1. 引言
   1. 研究背景
   2. ROP攻击和防御发展现状
   3. 本文主要工作和贡献
   4. 本文组织结构
2. ROP攻击与防御

2.1 ROP攻击

2.2 ROP防御

2.3 本章小结

1. 基于PIN的ROP攻击检测方法

3.1 假设

3.2 总体设计

基本上，当程序执行必须转移到子程序时，call指令本身将返回地址压入堆栈。然后，被调用的子例程完成其任务并通常返回其原始调用者。但是，有一些例外打破了传统的调用约定，函数返回到其他位置。

我们将在3.4节讨论这些例外。目前，我们假设函数总是返回到最初由堆栈顶部的调用指令（TOS）推送的返回地址。然而，我们的xxx原型实现也处理异常（参见第4节）。

ROP攻击滥用堆栈指针的任务。在普通程序中，在返回函数时，堆栈指针将指向返回地址，以便控制转移回调用函数。如第2节所述，攻击者错误地使用每个指令序列末尾的返回指令，以便将控制转移到后续指令序列。返回指令可以是最初由libc的程序放置的预期指令，或者是非预期的指令，其中返回指令的字节值只是另一个有效指令的后缀。

出于这个原因，我们特别关注如图2所示的调用和返回指令。为了评估在程序执行期间发出的每个返回指令，我们将返回地址的副本存储到单独的影子堆栈中（类似于[12,43,21]）一旦程序发出调用指令。我们检测程序执行期间发出的所有返回指令，并执行返回地址检查，如下所述：

1.在处理器执行指令之前，我们的解决方案拦截指令并评估指令的类型和目标。在实践中，这可以通过二进制检测框架来完成，我们将在3.3节和3.4节中解释。

2.首先，我们检查指令是否是一个调用指令。如果是这种情况，我们将推送的返回地址的副本存储到我们的影子堆栈中（图2中的过渡2a）。否则，我们检查截获的指令是否是返回指令（图2中的转换2b）。

3.如果指令是返回指令，我们的解决方案将检查影子堆栈上的顶部返回地址是否等于程序堆栈顶部的返回地址。如果两个返回地址不同，我们得出结论，发生了返回地址损坏，程序执行被重定向到程序员不想要的指令。

程序堆栈将保存第二个指令序列的起始地址。但是，由于我们的影子堆栈只保存由调用指令本身推送的返回地址，因此它不能包含从函数中间某处开始的该指令序列的起始地址。因此，我们的解决方案可以检测到任何返回地址违规。

显然，在我们的方法中，我们假设在ROP攻击期间发出的指令序列以返回指令结束。实际上，这是Shacham [38]提出的原始ROP攻击。绕过我们的解决方案的一个想法是使用以间接跳转/分支指令结束的指令序列。实际上，Shacham简要提到了使用间接分支指令的能力，最近发表了一篇论文[10]，描述了针对Intel x86的攻击。这些攻击超出了本文的范围。但是，我们将在第7节中讨论如何检测此类攻击。

在3.4节中，我们将描述如何将我们的解决方案实现到像Pin这样的二进制检测框架中。

3.3 技术及工具

3.4 框架概述

框架

3.5 本章小结

1. 实现

4.1 pin

4.2 实现框架

4.3 实验与评估

1. 总结展望

5.1 总结

5.2 展望