《软件安全》实验报告

姓名： 邢清画 学号：2211999 班级：1023（物联网）

**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。

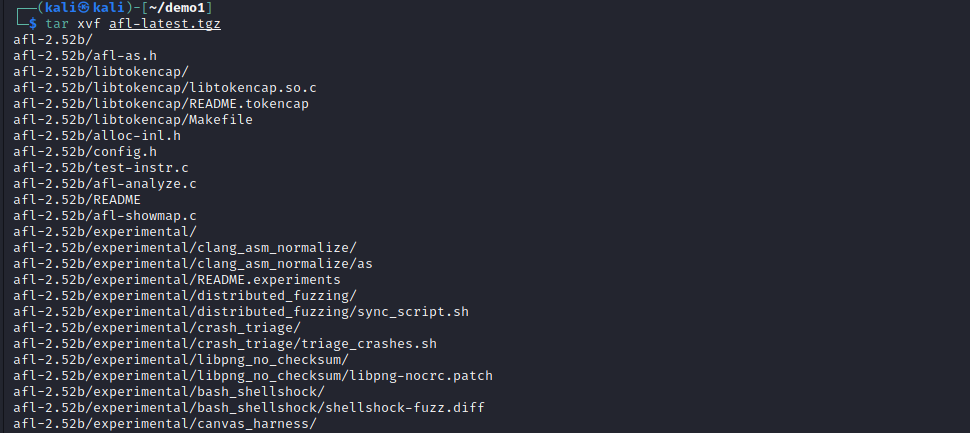
**实验过程：**

1. 打开虚拟机进入Kali Linux 系统，新建demo1，启动控制台，安装AFL。

利用sudo apt-get install afl，发现无法安装，因为Linux 的 AFL 已经停止更新。

使用wget http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz进行安装。

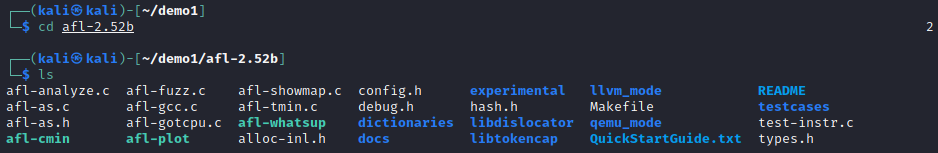


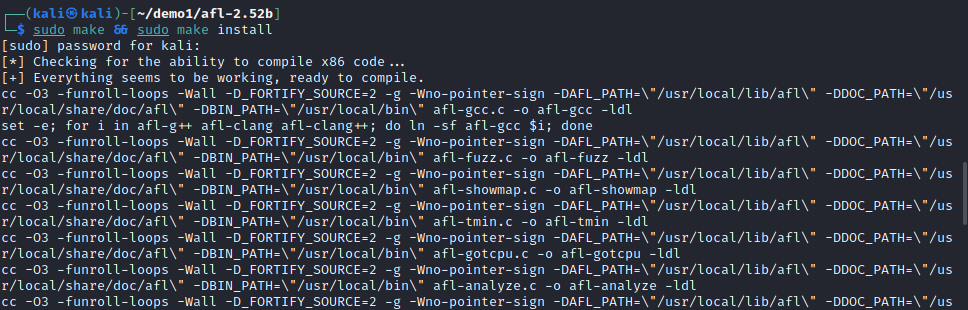
下载完成之后进行解压：  


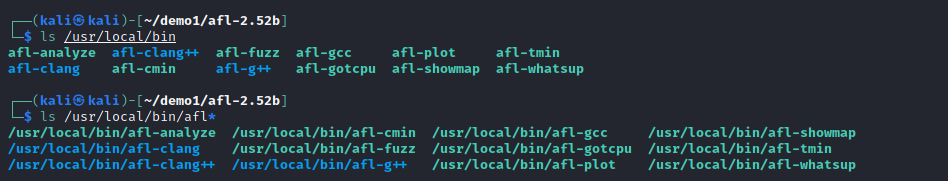
安装成功后，利用

cd afl-2.52b > ls > sudo make && sudo make install > ls /usr/local/bin > ls /usr/local/bin/afl\*

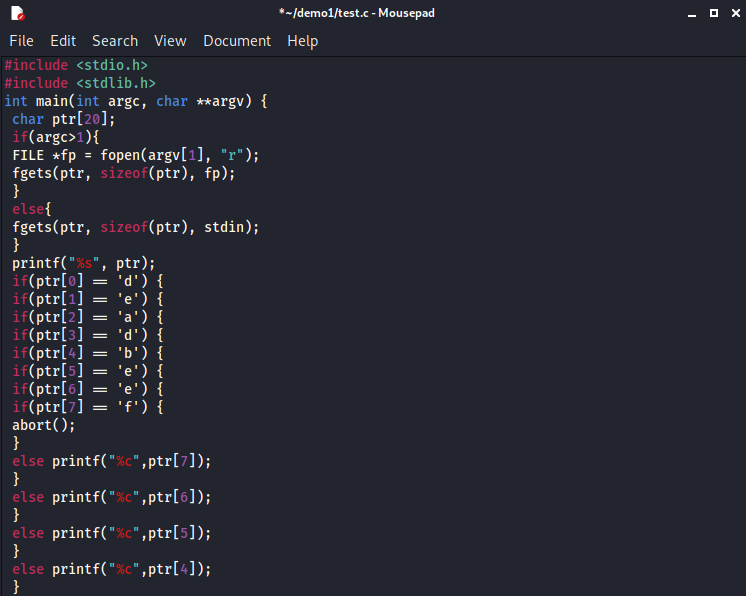
这些语句可以找到AFL的一些可执行文件







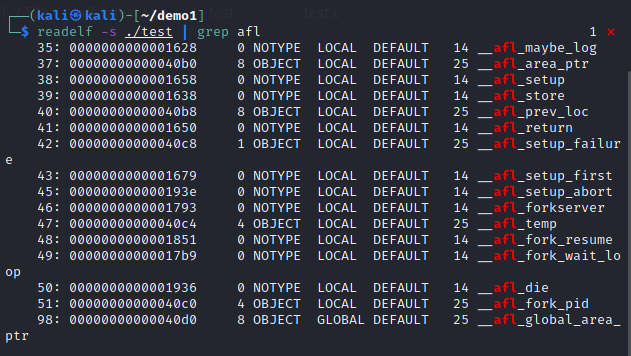
2. 在demo1 文件夹中新建文本文件test.c，复制实验代码。该代码编译后得到的程序如果传入“deadbeef”则会终止，如果输入其他字符则会原样输出。代码如下图所示：



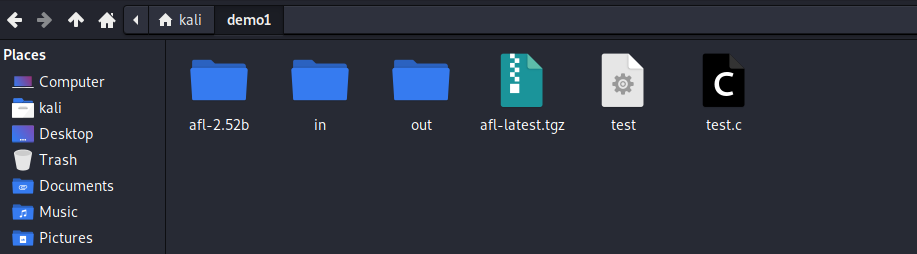
使用afl编译器，用afl-gcc命令afl-gcc -o test test.c创建本次的实验文件test:



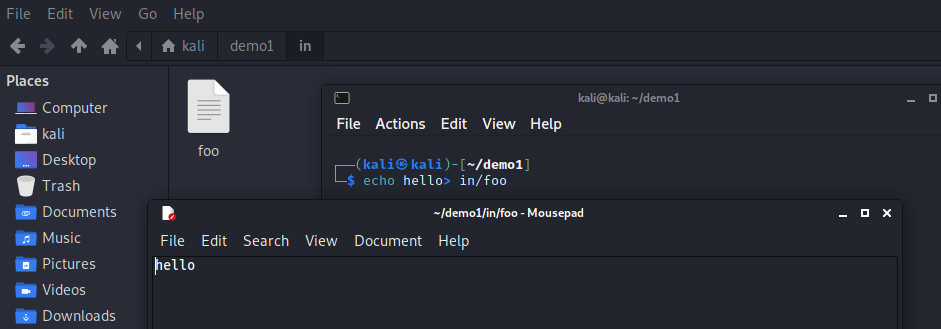
编译之后出现插桩，输入命令：readelf -s ./test | grep afl



创建输入（in）输出（out）文件夹，用来存储测试所需的相关输入输出信息

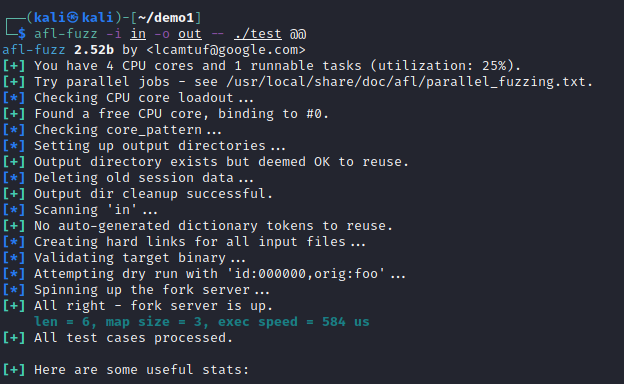


使用命令echo hello> in/foo，在in文件夹中写入hello

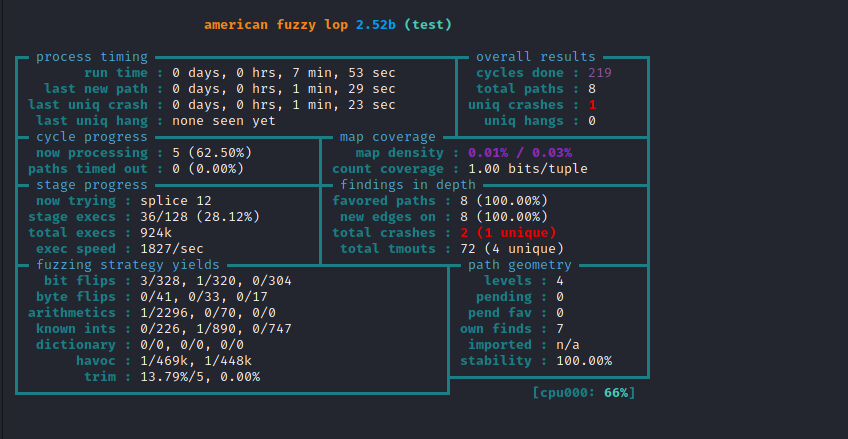


1. 有输入文件之后启动测试

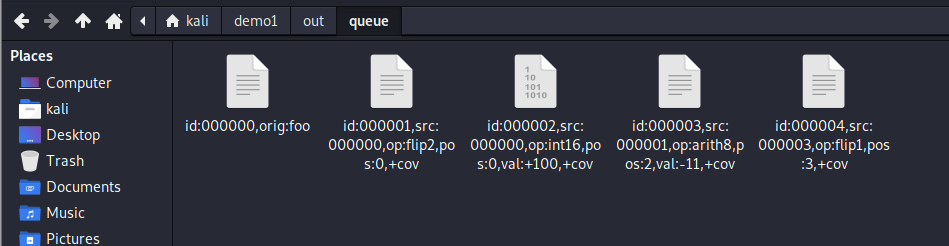
运行 afl-fuzz -i in -o out -- ./test @@ 命令，开始启动模糊测试：



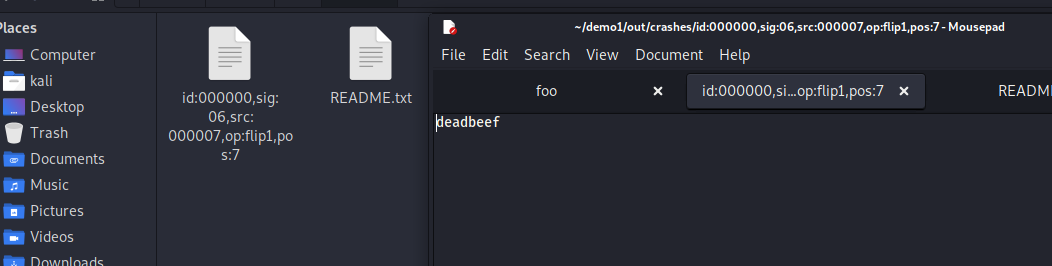
运行一段时间后，发现 total crashes 变为非0个数，即出现中断



在队列中可以看到已经产生了很多关于触发异常的记录



在out 文件夹中，打开 crashes 文件夹，打开错误日志，观察到字符串“deadbeef”。



**心得体会：**

1. 覆盖引导与文件变异的理解

在AFL模糊测试的上下文中，“覆盖引导”（coverage-guided）不同于面向对象编程中的覆盖概念。在AFL中，覆盖引导是指利用程序执行时收集的路径覆盖信息来优化后续的测试用例生成。这种方法能够有效地指导模糊测试工具生成那些探索新代码路径的测试数据，从而提高测试的全面性和发现错误的概率。

文件变异在AFL中指的是基于现有测试用例生成新测试用例的过程，其中原始的测试数据会经历各种变化，如位翻转、插入、删除等，以产生潜在能触发未探索代码路径的新数据。这些变异操作是自动化的，能够在短时间内生成大量的测试用例。

2. AFL的工作原理与实验步骤

AFL通过结合遗传算法和确定性模糊测试技术来有效地生成覆盖新代码路径的测试用例。其工作流程包括：

准备阶段： 对目标程序进行插桩处理，这一步是为了让AFL能够在运行时收集必要的执行路径信息。

运行AFL： 输入插桩后的程序，AFL开始自动地生成测试用例并执行，不断重复这一过程。

监控与调试： AFL自动监控程序执行情况，并在探索到新的代码路径时保存相应的测试用例。此外，AFL提供了多种工具以帮助用户分析和调试测试结果。

漏洞分析： 当AFL发现可能导致程序崩溃的输入时，用户需要对这些情况进行分析，确定是否存在安全漏洞，并进行相应的分类和修复。

3. 实验体会

通过这次实验，我不仅加深了对Linux系统的熟悉程度，也深入理解了AFL模糊测试工具的工作原理及其应用。AFL通过自动化的测试用例生成和覆盖率优化，极大地提高了发现程序安全漏洞的效率和可能性，相较于传统的手动测试方法，它能更快地识别和修复潜在的安全风险。

总结而言，AFL是一个强大的模糊测试工具，它的高效和自动化特性使得安全测试更加全面和可靠。通过实际操作和应用，我对模糊测试的理论与实践都有了更深刻的认识和体会。