EXERCISE #26

CONTROL FLOW INTEGRITY REVIEW

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

Of the various CFI solutions we explored, none were calling-context sensitive. Why not?

EXERCISE #26 SOLUTION

CONTROL FLOW INTEGRITY REVIEW

TESTING

EECS 677: Software Security Evaluation

Drew Davidson

ADMINISTRIVIA AND ANNOUNCEMENTS

LAST TIME: CONTROL-FLOW INTEGRITY

REVIEW: COMPUTABILITY

Instrument the program to prevent “illegal” jumps

- Intel CET
- Microsoft control-flow guard
- Clang instruction injection

TURNING THE PAGE ON THIS CLASS

NEXT TOPIC

Hopefully, you’ve got a taste of the challenges / benefits of instrumentation and mediation

- Static cost/benefit
- Runtime cost/benefit

TURNING THE PAGE ON THIS CLASS

NEXT TOPIC

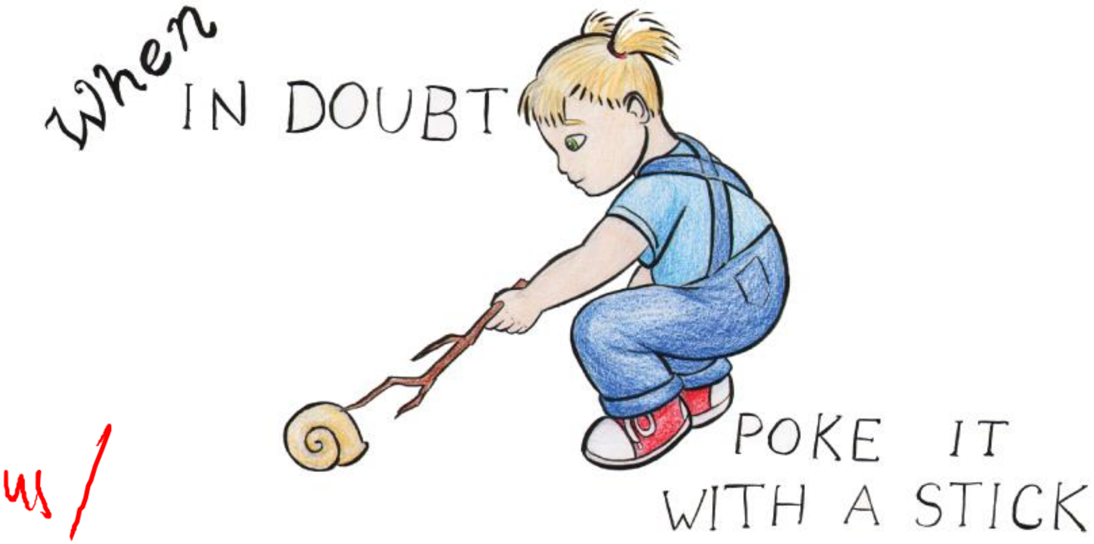
Dynamic Analysis

- Running the program to see what happens

Uses of dynamic analysis

- Program comprehension
- Bug elimination

Detect malicious / undesirable ~~d~~ behavior



LECTURE OUTLINE

- The Testing Perspective
- Test Generation

A FORMULATION OF DYNAMIC ANALYSIS

TESTING

Input/expected output pairs

- Does the program do what it’s supposed to do?

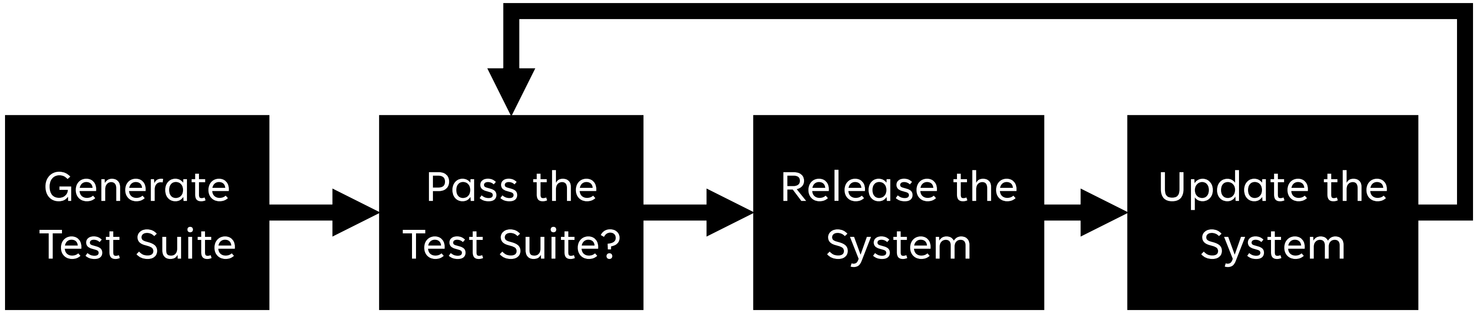
TEST SUITES
TESTING

Ideally, we’ll capture a variety of behaviors

- We’ll refer to the collection of test cases as our test suite

Regression suite

- Capturing sufficient behavior enables capture of breaking changes

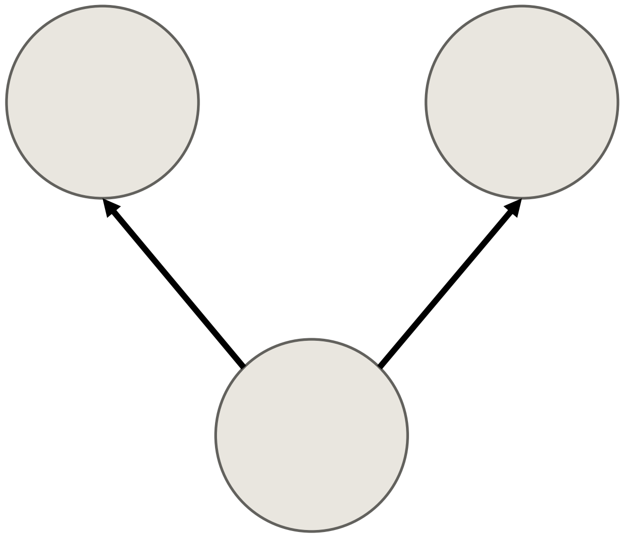


NON-DETERMINISM

TESTING

Factor out external details into the environment

- Time is an input
- Random seed is an input
- Network response is an input



TESTING SCOPE

TESTING

Unit testing

- Testing at the submodule level (e.g. function i/o)

Integration testing

- Testing at the boundary between modules (e.g. library interfaces)

Application testing

- Testing at the whole-program level

PROGRAM VISIBILITY

CATEGORIZING ANALYSIS

White box

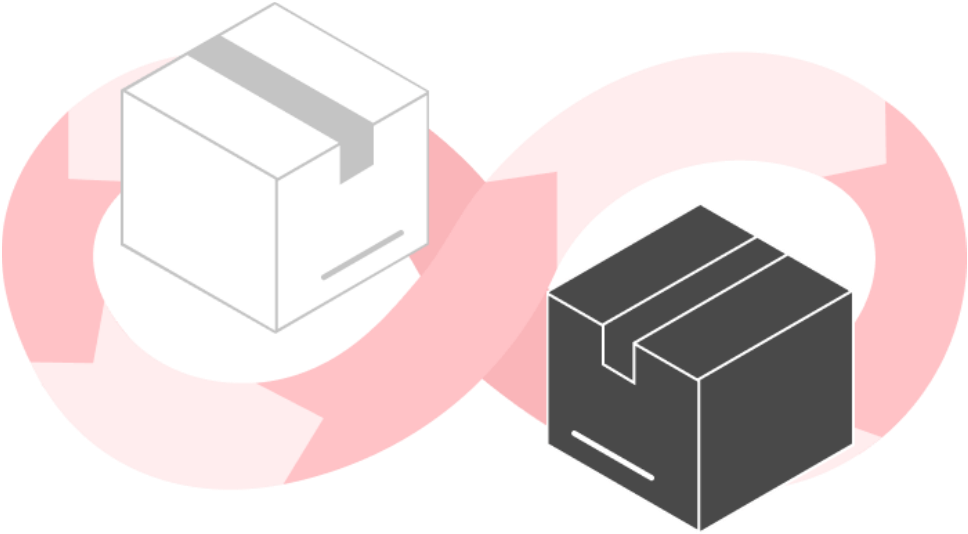
- Testing with “complete” information about the analysis target (typically means source code)

Black box

- Testing with “no” information about how the analysis target is architected (typically means binary only)

Grey box

- Testing with “some” information about how the analysis target is architected (binary + some static analysis / probing)



LECTURE OUTLINE

- The Testing Perspective
- Test generation

CLASSIC LIMITATIONS OF TESTING

TEST GENERATION

It’s hard to predict what might go wrong (presumably you’d have fixed it in this first place)

“FIXING” TESTING

TEST GENERATION

It’s hard to predict what might go wrong (presumably you’d have fixed it in this first place)

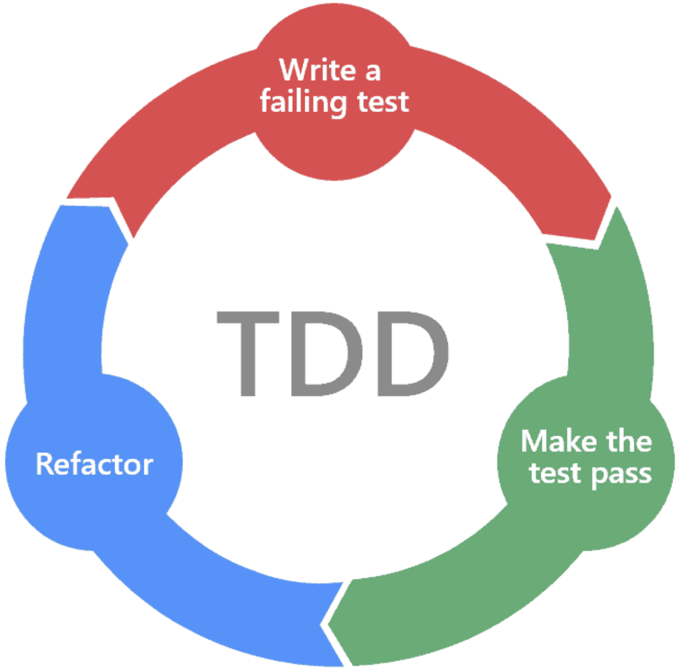
- Could try to make a more intentional correspondence (TDD)
- Could try to leverage tools (Fuzzing)

TEST-DRIVEN DEVELOPMENT

CATEGORIZING ANALYSIS

1. Write a test case (expecting it to fail)
2. Implement enough functionality to pass the test case
3. Fix up the program

(repeat)



TESTING VS STATIC ANALYSIS

TEST GENERATION

A fight (I guess?) in the software engineering community

The clean code blog
"The Dark Path", 1/2017
"Tools are not the Answer", 10/2017

I think that good software tools make it easier to write good software. However, tools are not the answer to the "Apocalypse".

Nowhere in the article did the author examine the possibility that programmers are generally undisciplined.

Ask yourself why we are trying to plug defects with language features. The answer ought to be obvious. We are trying to plug these defects because these defects happen too often.

Now, ask yourself why these defects happen too often. If your answer is that our languages don't prevent them, then I strongly suggest that you quit your job and never think about being a programmer again; because defects are *never* the fault of our languages. Defects are the fault of *programmers*. It is *programmers* who create defects – not languages.

And what is it that programmers are supposed to do to prevent defects? I'll give you one guess. Here are some hints. It's a verb. It starts with a "T". Yeah. You got it. *TEST!*

SOME DIFFICULTIES OF UNIT TESTING

TEST GENERATION

Integrating testing into a workflow

googletest

```
apt install googletest
apt install libgtest-dev
```

SOME DIFFICULTIES OF UNIT TESTING

TEST GENERATION

What to do about a function’s “environment”?

FUZZING

CATEGORIZING ANALYSIS

Automatically creating test cases