

《漏洞利用及渗透测试基础》实验报告

姓名：齐明杰 学号：2113997 班级：信安2班

实验名称：

AFL 模糊测试实验

实验要求：

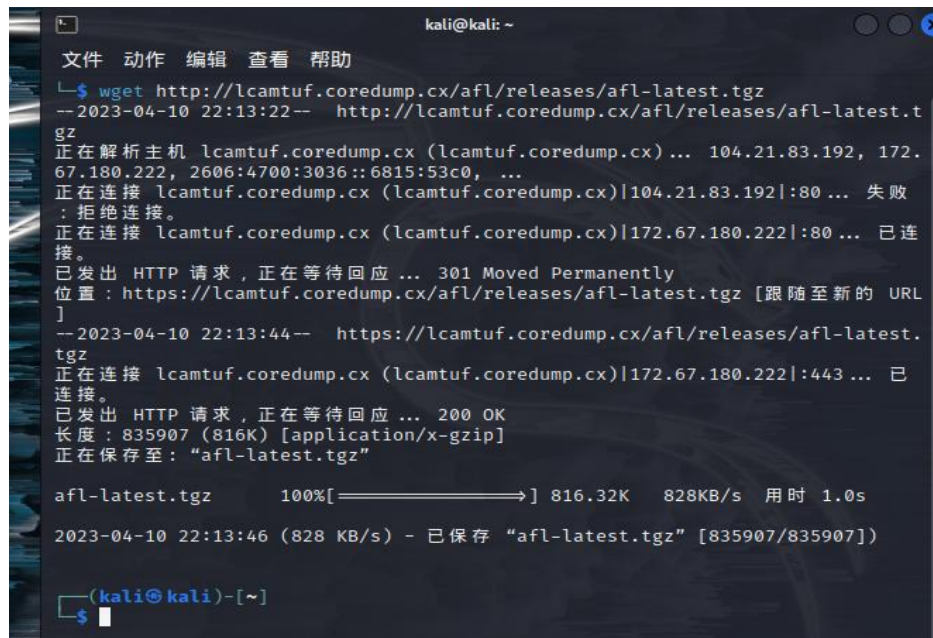
根据课本 7.4.5 章节，复现 AFL 在 KALI 下的安装、应用查阅资料理解覆盖引导和文件变异的概念和含义。

实验过程：

一、安装 AFL

1. 首先使用命令下载安装包：

wget http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz



```
kali@kali: ~  
文件 动作 编辑 查看 帮助  
└─$ wget http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz  
--2023-04-10 22:13:22-- http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz  
正在解析主机 lcamtuf.coredump.cx (lcamtuf.coredump.cx) ... 104.21.83.192, 172.67.180.222, 2606:4700:3036::6815:53c0, ...  
正在连接 lcamtuf.coredump.cx (lcamtuf.coredump.cx)|104.21.83.192|:80 ... 失败：拒绝连接。  
正在连接 lcamtuf.coredump.cx (lcamtuf.coredump.cx)|172.67.180.222|:80 ... 已连接。  
已发出 HTTP 请求，正在等待回应 ... 301 Moved Permanently  
位置：https://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz [跟随至新的 URL]  
--2023-04-10 22:13:44-- https://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz  
正在连接 lcamtuf.coredump.cx (lcamtuf.coredump.cx)|172.67.180.222|:443 ... 已连接。  
已发出 HTTP 请求，正在等待回应 ... 200 OK  
长度：835907 (816K) [application/x-gzip]  
正在保存至：“afl-latest.tgz”  
  
afl-latest.tgz      100%[=====>] 816.32K  828KB/s  用时 1.0s  
2023-04-10 22:13:46 (828 KB/s) - 已保存 “afl-latest.tgz” [835907/835907]  
  
(kali@kali)-[~]  
└─$
```

2. 解压命令：tar xvf afl-latest.tgz

```
kali@kali: ~  
文件 动作 编辑 查看 帮助  
(kali@kali)-[~]  
$ tar xvf afl-latest.tgz  
afl-2.52b/  
afl-2.52b/afl-as.h  
afl-2.52b/libtokencap/  
afl-2.52b/libtokencap/libtokencap.so.c  
afl-2.52b/libtokencap/README.tokencap  
afl-2.52b/libtokencap/Makefile  
afl-2.52b/alloc-inl.h  
afl-2.52b/config.h  
afl-2.52b/test-instr.c  
afl-2.52b/afl-analyze.c  
afl-2.52b/README  
afl-2.52b/afl-showmap.c  
afl-2.52b/experimental/  
afl-2.52b/experimental/clang_asm_normalize/  
afl-2.52b/experimental/clang_asm_normalize/as  
afl-2.52b/experimental/README.experiments  
afl-2.52b/experimental/distributed_fuzzing/  
afl-2.52b/experimental/distributed_fuzzing/sync_script.sh  
afl-2.52b/experimental/crash_triage/  
afl-2.52b/experimental/crash_triage/triage_crashes.sh  
afl-2.52b/experimental/libpng_no_checksum/  
afl-2.52b/experimental/libpng_no_checksum/libpng-nocrc.patch  
afl-2.52b/experimental/bash_shellshock/
```

3. 进入目标文件夹并且编译 AFL:

cd afl-2.52b

sudo make && sudo make install

```
kali@kali: ~/afl-2.52b  
文件 动作 编辑 查看 帮助  
(kali@kali)-[~]  
$ cd afl-2.52b  
(kali@kali)-[~/afl-2.52b]  
$ ls  
afl-analyze.c  afl-plot  dictionaries  Makefile  
afl-as.c       afl-showmap.c docs          qemu_mode  
afl-as.h       afl-tmin.c  experimental  QuickStartGuide.txt  
afl-cmin       afl-whatshup hash.h        README  
afl-fuzz.c     alloc-inl.h libdislocator testcases  
afl-gcc.c      config.h    libtokencap   test-instr.c  
afl-gotcpu.c   debug.h     llvm_mode     types.h  
(kali@kali)-[~/afl-2.52b]  
$ sudo make && sudo make install  
[sudo] kali 的密码 :  
[*] 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/" afl-gcc.c -o afl-gcc -ldl  
set -e; for i in afl-g++ afl-clang afl-clang++; do ln -sf afl-gcc $i; done  
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/" afl-fuzz.c -o afl-fuzz -ldl  
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/" afl-showmap.c -o afl-showmap -ldl  
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/" afl-tmin.c -o afl-tmin -ldl  
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/" afl-gotcpu.c -o afl-gotcpu -ldl  
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/" afl-analyze.c -o afl-analyze -ldl
```

```
文件 动作 编辑 查看 帮助
PATH="/usr/local/bin/" test-instr.c -o test-instr -ldl
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-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; ln -sf afl-clang-fast ${DESTDIR}/usr/local/bin/afl-clang-fast++; install -m 755 afl-llvm-pass.so afl-llvm-rt.o ${DESTDIR}/usr/local/lib/afl; 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)-[~/afl-2.52b]
$
```

之后，我们可以在/usr/local/bin 路径下找到 AFL：

```
cp -r dictionaries/ ${DESTDIR}/usr/local/share/afl

(kali@kali)-[~/afl-2.52b]
$ ls /usr/local/bin

afl-analyze  afl-clang++  afl-fuzz  afl-gcc  afl-plot  afl-tmin  mount-shared-folders
afl-clang  afl-cmin  afl-g++  afl-gotcpu  afl-showmap  afl-whatsup  restart-vm-tools

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

二、 AFL 测试

1. 创建本次实验的程序

新建文件夹 demo，并创建本次实验的程序 test.c，该代码编译后得到的程序如果被传入“deadbeef”则会终止，如果传入其他字符会原样输出：




```
~/demo/test.c - Mousepad
文件(F) 编辑(E) 搜索(S) 视图(V) 文档(D) 帮助(H)

1#include <stdio.h>
2#include <stdlib.h>
3int main(int argc, char **argv) {
4    char ptr[20];
5    if(argc>1){
6        FILE *fp = fopen(argv[1], "r");
7        fgets(ptr, sizeof(ptr), fp);
8    }
9    else{
10        fgets(ptr, sizeof(ptr), stdin);
11    }
12    printf("%s", ptr);
13    if(ptr[0] == 'd') {
14        if(ptr[1] == 'e') {
15            if(ptr[2] == 'a') {
16                if(ptr[3] == 'd') {
17                    if(ptr[4] == 'b') {
18                        if(ptr[5] == 'e') {
19                            if(ptr[6] == 'e') {
20                                if(ptr[7] == 'f')
21                                {
22                                    abort();
23                                }
24                                else
25                                {
26                                    printf("%c",ptr[7]);
27                                }
28                                else
29                                {
30                                    printf("%c",ptr[6]);
31                                }
32                                else
33                                {
34                                    printf("%c",ptr[5]);
35                                }
36                                else
37                                {
38                                    printf("%c",ptr[4]);
39                                }
40                                else
41                                {
42                                    printf("%c",ptr[3]);
43                                }
44                            }
45                        }
46                    }
47                }
48            }
49        }
50    }
51}
```

接下来使用 afl 的编译器编译，输入命令：afl-gcc -o test test.c，结果如下：

```
kali@kali: ~/demo
文件 动作 编辑 查看 帮助
(kali@kali)-[~/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)-[~/demo]
```

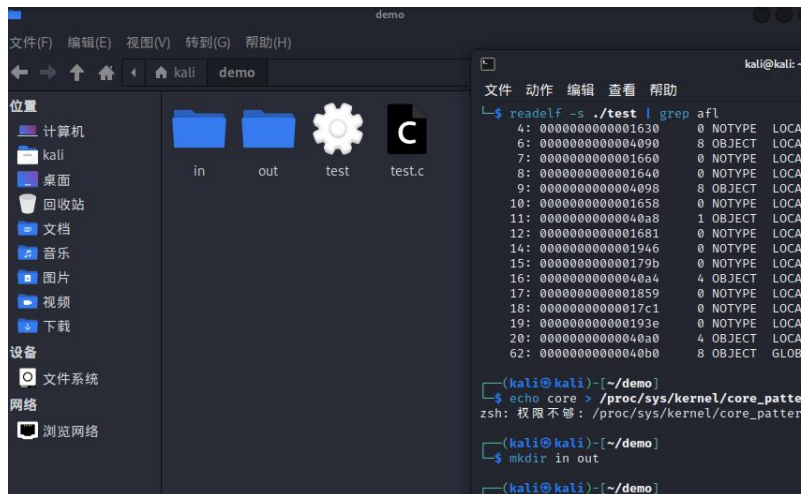
编译后会有插桩符号，使用如下命令验证：readelf -s ./test | grep afl，结果：

```
(kali@kali)-[~/demo]
$ readelf -s ./test | grep afl
4: 0000000000001630 0 NOTYPE LOCAL DEFAULT 15 __afl_maybe_log
6: 0000000000004090 8 OBJECT LOCAL DEFAULT 26 __afl_area_ptr
7: 0000000000001660 0 NOTYPE LOCAL DEFAULT 15 __afl_setup
8: 0000000000001640 0 NOTYPE LOCAL DEFAULT 15 __afl_store
9: 0000000000004098 8 OBJECT LOCAL DEFAULT 26 __afl_prev_loc
10: 0000000000001658 0 NOTYPE LOCAL DEFAULT 15 __afl_return
11: 00000000000040a8 1 OBJECT LOCAL DEFAULT 26 __afl_setup_failure
12: 0000000000001681 0 NOTYPE LOCAL DEFAULT 15 __afl_setup_first
14: 0000000000001946 0 NOTYPE LOCAL DEFAULT 15 __afl_setup_abort
15: 000000000000179b 0 NOTYPE LOCAL DEFAULT 15 __afl_forkserver
16: 00000000000040a4 4 OBJECT LOCAL DEFAULT 26 __afl_temp
17: 0000000000001859 0 NOTYPE LOCAL DEFAULT 15 __afl_fork_resume
18: 00000000000017c1 0 NOTYPE LOCAL DEFAULT 15 __afl_fork_wait_loop
19: 000000000000193e 0 NOTYPE LOCAL DEFAULT 15 __afl_die
20: 00000000000040a0 4 OBJECT LOCAL DEFAULT 26 __afl_fork_pid
62: 00000000000040b0 8 OBJECT GLOBAL DEFAULT 26 __afl_global_area_ptr
(kali@kali)-[~/demo]
```

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。

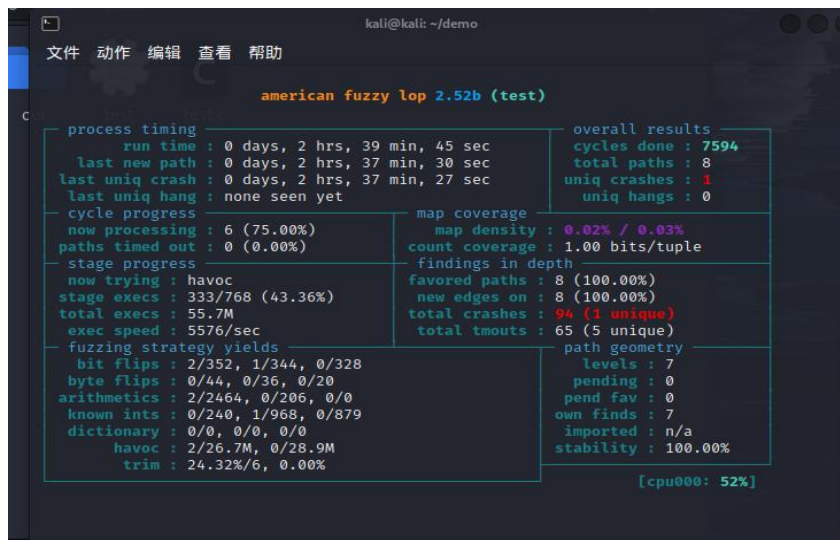
命令：echo hello> in/foo，结果如下图：



3. 启动模糊测试

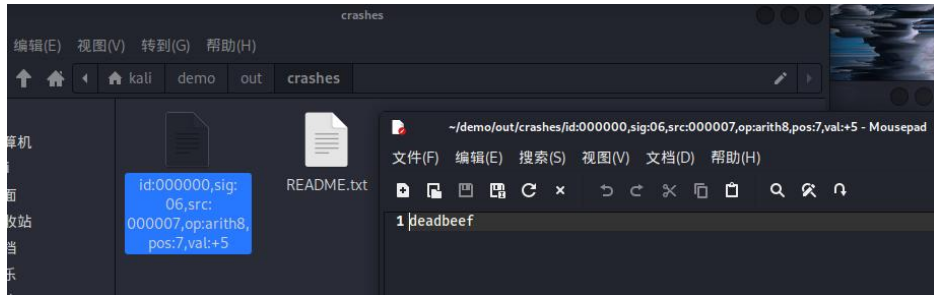
运行如下命令，开始启动模糊测试：

命令：afl-fuzz -i in -o out -- ./test @@，运行一段时间后，结果如下图：



4. 分析 crash

在 out 文件夹下的 crashes 子文件夹里面是我们产生 crash 的样例，如下图所示：



可见，这正是使我们代码产生崩溃的输入。

心得体会：

通过这次实验，我深刻认识到 AFL 是一款非常强大的模糊测试工具。相比于传统的手动测试，AFL 可以自动化地处理大量的测试用例，并通过覆盖率的记录和调整，增加发现漏洞的概率。同时，AFL 的安装和使用也非常简便，不需要复杂的配置和编程知识。这使得 AFL 在安全领域广受欢迎。总之，这次实验让我对 AFL 有了更深刻的理解和认识，也让我更加热爱信息安全这个领域。