Australian Government
Department of Defence
Science and Technology

Fuzzing Corpus Optimisation

Moonwalking with Moonbeams

Shane Magrath

Defence Science and Technology Group

July 15, 2018

Outline

1. Introduction
2. Corpus Optimisation Theory
3. Moonshine Results
4. Moonbeam
5. Conclusion

$ who

Team

- Liam Hayes, Johnathon Milford, Duy Khuu, Adrian Herrera
- Joint work between DSTG and ANU

$ whoami

- Researcher with the **Defence Science and Technology (DST) Group**
- Visiting researcher at the **Australian National University**
- Interested in *Vulnerability Discovery and Mitigation*
  - mainly high performance fuzzing...

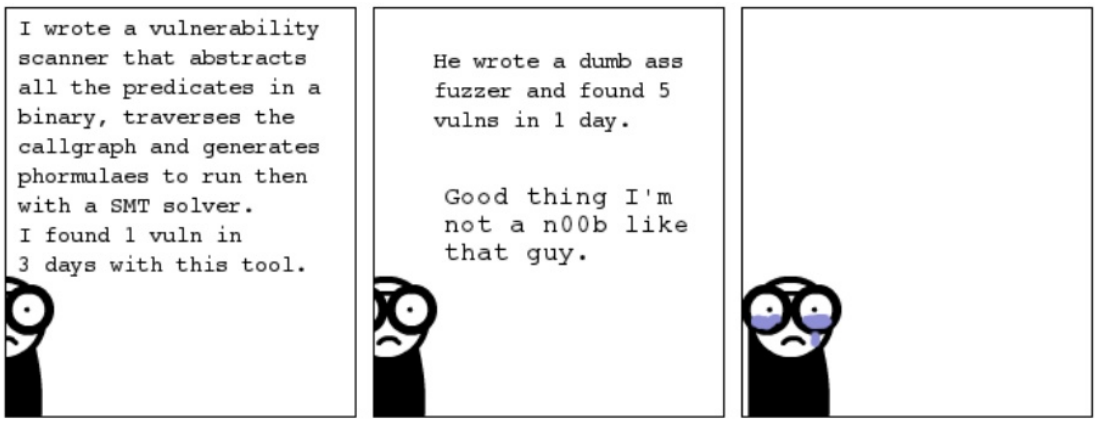


Figure 1: State of the Art

Introduction

Background

- **Fuzzing**: subject the program under test (PUT) to randomised input in the hope of producing a crash
- Different types of fuzzing *strategies*:
  - **Black Box**: High speed random *mutational* tests generated from a fuzzing corpus
    - **Example: Cisco Mutiny**
  - **White Box**: Use program analysis tools like symbolic execution on the PUT to *generate* high quality tests
    - **Example: Microsoft Sage**
  - **Grey Box**: Instrument the PUT to guide future fuzzing through test case *generation*
    - **Example: Google AFL**

Background

Different fuzzing strategies have *different*:

- **Use Cases**
  - Do you have source code?
  - Is it protocol fuzzing, file fuzzing, kernel fuzzing, ...?
  - Do you have a grammar/protocol specification for the PUT input?
  - Can instrument the binary by ”lifting” an IR, instrument and recompile?
  - ... so many ...

Background

Different fuzzing strategies have *different*:

- Pros and Cons
  - **Iteration Rate Performance**:
    Different fuzzers perform orders of magnitude differently
  - **Faulting Observability**
    Fuzzers differ in what they can see in terms of process corruption
  - **Usability**
  - **Triage Support**
    What happens after the campaign?
  - … so many …

Background

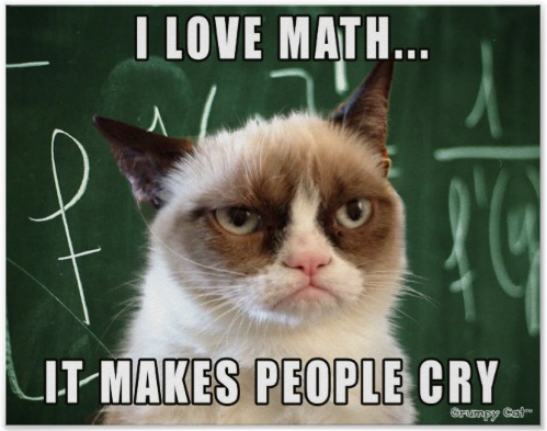
- This talk is about **Corpus Optimisation**
- Corpus Optimisation is a *pre-fuzzing* operation
  - We call it `"distillation"`
- Important for most types of fuzzing...
  - *Very* important for black box mutational fuzzing to reduce effective test case duplication

Corpus Optimisation Theory

Corpus Problems

- **Objective**: Vulnerability assessment of a *closed source proprietary version* of libPNG (say)
- **Strategy**: Block box mutational fuzzing
- **Corpus Requirement**: 1,000,000 examples of **.png** files
- **Problem**: 999,999 of the exemplars are cat photos!

Lack of diversity in the fuzzing corpus results in **lots** of test duplication



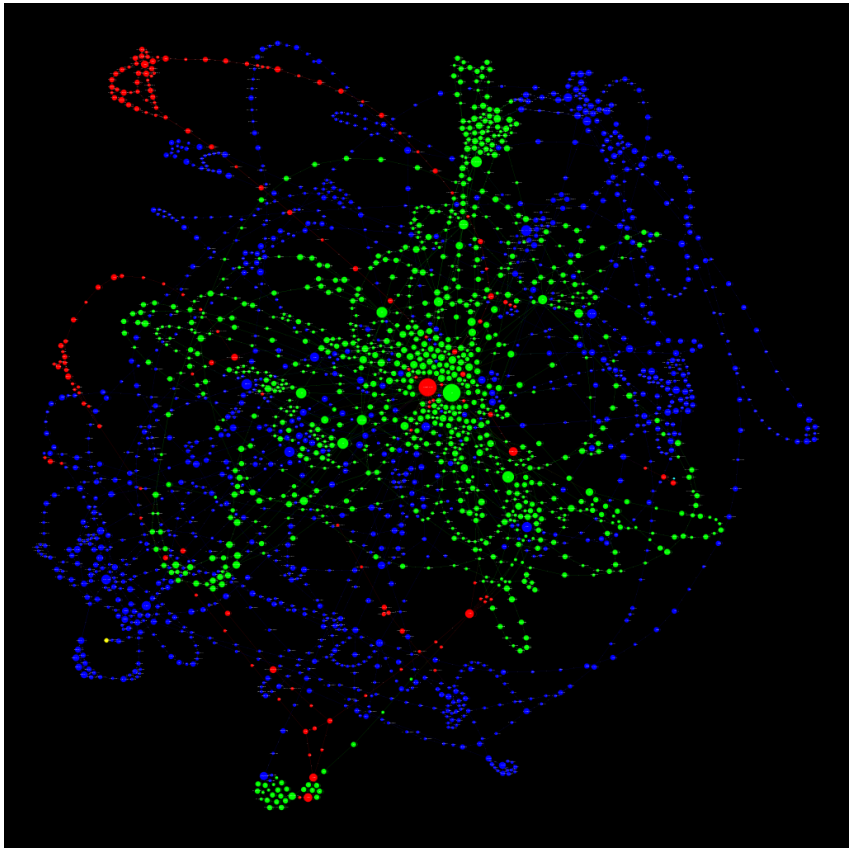
Corpus Problems

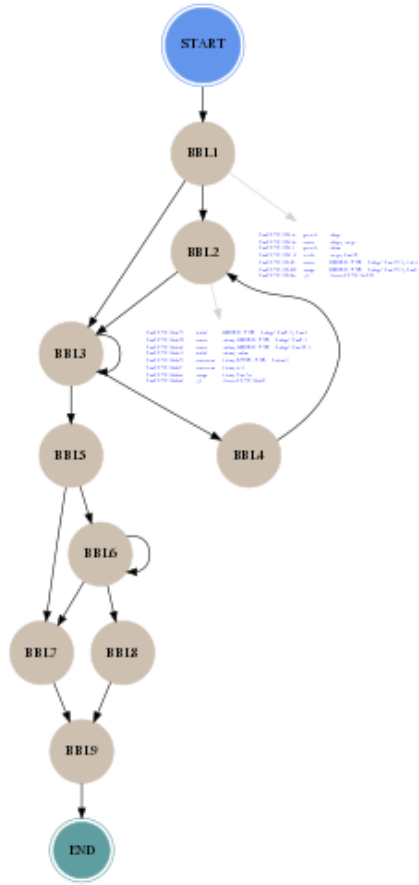
Problem: How do we measure a seed in order to compare it to other seeds?

Answer: We **trace** it using *dynamic analysis* tools

Aside: Program Execution and Basic Blocks

Figure 2: Basic Block Example: `TIFFInfo` Call Graph Trace





Working Example

Working example involves the following scenario:

- corpus of **5 seeds**
- dynamic analysis reveals **7 basic blocks**
- we have a coverage trace for each seed

A typical fuzzing campaign will

- order of $\mathbf{10^5}$ seeds
- order of $\mathbf{10^5}$ to $\mathbf{10^6}$ basic blocks

Problem Statement

An **optimum corpus** (*"distilled"*) has the following two properties:

- contains the smallest subset of seeds (rows) that still
- covers all the basic blocks (columns)

Things to Note

- This is called a **(weighted) minimum set cover** problem
- This problem is **NP-Hard**
- Current practice is to use **greedy methods** to compute an approximately optimal set cover
- We do better using **approximate dynamic programming**

Working Example

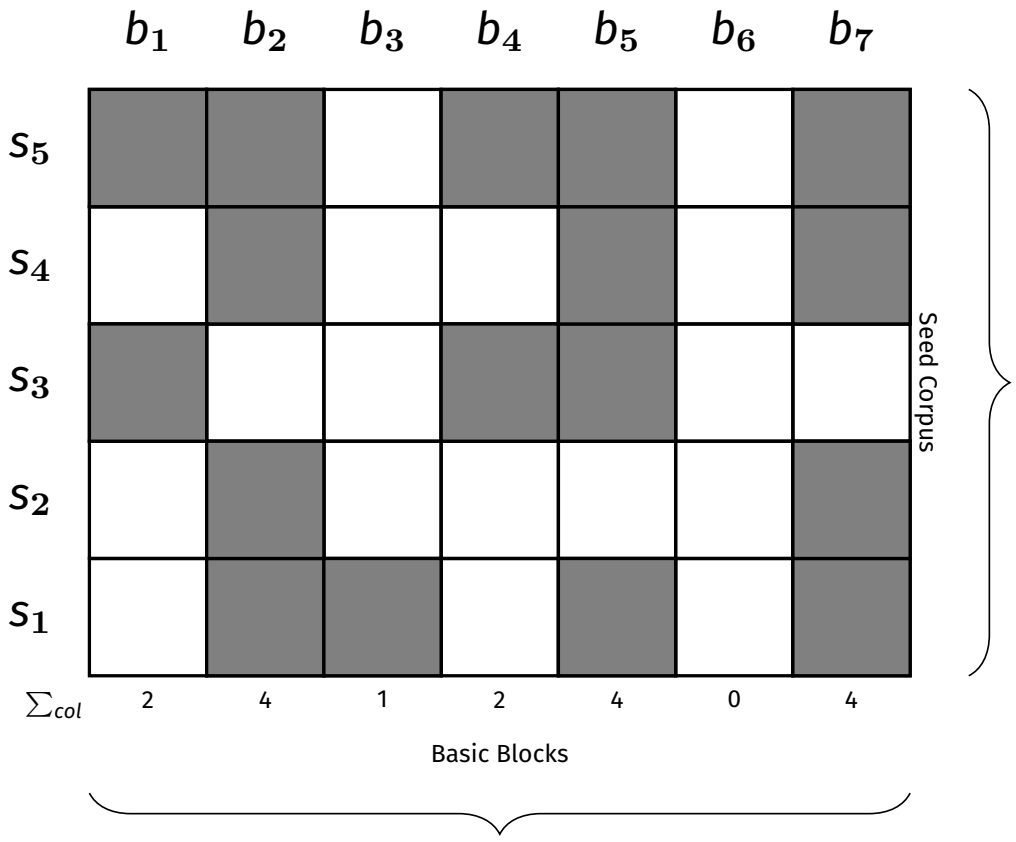


Figure 3: Coverage Matrix Working Example

Remove all Column Singularities

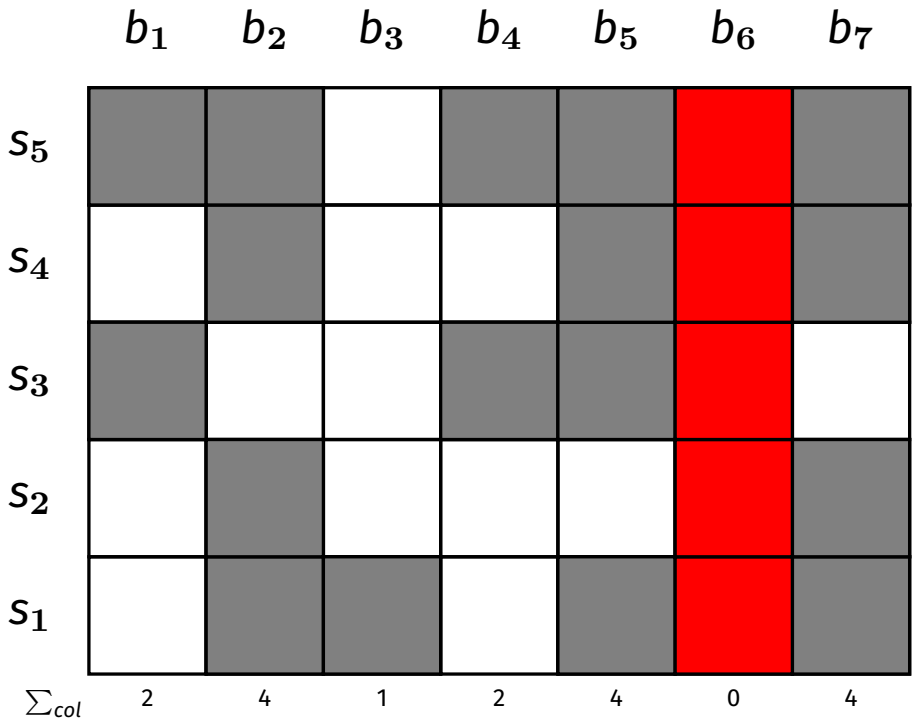


Figure 4: Singularity Example

Select All Exotic Rows

Working Solution $= \{s_1\}$

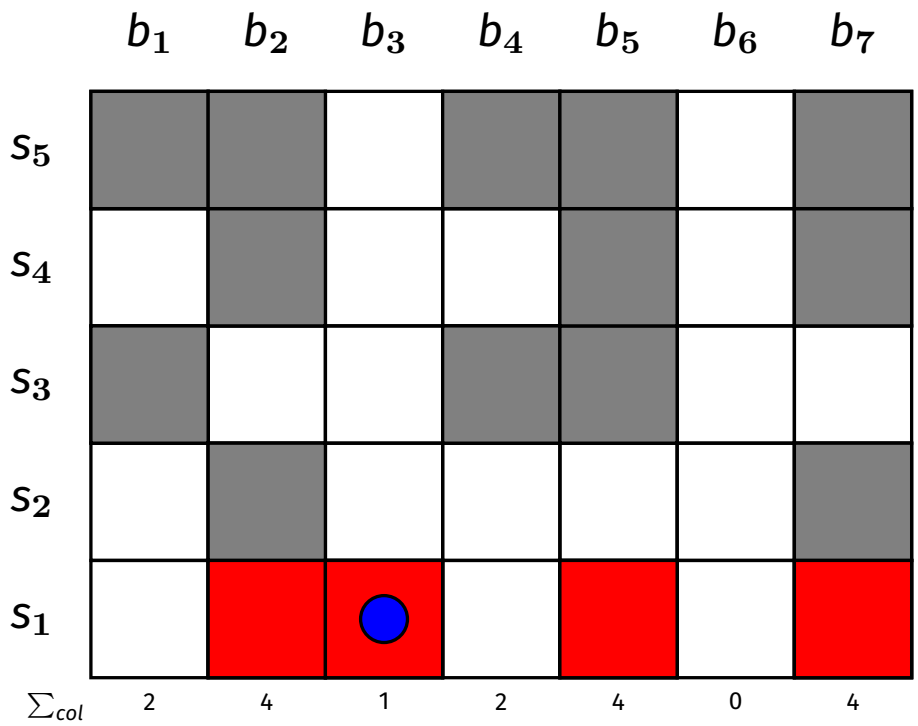


Figure 5: Exotic Row Example

Select All Dominant Rows; Delete All Submissive Rows

Working Solution $= \{s_1, s_5\}$

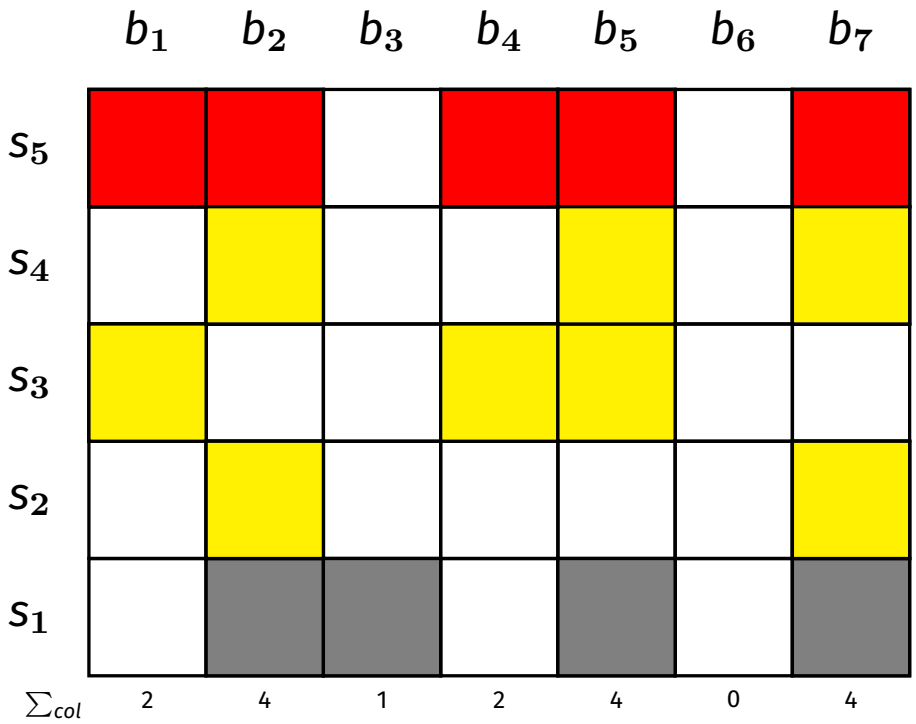


Figure 6: Row Dominance Example

Remove All Dominant Columns

Working Solution $= \{s_1, s_5\}$

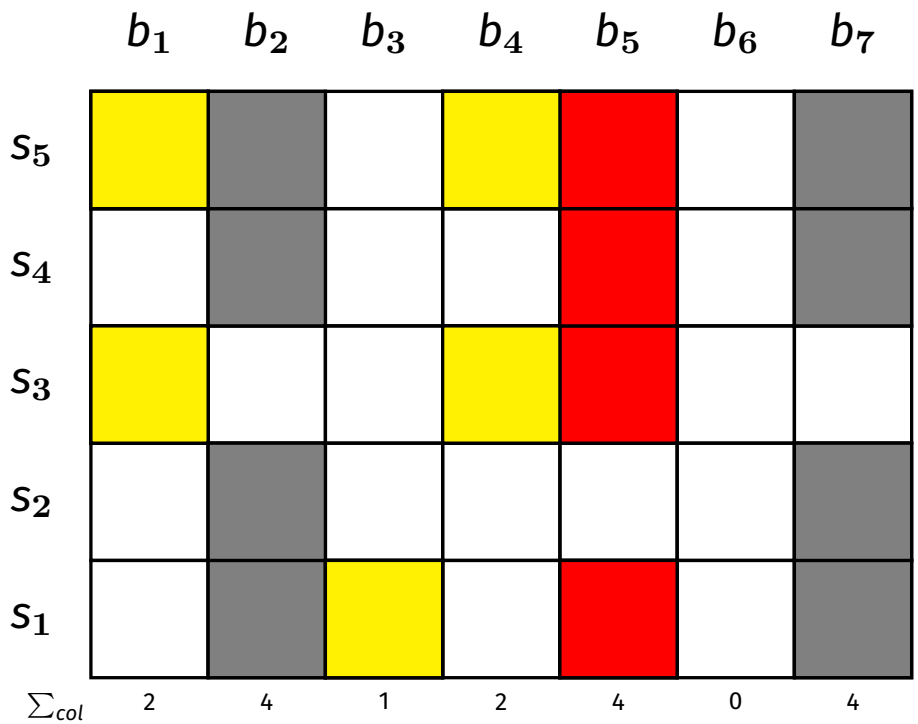


Figure 7: Column Dominance Example

Remove All Contained Columns

Working Solution $= \{s_1, s_5\}$

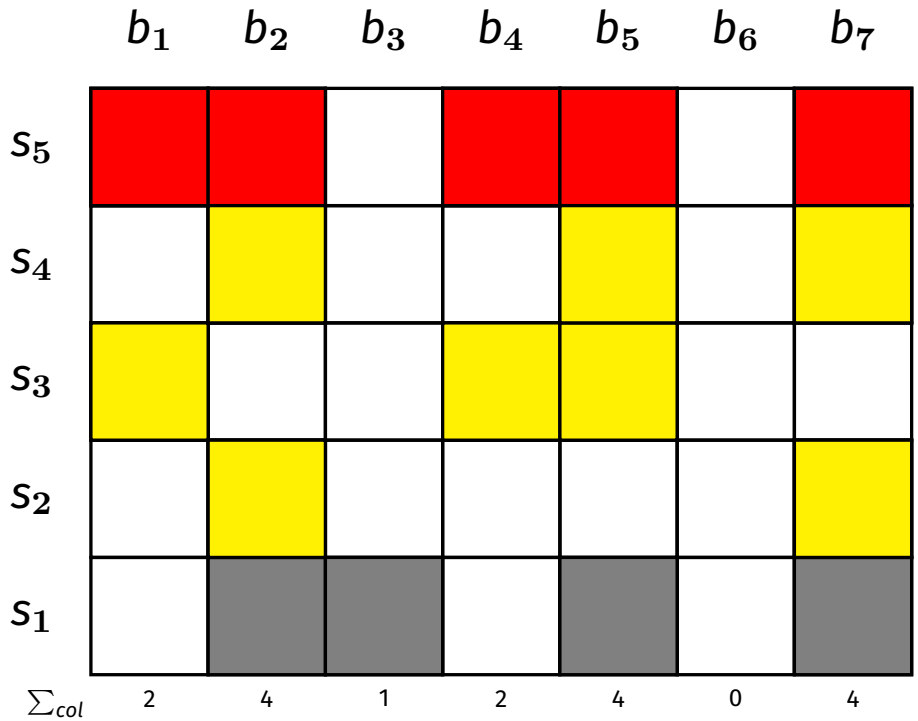


Figure 8: Contained Column Example

Moonshine Algorithm

```
Solution = ∅
While Matrix is Not EMPTY:
```

- Primary Eliminations:
  - Remove all singularities
  - Remove dominant columns
  - Remove submissive rows
- Exotic Eliminations:
  - Solution ⋃ Select all Exotic Rows
    Delete Contained Columns and Rows
- Heuristic Choice:
  - IF No Primary or Exotic Eliminations
    Solution ⋃ Select Longest Row
    Delete Contained Columns and Rows

Moonshine Results

Experimental Results

Table 1: Moonshine Results for Five Large Corpus Examples

File Type	Collection Size	Distilled Size	Improvement Gain
PDF	42,056	664	63
DOC	7,836	745	10
PNG	4,831	94	51
TTF	5,666	27	210
HTML	69,991	410	171

- ”Improvement Gain” is relative to *Greedy Algorithm*

Discussion

So What? The improvements seem only OK ...

- Greedy algorithm works well in practice but ...
- Small gains get compounded over the life of the campaign
- Moonshine also has a **tight theoretical bound** from an optimum solution
  - Greedy algorithm has a terrible bound from optimum
  - Moonshine: worse case bound from optimum *was found to be 3 seeds* !
- Why use Greedy when you can always do better (viz Moonshine)?

Discussion

However there is a problem … Science

- We need to show that in *fuzzing practice* a Moonshine optimum corpus **outperforms** a Greedy corpus in revealing:
  - Crashes
  - Bugs
- Need some good **experiments** at scale
  - Plan is to use ANU's supercomputer **Raijin**
  - Lot's of A/B hypothesis testing experiments
  - Against lots of targets

Final Notes

- Moonshine can also do a **weighted distillation**
  - Take into account *exemplar file sizes*
  - Really important because you are usually IO bound
  - Also significant improvement over weighted greedy

Moonbeam

Moonbeam

Great! I want to use Moonshine but how do I generate my own corpus trace data?

- Use **Moonbeam**
- Moonbeam is a tool developed by Duy Khuu and Adrian Herrera
- Why all this ”Moon X” stuff?
  - All our ”Moon X” projects have something to do with fuzzing corpus management

What is Moonbeam?

- Tracing tool based on the **S2E** software analysis framework
- S2E?
  - Symbolic execution engine for full-system analysis
  - No symbolic execution, but we can still use for dynamic analysis
- Why not Intel Pin, DynamoRIO, etc.?
  - Not cross-platform
  - What about overhead? Moonbeam is **offline**, so speed is less important
  - QEMU doesn't provide a nice way to instrument (unlike S2E)

How does it work?

1. Run PUT in S2E with a seed
2. Enable S2E's basic block coverage plugin
3. Compress S2E's basic block coverage data to Moonshine's **bit vector** format
4. ...

How does it work?

1. Run PUT in S2E with a seed
2. Enable S2E's basic block coverage plugin
3. Compress S2E's basic block coverage data to Moonshine's **bit vector** format
4. ...

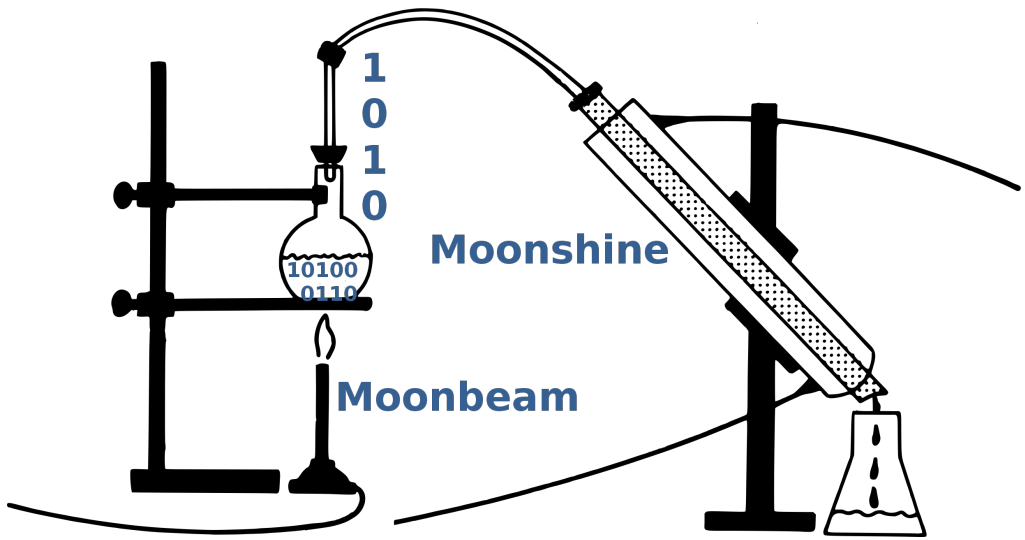


Figure 9: Distilling...

Conclusion

Source Code

Where do I get it?

- **Moonshine**: `https://gitlab.anu.edu.au/lunar/moonshine`
- **Moonbeam**: `https://gitlab.anu.edu.au/lunar/moonbeam`
- Contains source code and test benchmarking data

Questions?

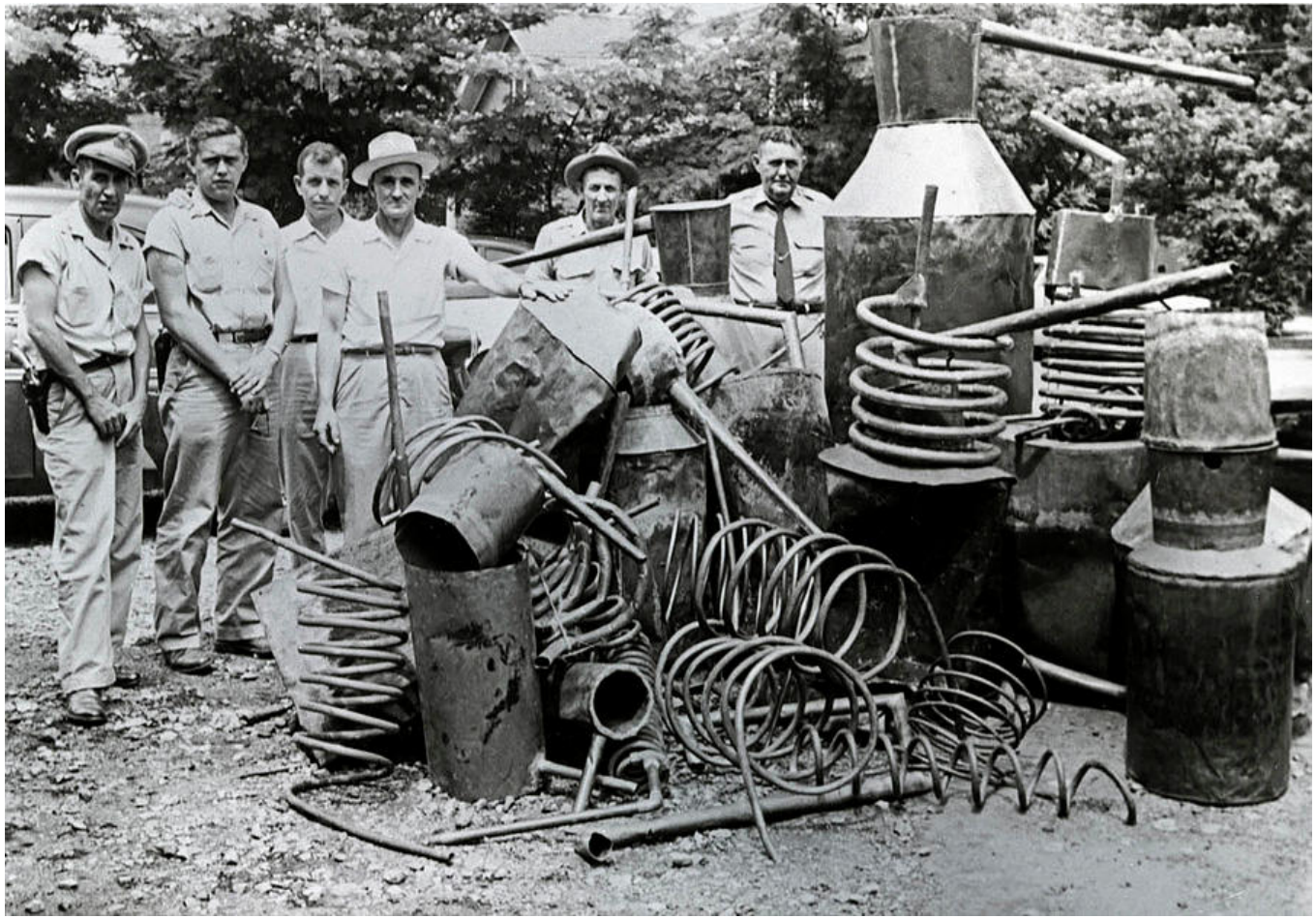


Figure 10: Moonshine DevOps Team