# 机器学习指导的静态类型状态分析use-after-free检测

## 摘要

类型状态分析依赖于指针分析来检测临时内存安全错误，例如，释放重用（UAF）。 对于大型程序，可伸缩指针分析通常难以准确地分析其难以解决的“危急情况”，例如不可行的路径，递归循环，循环，数组和链接列表。由于指向信息的声音过于逼近，因此将保守地报告大量虚假别名，从而导致相应的类型状态分析报告大量虚假警报。因此，实际上，对于堆密集型客户端，如UAF检测，类型状态分析的用途变得非常有限。

我们介绍了Tac，这是一种静态UAF检测器，它通过机器学习来弥合类型状态和指针分析之间的差距。 Tac通过使用支持向量机（SVM）了解程序功能与UAF相关的别名之间的相关性，并应用该知识进一步消除指针分析不准确报告的与UAF相关的别名的歧义，因此只有经过SVM分类器验证的别名才可通过typestate分析进一步研究。尽管Tac不健全，但它代表了一种用于UAF检测的实用typestate分析方法。我们已经在LLVM-3.8.0中实现了Tac，并使用一组八个开源C/C++程序对其进行了评估。 结果表明，Tac是有效的（就发现5个已知的CVE漏洞，1个已知的错误和8个虚假率低的新错误而言）和可扩展的（就在短短4个小时内用2,098 KLOC分析大型代码库而言）。

## 1.介绍

释放重用（UAF）漏洞，即C/C++程序中悬空的指针取消引用（访问已释放的对象）可能导致数据损坏[14、70]，信息泄漏[32、53]，拒绝服务攻击（通过程序崩溃）[11]和控制流劫持攻击[9、19、20]。 由于各种缓解技术[14，61，78]，其他内存损坏错误（例如缓冲区溢出）已变得难以利用，而UAF最近已成为利用的显着重要目标[32，77]。

近年来，目睹了关于检测或缓解UAF漏洞的越来越多的研究。大多数现有方法都通过维护影子内存[42、52、69]和执行运行时检查[9、32、77]来依靠动态分析技术。动态分析不会产生或几乎不会产生误报，但会产生不可忽略的运行时和内存开销，从而阻碍了其在生产环境中的采用。 此外，由于使用了代码工具，动态分析通常会遇到二进制不兼容的问题[61]。当用作漏洞检测器时，动态方法通常会受到所用测试输入的限制，因此可以提供较低的代码覆盖率并错过真正的错误。

静态分析近似于编译时的程序行为，它不受上述限制的影响，但需要进行可伸缩但精确的指针分析，以便在大型程序中以较低的误报率找到内存错误[32]。类型状态分析[21，57]代表了一种用于静态检测C/C++程序中临时内存安全错误的基本方法。例如，可以基于图1所示的有限状态自动机（FSA）来检测UAF错误。通过静态分析所有会影响可能程序路径上的状态转换的语句（例如，malloc点，free点和加载存储时的指针取消引用），可以跟踪对象o的类型状态。达到错误时，将报告对象o的UAF警告。当free点free(p)沿着控制流路径到达use点use（q）（表示由q指向的同一个对象（例如，\* q）上的内存访问）时，就会发生这种情况。 q是别名，即p和q指向o。在下文中，这种别名被称为与UAF有关。双重错误是UAF错误的特殊情况。