The Difference between American Fuzzy Loop and KLEE Symbolic Execution Engine in Use

Yike Yao[1,a]

[1]*Faculty of Information Technology Macau University of Science and Technology, Macau, China,*

[a]*1909853ui011006@student.must.edu.mo*

ABSTRACT
Fuzzing test is a very important method to detect software bugs, especially those will cause serious crashes. Usually, normal fuzzing tools needs to reach a high branch coverage as soon as possible. They need to find bugs that will be ignored in normal times. The current fuzzing test tools are KLEE and AFL they differ in principle and use. This paper finds that the difference between them. Klee can reach a higher branch coverage at shorter time than traditional fuzzing test tool AFL.

Keywords: *KLEE, AFL, fuzzing test*

1. INTRODUCTION

AFL and KLEE are both very important tools for fuzzing tests. A fuzzing test is a very important way for programmers to find bugs. AFL and KLEE are both very useful and easy tools to use. This paper will introduce how to use AFL and KLEE. The differences of AFL and KLEE in use. And how they perform when they are test programs. In theory, KLEE uses symbolic execution to do a fuzzing test. The result will be a performance related to input f. AFL's working process is rough as follows:1. From the source Code compilation program for staking, to record Code Coverage. We can stack source code on original code or machine code (using plugin). Test without original code will reduce the speed of testing2. Select some input files and add them to the input queue as the initial test set. The initial test set can be anything. I even use "Hello" as the initial test set.   3.Mutate the files in the queue according to certain policies. Most of the time the mutation is invalid. There are many tools to improve the probability of generating valid mutation, Witch means it can 4. If the coverage is updated after mutation, it will be retained and added to the queue;5. The above process will continue in a loop, during which the files that triggered the crash will be recorded. This paper will discuss the differences between KLEE and AFL from the principle and use.In order to find more efficient testing tool, I compared KLEE and AFL in their principles and in their use.[1-3] in this work.Using not appropriate fuzzing test tool in certain software may cause a large waste of time, Sometimes the difference in test times can be more than tenfold.[4]So it is very important to find the advantages and disadvantages of different testing tools.One way is to compare them in use.[5-6]

2. PROGRAM BUGS

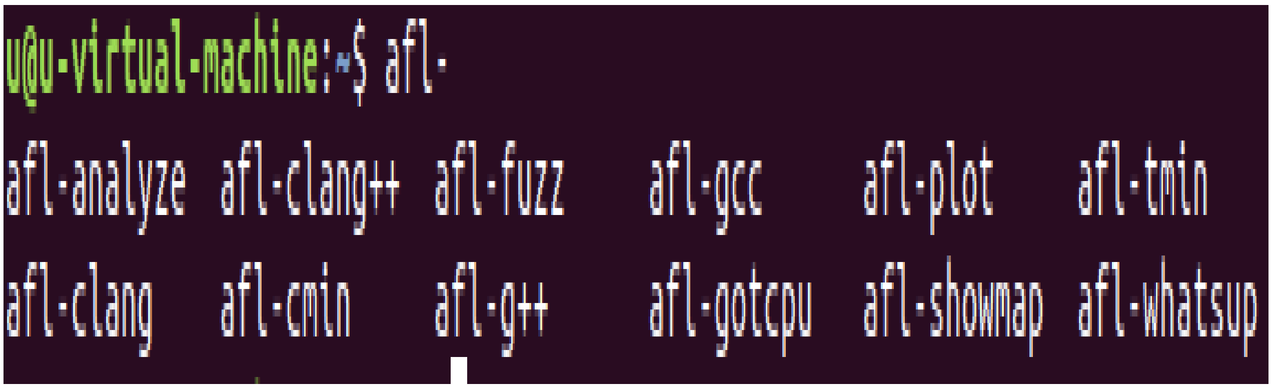
Fuzzing test tools are usually used to find serious bugs in software. Some bugs may not be found for a long time after the software came out. Some bugs can even come out with very bad results. There are some events one of the most famous software bugs is millennium bug. Some serious bugs can't be found when software is developed by programmers. Fuzzing test is one kind of method of software testing. Fuzzing test tools are usually used to test serious bugs.[7]

3. AFL

3.1. The Preparatory Work

American Fuzzy Loop is a coverage-guided Fuzzy test tool developed by security researcher Micha? Zalewski (@lcamtuf), records the code Coverage of incoming samples by calling them. Thus, the input samples can be adjusted to improve the coverage rate and increase the probability of finding vulnerabilities.

AFL can do fuzzing tests for some binary programs efficiently. But it needs to do Pile driving operation for programs. So AFL is used to do fuzzing tests for programs with complete source code.



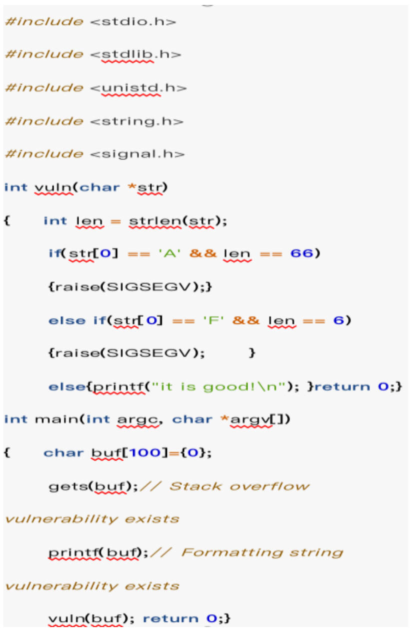
```
u@u-virtual-machine:~$ afl-
afl-analyze  afl-clang++  afl-fuzz    afl-gcc     afl-plot     afl-tmin
afl-clang    afl-cmin     afl-g++     afl-gotcpu  afl-showmap  afl-whatsup
```

Figure 1: Install AFL

The process of using AFL to test a program: Go directly to the official website to download the zip package and unzip it. At first, I thought I need to change some code of AFL because it can't be installed correctly, But a few days later I find that afl is already installed correctly. Enter AFL - at the terminal before TAB it will show the following tools, which means afl is already installed successfully. (Shown in figure 1)

3.2 The testing work

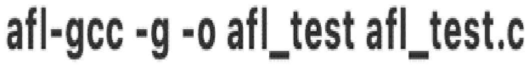
Now I use AFL to test a simple c program



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
int vuln(char *str)
{     int len = strlen(str);
      if(str[0] == 'A' && len == 66)
      {raise(SIGSEGV);}
      else if(str[0] == 'F' && len == 6)
      {raise(SIGSEGV);      }
      else{printf("it is good!\n"); }return 0;}
int main(int argc, char *argv[])
{     char buf[100]={0};
      gets(buf);// Stack overflow
vulnerability exists
      printf(buf);// Formatting string
vulnerability exists
      vuln(buf); return 0;}
```

Figure 2: A simple c program

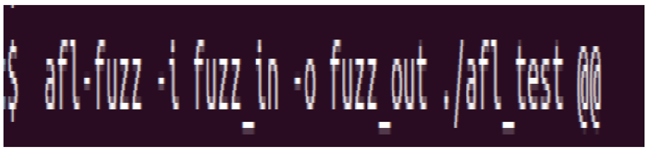
At first, we need to compile the program.



```
afl-gcc -g -o afl_test afl_test.c
```

Figure 3: Compile the program

Then I create two new folders fuzz_in and fuzz_out to store the input and output of AFL. I also need to create a test case file in fuzz in. Now we can start the fuzzing test.



```
$ afl-fuzz -i fuzz_in -o fuzz_out ./afl_test @@
```

Figure 4: Start the test

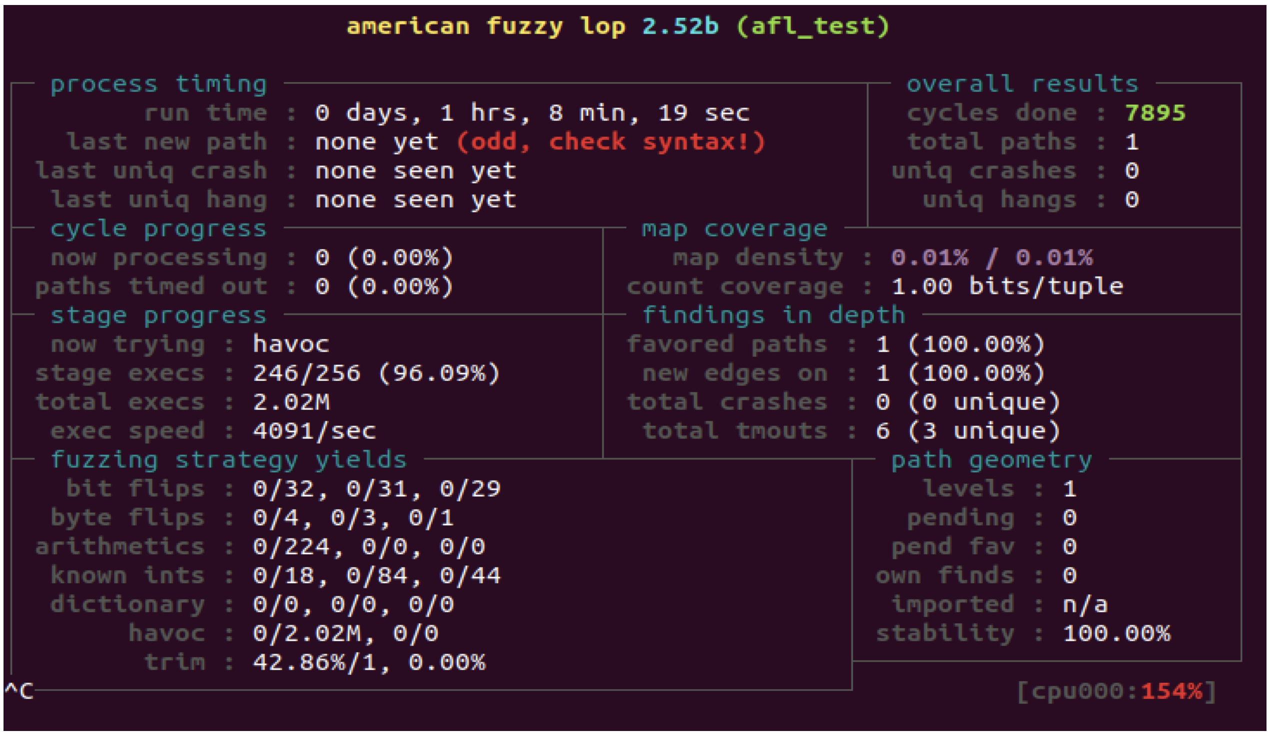
We will see the following interface (shown in figure 2):

Process timing shows how long the current AFL is running, and when the new execution path was last discovered.

overall results include the total number of cycles run, the total number of paths, the number of crashes, and the number of timeouts.

stage progress includes the fuzzing strategy being tested, progress, the total number of times the goal was executed, and how quickly the goal was executed

Execution speeds can visually reflect the current speed of running, and if they are too slow, such as less than 500 times per second, the test time can be very long. (Figure 5 shows the operation interface of AFL.)



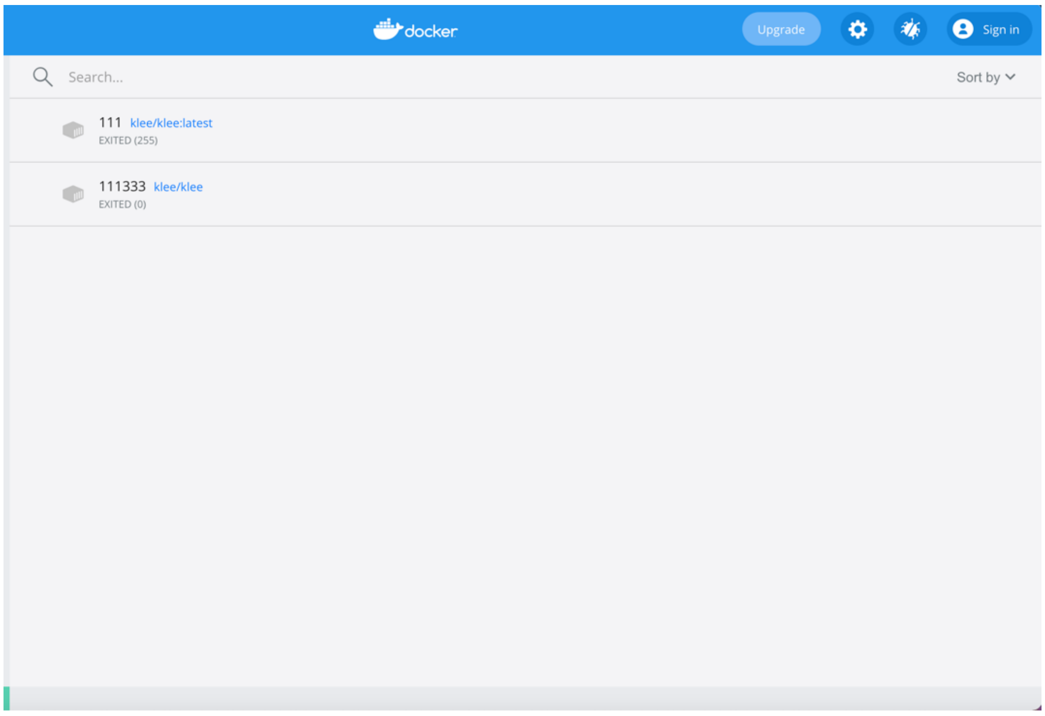
```
                    american fuzzy lop 2.52b (afl_test)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 1 hrs, 8 min, 19 sec       │  cycles done : 7895   │
│   last new path : none yet (odd, check syntax!)      │  total paths : 1      │
│ last uniq crash : none seen yet                      │ uniq crashes : 0      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ──────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 0 (0.00%)        │    map density : 0.01% / 0.01%         │
│ paths timed out : 0 (0.00%)        │ count coverage : 1.00 bits/tuple       │
├─ stage progress ──────────────────┼─ findings in depth ────────────────────┤
│  now trying : havoc                │ favored paths : 1 (100.00%)            │
│ stage execs : 246/256 (96.09%)     │  new edges on : 1 (100.00%)            │
│ total execs : 2.02M                │ total crashes : 0 (0 unique)           │
│  exec speed : 4091/sec             │  total tmouts : 6 (3 unique)           │
├─ fuzzing strategy yields ─────────┴─────────────┬─ path geometry ─────────┤
│   bit flips : 0/32, 0/31, 0/29                   │    levels : 1           │
│  byte flips : 0/4, 0/3, 0/1                      │   pending : 0           │
│ arithmetics : 0/224, 0/0, 0/0                    │  pend fav : 0           │
│  known ints : 0/18, 0/84, 0/44                   │ own finds : 0           │
│  dictionary : 0/0, 0/0, 0/0                      │  imported : n/a         │
│       havoc : 0/2.02M, 0/0                       │ stability : 100.00%     │
│        trim : 42.86%/1, 0.00%                    ├─────────────────────────┘
^C────────────────────────────────────────────────┘          [cpu000:154%]
```

Fig.5: Operation interface of AFL

4. KLEE BASED ON DOCKER

4.1 The Introduction to Docker

To use Klee, we need to install docker first. Docker can provide containers that allow users to run their programs or code inside the container. "Image" means "container" in docker. What happens inside the container will not make a difference to users' computers. So, it's very suitable for fuzzing tests. Compared with VMware, Docker is smaller and more convenient to use. It needs a relatively smaller memory. You can use docker to create images to test your software. When you finished testing, you just need to remove the image to save your memory. However, this also means that docker can't block the software completely. In my experience, I will use docker to run KLEE. The operation interface of Docker is shown in figure6.[8]



111 klee/klee:latest
EXITED (255)

111333 klee/klee
EXITED (0)

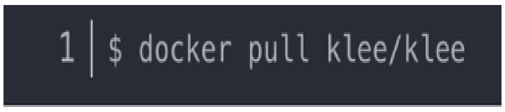
Fig.6: Operating interface of Docker

4.2. Introduction of KLEE

Klee is a tool that interprets LLVM bitcode to implement symbol execution. It symbolizes memory by inserting a function call (klee_make_symbolic). The usage of symbolic memory is tracked and constraints on its use are collected. If there is other memory that uses the preceding symbol memory, that memory will also be symbolized. When confronted with a branch that uses symbolic memory, Klee splits the execution state in half to see which side of the branch can find a solution that satisfies the symbolic constraint. Klee uses STP to solve these symbolic constraints.

4.3. The Process of Using KLEE

After installing docker we use docker to download Klee (Operation is shown in figure7):

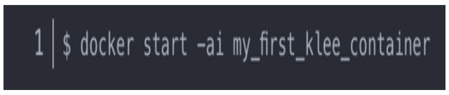


```
$ docker pull klee/klee
```

Figure 7: Download KLEE in Docker

We use docker to create an image at first.

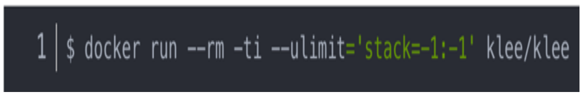
Create a temporary image (Operation is shown in figure8):



```
$ docker start -ai my_first_klee_container
```

Figure 8: Creation of temporary image

Tape "exit" to exit the image (Operation is shown in figure9).

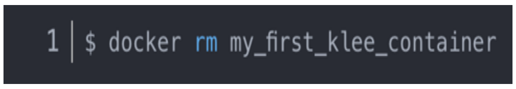


```
$ docker run --rm -ti --ulimit='stack=-1:-1' klee/klee
```

Figure 9: Exit the temporary image

Create a permanent image:

After using "exit" to exit the image, we can delete it by input(Operation is shown in figure10):



```
$ docker rm my_first_klee_container
```

Figure 10: Delete the container

4.4. the Testing Work

After entering the image, we can start to test some codes. There are three examples inside my every image. At first, I can't find the path of file accurately, so I went into the folders one by one.

To test this function with Klee, we first need to set up the symbolic input, that is, to symbolize the input variables. We then set the main function, call the Klee-make-symbolic function, and then use the symbolic input variable to call the function to be tested.

To use Klee to test software, we need to compile programs into LLVM bitcode, which is one kind of instruction set. After that, we can run Klee to do a fuzzing test for programs.

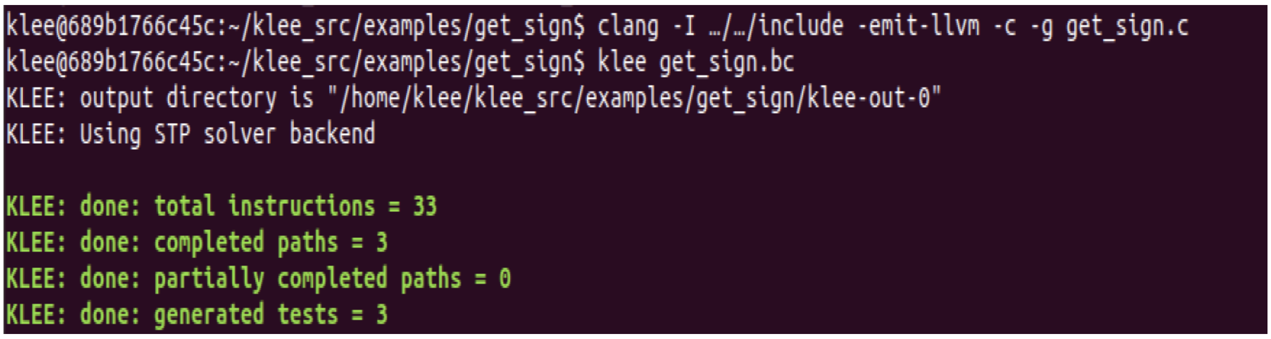
The total instructions may not be the same in different machines. It may also spend different time in different situations. (The results are shown in figure11)

If we want to test our own software, we need to copy our file into the docker first. At first, I use "locate" to find the path of the target file. But this command usually finds too many folders with same name. The path is also not correct. So I use "pwd" to print the path, which is correct.

Move file from host to container and remove it

5. THE DIFFERENCE BETWEEN AFL AND KLEE

AFL and KLEE are both very useful tools for fuzzing tests. AFL requires to Insert piles in the relevant code. If we want to stop fuzzing, we can push "control+c" to stop fuzzing. KLEE need to compile the program into an LLVM bitcode at first, then we need to use Klee to run the. bc file. Both Klee and AFL can set the time of fuzzing. Before we start fuzzing, we can set the total paths and maximum time. Using AFL and Klee we can get a very high coverage rate. to reach a very high coverage rate. Testing result is shown in figure11.



```
klee@689b1766c45c:~/klee_src/examples/get_sign$ clang -I …/…/include -emit-llvm -c -g get_sign.c
klee@689b1766c45c:~/klee_src/examples/get_sign$ klee get_sign.bc
KLEE: output directory is "/home/klee/klee_src/examples/get_sign/klee-out-0"
KLEE: Using STP solver backend

KLEE: done: total instructions = 33
KLEE: done: completed paths = 3
KLEE: done: partially completed paths = 0
KLEE: done: generated tests = 3
```

Figure 11: Result of KLEE testing

6. CONCLUSION

Both AFL and KLEE are fuzzing test tools. They differ in the way they operate. KLEE use symbolic execution. The result of the test is an expression with input f. Using this way KLEE can reach a high branch coverage in a very short time. But symbolic execution cannot deal with situations like some loop or recursion. f will enter a dead loop. So nowadays symbolic execution is always combined with seed input. [9-10] The AFL, however, uses seed input to generate input. AFL can hardly generate a valid seed input. So, it is very ineffective most of the time. Some people improve AFL by adding algorithms to generate valid seed input such as SFL. SFL can check the type of seed input and cut the input sequence into fields. So that it can generate valid input in a very short time. By using a gradient search algorithm, it can choose the right file to mutate. SFL can reach a high branch coverage faster than the traditional fuzzing test tool AFL [4]. Through comparing different testing tools, we can find what is the best tool for testing certain software.Hoping there will be more efficient fuzzing test tools or ways to combine different testing tools that can make more testing more efficiently.

REFERENCES

[1] Schwartz E J, Avgerinos T, Brumley D.All You Ever Wanted to Know about Dynamic Taint Analysis and Forward SymbolicExecution (but Might Have Been Afraid to Ask) [C]// Security & Privacy.DBLP, 2010:317-331.

[2] Cadar C, Sen K. Symbolic execution forsoftware testing: three decades later[M]. ACM, 2013.

[3] C. Cadar, D. Dunbar, and D. Engler. KLEE:Unassisted and Automatic Generation of High-Coverage Tests for Complex SystemsPrograms. In Proceedings of the 8th USENIX Symposium on Operating SystemsDesign and Implementation (OSDI'08), volume 8, pages 209–224, 2008.

[4] W. You, X. Liu, S. Ma, D. Perry, X. Zhang and B. Liang, "SLF: Fuzzing without Valid Seed Inputs," 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 712-723, doi: 10.1109/ICSE.2019.00080.

[5] Ren Zezhong, Zheng Han, Zhang Jiayuan, Wang Wenjie, Feng Tao, Wang He, Zhang Yuqing Overview of fuzzy testing technology [J] Computer research and development,2021,58(05):944-963.

[6] Wang Yan,Jia Peng,Liu Luping,Huang Cheng,Liu Zhonglin. A systematic review of fuzzing based on machine learning techniques.[J]. PloS one,2020,15(8).

[7] Shi Ji, Zeng Zhaolong, Yang congbao, Li peiyue Overview of fuzzy testing technology [J] Information network security, 2014 (03): 87-91

[8] Zheng Yuanping, he Jia, Hu Ju, Yin Zehua, Wu Qu Research on docker container log recovery technology [J] Network security technology and application, 2021 (12): 19-20

[9] N. Kumar, S. Neema, M. Das and B. R. Mohan, "Program Slicing Analysis with KLEE, DIVINE and Frama-C," 2021 26th International Conference on Automation and Computing (ICAC), 2021, pp. 1-5, doi: 10.23919/ICAC50006.2021.9594142.

[10] Zhang Zhiyi,Wang Ziyuan,Yang Fan,Wei Jiahao,Zhou Yuqian,Huang Zhiqiu. Random or heuristic? An empirical study on path search strategies for test generation in KLEE[J]. Journal of Systems and Software,2022(prepublish).