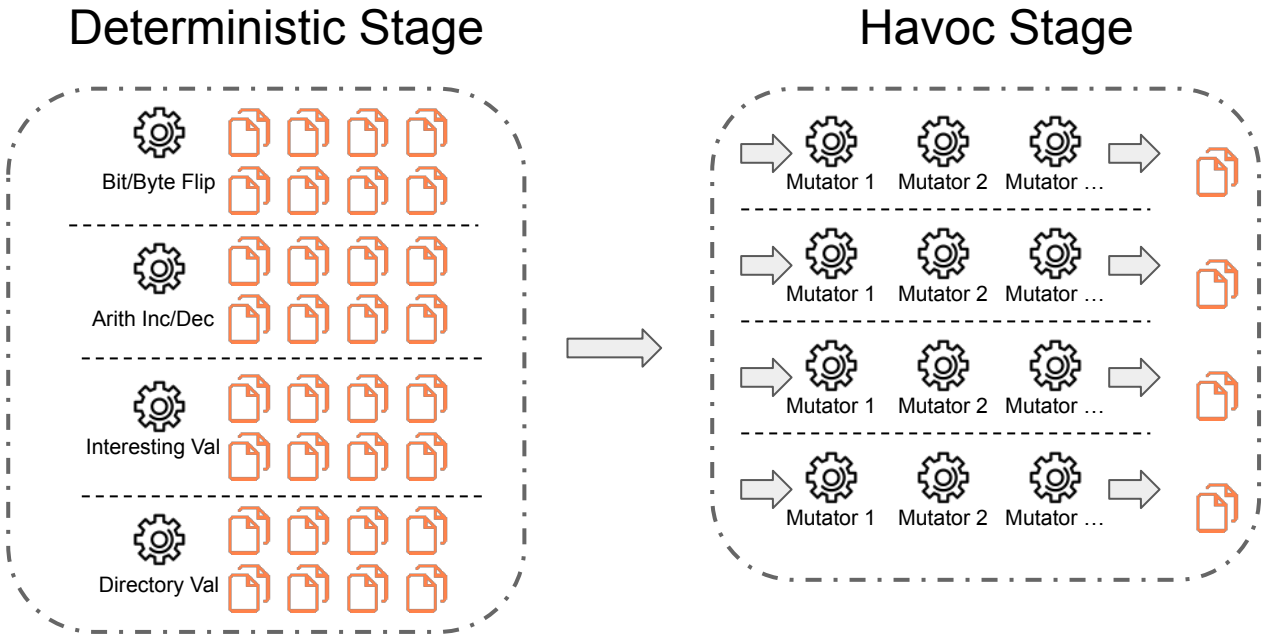


Exploitation: dynamic ranking to direct the fuzzer towards promising locations



FishFuzz: Catch Deeper Bugs by Throwing Larger Nets. *Han Zheng, Jiayuan Zhang, Yuhang Huang, Zezhong Ren, He Wang, Chunjie Cao, Yuqing Zhang, Flavio Toffalini, and Mathias Payer.* In SEC'23

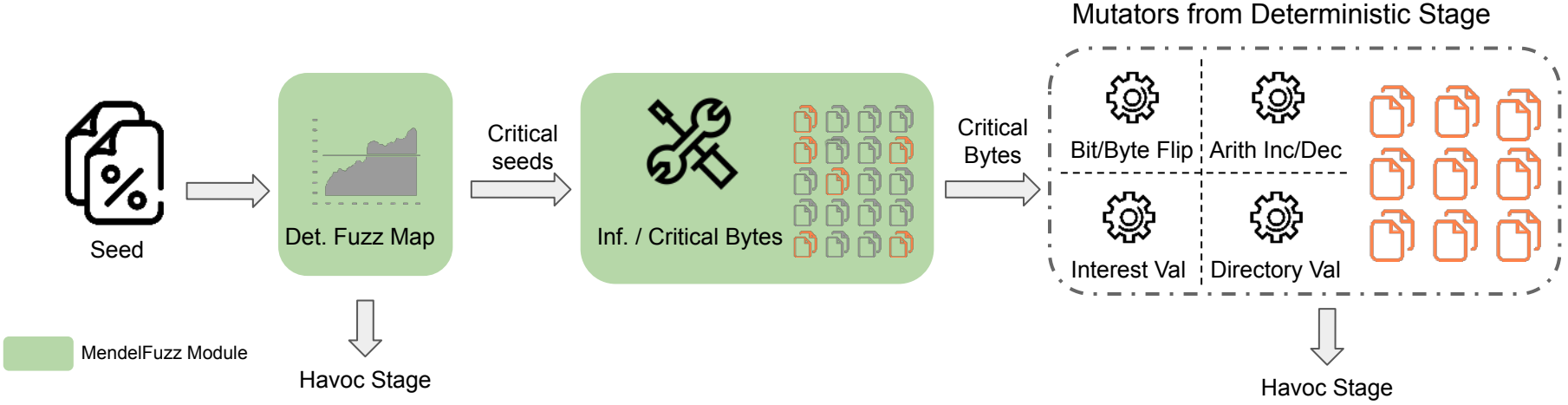
MendelFuzz: Mutation Quality Matters



Deterministic stage is too expensive!

MendelFuzz: Target Critical Bytes

MendelFuzz became the default mode in AFL++!



Takeaway: mutation quality is an information problem

Improving “Signal” By Tracking State

Path coverage disambiguates state

State variables give insight into higher-level state

Data flow abstracts state from code

Feedback design as a core challenge

Different Bugs Require Different Oracles

No source → Yolo

Memory safety → use Address Sanitizer

Concurrency issues → use Thread Sanitizer

Type safety → use a Type Sanitizer

Logic bugs → Good luck!

How to adapt oracles to bug types?

Oracles: Knowing When You Win!

Crashes are only one signal

Silent misbehavior

Differential testing

Invariant violations

How to infer specifications and more general oracles?

Evaluating Fuzzing Improvements

Code Coverage

is subjective

Crashes

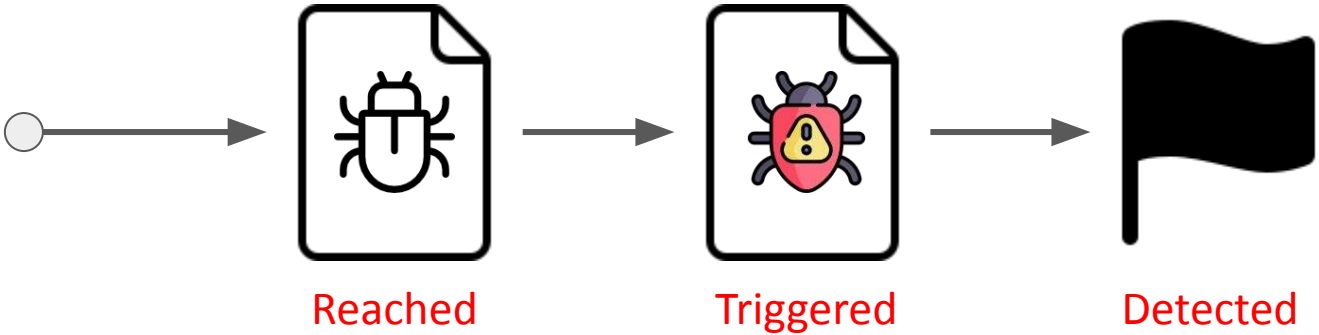
are imprecise

Bugs

lack ground truth

We optimized fuzzers for metrics that are not aligned with vulnerability discovery

Evaluating Fuzzing Improvements



Distinguish different capabilities

Magma's Community Impact

Cited 303 times

Used in 45+ papers across SE, SEC, SYS, PL

109 forks, 325 stars on GitHub

79 PRs for new targets, features, and fixes

Challenge: How to keep benchmarks "alive" and current?

Open-source at:
https://hexhive.epfl.ch/magma/

Year	Venue	Paper	Task	Category
2021	ISSTA	20	Seed Selection	Fuzzing
2022	CCS	59	Directed Fuzzing	Fuzzing
2022	NDSS	24	Mutation Scheduling	Fuzzing
2022	ACSAC	61	Directed Fuzzing	Fuzzing
2022	ACSAC	29	Mutation Scheduling	Fuzzing
2022	ASIACCS	9	Directed Fuzzing	Fuzzing
2022	ISSTA	46	Coverage Feedback	Fuzzing
2023	ICSE	32	Mutation Scheduling	Fuzzing
2023	TOSEM	21	Coverage Feedback	Fuzzing
2023	SBFT	28	Fuzzing Deployment	Deployment
2023	ISSTA	68	Directed Fuzzing	Fuzzing
2023	OOPSLA	35	Coverage Feedback	Fuzzing
2023	NDSS	73	Byte Selection and Mutation	Fuzzing
2023	Security	11	Parallel Fuzzing	Acceleration
2023	CCS	40	Input Generation	Fuzzing
2023	CCS	74	Fuzzing Acceleration	Acceleration
2023	ASE	47	Fuzzing for Program Analysis	Others
2024	ASIACCS	42	Seed Selection	Fuzzing
2024	CCS	60	Seed Selection and Mutation	Fuzzing
2024	TOSEM	65	Seed Selection	Fuzzing
2024	TOSEM	63	Coverage Feedback	Fuzzing
2024	TOSEM	54	Seed Selection	Fuzzing
2024	S&P	22	Directed Fuzzing	Fuzzing
2024	S&P	23	Directed Fuzzing	Fuzzing
2024	Security	34	Directed Fuzzing	Fuzzing
2024	Security	57	Directed Fuzzing	Fuzzing
2024	ISSTA	31	Mutation Scheduling	Fuzzing
2024	ISSTA	14	Directed Fuzzing	Fuzzing
2024	ASPLOS	37	Bug Sanitization	Fuzzing
2024	ASPLOS	71	Program Transformation	Others
2024	TSE	79	Seed Selection and Mutation	Fuzzing
2025	TOSEM	53	Coverage Feedback	Fuzzing
2025	TOSEM	36	Coverage Feedback	Fuzzing
2025	Security	72	Input Generation	Fuzzing
2025	Security	64	Coverage Feedback	Fuzzing
2025	Security	4	Directed Fuzzing	Fuzzing
2025	EuroS&P	17	Directed Fuzzing	Fuzzing
2025	EuroS&P	69	Coverage Feedback	Fuzing
2025	ISSTA	70	Modular-Based Fuzzing	Implementation
2025	ISSTA	76	Parallel Fuzzing	Acceleration
2025	FSE	75	Mutation Scheduling	Fuzzing
2025	FSE	26	Crash Deduplication	Post-Fuzzing
2025	ICSE	10	Directed Fuzzing	Fuzzing
2025	ICSE	30	Fuzzing for Backdoor Detection	Others
2025	ICSE	62	Bug Sanitization	Fuzzing

Open Greybox Challenges

Hybrid Fuzzing: How to improve feedback?

Understanding the target beyond CFGs

State explosion and path explosion: deep states

Scheduling and resource allocation

Steady stream of continuous improvement for greybox fuzzing

Lessons From Greybox Fuzzing

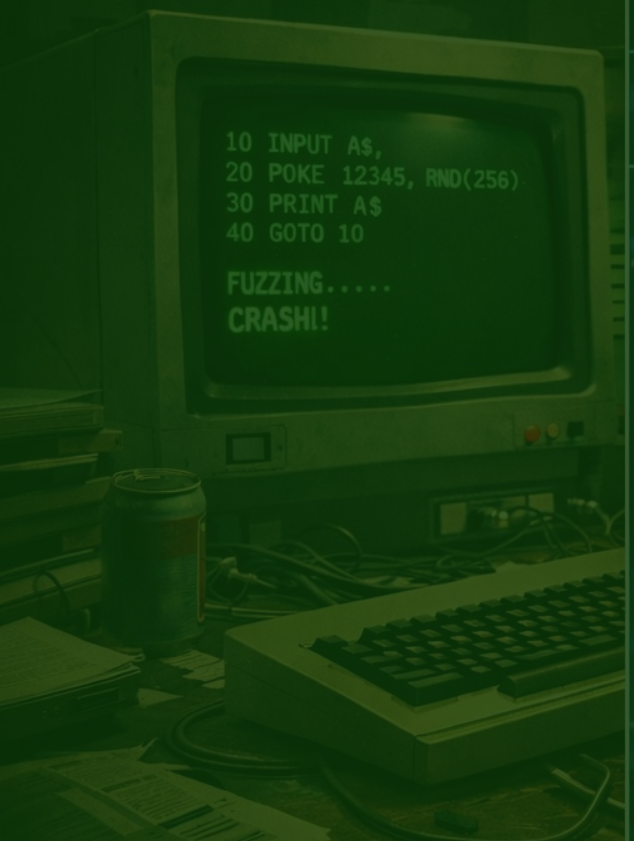
Fuzzing is no longer just about coverage

Mutation is no longer blind

Feedback, generation, and statefulness are key

Natural transition to domain-specific fuzzing

The Origins of Fuzzing



The Greybox Revolution

Fuzzing the Future

“Fuzzing Niches” Matter and are Key!

General-purpose fuzzers hit diminishing returns

Real-world systems are heterogeneous

Environment matters as much as code

Shift: not *what* inputs, but *where* and *how* to interact

Niche 1: Embedded Systems and Firmwares

Limited observability

Hardware dependencies

Emulation vs real devices

Increasing relevance for IoT security

How to rehost and emulate?

Niche 2: Browser Fuzzing

Huge attack surface

Massive codebases

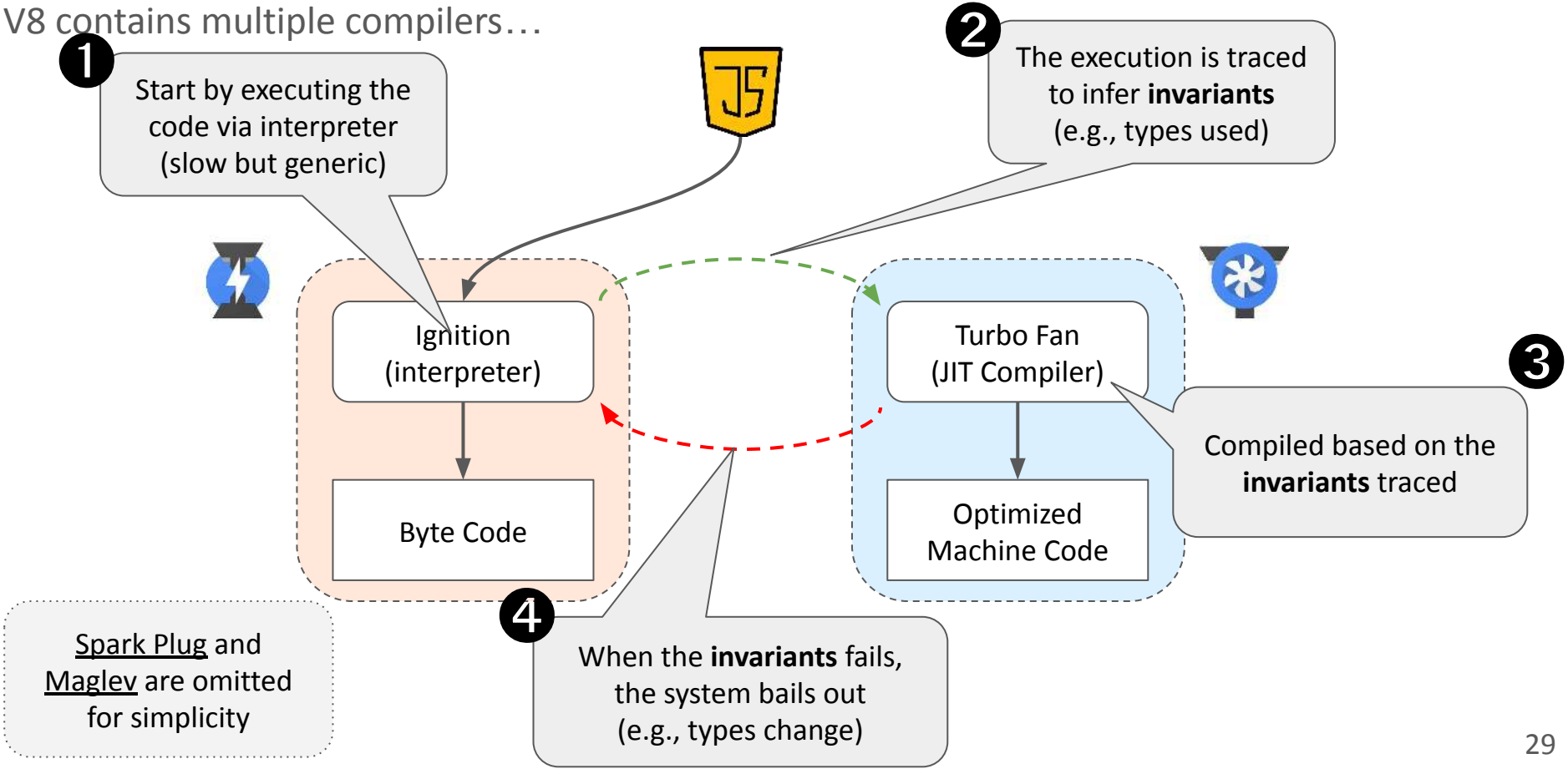
Rich, structured inputs

Deep semantic expectations

Browsers are hostile to fuzzing

How to Fuzz Complex Browser Engines?

V8 contains multiple compilers…

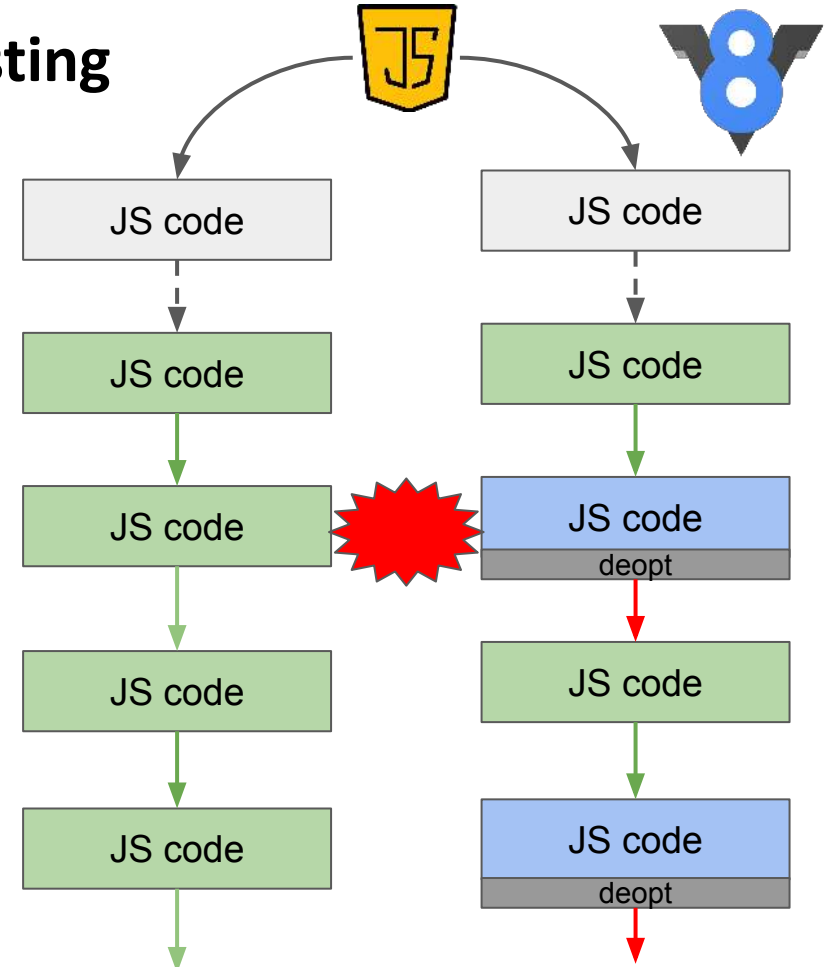


DUMPLING: Differential JS engine testing

Intuition: dump JS frames in key locations, observe frame divergences

Challenges:

- Where to **dump**?
- How to **dump frames?**
- How to compare **frames**?



DUMPLING: Fine-grained Differential JavaScript Engine Fuzzing. *Liam Wachter, Julian Gremminger, Christian Wressnegger, Mathias Payer, and Flavio Toffalini.* In NDSS'25 (***Distinguished Paper Award***)

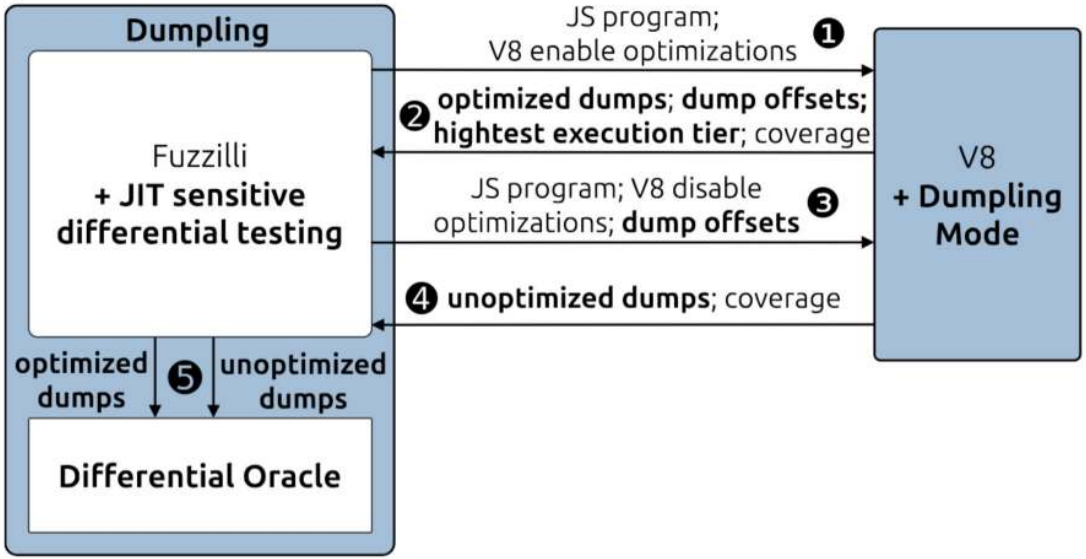
DUMPLING: Differential Browser Snapshot Analysis

Bugs in compiler optimizations lead to exploitable bugs

Key idea: compare memory dumps of baseline/optimized executions

Result: 8 new V8 bugs

Upstreamed and integrated into Google's browser test suite



DUMPLING: Fine-grained Differential JavaScript Engine Fuzzing. *Liam Wachter, Julian Gremminger, Christian Wressnegger, Mathias Payer, and Flavio Toffalini.* In NDSS'25 (***Distinguished Paper Award***)

Niche 3: Android Environment

Huge attack surface

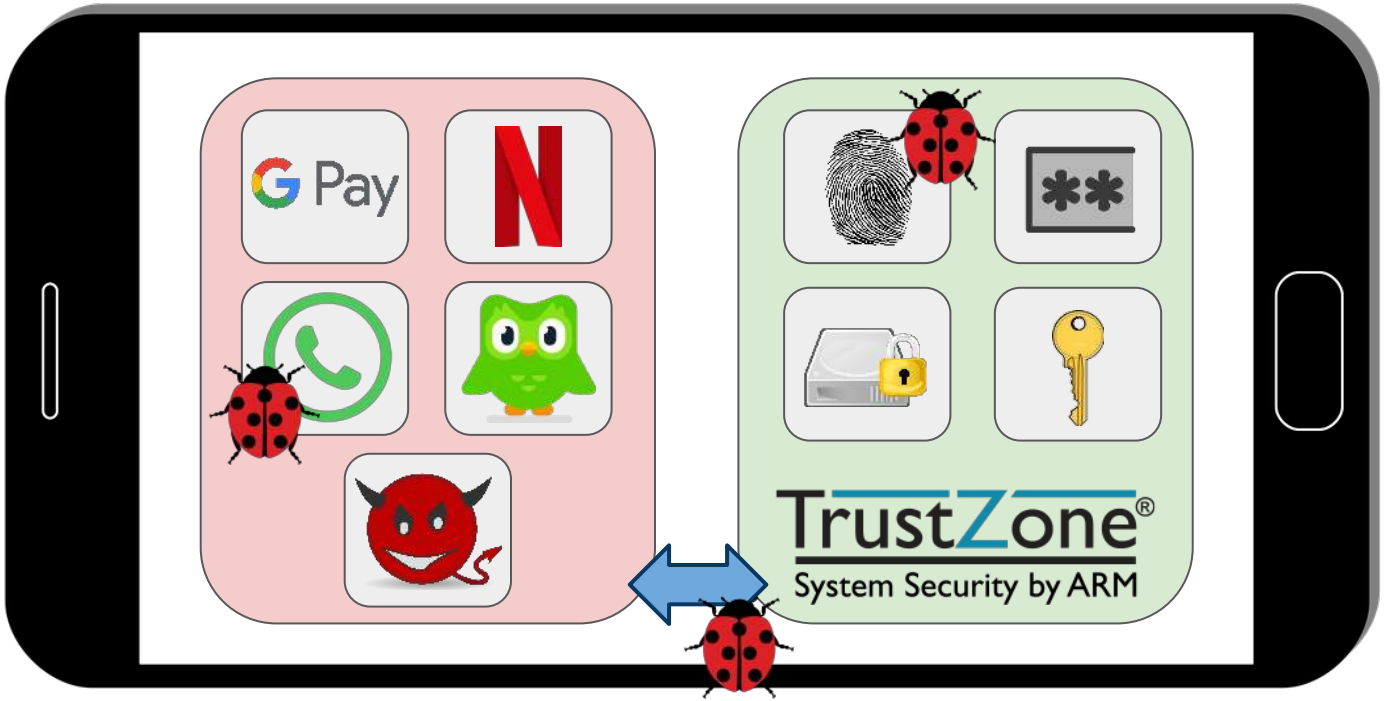
Massive codebases

Rich, structured inputs

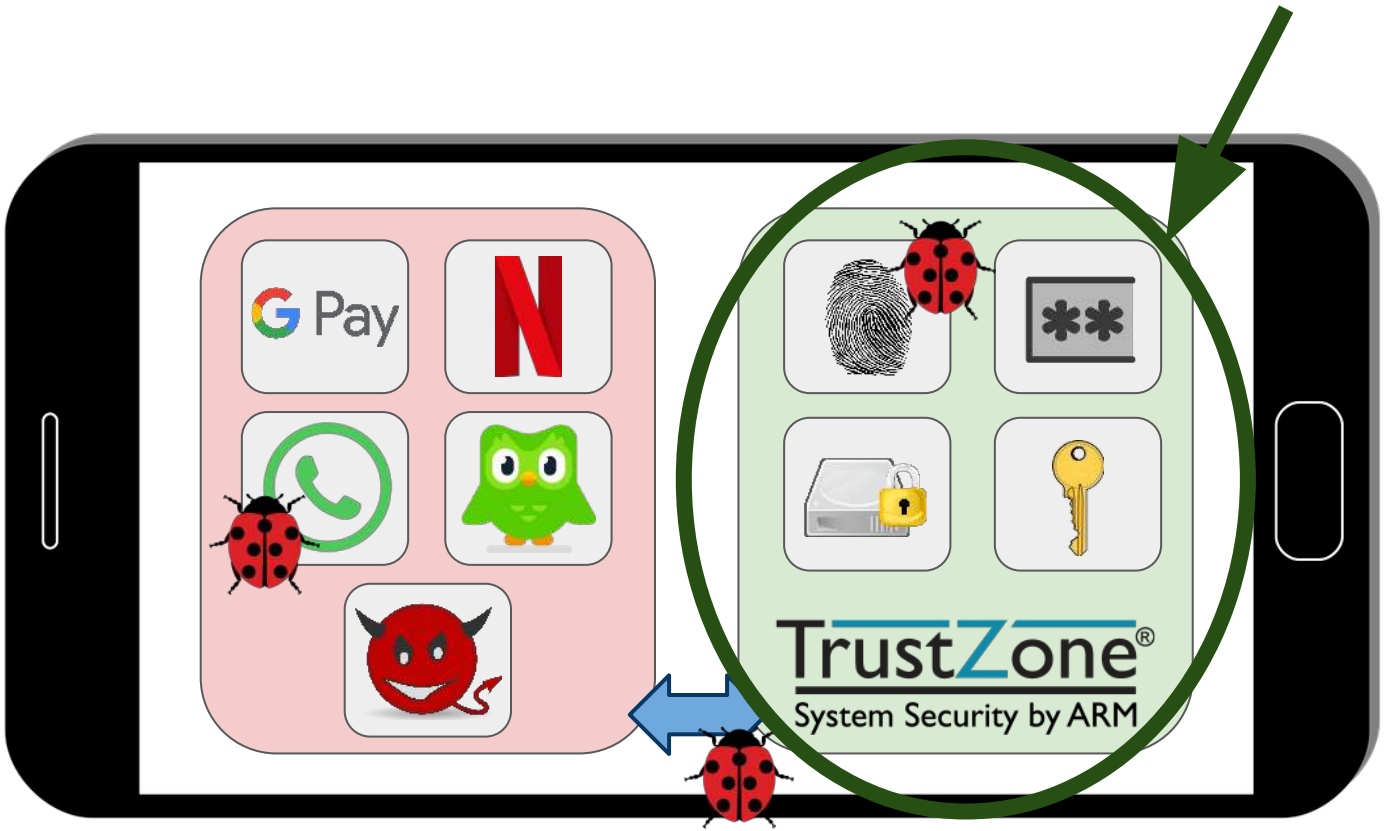
Deep semantic expectations

Android has many components and trust boundaries

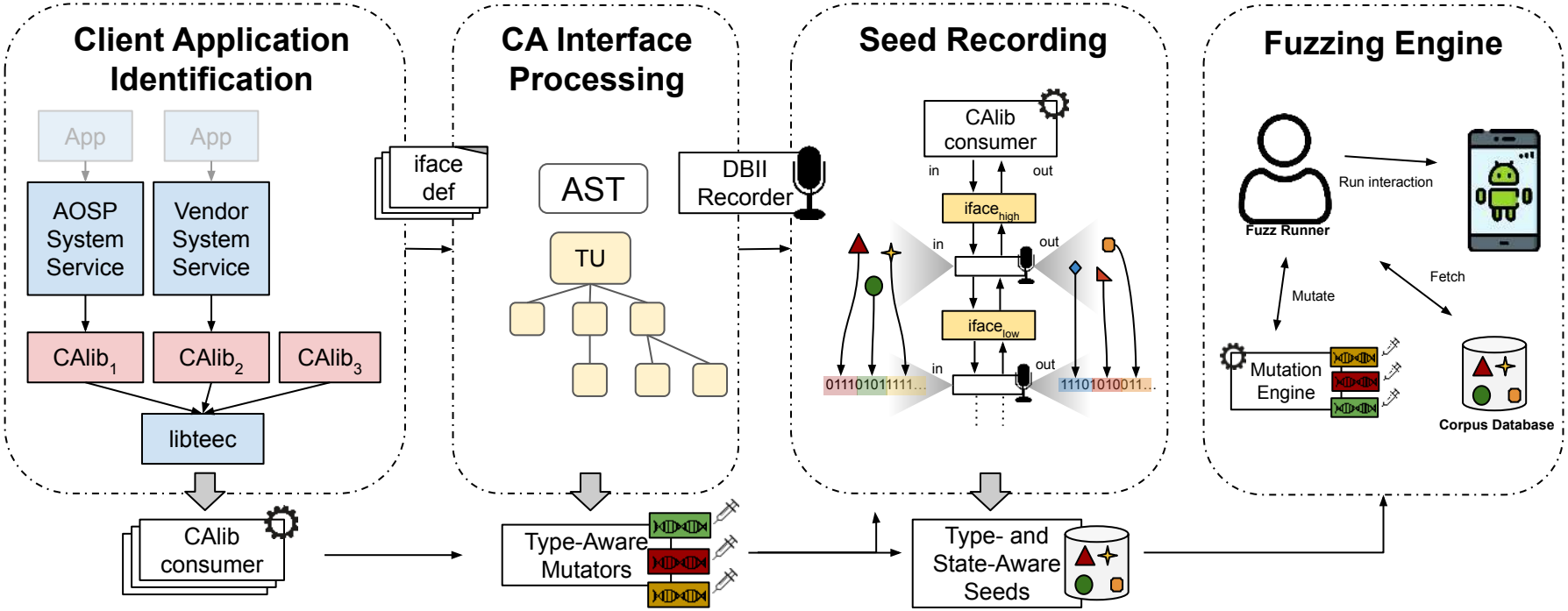
Android Architecture Overview



Fuzzing Trusted Applications



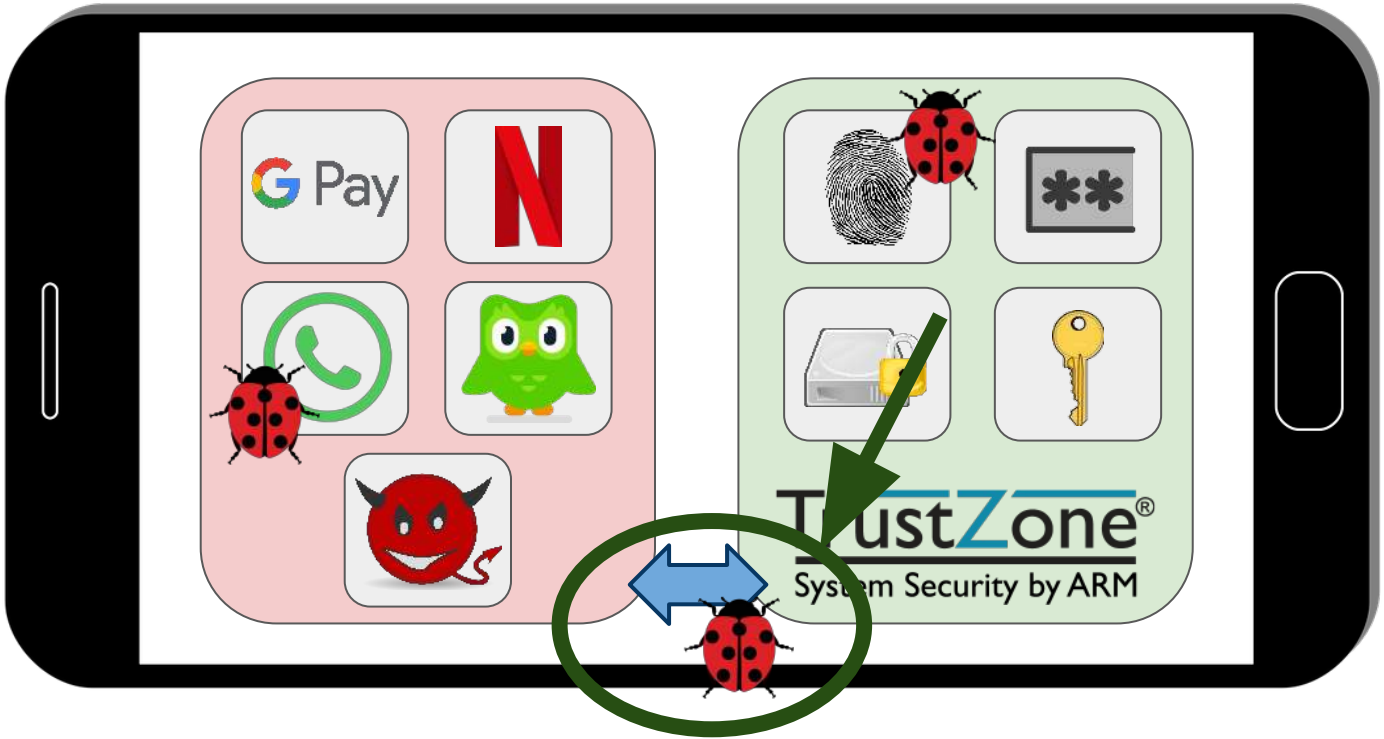
TEEzz Fuzzing Pipeline: Stateful Interface Fuzzing



TEEzz: Fuzzing Trusted Applications on COTS Android Devices.

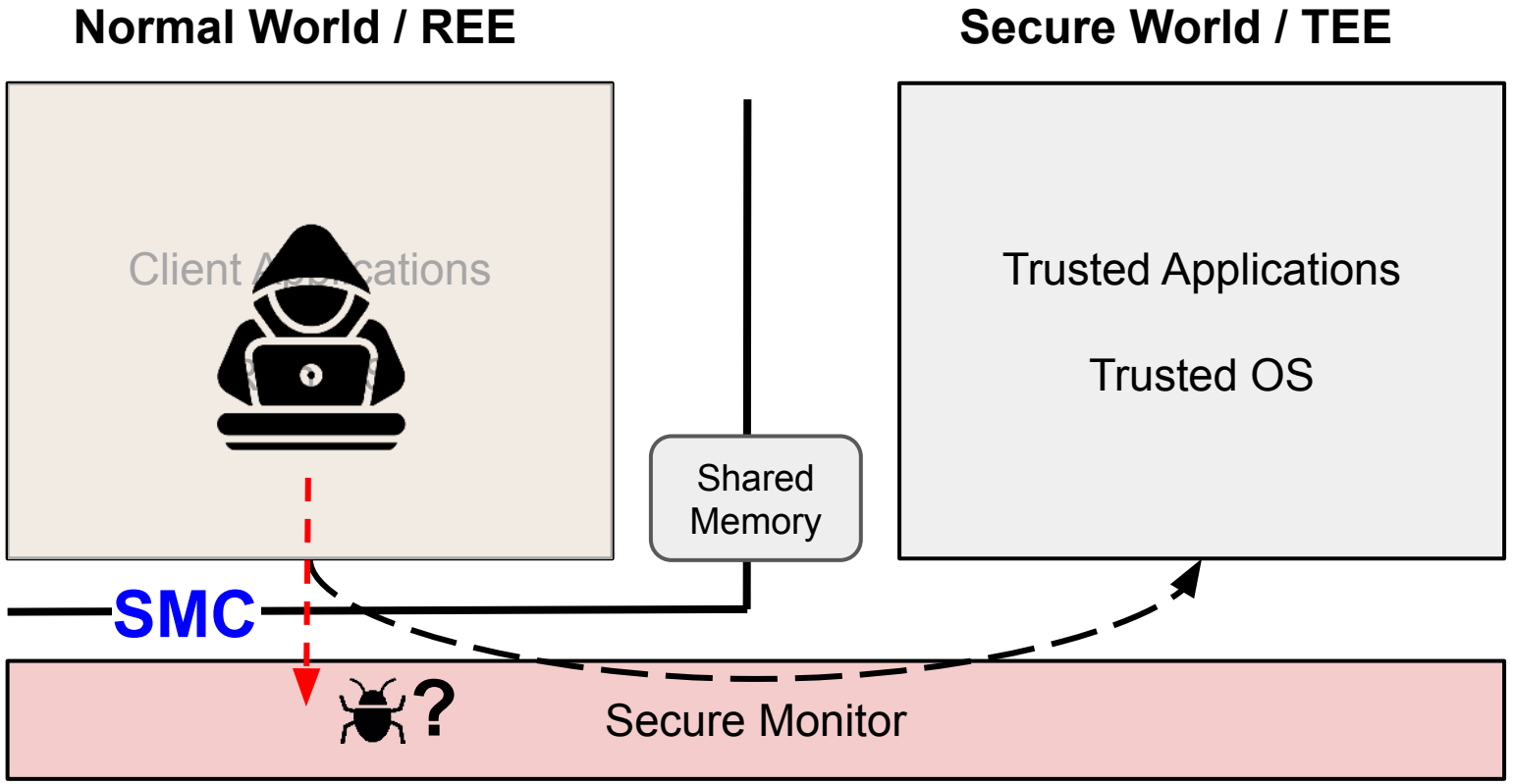
Marcel Busch, Mathias Payer, Aravind Machiry, Christopher Kruegel, Giovanni Vigna, and Chad Spensky. In Oakland'23

EL3XIR: Fuzzing the Most Trusted Layer 🧙

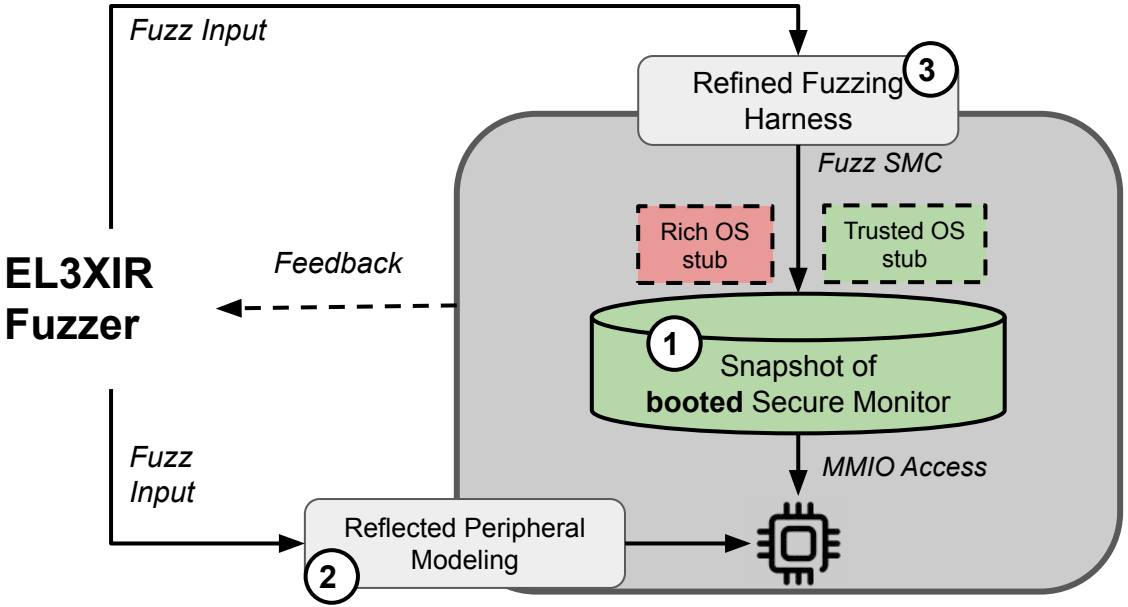


ARMv8-A TrustZone

arm TRUSTZONE



EL3XIR: Fuzzing COTS Secure Monitors



Rehosting Framework for proprietary TrustZone Firmware

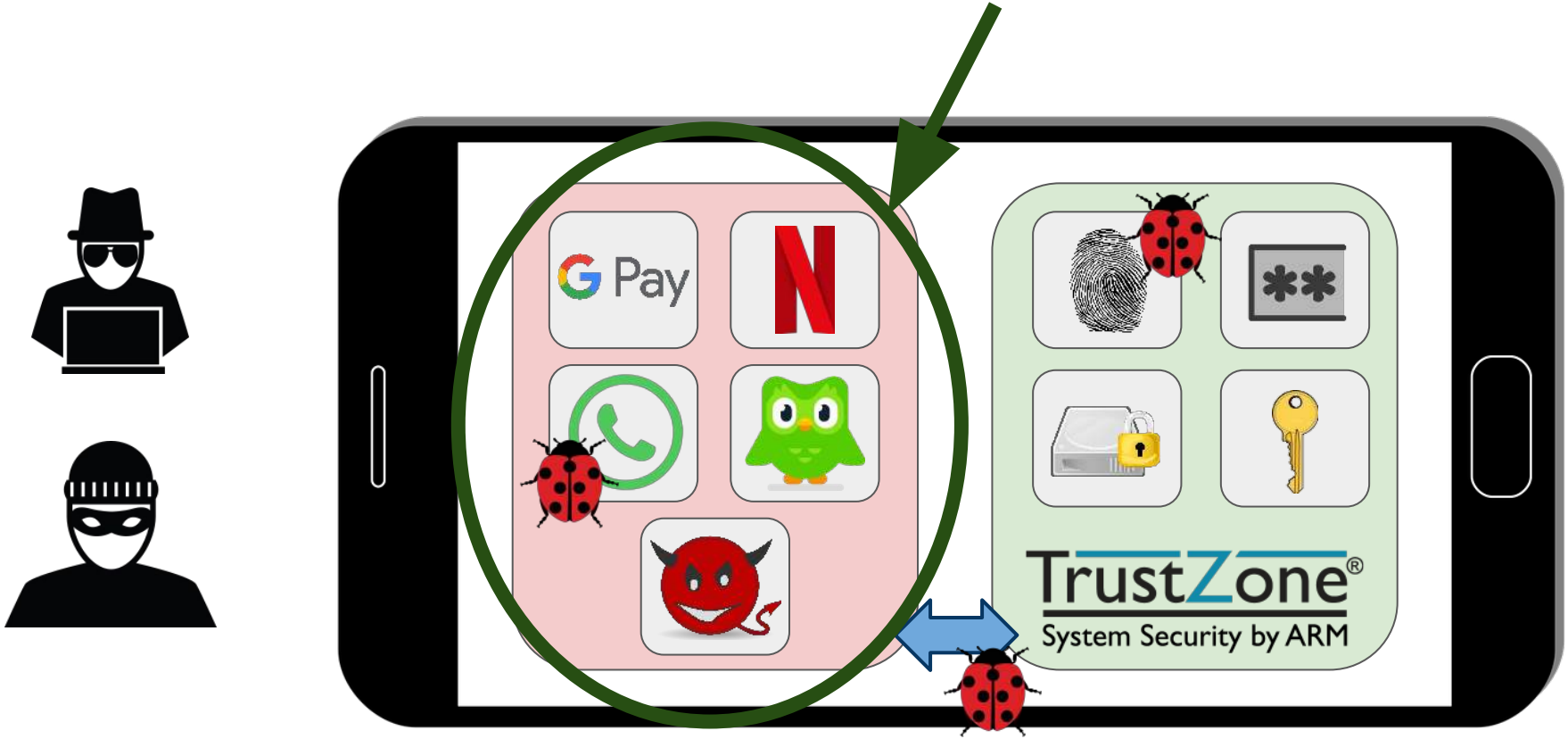
Highly automated Fuzzing Pipeline including Harness Synthesis and Peripheral Modeling

Fuzz your own Secure Monitor

github.com/HexHive/EL3XIR

EL3XIR: Fuzzing COTS Secure Monitors.
Christian Lindenmeier, Mathias Payer, and Marcel Busch. In SEC'24

How Privileged Services Become a Target 📚🔥



NASS: Fuzzing Native Android System Services

System services are exposed through binder interface

Kernel binder module rewritten in Rust, exploitation focuses to C++ services

We introduce Deserialization-Guided Interface Extraction, recovering interface grammars

- Our prototype fuzzer targets on-device system services
- We get coverage through binary rewriting
- 12 vulnerabilities, 5 CVEs assigned by Google
- https://github.com/HexHive/NASS

NASS: Fuzzing All Native Android System Services with Interface Awareness and Coverage.
Philipp Mao, Marcel Busch, and Mathias Payer. In Usenix SEC'25

Open Fuzzing Challenges

Security-relevant state outside traditional coverage

Model stateful interactions across trust boundaries

Program state modeling

Environment-aware vulnerability discovery

Make the implicit state explicit

From What (The Fuzz) to Where (To Fuzz)

The future of fuzzing is selective

CFGs → state machines → system interactions

Integrating semantics, state, and environments

Must choose *right* abstraction level

hexhive

EPFL

Join us on this research journey!

The Origins of Fuzzing

```
10 INPUT A$,
20 POKE 12345, RND(256)
30 PRINT A$
40 GOTO 10

FUZZING.....
CRASH!
```

Fuzzing has matured as a field

The Greybox Revolution

Coverage-guided fuzzing was a massive success

Fuzzing the Future

The next breakthroughs are contextual and semantic

EPFL

Mathias Payer

hexhive