控制流劫持攻击一直以来是软件安全领域的重要研究课题之一。面向返回的编程（Return-Oriented Programming，简称ROP）攻击作为当前最主要的控制流劫持攻击方式更是近年来的研究重点，ROP攻击是一种利用代码复用技术的攻击方法，攻击者通过内存漏洞影响程序的控制流，将代码中已有的以间接跳转为结尾的短指令片段（又称为配件，gadget）串联起来以实现预设的攻击目标[1,2,3,4]。

控制流完整性(Control Flow Integrity，简称CFI)技术作为一种直接面向控制流进行保护的方式，能够有效的防御ROP攻击。CFI技术通过监视程序运行过程中的控制流转移过程，使其始终处于原有控制流图所限定的合法范围内。具体的做法是分析程序的控制流图，重点关注间接转移指令，如间接跳转、间接调用和函数返回等指令，获取相应的白名单。在程序运行过程中对间接转移指令的目标进行检查核对，而攻击者对控制流的劫持会导致目标不在白名单中，此时可迅速进行阻断，保证系统安全[5,6,7,8,9]。CFI技术可分为细粒度和粗粒度两种实现方式。细粒度CFI技术（例如[5,6]）严格检查每一个间接转移指令的转移目标，但会严重影响程序的执行效率，通常会带来20%-50%的额外性能损耗；粗粒度CFI技术（例如[7,8,9]）将一组或相近类型的目标归到一起进行检查，可在一定程度上降低开销，但会使安全性降低，攻击者能够通过精心设计的配件，组合成配件链，继续实施代码重用攻击。以上的CFI技术均是基于二进制插桩的，需要修改源码或者通过反汇编重写二进制代码，这给这些技术的使用带来额外的工作量，并且会引入较大的性能开销或者检测效果不够精确。

此外，一些研究引入硬件机制来降低CFI技术的开销，提高检测精度。例如，Pappas等提出了kBouncer方法[10]，该方法利用最近分支记录（Last Branch Record，简称LBR）捕获最近的16次跳转信息，在敏感系统调用处对捕获的16次跳转进行安全性判断。Cheng等提出ROPecker方法[2]，该方法也是利用最近分支记录捕获程序流的方式进行ROP攻击检测。该方法在运行时检测过去和未来的执行流中是否存在长配件链来进行攻击检测，还通过滑动窗口的机制来进一步提高准确性和高效性。但是，这两种方法都是一次性针对16个间接分支进行检查，会面临历史覆盖不足的问题，导致检测精度下降。Xia等提出了CFIMon[11]，该方法采用分支跟踪缓冲区（Branch Trace Store，简称BTS）来捕获程序运行过程中跳转指令的信息。虽然分支跟踪缓冲区能够将程序整个执行过程中的所有跳转指令的历史信息都记录下来，但相比使用最近分支记录，使用分支跟踪缓冲区会引入更大的性能开销。此外，以上三种方法均是针对长配件链的攻击进行检测，对于短配件链的攻击情况检测效果不佳。

因此，针对已有CFI技术面临的影响防御效果和性能的几个问题：1）需要修改源码或者通过反汇编重写二进制代码[7,8,9]；2）需要针对所有间接分支进行检查，检查点过多[5,6,11]；3）依赖有限资源的历史纪录，容易遭受历史覆盖攻击[2,10]；4）对短配件链攻击不能进行有效检测[2,10,11]，本文提出了BranchChecker——一种基于硬件分支处理的ROP攻击防御方法，图1是该方法的基本框架。



图1. 方法的基本框架

该方法以硬件性能监控单元（Performance Monitor Unit，简称PMU）产生的性能监控中断（Performance Monitoring Interrupt，简称PMI）为检测触发点，基于性能监控中断捕获到的用户态预测失败的跳转间接分支提交事件，通过最近分支记录（Last Branch Record，简称LBR）获取精确的分支信息，然后结合内存指令信息和系统调用参数信息进行配件（gadget）检测，并识别出ROP攻击可能利用的配件链（gadgets chain）。

本方法具备以下优点：

1）基于LBR获取精确的分支信息，基于该信息结合内存指令信息检测疑似配件，不需要修改源码或者重写二进制码，能直接针对二进制代码进行保护，增加了方法的实用性。

2）通过性能监控单元采样预测失败的间接分支事件，针对每个预测失败的间接分支处触发性能监控中断进行CFI检测，避免了对预测正确的间接分支进行检查，能够有效减少检测点，控制性能开销。

3）针对历史覆盖攻击问题，一方面，本方法针对每个预测失败的间接分支进行配件检测，该检测使用的是上一分支信息，不会被覆盖；另一方面，针对配件链检测，本方法会识别“长空配件”，避免了长空配件引起配件链检测中断，覆盖配件链检测的历史信息。

4）引入系统调用参数检测方法，通过判断其与上一个配件对应的系统架构值之间的相似性来检测攻击，能够识别已有CFI检测方法覆盖不到的短配件链的ROP攻击。

本文后续章节组织方式如下：

1. 首先，介绍本文的研究背景；
2. 然后，对本文提出的BranchChecker检测方法进行详细介绍；
3. 其次，对BranchChecker的系统实现进行详细的描述；
4. 接着，对BranchChecker系统进行性能及安全性评估；
5. 第五，针对相关研究进行阐述；
6. 最后，对本文进行总结与展望。