Reading Notes

cxl

2020/05/30

目录

1 SAVIOR: Towards Bug-Driven Hybrid Testing @S&P'20

1.1 Background/Problems

混合测试结合了模糊测试和混合执行。模糊测试可以测试容易到达的区域，混合执行可以探索复杂的区域。混合测试结合这两部分可以很好的探索程序的状态空间。但是现在很多的 fuzzer 都是使用代码覆盖率作为反馈，以代码覆盖率为反馈其实就是将程序空间的所有部分同等化对待，但是实际上有的代码是没有 bug 的，这种同等化对待会使得我们浪费很多时间于没有 bug 的代码部分，所以我们需要将更多的注意力放在有 bug 的代码部分。这样可以使得漏洞检测的速度大大提升。

1.2 Methods/Techniques

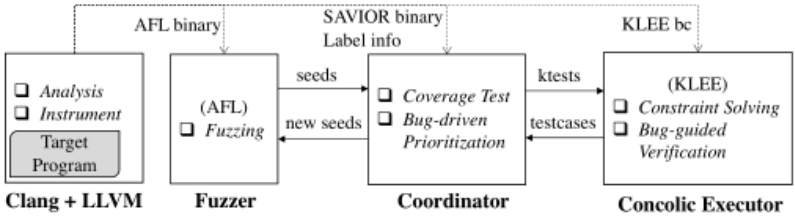

图 1: System architecture of SAVIOR.

本文作者提出了一种就 bug 驱动原则的混合模糊测试框架 SAVIOR，SAVIOR 的系统结构图见图1.SAVIOR 主要是通过 Bug-driven prioritization 和 Bug-guided verification 这两个部分来实现的。

Bug-driven prioritization:SAVIOR 初始时会对测试程序进行静态分析获得每个分支可到达每个基本块的集合，并且会利用 UBSan 工具为测试程序生成潜在 bug 标记，并对这些标记进行合理的筛减。在每个种子运行时，会根据种子的未探索的分支，以及这些分支所包含的潜在 bug 标记和这些边支约束求解的难度来计算这些种子的分数，从而给定种子的优先级。种子的优先级越高，即可以说明这个种子含有潜在 bug 越多，就会优先被混合执行来执行，这样可以使得时间更多的花费在有效的混合执行上，从而提高 bug 检测的

效率。

Bug-guided verification: SAVIOR 除了使用 bug 驱动的优先级来加速漏洞检测之外，还会沿着混合执行遍历的路径来验证标记的漏洞。特别在有些情况，程序的一个路径存在 bug，但是即使使用混合执行执行执行这条路径都无法触发这个 bug。因为这个 bug 可能存在潜在的约束，混合执行无法得知，例如本文中内存泄露的例子。SAVIOR 在为每个潜在 bug 添加标记之外还会添加相应的约束，如果 SAVIOR 可以解算约束，那么构造一个输入作为证明这个 bug，否则这个 bug 是不存在的。

1.3 Results/Evaluation

通过实验的评估，我们可以发现基于 bug 驱动的混合测试是优于传统的混合测试。SAVIOR 可以触发 481 次违规，其中 243 为真正的 bug。平均而言，SAVIOR 检测漏洞的速度快于 DRILLER 的 43.4% 和快于 QSYM 的 44.3%，结果分别的多发现了 88 个和 76 个 bug。

1.4 Limitations/Future work

1. Over-approximation in Vulnerability Labeling: SAVIOR 使用合理的算法来标记漏洞，但是可能由于 Over-approximation 导致许多假阳性标记被引入。解决这个问题可以采用更精确的静态分析来进行更细粒度的标记修剪。

2. Prediction in Vulnerability Detection：混合执行到达潜在 bug 的位置时，SAVIOR 会提取潜在 bug 标记的保护谓词，如果该保护谓词和当前路径约束条件相矛盾时，SAVIOR 会停止探索这个潜在的 bug，这样可以节约一定的时间。但是我们可以使用以前执行运行的信息来预测执行路径能否触发漏洞。本文作者为了实现这一点，使用了一种方法，该方法将标记点的路径约束向后总结到探索路径的前一个点，利用的核心技术是 the weakest precondition。

2 SLF: Fuzzing without V alid Seed Inputs @ICSE'19

2.1 Background/Problems

模糊测试是通过初始种子进行突变生成大量输入来测试程序，如果没有有效的种子，fuzzer 的性能会显著的下降。虽然符号执行可以从头开始生成种子的输入，但是它们有很多的限制，使得它们在现实中的复杂软件效率很低。所以生成有效的种子对于 fuzzing 是非常有必要的，本文提出一种不依赖源码和复杂程序分析的有效种子生成的 fuzzer。

2.2 Methods/Techniques

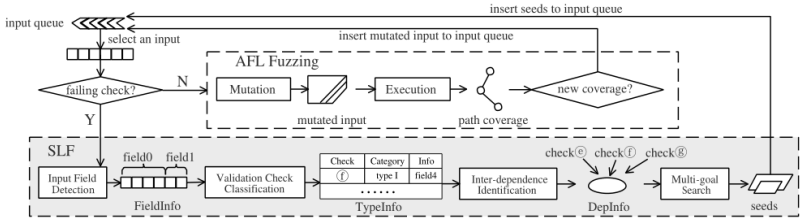

Fig. 3: Overview of SLF.

图 2: Overview of SLF.

SLF 的实现来自于两个重要的观察结果。第一，输入验证检查可以分为许多类别，每个类别对应特定的输入关系，并需要独特的变异策略。第二，可以通过预定义的格式中改变输入字节并观察位次的变化来有效检测类别。

SLF 的整体过程是首先随机生成 4 字节输入，通过执行输入来分析遇到的检查和错误信息来识别检查失败执行，如果没有找到验证失败，则当前种子良好，SLF 切换到 AFL，否则通过 SLF 生成种子方法来生成良好的种子。SLF 的整体过程图见图2。

SLF 生成良好种子主要由 4 个部分实现，第一个是 Input Field Detection，它将输入字节分组到字段，这允许在字段级别而不是字节级别上执行变异。第二个是 Input Validation Check Classification，它检测当前执行中的验证检查的类型。第三个是 inter- dependence identification，它标识前面的所有检查与当前执行失败的检查具有相

互依赖关系。第四个是 mutation algorithm ，它一个基于多目标梯度的搜索算法执行相应的突变，以满足所有相互依赖的检查。

Input Field Detection: 首先执行目标程序并收集检查信息，然后逐个字节的翻转并获得相应的检测信息，然后将相邻字节且检查信息相同的视为一组。

Input Validation Check Classification: 由于每种检查类型的不同特性，本文定义了三种检查策略来检查他们。具体是迭代各个输入字段，并尝试这三种检查策略，通过判断检查的谓词变量的更改是否预定类型的特征相匹配，如果匹配便将其设置为其类型。

inter-dependence identification: 根据每个检查的类型信息，我们关联那些谓词受到相同字段影响的检查。

mutation algorithm: 它会利用失败检查和失败检查的相依赖的检查的梯度来进行梯度搜索突变算法，并且该突变会使用两种方式针对两种不同的情况。

2.3 Results/Evaluation

我们对 SLF 评估了现实中 20 个真实项目，这些项目分别来自一些其他 fuzzer 的基准和谷歌的 fuzzer 测试套件，实验证明 SLF 比现有 fuzzer（AFL 和 AFLFast），符号执行引擎（KLEE 和 S2E）和混合模糊测试（Driller）效率都高。并且它可以生成比其他技术多 7.3 倍的有效种子输入，平均覆盖比其他技术多 3.5 倍的路径。

2.4 Limitations/Future work

1. SLF 不能处理与输入字段有复杂控制依赖关系的检查，这些复杂依赖关系的检查只能用常规的模糊测试方法来解决（例如突变）。

2. SLF 在字节级标识输入字段，因此无法推断包含位级字段检查。