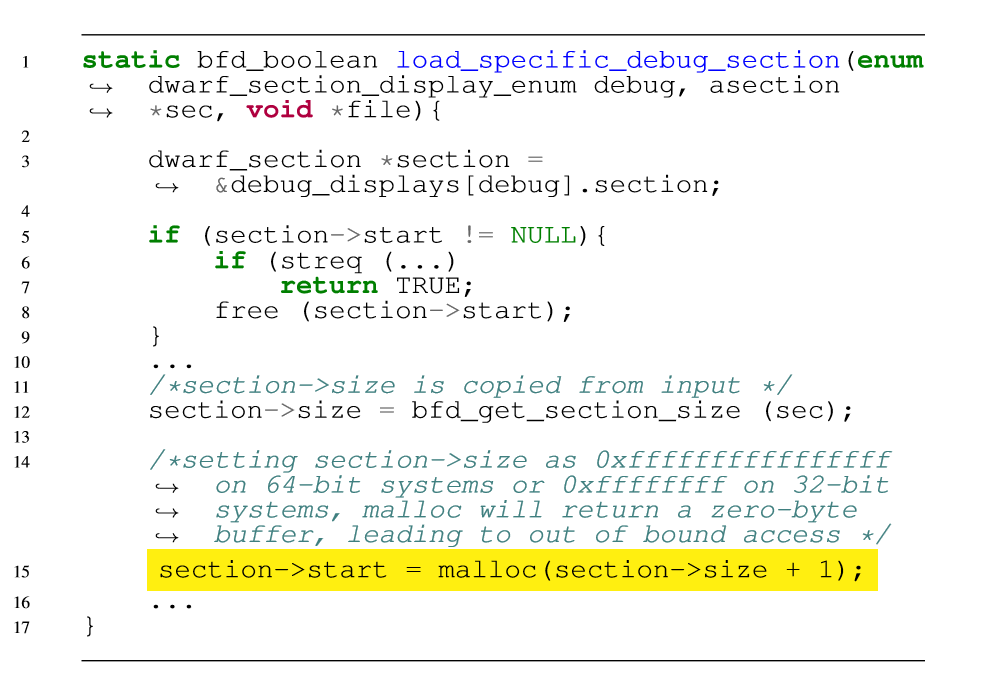
SAVIOR：Towords Bug-Driven Hybrid Testing阅读报告

1、研究背景：

当前已经有许多模糊测试器采用混合检测，但现有的混合检测存在以下两个问题：

①对种子没有进行挑选（即没有种子的优先级排序）

②虽然有些漏洞会被产生的路径覆盖，但是未必会被触发，如果变异的空间较大，那么触发漏洞的概率就会很小。例如：



（如图，第12行将sec值传给section->size,在第15行这里就有可能出现漏洞，如果sec为或者，那么在相应的32位或64位机器上就会触发漏洞。但是由于这里的数据空间太大了，采取随机变异的方式很难能够触发到这个漏洞）

SAVIOR针对以上两个问题进行了探索，提出了漏洞导向的种子排序机制和在符号化执行组件执行的漏洞验证机制。

2、SAVOR大体工作流程：

首先利用LLVM的UBsan工具对二进制潜在漏洞点进行标记，构建CFG进行分析，在运行中计算未发现边的漏洞数量进而打分实现对AFL产生的种子的排序，然后把标记了的UBSan标签的LLVM中间代码给符号化执行组件，在符号化执行过程中进行约束求解来验证漏洞是否存在。

3、具体实现

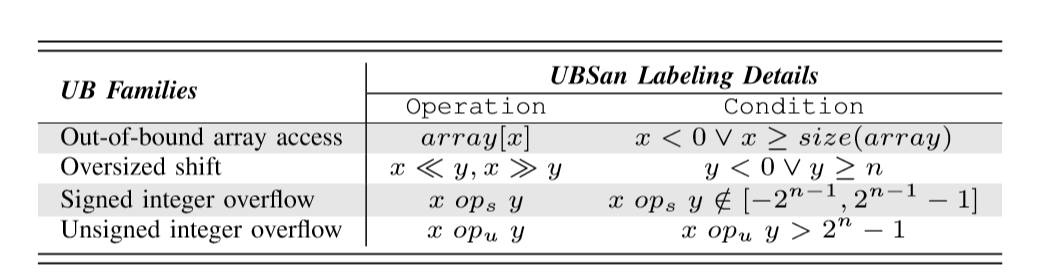
（1）种子排序

R1：知道一个种子在符号化执行后可能执行的边有哪些

R2：统计这些边能够导向的漏洞的数量

利用LLVM生成控制流图CFG，记录基本块ID、后继基本块ID、对应的边信息。在运行过程中，通过基本块ID获取执行顺序，从而找到那些没有经过但是在符号执行之后可以经过的边。

利用UBSan对内存操作进行标记，并在LLVM中间代码加入验证。



从CFG中得到边能指向的UBSan的数量，实现种子优先级。

种子的评分标准：

其中，表示某个种子没有发现的边

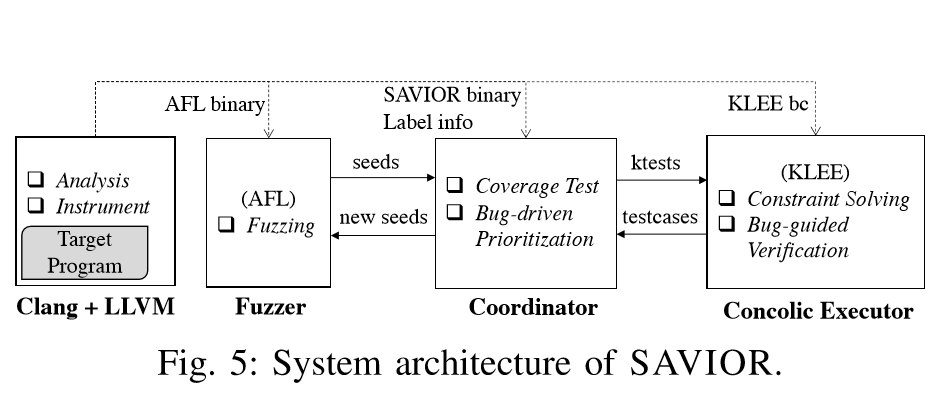
表示每条边在符号执行中的求解难度

表示每条边可以到达的UBSan的数量

1. 漏洞验证

LLVM可以在中间代码中插入求解约束，在符号化执行阶段就会进行约束求解，如果产生能触发漏洞的新种子，则说明该标签漏洞可以被触发。

1. 实现流程



Clang++LLVM：首先进行UBSan贴标签、构建CFG、记录基本块、边信息等工作，接着插桩。

Fuzzer：模糊测试是由AFL直接执行的。

Coordinator：这部分有两个方向的工作，其一是从AFL种子队列中挑选种子给Concolic做符号化执行，又要把符号化执行中有用的种子挑选出来加入AFL的种子队列。

挑选AFL提供的种子时先把AFL种子队列中的新种子在本地运行一遍，打印执行过的基本块ID得到基本块的执行序列，提取没有发现的边集合，对种子进行打分。

符号化执行完把新的种子跑一遍，只留下带来新的覆盖率或者触发新的标签的种子，把这些种子加入AFL的队列。同时，对于符号化执行后没有被求解的边，对应的求解难度加一。

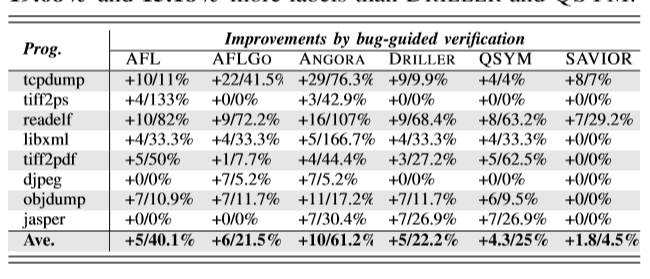
Conclic Executor：这部分对KLEE进行修改使其能够进行concolic testing。然后再本地建立一个bitmap来解决路径爆炸问题。同时为了减少符号化执行初始化的时间，采用ForkServer模式加快符号化执行速度。

1. 评估

这里将SAVIOR和已有的一些混合检测工具DRILLER\QSYM等在LAVA-M数据集上进行了比较，对每一个程序都进行了具体分析。

总体来看，SAVIOR的覆盖率虽然不是最高的，但是触发的漏洞最多，虽然UBSan减慢了运行过程、同时符号化执行采用KLEE，速度也不高。但是由于漏洞验证机制，能发现更多的漏洞。

SAVIOR又把所有fuzzer再24h运行后的种子放入SAVIOR的符号化执行部件，由于UBSan插入的LLVM IR，符号化执行部件能在种子输入中对漏洞标签处约束求解，得到能够触发漏洞的新种子。



1. 总结

其实种子优先级排序已经不是新鲜的内容了，但漏洞验证机制还是非常有效的。UBSan标签带来的漏洞验证机制更有效地挖掘漏洞，这在本周阅读的另一篇文章ParmeSan中也有所体现。