**Cross-Architecture Bug Search in Binary Executables**

主要讨论了现有软件在包含漏洞且被编译到三个热门的架构（x86/64，ARM和MIPS）时，如何利用某个架构的分析技术来帮助分析另外架构上代码实现的漏洞检测问题。

**当时错误发现方法不足：**

i) 需要源代码，

ii) 仅适用于单一架构（通常是 x86）。

iii)依赖于动态分析，这对于嵌入式设备来说本来就很困难。

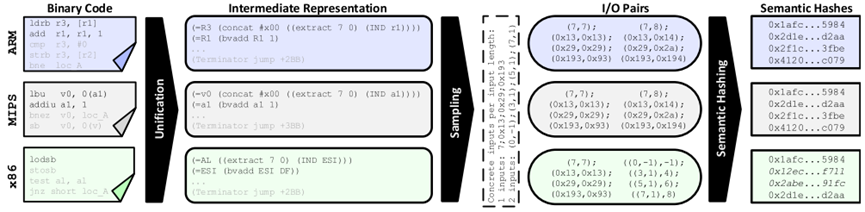
**解决方法：**

1. 将二进制代码转换为中间表示来解决这个问题，从而产生具有输入和输出变量的赋值公式。

2. 对具体输入进行采样以观察基本块的 I/O 行为，从而掌握它们的语义。

3. 使用 I/O 行为来查找行为类似于错误签名的代码部分，从而有效地揭示包含错误的代码部分。

**具体方法：**



1. 将汇编指令转换为IR（中间代码使用VEX-IR表示），从而生成一个赋值公式列表，将其表示为易于解析的 S-Expressions（符号表达式）。赋值公式详细说明了输出变量如何受其输入的影响。
2. 使用随机的具体输入值（虚线框），对这些公式的输入/输出行为进行采样（我们仅说明第一个公式的采样）。
3. 在 I/O 对上构建语义哈希，这使我们能够有效地比较 I/ O 基本块的行为。

在搜索阶段，使用转换后的bug签名 (即，它表示为赋值公式的图)，以识别类似转换的二进制文件中的错误。在目标程序中为错误签名的所有单个基本块寻找有希望的匹配候选。对于每个这样的候选对，应用CFG驱动的，贪婪的，但局部最优的展宽算法。

**具体方法：**

1. Bug Signatures

一个bug signature就是正常二进制代码：它由基本块和基本块之间可能的控制流转移组成。因此，选择任何一个基本块都可以成为一个bug signature。签名长度难以限定，过长容易绝对化、过短容易误报。

1. Unifying Cross-Architecture Instruction Sets

指令集、调用约定、通用和专用 CPU 寄存器集以及内存访问策略也因每个架构而异。使用现成的反汇编程序来提取二进制代码的结构（例如函数、基本块和控制流程图）。将复杂的指令转换为简单的、类似 RISC 的统一指令。

1. Extracting Semantics via Sampling

将基本块的每个输出变量的计算步骤累积了起来，这样就可以对于每个寄存器和输

出内存地址精确的assignment formulas。

为了从具体的跳转地址中进行抽象，定义继承基本块通过函数符号名或者基本块的编号来进行跳转。在为每个基本块提取出formulas后，需要使用定理证明来对assignment formulas进行简化来标准化。因此将这些RISC-like的指令累积起来，然后映射到定理证明器的结构，定理证明器返回S-表达式。

为了能够实现bug finding的目标，作者将代码等价性的条件放宽到代码相似性。为此，作者采用了sampling的方法来提取程序的语义信息。首选作者随机生成一些具体的值，输入到每个基本块的formulas来进行计算，观察formulas的输出结果，作为语义相似性的依据。

1. Similarity Metric via Semantic Hashes

由于计算两个基本块之间相同的I/O对的个数，来通过Jaccard等方式来计算它们之间的相似性，这种计算效率比较低，因此作者采用LSH中的MinHash方法通过合并I/O对来计算基本块的hashes，使得基本块I/O对集合的比较变成了基本块的hash值的比较。

1. Comparing Larger Binary Structures

错误签名通常由多个基本块组成，并根据 CFG 捕获代码结构。

扩展了单个基本块与旨在识别整个错误签名的算法的比较。

1. 从错误签名中选择一个基本块，并将其与相关程序中的所有基本块进行比较。
2. 使用了一种称为最佳命中扩展 (BHB) 的算法，该算法沿错误签名和目标程序中的 CFG 扩展初始候选匹配。 BHB 以贪婪但局部最优的方式运行，直到匹配跨越整个签名。
3. 对 bug 签名中的所有基本块重复 BHB，从而生成按与签名的相似性排序的函数列表

通过CFG把之前基本块的相似组合成更大规模的相似