

《软件安全》实验报告

姓名：陆皓喆 学号：2211044 班级：信息安全

实验名称：

*AFL*模糊测试

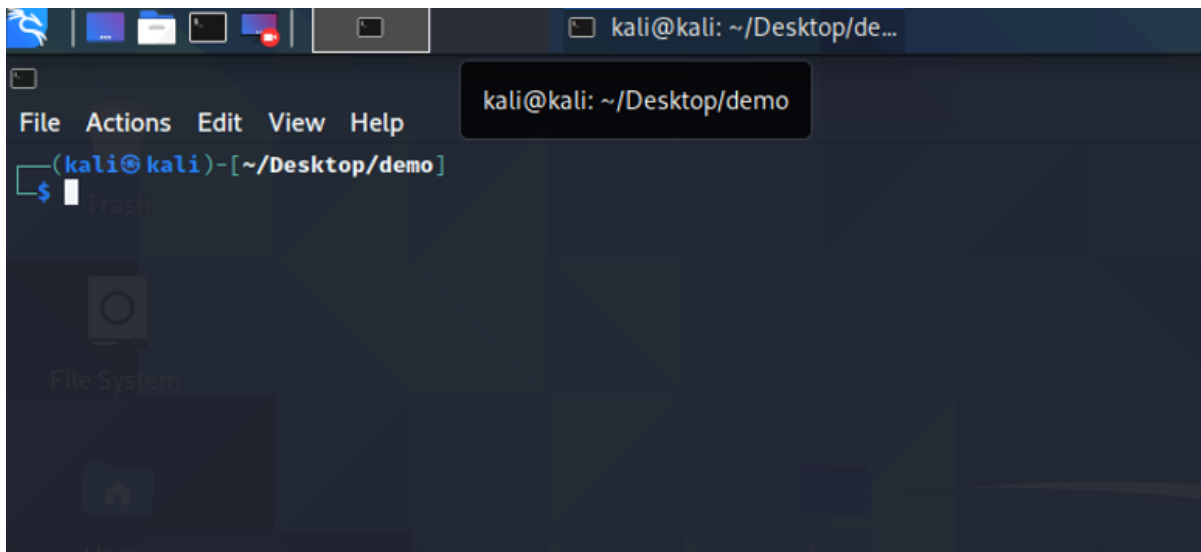
实验要求：

根据课本7.4.5章节，复现*AFL*在*Kali*下的安装、应用，查阅资料理解覆盖引导和文件变异的概念和含义。

实验过程：

1 安装AFL

我们先根据实验要求，进入*vmware*，开启*kali*虚拟机，并在其中创建一个新的文件夹，名字是*demo*。然后我们按下右键，打开终端。



然后，我们输入命令行 `wget http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz`，下载安装包。由于我们的*kali*连不上网（还是没有解决），所以我们在本地下载了文件夹，然后复制到*kali*虚拟机中去，拖入*demo*文件夹。然后我们输入命令行 `tar xvf afl-latest.tgz`，解压安装包。

```
File Actions Edit View Help
afl-2.52b/testcases/images/png/not_kitty.png
afl-2.52b/testcases/images/png/not_kitty_icc.png
afl-2.52b/testcases/images/png/not_kitty_alpha.png
afl-2.52b/testcases/images/png/not_kitty_gamma.png
afl-2.52b/testcases/README.testcases
afl-2.52b/testcases/others/
afl-2.52b/testcases/others/xml/
afl-2.52b/testcases/others/xml/small_document.xml
afl-2.52b/testcases/others/rtf/
afl-2.52b/testcases/others/rtf/small_document.rtf
afl-2.52b/testcases/others/elf/
afl-2.52b/testcases/others/elf/small_exec.elf
afl-2.52b/testcases/others/text/
afl-2.52b/testcases/others/text/hello_world.txt
afl-2.52b/testcases/others/js/
afl-2.52b/testcases/others/js/small_script.js
afl-2.52b/testcases/others/sql/
afl-2.52b/testcases/others/sql/simple_queries.sql
afl-2.52b/testcases/others/pcap/
afl-2.52b/testcases/others/pcap/small_capture.pcap
afl-2.52b/testcases/others/pdf/
afl-2.52b/testcases/others/pdf/small.pdf
afl-2.52b/afl-whatsup
afl-2.52b/debug.h
afl-2.52b/afl-gcc.c
afl-2.52b/afl-as.c
afl-2.52b/afl-plot
afl-2.52b/afl-tmin.c
afl-2.52b/types.h
afl-2.52b/hash.h
afl-2.52b/QuickStartGuide.txt
afl-2.52b/dictionaries/
afl-2.52b/dictionaries/README.dictionaries
afl-2.52b/dictionaries/png.dict
afl-2.52b/dictionaries/sql.dict
afl-2.52b/dictionaries/jpeg.dict
afl-2.52b/dictionaries/gif.dict
afl-2.52b/dictionaries/pdf.dict
afl-2.52b/dictionaries/js.dict
afl-2.52b/dictionaries/json.dict
afl-2.52b/dictionaries/html_tags.dict
afl-2.52b/dictionaries/webp.dict
afl-2.52b/dictionaries/xml.dict
afl-2.52b/dictionaries/tiff.dict
afl-2.52b/afl-fuzz.c
afl-2.52b/llvm_mode/
afl-2.52b/llvm_mode/afl-clang-fast.c
afl-2.52b/llvm_mode/afl-llvm-rt.o.c
afl-2.52b/llvm_mode/README.llvm
afl-2.52b/llvm_mode/afl-llvm-pass.so.cc
afl-2.52b/llvm_mode/Makefile
afl-2.52b/qemu_mode/
afl-2.52b/qemu_mode/build_qemu_support.sh
afl-2.52b/qemu_mode/README.qemu
afl-2.52b/qemu_mode/patches/
afl-2.52b/qemu_mode/patches/afl-qemu-cpu-inl.h
afl-2.52b/qemu_mode/patches/syscall.diff
afl-2.52b/qemu_mode/patches/elfload.diff
afl-2.52b/qemu_mode/patches/cpu-exec.diff
afl-2.52b/Makefile
(kali@kali) - [~/Desktop/demo]
$
```

我们通过命令行 `cd afl-2.52b` 进入对应的文件夹，然后输入 `sudo make && sudo make install` 来编译 *AFL*。

```
kali@kali: ~/Desktop/de...
File Actions Edit View Help
(kali@kali) - [~/Desktop/demo]
$ cd afl-2.52b
(kali@kali) - [~/Desktop/demo/afl-2.52b]
$ sudo make && sudo make install
[sudo] password for kali:
[*] Checking for the ability to compile x86 code...
[*] Everything seems to be working, ready to compile.
cc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -DAFL_PATH="/usr/local/lib/afl" -DDOC_PATH="/usr/local/share/doc/afl" -DBIN_PATH="/usr/local/bin" -c afl-clang-fast.c -o afl-clang-fast.o
cc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -DAFL_PATH="/usr/local/lib/afl" -DDOC_PATH="/usr/local/share/doc/afl" -DBIN_PATH="/usr/local/bin" -c afl-llvm-rt.o.c -o afl-llvm-rt.o
cc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -DAFL_PATH="/usr/local/lib/afl" -DDOC_PATH="/usr/local/share/doc/afl" -DBIN_PATH="/usr/local/bin" -c afl-llvm-pass.so.cc -o afl-llvm-pass.so
ln -sf afl-as as
[*] Testing the CC wrapper and instrumentation output ...
unset AFL_USE_ASAN AFL_USE_MSAN; AFL_QUIET=1 AFL_INST_RATIO=100 AFL_PATH=. ./afl-gcc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -DAFL_PATH=. -c afl-showmap.c -o afl-showmap.o
echo 0 | ./afl-showmap -m none -q -o .test-instr0 ./test-instr
echo 1 | ./afl-showmap -m none -q -o .test-instr1 ./test-instr
[*] All right, the instrumentation seems to be working!
[*] LLVM users: see llvm_mode/README.llvm for a faster alternative to afl-gcc.
[*] All done! Be sure to review README - it's pretty short and useful.

[*] Checking for the ability to compile x86 code...
[*] Everything seems to be working, ready to compile.
[*] Testing the CC wrapper and instrumentation output ...
unset AFL_USE_ASAN AFL_USE_MSAN; AFL_QUIET=1 AFL_INST_RATIO=100 AFL_PATH=. ./afl-gcc -O3 -funroll-loops -Wall -D_FORTIFY_SOURCE=2 -g -Wno-pointer-sign -DAFL_PATH=. -c afl-showmap.c -o afl-showmap.o
echo 0 | ./afl-showmap -m none -q -o .test-instr0 ./test-instr
echo 1 | ./afl-showmap -m none -q -o .test-instr1 ./test-instr
[*] All right, the instrumentation seems to be working!
[*] LLVM users: see llvm_mode/README.llvm for a faster alternative to afl-gcc.
[*] All done! Be sure to review README - it's pretty short and useful.

mkdir -p -m 755 ${DESTDIR}/usr/local/bin ${DESTDIR}/usr/local/lib/afl ${DESTDIR}/usr/local/share/doc/afl ${DESTDIR}/usr/local/share/afl
rm -f ${DESTDIR}/usr/local/bin/afl-plot.sh
install -m 755 afl-gcc afl-fuzz afl-tmin afl-gotcpu afl-analyze afl-plot afl-cmin afl-whatsup ${DESTDIR}/usr/local/bin
rm -f ${DESTDIR}/usr/local/bin/afl-as
if [ -f afl-qemu-trace ]; then install -m 755 afl-qemu-trace ${DESTDIR}/usr/local/bin; fi
if [ -f afl-clang-fast -a -f afl-llvm-pass.so -a -f afl-llvm-rt.o ]; then set -e; install -m 755 afl-clang-fast ${DESTDIR}/usr/local/bin; ln -sf afl-clang-fast ${DESTDIR}/usr/local/bin/afl-clang-fast; fi
if [ -f afl-llvm-rt-32.o ]; then set -e; install -m 755 afl-llvm-rt-32.o ${DESTDIR}/usr/local/lib/afl; fi
if [ -f afl-llvm-rt-64.o ]; then set -e; install -m 755 afl-llvm-rt-64.o ${DESTDIR}/usr/local/lib/afl; fi
set -e; for i in afl-g++ afl-clang afl-clang++; do ln -sf afl-gcc ${DESTDIR}/usr/local/bin/$i; done
install -m 755 afl-as ${DESTDIR}/usr/local/lib/afl/as
ln -sf afl-as ${DESTDIR}/usr/local/lib/afl/as
install -m 644 docs/ChangeLog docs/*.txt ${DESTDIR}/usr/local/share/doc/afl
cp -r testcases/ ${DESTDIR}/usr/local/share/afl
cp -r dictionaries/ ${DESTDIR}/usr/local/share/afl
(kali@kali) - [~/Desktop/demo/afl-2.52b]
$
```

发现编译成功，我们输入命令 `ls /usr/local/bin` 来验证是否安装成功。

```
mkdir -p -m 755 ${DESTDIR}/usr/local/bin ${DESTDIR}/usr/local/lib/afl ${DESTDIR}/usr/local/share/doc/afl ${DESTDIR}/usr/local/share/afl
rm -f ${DESTDIR}/usr/local/bin/afl-plot.sh
install -m 755 afl-gcc afl-fuzz afl-showmap afl-tmin afl-gotcpu afl-analyze afl-plot afl-cmin afl-whatsup ${DESTDIR}/usr/local/bin
rm -f ${DESTDIR}/usr/local/bin/afl-as
if [ -f afl-qemu-trace ]; then install -m 755 afl-qemu-trace ${DESTDIR}/usr/local/bin; fi
if [ -f afl-clang-fast -a -f afl-llvm-pass.so -a -f afl-llvm-rt.o ]; then set -e; install -m 755 afl-clang-fast ${DESTDIR}/usr/local/bin; fi
if [ -f afl-llvm-rt-32.o ]; then set -e; install -m 755 afl-llvm-rt-32.o ${DESTDIR}/usr/local/lib/afl; fi
if [ -f afl-llvm-rt-64.o ]; then set -e; install -m 755 afl-llvm-rt-64.o ${DESTDIR}/usr/local/lib/afl; fi
set -e; for i in afl-g++ afl-clang afl-clang++; do ln -sf afl-gcc ${DESTDIR}/usr/local/bin/$i; done
install -m 755 afl-as ${DESTDIR}/usr/local/lib/afl
ln -sf afl-as ${DESTDIR}/usr/local/lib/afl/as
install -m 644 docs/README docs/ChangeLog docs/*.txt ${DESTDIR}/usr/local/share/doc/afl
cp -r testcases/ ${DESTDIR}/usr/local/share/afl
cp -r dictionaries/ ${DESTDIR}/usr/local/share/afl

(kali@kali)-[~/Desktop/demo/afl-2.52b]
$ ls /usr/local/bin
afl-analyze  afl-clang  afl-clang++  afl-cmin  afl-fuzz  afl-g++  afl-gcc  afl-gotcpu  afl-plot  afl-showmap  afl-tmin  afl-whatsup

(kali@kali)-[~/Desktop/demo/afl-2.52b]
$
```

发现安装成功，并且输出了里面含有的 *AFL* 文件。

2 AFL的应用

我们接下来利用安装好的 *AFL* 文件，来复现课本上出现的模糊测试的案例，来进一步加深对于 *AFL* 应用的理解。

2.1 创建测试程序

在 *demo* 文件夹中新建一个 *test.c* 文件，并输入我们的源码：

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv) {
    char ptr[20];
    if(argc>1){
        FILE *fp = fopen(argv[1], "r");
        fgets(ptr, sizeof(ptr), fp);
    }
    else{
        fgets(ptr, sizeof(ptr), stdin);
    }
    printf("%s", ptr);
    if(ptr[0] == 'd') {
        if(ptr[1] == 'e') {
            if(ptr[2] == 'a') {
                if(ptr[3] == 'd') {
                    if(ptr[4] == 'b') {
                        if(ptr[5] == 'e') {
                            if(ptr[6] == 'e') {
                                if(ptr[7] == 'f') {
                                    abort();
                                }
                                else
                                    printf("%c",ptr[7]);
                            }
                        }
                    }
                }
            }
        }
        else
            printf("%c",ptr[6]);
    }
    else
        printf("%c",ptr[5]);
}
```

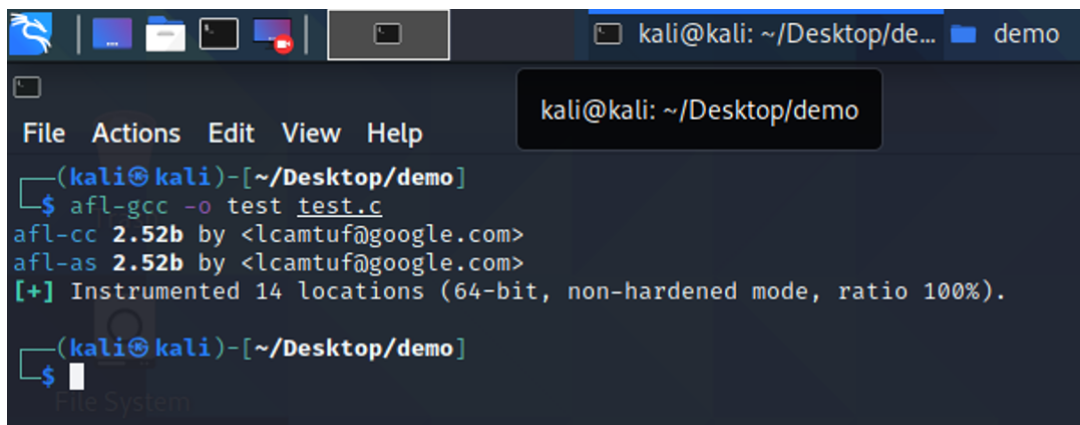
```

    }
    else    printf("%c",ptr[4]);
    }
    else    printf("%c",ptr[3]);
    }
    else    printf("%c",ptr[2]);
    }
    else    printf("%c",ptr[1]);
    }
    else    printf("%c",ptr[0]);
    return 0;
}

```

通过分析代码可知，当输入字符串“deadbeef”时程序捕捉到一个异常，程序终止。

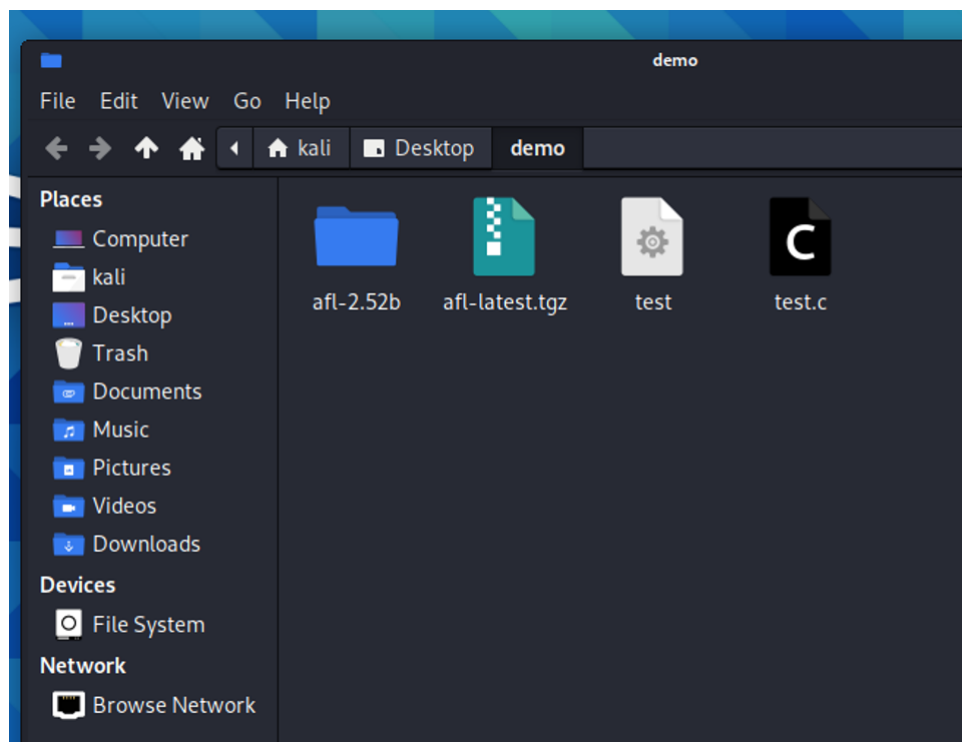
我们使用`linux`的编译器进行编译，可以使模糊过程更加高效。我们输入命令行 `afl-gcc -o test test.c`，来对源代码进行编译。发现得到一个编译后的文件。



```

kali@kali: ~/Desktop/de... demo
File Actions Edit View Help
(kali@kali)-[~/Desktop/demo]
$ afl-gcc -o test test.c
afl-cc 2.52b by <lcamtuf@google.com>
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 14 locations (64-bit, non-hardened mode, ratio 100%).
(kali@kali)-[~/Desktop/demo]
$

```



我们接下来用命令行 `readelf -s ./test | grep afl` 来验证插桩符号：

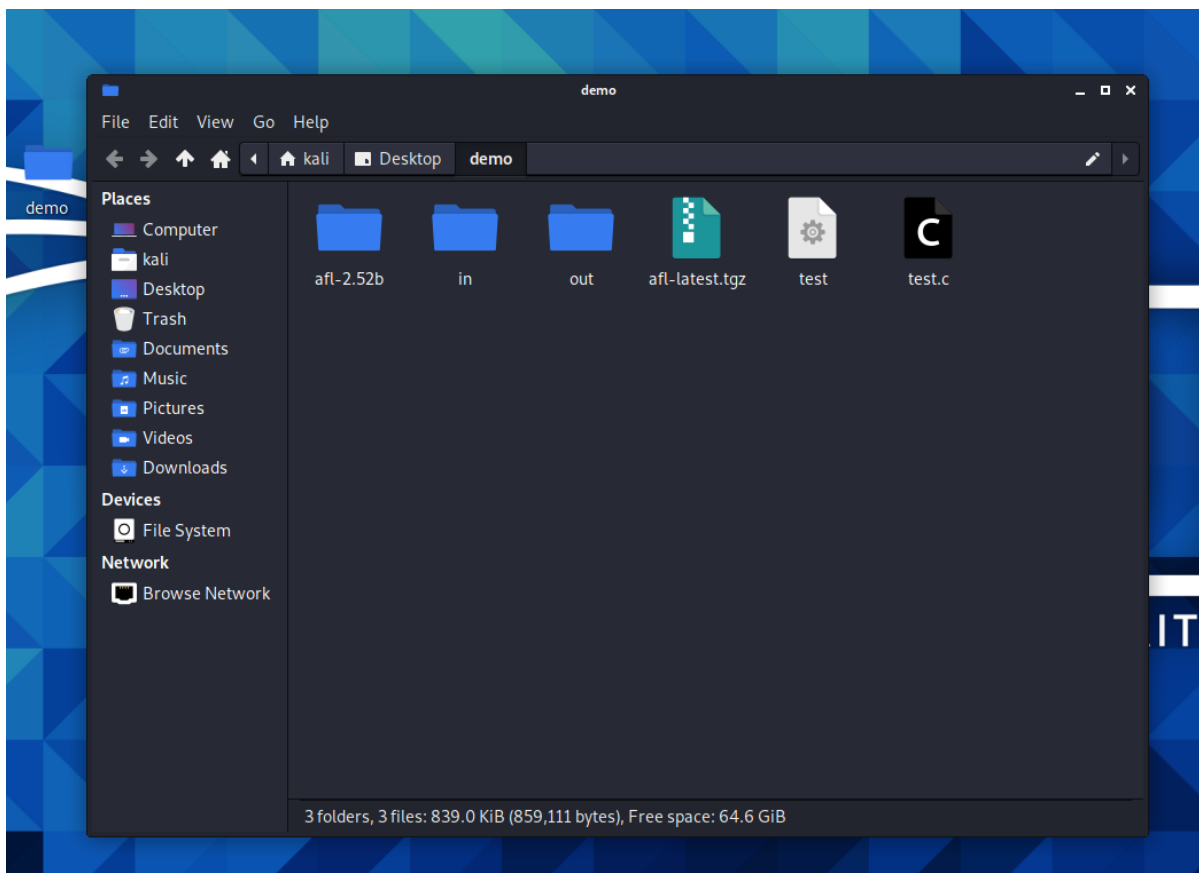
```
kali@kali: ~/Desktop/de... demo
File Actions Edit View Help
(kali@kali)-[~/Desktop/demo]
$ readelf -s ./test | grep afl
35: 00000000000001628      0 NOTYPE LOCAL DEFAULT 14 __afl_maybe_log
37: 000000000000040b0      8 OBJECT LOCAL DEFAULT 25 __afl_area_ptr
38: 00000000000001658      0 NOTYPE LOCAL DEFAULT 14 __afl_setup
39: 00000000000001638      0 NOTYPE LOCAL DEFAULT 14 __afl_store
40: 000000000000040b8      8 OBJECT LOCAL DEFAULT 25 __afl_prev_loc
41: 00000000000001650      0 NOTYPE LOCAL DEFAULT 14 __afl_return
42: 000000000000040c8      1 OBJECT LOCAL DEFAULT 25 __afl_setup_failure
43: 00000000000001679      0 NOTYPE LOCAL DEFAULT 14 __afl_setup_first
45: 0000000000000193e      0 NOTYPE LOCAL DEFAULT 14 __afl_setup_abort
46: 00000000000001793      0 NOTYPE LOCAL DEFAULT 14 __afl_forkserver
47: 000000000000040c4      4 OBJECT LOCAL DEFAULT 25 __afl_temp
48: 00000000000001851      0 NOTYPE LOCAL DEFAULT 14 __afl_fork_resume
49: 000000000000017b9      0 NOTYPE LOCAL DEFAULT 14 __afl_fork_wait_loop
50: 00000000000001936      0 NOTYPE LOCAL DEFAULT 14 __afl_die
51: 000000000000040c0      4 OBJECT LOCAL DEFAULT 25 __afl_fork_pid
98: 000000000000040d0      8 OBJECT GLOBAL DEFAULT 25 __afl_global_area_ptr

(kali@kali)-[~/Desktop/demo]
$
```

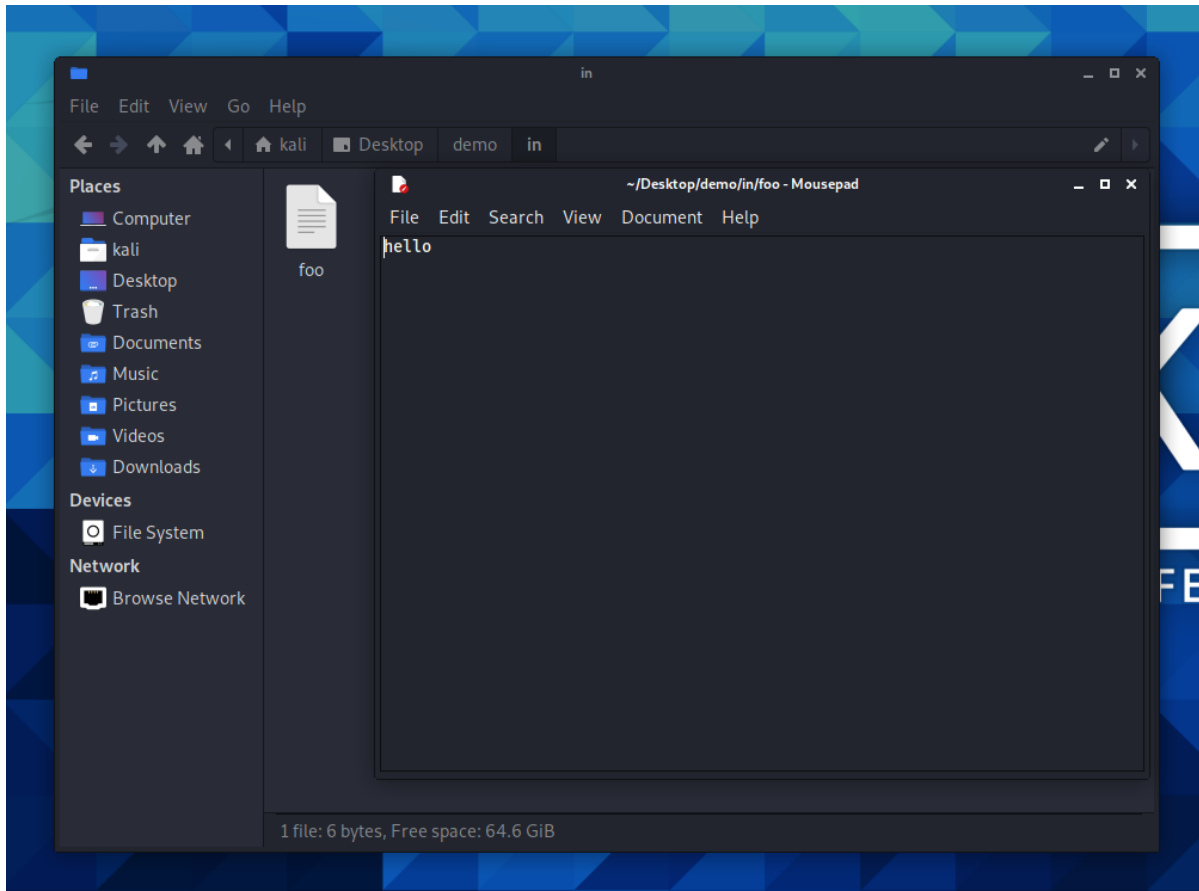
至此，我们创建好了我们本次测试需要用到的程序，接下来我们需要创建测试用例。

2.2 创建测试用例

首先，创建两个文件夹 `in` 和 `out`，分别存储模糊测试所需的输入和输出相关的内容。命令：`mkdir in out`，结果如下图：



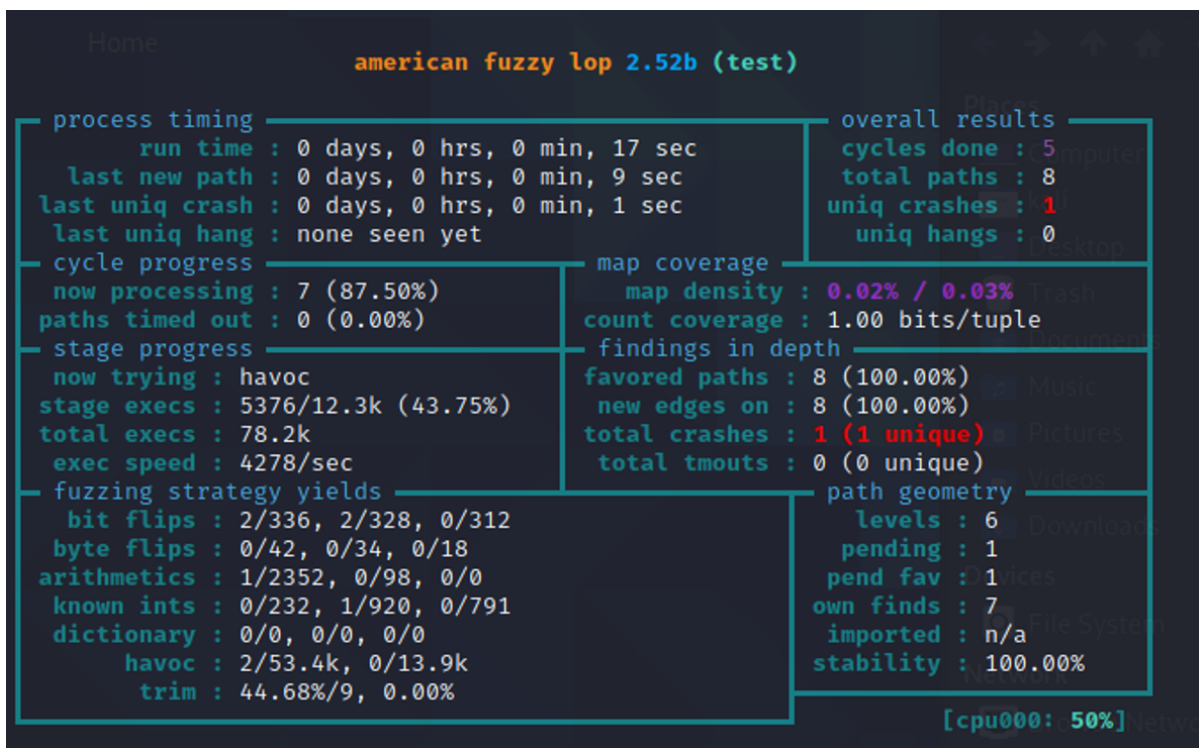
然后，在输入文件夹中创建一个包含字符串“hello”的文件。命令行：`echo hello> in/foo`。`foo` 就是我们的测试用例，里面包含初步字符串 `hello`。`AFL`会通过这个语料进行变异，构造出更多的测试用例。



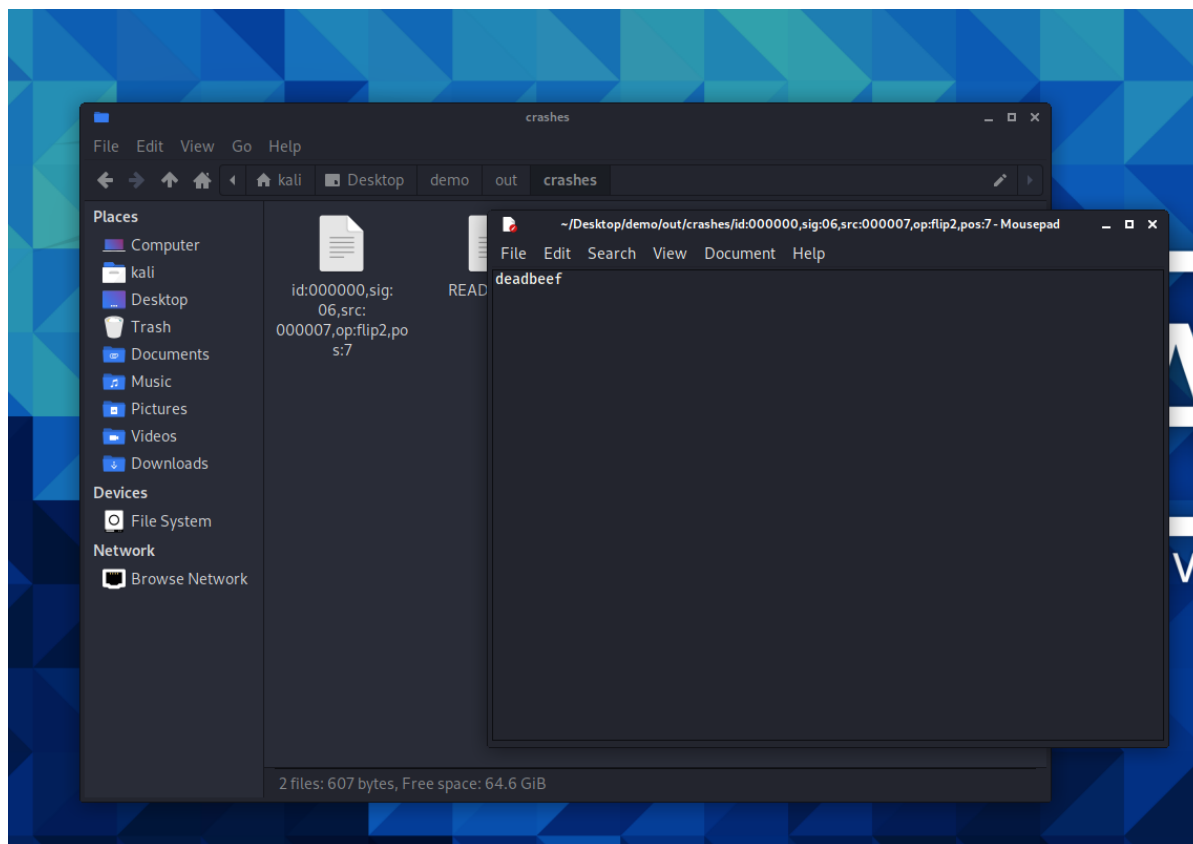
至此，我们就创建好了测试用例，接着就是启动模糊测试，然后观察结果。

2.3 启动模糊测试

我们使用如下的命令行 `afl-fuzz -i in -o out -- ./test @@` 来进行模糊测试的启动：



当观察到产生了一个`crash`时，我们去`out` → `crashes`中可以看到导致本次崩溃的输入：



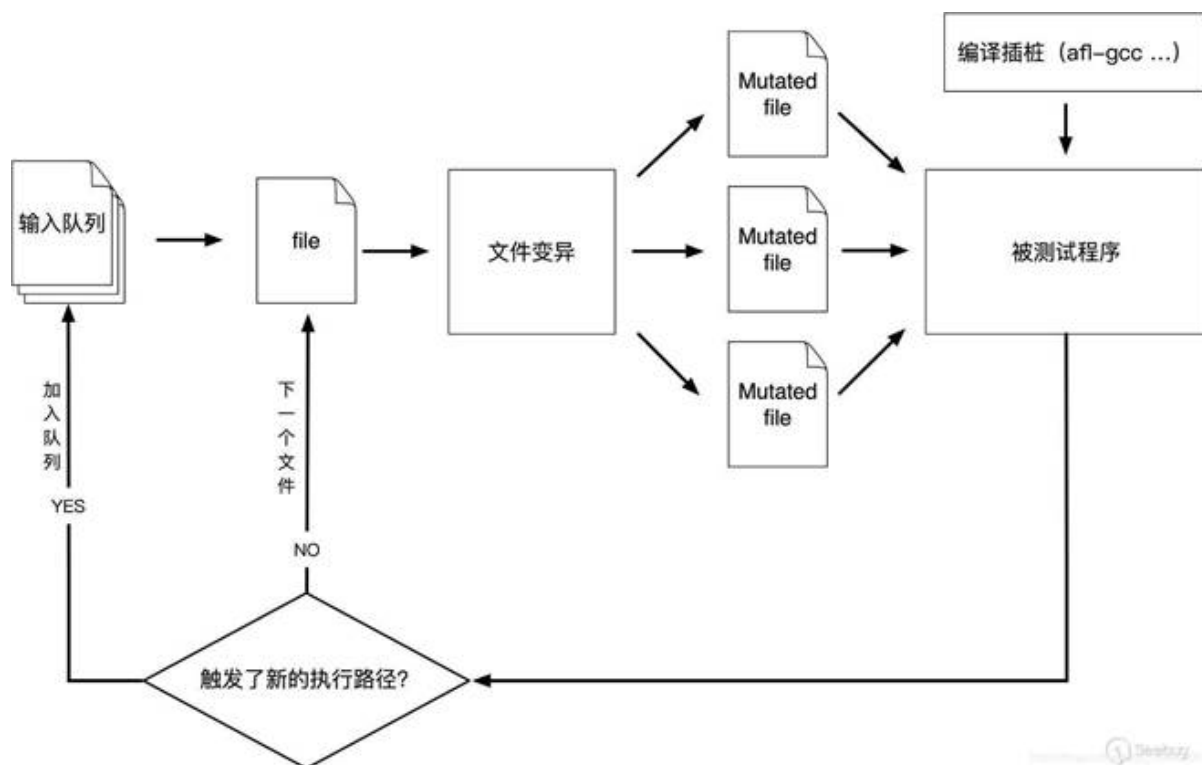
恰为我们之前分析的“`deadbeef`”，验证完毕！

心得体会：

对于AFL覆盖引导的理解：*AFL*是一款基于覆盖引导（*Coverage – guided*）的模糊测试工具，它通过记录输入样本的代码覆盖率，从而调整输入样本以提高覆盖率，增加发现漏洞的概率。

通过实验，进一步了解了*AFL*的工作流程：

1. 从源码编译程序时进行插桩，以记录代码覆盖率（*CodeCoverage*）；
2. 选择一些输入文件，作为初始测试集加入输入队列（*queue*）；
3. 将队列中的文件按一定的策略进行“突变”；
4. 如果经过变异文件更新了覆盖范围，则将其保留添加到队列中；
5. 上述过程会一直循环进行，期间触发了`crash`的文件会被记录下来。



这是本人第一次使用`linux`环境进行编程与调试，初步掌握了`linux`操作系统的操作模式，熟悉了利用命令行来做一些操作，也进一步熟悉了`linux`的一些常规操作语句，希望在后续的学习中能够更进一步的了解更多的操作语句。