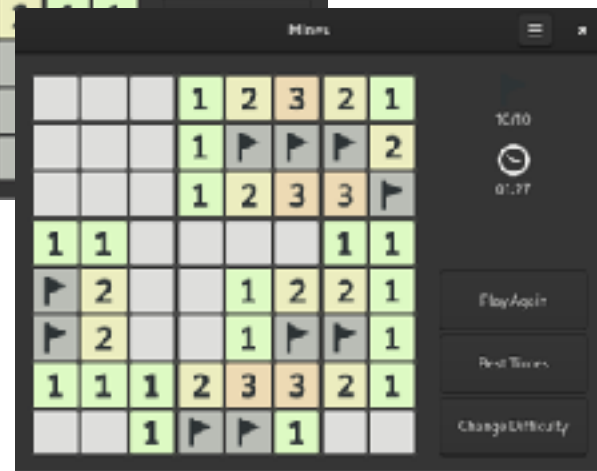


# Why fuzz about security?

Mathias Payer

 [@infosec.exchange](https://twitter.com/infosec.exchange) / [@gannimo](https://twitter.com/gannimo)



**EPFL**



# Fact 1: Software Has Bugs

Bene Brannover / Security

## 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.



Written by @BeneBrannover, Contributor on Feb 11, 2016



We closely study the root cause trends of vulnerabilities & search for patterns

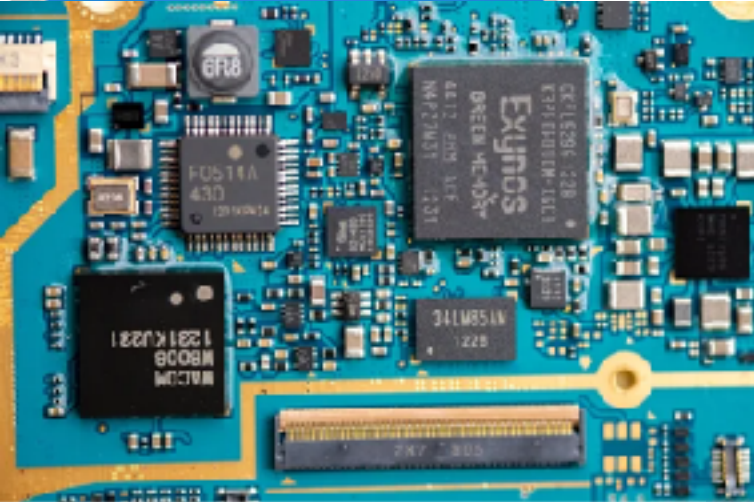
% of memory safety vs. non-memory safety CVEs by patch year



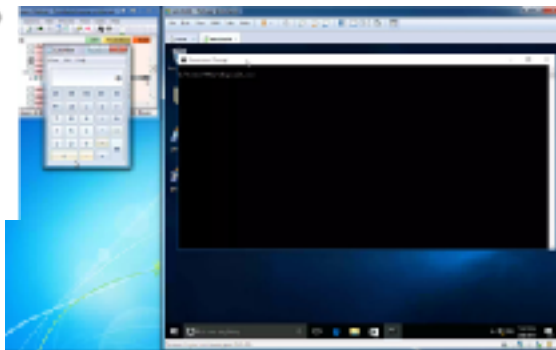
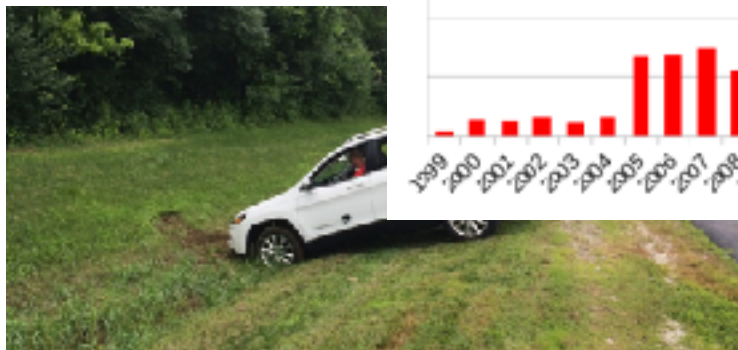
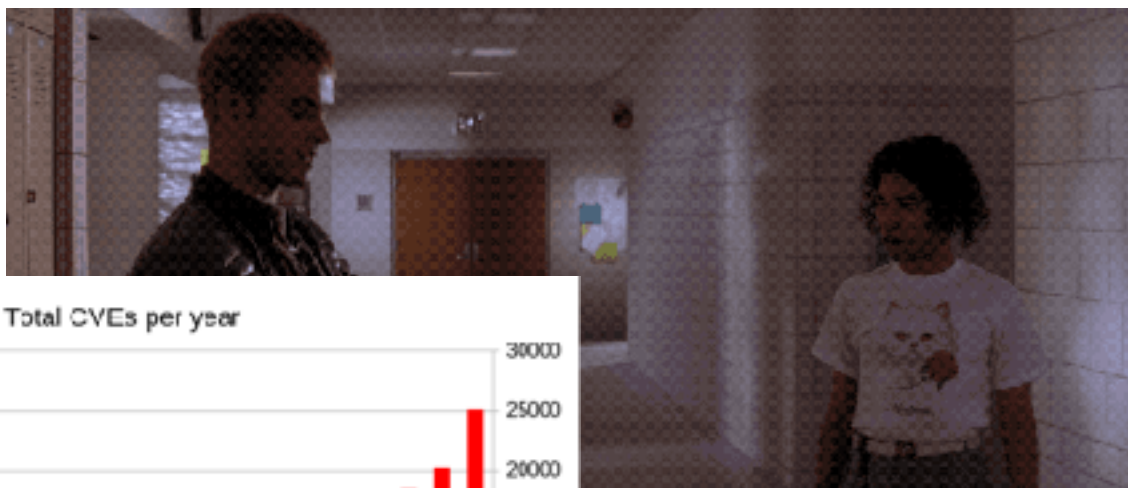
NAME	LAST ACTED	LOGS
0-gpms-updates	Nov 2014	1044
0-gpms-updates-2015	Nov 2015	2008
0-gpms-201512	Nov 2015	2407
0-gpms-201601	Nov 2016	2986
0-gpms-201602-201603	Nov 2016	2943
0-gpms-201604-201605	Nov 2016	3008
0-gpms-201606	Nov 2016	4211
0-gpms-201607-201608	Nov 2016	4488
0-gpms-201609-201610-201611	Nov 2016	5174
0-gpms-201612-201701-201702	Nov 2017	6436
0-gpms-201703-201704	Nov 2017	7497
0-gpms-201705-201706	Nov 2017	8441
0-gpms-201707-201708	Nov 2017	9441
0-gpms-201709-201710	Nov 2017	10441
0-gpms-201711-201712	Nov 2017	11441
0-gpms-201801-201802	Nov 2018	12441
0-gpms-201803-201804	Nov 2018	13441
0-gpms-201805-201806	Nov 2018	14441
0-gpms-201807-201808	Nov 2018	15441
0-gpms-201809-201810	Nov 2018	16441
0-gpms-201811-201812	Nov 2018	17441
0-gpms-201901-201902	Nov 2019	18441
0-gpms-201903-201904	Nov 2019	19441
0-gpms-201905-201906	Nov 2019	20441
0-gpms-201907-201908	Nov 2019	21441
0-gpms-201909-201910	Nov 2019	22441
0-gpms-201911-201912	Nov 2019	23441
0-gpms-202001-202002	Nov 2020	24441
0-gpms-202003-202004	Nov 2020	25441
0-gpms-202005-202006	Nov 2020	26441
0-gpms-202007-202008	Nov 2020	27441
0-gpms-202009-202010	Nov 2020	28441
0-gpms-202011-202012	Nov 2020	29441
0-gpms-202101-202102	Nov 2021	30441



NAME	LAST ACTED	LOGS
0-gpms-updates	Nov 2014	1044
0-gpms-updates-2015	Nov 2015	2008
0-gpms-201512	Nov 2015	2407
0-gpms-201601	Nov 2016	2986
0-gpms-201602-201603	Nov 2016	2943
0-gpms-201604-201605	Nov 2016	3008
0-gpms-201606	Nov 2016	4211
0-gpms-201607-201608	Nov 2016	4488
0-gpms-201609-201610-201611	Nov 2016	5174
0-gpms-201612-201701-201702	Nov 2017	6436
0-gpms-201703-201704	Nov 2017	7497
0-gpms-201705-201706	Nov 2017	8441
0-gpms-201707-201708	Nov 2017	9441
0-gpms-201709-201710	Nov 2017	10441
0-gpms-201711-201712	Nov 2017	11441
0-gpms-201801-201802	Nov 2018	12441
0-gpms-201803-201804	Nov 2018	13441
0-gpms-201805-201806	Nov 2018	14441
0-gpms-201807-201808	Nov 2018	15441
0-gpms-201809-201810	Nov 2018	16441
0-gpms-201811-201812	Nov 2018	17441
0-gpms-201901-201902	Nov 2019	18441
0-gpms-201903-201904	Nov 2019	19441
0-gpms-201905-201906	Nov 2019	20441
0-gpms-201907-201908	Nov 2019	21441
0-gpms-201909-201910	Nov 2019	22441
0-gpms-201911-201912	Nov 2019	23441
0-gpms-202001-202002	Nov 2020	24441
0-gpms-202003-202004	Nov 2020	25441
0-gpms-202005-202006	Nov 2020	26441
0-gpms-202007-202008	Nov 2020	27441
0-gpms-202009-202010	Nov 2020	28441
0-gpms-202011-202012	Nov 2020	29441
0-gpms-202101-202102	Nov 2021	30441



# 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)



Chrome and OS  
~100 MLoC,  
27 lines/page,  
0.1mm/page  $\approx$  370m



# Software Security: Key Research Questions

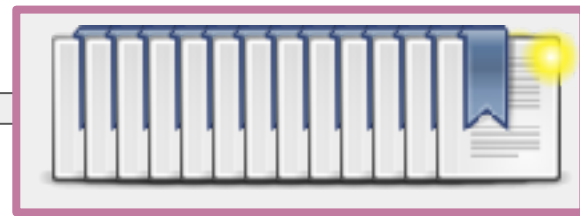
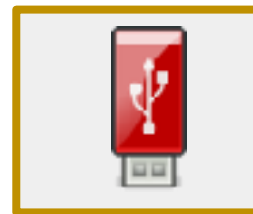
- RQ1: *Efficiently* detect security violations
- RQ2: *Automatically* generate test cases
- RQ3: *Scale* testing to *large* source repositories
- RQ4: *Effectively* test complex interfaces
- RQ5: *Mitigation* co-design based on feedback

```
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);
```



## 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

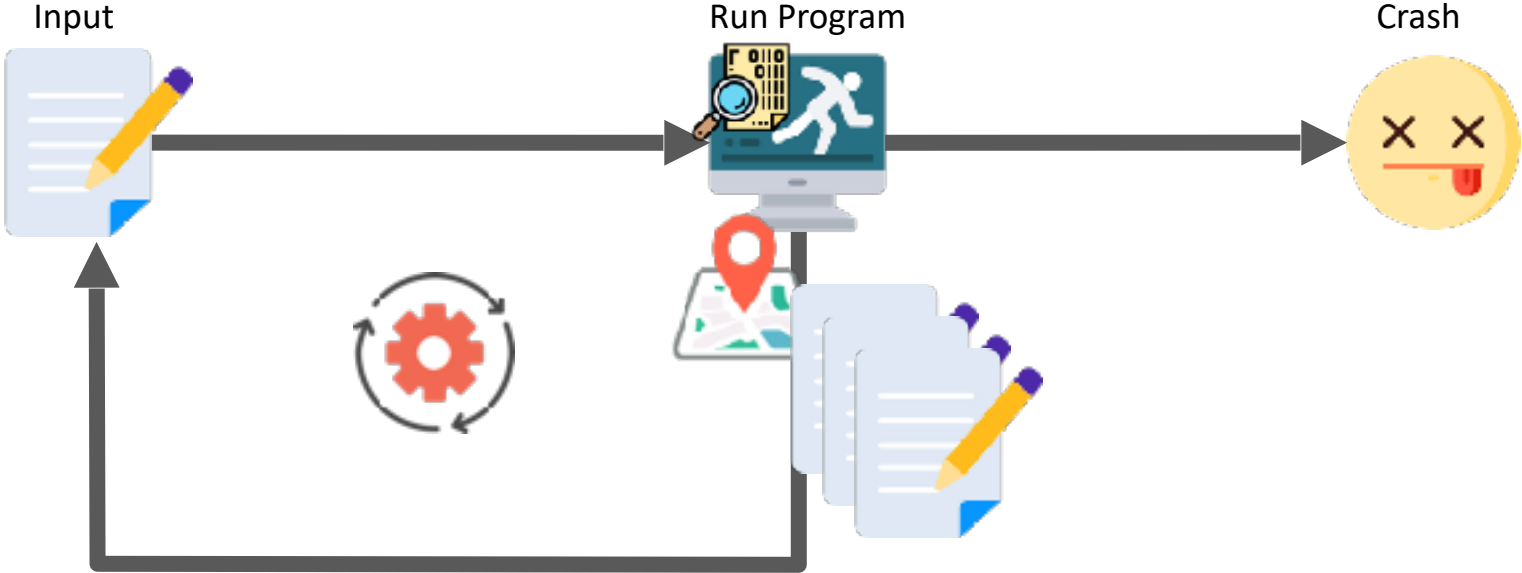
```
$ ./testme --help
Usage: testme <int32_arg>
```

```
$ ./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

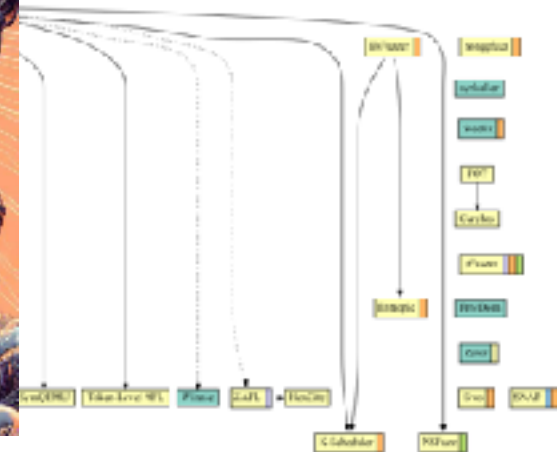


## *Performance* is key (zero sum game)!

- Finite cycles/time, must spend resources wisely!



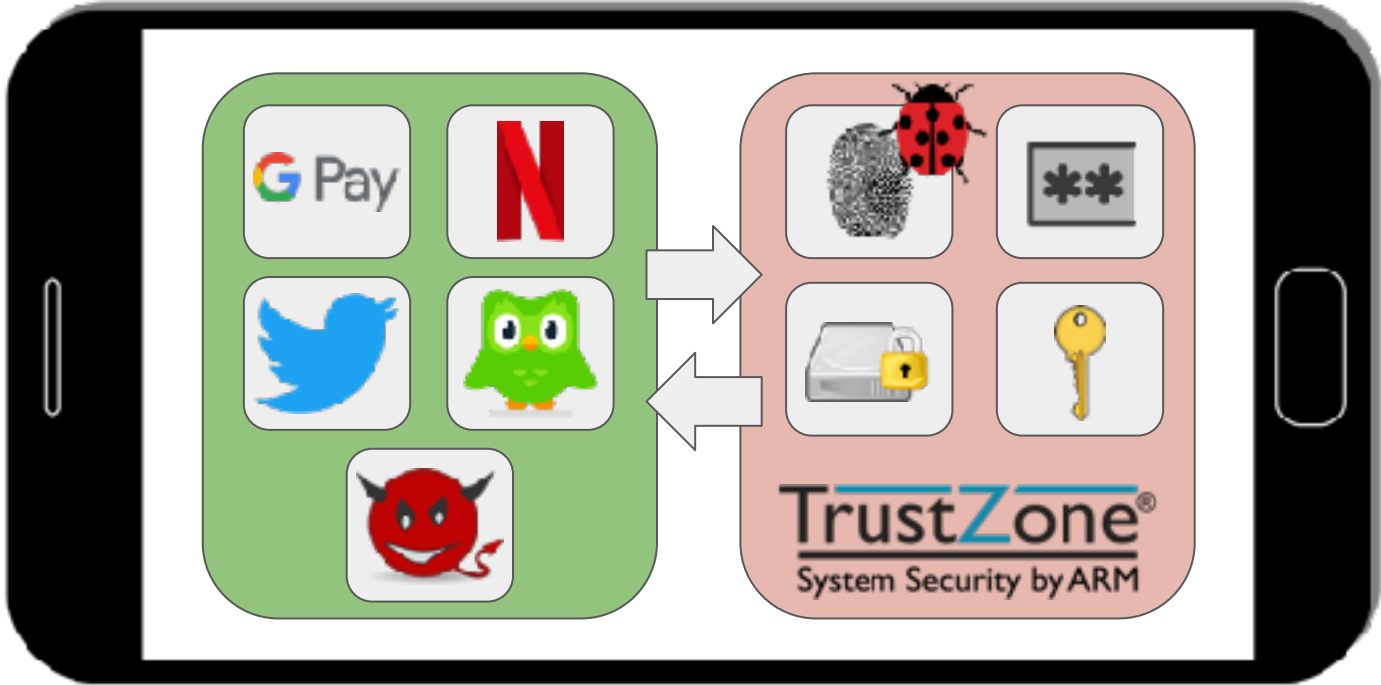
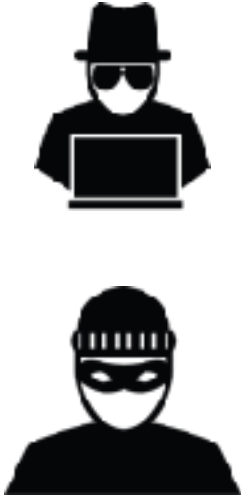
# Greybox Fuzzers: A Genealogy



**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



**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



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

