5.1 AFL逻辑概述

插桩阶段。AFL在进行模糊测试之前，首先对待测程序进行插桩。插桩用于提取部分程序结构来指导模糊测试。

模糊测试阶段。AFL得到插桩后的程序和初始种子集之后，第一步就是进行初始化，这个过程中AFL会检测系统，对于MacOS、Linux、32位操作系统和64位操作系统等会进行不同的设置，之后配置环境变量，为共享内存分配空间等；第二步是将人工提供的初始种子集读取到队列中；第三步是开始模糊测试的真正过程，取出队列中的每个种子，对这个种子进行变异——测试——观察这三个步骤。在变异步骤中，AFL对种子进行多种变异，变异之后产生新的输入数据。在测试步骤中，将产生的新的输入数据喂给被测程序，等待被测程序执行完毕。在观察步骤中，AFL会等到被测程序执行完毕或是崩溃后发来的信息，根据信息，并结合共享内存中存储的覆盖信息，AFL可以分析得到程序是否崩溃、程序是否执行了新的路径。如果覆盖到了新的路径或是程序崩溃挂起。那么这个输入数据就会被认为是感兴趣的，会被加入到队列中，等待下一次对其继续进行测试。测试过程就这样一直进行下去，直到用户发出停止指令Ctrl+C，在收到用户发出的停止指令之后，AFL会将目前的工作保存，并关闭文件，释放内存，报告结果，退出程序。

AFL将目前的工作保存是因为AFL可以接着上次执行的结果继续进行测试，无需从开头重新测试。而停止的时候主要保存的内容就是存储覆盖信息的共享内存。他决定队列中元素的增长，它决定这一个新的输入数据是否是感兴趣的。下面，我们会对AFL的各关键部分逻辑进行详细解读。

5.2 插桩逻辑

AFL可以对开源和闭源C语言程序插桩。对于闭源程序，只能够得到程序的二进制文件，AFL是在QEMU模式下运行该程序，因此对于闭源程序来说，插桩效率、运行效率大大降低，AFL对其的处理速度没有对开源程序的处理速度快，在此，本文只对开源程序进行讨论。对于开源程序，可以得到程序的源代码。AFL并没有在源代码层对程序进行插桩，而是采用了新型的编译时插桩。

5.2.1 插桩时机

C语言程序源代码需要经过预编译，编译，汇编，链接，四个步骤之后才能够产生可执行的二进制文件。预编译阶段的工作是对源代码进行预编译处理，包括处理源代码中的宏、头文件的引入等。编译阶段的工作是将进行过预编译处理的代码编译得到汇编代码。汇编阶段的工作是将汇编代码处理得到目标文件。链接阶段的工作是将目标文件和库文件等链接起来得到可执行文件。

AFL采用的编译时插桩是在编译阶段之后，通过对汇编代码的静态分析，找到感兴趣的位置，将两段人工定义的汇编代码插入其中。编译时插桩可以更准确的分析源程序的代码结构，对源程序的破坏最小，而且较其他的插桩时机执行速度更快。如果想要使用AFL的编译时插桩除了需要程序的源代码之外，还要使用AFL提供的编译器进行编译。AFL对不同的C语言程序分别提供了afl-gcc、afl-g++、afl-clang、afl-clang++这几种编译器。

5.2.2 插桩内容

AFL的插桩内容是两段汇编代码。对应不同的Linux系统，32位和64位系统插入不同的汇编代码。汇编代码配合模糊测试工具共同完成模糊测试，其中主要是处理程序执行中的路径覆盖，将程序的状态反馈给测试工具。

汇编代码可读性较强。大体逻辑如下：当待测程序执行到插桩位置的时候，会调用\_\_afl\_maybe\_log函数，即call \_\_afl\_maybe\_log。进入该函数之后，首先检查共享内存区域是否已经分配了空间，之后进行初始化，初始化过程中需要保存各个寄存器的值，各标志位状态。接着，告知测试工具当前程序的状态并等待测试工具发来响应。之后，将当前执行的路径记录到共享内存中。最后，返回函数调用，回到正常的程序流中。这就是在某一个插桩位置，探针所做的工作。

每个插桩点插入的就是上面的逻辑，当整个程序执行完毕之后，即所有的插桩位置的逻辑共同作用于共享内存之后，共享内存的状态对应的就是某个输入数据影响下被测程序的执行路径的情况。

5.2.3 插桩粒度与位置

AFL是以基本块的粒度进行插桩的。AFL在对汇编代码分析的过程中，主要对函数的进入点、分支标签、条件跳转处关注，在这些位置上进行插桩。忽略注释、非分支标签、非条件跳转这些位置。

5.3 初始化逻辑

AFL初始化的阶段是在真正的模糊测试之前配置环境的一个阶段。在这个阶段里，AFL会进行很多操作以确保测试过程能够正常稳定的执行。包括获取当前时间，设置随机种子，检测路径是否可用，获取指令参数，根据参数设置输入目录、输出目录、测试文件、内存限制、时间限制等，设置错误信号处理器，检查内存消毒剂的配置，检测是否并行测试，设置AFL需要的环境变量，检查CPU的核心数，检查崩溃处理器是否配置，检查CPU的工作模式，为共享内存分配空间，加载输入文件中的种子集到队列，如果是接着上次暂停的测试继续运行那么就载入上次队列中的元素，执行队列中的元素以获得当前队列对应的共享内存的状态，显示当前状态，准备进入模糊测试阶段。

特别说明，AFL虽然会在退出的时候保存共享内存的快照，但在继续某个测试任务的时候，并不会读取上次共享内存的快照到共享内存。而是载入上次暂停时保存的队列，通过运行每个队列中的元素，来获得共享内存的状态。

5.4 模糊测试循环逻辑

在这个阶段，每次迭代，都会从队列中取出队首元素，这个操作不会将队首元素删除。然后对这个元素进行多种变异操作，每种变异操作之后，都会产生一个新的输入数据。然后，这个新的输入数据就会被喂给被测程序，被测程序运行之后，告知测试工具程序是否产生崩溃。如果产生了崩溃，那么测试工具就会保留这个输入数据，并记下对应的崩溃信息。如果没有崩溃产生，那么就会检查共享内存是否产生了新的覆盖，是否是感兴趣的输入数据，如果是感兴趣的，那么就会将这个新的输入数据加入到队列中，否则就丢弃这个输入数据。然后准备进行下一次变异。当队列中的一个元素进行了所有的变异操作之后，就会继续对队列中的下一个元素继续进行如上操作。测试过程就这样一直进行下去，直到测试工具检测到用户发出的停止指令Ctrl+C。

5.4.1 变异

这个步骤是对某个种子进行多种变异产生新的输入数据。变异操作有很多种：位翻转算子包括1比特位翻转、2比特位翻转、4比特位翻转、1字节翻转、2字节翻转、4字节翻转；逻辑增减算子包括8比特算数增减、16比特算数增减、32比特算数增减，同时考虑了计数方式的大端模式和小端模式；设置感兴趣的值操作算子包括设置8位整数，设置16位整数，设置32位整数；还有随机改变、分割操作，以及以上变异算子组合，还可以引入人工定义的变异方式。

5.4.2 输入数据的保留

每当一个新的输入数据被执行之后，测试工具都会分析共享内存的信息，来得到该次执行所覆盖到的路径情况。

5.5 bitmap数组

5.6 关键逻辑

5.7 技术细节