A Survey of Return-Oriented Programming Attack, Defense and its Benign Use

一种面向返回编程的攻击，防御及其良性使用的研究

Abstract—The return-oriented programming(ROP) attack has been a common access to exploit software vulnerabilities in the modern operating system(OS). An attacker can execute arbitrary code with the aid of ROP despite security mechanisms are involved in OS. In order to mitigate ROP attack, defense mechanisms are also drawn researchers’ attention. Besides, research on the benign use of ROP become a hot spot in recent years, since ROP has a perfect resistance to static analysis, which can be adapted to hide some important code. The results in benign use also beneﬁt from a low overhead on program size. The paper discusses the concepts of ROP attack as well as extended ROP attack in recent years. Corresponding defense mechanisms based on randomization, frequency, and control flow integrity are analyzed as well, besides, we also analyzed limitations in this defense mechanisms. Later, we discussed the benign use of ROP in steganography, code integrity veriﬁcation, and software watermarking, which showed the signiﬁcant promotion by adopting ROP. At the end of this paper, we looked into the development of ROP attack, the future of possible mitigation strategies and the potential for benign use.

Index Terms—Return-Oriented Programming, ROP Defense, Benign Use of ROP, Code Reuse Attack

摘要：面向返回的编程（ROP）攻击已成为利用现代操作系统（OS）中的软件漏洞的常见途径。尽管操作系统涉及安全机制，但攻击者仍可以借助ROP执行任意代码。为了减轻ROP攻击，防御机制也引起了研究人员的注意。此外，由于ROP具有很好的抗静态分析能力，可以很好地隐藏一些重要的代码，因此对ROP的良性使用的研究成为近年来的热点。程序大小的低开销也有利于良性使用。本文讨论了近年来ROP攻击以及扩展ROP攻击的概念。还分析了基于随机性，频率和控制流完整性的相应防御机制，此外，我们还分析了这种防御机制的局限性。稍后，我们讨论了在隐写术，代码完整性验证和软件水印中ROP的良性使用，它们显示出采用ROP的显着提升。在本文的结尾，我们研究了ROP攻击的发展，可能的缓解策略的未来以及良性使用的潜力。

索引术语：面向返回编程，ROP防御，ROP的良性使用，代码重用攻击

I. INTRODUCTION

一、介绍

The software vulnerability is an important source of security risk in the modern computer system. In early exploit, the attack runs the shellcode on the stack directly. Data execution prevention(DEP) is involved as an emergence of security mechanisms in the modern operating system, using W ⊕ X security model in which the memory location can be writeable and executable at the same time [1]. Despite the employment of such method, the attacker is still able to exploit a software vulnerability. Return-to-libc is the first method to bypass W ⊕ X guard, it can execute functions already existing in libc. In order to execute arbitrary code, return-oriented programming(ROP) was proposed to achieve this goal [2]. In a ROP attack, the attacker finds code fragments already existing in the process’ address space to replace the previous shellcode. To chain these code fragment together, the attacker links them with indirect control transfer instruction, ret. ROP is proved to be Turing complete so that the attacker can use ROP to achieve any function. Up to now, ROP attack has become a necessary step in control flow hijacking.

软件漏洞是现代计算机系统中安全风险的重要来源。在早期利用中，攻击直接在堆栈上运行shellcode。数据执行保护（DEP）是使用W⊕X安全模型在现代操作系统中作为一种安全机制而出现的，其中存储位置可以同时可写和可执行[1]。尽管采用了这种方法，但攻击者仍然能够利用软件漏洞。返回libc是绕过W⊕X保护的第一种方法，它可以执行libc中已经存在的功能。为了执行任意代码，提出了面向返回的编程（ROP）以实现此目标[2]。在ROP攻击中，攻击者会找到进程地址空间中已经存在的代码片段，以替换先前的shellcode。为了将这些代码片段链接在一起，攻击者将它们与间接控制转移指令ret链接在一起。 ROP被证明是图灵完整的，因此攻击者可以使用ROP来实现任何功能。到目前为止，ROP攻击已成为控制流劫持中的必要步骤。

Since the performance of ROP attack on the software vulnerability, some general methods proposed for ROP attack mitigation [3]–[6]. A simple mitigation adapted by the operating system is the randomization approach, like Address space layout randomization(ASLR) [6]. ASLR is a good defense against ROP attacks. In ASLR, the base address of stack, heap, and external libraries is different for each execution, the attacker cannot acquire the exact address of code fragment, such that the attacker cannot chain code fragment together. In addition to randomization approaches, researchers also found defense mechanisms on instruction frequency and control flow integrity. These approaches mitigate ROP attacks to a certain extent, but most of the defense mechanisms have not been applied in practice for the overhead or security reasons.

由于对软件漏洞执行了ROP攻击，因此提出了一些通用的缓解ROP攻击的方法[3] – [6]。 操作系统采用的一种简单缓解措施是随机化方法，例如地址空间布局随机化（ASLR）[6]。 ASLR是针对ROP攻击的良好防御。 在ASLR中，每次执行时堆栈，堆和外部库的基地址都是不同的，攻击者无法获取代码片段的确切地址，因此攻击者无法将代码片段链接在一起。 除了随机方法外，研究人员还发现了针对指令频率和控制流完整性的防御机制。 这些方法在一定程度上缓解了ROP攻击，但是出于开销或安全原因，大多数防御机制尚未在实践中应用。

Return-oriented programming had always been considered as an attacking technique in previous research. However, recent research found the possibility of drive return-oriented programming from the ”dark side” [7]–[9]. Compared with the previous approaches, ROP has a good performance in two fields. On the one hand, ROP uses code fragments already existing in a program, which means there is no extra code added to the program, and thus has a low overhead in size. On the other hand, ROP was originally proposed as a dynamic attack, the static analysis has difficulty in finding ROP attack, which can be considered as a good approach to hide important functions. The concealment and low overhead in program size can fit well in special fields, like code steganography, code integrity and software watermarking. The results showed that drawing into ROP had a significant promotion compared with previous methods, which means ROP has a great potential in the field of security.

在以前的研究中，面向返回的编程一直被视为一种攻击技术。 但是，最近的研究发现，从“黑暗的一面” [7] – [9]可以进行面向驱动器返回的编程。 与以前的方法相比，ROP在两个领域都有良好的表现。一方面，ROP使用程序中已经存在的代码片段，这意味着没有多余的代码添加到程序中，因此开销很小。 另一方面，ROP最初是作为动态攻击提出的，而静态分析很难找到ROP攻击，可以将其视为隐藏重要功能的好方法。 程序大小的隐蔽性和低开销可以很好地适合特殊领域，例如代码隐写术，代码完整性和软件水印。 结果表明，与以前的方法相比，引入ROP具有显着的提升，这意味着ROP在安全领域具有巨大的潜力。

In the last ten years, the situation of software security has become more and more severe, and research has carried out extensive and in-depth results on ROP. These research cover the attack ideas, the corresponding protection methods, and security use. The rest of this paper is organized as follows. ROP are discussed in section II. In section III, we will explain the proposed defense mechanisms against ROP. Then, we will describe the benign use of ROP in section IV. In V, we conclude the report.

在过去的十年中，软件安全状况变得越来越严峻，有关ROP的研究已经取得了广泛而深入的成果。 这些研究涵盖了攻击思路，相应的保护方法和安全使用。 本文的其余部分安排如下。 ROP在第二节中讨论。 在第三节中，我们将说明针对ROP的防御机制。 然后，我们将在第四节中描述ROP的良性使用。 在V中，我们总结了报告。

II. ROP ATTACK MECHANISMS

二、ROP攻击机制

Return-Oriented programming(ROP) was introduced by Shacham [2], a technique provides the attacker with the possibility of executing codes despite the presence of security defense W ⊕ X. ROP was originally proposed to exploit software vulnerabilities in an Inter x86 system, later on, it was proven that ROP can be used in a wide range of architectures including ARM, which can be claimed that ROP is an architecture-independent exploitation mechanism.

Shacham [2]引入了面向返回的编程（ROP），该技术为攻击者提供了执行代码的可能性，尽管存在安全防御W⊕X。ROP最初是为利用Inter x86系统中的软件漏洞而提出的， 后来，事实证明ROP可以在包括ARM在内的各种体系结构中使用，可以说ROP是与体系结构无关的利用机制。

1. Return-Oriented Programming
2. 面向返回编程

In buff overflow and use-after-free (UAF), the attack can easily control the return address in a call stack or change the register of the jump target (function pointer) to form a control flow hijacking. In the early use, the attack laid out shellcode in the stack and redirected the hijacked control flow directly to it. However, with the development of memory executable technology, W ⊕ X, the attack cannot execute shellcode on the stack.

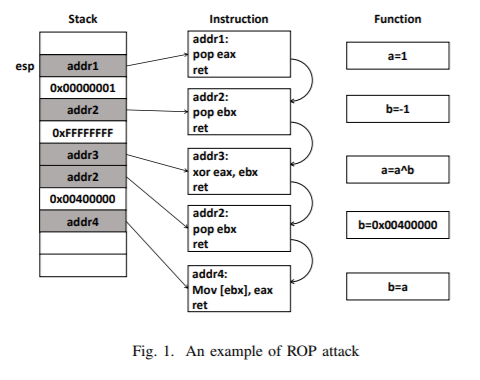
在字符串溢出和释放后使用（UAF）中，攻击可以轻松地控制调用堆栈中的返回地址或更改跳转目标的寄存器（功能指针）以形成控制流劫持。 在早期使用中，攻击将shellcode放置在堆栈中，并将被劫持的控制流直接重定向到它。 但是，随着内存可执行技术W⊕X的发展，攻击无法在堆栈上执行shellcode。

In order to bypass the W ⊕ X mechanism, Solar Designer proposed a method called return-to-libc to bypass it. In returnto-libc attack, the attacker overwrites the return address of the called function with the address of a libc function, such as system() and execve(), such that the attacker can execute unintended functions in an exploit. Unlike previous attacks, return-to-libc can achieve an attack without injecting code. Return-to-libc attack is able to bypass W ⊕ X, but still has some limitations, the attacker cannot run arbitrary code by calling existing functions in libc. To break through these restrictions, Shacham made use of code fragments ending with ret to replace original shellcode [2]. The attacker layouts address of code fragments ends with ret on the stack, achieves continuous calls of dispersive code fragments in memory, and thus forms a large-scale code fragment reuse attack. The code fragment is called gadget by Shacham and can be found by Galileo algorithm. The gadgets can be divided into intended gadgets and unintended gadgets, where intended gadgets come from function epilogue and unintended gadgets come from unaligned instruction.

为了绕过W⊕X机制，Solar Designer提出了一种称为return-to-libc的方法来绕过它。在return-to-libc攻击中，攻击者使用libc函数的地址（例如system()和execve()）覆盖被调用函数的返回地址，从而使攻击者可以在利用中执行意外的功能。与以前的攻击不同，return-to-libc可以在不注入代码的情况下进行攻击。 return-to-libc攻击可以绕过W⊕X，但仍有一些限制，攻击者无法通过调用libc中的现有函数来运行任意代码。为了突破这些限制，Shacham利用以ret结尾的代码片段替换了原始的shellcode [2]。代码片段的攻击者布局地址以堆栈上的ret结尾，实现了内存中分散代码片段的连续调用，从而形成了大规模的代码片段重用攻击。该代码片段被Shacham称为gadget，可以通过Galileo算法找到。这些gadget可以分为预期的gadget和意外的gadget，其中预期的gadget来自函数结尾，意外的gadget来自未对齐的指令。

Figure 1 shows an example of a ROP attack, the aim of the attack is to XOR two values and store the result to memory. The attacker should layout the address of existing instruction fragment and data needed during the attack. After a single gadget, the ret instruction updates general registers’ value and redirects the control flow to the address of stack top, the next gadget is then executed and finally, a specific complete function logic is implemented.

图1显示了ROP攻击的示例，该攻击的目的是对两个值进行异或运算并将结果存储到内存中。 攻击者应布局攻击过程中所需的现有指令片段的地址和数据。 在单个gadget之后，ret指令将更新通用寄存器的值，并将控制流重定向到堆栈顶部的地址，然后执行下一个gadget，最后实现特定的完整功能逻辑。



As figure 1 shows, to achieve a ROP, the attacker needs three factors:

* **An overflow vulnerability** In general, the attacker needs to take over the control flow of a vulnerable program, like the %esp register, to initialize the first gadget.
* **instruction fragment address** The attack needs to acquire the exact address of gadget to lay out the stack.
* **instruction fragment** The code sequences often contain two or three instructions, when executed by the processor, operate a small amount of work. ROP uses a set of these sequences to perform an atomic task like load, restore, arithmetic operations, etc. After executing the sequence of instructions in the gadget, the control flow needs to jump to a controllable address to continue to execute the subsequent gadget to form a gadget-chain.

如图1所示，要实现ROP，攻击者需要三个因素：

* **溢出漏洞** 通常，攻击者需要接管％esp寄存器之类的漏洞程序的控制流才能初始化第一个gadget。
* **指令片段地址** 攻击需要获取gadget的确切地址，以对堆栈进行布局。
* **指令片段** 代码序列通常包含两个或三个指令，当由处理器执行时，它们会执行少量工作。 ROP使用这些序列中的一组来执行原子任务，例如加载，还原，算术运算等。在执行gadget中的指令序列之后，控制流需要跳转到可控地址以继续执行后续gadget，以执行以下操作： 形成一个gadget链。

Shacham also proved that ROP attack is Turing complete, which theoretically illustrates that ROP can achieve any attack. In order to prove that, Shacham proved that the gadgets in ROP attack can implement the following functions 1)Memory load/store; 2) Arithmetic and logic; 3) Control flow transfer; 4) System calls and function calls.

Shacham还证明了ROP攻击是图灵完备的，从理论上说明ROP可以实现任何攻击。 为了证明这一点，Shacham证明了ROP攻击中的gadget可以实现以下功能：1）内存加载/存储； 2）算术与逻辑； 3）控制流转移； 4）系统调用和函数调用。

1. ROP Automation
2. 自动化ROP

It is difficult to find every single gadget in binary, automation must be involved to make ROP attack easier. There are two key points in ROP attack automation: 1)Finding the required gadget, which is called gadget searching; 2) Chaining gadget together to achieve the corresponding attack, which is called gadget compiling.

* **Gadget searching** The well-known tools used for gadget finding are ROPgadget, ROPEME, ropper, etc. The gadget search algorithms in these tools are based on the Galileo search algorithm proposed by Shacham in 2007.
* **Gadget compiling** Represent gadget with an intermediate language(IL), then using the instruction matching in the compiler, select gadget on the extracted gadgets set to represent the user input in IL for function description, and determine the gadget sequence of final execution. Q [10] was based on QooL.

很难找到二进制文件中的每个gadget，必须进行自动化才能使ROP攻击更容易。 ROP攻击自动化有两个关键点：1）查找所需的gadget，称为gadget搜索； 2）将gadget链接在一起以实现相应的攻击，这称为gadget编译。

* **gadget搜索**用于gadget搜索的著名工具是ROPgadget，ROPEME，ropper等。这些工具中的gadget搜索算法基于Shacham在2007年提出的Galileo搜索算法。
* **gadget编译**用中间语言（IL）表示gadget，然后使用编译器中的指令匹配，在提取的gadget集上选择gadget以表示IL中用于功能描述的用户输入，并确定最终执行的gadget序列。 Q [10]基于QooL。

1. Extended ROP attack
2. 拓展的ROP攻击

Shacham also analyzed the possibility of avoiding using ret to launch an attack. The ret instruction in gadget has two properties that make them useful for return-oriented programming: (1) transfer control of execution; and (2) load immediate values into registers, so a subsequent return will not lose control of the program. As a result, jmp and call instructions was used to take place of ret.

Shacham还分析了避免使用ret发动攻击的可能性。 gadget中的ret指令具有两个属性，这些属性使它们对于面向返回的编程非常有用：（1）执行的转移控制； （2）将立即值加载到寄存器中，因此后续返回不会失去对程序的控制。 最后，使用jmp和call指令代替了ret。

Stephen Checkoway proposed that it is possible to perform return-oriented programming without using ret instructions [11]. The replacement for the ret instruction is an updateload-branch sequence, which 1) updates the global state of registers that act as the return-oriented program’s instruction pointer, 2) uses the updated state to load from memory the address of the next instruction sequence to execute and 3) branches to the loaded address. The update-load-branch updates program’s global state and transfers control to the next instruction sequence. On the x86, the author recommends to use “pop x; jmp \*x”, where “x” is any general purpose register. The method proposed by Stephen showed the possibility of using jmp and call to replace ret.

Stephen Checkoway提出无需使用ret指令就可以执行面向返回的编程[11]。 ret指令的替代项是updateload-branch序列，该序列1）更新充当面向返回程序的指令指针的寄存器的全局状态，2）使用更新后的状态从内存中加载下一条指令序列的地址 执行并3）跳转到加载的地址。 update-load-branch更新程序的全局状态，并将控制权转移到下一个指令序列。 在x86上，作者建议使用“ pop x; jmp \* x”，其中“ x”是任何通用寄存器。 Stephen提出的方法显示了使用jmp和call替换ret的可能性。

In call-oriented programming, the attacker drive the control flow by proposing special gadgets that all end in a call instruction rather than ret or jmp. The call instruction will push next instruction address into the stack, such that although using call-ending gadgets are proved to be feasible, unnecessary address in the top of the stack makes it hard to redirect control flow to the intended target. There is no actual proof of concept of call-oriented programming until AliAkbar Sadeghi proposed Pure-Call Oriented Programming(PCOP) in 2017 [12]. AliAkbar use a strong trampoline gadget, like “pop eax;popad;cld;call eax;” to remove undesired return addressed pushed into the stack by previous gadget’s call instructions.

在面向调用的编程中，攻击者通过提出特殊的gadget来驱动控制流，这些gadget都以call指令结尾，而不是ret或jmp。 调用指令会将下一个指令地址压入堆栈，因此尽管使用以call结束的dadget被证明是可行的，但堆栈顶部的不必要地址使将控制流重定向到预期目标变得很困难。 直到AliAkbar Sadeghi在2017年提出纯面向呼叫的编程（PCOP）之前，还没有关于面向呼叫的编程概念的实际证明[12]。 AliAkbar使用强大的蹦床gadget，例如“ pop eax; popad; cld; call eax;” 删除先前gadget的呼叫说明推入堆栈的不希望有的返回地址。

III. ROP DEFENSE MECHANISMS

三、ROP防御机制

Return-oriented programming has a good performance in bypassing W ⊕ X, and ROP attack has become a necessary step in an exploit. In order to reduce the harm of ROP attacks, there are some general methods proposed for ROP mitigation. In this section, we discuss the defense mechanisms on the frequency of gadget, randomization, and control flow integrity.

面向返回的编程在绕过W⊕X时具有良好的性能，并且ROP攻击已成为利用漏洞的必要步骤。 为了减少ROP攻击的危害，提出了一些缓解ROP的常规方法。 在本节中，我们将讨论gadget的频率，随机化和控制流完整性的防御机制。

1. Frequency of Gadget
2. Gadget的频率

In the process of return-oriented programming attack, the gadget is the basic unit. A gadget is picked up in existing code area and contains two or three instructions, so it is difficult to detect a single gadget. However, during attack occurs the program stream has a large number of code fragments ending with ret. Such that the frequency of the ret instruction in a stream becomes a way to detect whether a binary is under attack. A defense system will hook the ret instruction, 1) count the length of this code fragment; 2) count the length of contiguous code fragments ending with ret. The effect of this defense mechanism depends on the length of the gadget instruction and the number of consecutive times of gadgets defined. By changing the two variables, the sensitivity of detection can be improved or reduced. However, high sensitivity means high false positive while low sensitivity means ROP cannot be detected. Ping Chen et. al. proposed DROP, a ROP attack defense method based on this principle [4]. Frequency approach can protect program in binary level, but this method is not widely used in the operating system for security reasons. Once the values of gadget length are determined, gadgets that length is not in the scope cannot be detected, thus cannot count the length of contiguous gadget, which means a high false negative.

在面向返回的编程攻击过程中，gadget是基本单元。在现有代码区域中拾取一个gadget，其中包含两个或三个指令，因此很难检测到单个gadget。但是，在攻击发生期间，程序流具有大量以ret结尾的代码片段。这样，流中ret指令的频率就成为检测二进制文件是否受到攻击的一种方式。防御系统将钩住ret指令：1）计算此代码片段的长度； 2）计算以ret结尾的连续代码片段的长度。这种防御机制的效果取决于gadget指令的长度和定义的gadget的连续次数。通过更改两个变量，可以提高或降低检测的灵敏度。但是，高灵敏度意味着高假阳性，而低灵敏度意味着无法检测到ROP。Ping Chen等人提出了DROP，一种基于该原理的ROP攻击防御方法[4]。频率方法可以以二进制级别保护程序，但是出于安全原因，此方法未在操作系统中广泛使用。一旦确定了gadget的长度值，就无法检测到长度不在范围内的gadget，因此无法计算连续gadget的长度，这意味着假阴性率很高。

The side effect of frequency approach makes it hard to use in practice, but recent research found that the frequency approach can combine with control flow integrity method to achieve a better performance in gadgets detected.

频率方法的副作用使其难以在实践中使用，但是最近的研究发现，频率方法可以与控制流完整性方法结合使用，以在检测到的gadget中实现更好的性能。

1. Randomization
2. 随机化

The implementation of ROP depends on the use of existing instructions in memory so that an attacker needs to know the exact address of instruction he or she needs. Randomization is used to resist ROP attack. Randomization includes Address space layout randomization(ASLR) and instruction randomization. ASLR randomizes the base address of stack, heap, external libraries and others, such that the adversary cannot predict the address where these part would be loaded, and cannot get the address of the gadgets to layout the stack. ASLR reduces the risk of being attacked but cannot completely eliminate it. In practice, not all part of the binary is randomized, where the attacker can find useful gadgets to finish an attack. As for instructions randomization, Hiser et al. provided a method that analyzed binary offline to get the information of actual locations of each instruction and the target address of indirect instructions, after that the method generates a re-write rule for these indirect instructions [13]. All of the offline analysis results are inserted into a database. The binary was started by a virtual machine, the virtual machine looked up the next instruction used by the binary with rewrite rules and picked the instruction from the database. The limitation of Hiser’s method is that the location of external libraries is not included and the average overhead is 13-16% in performance as a result of virtual machine involved. Compare with other defense mechanisms, the randomization method has a minimal modification to binary.

ROP的实现取决于内存中现有指令的使用，因此攻击者需要知道他或她所需指令的确切地址。随机用于抵抗ROP攻击。随机化包括地址空间布局随机化（ASLR）和指令随机化。 ASLR将堆栈，堆，外部库和其他库的基地址随机化，以使对手无法预测将要装载这些部分的地址，并且无法获取用于布局堆栈的gadget的地址。 ASLR降低了被攻击的风险，但不能完全消除它。实际上，并非二进制文件的所有部分都是随机的，攻击者可以在其中找到有用的gadget来完成攻击。至于指令的随机化，Hiser等人提供了一种方法，可以离线分析二进制以获取每个指令的实际位置信息和间接指令的目标地址，然后该方法为这些间接指令生成重写规则[13]。所有离线分析结果都插入到数据库中。二进制文件由虚拟机启动，虚拟机使用重写规则查找二进制文件使用的下一条指令，并从数据库中选择了该指令。 Hiser方法的局限性在于不包括外部库的位置，并且由于涉及虚拟机，因此平均性能开销为13-16％。与其他防御机制相比，随机方法对二进制的修改最少。

Among all defense mechanisms, ASLR is the only method adopted by the operating system. What encouraged by ASLR is that defense mechanisms are better transparent to software developers.

在所有防御机制中，ASLR是操作系统采用的唯一方法。 ASLR鼓励的是，防御机制对软件开发人员来说更加透明。

1. Control-flow Integrity

C. 控制流完整性

In typical return-oriented programming, the execution of code fragments is picked either from incomplete calls of functions(intended gadgets) or from unaligned instructions(unintended gadgets), which can be seen as a control flow exception. Control-flow integrity is used to detect ROP attack. Depending on the acquirement of source code or binary, control-flow integrity is divided into the compiler-based approach and dynamic approach. The compiler-based approach modifies the code layout during compile time, adding code that can check the behavior of free branch instructions [14]. If the target of a free branch instruction is not the expected address, the defense system will launch an alarm or an exception. Some compiler-based method also tries to rewrite the binary to reduce the number of unintended gadgets. Onarlioglu et. al. proposed G-free, involving a random key to check the return address [3]. G-free encrypted the return address with a random key when a function was called. At the point of function returned, G-free decrypted return address with the same random key. Compared with the compiler-based approach, the dynamic approach can be used in the binary level. The method is inspired by every ret must be called by a call, which means the number of ret is the same as the number of call. Pappas et. al. proposed a dynamic approach to control flow integrity, which was called kBouncer [15]. Pappas made use of the last branch record(LBR) feature provided by the Inter(R) processors. When a system call launched, kBouncer checked whether every “ret was located after the corresponding call site of the call function, meanwhile, kBouncer counted the length of contiguous code fragment ending with ret as in frequency approach. Though dynamic approaches can fit both source code level and binary level, monitoring the process of binary means a significant and noticeable performance overhead.

在典型的面向返回的编程中，代码片段的执行是从函数的不完整调用（预期的gadget）或未对齐的指令（意外的gadget）中选择的，这可以看作是控制流异常。控制流完整性用于检测ROP攻击。根据源代码或二进制代码的获取，控制流完整性分为基于编译器的方法和动态方法。基于编译器的方法在编译期间修改了代码布局，添加了可以检查自由分支指令行为的代码[14]。如果自由分支指令的目标不是预期的地址，则防御系统将发出警报或异常。一些基于编译器的方法还尝试重写二进制文件，以减少意外的gadget的数量。 Onarlioglu等人提出了G-free，涉及使用随机密钥检查返回地址[3]。调用函数时，G-free使用随机密钥对返回地址进行加密。在函数的返回处，G-free使用相同的随机密钥解密返回地址。与基于编译器的方法相比，动态方法可以在二进制级别使用。该方法的灵感来自必须通过调用来调用的每个ret，这意味着ret的数量与调用的数量相同。Pappas等人提出了一种控制流完整性的动态方法，称为kBouncer [15]。 Pappas利用了Inter（R）处理器提供的最后一个分支记录（LBR）功能。当启动系统调用时，kBouncer会检查每个“ ret”是否位于调用函数的相应调用位置之后，与此同时，kBouncer会以频率方法计算以ret结尾的连续代码片段的长度。尽管动态方法既可以适合源代码级别，也可以适合二进制级别，但是监视二进制进程意味着巨大的性能开销。

Nowadays, Control-flow integrity was not adapt as a common security mechanism in operating system for the reason that overhead on performance is high. Whats worse is that recent research found CFI was not as safe as in theory. Carlin et. al. proposed using control-flow bending to bypass control-flow integrity, which means the lack of security cannot meet the actual needs [16].

如今，由于性能开销过高，控制流完整性无法适应操作系统中的常见安全机制。更糟糕的是，最近的研究发现CFI不如理论上的安全。Carlin等人提出使用控制流弯曲来绕过控制流完整性，这意味着缺乏安全性无法满足实际需求[16]。

IV. BENIGN USE OF ROP

五、ROP的良性使用

Return-oriented programming had always been considered as a well-known software exploit technique [2], [17]–[19]. However, recent research showed that ROP can be applied in benign uses like program steganography, code integrity verification and software watermarking.

面向返回编程一直被认为是一种众所周知的软件利用技术[2]，[17] – [19]。 但是，最近的研究表明，ROP可用于程序隐写术，代码完整性验证和软件水印等良性用途。

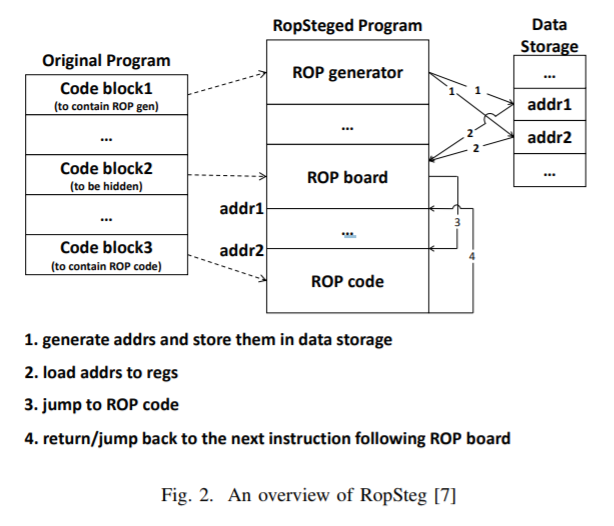
1. Program Steganography with Return-Oriented Programming
2. 面向返回编程的程序隐写术

Kangjie Lu et al. proposed RopSteg, a tool to use return-oriented programming in program steganography [7]. The aim of steganography is to hide certain instructions that are non-existent until program execution. RopSteg, as claimed by authors, is the first proposal of driving return-oriented programming from the “dark side”, using return-oriented programming in a non-attack application. The reasons for the combination of ROP and steganography are 1) most existing technique of steganography may violate W ⊕X or mandatory code signing security mechanisms and ROP is a common tool to bypass W ⊕ X in an attack; 2) a disassembler can never pick up the unintended gadgets in a binary, which can be seen as a way to hide message until program execution.

Kangjie Lu等人提出了RopSteg，一种在程序隐写术中使用面向返回的编程的工具[7]。 隐写术的目的是隐藏某些在程序执行之前不存在的指令。 正如作者所声称的，RopSteg是从“黑暗面”推动面向收益的编程的第一个建议，即在非攻击性应用程序中使用面向收益的编程。 ROP和隐秘术相结合的原因是：1）大多数现有的隐秘术可能会违反W⊕X或强制性代码签名安全机制，并且ROP是在攻击中绕过W⊕X的常用工具； 2）反汇编程序永远无法以二进制形式拾取意外的gadget，这可以看作是在程序执行之前隐藏消息的一种方式。

Figure 2 shows an overview of RopSteg. Code block 2(ending at addr 1) is the instruction sequences to be hidden. RopSteg finds or constructs unintended ROP gadgets(code block 3) to replace code block2, then uses ROP Board(locate on code block 2) to redirect control flow to ROP gadgets(starting at addr 2). After the completion of ROP gadgets, ROP Board converts the control flow to addr 1 to continue execution. To combat static analysis of finding values of addr1 and addr2, ROP generator is used to dynamically calculates these values and the results are stored in the stack, which has no conflict with existing defense mechanism ASLR. Finally, RopSteg will rewrite the binary to finish the steganography.

图2显示了RopSteg的概述。 代码块2（在地址1处结束）是要隐藏的指令序列。 RopSteg查找或构造意外的ROP gadget（代码块3）来替换代码块2，然后使用ROP Board（位于代码块2上）将控制流重定向到ROP gadget（从地址2开始）。 完成ROP gadget后，ROP板将控制流转换为地址1以继续执行。 为了对抗对addr1和addr2的查找值的静态分析，ROP生成器用于动态计算这些值，并将结果存储在堆栈中，与现有防御机制ASLR不冲突。 最后，RopSteg将重写二进制文件以完成隐写术。



The experiments taken by Lu showed that RopSteg can protect both secret algorithm and code fragment from static analysis. Besides, RopSteg achieves program steganography with a small overhead in program size and runtime, which means RopSteg has better performance on steganography

Lu进行的实验表明RopSteg可以保护秘密算法和代码片段免受静态分析。 此外，RopSteg实现了程序隐写，而程序大小和运行时开销却很小，这意味着RopSteg在隐写方面具有更好的性能。

The limitation of this method is the artificial constructed “flaw can be detected as an attack under some defense mechanism. RopSteg showed the possibility of adopting ROP in security use.

这种方法的局限性是人为构造的“缺陷可以在某种防御机制下被检测为攻击”。 RopSteg展示了在安全用途中采用ROP的可能性。

1. Code Integrity Verification using Return-Oriented Programming
2. 使用面向返回编程验证代码完整性

Dennis Andriesse et. al. proposed Parallax, another nonattack use of return-oriented programming on code integrity verification [8]. The idea of Parallax is that in a ROP attack, a single gadget error would result in failure of the attack. Parallax augments the code that needs to integrity verification with the gadgets, called overlapping gadgets. Parallax then uses these gadgets and other gadgets found in the binary or constructed by code rewrite to generate verification function. As a single change in code lead to the verification function failure, Parallax can identify whether the code is modified. In a word, Parallax converts the integrity of the code into the integrity of the ROP chain.

Dennis Andriesse等人提出了Parallax，这是在代码完整性验证中另一种无攻击的面向返回的编程方法[8]。 Parallax的想法是，在ROP攻击中，单个gadget错误将导致攻击失败。 Parallax使用称为重叠gadget的gadget来增强进行完整性验证所需的代码。 然后，Parallax会使用二进制中找到的或通过代码重写构造的这些gadget和其他gadget来生成验证功能。 由于单次更改代码都会导致验证功能失败，因此Parallax可以识别是否修改了代码。 总之，Parallax将代码的完整性转换为ROP链的完整性。

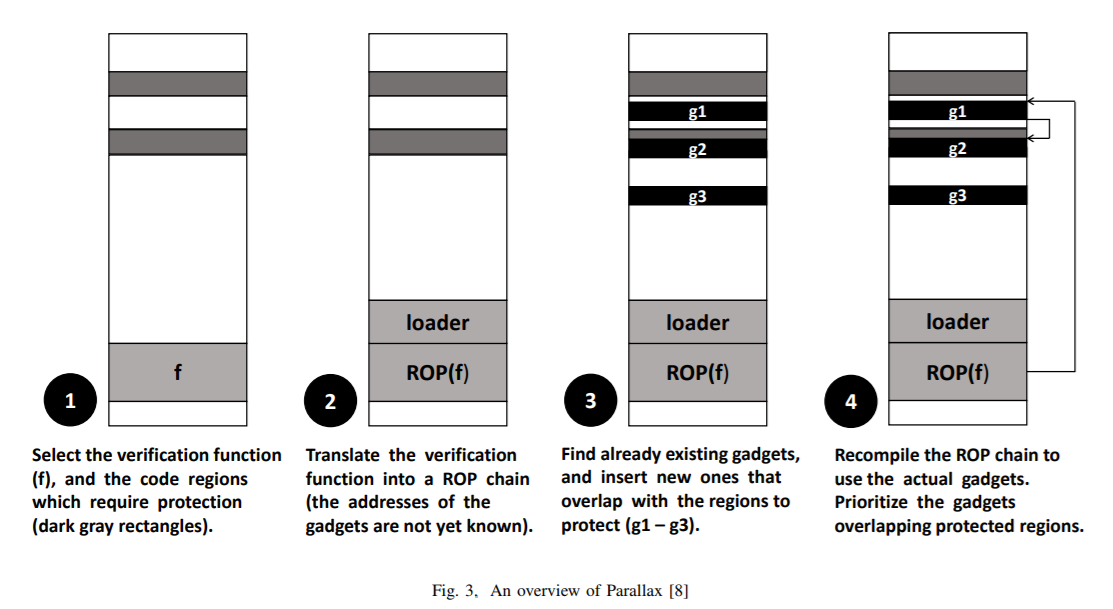


Figure 3 shows an overview of Parallax. To protect a binary, Parallax chooses one or more code fragments(functions) as verification code(step 1), then translates the verification function into ROP chains and inserts a loader routine to bootstrap the gadgets(step 2). Parallax will search the whole binary to generate existing gadgets set. Especially, Parallax will check whether gadgets in protected code can be rewritten as a gadget, if so, Parallax will rewrite instructions to create new gadgets, these gadgets are marked as overlapping gadgets. If the gadgets set is not Turing complete, Parallax will rewrite the binary to construct a Turing complete set (step3). Finally, Parallax creates a gadget mapping which categorizes the available gadgets and compiles verification code with ROP gadgets, during when overlapping gadgets are always preferred over non-overlapping gadgets (step 4).

图3显示了Parallax的概况。 为了保护二进制文件，Parallax选择一个或多个代码片段（功能）作为验证代码（步骤1），然后将验证功能转换为ROP链，并插入加载程序例程以引导小工具（步骤2）。 Parallax将搜索整个二进制文件以生成现有的小工具集。 特别是，Parallax将检查受保护代码中的小工具是否可以重写为小工具，如果可以，Parallax将重写指令以创建新的小工具，这些小工具被标记为重叠的小工具。 如果小工具集不是图灵完备，则Parallax将重写二进制文件以构建图灵完备（步骤3）。 最后，在始终优先使用重叠小工具而不是不重叠小工具的情况下，Parallax创建一个小工具映射，该映射对可用小工具进行分类并使用ROP小工具编译验证代码（第4步）。

The author claimed that Parallax is the first research using return-oriented programming techniques for tamper-proof, and Parallax can protect up to 90% of code bytes, including most control flow instructions, and has a performance overhead of under 4%. Besides, Parallax can use in binary level and can protect non-deterministic code regions (compared with oblivious hashing).

作者声称Parallax是第一个使用面向返回的编程技术进行防篡改的研究，并且Parallax可以保护多达90％的代码字节，包括大多数控制流指令，并且性能开销不到4％。 此外，Parallax可以在二进制级别使用，并且可以保护不确定的代码区域（与遗忘的哈希相比）。

Unfortunately, the use of intended ROP in Parallax is viewed as an attack by ROP defense technique.

不幸的是，Parallax中使用预期的ROP被视为ROP防御技术的攻击。

1. Software Watermarking using Return-Oriented Programming
2. 使用面向返回编程的软件水印

Previous software watermarking introduce special data structures and instruction patterns, which might cause suspicion [9]. Besides, code inserted in the watermarked program has independent control flow from the other parts of the program, so that an attacker can use dependency analysis to find the watermarking [20], [21]. In order to solve these problems, Haoyu Ma et al. took advantage of code reuse in return-oriented programming.

先前的软件水印引入了特殊的数据结构和指令模式，这可能会引起怀疑[9]。 此外，插入到带水印的程序中的代码具有与程序其他部分独立的控制流，因此攻击者可以使用依赖关系分析来找到带水印的内容[20]，[21]。 为了解决这些问题，Haoyu Ma等人在面向返回的编程中利用了代码重用。

The implementations of this watermarking can be divided into four steps. 1) Finding gadgets useful in watermark generation. The author tries to find gadgets in shared libraries instead of code area for the reason that using gadgets in code area may look suspicious for short programs. 2) Constructing payload distributive and dynamic. The design splits the watermarking payload into dierent functions of the program, and these functions are called “carriers”. A processor tracer will be involved to find functions that fit to be the “carriers”. With a number of carriers, the author manages to embed just a small piece of code in each carrier that controls only a few gadgets, largely reducing the suspicion raised since the inserted code is almost negligible compared to the size of the original carriers. 3) Chaining pieces of “carriers” with special gadgets. In order to chain distributed payload pieces, attach stack-shifting gadgets at the end of them so that each of the segments is responsible to relocate the stack frame correctly to the exact memory address of the next one. 4) Triggering ROP with function pointer overwriting.

该水印的实现可以分为四个步骤。 1）查找对水印生成有用的小工具。作者尝试在共享库而不是代码区域中找到小工具，原因是对于短程序而言，在代码区域中使用小工具可能令人怀疑。 2）分布式和动态的构建payload。该设计将水印payload拆分为程序的不同功能，这些功能称为“载体”。处理器跟踪器将被用来查找适合作为“载体”的功能。对于许多载体，作者设法在每个载体中仅嵌入一小段代码，而这些代码只能控制几个小工具，从而大大减少了人们的怀疑，因为插入的代码与原始载体的大小相比几乎可以忽略不计。 3）用特殊配件将“载体”的链条链接起来。为了链接分散的payload块，请在堆栈末尾附加堆栈移动小工具，以便每个段负责将堆栈帧正确地重新定位到下一个的确切内存地址。 4）使用功能指针覆盖触发ROP。

The experiments showed that this approach is able to make watermarking semantics untraceable by powerful static analysis tools, and the code reuse technique creates no suspicious data structures as in existing work. Another benefit is that the author used ROP trigger inside the watermarked program as to active ROP execution and extract the watermark, which avoids the watermarking extractor from leaking to the attacker.

实验表明，这种方法能够使强大功能的静态分析工具无法跟踪水印语义，并且代码重用技术不会像现有工作中那样创建可疑数据结构。 另一个好处是，作者使用了带水印的程序内部的ROP触发器来激活ROP执行并提取水印，从而避免了水印提取器泄漏给攻击者。

Similar to another approach, the ROP defense technique might regard the watermarking method as an attack.

类似于另一种方法，ROP防御技术可以将加水印方法视为攻击。

We show that, although initially proposed for malicious purposes, ROP can be applied in benign uses. ROP uses code fragments already existing in a program can lead to a less overhead in program size, besides, ROP has a strong resistance to static analysis. Although could be detected by defense technique, the benign uses of ROP can adopt novel ROP bypass technique to get rid of it.

我们表明，尽管最初出于恶意目的提出了ROP，但可以将其应用于良性用途。 ROP使用程序中已经存在的代码片段可以减少程序大小的开销，此外，ROP对静态分析具有强大的抵抗力。 尽管可以通过防御技术检测到，但是ROP的良性使用可以采用新颖的ROP绕过技术来摆脱它。

V. THE FUTURE OF ROP

ROP的未来研究方向

In practice, ROP attack is often combined with information leaks. An arbitrary memory read caused by information leak reduce the difficulty of ROP attack. In 2017, Ben Gras proposed a way to bypass ASLR wich MMU [22]. Therefore, how to prevent the leakage of memory information from software and hardware level is a recent research direction.

实际上，ROP攻击通常与信息泄漏结合在一起。由信息泄漏导致的任意内存读取减少了ROP攻击的难度。 2017年，Ben Gras提出了一种绕过MMU的ASLR的方法[22]。因此，如何防止存储器信息从软件和硬件层面泄漏是最近的研究方向。

Though a large number of ROP defense mechanisms are proposed in recent years, only ASLR is adopted as a common security mechanism in modern operating systems. The existing defense measures are merely theoretical and defective, and most of them are not transparent to software developers, which makes the defense method not be applied to the actual environment. Therefore, how to make the defense measures applied in practice is also a research direction.

尽管近年来提出了许多ROP防御机制，但在现代操作系统中仅采用ASLR作为常见的安全机制。现有的防御措施仅仅是理论上的和有缺陷的，而且大多数措施对软件开发人员都不透明，这使得防御方法无法应用于实际环境。因此，如何使防卫措施付诸实践也是研究方向。

Return-oriented programming can also be used for security in addition to attacks. A risk controlled “vulnerability” is set in a normal program, which allows the program to hide logic. We showed the benign use in program steganography, Code integrity verification and software watermarking. Compared with previous approaches, return-oriented programming have less overhead in program size and a better performance on static analysis. Therefore, how to use return-oriented programming execution in security an exploration work.

除了攻击之外，面向返回的编程也可以用于安全性。在常规程序中设置了风险控制的“漏洞”，这使程序可以隐藏逻辑。我们展示了在程序隐写术，代码完整性验证和软件水印方面的有益用途。与以前的方法相比，面向返回的编程在程序大小方面具有较少的开销，并且在静态分析方面具有更好的性能。因此，如何在安全性方面探索如何使用面向返回的编程执行工作。

VI. CONCLUSION

结论

Return-oriented programming was initially a type of software attack that exploited the buffer overow vulnerability. In the last ten years, ROP has carried out an in-depth study on attack and protection. In this paper, we provided the concepts of return-oriented programming attack and a brief classication of the defense mechanisms. This paper also provided returnoriented programming in benign use.

面向返回的编程最初是一种利用缓冲区溢出漏洞的软件攻击。 在过去的十年中，ROP对攻击和保护进行了深入的研究。 在本文中，我们提供了面向返回的编程攻击的概念以及防御机制的简要描述。 本文还提供了良性使用的面向返回的编程。

Return-oriented programing in computer system architecture and software environment under the current temporarily is unable to eliminate, offensive. Therefore, whether it is to take the initiative in the network security game, or for the purpose of protection, it is necessary to further study the binary code reuse technology.

在当前的计算机系统架构和软件环境下，面向返回的编程暂时无法消除。 因此，无论是主动在网络安全游戏中，还是出于保护目的，都有必要进一步研究二进制代码重用技术。

ACKNOWLEDGMENT

致谢

This work is supported by NSFCNo.61472439, National Natural Science Foundation of China under Grant.

这项工作得到了国家自然科学基金的支持，国家自然科学基金资助编号为61472439。

REFERENCES

参考文献

[1] Y. C. Gao, A. M. Zhou, and L. Liu, “Data-execution prevention technology in windows system,” Information Security & Communications Privacy, 2013.

[2] H. Shacham, “The geometry of innocent flesh on the bone,” in ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, Usa, October, 2007, pp. 552–561.

[3] K. Onarlioglu, L. Bilge, A. Lanzi, D. Balzarotti, and E. Kirda, “G-free: defeating return-oriented programming through gadget-less binaries,” in Computer Security Applications Conference, 2010, pp. 49–58.

[4] P. Chen, H. Xiao, X. Shen, X. Yin, B. Mao, and L. Xie, “Drop: Detecting return-oriented programming malicious code,” in International Conference on Information Systems Security, 2009, pp. 163–177.

[5] Y. Cheng, Z. Zhou, Y. Miao, X. Ding, H. Deng et al., “Ropecker: A generic and practical approach for defending against rop attack,” 2014.

[6] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, “On the effectiveness of address-space randomization,” in Proceedings of the 11th ACM conference on Computer and communications security. ACM, 2004, pp. 298–307.

[7] S. Xiong, S. Xiong, and D. Gao, “Ropsteg: program steganography with return oriented programming,” in ACM Conference on Data and Application Security and Privacy, 2014, pp. 265–272.

[8] D. Andriesse, H. Bos, and A. Slowinska, “Parallax: Implicit code integrity verification using return-oriented programming,” in Ieee/ifip International Conference on Dependable Systems and Networks, 2015, pp. 125–135.

[9] H. Ma, K. Lu, X. Ma, H. Zhang, C. Jia, and D. Gao, “Software watermarking using return-oriented programming,” in ACM Symposium on Information, Computer and Communications Security, 2015, pp. 369– 380.

[10] E. J. Schwartz, T. Avgerinos, and D. Brumley, “Q: exploit hardening made easy,” in Usenix Conference on Security, 2011, pp. 25–25.

[11] S. Checkoway, L. Davi, A. Dmitrienko, A. R. Sadeghi, H. Shacham, and M. Winandy, “Return-oriented programming without returns,” in ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, Usa, October, 2010, pp. 559–572.

[12] A. A. Sadeghi, S. Niksefat, and M. Rostamipour, “Pure-call oriented programming (pcop): chaining the gadgets using call instructions,” Journal of Computer Virology & Hacking Techniques, pp. 1–18, 2017.

[13] J. Hiser, A. Nguyen-Tuong, M. Co, M. Hall, and J. W. Davidson, “Ilr: Where’d my gadgets go?” in Security and Privacy, 2012, pp. 571–585.

[14] Y. Ruan, S. Kalyanasundaram, and X. Zou, “Survey of return-oriented programming defense mechanisms,” Security & Communication Networks, vol. 9, no. 10, pp. 1247–1265, 2016.

[15] V. Pappas, M. Polychronakis, and A. D. Keromytis, “Transparent rop exploit mitigation using indirect branch tracing,” in Usenix Conference on Security, 2013, pp. 447–462.

[16] N. Carlini, A. Barresi, M. Payer, D. Wagner, and T. R. Gross, “Controlflow bending: on the effectiveness of control-flow integrity,” in Usenix Conference on Security Symposium, 2015, pp. 161–176.

[17] E. Buchanan, R. Roemer, H. Shacham, and S. Savage, “When good instructions go bad: generalizing return-oriented programming to risc,” in ACM Conference on Computer and Communications Security, 2008, pp. 27–38.

[18] C. Castelluccia, “Code injection attacks on harvard-architecture devices,” in ACM Conference on Computer and Communications Security, 2008, pp. 15–26.

[19] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, “Return-oriented programming: Systems, languages, and applications,” Acm Transactions on Information & System Security, vol. 15, no. 1, p. 2, 2012.

[20] C. Collberg, C. Thomborson, and G. M. Townsend, “Dynamic graphbased software watermarking,” 2004.

[21] J. Nagra and C. Thomborson, “Threading software watermarks,” in International Conference on Information Hiding, 2004, pp. 208–223.

[22] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida, “Aslr on the line: Practical cache attacks on the mmu,” NDSS (Feb. 2017), 2017.