JOE ROZNER / @JROZNER

RE-TARGETABLE GRAMMAR BASED TEST CASE GENERATION

TESTING PARSERS IS HARD

HOW WE GOT HERE

- Mostly clean room (ish) implementation of complex languages (context-free-ish)
- ~35k lines of grammar in total (ANTLR)
- Implemented from incomplete, inaccurate, and contradictory documentation
- Radically different parsing algorithm(s) from original implementations
- Lack of public test cases for most dialects

ARE WE CORRECT?

- What is correct?
- Can we be 100% correct?
- How do we quantify how correct an implementation is?
- How do we test the implementations?
- How do we get better?

GETTING MORE TEST CASES?

- Request query logs from customers
- Stand up applications and record their queries
- Automatically generate test cases with a fuzzer

PROBLEMS WITH TRADITIONAL TEST CASE GENERATION

- Inflexibility with using test cases
- Inflexibility with providing feedback
- Existing tools solve many cases but as you deviate they become less useful

STYLES OF FUZZING

INSTRUMENTATION + RANDOM MUTATION

- Focus on path exploration and code coverage
- No concept of syntax/semantics
- Wont necessarily provide lot’s of coverage for variations of a specific parse tree
- Might spend a lot of time on uninteresting/non-relevant code paths
- Not immediately clear how to build a proper test harness
- Example of this strategy is AFL (American Fuzzy Lop)
- https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html

“THE FIRST IMAGE, HIT AFTER ABOUT SIX HOURS ON AN 8-CORE SYSTEM…”

“...CERTAIN TYPES OF ATOMICALLY EXECUTED CHECKS WITH A LARGE SEARCH SPACE MAY POSE AN INSURMOUNTABLE OBSTACLE TO THE FUZZER...”

```
if (strcmp(header.magic_password, "h4ck3d by p1gZ”))
  goto terminate_now;
```

“IN PRACTICAL TERMS, THIS MEANS THAT AFL-FUZZ WON'T HAVE AS MUCH LUCK ‘INVENTING’ PNG FILES OR NON-TRIVIAL HTML DOCUMENTS FROM SCRATCH…”

INSTRUMENTATION + SOLVING

- Focus on path exploration and code coverage
- Instrument the code and solve for new paths
- Still doesn’t care about syntax/semantics
- Still not clear how to build a more customer test harness
- Not necessarily easy to gate off specific paths that are uninteresting
- Example of this is KLEE

GRAMMAR BASED

- Uses a grammar to generate syntactically correct sentences
- Typically provide their own grammar language
- Mostly targeted at regular/context-free text based languages
- Example of this is Mozilla Dharma
- https://github.com/MozillaSecurity/dharma

LET'S ITERATE

- Create a platform with parser primitives that can generate instead of parse
- Provide support for multiple frontends so manual translation from one grammar language to another is not required
- Be expressive enough for regular, context-free, and context-sensitive languages both text and binary
- Embeddable and usable from any language
- Composable and flexible

STRUCTURE

- Composable libraries
- Target + Generation
- Frontends
- Test harnesses

DEMO

WHAT'S NEXT?

- Expose a C-compatible API
- Starting work on frontends
- Better negation logic
- Context-Sensitive/Introspective generators
- Functional comparison between results with traditional fuzzers

JOE@DEADBYTES.NET / @JROZNER

QUESTIONS?