利用返回地址保护机制防御代码复用类攻击

陈林博 江建慧 张丹青
(同济大学软件学院 上海 201804)

摘 要 尽管现有多种防御方法和技术,但是针对软件系统和网络的攻击仍然是难以防范的威胁。在引入只读/写和地址空间随机化排列方法后,现代操作系统能有效地应对恶意代码注入类型的攻击。但是攻击者可以利用程序中已经存在的代码,将其组装成具有图灵完全计算功能的连续的代码块,用以绕过已有的防御机制。针对代码复用类攻击防御方法的局限性,提出了一种利用返回地址实时保护机制的防御方法,以有效防御代码复用类攻击,特别是ROP攻击。在程序运行时,通过对其栈中返回地址值的加密保护和实时检测,防止所有的以0xC3字符(即ret指令)结尾的短序列代码段的连续执行。该方法不需要源代码和调试信息,能完全防御ROP攻击,并且其性能开销也具有明显的优势。

关键词 代码复用类攻击,ROP攻击,返回地址保护,二进制代码动态翻译
中图法分类号 TP303.08 **文献标识码** A

Prevention of Code Reuse Attacks through Return Address Protection

CHEN Lin-bo JIANG Jian-hui ZHANG Dan-qing
(School of Software Engineering, Tongji University, Shanghai 201804, China)

Abstract Despite the numerous prevention and protection techniques that have been developed, the exploitation of memory corruption vulnerabilities still represents a serious threat to the security of software systems and networks. Because of the adoption of the write or execute only policy (W ⊕ X) and address space layout randomization (ASLR), modern operate systems have been strengthened against code injection attacks. However, attackers have responded by employing code reuse attacks, in which software vulnerability is exploited to weave control flow through existing code base. Solutions targeting different aspects of the attack itself have got some success, but none of them can be a silver bullet. Under this situation, a novel defense technique was presented in order to prevent code reuse attacks, especially return-oriented programming (ROP) attacks. This new defense technique, which was benefit from the protection of return address, could dynamically prevent the execution of gadgets ending with 0xc3. Without requiring access to side information such as source code or debugging information, this defense technique could prevent ROP attacks with low performance overhead.

Keywords Code reuse attacks, Return-oriented programming attacks, Return address protection, Binary dynamic translation

1 引言

代码复用类攻击是指攻击者将程序中已存在的代码组装成具有连续操作的恶意代码,利用漏洞(如缓冲区溢出漏洞、字符串格式化漏洞等)将程序控制流转移到恶意代码处,以完成攻击者的攻击意图。如返回库攻击(return into libc, RILC)[1,2]是在篡改程序栈中控制流数据后,将控制流指向已存在的标准库函数。攻击者可通过此种攻击手段来调用system()函数生成新进程,或者调用mprotect()函数创建可写可执行的内存区域用以绕过W ⊕ X防御方法。此类攻击方法被改进后可连续调用程序中代码段和标准库中一系列的函数,完成更复杂的功能[2]。RILC攻击所利用的代码段颗粒度大,可供攻击选用的函数数量少,并且依赖于标准库中几种关键函数,因此不方便组装。攻击者更倾向于选择具有图灵完全运算能力的面向返回的编程攻击(Return Oriented Programming, ROP)。

面向返回的编程攻击(Return Oriented Programming, ROP)[3]将若干以ret结尾的短序列代码碎片(gadget)组装成一系列可用代码段,同时组装好的代码段具有图灵完全运算能力。攻击者将含有若干个指令地址(即指向代码碎片gadgets的地址)的数据注入到栈或者内存其他可写位置,并将控制流转移到这些指令所指向的地址处,通过执行连续的

到稿日期:2013-01-24 返修日期:2013-03-28
陈林博(1984—),男,博士生,主要研究方向为信息安全、入侵检测与防御等,E-mail:08chenlinbo@tongji.edu.cn;江建慧(1964—),男,博士,教授,博士生导师,主要研究方向为可信系统与网络、软件可靠性工程、VLSI/SoC测试与容错等;张丹青(1984—),女,博士生,主要研究方向为软件可靠性工程、容错计算等。

短序列代码段来完成攻击者的意图。由于程序中存在大量的以 0xC3 字符(即 ret 指令)结尾的短序列代码碎片,因此仅仅通过去除标准库中的若干个函数(如 system()函数),不能有效地防御代码复用类攻击。同时 ROP 攻击已经在真实攻击中被用来绕过 Windows 系统的 W⊕X 防御方法[4],即数据执行保护技术(Data Execution Prevention,DEP)。

现有的针对特定漏洞攻击(如缓冲区溢出漏洞、格式化字符串漏洞等)的防御方法[5-9]仅能保护部分控制流相关的数据,但不能保证其他控制流相关的数据不被篡改。ASLR 和控制流一致性检测[10-13]等通用的防御方法则在有效性或者性能开销上存有缺陷。而针对 ROP 攻击特征所提出的在编译器上实现[14,15]的防御方法和基于二级制代码重构的防御方法[16-18]需要程序提供其他信息,如源代码或者调试信息;同时这类方法均不能完全消除 ret 指令或含有 0xC3 字符的代码段。通过二进制代码动态翻译工具实现的防御方法,如 DROP[19]、DynIMA[20]和 ROPdefender[21]等,尽管不需要源代码或者调试信息,但是存在防御方法容易被攻击者绕过或性能开销较大的缺陷。

综上分析,本文提出一种针对 ROP 攻击的有效防御方法。在程序运行时,加密栈中所存储的返回地址值,实时监测 ret 指令并在指令执行前解密返回地址值,以此检测返回地址值是否被篡改,防止所有的以 0xC3 字符(即 ret 指令)结尾的短序列代码段的连续执行。本文所述的方法不需要源代码和调试信息,也不需要在运行过程设置容易被攻击者篡改的影子栈。对比同样采用二进制代码动态翻译工具实现的 DROP、ROPdenfender,本文所述的动态防御方法在完全防御 ROP 攻击的基础上,其性能开销也具有明显的优势。

本文第 2 节阐述 ROP 攻击并且分析了相关的防御方法;在此基础上,在第 3 节提出了利用返回地址保护机制防御代码复用类攻击的方法,并分析了可能造成误检率的几种异常情况,提出了相应的对策;利用二进制代码动态翻译工具 PIN 实现了本文所述的方法,并在第 4 节中给出了有效性验证和性能评估;最后总结全文。

2 ROP 攻击及其相应防御方法

2.1 ROP 攻击

由于以 x86 为典型代表的 CISC 指令集过于密集,任意组合的字符串均有极大概率被解释为有效的指令,因此在程序的代码中,任意一条以 0xC3 结尾的指令都可以被解释成含有 ret 指令的有效代码段。如图 1 所示,若从原地址后两个字节(即从 0x59 处)解释指令,则指令的语义将会发生变化,成为一段以 ret 指令结尾的代码段。可将这种代码段称为无意代码段(unintended code),在 ROP 攻击中大多数 gadget 都是由此类无意代码段所组成的。

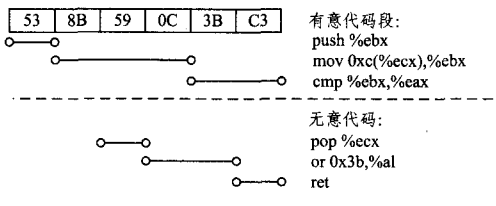

图 1 无意代码(unintended code)

由于不同的恶意代码段 gadget 存在于进程中不同的位置,且各个代码段是由 ret 指令连接,因此当程序执行恶意代码时,必须要通过各个代码段末尾的 ret 指令的执行将控制流转移到下一个代码段处,以完成有效的功能。一次典型的利用栈缓冲区溢出漏洞实现的 ROP 攻击如图 2 所示。

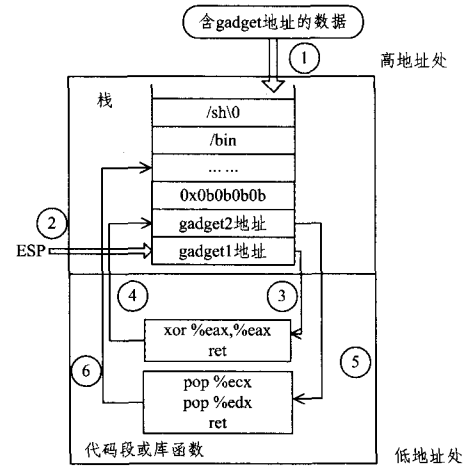

图 2 利用栈缓冲区溢出漏洞实现 ROP 攻击的步骤示意图

在图 2 中,攻击者首先将精心设计的含有 gadget 地址的数据输入到栈中,如步骤 1 所示。输入的数据覆盖原来的返回地址值及其相邻的栈空间,攻击者通过栈缓冲区溢出漏洞将栈指针 ESP 重定位至 gadget1,如步骤 2 所示。当函数正常返回后,EIP 寄存器的值将被 gadget1 地址覆盖,程序执行 gadget1 地址中所指向的指令,如步骤 3 所示。步骤 4、步骤 5 和步骤 6 代表程序将连续执行攻击所指定的恶意代码。从图 2 中所描述的攻击步骤可知,在实现 ROP 攻击时,需要控制栈指针 ESP,使其指向攻击者输入的恶意数据,再通过各个 gadget 中的 ret 指令的执行,保证所有的恶意代码段的连续执行。

2.2 相关的防御方法

实现代码复用类攻击,攻击者首先要利用程序中的内存漏洞来实现控制流的非法转移。因此已有的针对特定类型漏洞的防御方法,可以防御利用特定类型的漏洞(如栈缓冲区溢出漏洞)实现代码复用类攻击。StackGuard[5]和 ProPolice[6]是在程序编译时,在栈中插入 canary 保护字,实现对栈中控制流相关数据(如返回地址值、栈基指针值等)的保护,以检测针对栈缓冲区溢出的漏洞攻击。PointGuard[7]通过对内存中所存储的指针值加密,防止攻击者利用指针篡改内存中影响控制流的数据。StackShield[8]和 StackGhost[9]则建立影子栈存储返回地址值,并在函数返回时检测栈中返回地址值是否变化,以此检测是否受到攻击。

但是此类防御方法只能防御针对特定漏洞的攻击,面对层出不穷的新型漏洞,方法较为单一。例如攻击者可以通过其他的攻击手段,如格式化字符串攻击,篡改其它影响控制流的数据(如二进制文件中 GOT 节区和 dtors 节区中的数据容易被篡改[29]),达到控制流非法转移的目的,实现代码复用类攻击。

2.2.1 通用的防御方法

从攻击者角度分析,完成一次攻击首先需要了解进程空间的分布情况。进程空间地址随机化的方法(ASLR)[28]通过将进程地址中各段的基地址随机分布,即在地址空间中随机排列代码段、数据段、堆、栈和程序加载时映射到内存中的程

序段，来增加攻击者猜测地址的难度。作为针对攻击的通用防御方法，该类方法可以有效防御代码复用类攻击，但是ASLR方法也面临随机量过小、随机化对象的覆盖率不高等问题[31]，容易被攻击者破解。

ILR方法[17]通过对二级制代码的重写，将内存空间中的指令地址重新排列，以此迷惑攻击者；并在虚拟机上运行重构后的二进制文件，引导程序正常执行随机分布后的指令，保证程序的控制流不发生转移。但是ILR方法不能完全重排所有的指令，在未被重排的指令中仍然存在可被利用的gadget。

由于攻击最本质的特征是控制流的异常转移，因此可以利用控制流的一致性防御攻击。控制流一致性(CFI)[10]是指程序在运行期间，每次跳转的目的地址都应符合程序控制流图(CFG)，因此对控制流的检测能全面有效地检测各种类型的代码复用攻击。类似的针对控制流一致性的检测方法有Program Shepherding[11]、CFL[12]和CFLC[13]等。尽管该类方法能有效并全面检测代码复用类攻击，但是该类检测方法普遍都存在性能开销过大及误检率过高等问题。

2.2.2 *针对ROP攻击的防御方法*

除了上述地址混淆和控制流一致性检测等通用防御方法之外，目前针对ROP攻击特性所提出的防御方法可以分为以下几种。

1)代码混淆

由于ROP攻击的特征是以ret指令作为链接，将多个代码段组装使用，因此Li[14]在程序编译时用其他类型指令替换ret类指令，并在二进制代码中用同等语义且不含0xc3字节的指令串替换含有0xc3字节的指令，以达到消除ret指令的目的，防止ROP攻击。类似的方法还有G-free[15]，它不仅在编译时消除ret指令，而且将程序中间接跳转严格控制在同一函数内，以防止攻击者利用其他类型的间接跳转指令实现攻击。此类基于编译器的防御方法虽能消除一部分ret指令的隐患，但需要对程序重新编译。并且程序中的部分库函数不适用此类编译器，因此不能全面有效地防御ROP攻击，漏检率较高。

IPR[16]则是在保证代码长度不变且语义不变的情况下，利用代码替换、寄存器替换、不相关指令重排和寄存器重命名等方式，对二进制代码进行重写，以消除程序中可能存在的恶意代码。尽管有大约77%的gadget可以通过IPR消除或者部分消除，但是在大型程序中本身就存在数以万计的gadget，77%的覆盖率显然不够。

2)控制流相关数据的保护与检测

ROP攻击所采用的gadget的长度小，且其ret指令执行的频率高。DROP[19]和DynIMA[20]通过对程序中的二进制代码动态插装，实时检测ret指令的调用频率以防御ROP攻击。但是攻击者可以通过增加指令的条数绕过此类防御方法，其性能开销较大，如DROP的平均性能开销高达5x。

ROPdefender[21]则是通过二进制代码动态翻译工具在函数被调用时插入指令，建立影子栈用以保存函数的返回地址值，并在函数返回时插入指令，检测当前返回地址值是否有效。ROPdefender相比DROP在性能上有所提升，但其平均性能开销仍然在2x以上。

3 返回地址加密保护的防御方法

本文假设目标系统受W⊕X防御方法保护，且攻击者具备修改进程中可写内存的能力，攻击者可以通过各种漏洞(如缓冲区溢出、格式化字符串等)篡改进程中的控制流数据(如返回地址、GOT、指针函数地址等)，从而将程序的控制流转移到程序中的任意代码处，实施ROP攻击。

3.1 传统的返回地址保护方法的缺陷

在ROP攻击中，攻击者需要精心设计包含有各个代码段的返回地址值的数据，将其注入到被攻击程序的内存空间中，使得ROP攻击能顺利执行。因此根据ROP攻击这一特点，利用传统的对栈中返回地址值保护的方法(如Stackguard、Propolice等方法)可以避免栈中返回地址值被篡改，避免图2中步骤1和步骤2的顺利执行。但是攻击者可以尝试利用其它类型漏洞(如堆缓冲区溢出漏洞、格式化字符串漏洞等)篡改函数指针或者GOT的入口值等其他控制流相关的数据，用以将控制流转移到gadget1的指令中，并利用下列两种类型的指令实现ESP指针的重定位：

```
xchg %esp,reg;ret;
mov reg,%esp;ret;
```

第一种类型的指令是将任意寄存器reg的值与%esp替换，第二种类型的指令是将任意寄存器reg的值直接赋给%esp。因此通过控制任意寄存器的值，如%eax，再执行任意一条指令即可修改ESP指针值，而这两种类型的指令大量地存在于X86结构中[27]。

另一种返回地址保护的方法是在函数返回时的epilogue中，即在ret指令前插入安全检测指令(如RAD[30]、StackShield[8]等)，用以检测或恢复利用有意代码段(intended code)实现攻击时被篡改的返回地址值。但是此类防御方法的本质是加固进程中的有意代码段，所设计的防御机制不能加固无意代码段中的返回指令。因此攻击者完全可以构造全部使用无意代码段组装的gadget序列指令来绕过这类防御方法。

3.2 返回地址实时保护的防御方法

针对ROP攻击的特点，即需要连接指令ret的执行将各个gadget组装并连续运行，结合前述传统的栈中返回地址值保护方法的缺陷，本文采用一种利用返回地址值实时保护的方法，来检测ROP攻击。

与传统的对返回地址值加密的方法不同，本文所述方法在函数被调用时，即在执行call指令前，用模拟的call操作替换即将执行的call指令。模拟的call操作将call指令后的下一条指令地址(即函数返回地址值EIP)加密并将其值(即encoded(EIP))存入栈中，同时将栈指针ESP减4。模拟的call操作在执行完后，栈中的返回地址值已被加密，程序也将执行被调用函数。

当函数返回时，即在执行ret指令前，用模拟的ret操作替换即将执行的ret指令。模拟的ret操作将栈中返回地址值取出，将解密后的值(即decoded(encoded(EIP)))传给EIP寄存器，同时将栈指针ESP加4。如果此时栈中的返回地址值未经篡改，则decoded(encoded(EIP))==EIP，函数将正常返回。假设在32位系统中，在未获得加密密钥的情况下，攻击者需要猜测加密密钥，能成功构造指向攻击代码的加密地址的概率为$P=1/2^{32}$。因此在$1-P$的大概率下，本文所述的保护机制将能有效检测出ROP攻击，其检测流程如图3所示。

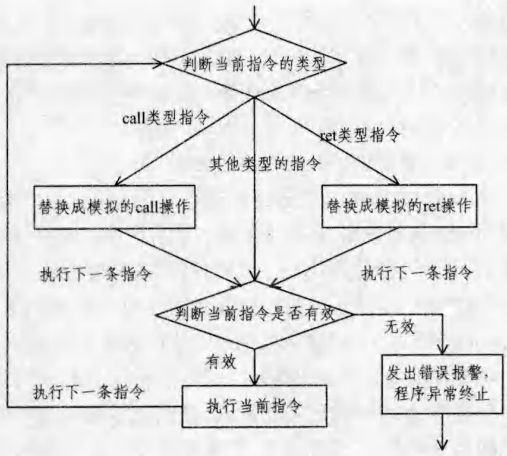

图 3　检测流程图

由于采用了插桩 instrumentation 技术，所有在 CPU 上执行的 ret 指令，包括无意代码中的 ret 指令，都会被实时检测。因此当攻击者实施 ROP 攻击时，假设栈中的返回地址值被攻击者篡改并指向了恶意代码处。随着程序的继续运行，即将运行的属于恶意代码中的 ret 指令将被检测并被模拟的 ret 操作代替执行。由于无法得知原程序对返回地址值加密的密钥，使得攻击者不可能设置正确地址值以覆盖栈中原返回地址值。因此在栈中返回地址值解密后，程序的控制流将被转移至未知位置，而非攻击者指定的 gadget 位置，如图 4 所示。在图 4 中，攻击者可以成功实施步骤 1 和步骤 2，然而在函数返回时，decoded(addr(gadget1))的值将被传递给 EIP 寄存器，而非 addr(gadget1)。最终控制流不能转移到 gadget1 处，攻击者所设计的连续代码段将不会被执行，攻击失效，如步骤 3 所示。

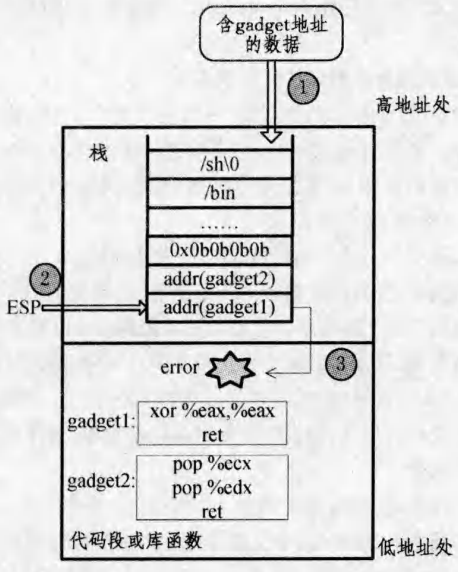

图 4　在受保护的程序中实施 ROP 攻击

3.3　返回地址实时保护方法的实现

本文所述防御方法中采用的加解密算法可采用简单的异或运算，将返回地址值与程序启动时生成的任意随机数进行异或运算。由于不需要改变栈中存储的其他数据，因此在实现中不需要修改原程序中的代码。

以图 4 中所举的 ROP 攻击为例，当程序从被调用函数返回时，原栈中的返回地址值已经被篡改，假设其值被篡改为 addr(gadget1)，且指向 gadget1 代码段。若程序当前加密的密钥是 K，则在 ret 指令即将执行前，栈中的返回地址值 addr(gadget1)将被重新计算并被送入 EIP 寄存器，此时 EIP 寄存器的值为 addr(gadget1)^K。因此，当 ret 指令执行后，控制流将转移到地址 addr(gadget1)^K 处，而并非攻击者所指定的位置 addr(gadget1)，因此攻击失效。而大部分情况下，地址 addr(gadget1)^K 都会指向一个未定义的虚拟存储器区域，最终会引起段故障，进程终止。

为了实现上述防御方法，本文在二进制代码翻译工具 PIN[22] 框架上实现了返回地址值实时保护方法。PIN 在处理器运行指令前，能实时检测即将运行的指令。在被插桩指令运行前可执行新生成的代码，并在代码运行完后执行程序中原有的指令。在 PIN 上实现的基于返回地址值加密检测工具的框架如图 5 所示。

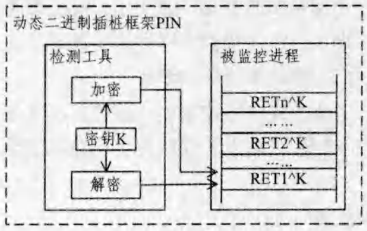

图 5　返回地址值保护的实现框架

图 5 中所述的密钥 K 是在 PIN 上启动程序时随机生成的数据，且在被监控程序运行期间均有效。返回地址值 RET1、RET2 和 RETn 与 K 异或加密后存储在栈中。为了实现框架中所示的加密和解密，这里使用 PIN 中 API 所提供的函数 Ins_IsCall(INS ins)和 Ins_IsRet(INS ins)来判断即将运行的指令是 call 指令或 ret 指令。在检测出指令后，一种直接的加解密方法是将栈中的值取出，加密或解密后再存入栈中，这种方法在执行 call 和 ret 指令时需要两次访问栈。而本文采用模拟的 call 和 ret 操作代替原程序中的 call 和 ret 指令，这种方法可以避免多次访问栈中数据，有效减少本文所述方法的性能开销。

3.4　几种异常情况的处理

程序在正常执行时，进入被调用函数前都会执行 call 指令，而从被调用函数返回时执行 ret 指令。但仍然会出现特殊情况，程序可能不经过 call 类指令而直接进入被调用函数，最后通过 ret 指令返回。此时，由于将被访问的返回地址值没有通过 call 操作加密后入栈，因此在执行 ret 操作中的解密后，其返回地址值发生错误，这将会到导致程序异常。

其中一种特殊情况是 Unix 信号处理。在 Unix 类系统中接收到信号后，如果进程启动信号处理程序处理该进程，则控制流将会转移到信号处理函数中，且该次转移不需要执行 call 指令，而返回地址值仍被压入栈中。当信号处理程序返回时，将执行 return 语句。而此时因为没有执行 call 指令，栈中的返回地址值未被加密。在执行 ret 操作后会出现错误，控制流将被转移到未知地址处，引起程序异常。为了避免此类异常情况，可以在实现时加入对信号的检测，当接收到信号并执行信号处理程序时，将栈中返回地址值加密。加密后的返回地址值在从信号处理函数返回时会经过解密，从而得到正确的返回地址值。

另一种可能会引发指令加解密异常的情况是延迟绑定，延迟绑定是将被调用函数的地址绑定推迟到第一次调用该函数时。在 Unix 类系统，如 Ubuntu 系统中，延迟绑定是通过动态链接库 linux-ld. so 中的两个函数_dl_rtld_di_serinfo 和_dl_make_stackexecutable 实现。在_dl_rtld_di_serinfo 获取被调用函数地址后，控制流通过跳转指令而非 call 指令转移到_dl_make_stackexecutable 函数中，而_dl_make_stackexecutable 函数将通过 ret 指令返回到被调用函数。由于_dl_rtld_di_serinfo 函数执行后所获得的被调用函数的地址存储在寄存器%eax 中，因此为避免此类引发加解密异常的情况，可在_dl_rtld_di_serinfo 函数执行后，即执行跳转到_dl_make_stackexecutable 函数的语句前，对寄存器%eax 的值加密。

4 有效性验证和性能评估

4.1 有效性验证

在实际的攻击中，ROP 攻击在大部分情况下是被用于在进程的地址空间中分配既可用于写也可用于执行的空间，以绕过系统中的 W⊕X 防御。为验证本文所述防御方法的有效性，在 Dell 微机(3GHz core2 CPU、2GB 内存)上运行 win7 SP1 系统，利用 PIN 实现了本文所述工具，建立了相应的实验平台。

实际的攻击对象包括 Adobe Reader v9. 3. 4[23]、Integard Pro v2. 2. 0[24]和 Mplayer Lite r33064[25]等程序，针对这 3 种程序的攻击均是从程序调用的库中获得代码块，并能在受 W⊕X 保护的系统中成功攻击。以针对 Adobe Reader 的攻击为例，攻击者利用渲染 TIFF 图像格式的 libtiff 库中的整形数溢出漏洞，在内存中分配新的区间并将其设置为可写并可执行，以绕过 W⊕X 保护。然后通过 memcpy 函数将隐藏在 PDF 文件中的恶意代码拷贝至该区间，并将控制流重定向到恶意代码处，执行恶意代码并为攻击者创建远程 shell。

在实验中，被攻击的程序运行在基于 PIN 开发的防御工具上。以 Adobe Reader 的攻击为例，攻击者发送的恶意 PDF 文件将在本文所述的防御工具上运行 Adobe Reader 程序打开。由于攻击者利用整形数漏洞并调用程序中无意代码块，因此防御工具在运行完第一个无意代码块并转移到下一个无意代码块前，检测出攻击，并终止程序运行。

在实际攻击中，可使用的代码块个数并不多，在针对 Adobe Reader v9. 3. 4、Integard Pro v2. 2. 0 和 Mplayer Lite r33064 程序的攻击中，所需的代码块个数分别为 11、16 和 18。但多数代码块将被重复利用，执行的总次数分别为 33、165 和 179 次。尽管如此，实验结果表明，针对这 3 种程序的 ROP 攻击均在运行完第一个代码块并转移到下一个代码块前被防御工具检测出。

以 Adobe Reader 程序为例，第一个无意代码块执行后应将控制流转移到 0x20CB5955 处。在攻击者计划中，此处返回地址值应为 0x20CB5955。但由于加密密钥的存在，并且其为 0x2335488，经防御工具解密后，原来的返回地址值将变为 0x20CB5955 ⊕ 0x2335488 = 0x22F80DDD。程序将执行 0x22F80DDD 指令而非 0x20CB5955，导致程序异常终止，攻击被检测出。

值得注意的是，在引入加密保护返回地址值后，被篡改的控制流大部分将转移到非法地址处，引起程序执行失效，从而检测出攻击。但是并不排除控制流可能会转移到程序中的合法指令处，程序并不终止，而将继续执行。从攻击者角度分析，此时的程序执行将超出攻击者的预测，且程序的执行将不受攻击者的控制。从用户角度考虑，尽管此时程序的错误执行会造成错误的结果，但相比被攻击所带来的恶意入侵，程序由于错误执行而产生的错误计算结果(如错误的输出)对系统的危害性远低于恶意入侵。

为了测试本文所述开发工具的误检率，用 SPEC2000 和 SPEC2006 共 23 个基准程序及其所包含的 ref、train 和 test 负载来测试程序的正常行为。由于 call 和 ret 指令成对执行，实验结果表明，本文所述开发的工具在正常输入负载下，不仅能有效识别数量高达 3. 75E＋09 次的 call 和 ret 指令(SEPC2000 中的 gap 程序，运行 ref 负载)，并且能用模拟的 call 和 ret 操作替换这些指令，同时能保证程序正常执行，没有误报。

4.2 性能评估

为了评估上述方法的性能，在 Dell 微机(3GHz core2 CPU、2GB 内存)上运行 Ubuntu10. 04 操作系统(v2. 6. 32)，建立实验环境，其中编译器选用 GCC 4. 4. 3，并且采用 pin 2. 12-53271，SPEC 基准程序采用标准的参考工作负载。

SPEC 基准程序采用了 SPEC2006 和 SPEC2000。图 6 是 SPEC2006 的性能开销和运行时所调用的 RET 指令总数示意图。在图 6 中，PIN without instrumentation 是指在 PIN 平台上直接运行基准程序，不做任何修改。可以得知，在 PIN 平台上运行基准程序的平均开销为 1. 49x，其范围从基准程序 mcf 的 1. 04x 到 perlbench 的 1. 89x。PIN with RET protection 是指在 PIN 平台运行添加了本文所述保护机制后的基准程序，所得到的性能开销平均值为 2. 07x，最小开销 mcf 为 1. 1x，最大开销 sjeng 为 3. 11x。

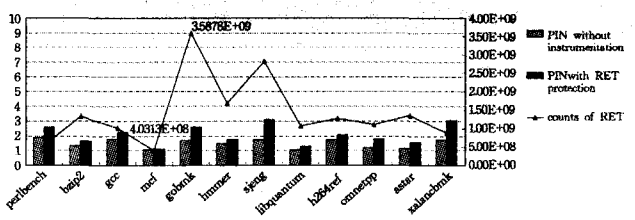

图 6 SPEC2006 性能开销

图 7 是 SPEC2000 的性能开销和运行时所调用的 RET 指令总数示意图。在图 7 中，在 PIN 平台上运行基准程序的平均开销为 1. 68x，最大值为 perlbmk 的 2. 65x，最小值为 mcf 的 1. 07x。在 PIN 平台运行添加了本文所述保护机制后的基准程序，所得到的性能开销平均值为 2. 28x，其范围从 mcf 的 1. 15x 到 gap 的 3. 66x。

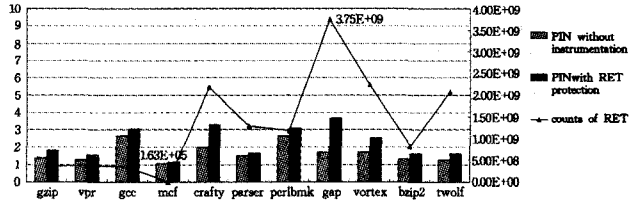

图 7 SPEC2000 性能开销

由于在 PIN 平台上添加对了返回地址值的保护机制，因此基准程序的性能开销将与程序中调用函数的次数成正比例，即在程序运行中函数调用次数越多，对返回地址值的保护

次数越多，因此相应地增加了性能开销。在图 6 和图 7 中，counts of RET 是指在程序运行期间 RET 指令的调用次数。由于大部分函数返回时均要通过 RET 指令，因此 RET 指令的调用次数与函数调用次数成正比例关系。在 SPEC2006 中调用函数次数最多的是 gobmk，最少的是 mcf，因此 mcf 的性能开销也相对 gobmk 较低，如图 6 所示。在 SPEC2000 中调用函数次数最多的是 gap，最少的 mcf，因此 gap 的性能开销也明显要高于 mcf，如图 7 所示。

相对于同样在二进制代码动态翻译工具上实现的 DROP 和 ROPdefender，本文所述的方法在性能开销上比 DROP 有优势，但与 ROPdefender 相当。

4.3 局限

本文所述的防御方法的密钥是在程序启动时随机生成的，并在程序运行期间一直有效，若攻击者通过其他攻击手段入侵操作系统甚至 PIN 程序，则该密钥有可能被攻击者所获得。因此进一步的研究可以考虑将返回地址值加密保护的方法在更独立的层面上（如操作系统层，甚至在硬件层面上）实现。

本文所述的防御方法是以返回地址值作为保护对象，在程序运行时动态地加解密，因此能有效检测出以 ret 指令结尾的代码块的 ROP 攻击。但是已有研究表明，可以不需要 ret 指令作为代码块的连接指令。如面向跳转指令编程的攻击（jump oriented programming，JOP）[26] 采用与 ret 指令同等功能的指令组合，如 x86 结构中 pop X；jmp X 指令对或者 ARM 结构中的 BLX 指令，用以替换 ret 指令，同样能构造具有同等图灵完全运算能力的 gadget。本文所述防御方法由于只对 ret 指令使用的返回地址值加解密，因此不能有效防御 JOP 攻击。

结束语 本文阐述了 ROP 攻击及其现有的防御方法，在分析各种防御方法的优缺点基础上，提出了利用返回地址保护机制实现代码复用类攻击的防御，并利用二进制代码翻译工具实现了该方法。其实验结果表明，所提出的方法能有效阻止以 0xC3 字符（即 ret 指令）结尾的短序列代码段的连续执行，从而防御 ROP 类型的代码复用攻击，并且其性能开销较小。

参 考 文 献

[1] Designer S. Getting around non-executable stack (and fix) [EB/OL]. http://seclists.org/bugtraq/1997/Aug/63, Bugtraq, 1997

[2] Nergal. The Advanced Return-into-libc(c) Exploits: PaX Case Study[J]. Phrack Magazine, 2001, 11(0x58)

[3] Shacham H. The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86) [C]// Proceedings of ACM Conference on Computer and Communications Security (CCS). Whistler: ACM New York Press, 2007: 552-561

[4] Rop attack against data execution prevention technology [EB/OL]. http://www.h-online.com/security/news/item/Exploits-new-technology-trick-% dodges-memory-protection-959253.ht-ml, 2012-12

[5] Cowan C, Pu C, Maier D, et al. StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks [C]// Proceedings of the 7th Conference on USENIX Security Symposium. San Antonio: USENIX Association, 1998: 63-78

[6] Etoh H. ProPolice: GCC extension for protecting applications from stack-smashing attacks [EB/OL]. http://www.trl.ibm.com/projects/security/ssp/

[7] Cowan C, Beattie S, Johansen J, et al. Point-guard: Protecting pointers from buffer overflow vulnerabilities [C]// Proceedings of the 12th USENIX Security Symposium. Washington: USENIX Association, 2003: 91-104

[8] Vendicator. Stack Shield technical info file v0.7 [EB/OL]. http://www.angelfire.com/sk/stackshield/, 2012-12

[9] Frantzen M, Shuey M. StackGhost: Hardware facilitated stack protection [C]// Proceedings of the 10th USENIX Security Symposium. Washington: USENIX Association, 2001: 271-286

[10] Abadi M, Budiu M, Erilingsson U, et al. Control-Flow Integrity: Principles, Implementations, and Applications[J]. ACM Transactions on Information and System Security, 2009, 13(1)

[11] Kiriansky V, Bruening D, Amarasinghe S. Secure Execution Via Program Shepherding [C]// Proceedings of 11th USENIX Security Symposium. San Francisco: USENIX Association, 2002: 191-206

[12] Bletsch T, Jiang Xu-xian, Freeh V. Mitigating Code-Reuse Attacks with Control-Flow Locking [C]// Proceedings of the 27th Annual Computer Security Applications Conference. Orlando: ACM New York Press, 2011: 353-362

[13] Chen Lin-bo, Jiang Jian-hui, Zhang Dan-qing. Code Reuse Prevention through Control Flow Lazily Check [C]// Proceedings of the 2012 IEEE 18th Pacific Rim International Symposium on Dependable Computing. Niigata: IEEE Computer Society, 2012: 51-60

[14] Li J, Wang Z, Jiang X, et al. Defeating return-oriented rootkits with return-less kernels [C]// Proceedings of the 5th European Conference on Computer Systems. Paris: ACM New York Press, 2010: 195-208

[15] Onarlioglu K, Bilge L, Lanzi A, et al. G-Free: Defeating return-oriented programming through gadget-less binaries [C]// Proceedings of 26th Annual Computer Security Applications Conference. Austin: ACM New York Press, 2010: 49-58

[16] Pappas V, Polychronakis M, Keromytis A D. Smashing the Gadgets: Hindering Return-Oriented Programming Using In-Place Code Randomization [C]// Proceedings of IEEE Symposium on Security and Privacy. Oakland: IEEE Computer Society, 2012: 601-615

[17] Hiser J, Nguyen-Tuong A, Co M, et al. ILR: where'd my gadget go [C]// Proceedings of IEEE Symposium on Security and Privacy. Oakland: IEEE Computer Society, 2012: 571-585

[18] Wartell R, Mohan V, Hamlen K W, et al. Binary Stirring: Self-randomizing Instruction Addresses of Legacy x86 Binary Code [C]// Proceedings of the 2012 ACM Conference on Computer and Communications Security. Raleigh, North Carolina: ACM New York Press, 2012: 157-168

[19] Chen P, Xiao H, Shen X, et al. Drop: Detecting Return-oriented Programming Malicious Code [C]// Proceedings of the 5th International Conference on Information Systems Security. Kolkata, India: Springer, 2009: 163-177

[20] Davi L, Sadeghi A, Winandy M. Dynamic Integrity Measurement and Attestation: Towards Defense against Return-oriented Programming Attacks [C]// Proceedings of the 2009 ACM Workshop on Scalable Tursted Computing. Chicago: ACM New York Press, 2009: 49-54

（下转第 102 页）

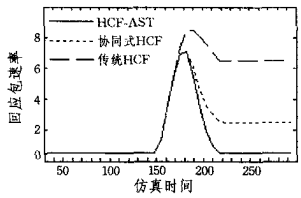

图 6　3 种防御策略下 target 回应包速率统计

表 2　3 种策略防御效果比较

防御策略类型	回应包速率最大值(个/s)	攻击开始减弱时间(t)	过滤攻击类型
传统 HCF	8.5	189	第①类
协同式 HCF	7.0	183	第①②类
HCF-AST	7.0	183	第①②③类

比较 3 种防御策略下的回应包速率最高值可知，传统 HCF 策略最高，另外两种防御策略稍低。比较图 6 中 3 条曲线(183s 后)的斜率可知，HCF-AST 策略下攻击流量减弱最快，协同式 HCF 策略次之，传统 HCF 策略最差。这是因为，在传统 HCF 策略下，只有 detector6 能检测到攻击，并令 filter6 过滤攻击流量，而在其它两种策略中，防御设备之间共享防御知识，使距离攻击源较近的 detector1 和 detector2 能够较早地检测到攻击并开始过滤，把攻击流量阻断在攻击源处，大大减少了进入网络内部的攻击流量。

在传统 HCF 策略下，防火墙只能消除第①类 DRDoS 攻击；在协同式 HCF 策略下，防火墙能够有效消除第①、②类 DRDoS 攻击；而在 HCF-AST 策略下，经过 53s，协同防御体系消除了所有的 DRDoS 攻击。另外，HCF-AST 防御发生在攻击流量反射前，在一定程度上弥补了传统方法反应滞后的不足。

总之，HCF-AST 协同防御模型能够及早发现并较快过滤所有类型的 DRDoS 攻击，有效地减少了攻击造成的影响损失。

结束语　本文提出的 DRDoS 协同防御模型，实现了不同防御手段及同一防御手段不同设备间的协同合作，充分利用有限的防御资源，弥补了设备单独防御的不足，能够有效消除所有类型的 DRDoS 攻击，并且防御反应速度较以往方法有较大提高。仿真实验证实了防御模型的有效性。在该模型中，防御设备间协同关系的设置至关重要，文中只考虑了不同类防御设备间的交互协同，而实际上，同类防御设备之间也可以通过协同共享防御资源，这将是今后需要关注的方向。

参 考 文 献

[1] 严芬，高玉龙，殷新春. DDoS 攻击检测进展研究[J]. 苏州大学学报：自然科学版，2011，27(13)：35-41

[2] Peng T. Detecting reflector attacks by sharing beliefs[J]. IEEE Global Telecommunication Conference，2003，46：1358-1362

[3] Jin C，Wang H，Shin K，et al. An effective defense against spoofedtraffic[C]// ACM International Conference on Computer and Communications Confercnce Security. 2003，10：30-41

[4] Noureldien N A，Osinan I M. A stateful inspection module architecture[C]//IEEE/RENCON. 2000，2：259-265

[5] Tsunoda H，Ohm K，Yamamoto A，et al. Detecting DDoS attacks by a simple response packet confirmation mechanism[J]. Computer Communications，2008：3299-3306

[6] 何雪妮. 一种改进的 DRDoS 检测算法[J]. 自动化与仪器仪表，2012，161(3)：150-151，155

[7] Wang Hai-ning，Jin Cheng，Shin K G. Defense against spoofed IP traffic using hop-count filtering[J]. IEEE/ACM Trans on Networking，2000，15(1)：40-53

[8] 张永花，崔永君. DRDoS 攻击及其防御技术研究[J]. 计算机安全，2009，4：53-55

[9] Mitzenmacher M，Upfal E. Probability and Computing，Randomized Algorithms and Probabilitic Analysis[M]. Cambridge：Cambridge University Press，2005：217-223

(上接第 98 页)

[21] Davi L，Sadeghi A，Winandy M. ROPdefender：A detection tool to defend against return-oriented programming attacks [C]// Proceedings of the 6th ACM Symposium on Information，Computer and Communications Security. Hong Kong：ACM New York Press，2011：40-51

[22] Chi-Keung Luk，Cohn R，Muth R，et al. Pin：Building Customized Program Analysis Tools with Dynamic Instrumentation [C]// Proceedings of 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). Chicago：ACM New York Press，2005：190-200

[23] Adobe CoolType SING Table 'uniqueName' Stack Buffer Overflow [EB/OL]. http://www.exploit-db.com/exploits/16619/，2010-09-25

[24] Integard Pro 2.2.0.9026 (Win7 ROP-Code Metasploit Module) [EB/OL]. http://www.exploit-db.com/exploits/15016/，2010-09-25

[25] MPlayer (r33064 Lite) Buffer Overflow + ROP exploit [EB/OL]. http://www.exploit-db.com/exploits/17124/，2011-04-06

[26] Checkoway S，Davi L，Dmitrienko A. Return-Oriented Programming without Returns [C]//Proceedings of ACM Conference on Computer and Communications Security (CCS). Chicago：ACM New York Press，2010：559-572

[27] Zovi D D. SOURCE Boston 2010：Practical return-oriented programming[EB/OL]. http://trailofbits.files.wordpress.com/2010/04/practical-rop.pdf

[28] Bhatkar S，Sekar R，DuVarney D C. Efficient Techniques for Comprehensive Protection from Memory Error Exploits [C]// Proceedings of 14th USENIX Security Symposium. Baltimore：USENIX Association，2005：105-120

[29] Roglia G，Martignoni L，Paleari R，et al. Surgically returning to randomized lib(c) [C]//Proceedings of Annual Computer Security Applications Conference. Honolulu：ACM New York Press，2009：60-69

[30] Chiueh T-C，Hsu F-H. RAD：A compile-time solution to buffer overflow attacks [C]// Proceedings of the 21st International Conference on Distributed Computing Systems. Phoenix：IEEE Computer Society，2001：409-420

[31] Schwartz E J，et al. Q：exploit hardening made easy[C]// Proceedings of 20th USENIX Security Symposium. San Francisco：USENIX Association，2011：379-394