G.O.S.S.I.P 阅读推荐 2023-06-30 BinWrap

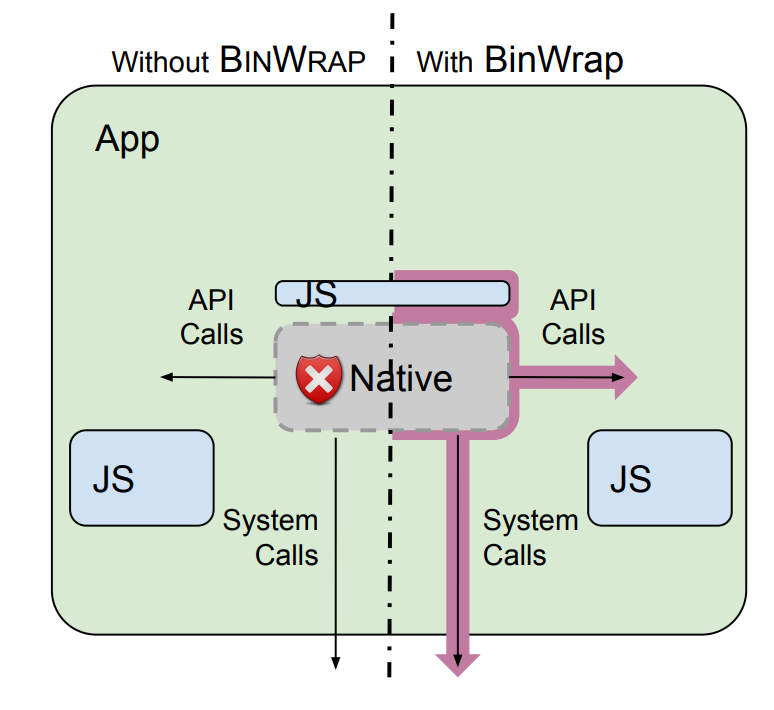
2023-06-30 20:58

2023年上半年的最后一次阅读推荐，我们要为大家介绍ASIA CCS 2023的获奖论文 BinWrap: Hybrid Protection Against Native Node.js Add-Ons

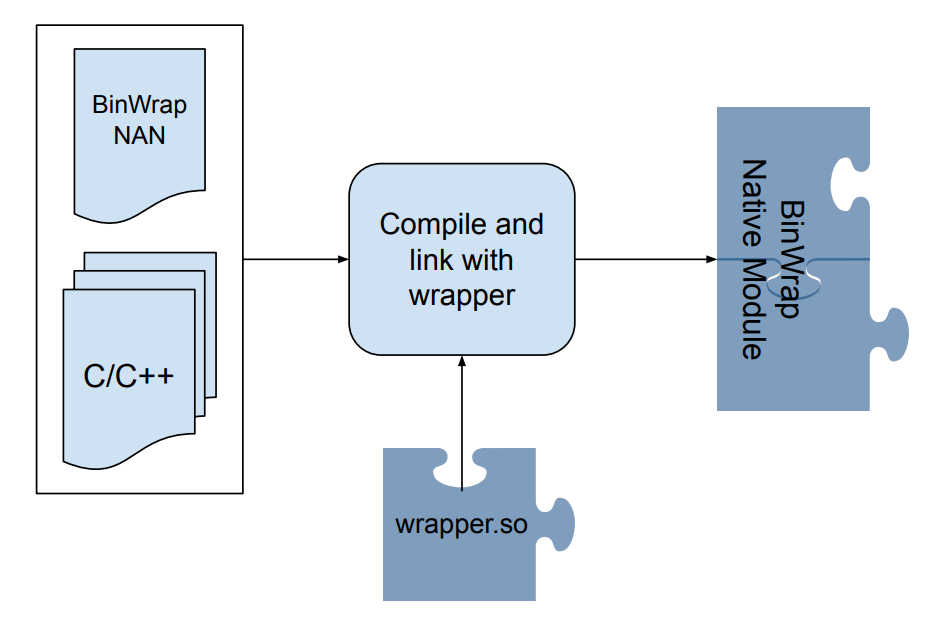


在这篇论文中，作者讨论了针对Node.js环境中的原生代码插件（native code add-on）的隔离。对于这种针对某个执行环境中的三方组件进行隔离的工作，我们并不陌生，比如浏览器沙盒对三方库的限制，以及Rust语言中对unsafe代码的使用的隔离等等。Node.js中同样包含了许多以原生代码形式执行的插件（当然无外乎是“历史遗留代码重用”以及“代码执行效率”这两方面考量），此前并没有专门为此类型的插件设计的隔离，因此作者在本文中设计并实现了BinWrap隔离机制，为Node.js中代码的执行提供更细粒度的安全防护。

下图展示了（高层次的）BinWrap隔离机制的设计。更具体一点来说，对于（不可信的）原生代码插件，BinWrap首先对其进行编译，然后在二进制代码的层面进行分析（作者专门强调他们更喜欢分析二进制代码，分析结果会更加精确一些），找到所有调用的syscall，也附带把那些Node.js内部的一些敏感API（例如v8::External::New(v8::Isolate\*, void\*)这种涉及到内存管理的）都标记出来。接下来，BinWrap会引入一个隔离层（果然是那句经典名言——“计算机领域的任何问题都可以通过增加一个中间层来解决”），接管所有的敏感syscall和API call调用。

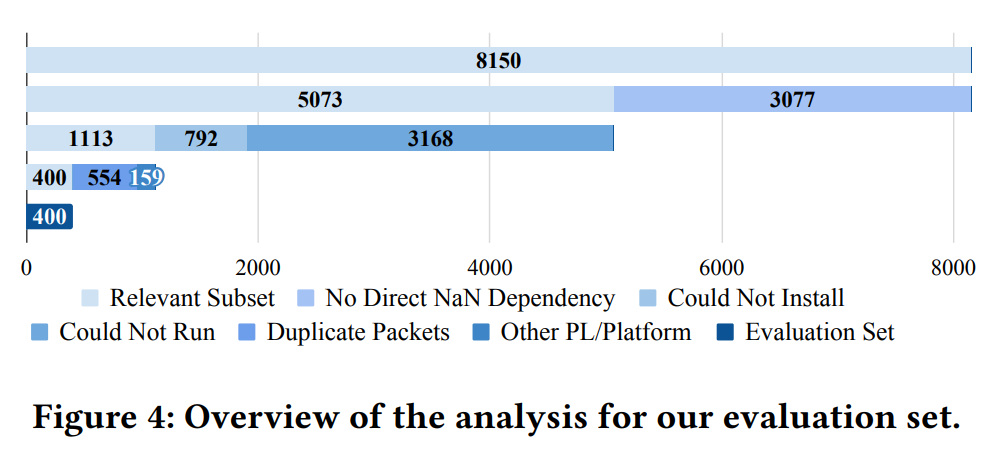


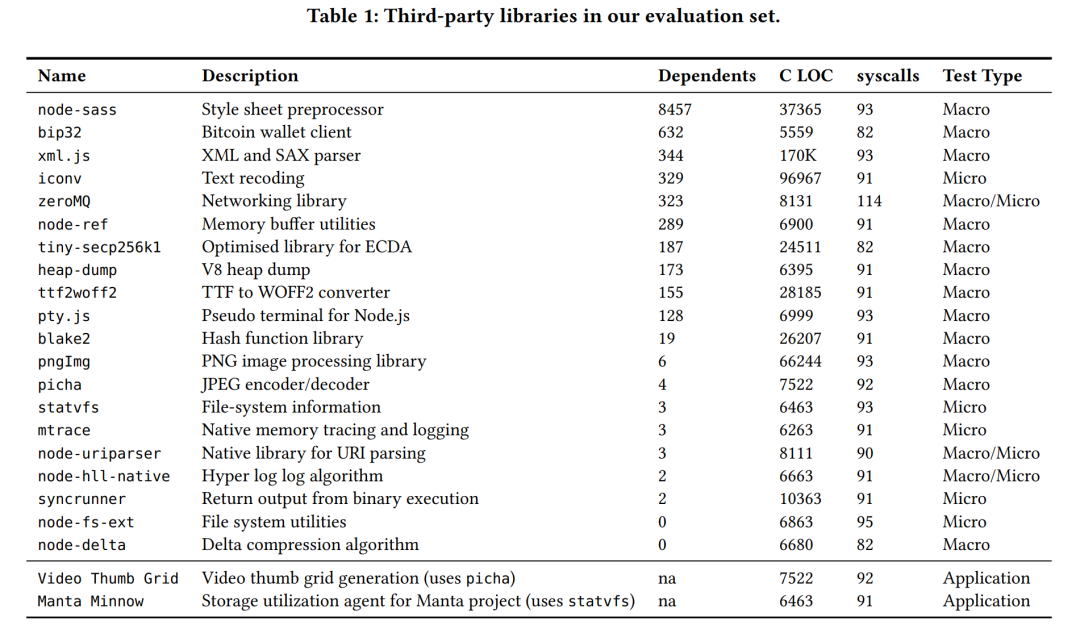
如下图所示，BinWrap会在针对原生代码插件的编译过程中，引入一个wrapper.so也就是前面提到的隔离层，当这个so加载的时候，就会利用dynamic symbol interposition（可以参考一下 https://maskray.me/blog/2021-05-16-elf-interposition-and-bsymbolic 这篇文章）的技术，把原生代码插件里面的原来的敏感函数调用全部拦截下来，不允许插件直接去调用任何高权限的API。



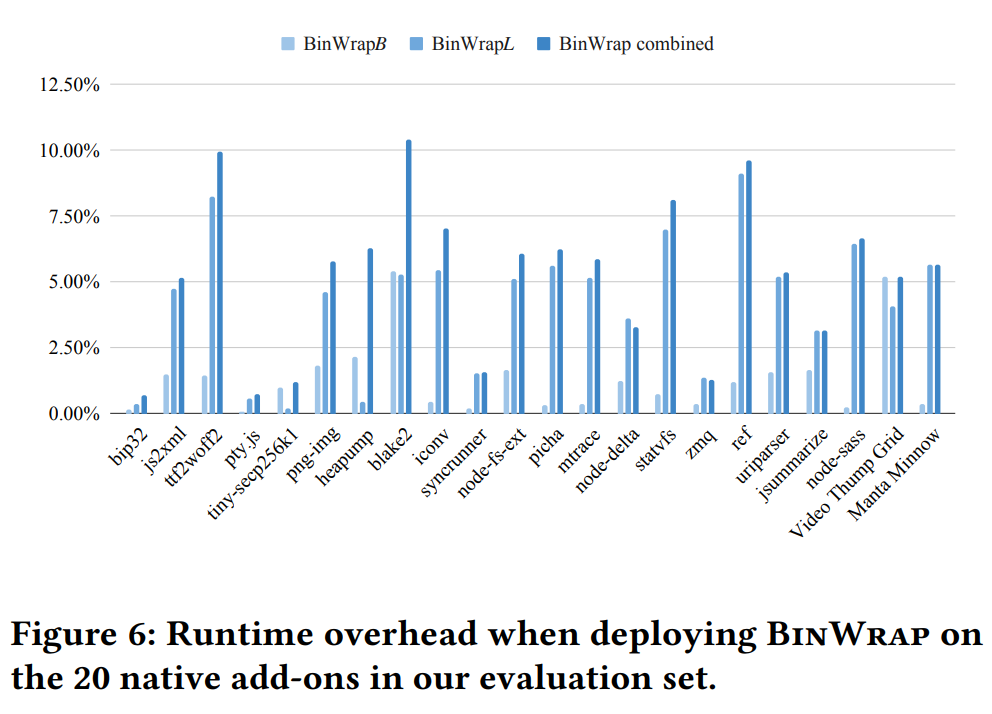
从更为具体的实现细节角度，BinWrap会将原生代码插件以一种restricted thread的形式来执行，并通过两个我们现在应该都比较熟悉的技术——Intel MPK和seccomp-BPF——来实现对线程的限制：通过设置seccomp-BPF filter，可以限制syscall的访问，通过Intel MPK可以对不同的内存区域设置访问控制，这样就可以高效地实现对不同thread的隔离。

作者调查了NPM ecosystem中的1508366个package，逐步缩小调查范围，最终选择了400个具备test case的原生代码插件（如下图所示），然后从里面挑出来20个作为最终的evaluation set进行实验（见下面的Table 1）

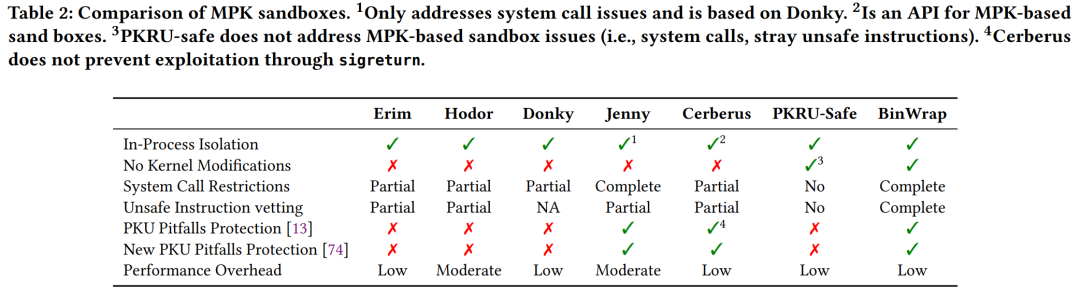




BinWrap在实验中对抗了已知的CVE（CVE-2018-11499、CVE-2018-18577、CVE-2019-3822、CVE-2020-28248），同时从性能指标上看，BinWrap和大部分使用MPK技术的隔离系统差不多，overhead最高也就是10%左右：



最后，作者还和其他一些通用的基于MPK的隔离机制进行了对比：



论文：https://cs.brown.edu/people/vpk/papers/binwrap.asiaccs23.pdf  
代码：https://github.com/atlas-brown/binwrap 【你想多了，这是个空的repo】