Pruneskin: Investigation into Fuzzing Program Slices

High-level version of algorithm:

```
Data: Program P, sample input I
Result: $C = \{c \in inputs(P) | exec(P, c) \to \text{crash}\}$
$A \leftarrow Analysis(P, I)$;
$S_d \leftarrow Slice(P, A, \text{depth}^a\ d)$;
$\{c'_{d,i}\}_i \leftarrow Fuzz(S_d, I)$;
for $c' \in \{c'_{d,i}\}_i$ do
    $c \leftarrow MapCrash(P, S_d, c')$;
    $C.append(c)$;
end
```

[a] The reasoning for *depth* is that the goal is to fuzz a subgraph of a graph and not separate.

Conjecture

There exists a Slice algorithm such that the MapCrash algorithm either

- *acts like Identity on input sample, or*
- *maps samples crashing S_d to samples crashing P with minimal difficulty or human interaction (heh).*

A **naive** starting approach is implemented:

- For *Analysis()*: execution flow from a given desirable input (i.e. execution trace log)
- For *Slice()*: attempt to remove the non-exec'd basic blocks of functions reached, unless they are the destination block for a compare-branch that we wish to keep (such as if (read() ¡ 0) ...).
- For *MapCrash()*: fuzz original P with input samples that are crashes from fuzzing S_d. with the idea being that under some measurable fuzzing space $|d(c) - d(c')| \leq$ something reasonable.

and using afl-fuzz. But, many possibilities for Analysis, Slice, Fuzz, & MapCrash functions.

Find this code and horseplay: `https://github.com/roachspray/acsac17wip/blob/master/naive`

Minimal set of results

Program	Input[1]	Slice Depth	P 5h fuzz	S_d 1h fuzz	Mapped 4h fuzz[2]
cracklib-2.9.2-5	smalldict	1	0	3	0
		2	0	0	0
		3	0	0	0
exifprobe current	tryone.jpg	3	4	27	37
		4	4	30	50
		5	4	27	13
		6	4	28	30
tcpdump current	ntp.pcap	6	0	0	0
		7	0	0	0
		8	0	0	0
		9	0	0	0
	hcrt.pcap	7	0	1	0
		8	0	6	0
		9	0	0	0
		10	0	0	0
	arp.pcap	3	0	0	0
		4	0	0	0
		5	0	0	0
		6	0	0	0
		7	0	0	0
		8	0	0	0

Fundamental issues with the above:

- Timebox time selections and averaging of timebox results (handle randomness)
- How to deal with (not) having groundtruth? Equality of bugs found, etc pose non-trivial issues. I yen for a dataset of code, crash, time-til-found, hardware, etc etc.
- Not listed: slice sizes seem invalid

[1] all in repo

[2] uses inputs from slice output

Next Steps

- Improve *Analysis* step. Initially included static value flow analysis, but the code was not working. Investigate clang analyze or hand developed. Also include the use of either manticore[3] or libdft[4].
- Improve *Slice* step to use data flow information in block / function removal selection
- Improve *Slice* step by better dead code removal
- Investigate possible *MapCrash* methods:
  - Cascade AFL

$$(C_d \leftarrow Fuzz(S_d, In)) \rightarrow (C_{d-1} \leftarrow Fuzz(S_{d-1}, C_d)) \rightarrow \ldots \rightarrow (C_0 \leftarrow Fuzz(P, C_1))$$

  - Selective AFL instrumentation based on slicing information. See [5]
  - Investigate signaling AFL from undesirable paths. Equivalent to VUZZer with weighting.
  - Generate crash trace from sample, use with manticore to guide program through solving CSPs as it goes toward crash state.

As you can tell, this work is still in it's infancy and there are many paths to investigate. Trying to determine best routes and choices to keep the science legit is an important piece (i.e. some understanding of a ground truth). But, many other areas to play. I would greatly appreciate anyone interested to combine forces, or outright take some ideas and go forth.

[3] https://github.com/trailofbits/manticore

[4] https://www.cs.columbia.edu/~vpk/research/libdft/

[5] https://github.com/roachspray/acsac17wip/blob/master/patches/afl-2.51b-optionally-instrument-based-on-trace.patch