《漏洞利用及渗透测试基础》实验报告

姓名：张洋　　学号：2111460　　班级：信安二班

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：

1. **进入 Kali，打开控制台**

2. **AFL 安装**

利用 sudo apt-get install afl 即可安装。

```
┌──(kali㉿kali)-[~]
└─$ sudo apt-get install afl
```

查看路径可以看到 afl 安装的文件：ls /usr/bin/afl*

```
┌──(kali㉿kali)-[~]
└─$ ls /usr/bin/afl*
/usr/bin/afl-analyze        /usr/bin/afl-gcc-fast
/usr/bin/afl-c++            /usr/bin/afl-g++-fast
/usr/bin/afl-cc             /usr/bin/afl-gotcpu
/usr/bin/afl-clang          /usr/bin/afl-ld-lto
/usr/bin/afl-clang++        /usr/bin/afl-lto
/usr/bin/afl-clang-fast     /usr/bin/afl-lto++
/usr/bin/afl-clang-fast++   /usr/bin/afl-network-client
/usr/bin/afl-clang-lto      /usr/bin/afl-network-server
/usr/bin/afl-clang-lto++    /usr/bin/afl-persistent-config
/usr/bin/afl-cmin           /usr/bin/afl-plot
/usr/bin/afl-cmin.bash      /usr/bin/afl-showmap
/usr/bin/afl-fuzz           /usr/bin/afl-system-config
/usr/bin/afl-g++            /usr/bin/afl-tmin
/usr/bin/afl-gcc            /usr/bin/afl-whatsup
```

3. AFL 测试

（1）**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：

Terminal

File Edit View Search Terminal Help

```
#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;
}
~
```

使用 afl 的编译器编译，可以使模糊测试过程更加高效。

命令：**afl-gcc -o test test.c**

```
┌──(kali㉿kali)-[~/demo1]
└─$ afl-gcc -o test test.c
afl-cc++4.04c by Michal Zalewski, Laszlo Szekeres, Marc Heuse - mode: GCC-GCC
[!] WARNING: You are using outdated instrumentation, install LLVM and/or gcc-
plugin and use afl-clang-fast/afl-clang-lto/afl-gcc-fast instead!
afl-as++4.04c by Michal Zalewski
[+] Instrumented 14 locations (64-bit, non-hardened mode, ratio 100%).
```

编译后会有插桩符号，使用下面的命令可以验证这一点。

命令：**readelf -s ./test | grep afl**

```
┌──(kali㉿kali)-[~/demo1]
└─$ readelf -s ./test | grep afl
     4: 0000000000001628     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: 0000000000001638     0 NOTYPE  LOCAL  DEFAULT   15 __afl_store
     9: 0000000000004098     8 OBJECT  LOCAL  DEFAULT   26 __afl_prev_loc
    10: 0000000000001655     0 NOTYPE  LOCAL  DEFAULT   15 __afl_return
    11: 00000000000040a8     1 OBJECT  LOCAL  DEFAULT   26 __afl_setup_failur
e
    12: 0000000000001681     0 NOTYPE  LOCAL  DEFAULT   15 __afl_setup_first
    14: 0000000000001949     0 NOTYPE  LOCAL  DEFAULT   15 __afl_setup_abort
    15: 000000000000179e     0 NOTYPE  LOCAL  DEFAULT   15 __afl_forkserver
    16: 00000000000040a4     4 OBJECT  LOCAL  DEFAULT   26 __afl_temp
    17: 000000000000185c     0 NOTYPE  LOCAL  DEFAULT   15 __afl_fork_resume
    18: 00000000000017c4     0 NOTYPE  LOCAL  DEFAULT   15 __afl_fork_wait_lo
op
    19: 0000000000001941     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
```

进行下一步之前，还需要输入如下命令指示系统将 coredumps 输出为文件，而不是将它们发送到特定的崩溃处理程序应用程序。

命令：echo core > /proc/sys/kernel/core_pattern

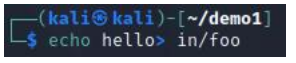
（2）创建测试用例

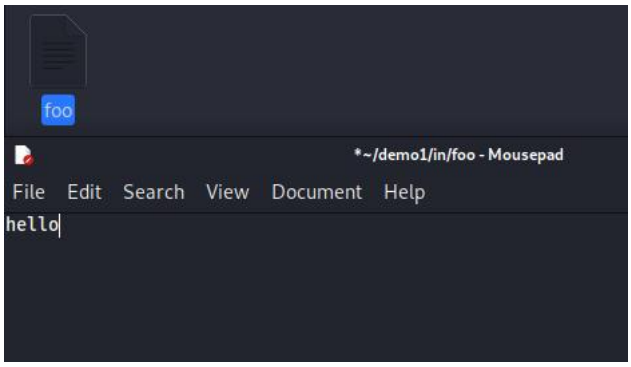
首先，创建两个文件夹 in 和 out，分别存储模糊测试所需的输入和输出相关的内容。

命令：mkdir in out

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



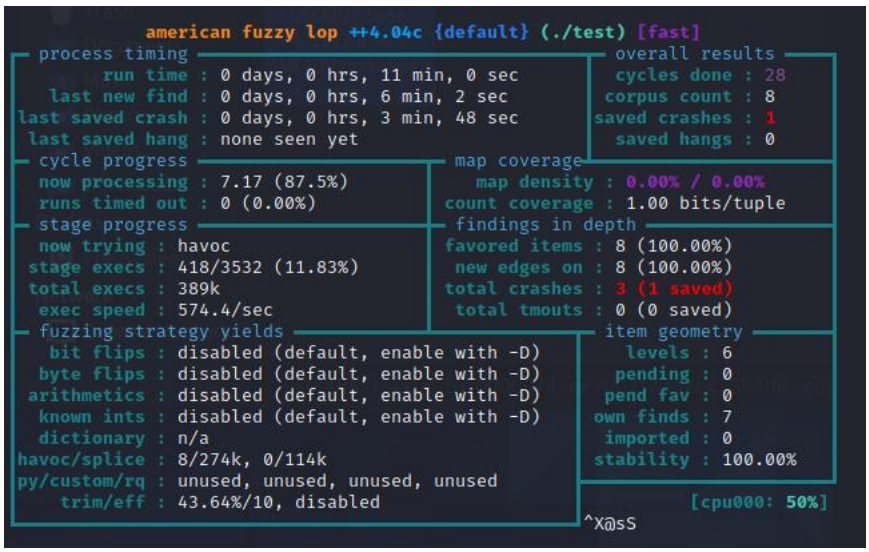


foo 就是我们的测试用例，里面包含初步字符串 hello。AFL 会通过这个语料进行变异，构造更多的测试用例。

（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 @@

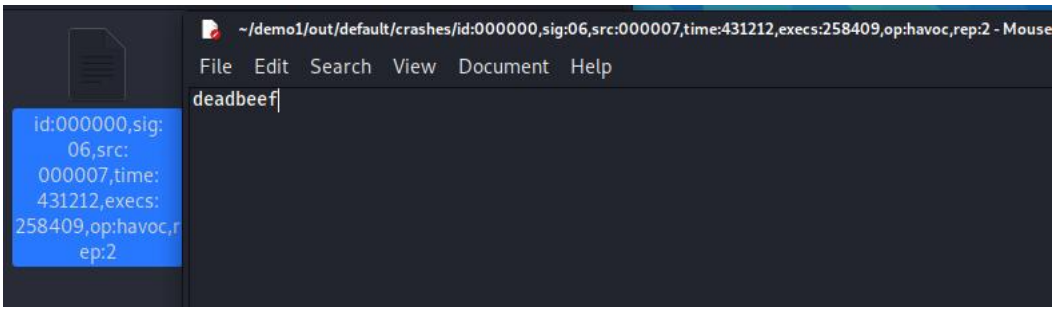


process timing：这里展示了当前 fuzzer 的运行时间、最近一次发现新执行路径的 时间、最近一次崩溃的时间、最近一次超时的时间。

overall results：这里包括运行的总周期数、总路径数、崩溃次数、超时次数。其中，总周期数可以用来作为何时停止 fuzzing 的参考。随着不断地 fuzzing，周期数会不断增大，其颜色也会由洋红色，逐步变为黄色、蓝色、绿色。一般来说，当其变为绿色时，代表可执行的内容已经很少了，继续 fuzzing 下去也不会有什么新的发现了。此时，我们便可以通过 Ctrl-C，中止当前的 fuzzing。

stage progress：这里包括正在测试的 fuzzing 策略、进度、目标的执行总次数、目标的执行速度。执行速度可以直观地反映当前跑的快不快，如果速度过慢，比如低于 500 次每秒，那么测试时间就会变得非常漫长。如果发生了这种情况，那么我们需要进一步调整优化我们的 fuzzing。

（4）分析 crash

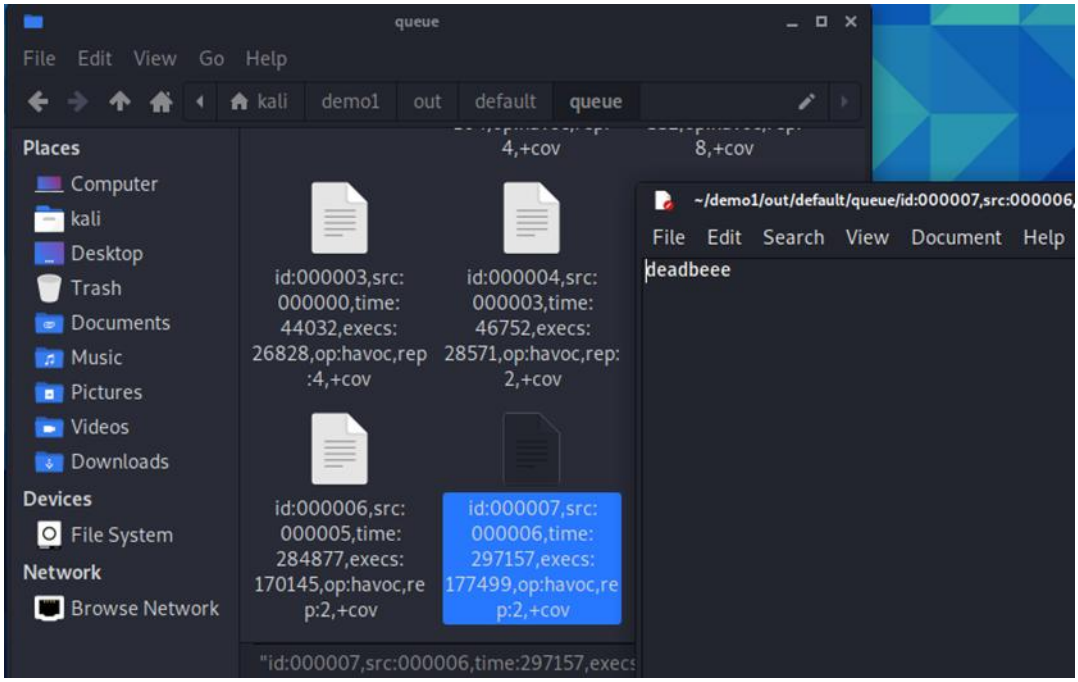


在 out 文件夹下的 crashes 子文件夹里面是我们产生 crash 的样例，hangs 里面是产生超时的样例，queue 里面是每个不同执行路径的测试用例。

通常，得到 crash 样例后，可以将这些样例作为目标测试程序的输入，重新触发异常并跟踪运行状态，进行分析、定位程序出错的原因或确认存在的漏洞类型。

3. 覆盖引导和文件变异的概念以及含义

变异字符存放在/out/default/queue 文件夹下的文档中。



覆盖引导是一种测试方法，它通过对应用程序的代码路径进行监视来确定哪些输入数据可以探索更多的代码路径。在这种方法中，AFL 会收集输入数据的覆盖率信息，并使用这些信息来生成更多有可能达到新代码路径的测试用例。

文件变异是另一种测试方法，它通过对输入文件进行随机变异来创建新的测试用例。变异可以包括添加、删除、替换或重新排列输入文件中的字节。这种方法通常可以在短时间内发现一些易于检测的错误，但并不一定能够发现所有类型的漏洞。

心得体会：

在进行 AFL 模糊测试实验时，我深刻认识到模糊测试的重要性以及 AFL 的强大功能。以下是我对模糊测试实验的总结和体会：

1. 模糊测试是一种有效的软件测试方法，它可以自动化地发现应用程序中的漏洞和错误。通过模糊测试，我们可以对应用程序进行全面的测试，并发现不同类型的漏洞和错误。
2. AFL 是一种非常强大的模糊测试工具，它具有许多有用的功能和选项，可以帮助我们进行高效的模糊测试。例如，AFL 可以自动化地生成测试用例，收集代码覆盖率信息，并使用多种测试模式来测试不同类型的应用程序。
3. 在进行模糊测试时，我们需要了解应用程序的基本结构和功能，并识别可能的输入点。我们还需要了解测试用例的生成方法，并根据需要对其进行修改和定制。
4. 模糊测试需要一定的计算资源和时间，特别是当我们测试大型应用程序时。因此，我们需要为测试分配足够的计算资源，并制定测试计划和策略，以优化测试效率和结果。
5. 最后，我们需要对测试结果进行分析和总结，以确定应用程序中的漏洞和错误，并提供有用的建议和解决方案。这需要我们具备一定的技术和经验，以便有效地解决发现的问题。

总之，模糊测试是一种非常重要的软件测试方法，可以帮助我们发现应用程序中的漏洞和错误。AFL 是一种非常强大的模糊测试工具，可以帮助我们进行高效的模糊测试。在进行模糊测试实验时，我们需要了解应用程序的基本结构和功能，分配足够的计算资源，并对测试结果进行分析和总结。