Improving Grey-Box Fuzzing by Modeling Program Control Flow

Siddharth Karamcheti, Gideon Mann, David Rosenberg
Bloomberg - CTO Data Science

- *Use Machine Learning to Intelligently Hunt Bugs in Programs*

- *Learn Models of Program Behavior & Exploit Them*

Fuzzing – A Background

- **Fuzzing** is a technique for automated software testing.
  - *Core Idea:* Provide programs with unexpected inputs, with goal of finding bugs, maximizing coverage, etc.

- Different types of fuzzers and testing tools:
  - **Black-Box:** Assume no transparency into program.
    - Random generators (good for testing parsers)
  - **White-Box:** Assume lots of transparency (e.g. KLEE)
    - Suffer from path explosion as programs get big

Grey-Box Mutational Fuzzers & AFL

- **Grey-Box Mutational Fuzzers**
  - Mutate existing seed(s) to generate new test inputs
  - Light instrumentation, to check code paths
  - Execute test input, add to set of seeds if new behavior
- AFL is best of the bunch!
  - Uses heuristics to pick inputs
  - Randomness for mutations

```
// Core Algorithm for American Fuzzy Lop (AFL)
// time: Fixed time window to fuzz (e.g. 24 hours)
// queue: Queue of inputs that exercise new code paths.
while time has not elapsed do
    parent, energy ← pick_input(queue)
    for i ∈ range(energy) do
        child ← parent
        for j ∈ 1 to sample_num_mutations() do
            mutation ← sample_mutation()
            site ← sample_mutation_site()
            child ← apply_mutation(mutation, child, site)
        end for
        path ← execute_path(child, code)
        if (path is new) then queue ← child
    end for
end while
```

Related Work - Improving Grey-Box Fuzzing

AFL relies on heuristics, randomness, and scale to find bugs

Q: Can we be more efficient?

A: Yes, with **data-driven control**!

```
// Core Algorithm for American Fuzzy Lop (AFL)
// time: Fixed time window to fuzz (e.g. 24 hours)
// queue: Queue of inputs that exercise new code paths.
while time has not elapsed do
    parent, energy ← pick_input(queue)
    for i ∈ range(energy) do
        child ← parent
        for j ∈ 1 to sample_num_mutations() do
            mutation ← sample_mutation()
            site ← sample_mutation_site()
            child ← apply_mutation(mutation, child, site)
        end for
        path ← execute_path(child, code)
        if (path is new) then queue ← child
    end for
end while
```

AFLFast
Böhme et al, 2016

Fuzzing by
Thompson Sampling
Karamcheti et al, 2018

FairFuzz
Lemieux and Sen, 2017

... and many many more!

Problem: Wasteful Executions

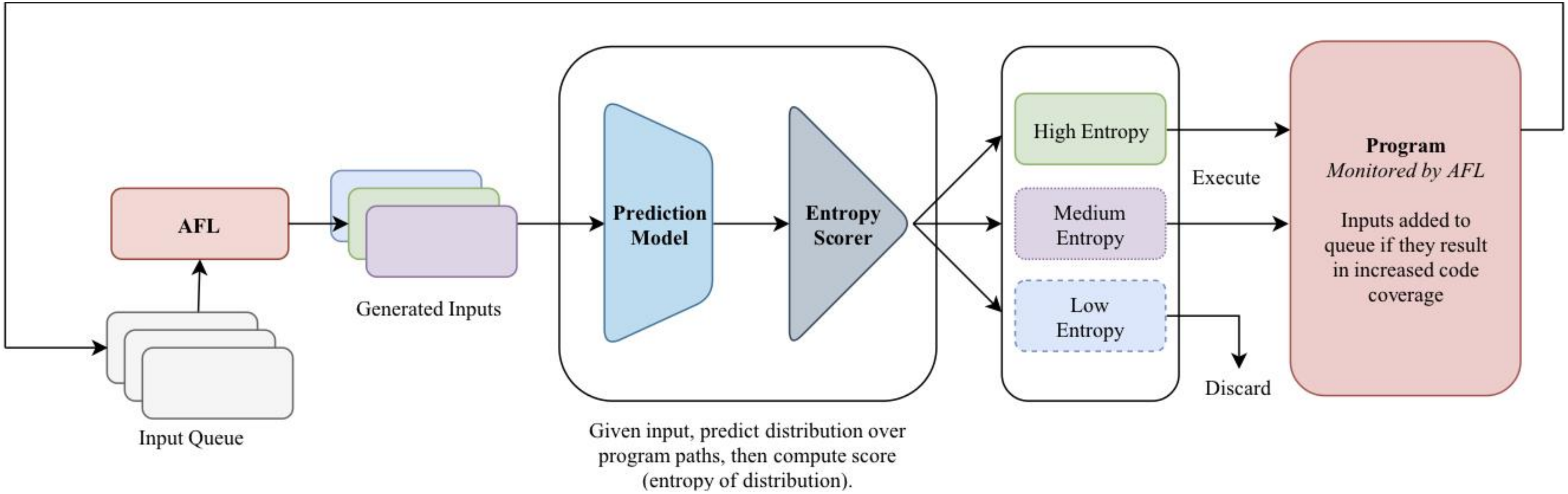
- **Generating millions -> billions of inputs and executing all of them**
  - Fine when program execution is cheap/fast
  - Of billions of inputs generated, maybe only a couple hundred are useful!

- **Let's reframe AFL...**
  - Break into two phases
    i. **Generate:** Generate a large number of inputs without execution
       - Lightning Fast - just string manipulation
    ii. **Execute:** Pick some subset of these inputs to execute
       - Expensive - depends on the program under test

Q: How do we learn which inputs to execute?

A: Focus on inputs we can’t characterize or understand!

Fuzzing by Modeling Program Behavior



Learning Prediction Models

- **Learn mappings of inputs (strings) to program paths!**
  - For every input AFL executes, it stores the set of control flow graph edges traversed.
  - Treat string input as **x**, set of edges as **y**
  - Fit a model using a classifier of your choice —> just needs to output probabilities!
  - Can update model online! After executing new inputs, train on resulting data!
- In our work, we *keep it simple*
  - *Featurizer:* Bag of Bytes (0 - 255) for encoding input strings
  - *Classifier:* Logistic Regression

Critical Point: Ranking Inputs by Entropy

- **We have a model that maps inputs to a distribution over control flow paths**
  - What now?

- Run generated inputs through our model!
  - Look at the probability distribution over possible code paths
    - If high-entropy - *crucial input* - means our model isn't sure
      - *Two Benefits to Execution*
        - Could traverse new code path (or hit a bug!)
        - Improve our model by re-training
    - If low-entropy - *discard* - we know it is redundant

Framework Pseudocode:

```
// Algorithm for AFL + Program Modeling
// afl: Instance of AFL for generating/executing inputs
// iterations: Fixed number of generation iterations
// num_generate: New inputs to generate each iteration
// α: Fraction of generated inputs to execute each iteration
// queue: Queue of inputs that exercise new code paths
// model: Predicts distribution over execution paths for an input
// ranker: Given predictions, ranks by entropy values (high - low)
for i ∈ range(iterations) do
    generated ← afl.generate(queue, num_generate)
    execute := []
    for g ∈ generated do
        execute ← model.predict(g)
    end for
    execute ← ranker.rank(execute)
    for j ∈ range(α· num_generate) do
        queue, path ← afl.execute(queue, execute[j])
        model ← model.retrain(execute[j], path)
    end for
end for
```

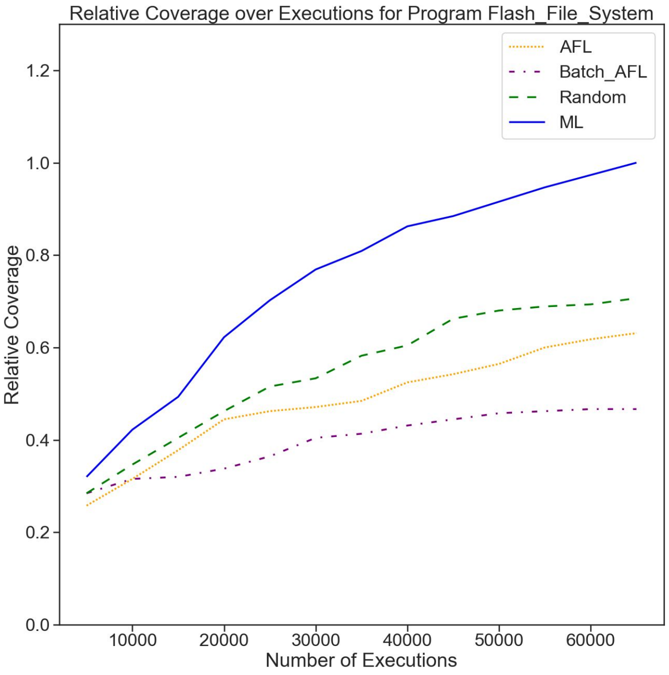
Preliminary Experiments - Datasets

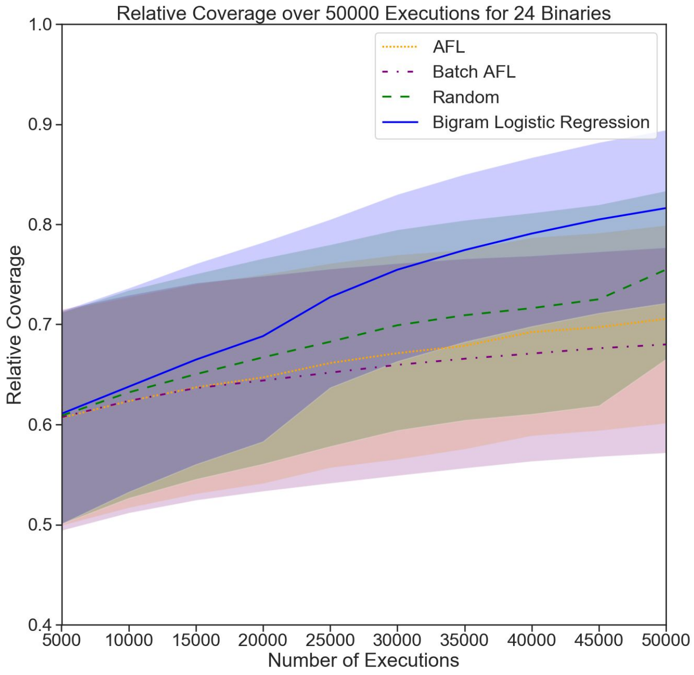
- **We use the DARPA Cyber Grand Challenge Binaries**
  - Set of 200 binaries released for testing bug discovery + patching
  - Each binary has a bug added by a human user (meant to be somewhat "realistic" as opposed to synthetically injected bugs)
  - Due to time constraints, we evaluate on a subset of 24 of these binaries

Preliminary Experiments - Baselines

- **We use the following procedure across all experiments:**
  - Run AFL for 3 minutes to warm-start/populate queue.
  - Start the following 4 Strategies using the resulting queue:
    i. **AFL:** Generates an input then immediately executes
    ii. **Batch-AFL:** Meant to mimic our program modeling procedure.
       - Generate 50,000 inputs, execute the first 5000 (slightly different than standard AFL due to the heuristic sampling seed inputs)
    iii. **Random Batch-AFL**: Generate 50,000 inputs, select random 5000
    iv. **ML/Logistic Regression:** Generate 50,000, use model to pick 5000

Results on 24 CGC Binaries

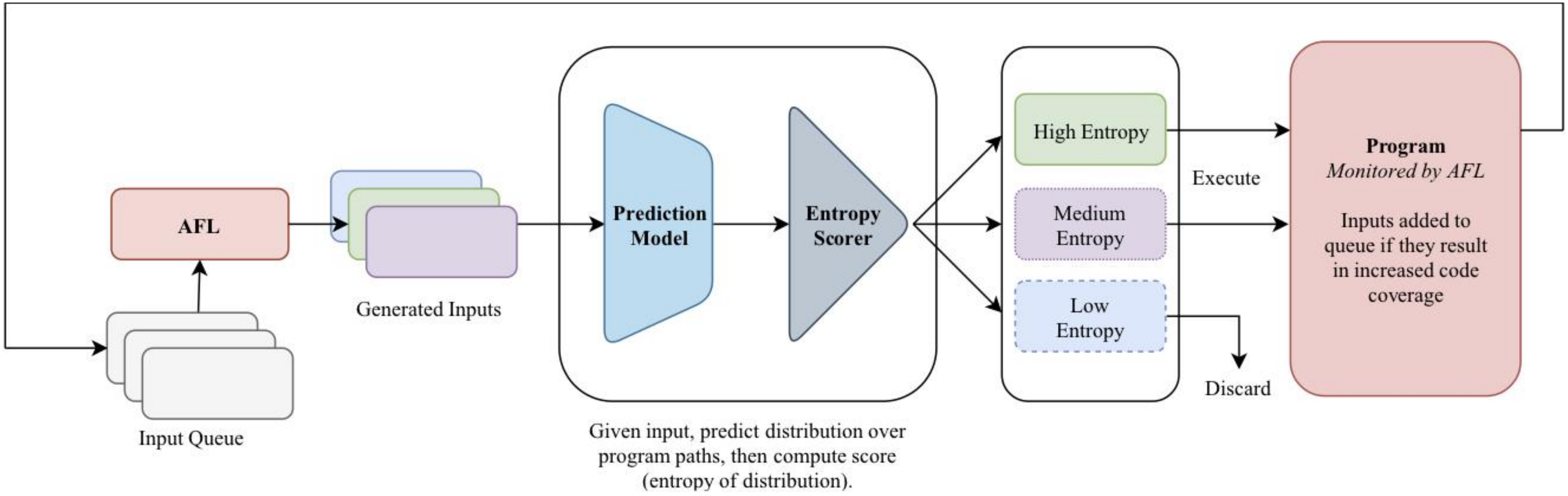




True Wisdom is Knowing What You Don’t Know

–Confucius

Summary



Questions: Email me @ sidd.karamcheti@gmail.com

Thanks for listening!