Cross-Architecture Bug Search in Binary Executables

JAN 6TH, 2016

论文下载：http://www.christian-rossow.de/publications/crossarch-ieee2015.pdf

- 问题限定在Bug Search，就是说，我们已知有一个bug，一个可能使用了该Bug代码的另一个架构的binary钟是否也有相同的Bug
    - 问题范围缩小，不再是多对多的二进制代码(匹配,识别),对速度的要求降低
    - 因为是我们已知Bug，所以需要寻找的目标代码的模式（签名）是允许半手工生成的
- 贡献:
    - 跨架构
    - 生成基本快签名的方式很有趣
- 方法:
    - 二进制代码反汇编(IDA)
    - 中间语言VEX-IR
    - 生成基本块符号表达式
    - 利用符号表达式生成基本块签名
- 基本块签名:
    - 表达式树的签名

        - 每个基本块都是由若干个输出的表达式树构成的
        - 每个基本块的输出依赖的输入都不多(0.3%少于等于4个)
        - 我们队每个表达式树做签名
        - 签名的方式就是:如果这个表达式树有n个输入，就使用预先生成好的生成100个n维tuple做输入，算出输出，得到一个长度为100的序列，再把每个输入的n维重排列，一共生成一个长度为100*n!的序列作为这个表达式树的输入
        - 我们现在有了很多输入输出的关系(a, b, c, d) → e 表示 作为基本块某四个输入为(a, b, c, d)某个输出为(e)
        - 生成CRC(abcde)备用
    - 那么现在我们有很多CRC了，如果两个基本块的CRC列表完全相同那我们可以认为这两个基本块是相同的
        - 但是比较两个CRC列表有多少相同元素还是满，(主要是CRC列表太大了?)
        - 那么我们可以比较两个CRC list的相似性
            - MiniHash,使用800个简单的Hash函数把一个基本块的CRC list转换成800维的向量，然后直接比较向量的就可以了
            - 有论文可以证明这样的得到的相似度的是O(800^-0.5)的，作者给出了误差是3.5%
            - 不过我们没有直接这样做，而是首先把CRC list按照输入的个数分组，对应组分别求相似度，然后按照双方的组大小的和加权(需要公式图)

- 目标代码搜索:
  - 现在我们有什么:
    - 两个基本块是否相似的快速判定方法
    - 需要寻找的BUG的代码（用户给出有入口点和出口点一个CFG）
  - 我们要解决的问题:
    - 稍微模糊一些，由于优化级别的问题可能有一些东西会被优化,导致一些基本快不见了。。
  - 算法:
    - 首先找一个可能的入口点(相似度足够高的基本块)
    - 然后将 (相似度，原入口点，可能目标入口点) 放入优先级队列
    - 然后不断从队列中取出里面相似度最高的那一组
    - 如果取出的这一组的两个基本快没有任何一个被匹配过就认定他们应该配对，
    - 然后将得到两个基本块的后向节点组，任意组合(相似度，基本块1， 基本快2) 放入队列
    - 然后将得到两个基本块的前向节点组，任意组合(相似度，基本块1， 基本快2) 放入队列
- 最后，我们最多得到一个和用户给出CFG同样大小，也可能更小的CFG
- 根据这个CFG每个基本快对应的相似度，加权，算算算。。