《软件安全》实验报告

姓名： 冯思程 学号： 2112213 班级： 1120（张健）

**实验名称：**

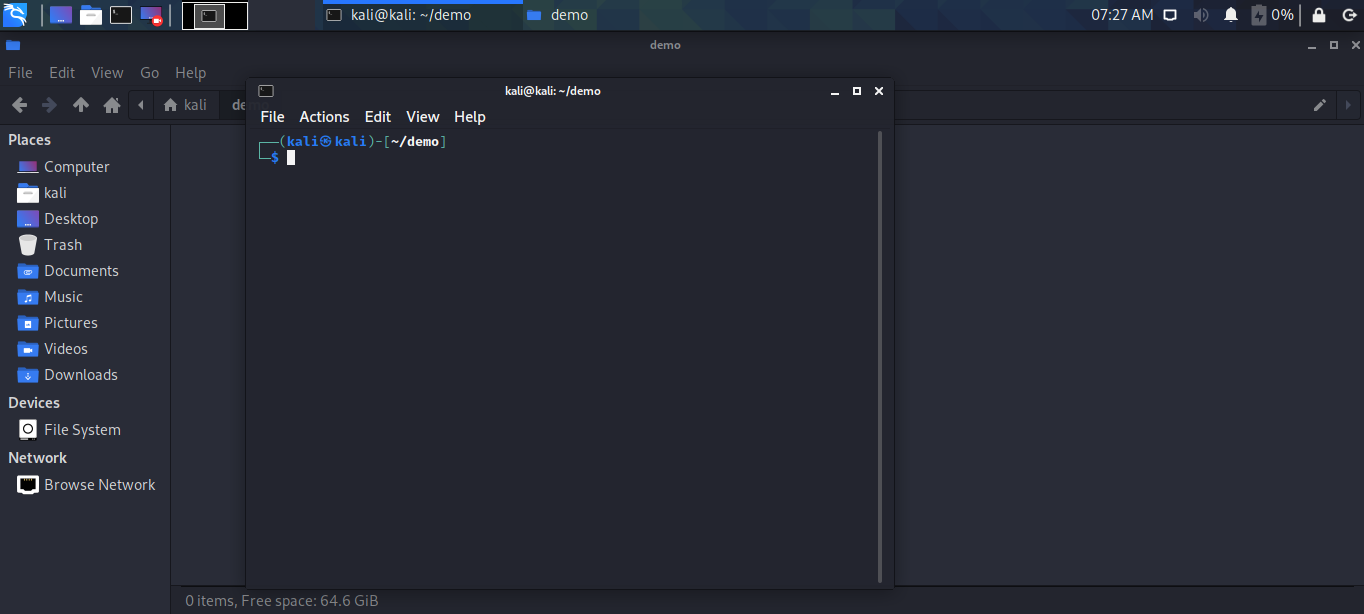
AFL模糊测试实验

**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. 在kali下安装AFL，下载kali然后配置好linux系统虚拟机，打开界面，新建文件夹demo，点开文件夹并打开控制台，如下图：



由于原语句sudo apt-get install afl已经失效无法进行afl的安装。所以根据csdn博客进行安装：

（1）首先使用命令下载安装包：

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

1. 然后解压：

tar xvf afl-latest.tgz

1. 进入目标文件夹并且编译AFL

cd afl-2.52b

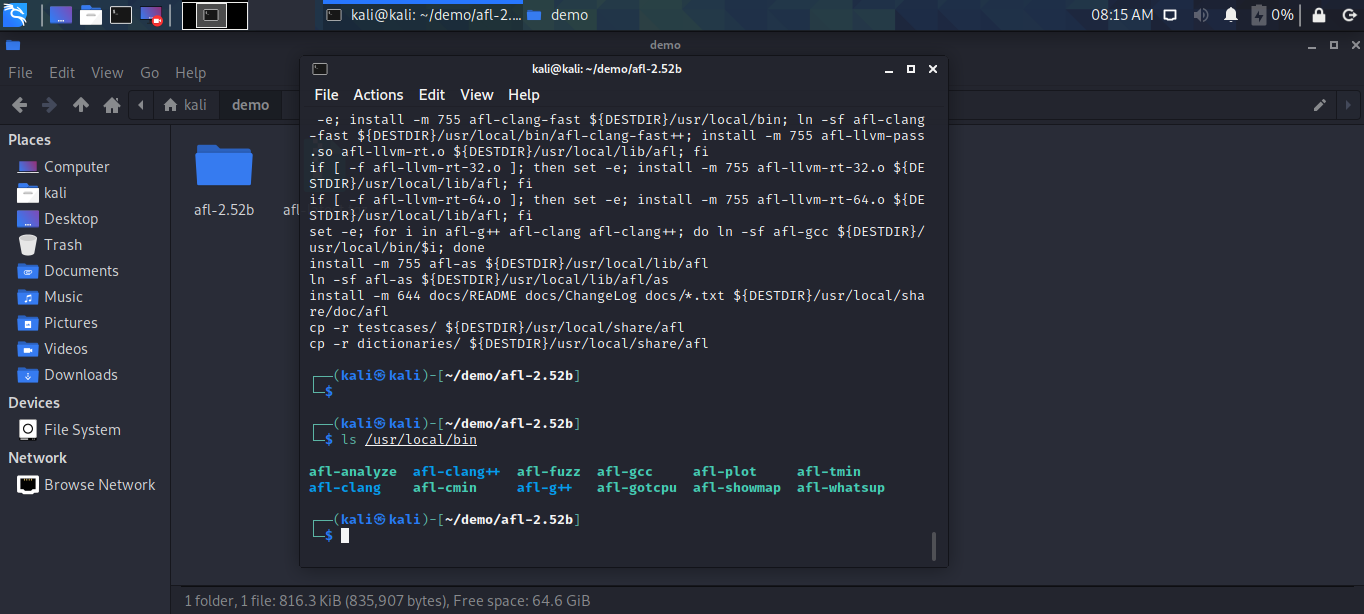
sudo make && sudo make install

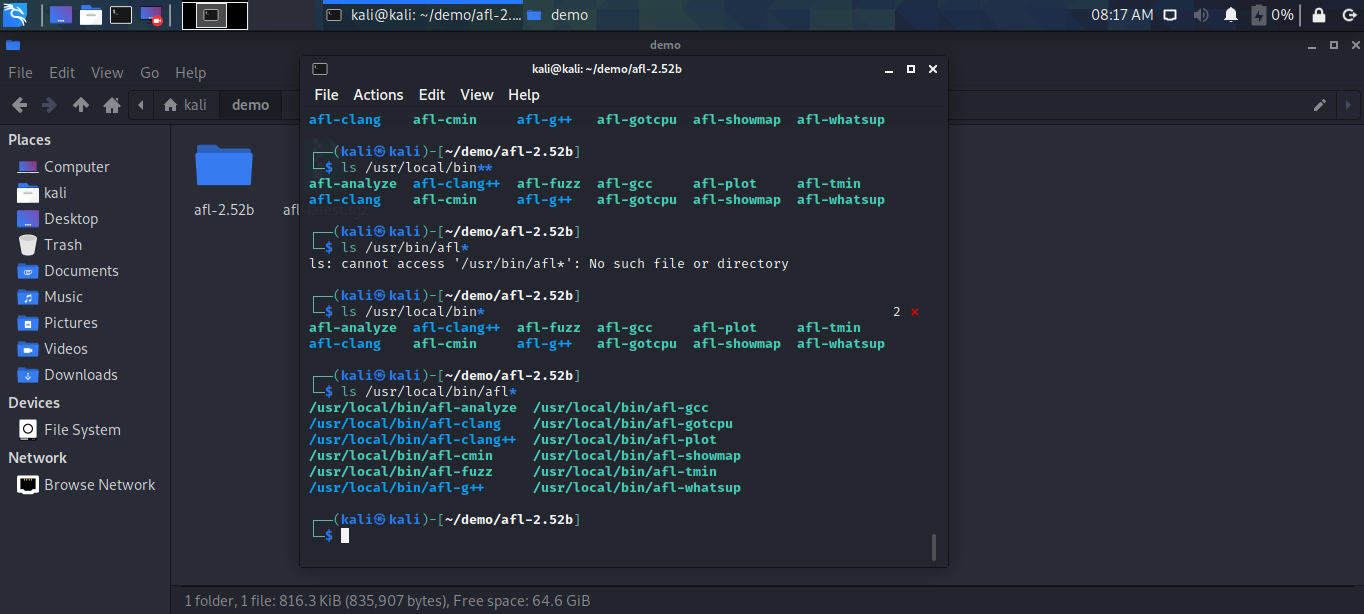
1. 然后可以在/usr/local/bin路径下找到afl：

ls /usr/local/bin

或者ls /usr/local/bin/afl\*

安装成功，结果如下：



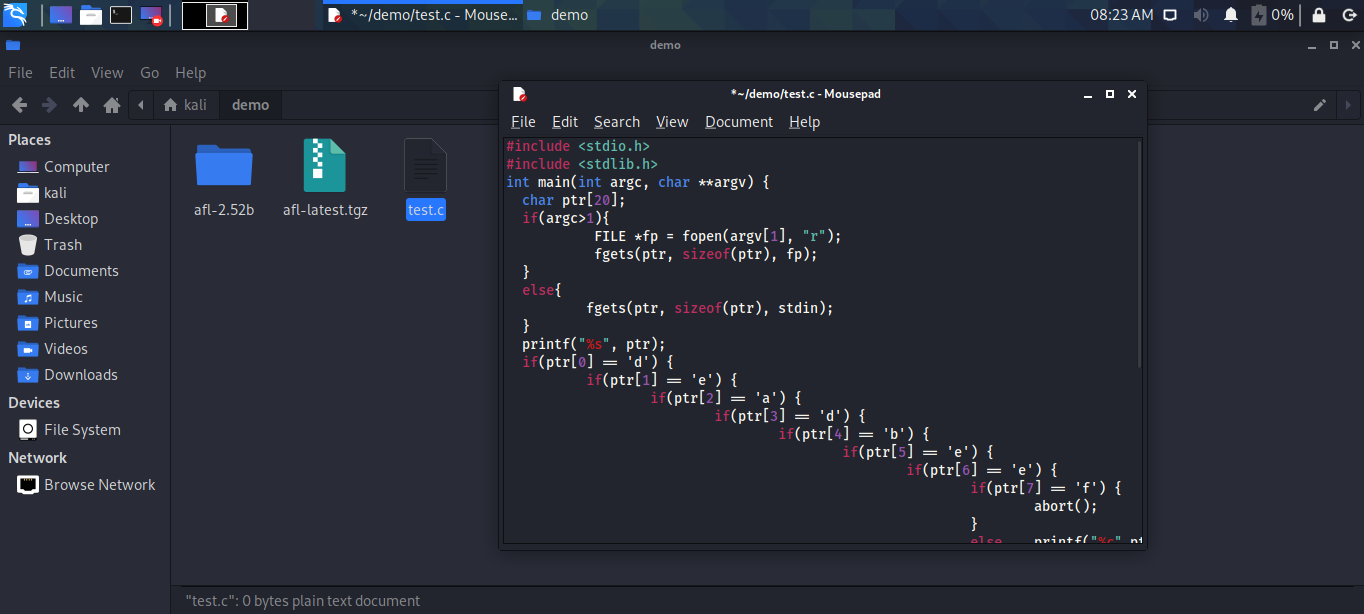


（注意到这里的安装路径是在local下，与教材上不同）

所以需要进行环境变量的重新配置，来让后续指令能够正常运行：

命令：export PATH=/usr/local/bin:$PATH

1. 进行AFL测试
2. 首先创建本次测试所用的程序，代码在教材上，由于篇幅有限这里不给出展示。在demo中新建test.c文件，打开将程序代码复制上去，如下图：（该代码编译后得到的程序如果被传入“deadbeef”则会终止）

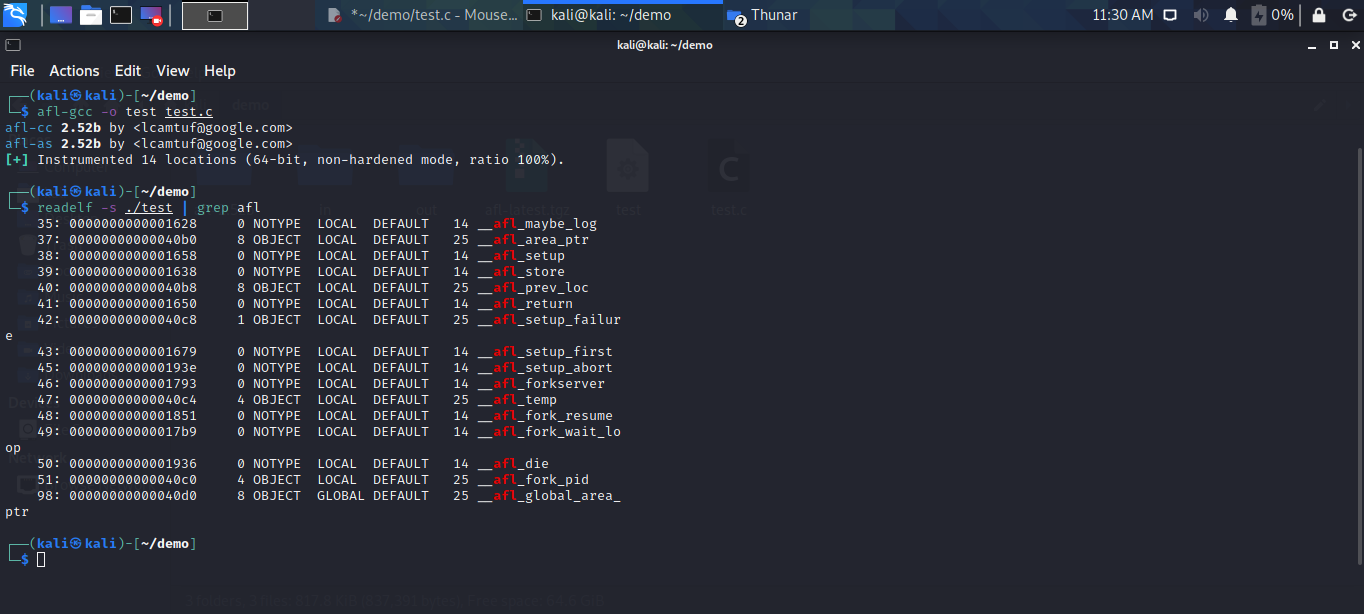


1. 然后分别使用afl编译器进行编译，然后进行插桩。命令分别如下：

编译命令：**afl-gcc -o test test.c**

插桩命令：**readelf -s ./test | grep afl**

结果如下图：



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，命令如下：

命令：**echo core > /proc/sys/kernel/core\_pattern**

1. 创建测试用例：（即为要变异的种子）

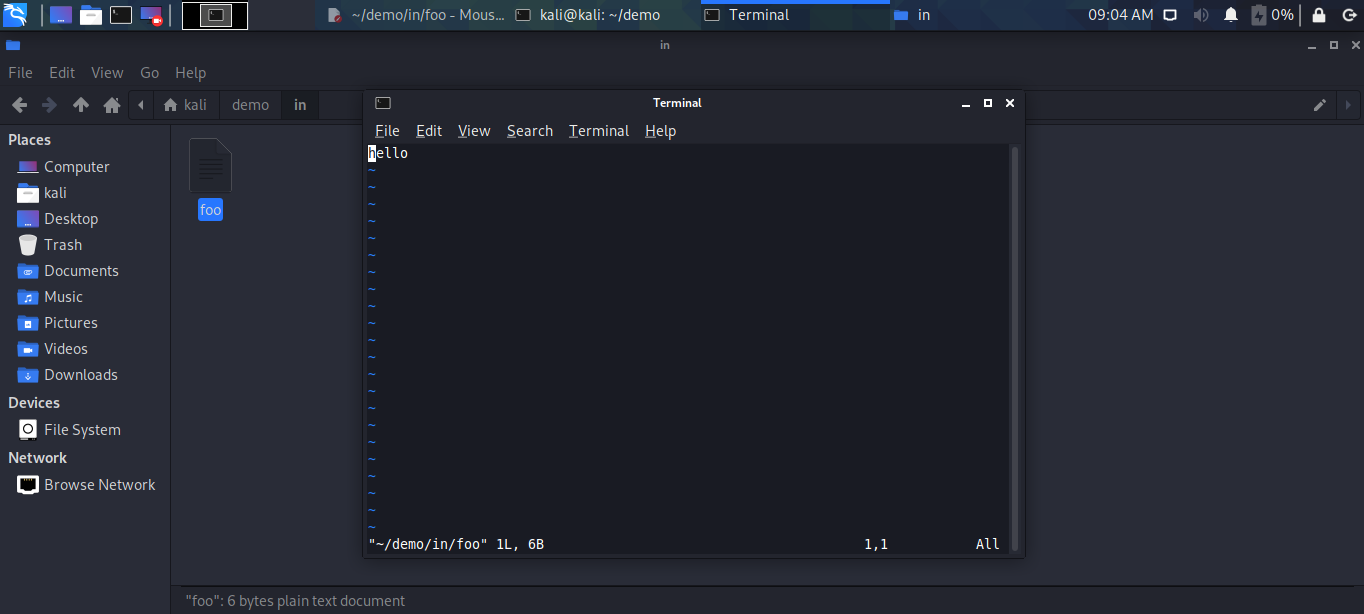
首先，创建两个文件夹in和out，分别存储模糊测试所需的输入和输出相关的内容。

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。结果如下：



1. 然后启动模糊测试，结果界面如下：



下面对界面进行介绍：

**·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**

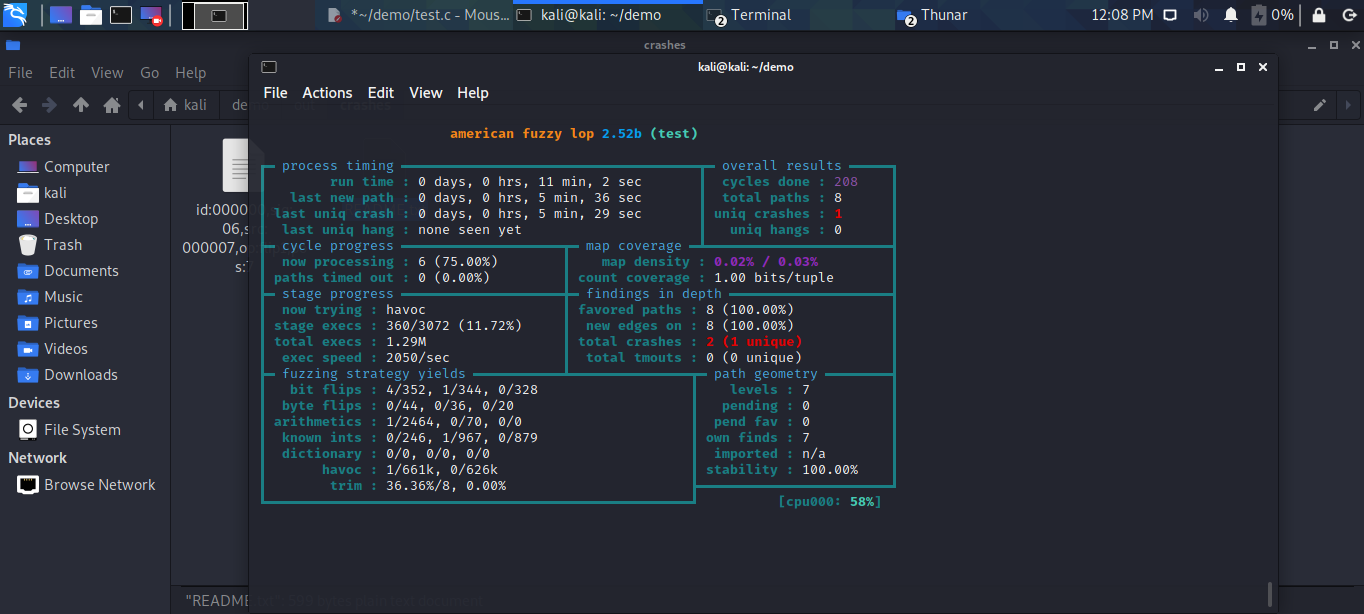
这里包括运行的总周期数、总路径数、崩溃次数、超时次数。其中，总周期数可以用来作为何时停止fuzzing的参考。随着不断地fuzzing，周期数会不断增大，其颜色也会由洋红色，逐步变为黄色、蓝色、绿色。一般来说，当其变为绿色时，代表可执行的内容已经很少了，继续fuzzing下去也不会有什么新的发现了。此时，我们便可以通过Ctrl-C，中止当前的fuzzing。

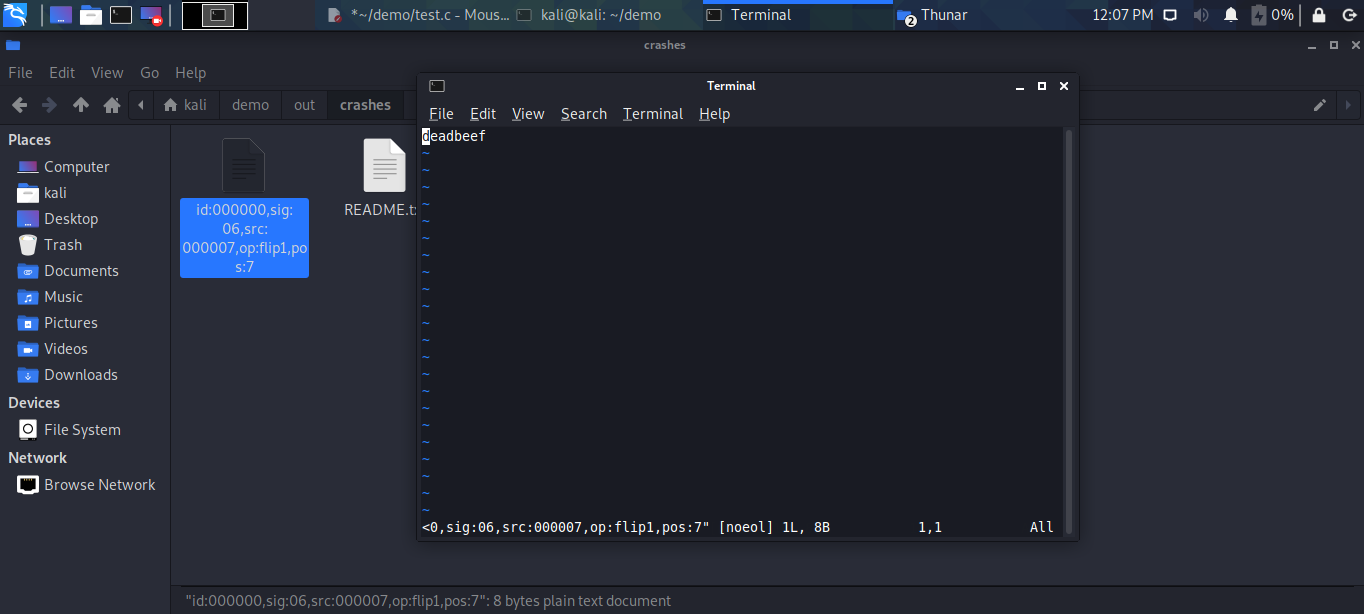
·**stage progress**

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。

1. 分析crash

观察fuzzing结果（当uniq crashes从0变1时，如下图）下下图是out文件夹下crashes子文件夹包含产生crash的样例：（成功确定了异常字符串deadbeef）





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.查阅资料后的个人理解：

在AFL中，"覆盖引导"指的是一种技术，它可以帮助AFL有效地测量程序代码的覆盖率。具体来说，AFL通过插入特定的插桩代码，记录程序在处理每个输入文件时经过的代码路径。通过这种方式，AFL可以测量程序处理输入文件时覆盖的代码量，并利用这些信息来引导后续的变异过程，以产生更多具有挑战性的测试用例。

文件变异是AFL的核心功能之一。在文件变异过程中，AFL会对原始输入文件进行修改，以生成具有不同特征的新输入文件。这些新文件包括了原始文件的变体，如添加、删除、替换和重排输入文件中的内容。这些变异过程旨在产生尽可能多的不同测试用例，以便在测试过程中尽可能地覆盖程序的代码路径，从而发现潜在的漏洞。

综合来说，AFL使用文件变异技术生成大量的测试用例，并使用覆盖引导来指导变异过程以尽可能覆盖程序的代码路径。这种方法被广泛应用于软件安全测试，可以帮助发现潜在的漏洞和安全问题。

另外来说，根据我自己的理解来说，简单讲，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。变异的方式有很多，包括不同位的翻转、数字的加减等等，所以种子变异来的结果是大量的。为了能较为简洁地完成目标，所以比较不同变异结果的代码覆盖率，将代码覆盖率高的结果存入队列并以此为基准进行下一轮的变异。

**心得体会：**

在这次实验中，我学习并复现了AFL在KALI下的安装和应用，并通过查阅资料来理解了覆盖引导和文件变异的概念及其在AFL中的作用。在实验过程中，我遇到了一些问题，但是通过不断地尝试和寻找解决方案，最终成功地安装了AFL并完成了实验。

通过这次实验，我学习到了自动化测试中一种非常常见的技术，即fuzz testing。AFL作为一种fuzz testing工具，可以通过覆盖引导和文件变异等技术来发现程序的漏洞，从而提高软件测试的效率和准确性。

在实验过程中，我也发现了自主学习和解决问题的重要性。当我遇到问题时，我不仅仅是停留在困惑的状态，而是积极寻找解决方案，并不断尝试直到找到正确的解决方案。这种自主学习和解决问题的能力对于我的未来学习和工作都是非常重要的。

总之，这次实验让我对自动化测试和fuzz testing有了更深入的了解，并提高了我的自主学习和解决问题的能力，让我在软件测试领域中更加自信和熟练。

补：在实验中遇到无法通过教材上的语句进行AFL安装的问题，最后通过询问同学和助教学长和百度成功的解决了问题，完成了实验要求，这也是一种自主学习能力的进步与提升。