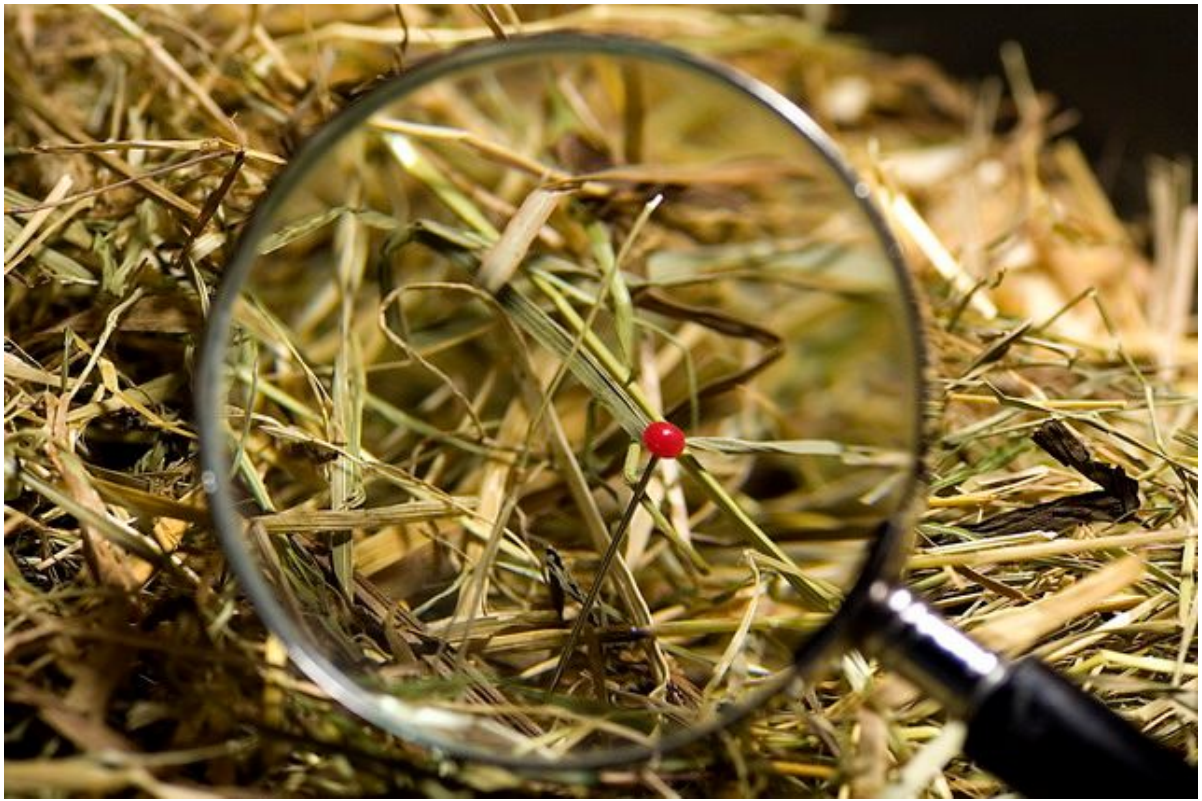
No coverage



Problems of old school fuzzing

- Large search space
- Cannot fuzz specific function
- Hard to fuzz network protocols
- Speed of regular fuzzers (html, css, dom, etc mutators)

Coverage

Goals

- More focused fuzzing
- Faster fuzzing
- Smarter fuzzing
- Easier fuzzer writing

LibFuzzer

New school fuzzing

LibFuzzer

- In-process, in-memory

LibFuzzer

- In-process, in-memory
- Guided fuzz testing

LibFuzzer

- In-process, in-memory
- Guided fuzz testing
- Very effective at a function / protocol level

LibFuzzer

- In-process, in-memory
- Guided fuzz testing
- Very effective at a function / protocol level
- 1000x faster

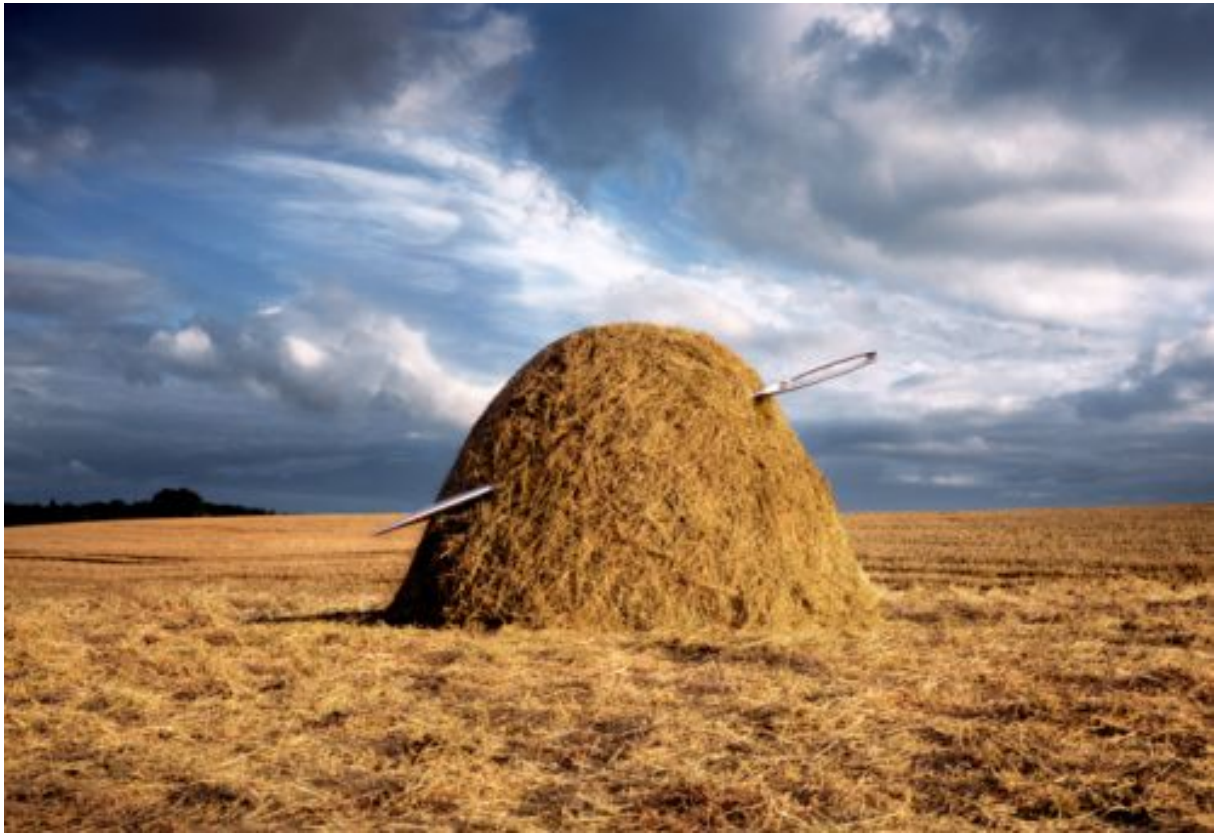
LibFuzzer

- In-process, in-memory
- Guided fuzz testing
- Very effective at a function / protocol level
- 1000x faster
- It’s easy to write a *libFuzzer*-based fuzzer

LibFuzzer

- In-process, in-memory
- Guided fuzz testing
- Very effective at a function / protocol level
- 1000x faster
- It’s easy to write a *libFuzzer*-based fuzzer
- Can be checked along with unit-tests

Coverage-guided fuzz testing



Memory Tools

How to see the invisible

Memory Tools

- **AddressSanitizer** (aka **ASan**)
  - Detects use-after-free, buffer overflows (heap, stack, globals), stack-use-after-return, container-overflow
  - Cpu: 2x, memory 1.5x-3x

- **MemorySanitizer** (aka **MSan**)
  - Detects uninitialized memory reads
  - Cpu: 3x, memory: 2x
  - Special mode: origins

- **UndefinedBehaviorSanitizer** (aka **UBSan**)
  - Detects several classes of bugs (19?), esp on type confusion, signed-integer-overflow, undefined shift, etc.
  - Cpu: 10-50%
  - Memory: ~1x (no allocator, no shadow)

Memory tools: example

- Container-overflow (**ASan**):

```
#include <vector>
#include <assert.h>
typedef long T;
int main() {
  std::vector<T> v;
  v.push_back(0);
  v.push_back(1);
  v.push_back(2);
  assert(v.capacity() >= 4);
  assert(v.size() == 3);
  T *p = &v[0];

  // Here the memory is accessed inside a heap-allocated buffer
  // but outside of the region `[v.begin(), v.end())`.
  return p[3]; // OOPS.
}
```

Let’s write some code

Lessons 03 - 06