# EXERCISE #28

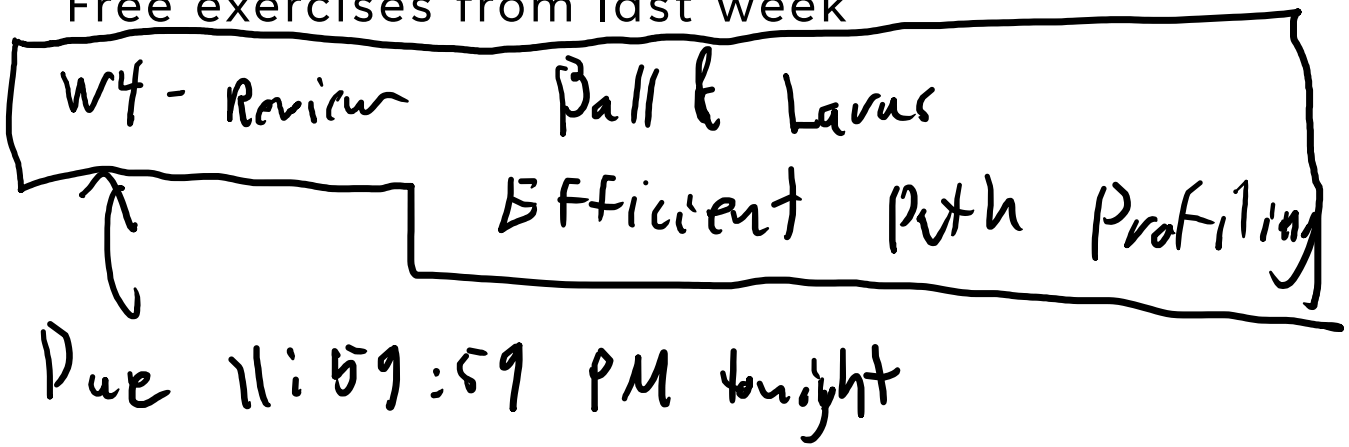
*LLVM INSTRUMENTATION REVIEW*

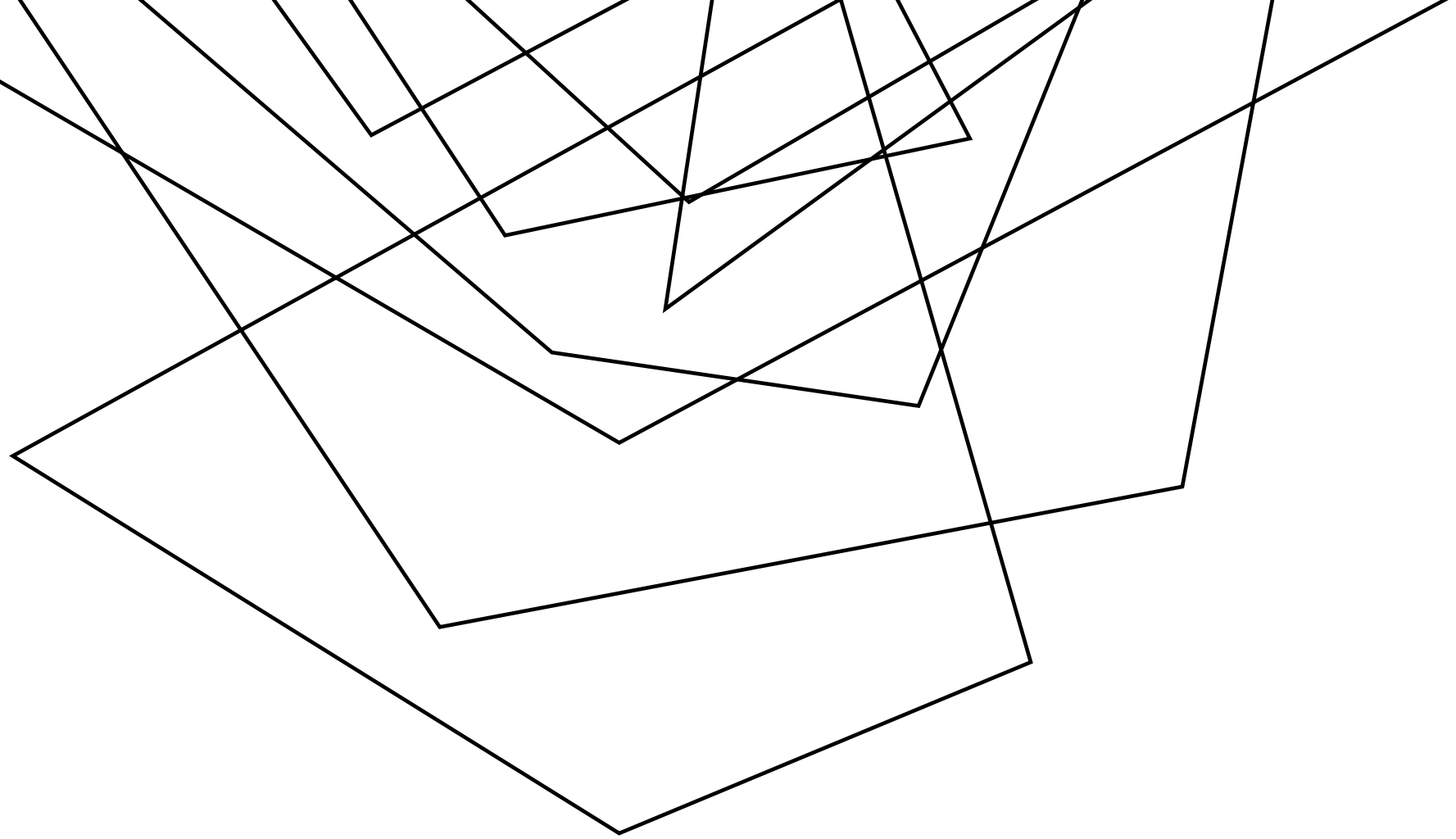
## Write your name and answer the following on a piece of paper

*Describe the difference between the profile-instr-generate and profile-generate options for LLVM instrumentation?*

# ADMINISTRIVIA AND ANNOUNCEMENTS

Free exercises from last week

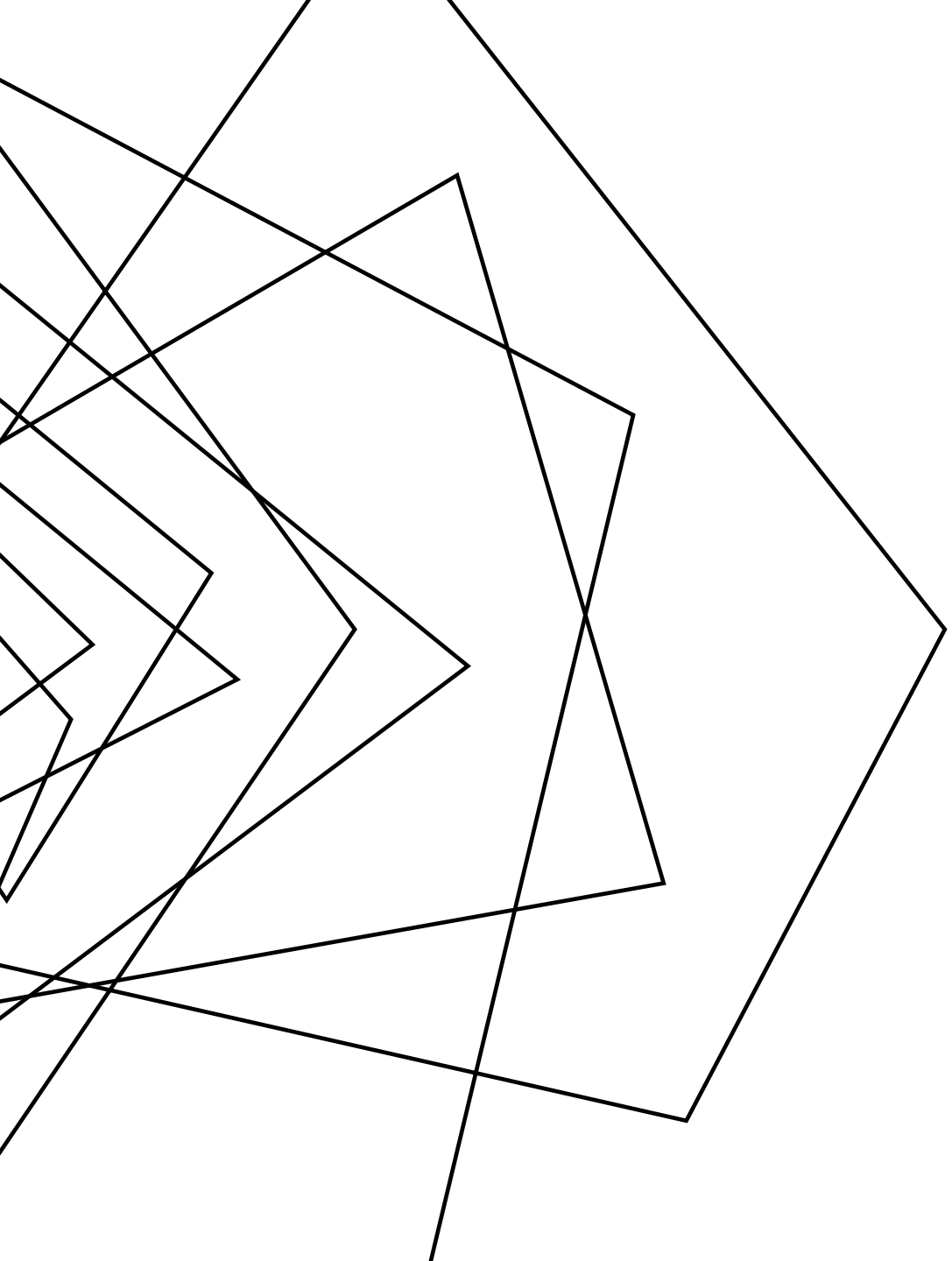




# FUZZING

EECS 677: Software Security Evaluation

Drew Davidson



# WHERE WE'RE AT

DYNAMIC ~~INSTRUMENTATION~~ Analysis

Use the execution of a program to find (security) bugs

Necessarily dependent on encountered execution behavior

# PREVIOUSLY: LLVM INSTRUMENTATION

## REVIEW: LAST LECTURE

### Usage of LLVM built-in instrumentation Analysis

Described commands to use PGO for line coverage analysis

### Setup for a Custom LLVM Analysis

Described the basic infrastructure necessary to craft a custom instrumentation

# THIS LESSON: FUZZING

## OUTLINE / OVERVIEW

```
int main(int argc) {
    return 1/(1-argc);
}
```

### GENERATING GOOD TEST CASES

Cases that increase coverage of program behaviors

Cases that exercise unexpected behavior

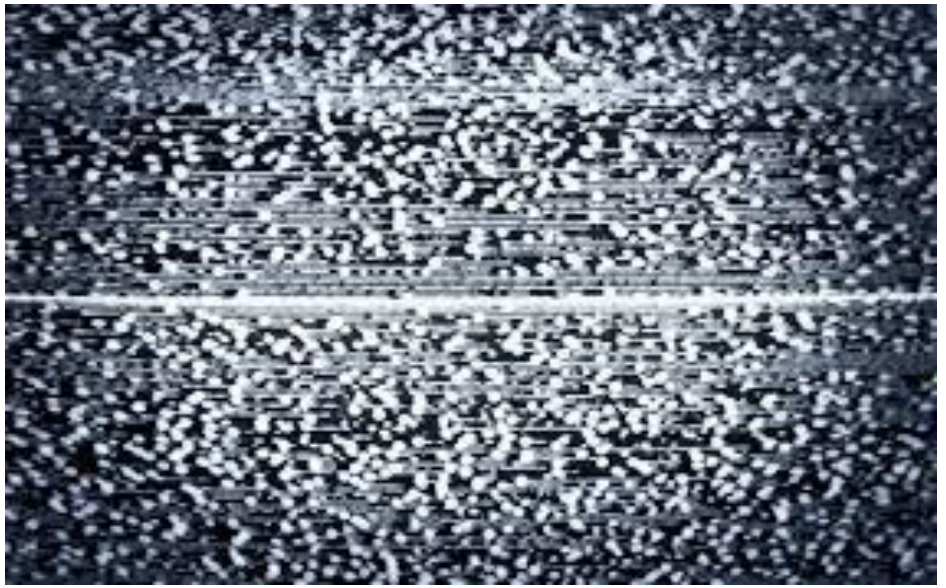
### PREVIOUS STABS AT THIS TOPIC

Consider testing as an intrinsic part of the SSDLC methodology

Test-driven development

Post-hoc evaluation via coverage metrics

### TODAY: JUST GUESS



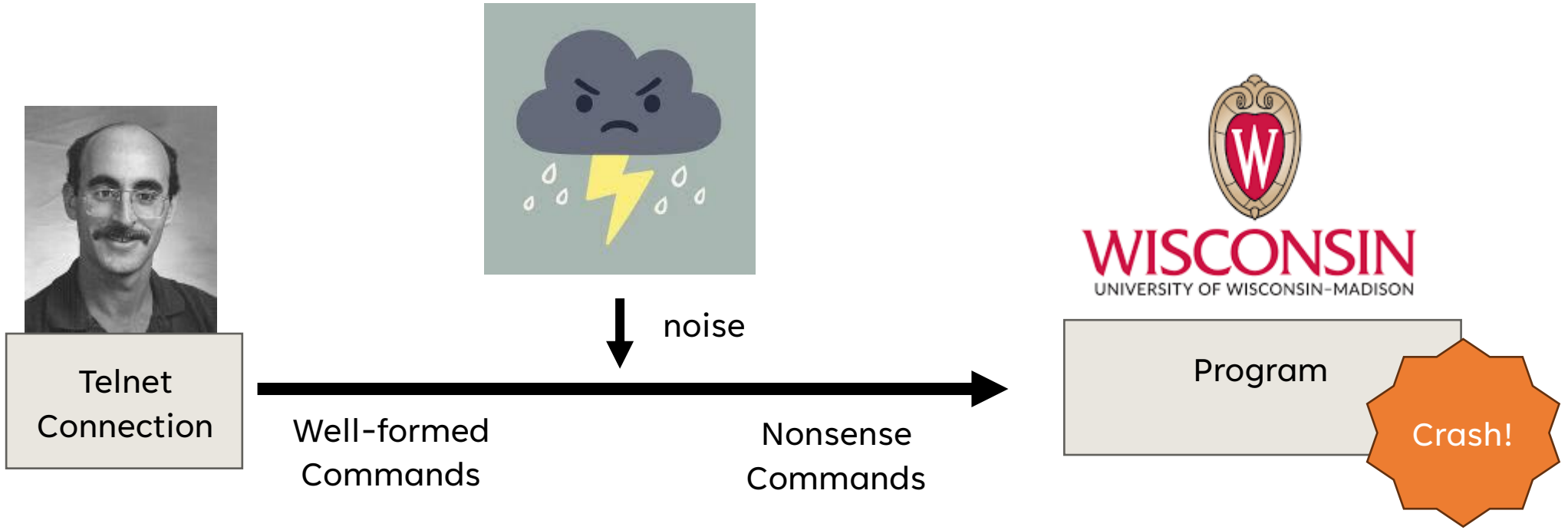
**The random “fuzz” of white noise**

# HISTORY OF FUZZING

## OUTLINE / OVERVIEW

### 1988: It was a Dark and Stormy Night

Professor Bart Miller attempts to work from home...



# BREAKING CIRCULAR LOGIC

## OUTLINE / OVERVIEW

### AUTOMATED TEST CASE GENERATION RESOLVES A FUNDAMENTAL CONFLICT IN TESTING...

Tautologically impossible to predict unpredictable behavior

Apply a technique that obviated the need for expectations

# GRACEFUL FAILURE

## OUTLINE / OVERVIEW

Any error should be anticipated and handled by the system, with an informative error message should recovery become impossible

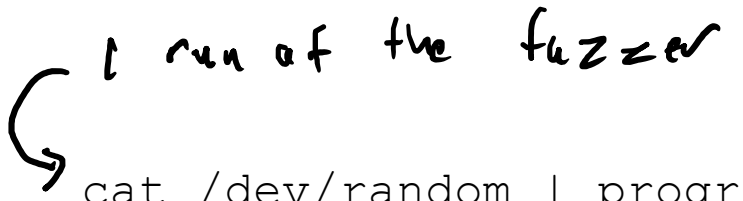
### A KEY PRINCIPLE IN THE VALIDITY OF FUZZING

“The user should never see a seg fault”

# THE SIMPLEST FUZZER

## FUZZ TESTING

### THE MOST BASIC FORM OF FUZZING




**A study in the 90s basically did this, finding bugs in...**
adb, as, bc, cb, col, diction, emacs, eqn, ftp, indent, lex, look, m4, make, nroff, plot, prolog, ptx, refer!, spell, style, tsort, uniq, vgrind, vi

# EXPLORING UNEXPECTED BEHAVIOR

## FUZZING

### Random input is surprisingly effective

Numerous bugs found in practice via fuzzing...

Busybox utilities

Windows bugs

Linux Kernel bugs

### Benefits of fuzzing

Very easy to run

Instant results

Highly scalable

# PRIORITIZING INPUT

## FUZZING

The challenge of fuzzers is (usually) getting past the first validation check

```
if (!sane_input()){
  exit 1;
}
//The rest of the program
```

# SIMPLE TESTING STRATEGY

## FUZZING

### Consider "Interesting" Input

Values close to the maximum, minimum, middle, etc

### Case Study: Card reader input: [Frisby et al., 2012]

# MUTATION-BASED FUZZERS



## EXPLORE DEVIATIONS FROM KNOWN INPUT

**Example mutations:**

*Binary input*

– Bit flips

- Byte flips
- Change random bytes
- Insert random byte chunks
- Delete random byte chunks
- Set randomly chosen byte chunks to interesting values e.g. INT_MAX, INT_MIN, 0, 1, -1, ... §

*Text input*

- Insert random symbols or keywords from a dictionary

# REPRESENTATIVE TOOL: AFL

FUZZING

## AFL (American Fuzzy Lop)

Maintained by Google

## State of the art

Generally considered the best, state-of-the-art fuzzer

# REPRESENTATIVE TOOL: AFL

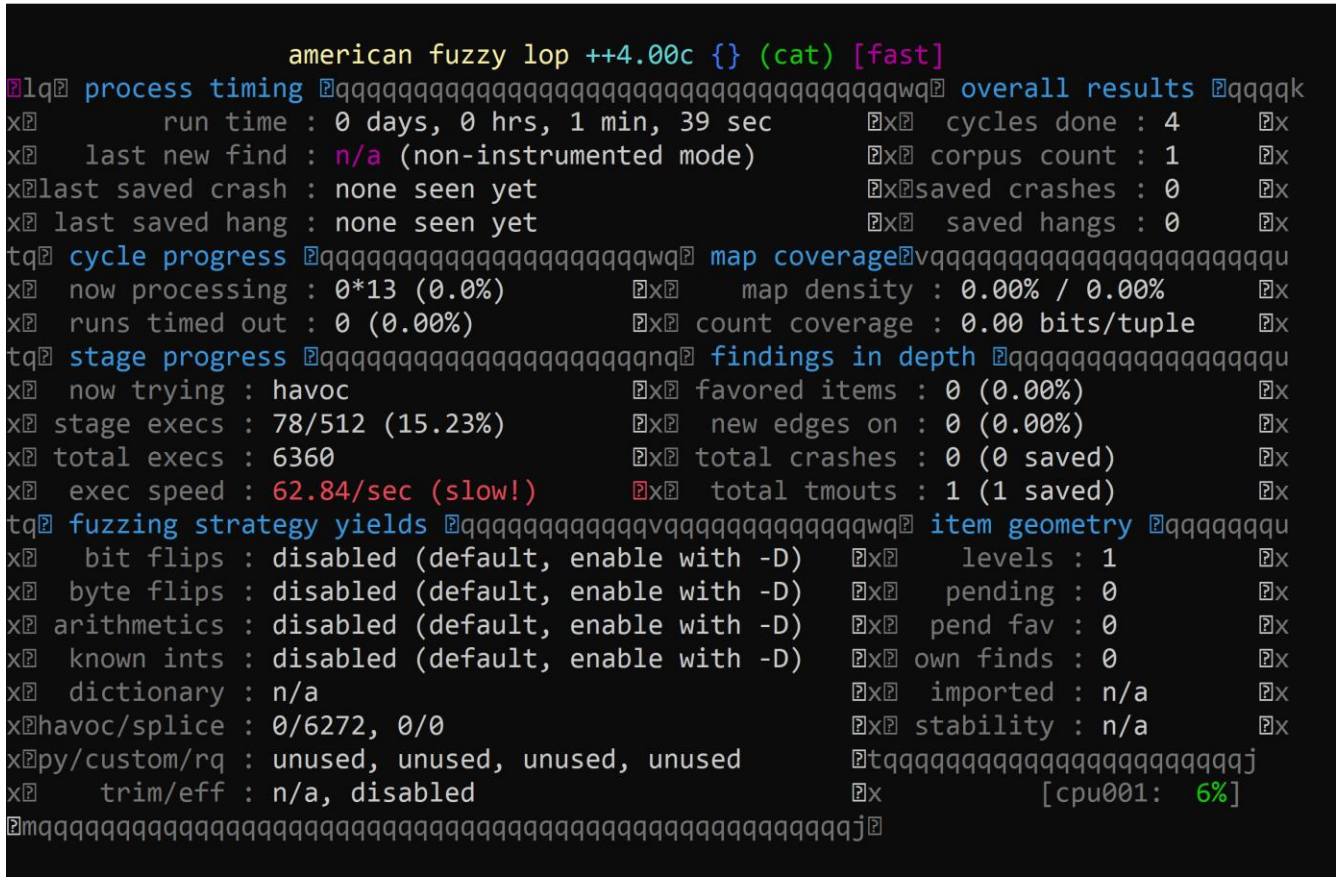
## OUTLINE / OVERVIEW

EXAMPLE COMMAND

"TRADITIONAL FUZZING"

```
mkdir in_dir
echo "hello" > in_dir/hello
afl-fuzz -n -i in_dir -o out_dir cat
```

traditional fuzzing



# REPRESENTATIVE TOOL: AFL

## FUZZING

### INSTRUMENTATION MODE

1) Compile the program with coverage probes
2) Attempt to prioritize / mutate test cases that extend coverage

```
afl-clang++ <build command>
```

↑ command-line-compatible with clang++

# FUZZING ORACLES



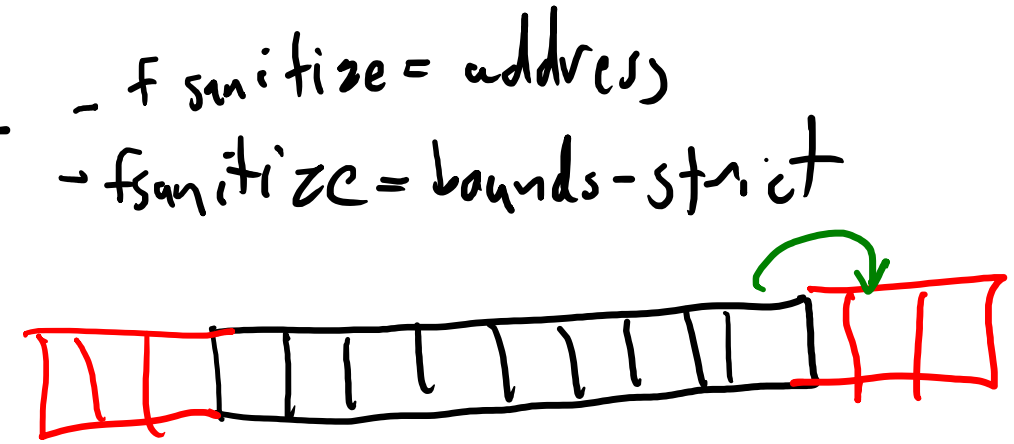
## Beyond graceful Failure

In C/C++ there are a lot of violations of proper behavior that are invisible

“Seems fine until it’s a huge problem”

## Sanitizers

UBSan – Undefined behavior sanitizer
ASan – Address sanitizer
TSan – Thread sanitizer



# RESEARCH DIRECTION: “GUNKING”

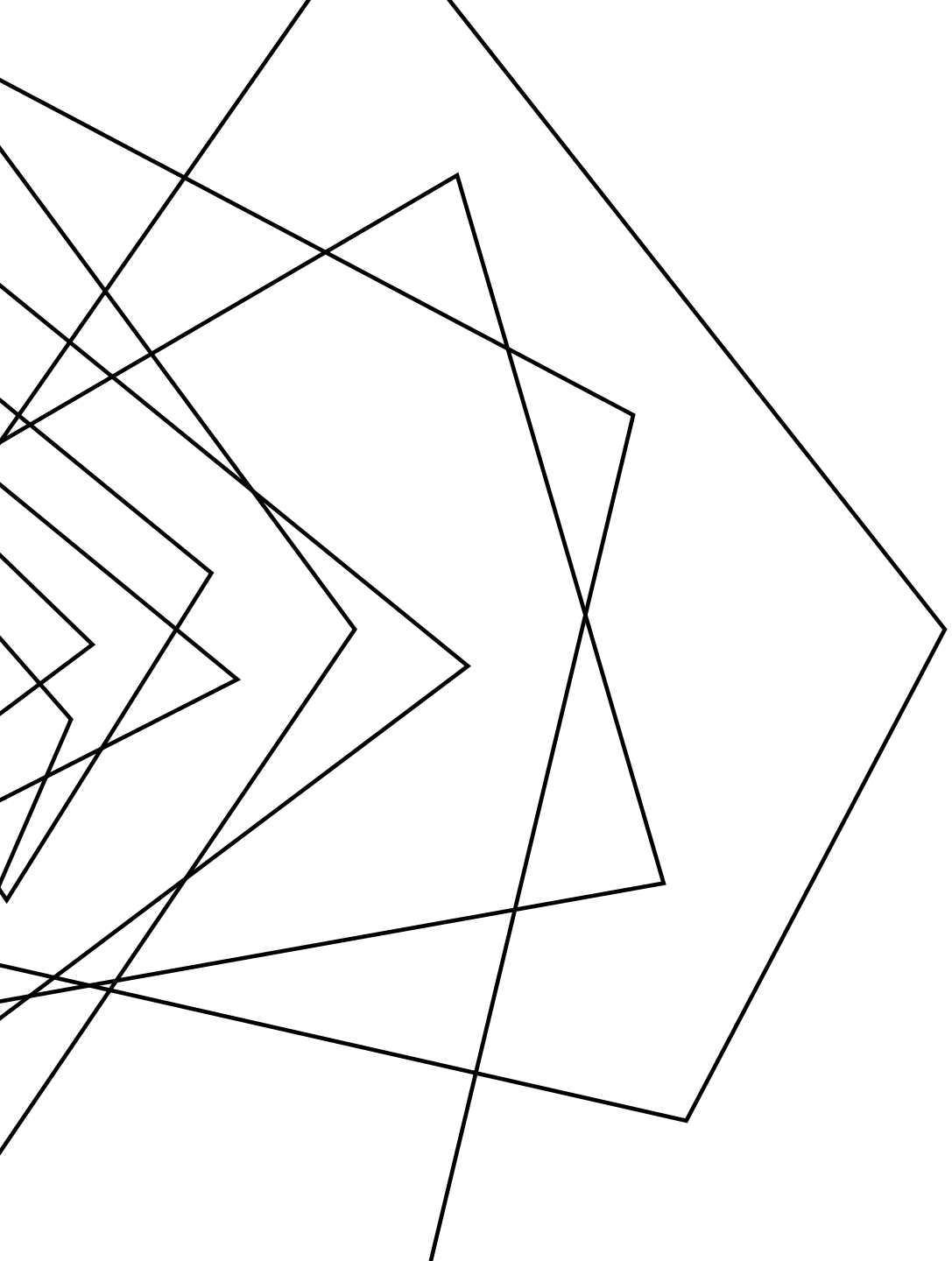
## FUZZING

### Fuzzing as adversarial recon

Fuzzing is so good at finding bugs that even the bad guys do it

### Perhaps a program should deploy Anti-Fuzzing tech

What would that look like?



# WRAP-UP

## INTRODUCED THE CONCEPT AND THE "INDUSTRY STANDARD" TOOL OF FUZZING

A simple, elegant idea