Seph

12 Dec 2014

Bug hunting with American Fuzzy Lop

I read about American Fuzzy Lop (AFL) the other day, and wanted to try it. For the non-technical, fuzzers are testing tools for finding obscure bugs. They work by effectively sitting a million monkeys at keyboards pressing buttons in your program, to see if it crashes or hangs. AFL is a little different than normal fuzzers because it uses flow analysis to make particularly diabolical input data. It does this by instrumenting your (C/C++) program during compilation.

As an excuse to mess around with it, I pulled out a little library I wrote a few years ago called librope. Librope implements a rope data structure (wikipedia), which is a more complex version of a string (har har). If you use an array to store a string, inserting or deleting characters in the middle of the string is a O(n) operation, whereas if you use a tree, inserts and deletes are O(log n). The extra complexity of the data type starts paying off when you're making inline edits on a string thats more than a kilobyte or so.

The API is pretty simple:

```
rope *r = rope_new();
// Insert "Hi there!" at unicode position 0
rope_insert(r, 0, "Hi there!");
// Delete 6 characters at unicode position 2
rope_delete(r, 2, 6);
```

There's some more functions, but those are the only functions that I'm going to test for now.

Fuzzers are famous for finding bugs and security holes in software, but librope is pretty small and very well tested, so I went in pretty confident that I wouldn't find any new surprises.

Wrapping librope for AFL

AFL is designed for tools like libpng, which take a file as an input. To test these C functions, we'll need to make a simple way to call them from stdin. So the next step is making a simple file format. The test harness will read the file and call `rope_insert` and `rope_delete` based on the contents of the file.

Its tempting to use a binary format - AFL quite likes binary files and its usually easier to make a binary encoding format. But we're making text edits, and the library is simple and small anyway. So I went with a text format. The simplest possible file format I could think of is line delimited. The file looks like this:

```
0         // position
Hi there! // an insert
2         // position
```

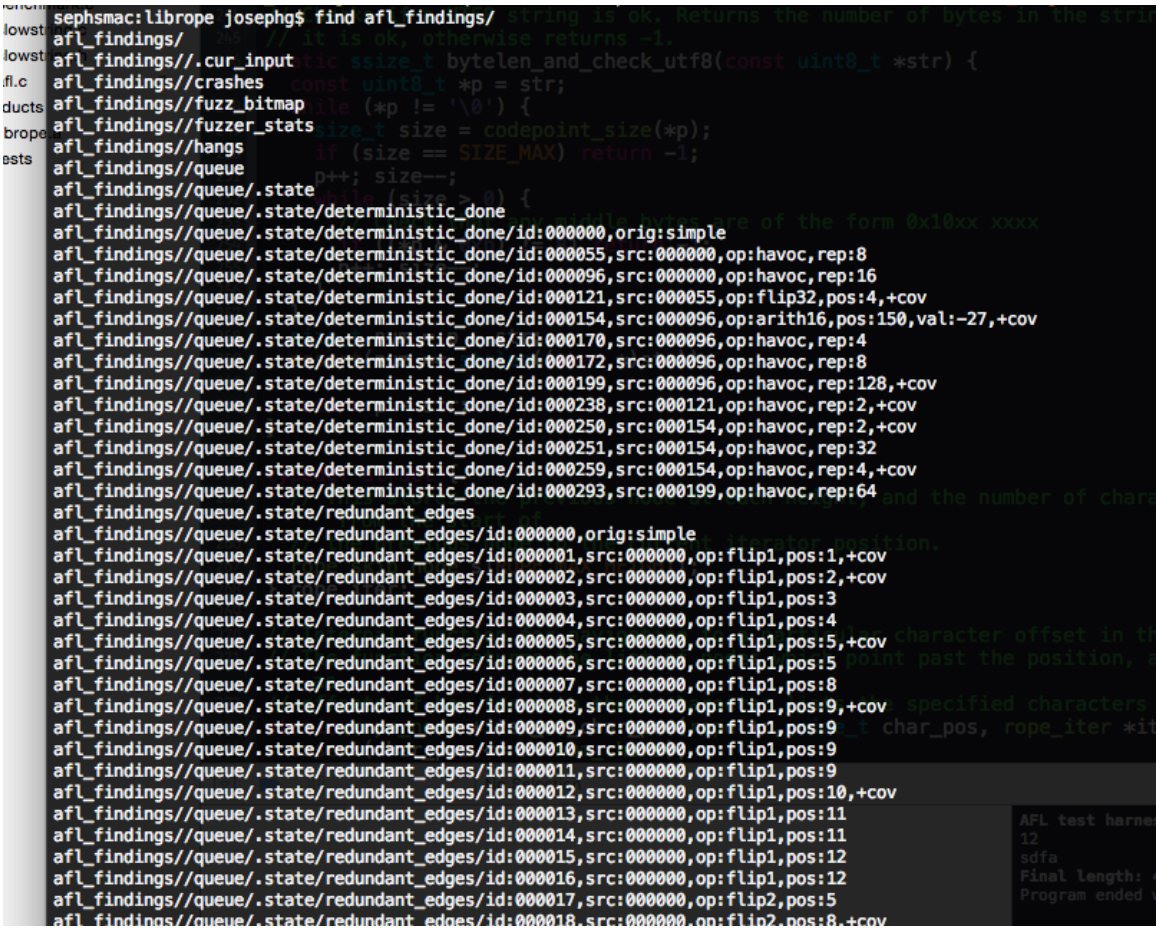
```
-6        // delete (deletes starts with a '-')
... more positions and insert/deletes
```

The harness simply reads lines from standard in. The first of every pair of lines is the position in the document. The second line is either the text to insert, or '-num' to delete. This harness code can't insert any strings starting with a '-' or ending in a newline, but neither of those characters are special in any way to the rope library so I'm not worried.

The harness code I ended up using is here. AFL also needs at least 1 example input file, so I saved that example to `sample_input`.

Running AFL without breaking my computer

Internally, AFL works with an internal queue of test data it finds interesting. Weirdly, instead of storing the test data in memory it stores it in a big directory of test files. On my laptop AFL runs at about 700 runs per second, so its creating & deleting about 700 little files per second.



Modern SSDs are terrible for this sort of workload, and I'm working from my little macbook air. So I want all these files to live in RAM. To do this, I created a 100mb ramdisk and put everything there. A ramdisk is a chunk of ram which pretends to be a normal disk to the operating system. But of course, being ram all the data gets deleted when your computer restarts.

To do this, run this command:

```
diskutil erasevolume HFS+ 'RAM Disk' `hdiutil at-
tach -nomount ram://204800`
```

This creates a new ramdisk with 204800 blocks of 512 bytes (100mb) and puts an HFS+ filesystem on it. The commands are different on linux (you'll have to look them up).

My drive was mounted in `/Volumes/RAM Disk`, so I copied all my files over there.

Running AFL

To run librope through AFL, we need to use the afl-wrapped version of clang / gcc. First, `brew install afl-fuzz` then:

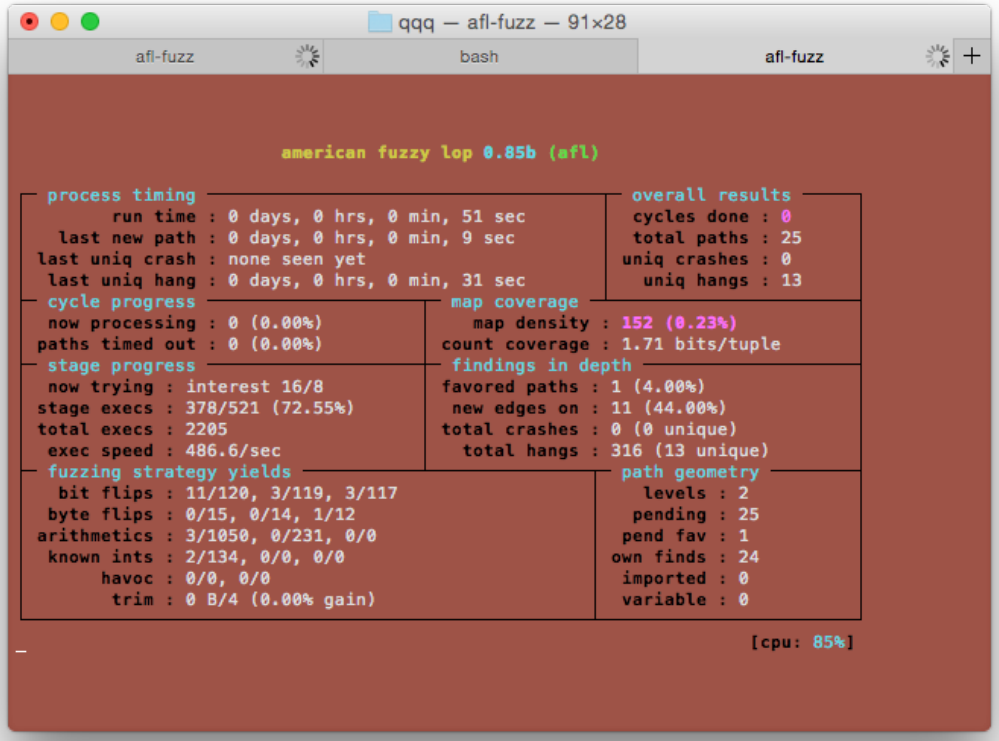
```
% mkdir afl_testcases afl_findings
% cp sample_input afl_testcases/
% afl-clang -O2 -Wall -I. -std=c99 -arch x86_64
rope.c test/afl.c -o aflharness
```

Building with `afl-clang` is really easy for me because there's literally 2 C files to build. If your build environment is more complicated, you should take a look at the afl readme.

Actually running the fuzzer is pretty easy:

```
% afl-fuzz -i afl_testcases -o afl_findings
./aflharness
```

Woo it works!



There's lots of numbers here which don't mean anything to me. The important part is the top right box. `uniq crashes : 0` looks good, but whats that `uniq hang : 13`? That means its found bugs, and it found them in the first few seconds of its run. Ouch.

> ***EDIT**: It turns out AFL will report crashes as hangs on macos sometimes because of the crash reporter. Read here for instructions on how to disable it*

As you'd expect, the hangs are all in the afl_findings directory:

```
josephg$ ls afl_findings/hangs
id:000000,src:000000,op:flip1,pos:5
id:000007,src:000000,op:flip4,pos:8
id:000001,src:000000,op:flip1,pos:6
id:000008,src:000000,op:flip4,pos:8
... lots more
```

These are all test inputs which break my program:

```
./aflharness < afl_findings/hangs/id\:000001*
AFL test harness
Segmentation fault: 11
```

:(

It turns out my `rope_insert` function assumes the string you're inserting is valid UTF8. If the UTF8 encoding is invalid, it explodes violently. Is this a real bug? I'm not sure - I did mention that you need to do your own validation in the docs, but my library shouldn't crash if you don't. Some simple validation there is pretty easy, so I added it.

Thats a brief introduction to AFL. The jury is still out on whether my library has more bugs that AFL will find. Its been running for the last hour without finding anything but I'm going to leave it running overnight just in case.

Its made my code better already anyway, so I'm pretty delighted with it. I wonder if it can find bugs in redis...

Joseph Gentle
http://josephg.com