InsFuzz: Fuzzing Binaries With Location Sensitivity

笔记本: c++
创建时间: 2019/11/5 16:44 **更新时间:** 2019/11/5 19:31
作者: Mrs.Lee
URL: https://translate.google.cn/#view=home&op=translate&sl=auto&tl=zh-CN&tex...

书名	InsFuzz: Fuzzing Binaries With Location Sensitivity	作者	HANFANG ZHANG
出版社		阅读日期	2019.11.5

ABSTRACT(主要关注相关工作)

在本文中，我们提出了一种位置敏感的模糊方法，称为InsFuzz，它利用了轻量级程序分析技术。我们使用静态分析和二进制工具来推断可能影响比较指令的字节（我们称为键字节），然后在执行期间推断键字节与比较指令之间的关系。这使模糊测试器可以知道哪些字节值得更改以及应如何更改这些字节。另外，我们在执行期间收集比较进度信息（即，我们记录一条指令的两个操作数之间的匹配字节数），并保留具有较高比较进度的变异输入。因此，模糊器可以有效地破坏比较指令。我们首先将LAVA-M数据集上的InsFuzz与其他模糊器（包括AFL-Dyninst）进行了评估，然后在五个实际程序中将InsFuzz与AFL-Dyninst进行了比较。结果表明，与我们在LAVA-M数据集上比较的模糊器相比，InsFuzz发现了更多的错误。

INTRODUCTION

模糊测试是一种用于测试软件安全性的有效技术[33]，[46]。 它为目标程序提供突变的输入，并监视程序的状态以发现错误。 与其他漏洞挖掘技术相比，模糊测试更加简单，可扩展性强，并且不会占用太多计算资源。

根据模糊器使用的结构知识的程度，模糊测试可分为三大类：黑盒模糊测试[49]，白盒模糊测试[24]，[35]，[37]和灰盒模糊测试[13]。 黑盒模糊测试不需要任何有关程序的内部知识，而只需使用随机输入即可执行程序。 白盒模糊测试需要使模糊测试者能够进行重量级程序分析的知识，或者可以访问程序的源代码。 Greybox模糊测试使用轻量级程序分析（例如二进制检测）来收集程序结构。

传统的模糊测试具有一些局限性，例如模糊器浪费大量时间来改变输入中不重要的字节。有时会在关键字节上发生突变，但由于输入是随机构造的，因此模糊检查人员仍然很难绕过复杂的检查。随机输入在执行的早期阶段可能会被拒绝，并且不能渗透深层的程序代码。因此，随机输入很难触发程序中深处的错误。已经提出了一些方法，例如Driller [45]，VUzzer [40]和Angora [17]，通过利用

符号执行[18]，[19]，[30]，[34]，[43]，[45]，[36]或污点分析[12]，[23]，[40]，[17]。但是，由于这些方法依赖于符号执行或污点分析，这些方法被称为重量级程序分析技术[31]，因此它们的可伸缩性较差或需要更多的计算资源。提出了其他一些轻量级的模糊器，例如AFLFast [38]，Stelix [31]和**FairFuzz [28]**，以提高代码覆盖率

首先，InsFuzz收集比较指令（例如cmp指令）的信息，这些信息可防止输入穿透目标程序的深层代码。 然后，它对二进制程序进行检测，以获取操作数的实际值，并在执行过程中收集比较进度信息。 其次，它收集信息以通过连续改变字节来推断输入文件中的字节与比较指令之间的依赖性。 第三，它根据第二步收集的信息对特定位置的字节进行突变，并根据比较进度指导突变。

总而言之，本文做出了以下贡献：

1）我们提出了一种方法来推断输入文件中的字节与比较指令之间的依赖关系，这些比较指令在执行的早期总是拒绝输入。

2）我们使用了一种轻量级的仪器方法来收集模糊过程中的比较进度信息，以指导突变。

3）我们根据本文提出的方法InsFuzz实现了一个原型，并在LAVA-M数据集和5个现实程序中对其进行了评估。 结果表明，InsFuzz比其他工具更有效。

4）我们在现实程序中发现了一些新错误，其中四个已经分配了CVE编号

MOTIVATION EXAMPLE

与我们的工作最接近的**Steelix通过启发式局部穷举搜索解决了魔术检查**问题。 Steelix通过C1检查很容易。 但是，C3检查很难让Steelix绕过，因为比较字节来自输入的不同部分。 在这种情况下，Steelix应用的局部穷举突变无效。尽管Driller，VUzzer，T-Fuzz和Angora可以.

为了找到值得突变的字节，我们考虑添加一个突变阶段，例如AFL的确定性突变。 在确定性更改中，AFL会连续更改输入文件的字节。 我们在模糊开始时记录比较操作数的原始特定值，然后逐字节对输入进行突变，并在每次运行后将当前操作数的值与原始值进行比较，以找到影响比较指令的字节。 我们将可能影响重要比较指令的字节称为关键字节。 如动机示例所示，输入的前四个字节对校验1产生影响，因为当使这些字节发生突变时，比较指令的操作数的值将被更改。

通过这样做，我们可以推断出应该改变的字节，并获得推断信息，即改变哪些字节将影响目标比较指令。 然后，我们只对那些影响至少一个比较指令的字节进行突变，而不是对输入文件的所有字节进行突变。 此外，我们可以引导变异，以帮助输入基于推理信息绕过目标比较检查。

比较指令中的两个操作数的值应完全匹配，以绕过比较检查。 有时，变异的输入文件与目标比较指令的某些但并非全部字节匹配。 在这种情况下，AFL会丢弃此变异的输入，因为代码覆盖率没有改变，这提高了AFL生成完全匹配的输入的难度。 我们考虑收集和记录比较进度信息。 如果新突变的输入改善了比较进度，它将保留用于进一步的突变。

静态分析的目的是获得有关比较指令的详细信息。 通常，程序中有太多比较指令。 有必要滤除我们不感兴趣或AFL容易绕过的说明。 我们用来过滤无趣的比较指令的规则如下。

1）如果要比较的值的长度为1个字节，则该指令将被滤除。 AFL可以轻松绕过Onebyte长度比较的说明。

2）一些比较指令与立即值0xFFFFFFFFh进行比较，或者可以在随机突变过程中轻松地被AFL忽略。 这些说明也将被过滤掉

3）当模糊目标程序的特定功能时，我们不需要在其他未调用的功能中提取比较指令。

密钥字节推断的目的是找到影响比较指令的字节，并推断密钥字节与比较指令之间的关系。 AFL具有许多突变阶段：bitflip（连续位置的翻转位/字节），arith（算术增量/增量），interest （用有趣的值替换），Extras（用字典中的值替换）， havoc（随机改变输入，包括删除字节，插入字节等），然后进行splice（将当前输入文件与另一个输入拼接在一起）。它们分为确定性突变和随机突变[1]，[11]。

在确定性突变的位翻转阶段，AFL将输入从头到尾按位或按字节翻转。例如，在bitflip 8/8阶段中，第一个字节将被翻转，然后变异的输入将作为目标程序的种子被执行。 AFL监视程序的状态，以查看突变的输入是否提高了代码覆盖率，或者目标程序是否发生崩溃。如果变异的输入改善了代码覆盖率或导致崩溃，则将其添加到队列中以进行进一步的模糊处理，否则它将被丢弃。然后，输入的第一个字节将被恢复，第二个字节将被翻转。 AFL重复此过程，直到所有字节都被翻转为止。受确定性突变功能的启发，我们添加了

一个连续改变字节的阶段。我们称这个阶段为扰动。为了最大程度地减少程序在扰动期间的影响，并避免在输入文件中引入其他字符，这可能会导致程序执行错误路径（即，程序将提前终止），我们通过替换字节来改变字节当前字节和下一个字节。

RELATED WORK

基于反馈的方法通常用于选择种子输入，还用于基于模糊过程的状态来提高代码覆盖率。 AFLFast [38]使用马尔可夫链对模糊测试中的挑战和机遇进行建模，并基于模糊反馈选择种子输入，该输入将行使低频路径。它分配具有高能量的低频路径输入，以允许它们生成更多新的输入文件，进而击中更多稀有路径。Steelix [31]使用仪器来获取比较进度信息，并在基于反馈比较进度的基础上改进比较进度后，采用启发式方法解决魔术检查问题。然后Steelix根据反馈对目标字节执行局部穷举突变。 FairFuzz [28]修改了AFL的确定性突变，以确定输入中无法突变以保证击中稀有分支的部分。它对执行程序中稀有部分的输入进行优先排序，并根据模糊反馈调整突变。在上面提到的模糊器中，Steelix是最接近InsFuzz的一种。由于按近字节进行启发式搜索，因此Steelix无法突破来自输入不同部分的魔术字节比较。

基于污点分析的方法通常用于有效地推断种子输入与目标程序逻辑之间的关系。 BuzzFuzz [23]通过动态污点分析检测影响潜在攻击点的位置，然后对这些区域进行模糊处理。 TaintScope [48]使用动态污点分析来识别校验和字段，并与符号执行相结合以生成触发潜在漏洞的输入。 AUTOGRAM [26]使用动态污点分析来追踪每个输入字符的数据流，并学习反映有效输入结构的上下文无关语法。 VUzzer [40]通过使用基于Pin动态分析框架[32]的动态污点分析来推断输入中魔术字节的偏移量。 **Angora [17]使用可伸缩的字节级别的污点跟踪来识别这些输入中值得突变的字节。**

基于符号执行的方法通常用于生成可以绕过程序中的健全性检查的输入[44]。它收集路径约束，然后生成可以通过约束解决引擎绕过检查的新输入。 符号执行面临着路径爆炸的挑战，但是由于它具有生成高覆盖率输入的能力，因此一些方法将模糊处理与其结合起来以提高模糊处理性能，例如KLEE [15]，SAGE [29]，DART [21]，SYMFUZZ [42]和Driller [45]。 Driller是其中最新的模糊器。 当模糊陷入困境时，Diller使用选择性符号执行来收集路径约束并生成新输入以绕过检查。 符号执行由于其重量级的特性而减慢了软件测试的速度。 因此很难应用于现实世界中的应用[16]。

这些工具从有效输入文件中学习以丢弃无效输入或生成新的种子输入。MoWF [37]利用有效文件中有关文件格式和数据块的信息来排除最无效的输入。

Skyfire [47]利用从大量现有输入中学习的知识来生成新输入。 Learn＆Fuzz [25]使用样本输入和基于神经网络的机器学习来生成适合于输入模糊的输入语法。 深度强化模糊测试[14]使用强化学习来产生高回报的新输入。 Rajpal等。 [39]提出了一种方法，该方法从过去的模糊探索中学习输入文件中的模式，以指导未来的神经网络模糊探索。 所有这些工具都需要大量输入文件作为训练集。 但是，InsFuzz没有这样的要求。

Rebert等。 **[41]专**注于优化种子选择问题并设计六种不同的算法。 他们将问题表述为整数线性规划问题，以衡量种子选择算法的质量。 文旭等。 设计并实现了专门用于模糊测试的新操作原语，以提高模糊测试性能[50]。 **CollAFL [22]**观察了AFL中的路径冲突，这阻止了模糊测试者发现潜在的路径并就模糊测试策略做出明智的决策。 CollAFL通过确保每个边缘都有唯一的哈希来解决AFL中的哈希冲突问题。