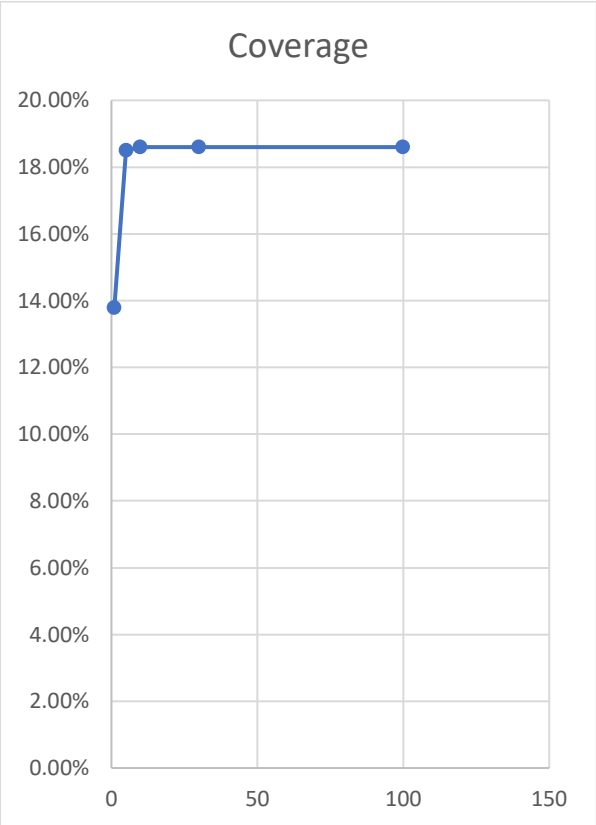
CS6888 Software Analysis HW1 report

Jianqiang Chen(jc3ze)

1. Specify the machine, OS, and the versions of AFL and TCC used.

Machine1: google cloud n1-standard-1　1 vCPU，3.75 GB memory
OS: Ubuntu 18.04.4 LTS (GNU/Linux 5.3.0-1018-gcp x86_64)
AFL: 2.52b
TCC: 0..6.2
Machine2: google cloud e2-highcpu-4　4 vCPU，4 GB memory
OS: Ubuntu 18.04.4 LTS
AFL: 2.52b
TCC: 0..6.2

2. Plot the line coverage achieved by AFL when running the provided sample program for 1, 5, 10, 30, and 100 minutes (plot x-axis is time, y-axis is coverage). Discuss what you observe.



1min 13.8% 1699/12310;
5min 18.5% 2273/12310;
10min 18.6% 2288/12310;
30min 18.6% 2293/12310;
100min 18.6% 2293/12310

I find that before 5min, the line coverage grows rapidly. But after it is over 18.5%, it almost stays there.

It may be caused by the limitation of new line generation. Since the testcase is only 1, AFL could only generate limited types of new lines. After the new lines increases to a certain amount, it's hard to generate new lines.

3. Explain the strategies used to increase the coverage generated by the Fuzzer.

- Increase the testcase number but keep the size small:
  I have searched different types of C program(https://github.com/Ratheshprabakar/C-Complete-practice), and put it in the testcase. But the coverage is always 47%~49%. I think the reason is the Basic Block of this practice is too simple and similar. So I add

some complex questions like DFS, DP, C pointers. After this implementation, the coverage increases to 50.8% in 30 minutes.

- Parallelize the fuzzers
  I rent a 4 cpus server provided by Google cloud. In 30 minutes, the coverage increases to 53%.

```
afl-fuzz -M fuzzer1 -i /usr/local/share/aflIO/afl-input -o /usr/local/share/aflIO/afl-output /usr/local/share/tcc-0.9.27/tcc @@
afl-fuzz -S fuzzer2 -i /usr/local/share/aflIO/afl-input -o /usr/local/share/aflIO/afl-output /usr/local/share/tcc-0.9.27/tcc @@
afl-fuzz -S fuzzer3 -i /usr/local/share/aflIO/afl-input -o /usr/local/share/aflIO/afl-output /usr/local/share/tcc-0.9.27/tcc @@
afl-fuzz -S fuzzer4 -i /usr/local/share/aflIO/afl-input -o /usr/local/share/aflIO/afl-output /usr/local/share/tcc-0.9.27/tcc @@
```

- Add parameter
  I use some suggestion about keep memory use and timeouts. Add -m [memory] -t [time]. But no difference.

4. Provide the LCOV code coverage report (screenshot of the tool output is sufficient) after implementing those strategies. If you found any failures, please explain how they manifested and what input led to them.

LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - tcc-0.9.27-cov	Lines:	6258	11815	53.0 %
Test:	trace.lcov_info_final	Functions:	354	506	70.0 %
Date:	2020-09-17 07:41:43				

Filename	Line Coverage		Functions	
i386-asm.c	0.0 %	0 / 848	0.0 %	0 / 18
libtcc.c	52.1 %	420 / 806	72.9 %	43 / 59
tcc.c	49.6 %	63 / 127	33.3 %	2 / 6
tcc.h	74.2 %	23 / 31	76.9 %	10 / 13
tccasm.c	0.0 %	0 / 706	0.0 %	0 / 28
tccelf.c	77.5 %	1175 / 1517	80.3 %	61 / 76
tccgen.c	60.4 %	2408 / 3988	87.4 %	132 / 151
tccpp.c	69.4 %	1527 / 2201	79.8 %	67 / 84
tccrun.c	1.6 %	4 / 248	9.1 %	1 / 11
tcctools.c	0.0 %	0 / 170	0.0 %	0 / 6
x86_64-gen.c	54.6 %	577 / 1056	66.7 %	32 / 48
x86_64-link.c	52.1 %	61 / 117	100.0 %	6 / 6

Generated by: LCOV version 1.12

Unfortunately, AFL doesn't find bugs in the tcc.

5. Discuss what strategies were the most cost-effective and explain why that was the case.
I think add small size testcase is the most cost-effective strategy and this method is actually the best performance method.
Because the AFL will select favored testcase. As we talked in our class, if the testcase is "interesting", it will more likely to generate more new lines and find bugs. Different BBs, lines, edges will help AFL to generate more lines to cover more lines of source code.

6. Name the functions not being covered in the "tccgen.c", "tccpp.c", and "tccrun.c" files, and explain why they are not covered and what it would take to cover it.
In tccgen.c:

Function name	Describtion	Why not cover	How to cover
ST_FUNC int ieee_finite(double d)	It is to avoid potential problems with non standard math libs	Type 1: no valid input We have defined libs for Math. And all testcase follow standard c libs.	Solution1: spcial testcase Import some testcase using undefined Math lib.
ST_FUNC void vrote (SValue *e, int n)	rotate the n elements before entry e towards the top	Type2: linked func not run This function only used by the vrott function	Solution2: make link func run Cover the vrott function will cover this function
ST_FUNC void vla_runtime_type_size(CType *type, int *a)	push type size as known at runtime time on top of value stack. Put alignment at 'a'	Type2: linked func not run Other functions using this function is not covered	Solution2: make link func run
ST_FUNC void vrott(int n)	rotate n first stack elements to the top	Type2: linked func not run	Solution1: spcial testcase

In tccpp.c:

Function name	Describtion	Why not cover	How to cover
cstr_wccat(CString *cstr, int ch)	add a wide char	The input string is not long enough, jumped	Solution1: spcial testcase
add_char(CString *cstr, int c)		Related to go-to statement	Solution3: goto statement is harmful, change
Sym *label_find(int v)	label lookup	Type 3: No func use this func	Solution4: Make other func use it
Sym *label_push(Sym **ptop, int v, int flags)		Type 3: No func use this func	Solution4: Make other func use it
int tcc_preprocess(TCCState *s1)	Preprocess the current file	Type 3: No func use this func	Solution4: Make other func use it

In tccrun.c:

Function name	Describtion	Why not cover	How to cover
tcc_set_num_callers(int n)		Type 3: No func use this func	Solution4: Make other func use it

Bonus question:

I want to test https://github.com/BennyQBD/CGFX5 But failed… May be not compiled correct.

- Compile CGFX5:

export CC=/usr/local/share/afl-2.52b/afl-gcc CXX=/usr/local/share/afl-2.52b/afl-gcc

install dependencies

mkdir build

cd build

cmake ../

make

- Compile CGFX5-cov:

for cov:

export CC=/usr/bin/gcc5.4.0 CXX=/usr/bin/g++5.4.0

install dependencies

mkdir build

cd build

cmake ../

make

```
root@cs6888:/usr/local/share/CGFX5# cd build
root@cs6888:/usr/local/share/CGFX5/build# cmake ../
-- The C compiler identification is GNU 5.4.0
-- The CXX compiler identification is GNU 5.4.0
-- Check for working C compiler: /usr/local/share/afl-2.52b/afl-gcc
-- Check for working C compiler: /usr/local/share/afl-2.52b/afl-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/local/share/afl-2.52b/afl-g++
-- Check for working CXX compiler: /usr/local/share/afl-2.52b/afl-g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenGL: /usr/lib/x86_64-linux-gnu/libGL.so
-- Looking for GLEW - found
-- Looking for SDL2 - found
-- Looking for ASSIMP - found
-- Configuring done
-- Generating done
-- Build files have been written to: /usr/local/share/CGFX5/build
root@cs6888:/usr/local/share/CGFX5/build#
```

Fuzz result:

```
[-] Oops, the program crashed with one of the test cases provided. There are
    several possible explanations:

    - The test case causes known crashes under normal working conditions. If
      so, please remove it. The fuzzer should be seeded with interesting
      inputs - but not ones that cause an outright crash.

    - The current memory limit (300 MB) is too low for this program, causing
      it to die due to OOM when parsing valid files. To fix this, try
      bumping it up with the -m setting in the command line. If in doubt,
      try something along the lines of:

      ( ulimit -Sv $[299 << 10]; /path/to/binary [...] <testcase )

      Tip: you can use http://jwilk.net/software/recidivm to quickly
      estimate the required amount of virtual memory for the binary. Also,
      if you are using ASAN, see /usr/local/share/doc/afl/notes_for_asan.txt.

    - Least likely, there is a horrible bug in the fuzzer. If other options
      fail, poke <lcamtuf@coredump.cx> for troubleshooting tips.

[-] PROGRAM ABORT : Test case 'id:000000,orig:vecmath_tests.cpp' results in a crash
        Location : perform_dry_run(), afl-fuzz.c:2852
```

```
[-] Looks like there are no valid test cases in the input directory! The fuzzer
    needs one or more test case to start with - ideally, a small file under
    1 kB or so. The cases must be stored as regular files directly in the
    input directory.

[-] PROGRAM ABORT : No usable test cases in '/usr/local/share/CGFXIO/afl-input'
        Location : read_testcases(), afl-fuzz.c:1496
```