**ROP-Hunt：在应用程序中检测返回导向编程(ROP)攻击**

摘要:

Return-oriented Programming (ROP) is a new exploitation technique that can perform arbitrary unintended operations by constructing a gadget chain reusing existing small code sequences.

返回导向编程(ROP) 是一种新型漏洞利用技术，该技术通过重复使用已有的小代码序列(gadget)构造出一条gadget链，从而执行任意的非法操作。

Although many defense mechanisms have been proposed, some new variants of ROP attack can easily circumvent them.

尽管很多防御机制已经被提出，但是一些新的ROP攻击的变种能够轻易绕过这些机制防御。

In this paper, we present a new tool, ROP-Hunt, that can defend against ROP attacks based on the differences between normal program and ROP malicious code.

在本文中，我们将介绍一个新工具ROP-Hunt，它能依据正常程序与恶意ROP代码间的差异来防御ROP攻击。

ROP-Hunt leverages instrumentation technique and detects ROP attack at runtime.

ROP-Hunt利用**插桩**(instrumentation)技术并在程序运行时检测ROP攻击。

In our experiment, ROP-Hunt can detect all types of ROP attack from real-world examples.

在我们的实验中，ROP-Hunt可以在众多真实例程中检测出所有类型的ROP攻击。

We use several unmodified SPEC2006 benchmarks to test the performance and the result shows that it has a zero false positive rate and an acceptable overhead.

我们使用了几个原版的SPEC2006标准来测试ROP-Hunt的性能，结果表明它具有零误报率和可接受的系统开销。

Keywords: Return-oriented Programming · Buffer overflow Detection · Code reuse attack · Binary instrumentation

关键词：返回导向编程、缓冲区溢出、检测、代码复用攻击、二进制插桩

1 Introduction

一、引言

Since the widespread adoption of data execution prevention (DEP) [1], which ensures that all writable pages in memory are non-executable, it's hard for attackers to redirect the hijacked control flow to their own injected malicious code.

由于数据执行保护（DEP）[1]的广泛采用，确保了内存中的所有可写页面都是不可执行的，因此攻击者很难将被劫持的控制流重定向到他们自行注入的恶意代码。

To bypass DEP mechanism, code reuse attack (CRA) techniques are proposed and have become attacker's powerful tools.

为了绕过DEP机制，代码复用攻击（CRA）被提出并成为了攻击者们的利器。

Instead of injection code, they reuse instructions already residing in the attacked vulnerable process to induce malicious behaviors.

攻击者们不再注入代码，而是通过复用被攻击的漏洞进程中的现有指令来构造恶意行为。

Return-into-libc technique [37] is one simple practice of it, in which the attacker uses a buffer overflow to overwrite the return address stored in the stack to the address of the library function chosen to be executed.

Return-into-libc技术[37]是代码复用攻击的一种简单应用，攻击者利用缓冲区溢出漏洞，将位于栈中的返回地址覆写为攻击者挑选出的将要被执行的库函数地址。

Traditional return-into-libc attack leverages libc functions and cannot support arbitrary computation on the victim machine.

传统的return-into-libc攻击利用libc函数，并不支持在受害计算机上的执行任意操作。

Return-oriented Programming (ROP) is another code reuse attack technique, which executes short instruction sequences called gadgets instead of an entire function.

返回导向编程（ROP）是另一种代码复用攻击技术，它执行称为gadgets的短指令序列，而不是执行一整个函数。

ROP was first demonstrated by Shacham [35] for the x86 platform,and was subsequently extended to other architectures [13,16,23,26].

ROP最初由Shacham[35]提出并应用于x86平台，随后被拓展到其他体系结构[13,16,23,26]。

It has been proved that ROP can perform Turing-complete computation [36].

ROP可实现图灵完备计算已被证明[36]。

Some tools have been developed that allow attackers to construct arbitrary malicious programs using ROP automatically [22,24,33,34].

一些允许攻击者使用ROP自动构造任意恶意程序的工具已被开发[22,24,33,34]。

In the last few years, a number of software and hardware defenses have been proposed to mitigate ROP-based attacks.

在过去几年中，许多用于减缓基于ROP攻击的软件或硬件防御方法已被提出。

For example, DROP [17] and DynIMA [20] will trigger an alarm if the small instruction sequences each ending with a ret instruction are executed consecutively.

例如，如果连续执行以ret指令为结尾的小指令序列，DROP [17]和DynIMA [20]将触发警报。

ROPdefender [21] maintains a shadow stack and verifies all return addresses.

ROPdefender [21]则维护一个影子栈，并验证所有返回地址。

Li et al. [27] proposed a compiler for the x86 platform that avoids issuing "0xc3" bytes that can be used as unintended return instructions. Further more, it replaces intended call and return instructions with an indirect call mechanism. However, these mechanisms only focus on the ROP gadgets ending with return instructions and can not defeat other types of ROP-like attacks that capture gadgets without return instructions.

李等人[27]提出了一个用于x86平台的编译器，它能避免生成可用作恶意返回指令的“0xc3”字节。此外，它使用间接调用机制替换了预期的调用和返回指令。但这些机制只关注了以返回指令为结尾的gadget，并不能防御其他类型，即不以返回指令为结尾的gadget的类ROP攻击。

CFLocking [11] and G-Free [30] aim to defend against all types of ROP attacks, but they require the source code which is often unavailable to the end users in the real world.

CFLocking [11]和G-Free [30]旨在防御所有类型的ROP攻击，但它们需要源代码，对于实际的终端用户而言，这些源代码通常难以取得的。

KBouncer [32] covers all ROP attack types, requires no side information and achieves good runtime efficiency. However, it only monitors the application execution flow on selected critical paths, e.g., system APIs. It inevitably misses the ROP attacks that do not use those paths.

KBouncer [32]涵盖了所有ROP攻击类型，且不需要辅助信息，并实现不错的运行时效率。然而，它只是监视目标关键路径上的应用程序执行流，例如系统API。如此，它不可避免地漏掉了那些不使用这些路径的ROP攻击。