RetroWrite: Statically Instrumenting COTS Binaries for Fuzzing and Sanitization 阅读报告

1. 当前软件安全分析中存在的问题：

对于开源软件我们已有的许多安全分析方法已经能够较好地解决其安全分析的问题。但是对未开源的软件，或者是依赖于第三方的库等文件，我们很难对其进行有效的安全分析。

当前常用的方法有两种，一种是动态二进制转换加上sanitizer，但其缺陷是开销较高；另一种方法是静态二进制重写技术，但是纯靠静态是不可靠的。不能完全恢复符号信息，难以添加一些安全性检查等等。

已有的工具及其局限性如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | Real-world applications | Performance | Flexibility |
| Dynamic Binary Rewriting[PIN DynamoRIO] | ✔ | ❌ | ❌ |
| Trampoline-based Static Rewriting [Dyninst] | ✔ | ❌ | ❌ |
| Reassembleable Assembly [Uroboros ramblr] | ❌ | ✔ | ✔ |

因此，本文要做的就是要构成一个能够满足以上三个需求的框架。理想的方案就是利用静态重写器智能地进行插桩。创建一个类似于编译后的文件的程序文件集，并带有重定位符号供链接器分析。

1. 存在的挑战：
   1. Hardcoded Relative Offsets

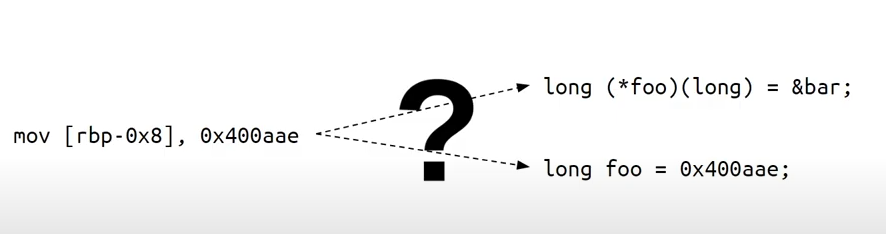
对于这类代码，如果简单地添加一些代码会破坏整个代码内容，如图：



左边部分是源代码，右边部分是加入新代码，可以看到原本跳转到第一行的代码功能现在跳转到了新代码，破坏了源代码的功能。所以需要识别所有的指针并重新调整它们。

* 1. Scalar and References are Indistinguishable

由于二进制文件缺少类型信息，所以消除引用和标量常量之间的歧义是二进制重写需要完成的另外一项挑战。

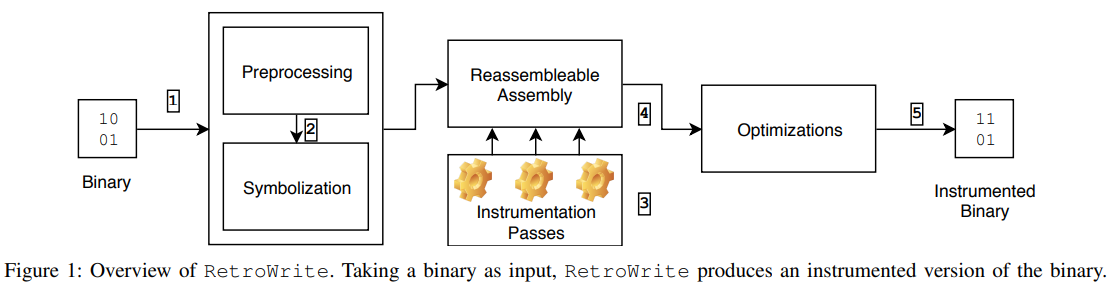


这种误判会影响程序功能，而此前的解决方案依赖于启发式来解决。

基于以上，Retro write原则上是一个静态重写位置独立的代码（PIC）的框架，使用重定位信息来替代启发式的方式。

1. Retro write实现原理

本文实现了两种工具分别是afl-retrowrite和asan-retrowrite。其中，afl-retrowrite性能近似于afl-clang优于仅限于二进制的一些工具，例如afl-qume、afl-dyninst等等。而asan-retrowrite则仅限于二进制文件，集合了Asan，在吞吐量和错误检测率方面优于Valgrind memcheck。

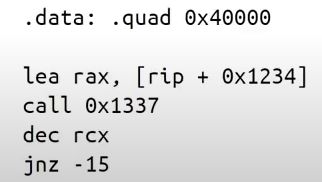


第一步进行预处理，加载一些汇编所需要的二进制文件的文本和数据，加载一些辅助信息例如二进制文件中的符号和重定位，识别和添加直接控制流转移的边缘，构成CFG。

第二步使用上步中的CFG的重定位信息识别数据和代码部分中可符号化常量转变为汇编标签，输出可以重新组装的程序集。

例子：

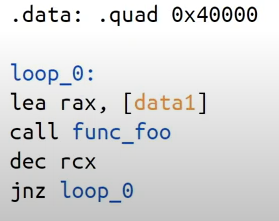




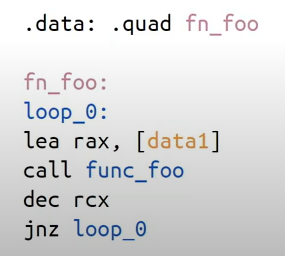
1. PC-relative control flow



1. Rip-relative addressing



1. Data relocations



第三步即进行插桩工作。

第四步就是进行一些优化工作，分析插桩、确定需要的寄存器数量等等。

第五步，最后重新组装这些程序集，生成已插桩的二进制文件。

1. Retro write具体实现
2. afl-retrowrite: coverage instrumentation

位图计数基本块命中次数；

对基本块进行插桩后，基本块图更新；

与afl-gcc进行交互；

1. asan-retrowrite：binary-only address sanitizer

插桩部分:

Retrowrite API识别插桩位置，两种插桩方式:①内存检查插桩（不允许非法访问）②分配检测:跟踪字节的分配状态

内存区域：

堆heap：拦截malloc或其他分配函数等等

栈stack：识别栈帧

全局global：

需要确定全局边界

1. 实验评估

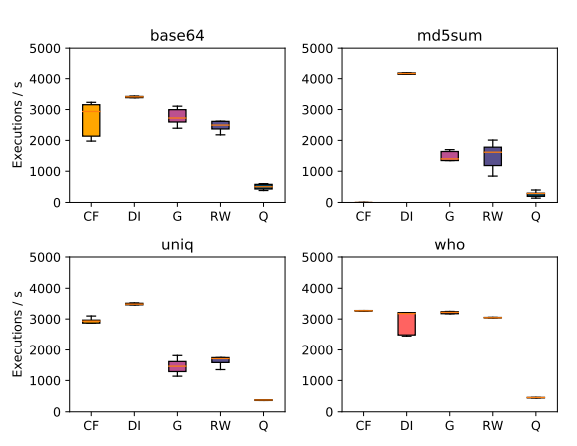
RQ1:RetroWrite能适应大型的二进制文件吗?

RQ2:是否在以下方面显著改进了最先进的二进制内存检查器:运行时开销、覆盖率（相当于漏洞检测率）

RQ3:是否在以下几个方面优于基于源的内存损坏检测器(如ASAN):运行时开销、覆盖率（相当于漏洞检测率）

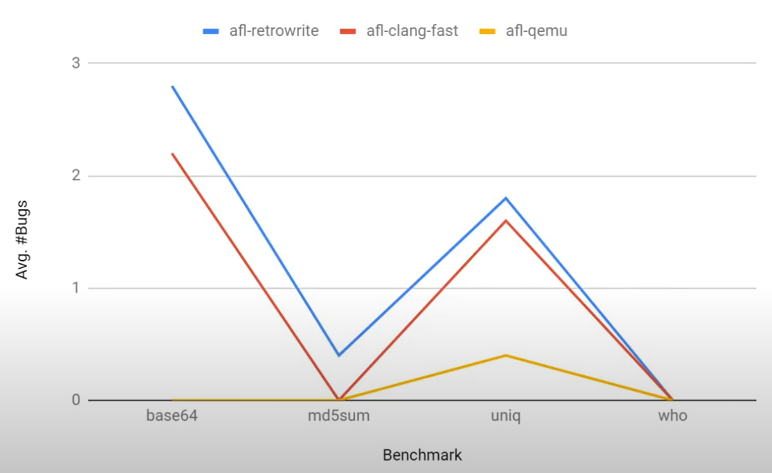
RQ4:本文覆盖测量与基于源的AFL测量相比如何？该方案是一个可行的afl-qemu的替代吗？

1. afl-retrowrite的覆盖率方面



统计了运行5次的吞吐量，可以看到retrowrite优于仅限于二进制的afl-qemu4.5倍，与开源检测时具有相同的性能。

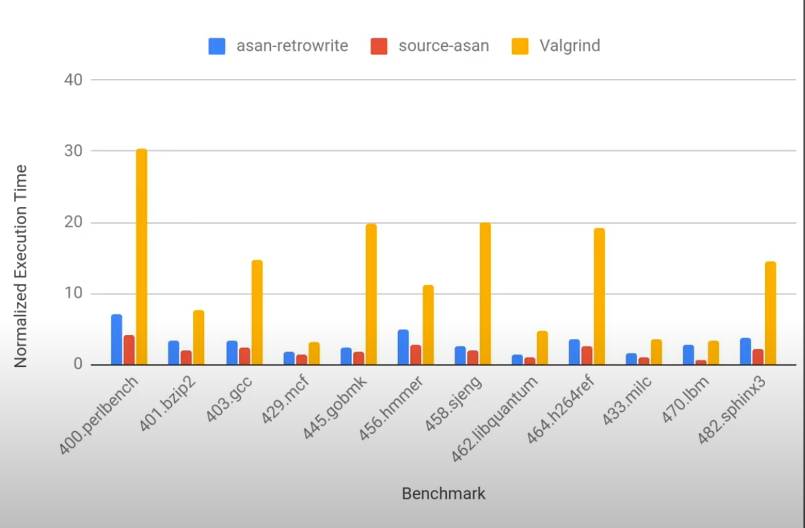
1. afl-retrowrite的漏洞检测方面



明显优于afl-qemu，且与源代码的插桩检测效果相差不大，甚至优于有源代码的检测。

（3）asan-retrowrite

比现有的二进制解决方案更快、性能类似于addresssanitizer



1. 总结

本文实现的是一个用于静态重写二进制文件的可靠的、无启发式的框架。

afl-retrowrite：仅用于二进制应用程序的覆盖插桩，在bug查找能力方面优于afl-gemu和afl-dyninsto，性能相似于基于源代码的模糊测试。

asan-retrowrite：使用retrowrite来测量二进制文件的内存检查，比当前最先进的二进制内存检查更低的开销和更精确的。