Week 5: Lecture A

Bugs & Triage I

Monday, February 5, 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**

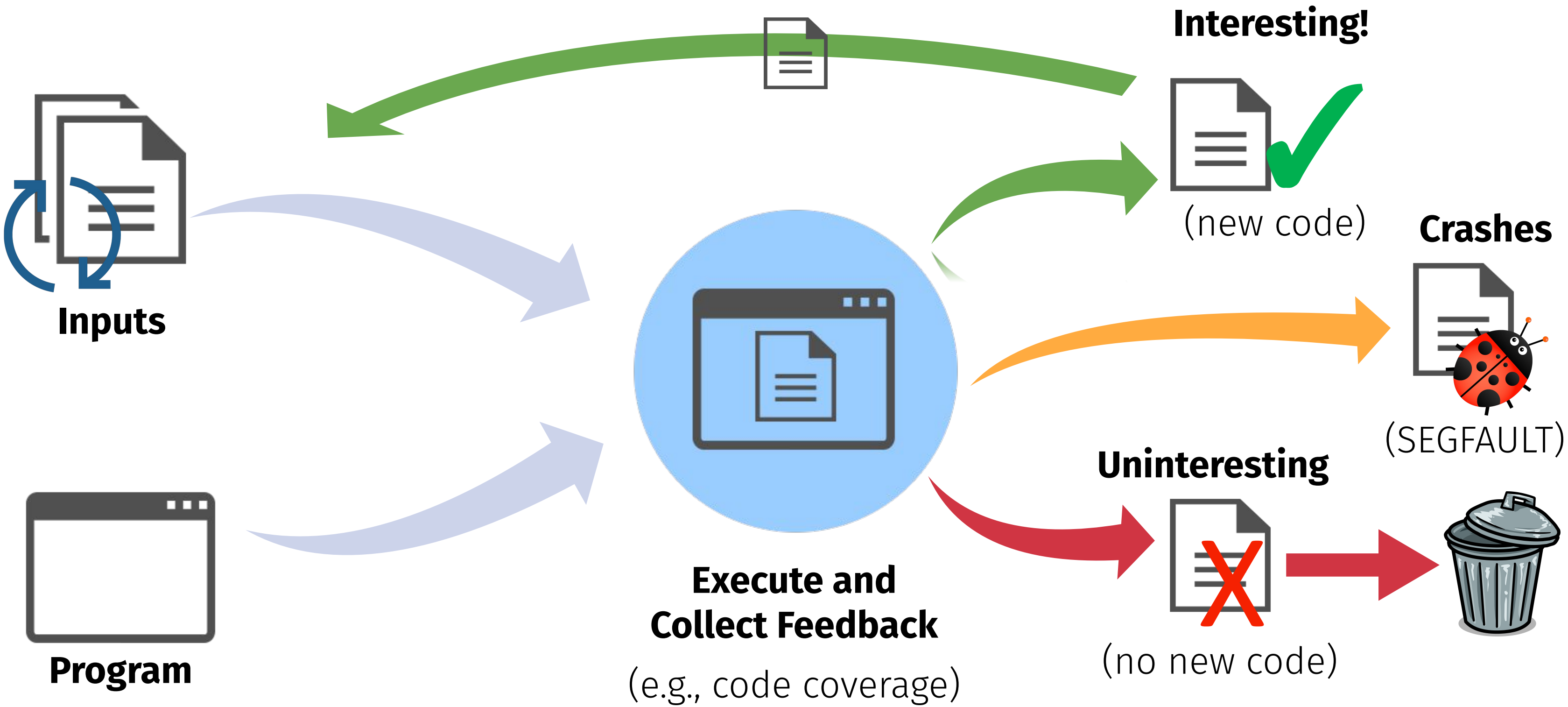
Part 2: Fuzzing Fundamentals

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**

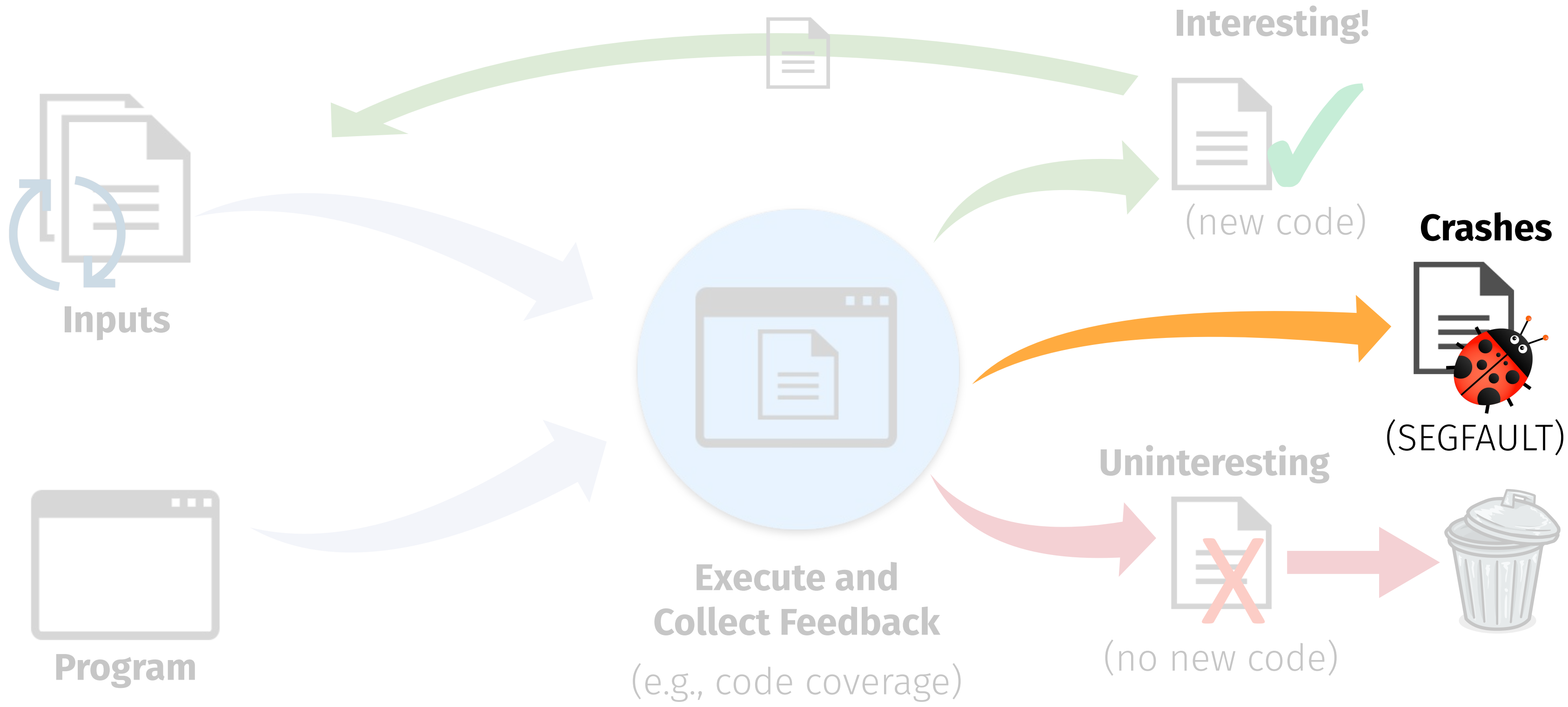
Questions?

Fuzzing for Bugs

Recap: Coverage-guided Fuzzing

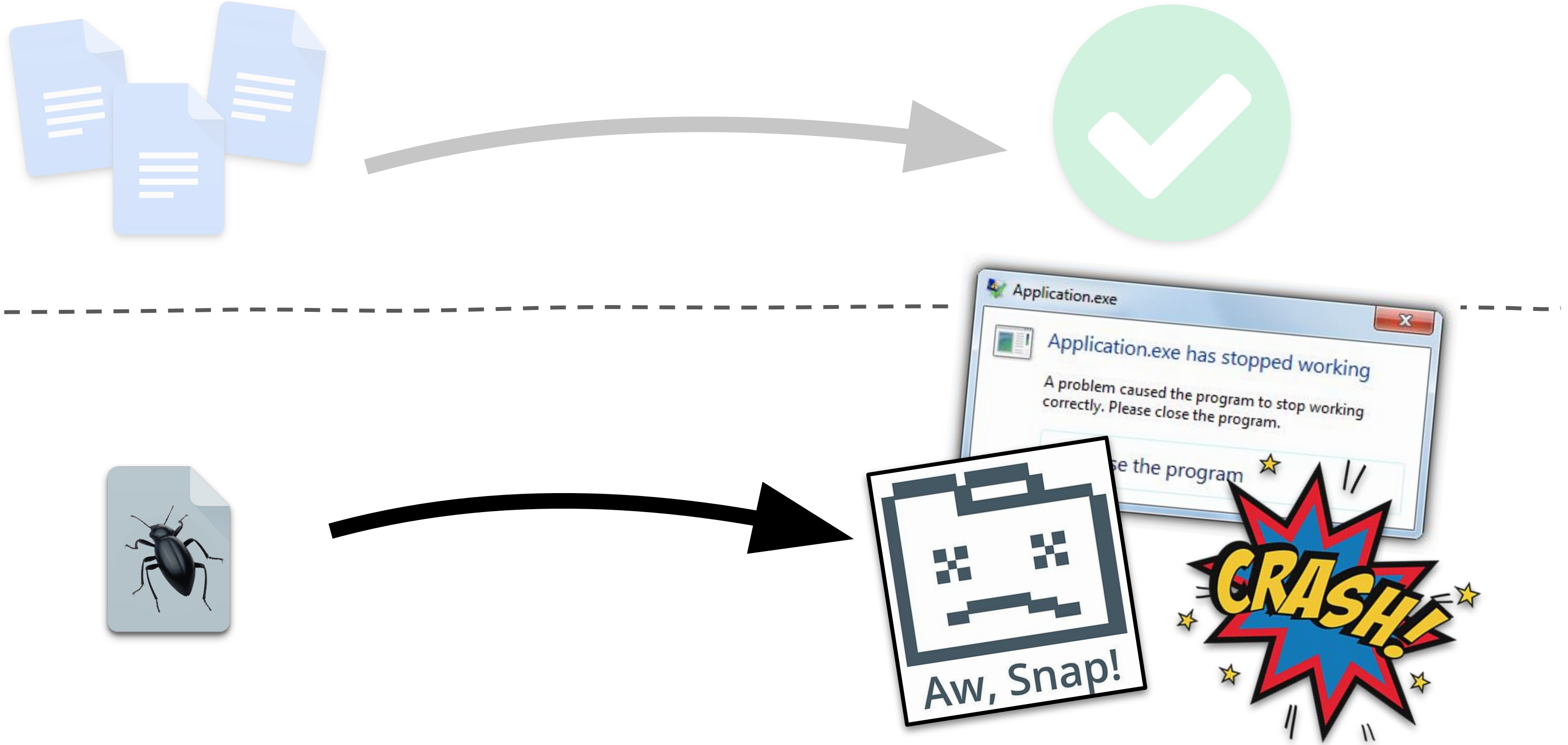


Recap: Coverage-guided Fuzzing

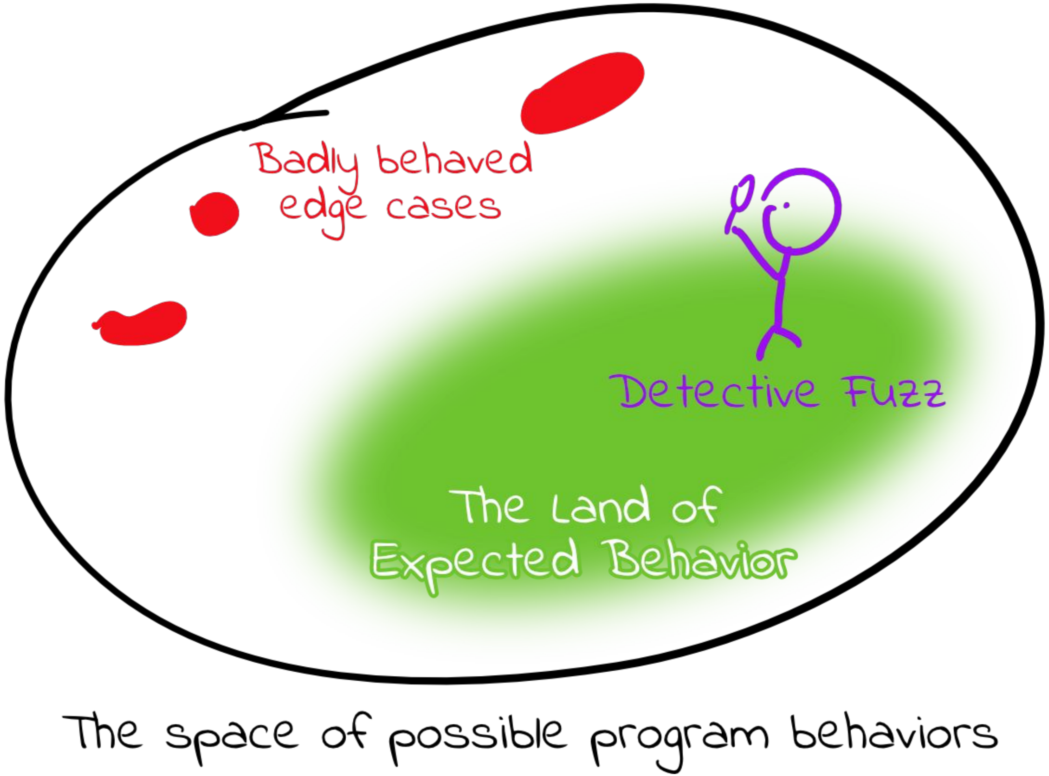


Recap: Software Bugs

Recap: Software Bugs



Recap: Finding Bugs with Fuzzing



Source: https://blog.trailofbits.com/2020/10/22/lets-build-a-high-performance-fuzzer-with-gpus/

Before you start: choose your oracle!

- **Oracles:** proxies for triggering software bugs
  - Can be general-purpose
  - Can be program-specific
  - **Up to you to decide**

Before you start: choose your oracle!

- **Oracles:** proxies for triggering software bugs
  - Can be general-purpose
  - Can be program-specific
  - **Up to you to decide**

- Common oracles:
  - **Crashes:** memory safety bugs
  - **AddressSanitizer:** a better memory safety oracle
  - **Assertion failures:** program logic bugs
  - **Differential testing:** implementation-specific bugs

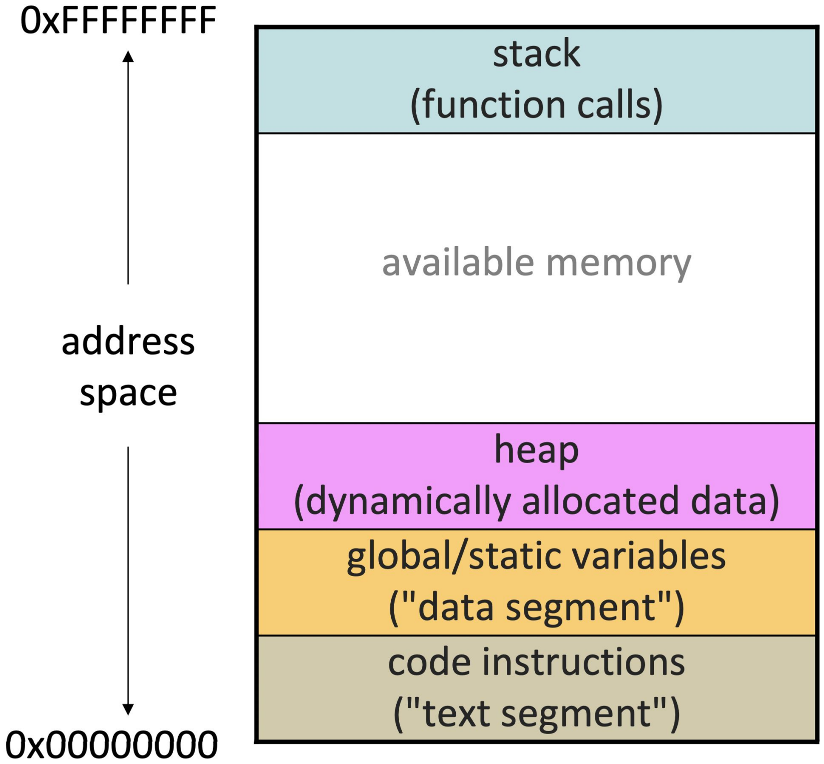
Considerations and Trade-offs

- What **kind of bugs** are you looking for?
  - Does it require fundamentally new tooling?
  - E.g., resource exhaustion vs memory corruption
  - What are the engineering obstacles?

- At what **cost**?
  - High speed is critical for effective fuzzing
  - E.g., AddressSanitizer adds over 6x overhead

Memory Corruption Oracles

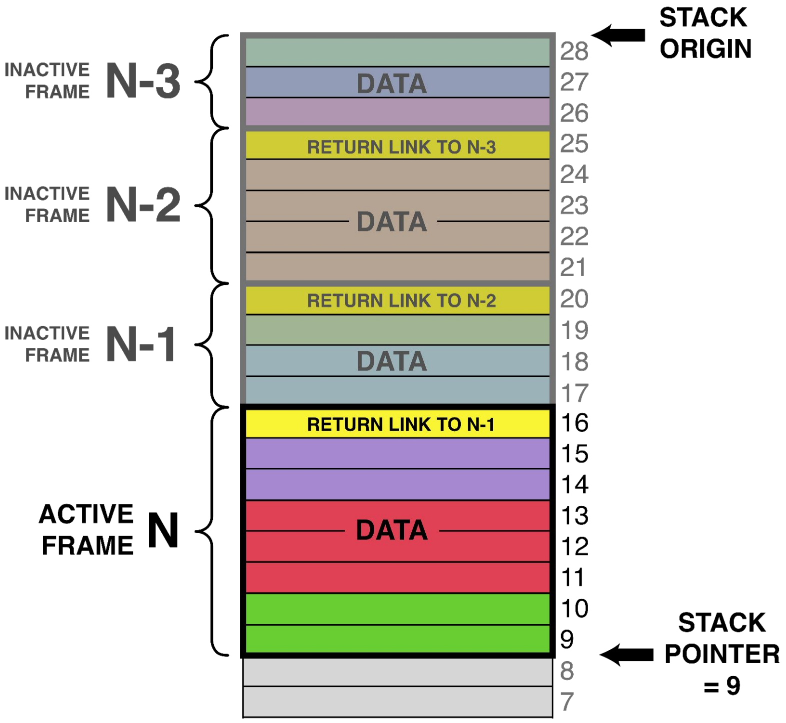
Process Memory



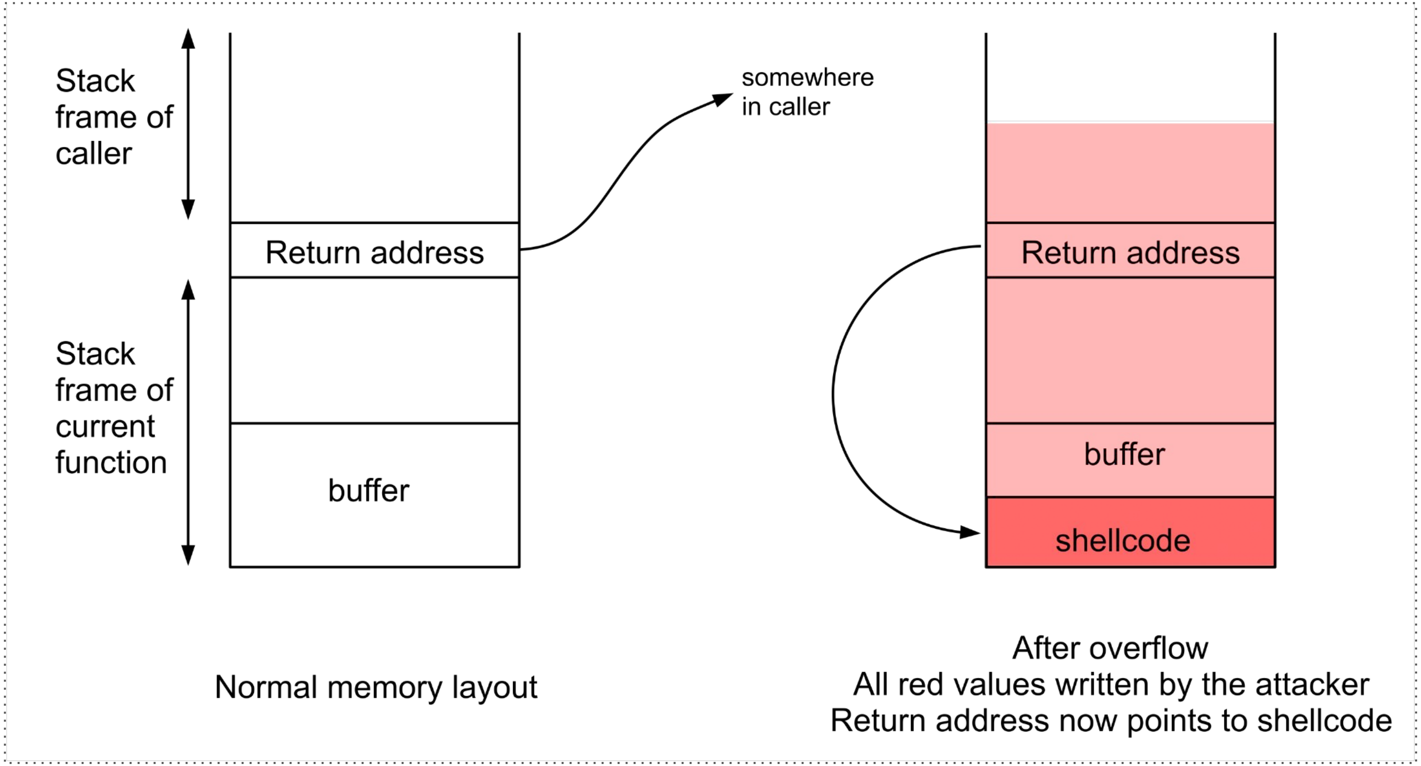
Source: https://courses.cs.washington.edu/courses/cse303/09sp/lectures/2009-04-22/11-heap.pdf

The Stack

- Memory for storing **function data**
  - Arguments
  - Local variables
  - Return addresses
- Allocation the compiler's job
  - Deallocation done on function exit
- **Bounds-checking is programmer's job**

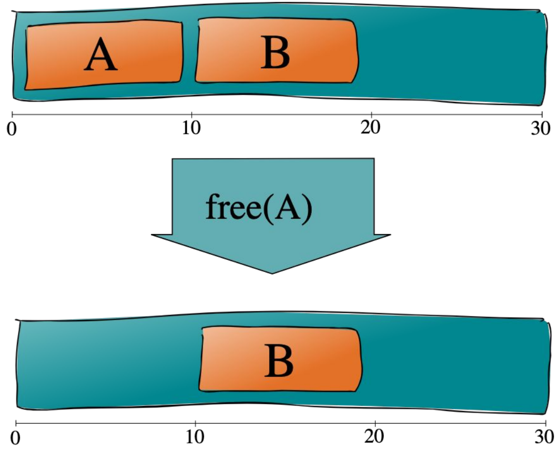


Stack Corruption



The Heap

- **Dynamically**-allocated memory
  - Allocated via **malloc()**, and freed via **free()**
  - Chunks may get allocated, freed, split, coalesced
  - Regions accessed via **pointers**
- Management is **programmer's job**
  - Pointers must point to **live objects**
  - Must point to objects of the **right type**
  - Only pointers to **functions** can be executed
  - …



Heap Corruption

```c
int* a1 = (int*) calloc(1000, sizeof(int));
int a2[1000];
int* a3;
int* a4 = NULL;

free(a1);      // ok
free(a1);      // bad (already freed)
free(a2);      // bad (not heap allocated)
free(a3);      // bad (not heap allocated)
free(a4);      // bad (not heap allocated)
```

Source: https://courses.cs.washington.edu/courses/cse303/09sp/lectures/2009-04-22/11-heap.pdf

Crashes as Oracles

- Memory corruption messes with **program state**
- Injecting random data may **redirect execution**
  - Overwriting a return address on the stack
  - Overwriting a called function pointer on the heap
- **Result:** garbage operations that **crash the program**
  - **SIGILL:** invalid instruction
  - **SIGSEGV:** invalid memory access
  - **SIGFPE:** erroneous arithmetic operation

Where crashes fall short

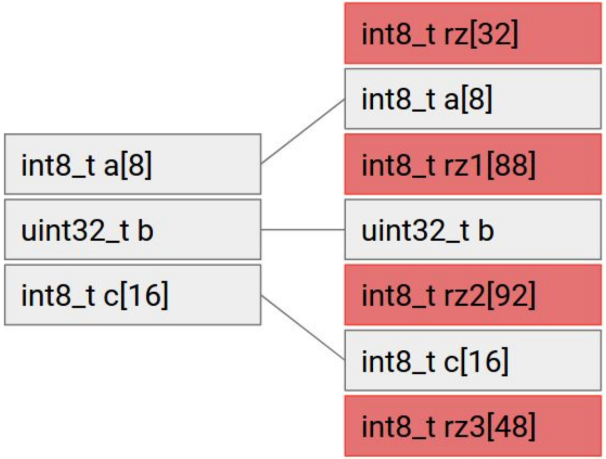
- **Not every** corruption causes a crash
  - Overwriting an unused heap object
  - Overwriting unused “padding” bytes
  - Redirecting to valid instructions
  - **Other weird undefined behavior**

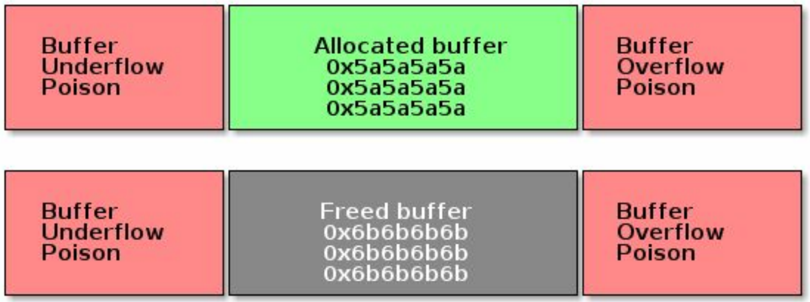
- **A crash-only fuzzing oracle will miss many bugs**

“Better” Oracles

AddressSanitizer (ASAN)

- **Key idea:** inject poisoned "red zones" before and after all memory objects
  - Force a crash when accessing a red zone
  - **Catch all subtle (non-crashing) corruptions**
  - Implement via instrumentation, custom malloc()
  - **Trade-off: over 6x execution overhead**





UndefinedBehaviorSanitizer (UBSan)

- Instrumentation to check for undefined behavior
  - Integer overflows
  - OOB array indexes
  - Illegal shift operations
  - Missing return statements
  - **And many more**
- **Trade-off: more overhead**

```
int main()
{
        int m = std::numeric_limits< int >::max();
        return m + 1;
}
```

↓

```
runtime error: signed integer overflow: 2147483647 + 1 cannot be represented in
type 'int'
```

SystemSan / ExecSan

- Brand-new (Sept. 2022) sanitizer to hunt **command injection** bugs
  - Trace system calls and force crash if seeing weird arguments

This detector currently works by

- Checking if `execve` is called with `/tmp/tripwire` (which comes from our dictionary).
- Checking if `execve` is invoking a shell with invalid syntax. This is likely caused by our input.

CVE-2022-3008 Detail

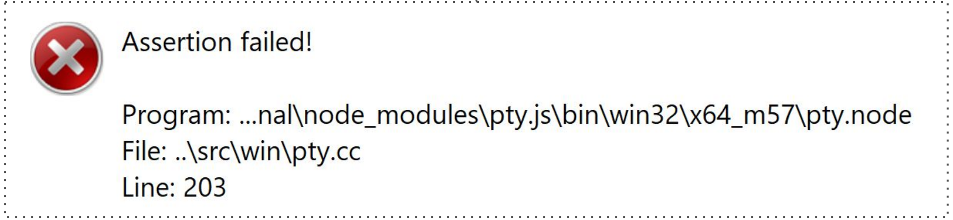
Current Description

The tinygltf library uses the C library function wordexp() to perform file path expansion on untrusted paths that are provided from the input file. This function allows for command injection by using backticks. An attacker could craft an untrusted path input that would result in a path expansion. We recommend upgrading to 2.6.0 or past commit 52ff00a38447f06a17eab1caa2cf0730a119c751

Assertion Violations

- **Checks** on specific variables
  - If satisfied, continue
  - **If violated, force crash**
- Typically added by developer
  - Or automatically mined
- Potential sources:
  - Pre- and post-conditions
  - Likely invariants
  - Input specification

```
while (Record[i] != 0 && i != e)
  KindStr += Record[i++];
assert(Record[i] == 0 && "Kind string not null terminated");
```

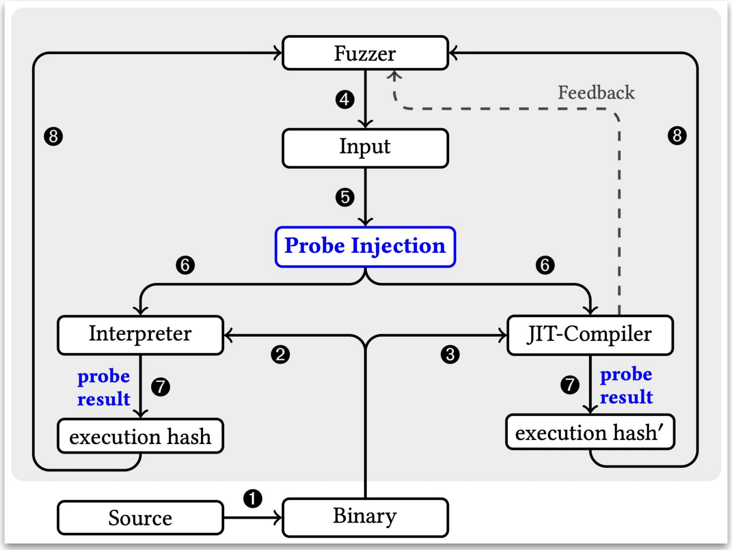


https://blog.regehr.org/archives/1091

Differential Testing

- **Compare multiple implementations** of the same specification
  - Divergence = one (or more) are buggy
  - Compare to a ground-truth implementation
  - Compare with majority voting

- Well-known examples:
  - C++ compilers (CSmith)
  - TLS implementations (FrankenCerts)
  - Interpreters vs. JIT compilers (JIT-Picking)



A note on oracles...

The best oracles are ones that reveal bugs that **you could not find before**.

Questions?

Lab 2

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?