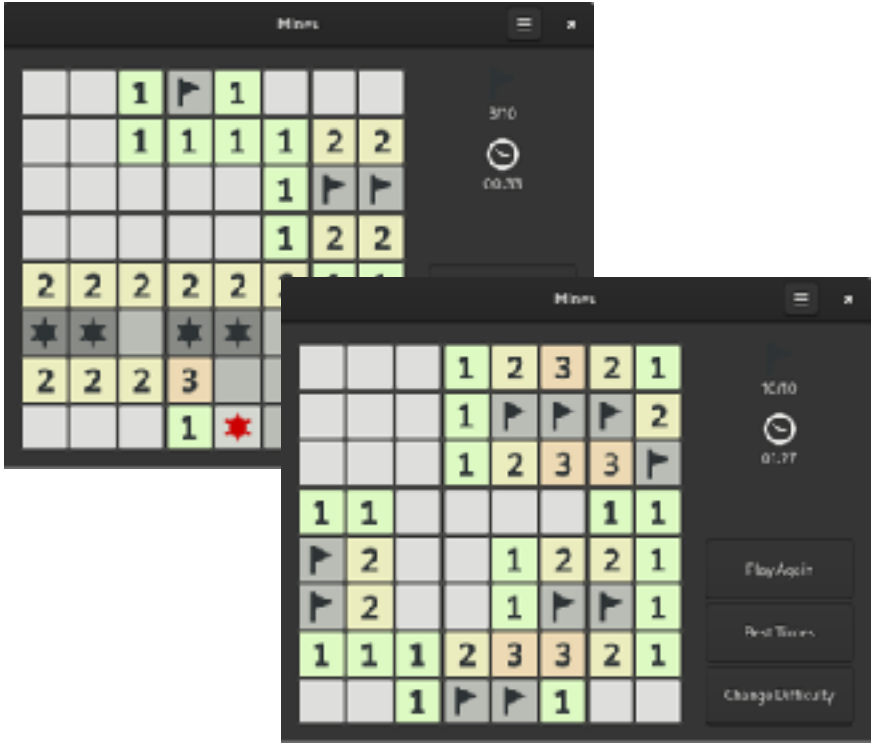
Why fuzz about security?



Mathias Payer
infosec.exchange/@gannimo

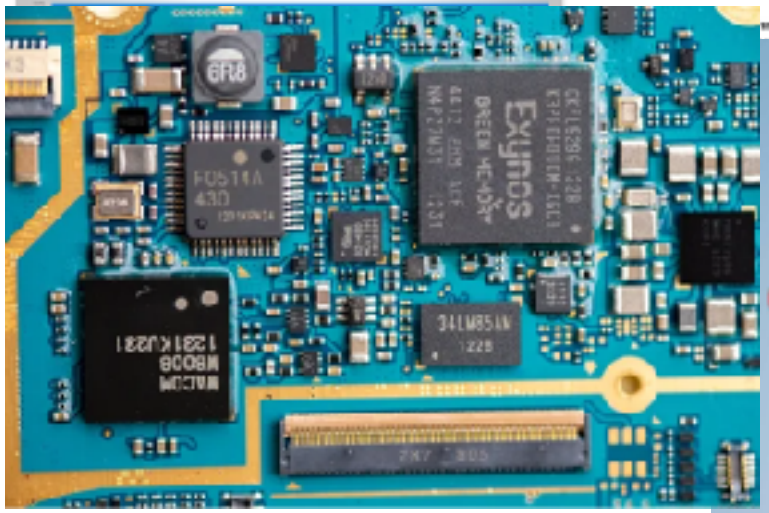
EPFL

hexhive

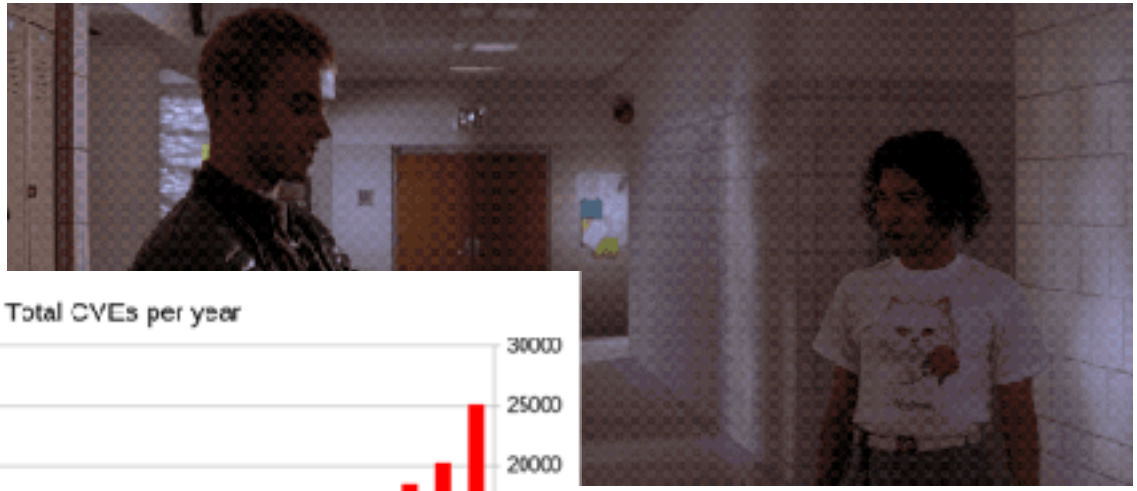
Fact 1: Software Has Bugs

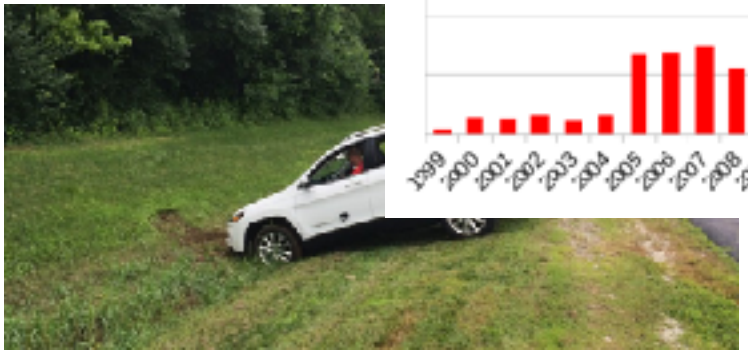
Microsoft: 70 percent of all security bugs are memory safety issues

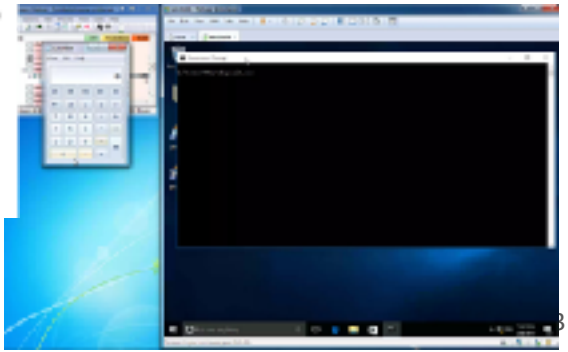
Percentage of memory safety issues has been hovering at 70 percent for the past 12 years.



Fact 2: Many Bugs are Exploitable







Fact 3: Software is Incredibly Complex

Google Chrome: 76 MLoC

Gnome: 9 MLoC

Xorg/Wayland: 1 MLoC

glibc: 2 MLoC

Linux kernel: 17 MLoC

Margaret Hamilton with code for Apollo Guidance Computer (NASA, ‘69)

Software Security: Key Research Questions

- RQ1: ***Efficiently*** detect security violations
- RQ2: ***Automatically*** generate test cases
- RQ3: ***Scale*** testing to ***large*** source repositories
- RQ4: ***Effectively*** test complex interfaces
- RQ5: ***Mitigation*** co-design based on feedback

```
vuln("ABC");
```

```
vuln("AAAABBBB");
```

```
strcpy_chk(buf, 4, str);
```

C/C++

```
void log(int a) {
  printf("A: %d", a);
}

void vuln(char *str) {
  char *buf[4];
  void (*fun)(int) = &log;
  strcpy(buf, str);

  fun(15);
}
```

```
CHECK(fun, tgtSet);
```

hexhive

Software Testing

- Goal: prune bugs
- A tool for developers

Mitigation

- Goal: stop exploitation
- Last line of defense

Compartments

- Goal: least privilege
- Divide & conquer security

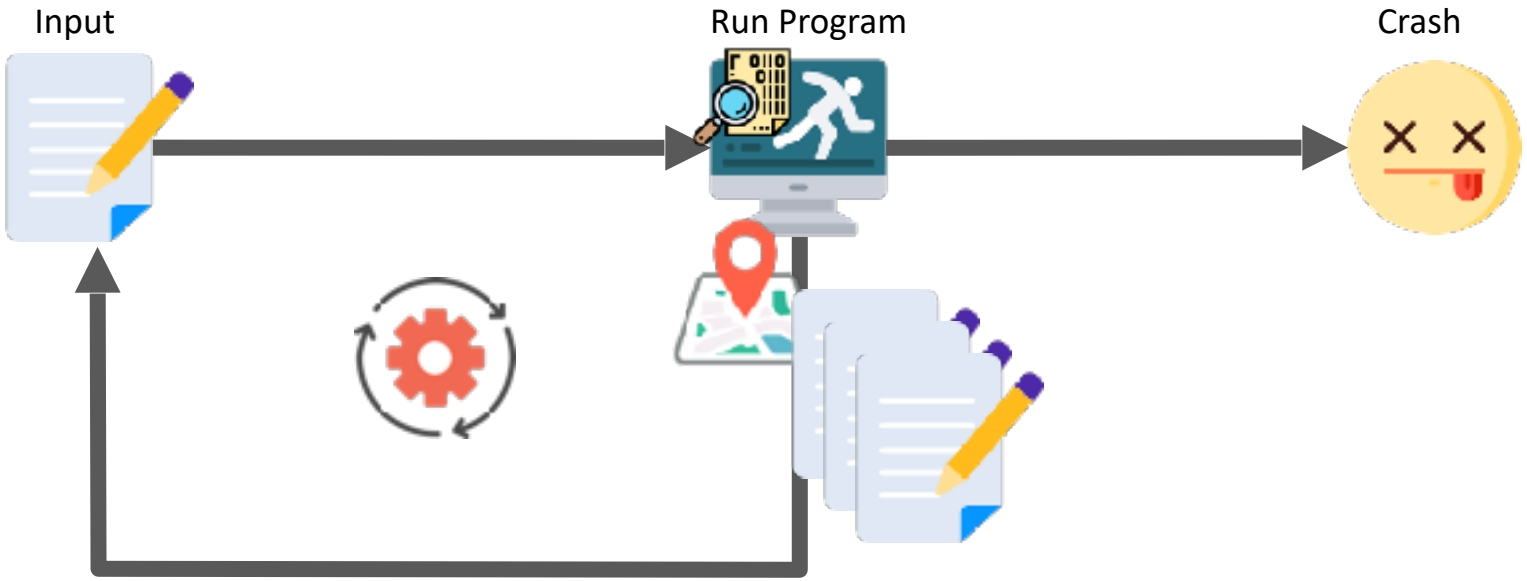
Fuzzing in a Nutshell

```
$ ./testme --help
Usage: testme <int32_arg>

$ ./testme AAAA
Please enter an integer!

$ cat fuzzer.sh
while :
do
  len=$(($RANDOM % 255))
  input="$(dd if=/dev/urandom bs=$len count=1)"
  ./testme $input || echo $input >> crash_seeds
done
```

Fuzzing: Automated (Fuzz) Testing



Test cases must ***reach bugs***

- Exploration through coverage-guided fuzzing

The fuzzer must ***detect bugs***

- Exploitation through sanitization and triaging

Performance is key (zero sum game)!

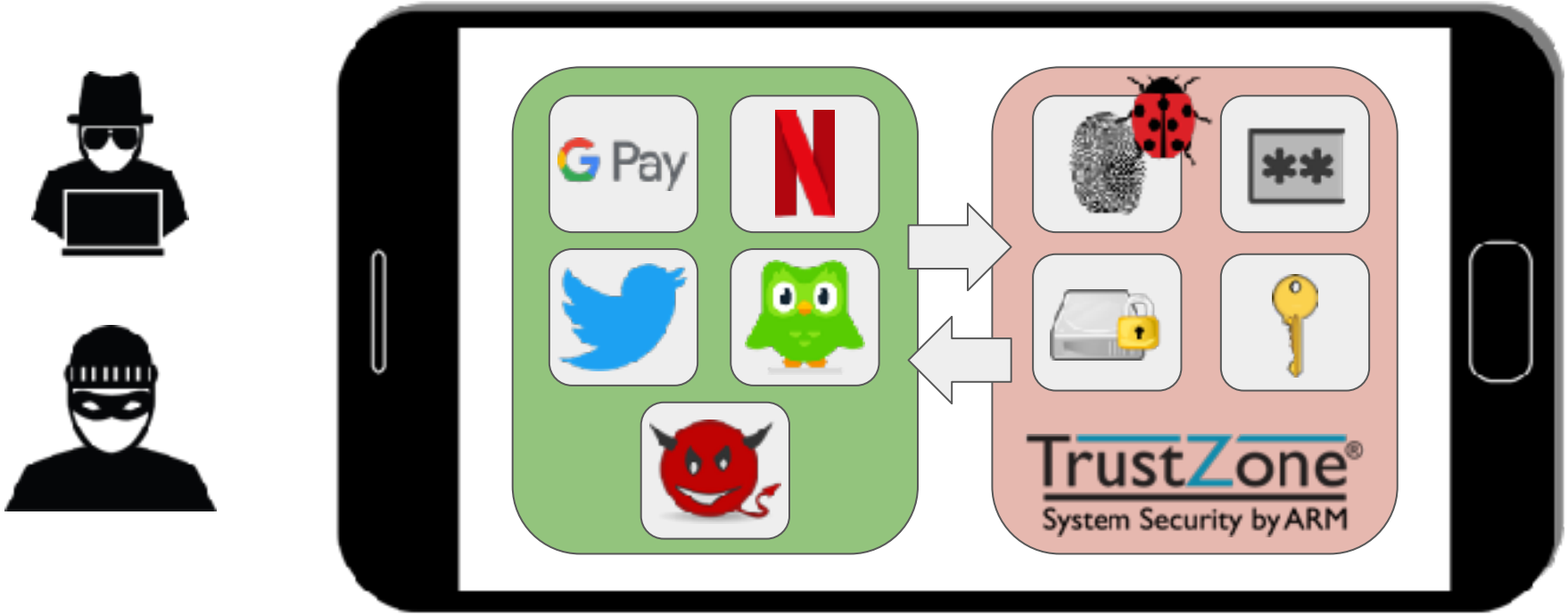
- Finite cycles/time, must spend resources wisely!

Greybox Fuzzers: A Genealogy

datAFLow: Toward a Data-Flow-Guided Fuzzer. *Adrian Herrera, Mathias Payer, and Antony Hosking.* In TOSEM'23

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

Spill the TeA and TEEzz: Trusted Applications on Android Devices



From Crashes to Ranked Bugs

Fuzzing produces (many) crashes, mapping to real bugs

- Programmers are overwhelmed by large amount of crashes
- Crashes need to be distilled into bugs to be useful
- Bugs need complete descriptions

Our findings

- Minimizing path length of seeds enables similarity matching
- Igor groups 254’000 crashes across 39 bugs into 48 distinct clusters
- Evocatio summarizes bug capabilities, bypasses 7 out of 16 CVEs

Igor: Crash Deduplication Through root-Cause Clustering *Zhiyuan Jiang, Xiyue Jiang, Ahmad Hazimeh, Chaojing Tang, Chao Zhang, and Mathias Payer*. In CCS’21.
Evocatio: Conjuring Bug Capabilities from a Single PoC *Zhiyuan Jiang, Shuitao Gan, Adrian Herrera, Flavio Toffalini, Lucio Romerio, Chaojing Tang, Manuel Egele, Chao Zhang, and Mathias Payer*. In CCS’22.

Fuzzing is Maturing. What's next?

Metrics for starting seed corpora and how to generate them

Cross-distillation to reuse seeds among targets

Stateful programs and network protocols

Handling (diverse) peripherals for embedded systems

Helping developers cope with inferred information

Distinguishing exploration and exploitation phases

. . .

Conclusion

hexhive

EPFL

Join us on this research journey!

Join the Software Security Fun Ride!

Bugs are ubiquitous and a (re-)growing resource:

- Software testing weeds them out early
- Mitigations stop attack classes
- Compartmentalization limits their impact

Our research focuses on:

- Specializing fuzzing to new environments
- Enable developers to “understand” bugs
- Customize mitigations per-program
- Infer strong compartmentalization mechanisms

EPFL *Mathias Payer (infosec.exchange/@gannimo)* hexhive