CollAFL-Path Sensitive Fuzzing（未开源）阅读报告

1、现有模糊器存在的问题：

现有的一些模糊测试器会利用代码覆盖信息来作为模糊测试的引导，例如：

Libfuzzer和honggfuzz利用clang中的SanitizerCoverage组件来跟踪块覆盖信息，VUzzer利用PIN工具来跟踪块覆盖信息，AFL利用动态/静态分析来跟踪边缘覆盖信息。

以上这些模糊器都是使用较为简单的覆盖信息，这种覆盖的不准确和不完整给测试带来了严重的局限性。其主要存在两个弊端：

1. 导致路径冲突，从而影响fuzzing挖掘出导致新崩溃的潜在路径。
2. 影响fuzzing的最优决策（种子优先级排序等等）。

* 路径冲突问题：

例如AFL中，AFL用64KB大小的bitmap来保存边缘覆盖信息，通过计算哈希值来表示一条边，但是不同边的哈希值可能是相同的，假如hash（A）=hash（B），如果这时候AFL已经测试过A边了，那么在产生新的边B的时候，AFL就不会认为B是一条值得测试的边，会误以为B已经被覆盖过了。这就是A和B发生路径碰撞，从而导致某些输入到达新的路径，但AFL却没有将该输入作为有效的种子。

2、问题的解决

1. 如何解决路径冲突问题

由于AFL中对边计算哈希值是采用固定的公式进行计算的，因此很容易产生路径冲突的情况。基本块A->B，给基本块赋关键值，,：



提高覆盖信息的精确性

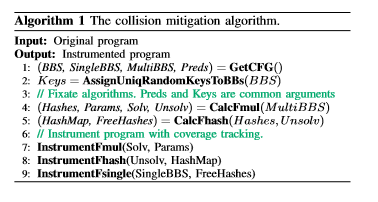
* 1. 覆盖信息粒度：边覆盖
  2. 通过对不同的边应用不同的哈希公式来进行细化，以消除哈希冲突，同时保持哈希计算速度和覆盖跟踪。

具体算法：



式中x, y, z是待确定的参数，对于不同的边，这些参数可能不同。这需要为每个基本块找到参数的解决方案，确保通过计算的所有边哈希都是不同的。但是，我们不能保证能够为给定的应用程序找到参数的解决方案，因为应用程序中有太多的基本块，不能遍历所有可能的参数。即使可以这样做，也不能保证存在一个解决方案，因为基本块的键是随机分配的。

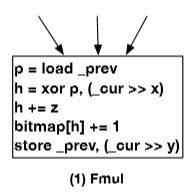
进一步改进：



输入程序，通过CFG获取每个基本块的相关信息，eg:有多个先继的基本块，只有单个先继的基本块，preds（存储先继基本块的相关信息），通过CalcFmul算法分析其是否可解，将基本块分为可解和不可解两种类型。

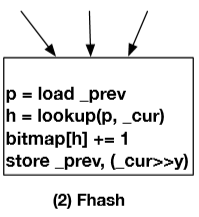
对于不同类型的基本块，采取不同的处理方法：

* + 1. 具有多个先继且不可解的基本块



通过CalcFhash算法给这类基本块赋一个此前没有用过的hash值，并将对应关系存储在hashmap中。

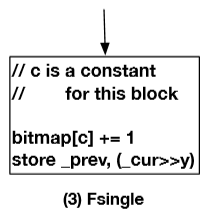
* + 1. 具有多个先继且可解的基本块



通过CalcFmul求解出x,y,z后通过以下公式求出其hash值：



* + 1. 具有单个先继的基本块

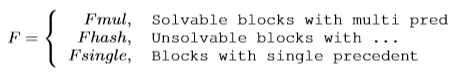


对于这类基本块的hash值计算在以上过程完成后，用一个独有的hash赋给每个基本块。

* + 1. 特殊情况：间接调用问题

如果没有被任何函数直接调用，那么这个基本块很可能会被错误地归类为没有先继块，因此在处理各种基本块之前，先将这类基本块归类为多个先继的基本块，按照多个先继的基本块继续处理。

综上，在确保bitmap size大于边数的情况下，根据不同的类型，使用Fmul、Fsingle和Fhash这三个hash计算公式：



1. 如何解决fuzzing的最优决策（种子选择的问题）

本文提出了三个不同的决策方法：

* 1. memory-access guided

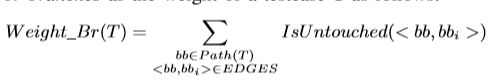
内存操作次数越多，内存发生崩溃的可能就越大。拥有更多内存访问操作的种子将优先于模糊，该策略使用内存访问操作的数量作为测试用例的权重。



其中，函数NumMemInstr返回参数基本块中的内存访问操作数，可以静态计算。因此，与前两个策略不同，以这种方式计算的权重是确定性的。

* 1. untouched-branch guided

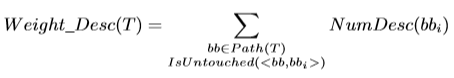
拥有更多未触及的邻近分支的种子将优先于模糊，该策略使用未接触的临近分支数作为测试用例t的权重。未触及的邻居分支的数量作为该种子的权重。



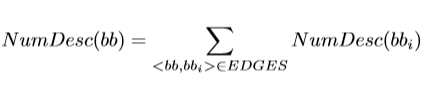
此公式只在且仅当边缘未被任何先前的测试用例覆盖，否则为0。通过此公式，可用权重来考虑种子的选择，即权重更高的种子将被优先考虑模糊化，值得注意的是，随着测试的进行，先前运行的测试用例集将发生变化，因此所接触的函数的返回值也将发生变化。因此，测试用例的权重是动态的。

* 1. untouched-descendant guided

拥有更多未受影响的邻近后代的种子将优先考虑模糊，该策略使用未接触的邻近后代的数量作为测试用例的权重。



其中函数IsUntouched与CollAFL-br策略中使用的相同，函数NumDesc返回从参数基本块开始的子代路径数。其形式定义如下：



这里的权重不是确定的，因为函数IsUntouched是动态的。但是对于每个基本块，子路径的数量是确定的。

综上，memory-access guided考虑的是内存访问，CollAFL统计这个种子所走的路径，基本块访问的数量，那些访问数量多的，优先级就相应高一点。

untouched-branch guided考虑的是，每个种子会走一条路径，一条路径实际有不同分支的，有些分支是被其他种子测过，有的分支没有。然后CollAFL统计这个种子多样分支被测过，有多少分支没有被测过，于是就有可能出现两个种子。第一个种子有一个分支没测过，第二个种子有N个分支没测过。CollAFL选择的就是后者，因为在第二个进行变异的时候有非常大的概率，种子存在没有被触发、测试过的分支。

untouched-descendant guided是在untouched-branch guided的基础上改进，untouched-branch guided中的分支记为 1，此处分支后面会跟着一些子路径，所以也要考虑子路径的数目，因此计数不再是 1，而是把后面的分支根据路径数量加进来。

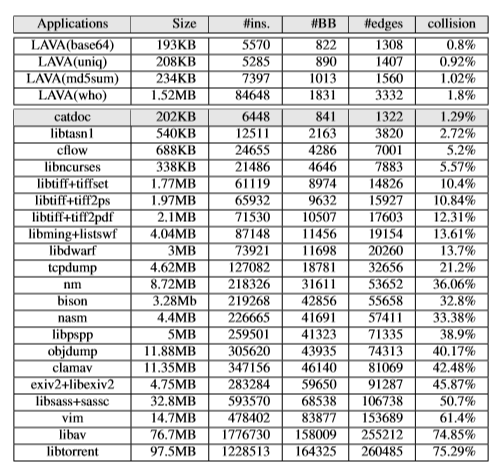
3、实验评估

本文在各类应用中做了一个实验，展示了哈希碰撞问题和我们的动机的影响，代码覆盖率和崩溃增长的改善，CollAFL在现实应用中的漏洞发现的有效性，模糊测试的随机性，以及模糊器之间的比较。

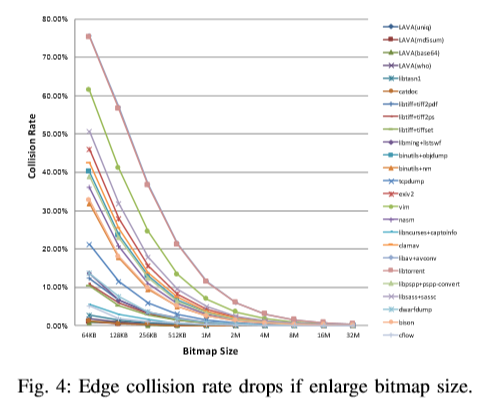
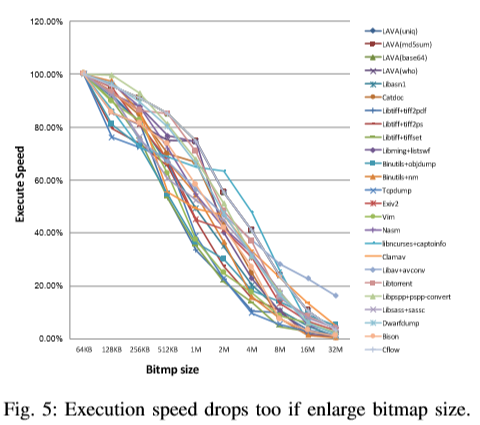
选择了24个流行的开源Linux应用程序例如：nm, tcpdump, clamav, libtiff, libexiv2, libmpg123, libav, vim, catdoc, libgxps等等以及LAVA-M。

Step1：CollAFL对于解决hash collision有效性的验证

目标应用程序的统计信息，包括文件大小、指令数量、基本块和边缘。在最后一列中，它显示了冲突比率，演示了边哈希冲突的普遍性。



虽然扩大bitmap能够有效减少路径冲突，但在实际中这是不可行的，运算速度下降，性价比低。

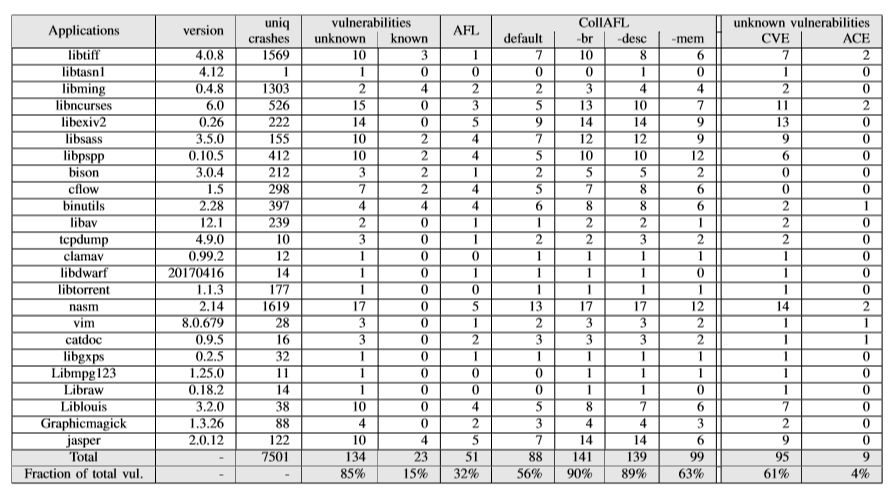
基本上Fsingle比Fmul的块更多，并且Fhash的块数量接近于0。而Fsingle的成本远低于Fmul。而且，平均而言，CollAFL仪器比AFL少2.93%的应用指令。



所以，CollAFL可以解决所有已知边缘的冲突，并且稍微提高执行速度，都优于AFL。

Step2：代码覆盖率 & Step3：漏洞检测数量

这两部分对CollAFL和AFL做了一些横向对比，实验证明，CollAFL在24个实际应用程序中发现了157个新的安全漏洞，其中95个被CVE确认。



1. 总结

代码覆盖率的准确性对于覆盖率引导的模糊器是至关重要的。CollAFL为路径冲突问题提出了一个较好的解决方案，没有单纯地利用扩大bitmap来避免路径冲突，在覆盖精度和性能之间找到一个平衡点。