《漏洞利用及渗透测试基础》实验报告

姓名：孙蕗 学号：2112060 班级：信安1班

**实验名称：**

AFL模糊测试实验

**实验要求：**

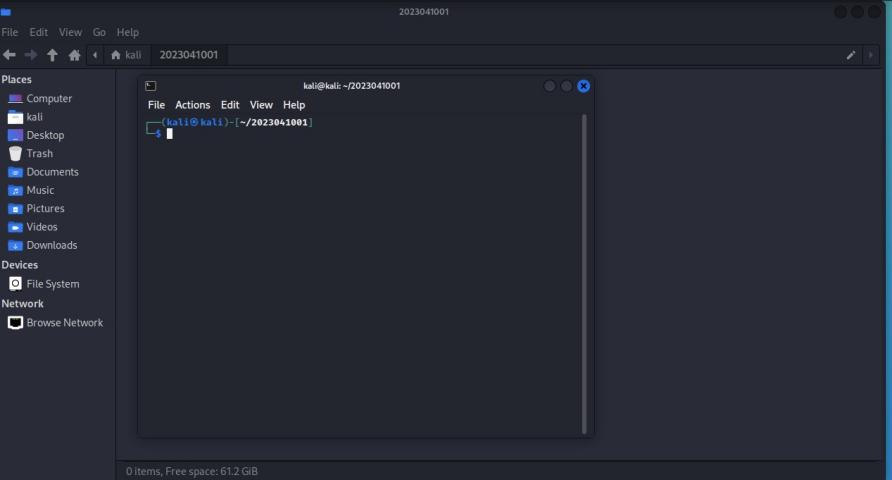
根据课本7.4.5章节，复现AFL在KALI下的安装、应用，查阅资料理解覆盖引导和文件变异的概念和含义。

**实验过程：**

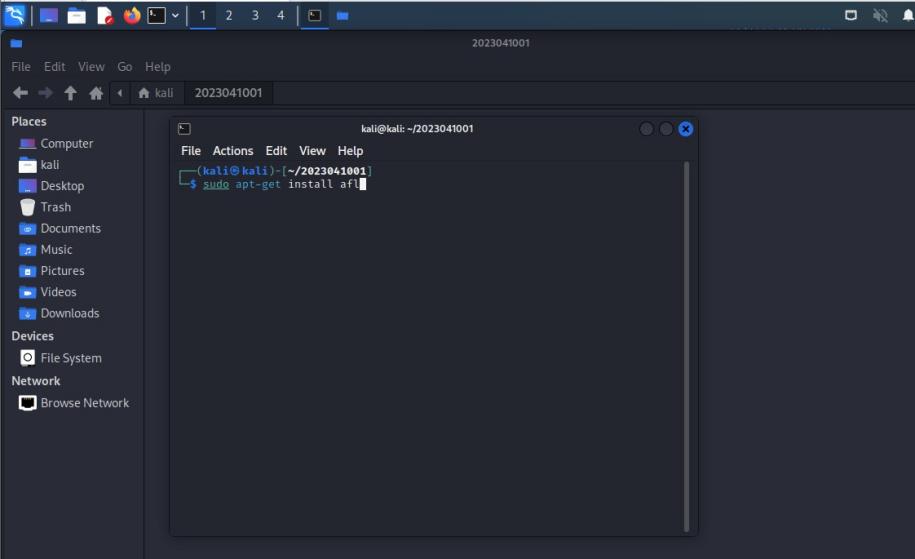
AFL可以进行白盒测试，也可以进行黑盒测试

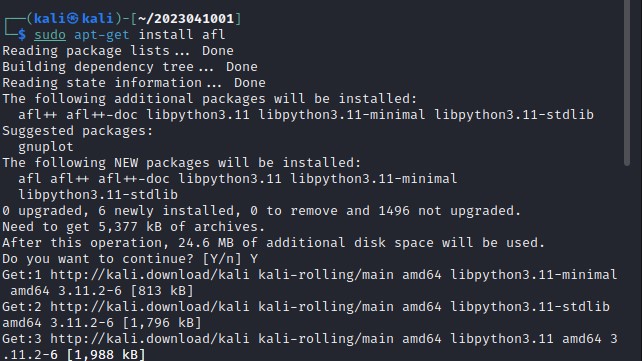
以一个白盒模糊测试为例，尝试AFL的简单用法

1. 启动控制台

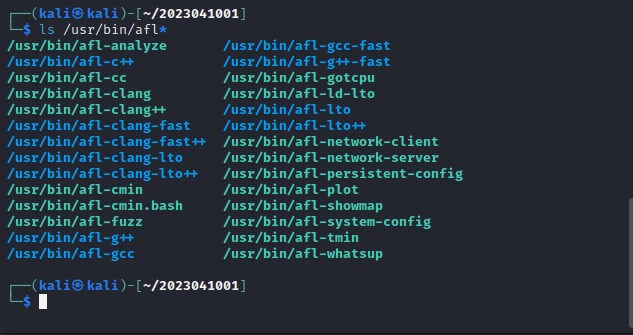


1. sudo apt-get install afl安装AFL





1. ls /usr/bin/afl\* 查看路径可以看到AFL安装的文件



afl-gcc和afl-g++分别对应的是gcc和gcc++的封装

afl-clang和afl-clang++分别对应chang的C和C++编译器的封装

afl-fuzz是AFL的主体，用于对目标程序进行模糊测试

afl-analyze可以对用例进行分析，看能否发现用例中有意义的字段

afl-plot生成测试任务的状态图

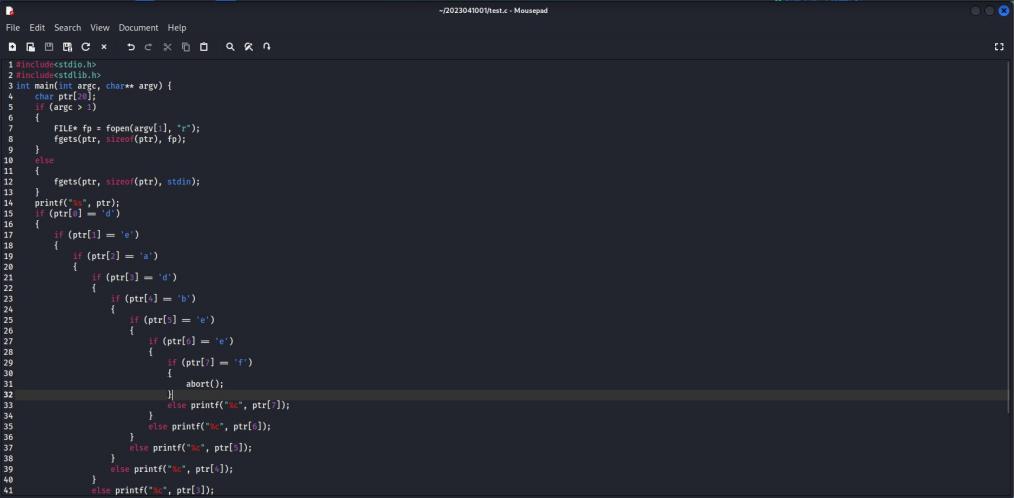
afl-tmin和afl-cmin对用例进行简化

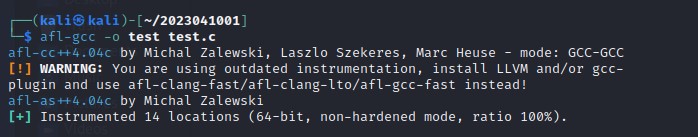
afl-whatsup用于查看模糊测试任务的状态

afl-gotcpu用于查看当前CPU状态

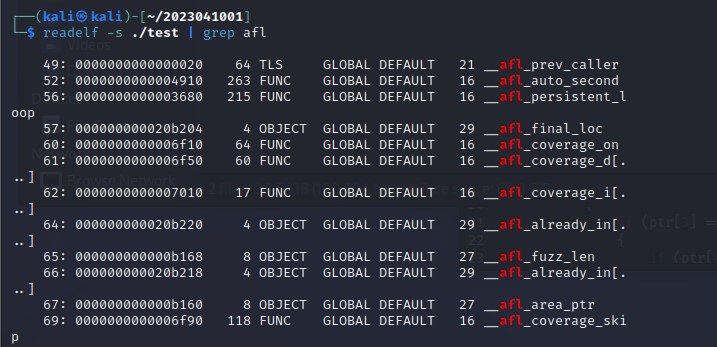
afl-showmap用于对单个用例进行测试路径跟踪

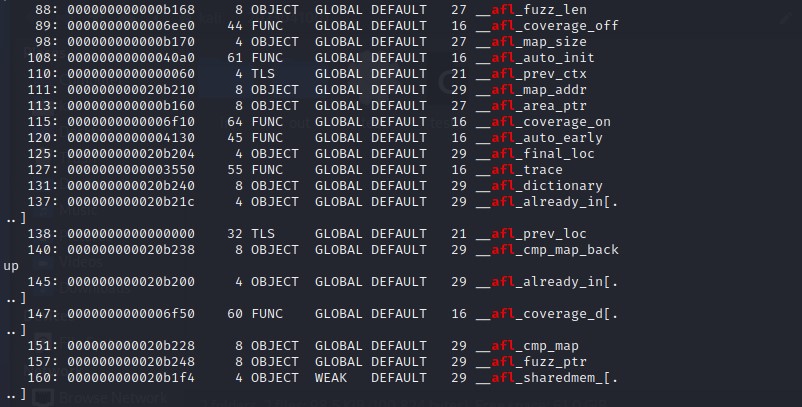
1. afl-gcc -o test test.c建立test.c文件，如果依次输入deadbeef这些字符，会导致中断，如果传入其他字符会原样输出。



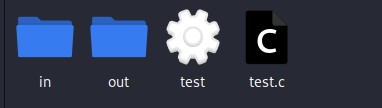


1. 编译后会有插桩，进行验证

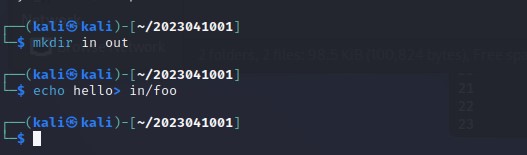




1. 输入命令echo core > /proc/sys/kernel/core\_pattern指示系统输出将coredumps输出为文件，而不是将它们发送到特定的崩溃处理程序。
2. 创建测试用例文件夹



1. 创建测试用例

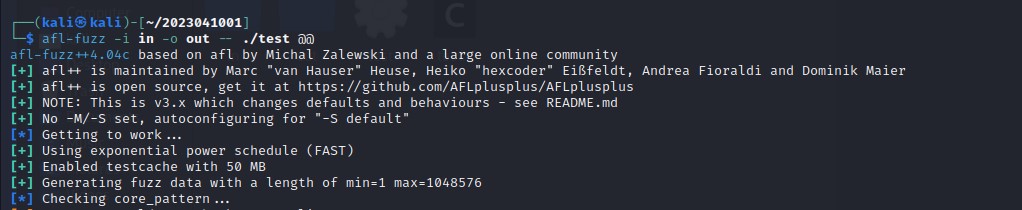


创建文件夹in和out，分别存储模糊测试所需的输入输出相关内容。

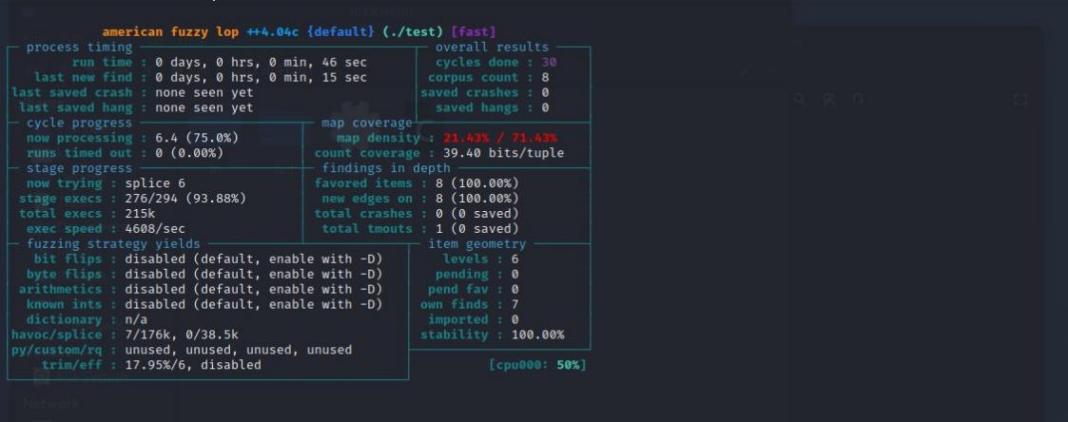
在文件夹中创建一个包含字符串hello的文件

foo为测试用例，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。

1. 启动模糊测试



1. winafl运行界面

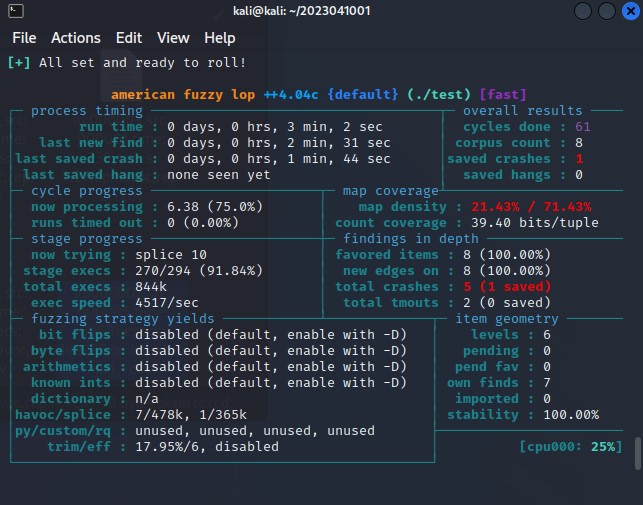


processing timing展示了当前Fuzzer的运行时间、最近一次发现新执行路径的时间、最近一次崩溃的时间、最近一次超时的时间。

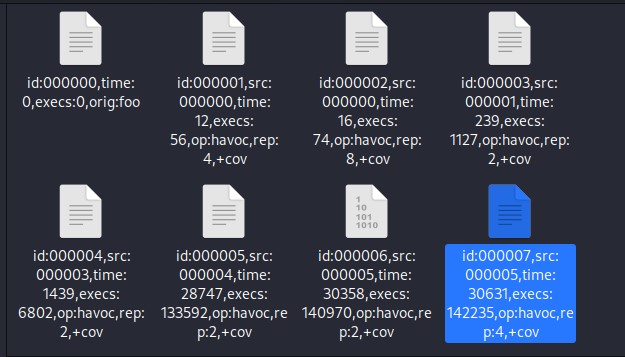
overall results包括运行的总周期数、总路径数、崩溃次数、超时次数。其中总周期数可以用来作为何时停止模糊测试的参考。随着不断地进行模糊测试，周期数会不断增大，其颜色也会由洋红色逐步变为黄色、蓝色、绿色。一般来说，当其变为绿色时，代表可执行的内容已经很少了，继续进行模糊测试也不会有什么新的发现了。此时，可以通过ctrl-c命令中止当前的模糊测试 。

stage processing包括正在执行的模糊测试策略、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执行的快慢，如果速度过慢，如低于500次每秒，那么测试时间会变得非常漫长。如果发生了这种情况，就需要进一步调整优化模糊测试。

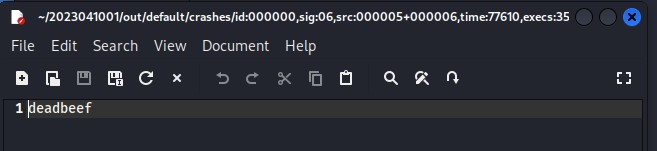
1. 运行结束



1. queue记录



1. crash结果



在out文件夹下的crashes子文件夹里面是产生的crash样例，hangs子文件夹里面是产生超时的样例，queue子文件夹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，分析、定位程序出错的原因或确认存在的漏洞类型。

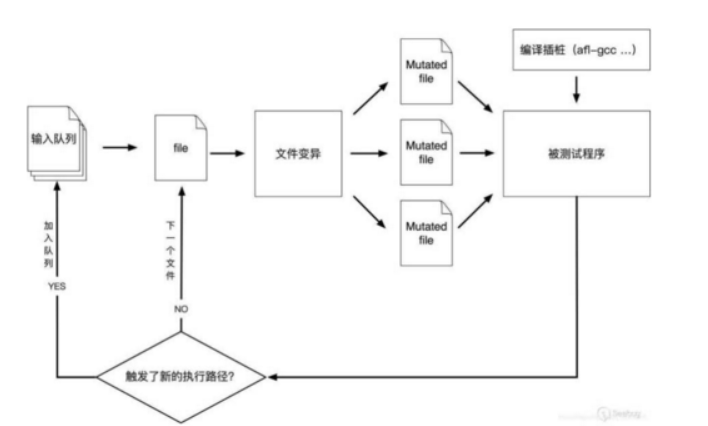
**心得体会：**

AFL是一款基于覆盖引导的模糊测试工具，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。AFL主要用于C/C++程序的测试，被测程序有无程序源代码均可，有源代码时可以对源代码进行编译时插桩，无源代码时可以借助QEMU的用户模式进行二进制插桩。

其工作流程大致如下：

1. 从源代码编译程序时进行插桩，以记录代码覆盖率
2. 选择一些输入文件，作为初始测试集加入输出队列
3. 将队列中的文件按一定的策略进行突变
4. 如果经过编译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。

AFL工具工作流程图如下图所示。



覆盖引导，即通过向目标程序插桩，统计代码覆盖，反馈给模糊测试引擎（fuzzer，即模糊测试工具），反馈信息用于变异种子，生成更高质量的输入，使得 fuzzer 能够用更好的输入让被测程序达到更高的代码覆盖率。

对于每个目标，fuzzer 都会构建一个输入的语料库，随着 fuzzer 通过变异语料库发现新的输入，覆盖率会不断增长。常见的模糊测试引擎（fuzzer，即模糊测试工具）有 afl、honggfuzz、libfuzzer 以及本次所说的 syzkaller，这些都是经典的 fuzzing 工具。

AFL维护了一个队列(queue)，每次从这个队列中取出一个文件，对其进行大量变异，并检查运行后是否会引起目标崩溃、发现新路径等结果。变异的主要类型如下：

bitflip，按位翻转，1变为0，0变为1

arithmetic，整数加/减算术运算

interest，把一些特殊内容替换到原文件中

dictionary，把自动生成或用户提供的token替换/插入到原文件中

havoc，中文意思是“大破坏”，此阶段会对原文件进行大量变异，具体见下文

splice，中文意思是“绞接”，此阶段会将两个文件拼接起来得到一个新的文件