Week 5: Lecture B

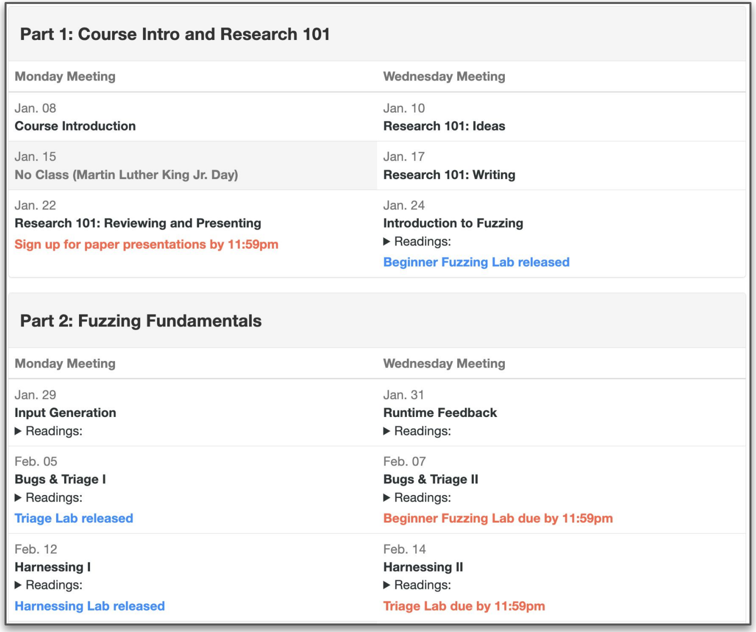
Bugs & Triage II

Wednesday, February 7, 2024

Recap: Key Dates

Feb. 05	**Lab 2 released**
Feb. 07	**Lab 1 due**
Feb. 14	Lab 2 due
Feb. 19	No class (President's Day)
Feb. 28	Lab 3 due
Feb. 28	**5-minute project proposals**
Mar. 04 & 06	No class (Spring Break)
Apr. 17 & 22	**Final project presentations**

cs.utah.edu/~snagy/courses/cs5963/schedule



Part 1: Course Intro and Research 101

Monday Meeting	Wednesday Meeting
Jan. 08 **Course Introduction**	Jan. 10 **Research 101: Ideas**
Jan. 15 **No Class (Martin Luther King Jr. Day)**	Jan. 17 **Research 101: Writing**
Jan. 22 **Research 101: Reviewing and Presenting** **Sign up for paper presentations by 11:59pm**	Jan. 24 **Introduction to Fuzzing** ▶ Readings: **Beginner Fuzzing Lab released**

Part 2: Fuzzing Fundamentals

Monday Meeting	Wednesday Meeting
Jan. 29 **Input Generation** ▶ Readings:	Jan. 31 **Runtime Feedback** ▶ Readings:
Feb. 05 **Bugs & Triage I** ▶ Readings: **Triage Lab released**	Feb. 07 **Bugs & Triage II** ▶ Readings: **Beginner Fuzzing Lab due by 11:59pm**
Feb. 12 **Harnessing I** ▶ Readings: **Harnessing Lab released**	Feb. 14 **Harnessing II** ▶ Readings: **Triage Lab due by 11:59pm**

Lab 2: Crash Triage

- **Assignment:** learn how to use AddressSanitizer (ASAN)
  - Read its documentation in https://clang.llvm.org/docs/AddressSanitizer.html
- **Replay the crashes you found in Lab 1 on an ASAN-instrumented binary**
  - Collect information on each crash
  - What do you observe?
- **Deliverable:** a **1–3 page report** detailing your findings
  - Feel free to make it your own (e.g., pictures, text, etc.)
- **Linux environments are recommended**
  - Use a VM if you don't have one!

Lab 2 Tips

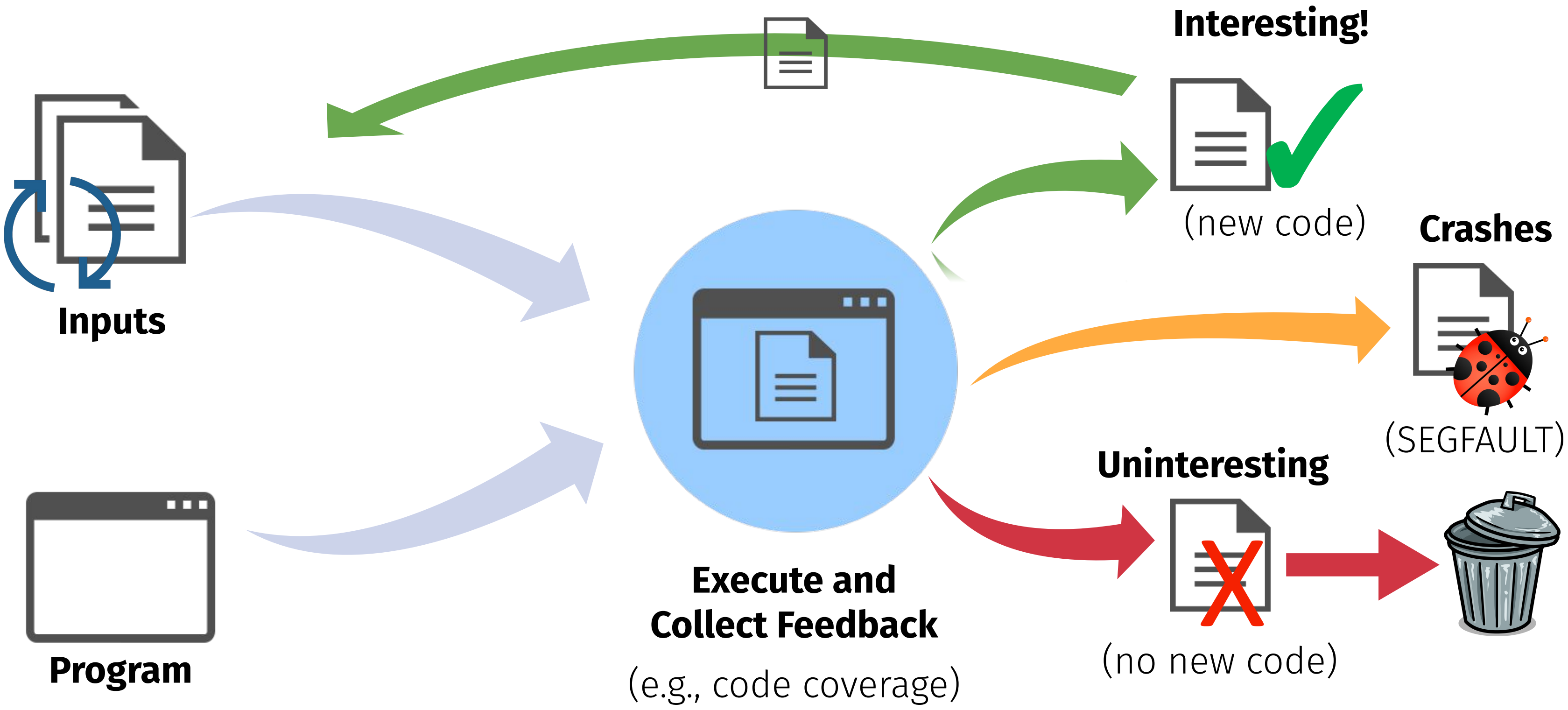
- **Re-run crashes on the ASAN instrumented binary**
  - Use Python to script collection of ASAN outputs
  - Do string post-processing to collect error types, crashing source line, etc.
  - Group and deduplicate crashes as you see fit

- **Didn't find any crashes in Lab 1?**
  - Try fuzzing fuzzgoat from **https://github.com/fuzzstati0n/fuzzgoat**
  - Should yield **lots** of crashes quickly

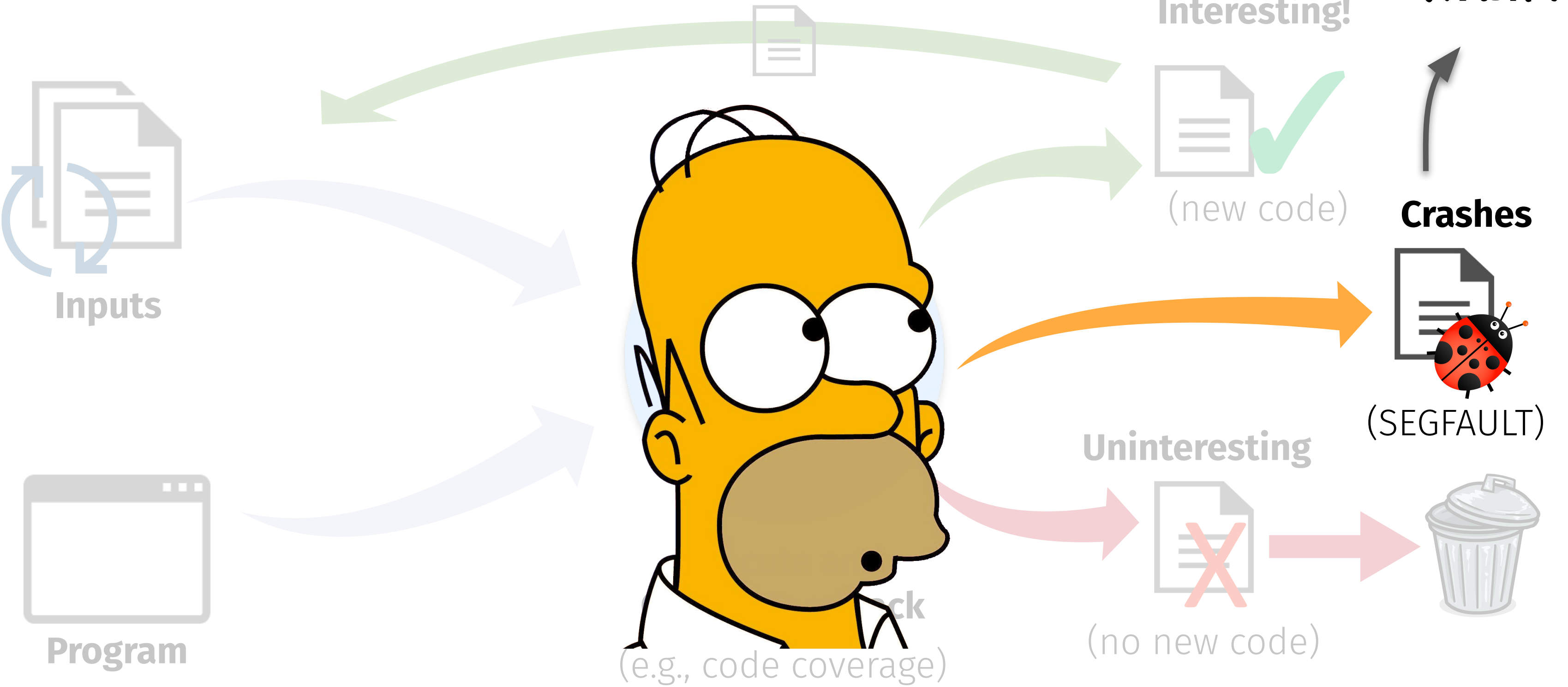
Questions?

Crash Triage

Recap: Coverage-guided Fuzzing



Recap: Coverage-guided Fuzzing



So your fuzzer found some crashes...

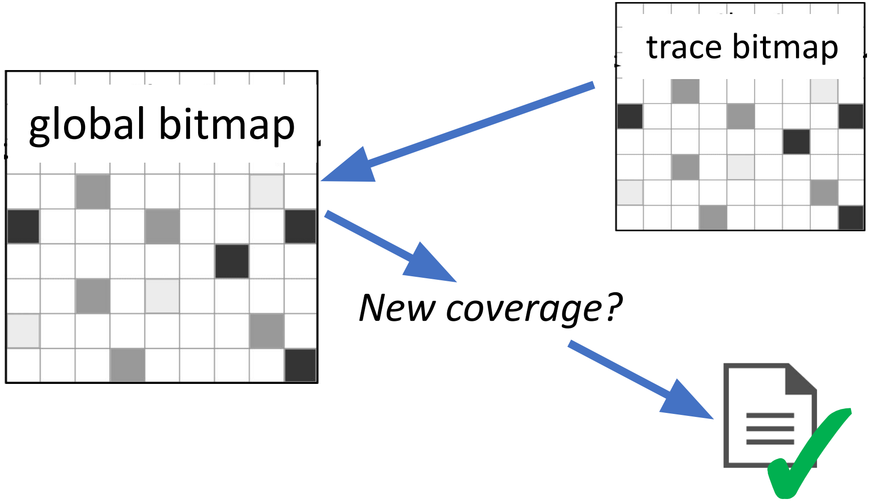
- Are they actually real bugs?
  - Your fuzzer may be lying to you...
- What kind of bugs were found?
  - Type (e.g., logic, memory safety)
  - Root cause
- How severe is each bug?
  - **Developers:** which to prioritize
  - **Reporters:** convince developers

total crashes : 200 (99 unique)

Crash Deduplication

AFL's "Unique" Crashes

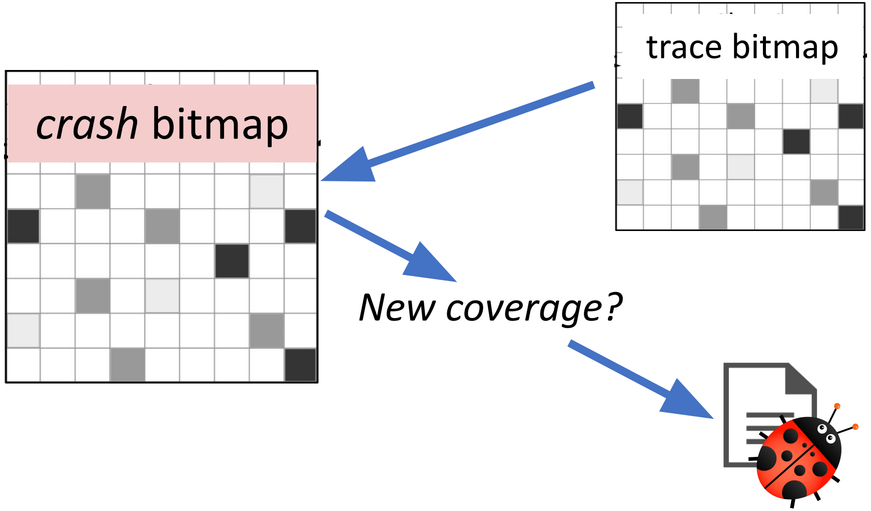
- AFL repurposes its coverage bitmap to count unique crashes



```
cur_location = <COMPILE_TIME_RANDOM>;
Shared_mem [cur_location ⊕ prev_location]++;
prev_location = cur_location >> 1;
```

AFL's "Unique" Crashes

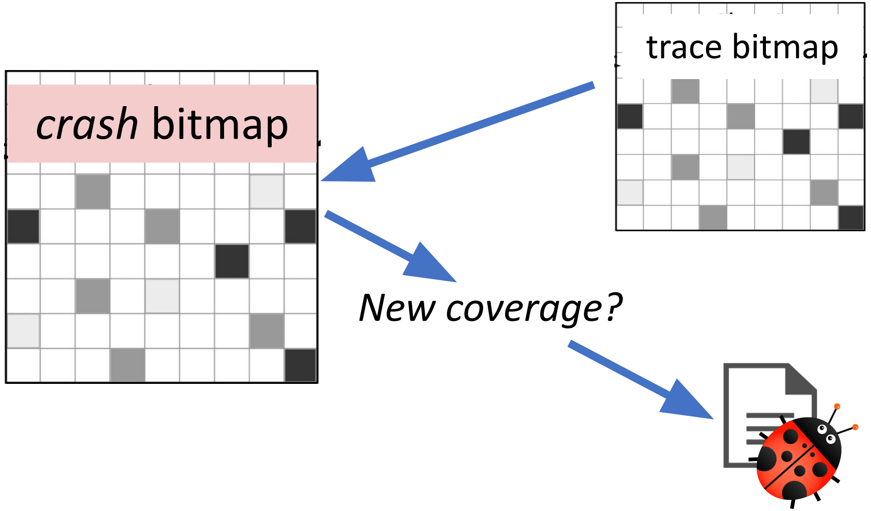
- AFL repurposes its coverage bitmap to count unique crashes
  - **New crash edge? New unique crash**



```
cur_location = <COMPILE_TIME_RANDOM>;
Shared_mem [cur_location ⊕ prev_location]++;
prev_location = cur_location >> 1;
```

AFL’s “Unique” Crashes

- AFL repurposes its coverage bitmap to count unique crashes
  - **New crash edge? New unique crash**
- Influenced by weird things
  - Non-deterministic behavior
  - Undefined behavior
  - **Bitmap collisions**
- **Not a sound metric for “bugs”**



```
cur_location = <COMPILE_TIME_RANDOM>;
Shared_mem [cur_location ⊕ prev_location]++;
prev_location = cur_location >> 1;
```

How should we group crashes?

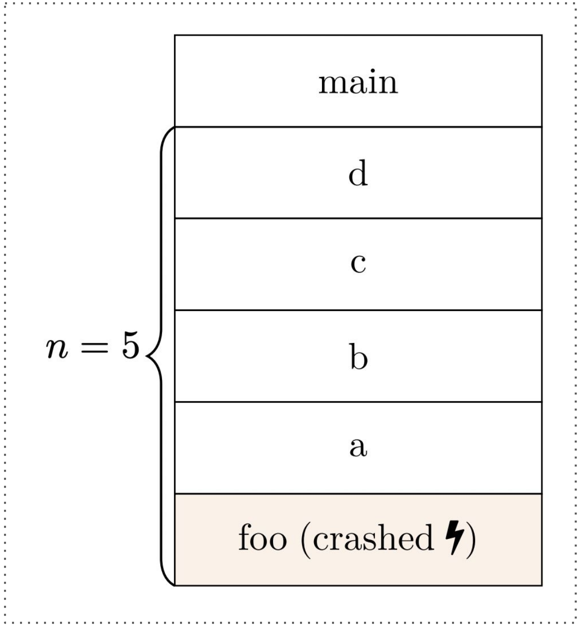
- **Manually**
  - Need domain expertise
  - Hard to enumerate lots of crashes

- **Automatically**
  - Scripted tooling
  - Requires a good “proxy” metric
    - Performance vs. precision

Fuzzy Stack Hashing

- **Approximated measure of bugs found**
  - E.g., `MD5(“foo|a|b|c|d”)`
  - Most popular proxy metric in use today

- **Idea:** concatenate top-*N* stack frames for each crashing test case
  - **Large *N*** = every crash unique (**over-count**)
  - **Small *N*** = few crashes unique (**under-count**)
  - **Most set *N* arbitrarily**



Source: The Art, Science, and Engineering of Fuzzing: A Survey

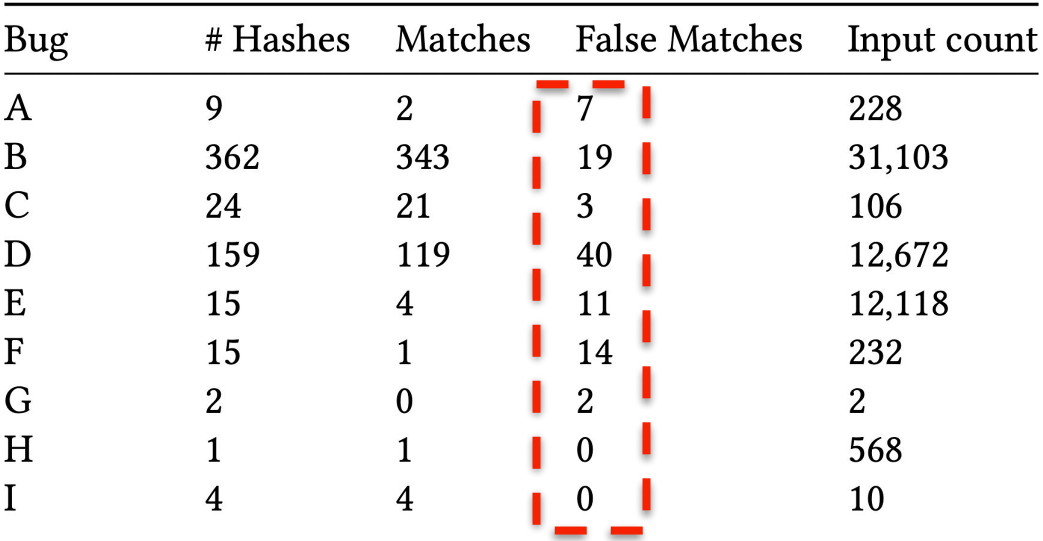
Fuzzy Stack Hashing

- **Concatenate more information**
  - Source code lines
  - Addresses
  - Crashing signal
  - ASAN-reported bug type
    - E.g., `MD5("UAF:foo|a|b|c|d")`

```
== ASAN: heap-use-after-free on address
0x61900000047f at pc 0x00000040a52c bp
0x7fff9200dbf0 sp 0x7fff9200dbe0
READ of size 1 at 0x61900000047f thread T0
    #0 0x40a52b in src/main.cpp:30
    #1 0x40e088 in std_function.h:297
    #2 0x40d605 in std_function.h:687
    #3 0x40b8d5 in src/main.cpp:130
    #4 0x7f9a498ff412 in libc-start.c:308
```

Trade-offs

- Fast to collect, but…
- *N*-values completely change results
- **Still over-counts bugs**
  - But not as much as AFL

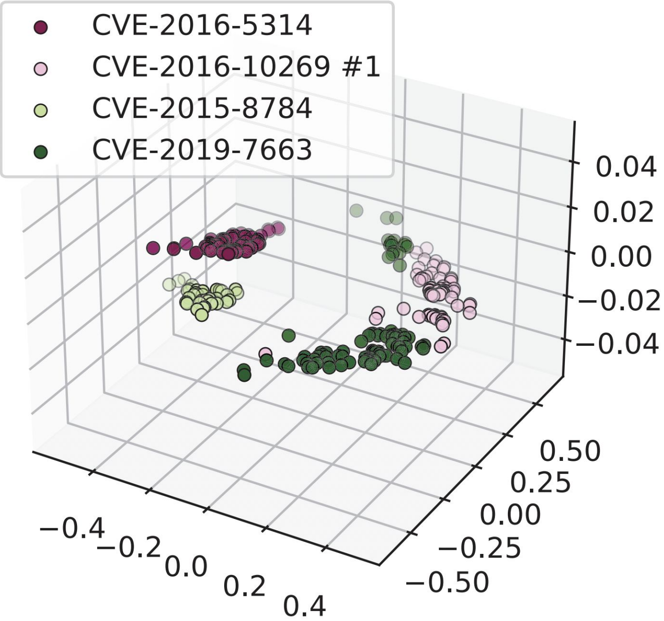


Bug	# Hashes	Matches	False Matches	Input count
A	9	2	7	228
B	362	343	19	31,103
C	24	21	3	106
D	159	119	40	12,672
E	15	4	11	12,118
F	15	1	14	232
G	2	0	2	2
H	1	1	0	568
I	4	4	0	10

Source: Evaluating Fuzz Testing

Crash Clustering

- **Idea:** mutate crashing test cases
  - Group them by similar characteristics
    - E.g., crashing vs. not crashing
    - E.g., coverage of buggy path
  - **Infer bug root causes from clusters**
    - Find common input properties
- **Trade-offs:** results not instant
  - A lot more fuzzing is needed
  - Sacrifice speed for precision



Igor: Crash Deduplication Through Root-Cause Clustering

Exploitability Assessment

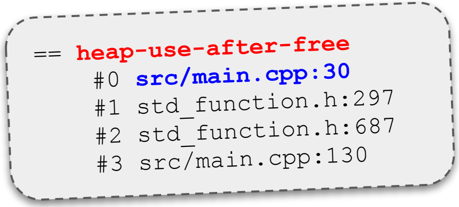
- **What is needed to exploit this bug?**
  - E.g., process and kernel state
  - In other words: **can you write an exploit for it?**

- Automatic Exploit Generation (AEG)
  - Only works for simple bugs
  - Many assumptions that don't hold
  - **Unsolved (and not-easily-solvable) problem**

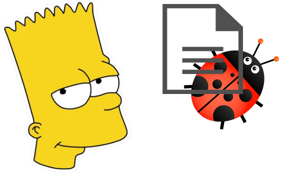
- **Best option today: do it by hand**
  - A "dark art" with a steep learning curve
  - *Did someone say a **CTF Team**...?*

Responsible Disclosure

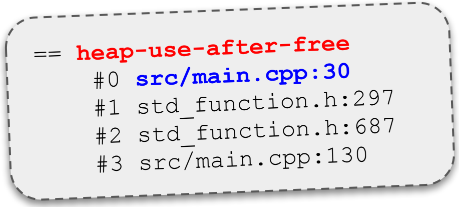
Disclosing Bugs Responsibly



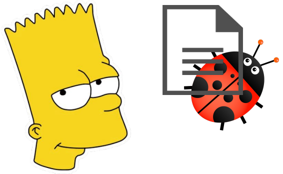
```
== heap-use-after-free
    #0  src/main.cpp:30
    #1  std_function.h:297
    #2  std_function.h:687
    #3  src/main.cpp:130
```

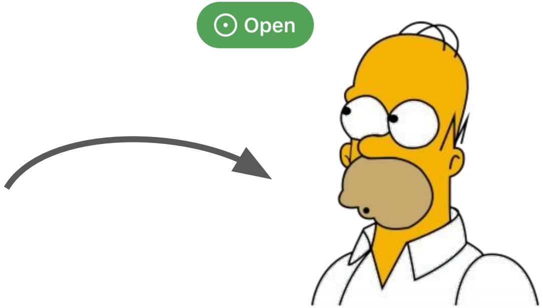


Disclosing Bugs Responsibly



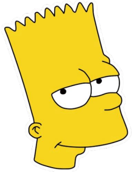
```
== heap-use-after-free
   #0 src/main.cpp:30
   #1 std_function.h:297
   #2 std_function.h:687
   #3 src/main.cpp:130
```



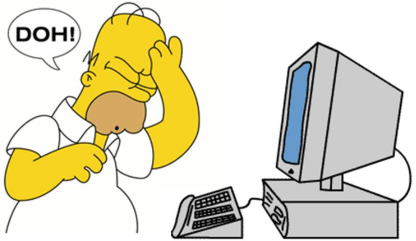


Disclosing Bugs Responsibly

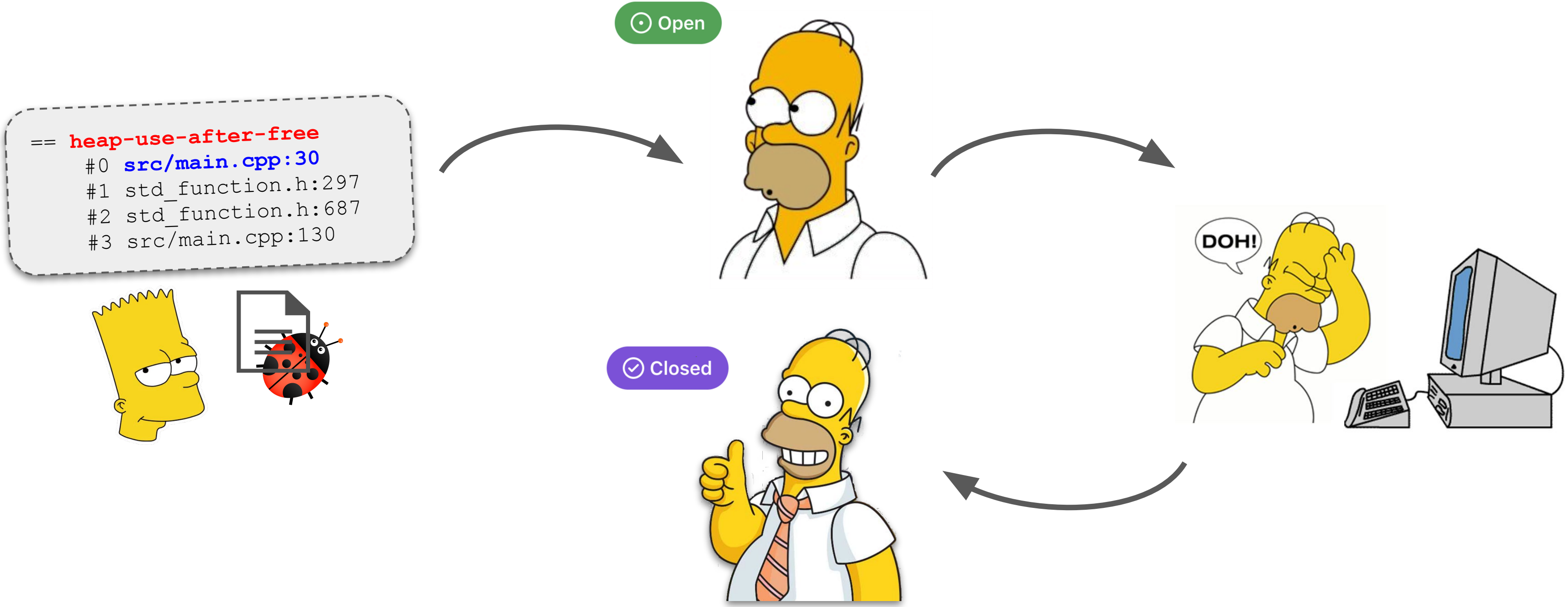
```
== heap-use-after-free
   #0 src/main.cpp:30
   #1 std_function.h:297
   #2 std_function.h:687
   #3 src/main.cpp:130
```



⊙ Open

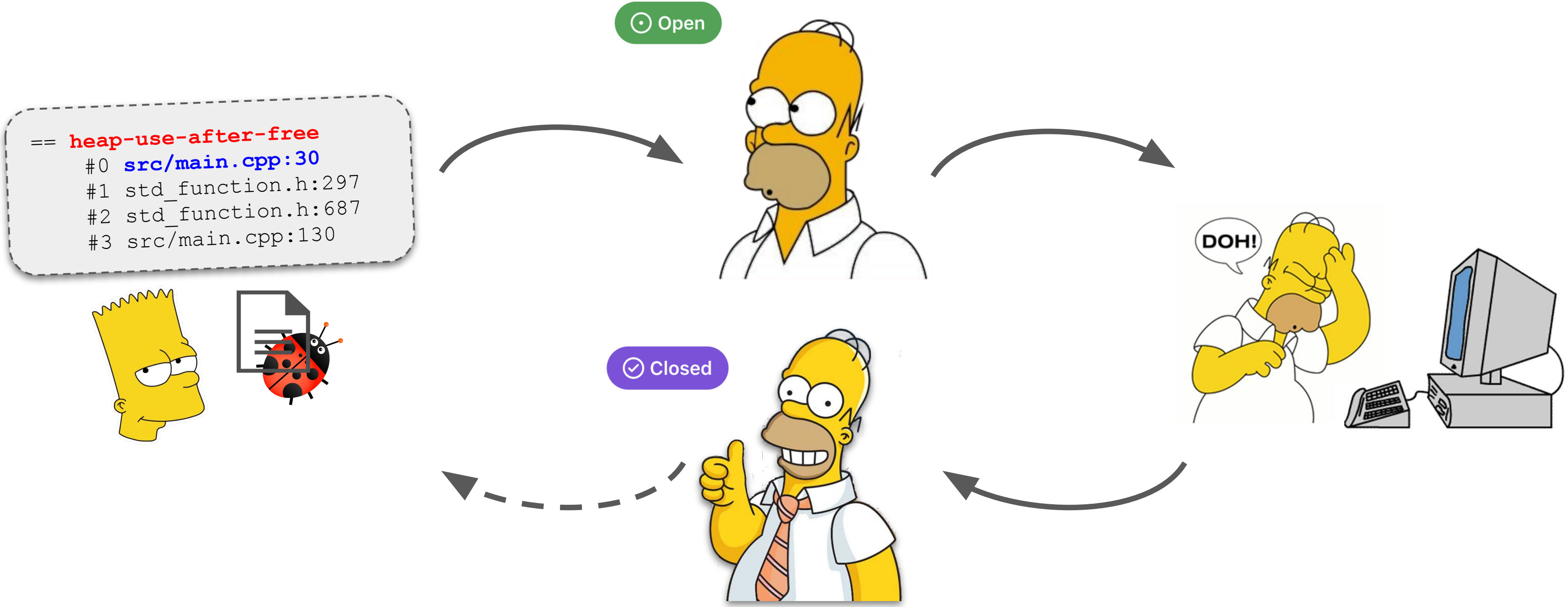


Disclosing Bugs Responsibly



```
== heap-use-after-free
    #0 src/main.cpp:30
    #1 std_function.h:297
    #2 std_function.h:687
    #3 src/main.cpp:130
```

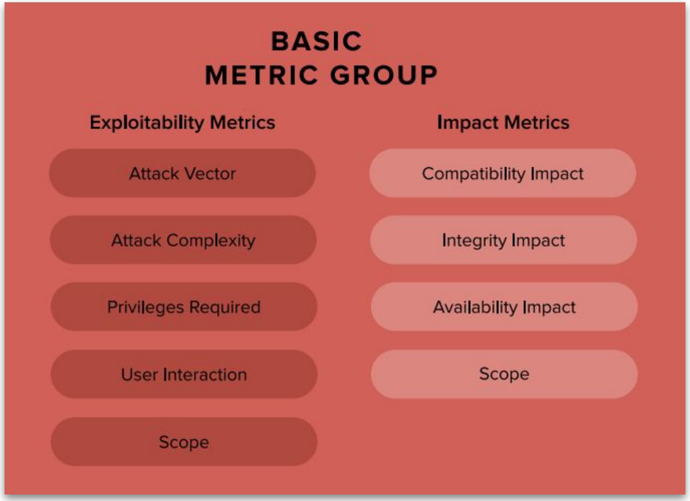
Disclosing Bugs Responsibly



```
== heap-use-after-free
    #0 src/main.cpp:30
    #1 std_function.h:297
    #2 std_function.h:687
    #3 src/main.cpp:130
```

What developers love...

- **Proof-of-concept test cases**
  - Devs need to reproduce your bug
  - Perform their own severity analysis
    - Limited time and resources
    - Fix most severe ones first
    - E.g., MS Patch Tuesday
  - Help them improve their test suites



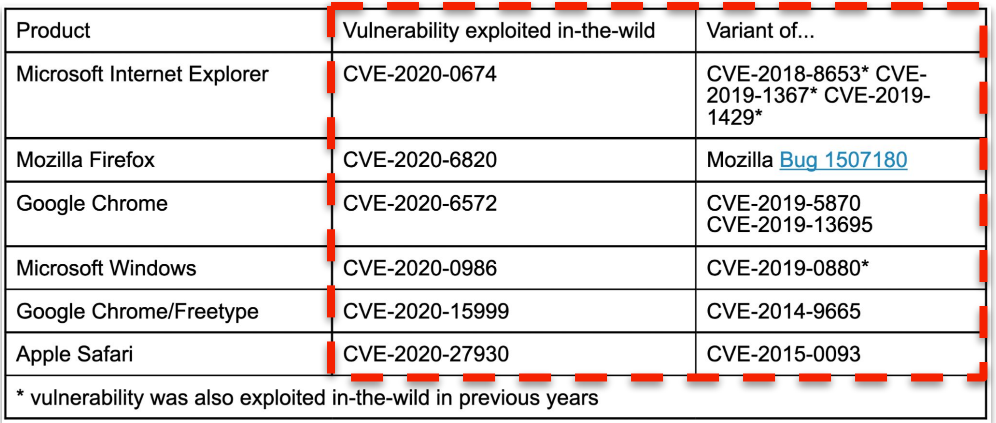
What developers love...

- **Actionable insights**
  - ***Basic:*** build information
    - E.g., compiler, version, OS, etc.
    - Only report bugs in the latest version!
  - ***Good*:** crashing source lines, PoCs
  - ***Better:*** root cause analysis
    - E.g., *Missing a check on chunk X*
    - You'll need to get your hands dirty
  - ***Best*:** proposed patches
    - May be a back-and-forth battle

Come on you guys! You're dereferencing a null pointer. It's right there!

What developers love…

- **Follow-up testing**
  - Initial fixes may be incomplete
  - Re-run your fancy fuzzer
  - **Open-source your fancy fuzzer**



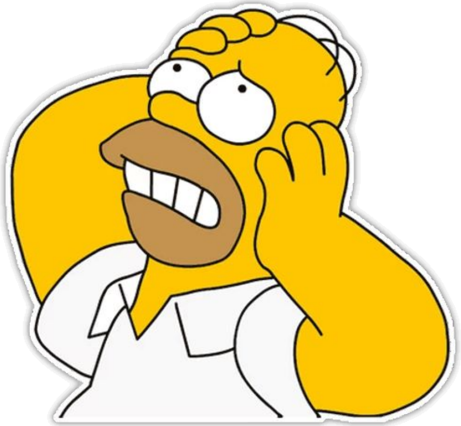
Product	Vulnerability exploited in-the-wild	Variant of...
Microsoft Internet Explorer	CVE-2020-0674	CVE-2018-8653* CVE-2019-1367* CVE-2019-1429*
Mozilla Firefox	CVE-2020-6820	Mozilla Bug 1507180
Google Chrome	CVE-2020-6572	CVE-2019-5870 CVE-2019-13695
Microsoft Windows	CVE-2020-0986	CVE-2019-0880*
Google Chrome/Freetype	CVE-2020-15999	CVE-2014-9665
Apple Safari	CVE-2020-27930	CVE-2015-0093

* vulnerability was also exploited in-the-wild in previous years

Source: Deja Vulnerability by Google Project Zero

What developers *hate*…

- **Little (or unhelpful) information**
  - No PoC test cases or stack traces
  - Bugs on obsolete versions
    - E.g., *I installed this via apt-get*
  - Spamming tons of bug reports
    - Duplicate bug reports
    - Already-reported bugs



What developers *hate*...

- **Selfish resumé padding**
  - Requesting CVE assignment without first asking them
    - Common in academic papers
    - Reviewers are partially to blame
  - **Developers can (and do) dispute CVEs**

What developers *hate*…

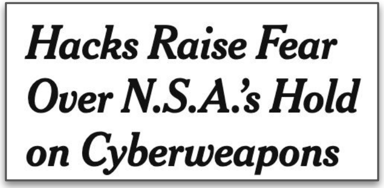
- **Weaponizing and selling an exploit**
  - A huge underground economy
    - Nation-state actors
    - Cyber-criminal gangs

What developers *hate*…

- **Weaponizing and selling an exploit**
  - A huge underground economy
    - Nation-state actors
    - Cyber-criminal gangs
  - **Don’t do this**

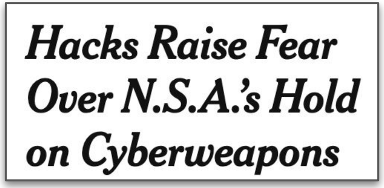
What developers *hate*…

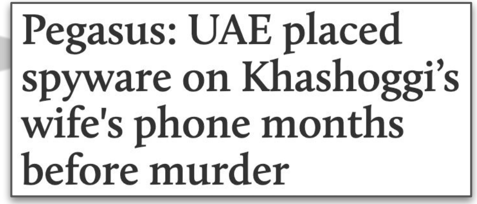
- **Weaponizing and selling an exploit**
  - A huge underground economy
    - Nation-state actors
    - Cyber-criminal gangs
  - **Don't do this**
    - Likely to end up in bad hands regardless of who brokered it


Hacks Raise Fear Over N.S.A.'s Hold on Cyberweapons

What developers *hate*...

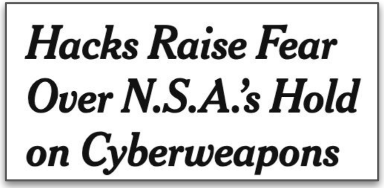
- **Weaponizing and selling an exploit**
  - A huge underground economy
    - Nation-state actors
    - Cyber-criminal gangs
  - **Don't do this**
    - Likely to end up in bad hands regardless of who brokered it
    - Authoritarian regimes use these all the time for **evil acts**
    - You are very likely causing people to get hurt **(or worse)**

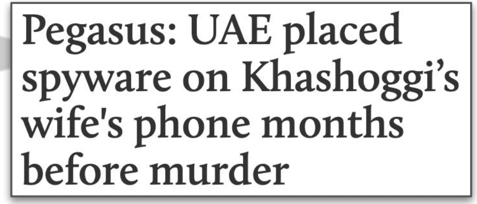

Hacks Raise Fear Over N.S.A.'s Hold on Cyberweapons


Pegasus: UAE placed spyware on Khashoggi's wife's phone months before murder

What developers *hate*…

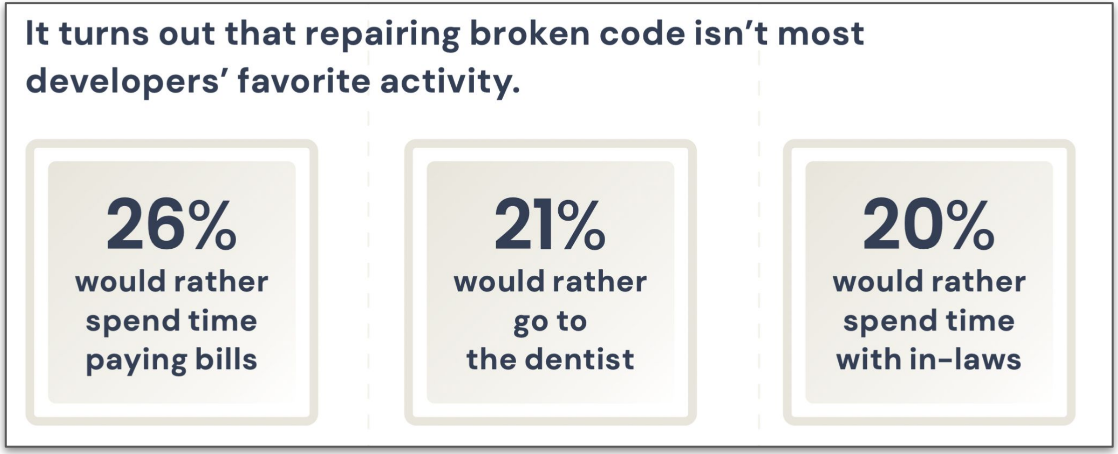
- **Weaponizing and selling an exploit**
  - A huge underground economy
    - Nation-state actors
    - Cyber-criminal gangs
  - **Don’t do this**
    - Likely to end up in bad hands regardless of who brokered it
    - Authoritarian regimes use these all the time for **evil acts**
    - You are very likely causing people to get hurt **(or worse)**
    - **You will fail this class (and worse)**


Hacks Raise Fear Over N.S.A.’s Hold on Cyberweapons


Pegasus: UAE placed spyware on Khashoggi’s wife's phone months before murder

Developers are people, too

- Data suggests that fixing bugs is a really tough job



It turns out that repairing broken code isn’t most developers’ favorite activity.

26% would rather spend time paying bills

21% would rather go to the dentist

20% would rather spend time with in-laws

- **Treat developers with courtesy, respect, and patience**

Source: https://content.rollbar.com/hubfs/State-of-Software-Code-Report.pdf

Questions?