高技术通讯 2014 年 第24 卷 第4 期:355 ~364

doi:10.3772/j.issn.1002-0470.2014.04.003

用多密钥加密方法防御面向返回编程的攻击①

陈林博② 江建慧 张丹青

(同济大学软件学院 上海 201804)

摘 要 针对面向返回编程(ROP)攻击的特性,提出了一种利用多密钥加密函数返回地址值防 ROP 攻击的方法。这种方法通过密钥指定位以及密钥迷惑位的巧妙设置,防止密钥在泄露情况下攻击者实施有效攻击的可能。利用二进制代码动态翻译工具 PIN 开发了原型系统,在实现时,为了避免攻击者攻击原型系统而直接访问密钥,引入了诱饵密钥,在增加攻击难度的同时降低了可信计算基。其有效性分析和实验结果表明,在不需要其他信息(如源码、调试信息等)条件下,多密钥加密防御方法能在适当的性能开销下,有效防御 ROP 攻击,并且其误报率极低。

关键词 代码复用类攻击,返回编程(ROP)攻击,多密钥加密保护,密钥迷惑位,诱饵密钥

0 引 言

最早代码复用类攻击是返回库(return into libc,RILC)攻击[1,2],由于 RILC 攻击所利用的代码段粒度大,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。攻击者倾向于选择具有图灵完全运算能力的面向返回编程(return-oriented programming,ROP)方式实施攻击[3]。由于程序中存在大量以 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),难以有效地防御 ROP 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(data execution prevention,DEP)技术。为防御 ROP 攻击,研究者提出了多种防御方法。包括指令对齐方法[5,6]、恶意代码消除[7-10]、设置影子栈保护返回地址值[11]、控制流一致性检测[12-15]等。然而现有的方法或是需要源代码信息,或者性能开销过大。部分方法需要依靠动态翻译工具实现防御,从而扩大了可信计算基的范围。

针对 ROP 攻击需要连接指令 ret 的执行将各个 gadget 组装并连续运行的特点,可采用对返回地址值加密保护的方法予以防御。然而这种方法由于只需要一个随机生成的密钥,并且加密方法简单,因此在加密地址泄露后(如通过格式化字符串攻击),很容易被攻击者猜出密钥值。为解决上述问题,本文提出利用多密钥加密方法防御代码复用类攻击。多密钥加密方法在程序运行时引入多个随机生成的密钥,同时为返回地址随机指定不同的密钥。为防止攻击者在获取全部密钥的情况下成功猜测密钥与地址的匹配关系,本文引入了密钥迷惑位,进一步提高了攻击者成功猜测加密方法的难度。多密钥加密方法不仅继承了返回地址值加密保护的优点,并且能在部分密钥泄露甚至在全部加密密钥被攻击者所获知的情况下,有效防御 ROP 攻击。本文阐述了多密钥加密的防御方法,介绍了多密钥加密的实现,分析了多密钥加密方法的有效性并用实验予以验证,评估了其性能开销并分析其局限性,讨论了相关工作,最后给出了结论。

1 多密钥加密方法

1.1 攻击模型假定

本文遵循如下几点假设:

(1)目标系统受 W ⊕ X 防御方法保护,攻击者

① 863 计划(2007AA01Z142)资助项目。

② 男,1984 年生,博士生;主要研究方向:信息安全,入侵检测与防御等;联系人,E-mail:08chenlinbo@tongji.edu.cn

(收稿日期:2013-06-14)

不能实施代码注入类攻击。但攻击者具备修改进程中可写内存的能力,攻击者也可以通过程序中的漏洞获得程序中的关键信息(如栈中返回地址值)。

(2)攻击者可以获得用于发动攻击的代码碎片,如可从动态加载的库函数中获取足够的代码碎片。并能通过各种漏洞(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)攻击者不能对操作系统及底层硬件环境发起攻击,即当前的可信计算基仅包括操作系统和硬件。

1.2 密钥指定位

多密钥加密方法是从一组程序启动时随机生成的密钥集中取出任一密钥用于返回地址值加密,而这一密钥也将用于返回地址值的解密,如图 1 所示。

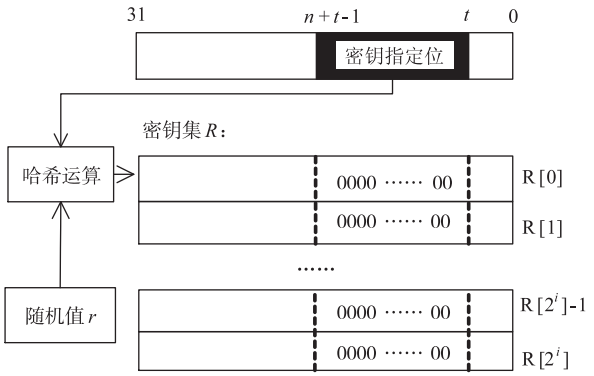

图 1 多密钥加密方法

为了从密钥集中取得随机指定的密钥,可将地址中的 n 位作为密钥指定位,并与一随机值进行哈希运算,用计算得到哈希值定位密钥。同时为了保证在解密时能引用相同的加密密钥,需要保证经过加密后的返回地址值相应的密钥指定位保持不变,在解密时利用相同的 n 位作为输入,经哈希运算后取得同一密钥。这里的加密算法采用异或运算,因此可对所有密钥相应的 n 位设置为 0,则加密后可保证地址值的 n 位保持不变。

设在 32 位地址中,n 位作为密钥指定位,密钥集 R 中共有 2^i 个密钥,为保证每个密钥都可被引用,可设置 $n > i$。每个密钥相应的 n 位均设置为 0。密钥指定的哈希运算可以是地址中 n 位值与程序启动时生成的随机值 r 相加,再取 2^i 模,即用 $(n + r) \bmod 2^i$ 的值就可确定密钥。由于加密算法采用异或运算,因此在地址值加密后,n 位的值不变。

设攻击者通过格式化字符串漏洞获取某一加密后的地址值,与本地相同的地址异或运算后获取一个密钥。由于 ROP 攻击中一般需要多个 gadget 实施攻击,不同的 gadget 的起始地址均不同,不同的地址可能匹配不同的密钥,因此攻击者不可能利用单一密钥伪装 gadget 的地址值以绕过多密钥防御方法。若密钥集有 2^i 个密钥,且一次 ROP 攻击需要 k 个 gadget,攻击者在获得一个密钥情况下,除非 k 个 gadget 都适应于该密钥,则攻击者可以破解多密钥方法。此时的成功攻击的概率为 $P_1 = (1/2^i)^k$,若 $k = 8$ 且 $2^i = 32$,则成功的概率仅为 $P_1 \approx 9.09e^{-13}$。

倘若攻击者通过格式化字符串漏洞进一步获取多个加密后的地址值,从而获取密钥集中的部分密钥。但由于不清楚返回地址与密钥的匹配关系,只能通过猜测来构造合适的返回地址值,在此种情况下成功攻击的概率仍然为 $P_1 = (1/2^i)^k$。在最坏的情况下,即使攻击者可以获得全部的密钥,但仍只能猜测返回地址值与密钥匹配关系来构造合适的恶意数据,成功攻击的概率仍为 P_1。从攻击者角度分析,攻击者需要推测出密钥与地址之间的匹配关系以实施成功的攻击。

密钥指定位 n 的大小与有效性成正比,即 n 越大,意味着可以设置更多的密钥(即 2^i 的值越大),攻击者成功猜测的概率也就越低,系统安全性越高。然而随着密钥指定位 n 的增大,由于部分地址中高比特位的值相似度较高,可能会造成哈希运算的值不够分散,会集中分布在某一区间内。而这种情况有可能会造成密钥集中某一段密钥会被大量使用,若攻击者使用这一段密钥,则会间接提高攻击者成功的概率。因此密钥指定位 n 的选择需要尽量避免地址中相似度较高的比特位,如地址高比特位。

1.3 密钥迷惑位

多个密钥可以防止单一密钥泄露后防御方法被绕过的问题,但是在最坏的情况下,攻击者可以获得全部的密钥,此时虽然攻击者不能从获得的密钥中直接推断出返回地址与密钥的匹配关系,但是有可能通过全部密钥中比特位为 0 的数量和位置,推测出密钥指定位。由于密钥指定所用的哈希算法较为简单,任意两组泄露的加密返回地址值就有可能被攻击者猜测出随机值 r。而一旦密钥指定位 n 和随机值 r 被攻击者所获得,则意味着密钥与地址之间的匹配关系被破解,多密钥加密方法也随即被破解。

为有效应对 n 位被猜测及随机值 r 被逆运算破解,防止返回地址与密钥之间的匹配关系被攻击者

所获知,可以通过在密钥指定位(即 n 位)的相邻两端增设零,用以迷惑攻击者,如图 2 所示。将新增加的设置为零的比特位称为密钥迷惑位,其本质是防止攻击者成功猜测 n 的位置和大小。在增设密钥迷惑位后,用 $m_{\min}$ 表示密钥中相同位为零的最小位的个数。攻击者在获取全部密钥后,需要在 $m_{\min}$ 位中猜测出正确的连续 n 位大小,然后才能通过逆哈希运算获得随机值 r。

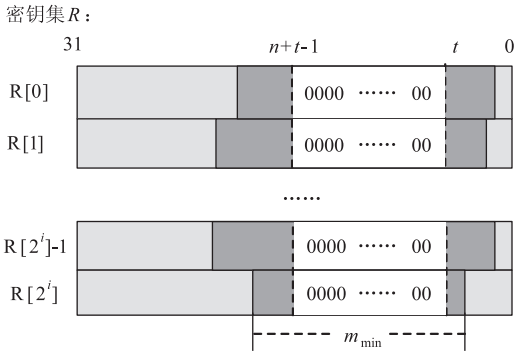

图 2　增设密钥迷惑位

为了进一步降低攻击者成功猜测 n 位的概率,增加攻击难度,可将连续的 n 位用离散的 n 位来替换。通过在地址中随机指定离散的 n 位作为密钥指定位,相应增设的密钥迷惑位也将与密钥指定位一起离散分布在地址中,如图 3 所示。其中 $R[j]$ 为密钥集中的密钥,m_1、m_2 和 m_3 代表在地址中分布的密钥指定位和密钥迷惑位的大小,因此 $m_{\min} = m_1 + m_2 + m_3$。由于所有密钥中其他比特位存在都为零的可能性,因此 $m_{\min}$ 的数量会有所增加。

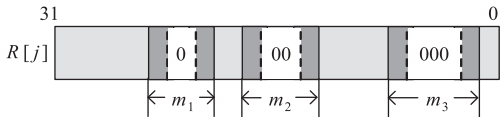

图 3　离散的密钥指定位与密钥迷惑位

需要注意的是,随着 $m_{\min}$ 位的增加,攻击者猜测连续 n 位的难度也在增加。但是随着密钥中值为零的比特位相应增加(这等同于异或加密后的地址值中不变的位数也会增加),需要变化的比特位也相应减少。若 $m_{\min}$ 位足够大,攻击者为提高成功率,可猜测地址中发生变化的 $32 - m_{\min}$ 位的值而非 n 位的信息(如 n 位的长度与位置)。这种攻击方法在 $m_{\min}$ 足够大时能增加成功攻击的概率,假设极端情况下 $m_{\min} = 31$,地址中只有一位在加密时需要改变,攻击者只需要猜测唯一发生变化的比特位是 0 或 1 就能破解多密钥方法,而此时的密钥已经失去了意义。因此需要权衡考虑 $m_{\min}$ 位的大小,本文第 3 节将分析这种攻击成功的概率,以确定最优的 $m_{\min}$ 位。

2　多密钥加密方法的实现

为评价多密钥加密方法,本文在 Linux/IA32 平台上利用二进制代码翻译工具 PIN[16] 开发了原型系统,实现了多密钥加密防御方法。

2.1　原型系统

PIN 不需要源码、调试信息等附件的信息,并且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被插桩的指令。利用 PIN 所开发的原型系统的框架如图 4 所示。

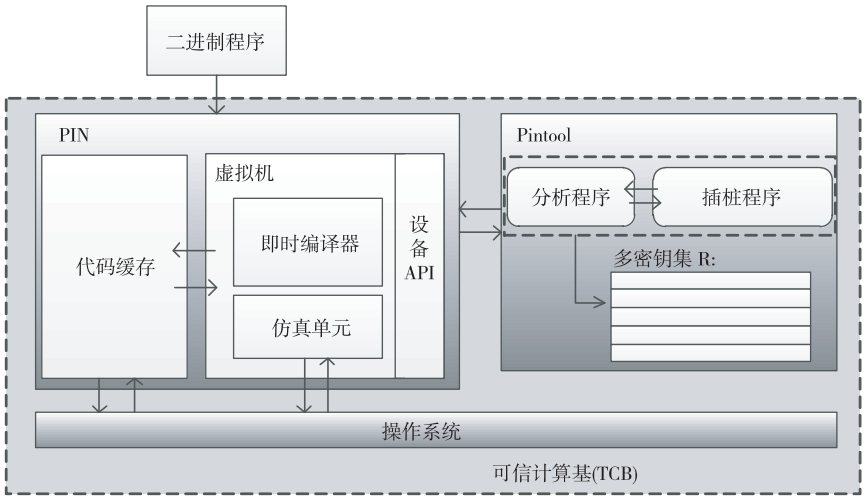

图 4　原型系统框架

原型系统主要由 PIN 和利用 PIN 所提供的应用程序编辑接口(application programming interface, API)函数开发的 Pintool 组成。其中 Pintool 中的分析程序(analysis routines)用于判断即将运行的指令是 call 指令或 ret 指令,插桩程序(instrumentation routines)则用模拟的 call 操作和模拟的 ret 操作替代原 call 指令和 ret 指令。模拟的 call 操作可以将即将压入栈中的返回值与与通过哈希运算得到的指定密钥(来自密钥集 R)异或后再压入栈中,实施加密操作。模拟的 ret 操作则是将栈中的返回值取出后,并与匹配的密钥异或后再放入指令寄存器企业信息门户(enterprise information portal, EIP)中实施解密操作。同时为保证多密钥方法的随机性,在每一次程序运行前随机生成密钥集 R 和随机值 r,并随机设置密钥指定位 n。

2.2 设置诱饵密钥

由于在开发原型系统过程中借助 PIN,因此 PIN 也需要作为可信计算基(trurted computing base, TCB)的一部分,假设其不受攻击的影响。然而作为软件,不能保证 PIN 本身不存在漏洞。若攻击者利用 PIN 或者其他类似的动态二进制翻译工具的漏洞直接获得密钥集中的全部密钥和随机值 r,则密钥指定位 n 的大小和位置也能被猜到。相对于通过泄露的加密地址值间接得到密钥的方法,攻击者可以更方便地通过对二进制翻译工具的攻击直接获取密钥。

为应对针对二进制翻译工具的攻击,缩小可信计算基的范围,一种有效的防御方法是在密钥集中增加诱饵密钥。诱饵密钥在程序启动时与其他真正用于加解密过程的密钥同时生成,但诱饵密钥并不会真正用于加解密中。诱饵密钥的存在是为了提高攻击者在获得真正密钥后猜测指定位 n 的难度,如图 5 所示,在连续 2^i 个密钥前后均设有诱饵密钥。由于存在诱饵密钥,攻击者需要多次获取加密后返回地址值,然后通过异或运算判定真密钥,这种穷举方法对于攻击者来说难度较大。本文第 3 节将详细分析诱饵密钥的有效性。

值得一提的是,既然攻击者可以通过对原型系统的攻击直接获取密钥,攻击者也能篡改在原型系统上保存的密钥值。因此为了避免关键数值被直接篡改,需在系统产生密钥集 R 和随机值 r 后,将其设置为可读,任何改变 R 和 r 值的写操作都被视为攻击,从而避免攻击者直接攻击原型系统后篡改关键数值。

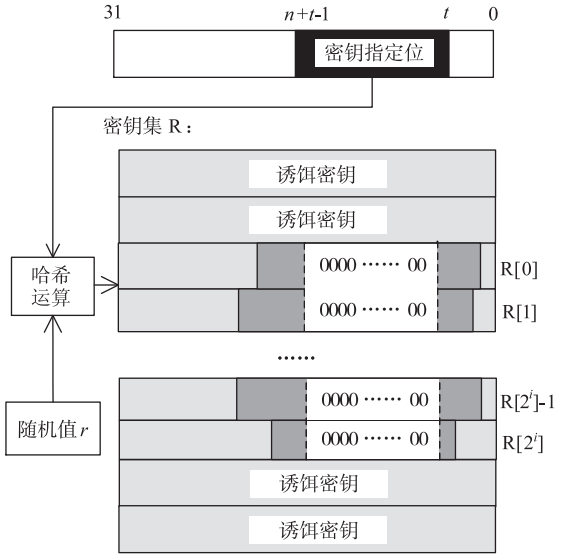

图 5 诱饵密钥

2.3 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,导致程序执行时 call 类指令和 ret 指令不会成对出现。其中一类情况是 Unix 信号处理,当控制流转移到信号处理函数时,返回地址值在不执行 call 指令的情况下仍被压入栈中,此时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。

另一类情况是延迟绑定,如在 Ubuntu 系统中,实现延迟绑定的函数之一_dl_rtld_di_serinfo 在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,而该函数将通过 ret 指令返回。由于_dl_rtld_di_serinfo 函数执行后所获得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。

3 评 估

3.1 有效性分析与验证

本节将分别分析多密钥加密方法及所开发的原型系统的有效性,并将原型系统移植到 Windows 系

统中以验证方法的有效性。

3.1.1 多密钥加密方法有效性分析

(1)猜测密钥指定位的成功率

当攻击者获得密钥集中单个密钥或部分密钥时,成功实施一次含有 k 个 gadget 的 ROP 攻击的概率为 $P_3 = 1/\sum_{i=1}^{m_{\min}} C_{m_{\min}}^{i}$,其中 2^i 为密钥集 R 中的密钥个数。若密钥集 R 中共有 $2^i = 32$ 个密钥,一次 ROP 攻击需要 $k = 8$ 个 gadget,则 $P_1 \approx 9.09e^{-13}$。

假若攻击者能获得全部的加密密钥,但由于密钥集中存在密钥迷惑位,因此攻击者需要在 $m_{\min}$ 位中猜测出正确的密钥指定位 n。假设当前地址为 32 位,当 n 个密钥指定位连续分布在 32 位地址中时,攻击者成功估计密钥指定位的概率为 $P_2 = 2/(m_{\min}(m_{\min}+1))$。当 n 个密钥指定位在 32 位地址中离散分布,则攻击者成功估计密钥指定位的概率为 $P_3 = 1/\sum_{i=1}^{m_{\min}} C_{m_{\min}}^{i}$。$P_2$ 与 P_3 的概率曲线图如图 6 所示。显然,采用离散的密钥指定位被成功猜测的概率要低于离散的密钥指定位,如 $m_{\min} = 10$,则 $P_2 \approx 0.0181$,$P_3 \approx 0.000978$。因此多密钥防御方法采用离散的密钥指定位的方法,在有效性上要优于连续的密钥指定位。

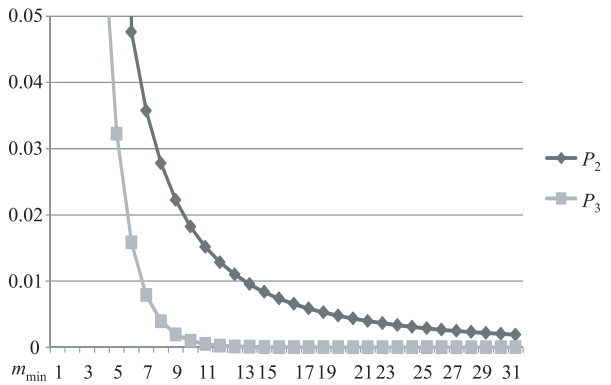

图 6 P_2 与 P_3 的概率曲线图

(2)猜测变化的比特位值的成功率

若攻击者采用猜测变化的比特位的值以替代猜测密钥指定位 n,即猜测剩余 $32 - m_{\min}$ 位的值,则此时能成功构造出符合多密钥加密方法的地址值的概率为 $P_4 = 1/2^{32-m_{\min}}$。当多密钥方法采用连续密钥指定位时,P_2 与 P_4 的概率曲线图如图 7 所示。

从图 7 中可以看出,随着所设置 $m_{\min}$ 数量的增加,攻击者成功猜测密钥指定位的概率随之降低,但

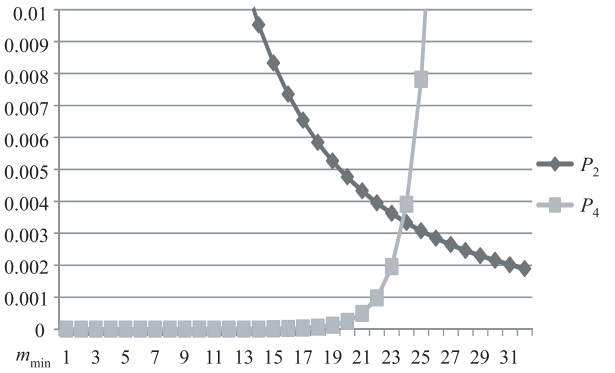

图 7 P_2 与 P_4 的概率曲线图

是猜测剩余 $32 - m_{\min}$ 位的值的概率却增加。而当 $m_{\min} = 24$ 时两者概率相差不大,此时 $P_2 \approx 0.003333$,$P_4 \approx 0.003906$。为达到较为理想的安全性,在采用连续密钥指定位时,$m_{\min}$ 的值可取 24。

而在采用离散密钥指定位时,P_3 与 P_4 的概率曲线图如图 8 所示。若 $m_{\min} = 16$,则 $P_3 \approx 1.5259e^{-5}$,$P_4 \approx 1.5258e^{-5}$,两者相差不大。因此,在采用离散密钥指定位时,可以使 $m_{\min} = 16$。

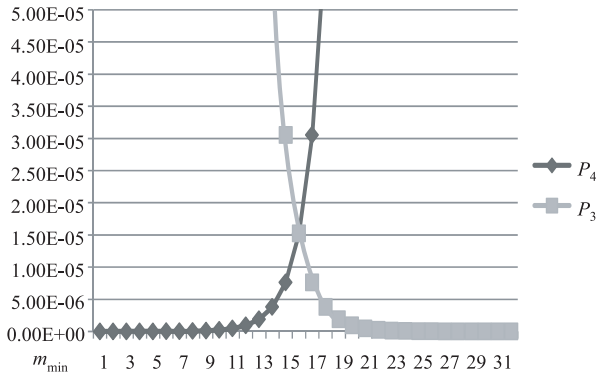

图 8 P_3 与 P_4 的概率曲线图

综上所述,对于攻击者而言,在获得全部密钥的情况下比获得单一密钥实施成功攻击的概率要高。在多密钥防御方法中,采用离散的密钥指定位的方法要优于连续的密钥指定位。而在 $m_{\min}$ 分别为 24 和 16 时,攻击者成功破解连续密钥指定位和离散密钥指定位的概率最低。

3.1.2 原型系统的有效性分析

在 3.1.1 节评估多密钥加密方法时,当前的可信计算基不仅包括底层硬件、操作系统,也包含了实现多密钥加密方法的动态二进制翻译工具。

然而在实际环境中,由于针对目标程序的直接攻击而获得全部的加密密钥难度较大,需要通过穷

举方法获得程序执行时所有的加密地址值以最终确定被使用的密钥。攻击者将会采用另一种较为简便的方法,即直接攻击原型系统,利用原型系统中的漏洞获得全部的加密密钥和哈希运算中的随机值 r。此时的可信计算基将不包括原型系统,因此有必要分析原型系统的有效性。

假设攻击者获得全部真实密钥的概率为 P_5,由于在实现原型系统时引入了诱饵密钥,若密钥集中包括诱饵密钥共有 L 个密钥,此时攻击者成功猜测真实密钥的概率为 $P_5 = 2/(L(L+1))$。在此基础上,攻击者成功破解连续的密钥指定位的概率为 $P'_2 = P_2 \times P_5 = 4/(L \cdot m_{\min}(m_{\min}+1)(L+1))$,成功破解离散的密钥指定位的概率为 $P'_3 = P_3 \times P_5 = 2/(L(L+1)\sum_{i=1}^{m_{\min}} C^i_{m_{\min}})$,成功猜测剩余 $32 - m_{\min}$ 位的值的概率为 $P_4' = P_4 \times P_5 = 1/(2^{31-m_{\min}} \times L(L+1))$。

假设密钥集中真实使用的密钥有 32 个,诱饵密钥有 224 个,密钥集中共有 256 个密钥,并且 $m_{\min} = 10$,攻击者成功破解连续密钥指定位的概率为 $P_2' = 5.527e^{-7}$,成功破解离散密钥指定位的概率为 $P_3' = 2.973e^{-8}$,成功猜测剩余 $32 - m_{\min}$ 位的值的概率 $P_4' = 3.624e^{-12}$。

3.1.3 有效性验证

为验证多密钥加密方法的有效性,在 Windows 系统中移植了本文所开发的工具,建立相应的实验平台,用实际的攻击验证方法的有效性。

实际的攻击对象有 Adobe Reader v9.3.4[17]、Integard Pro v2.2.0[18]和 Mplayer Lite r33064[19]等程序,针对这三种程序的攻击都能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。其中可使用的代码块个数在 10 到 18 之间,而大多数的代码块在 ROP 攻击中将被重复利用。

在实验中,被攻击的程序运行在所移植的原型系统上。实验结果表明,针对这三种程序的 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,程序发生异常终止,防御工具检测出攻击。

另外,在性能评估中以 SPEC2000 作为评估对象,所开发的工具在正常输入负载下不仅能有效识别数量高达 1.12E + 11 次 call 和 ret 指令(SEPC2000 中的 parser 程序,运行 ref 负载),并且能保证程序正常执行,没有误报。

3.2 性能评估

为了评估多密钥加密防御方法的性能,本文在 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,SPEC2000 基准程序采用标准的参考工作负载。其性能开销如图 9 所示。

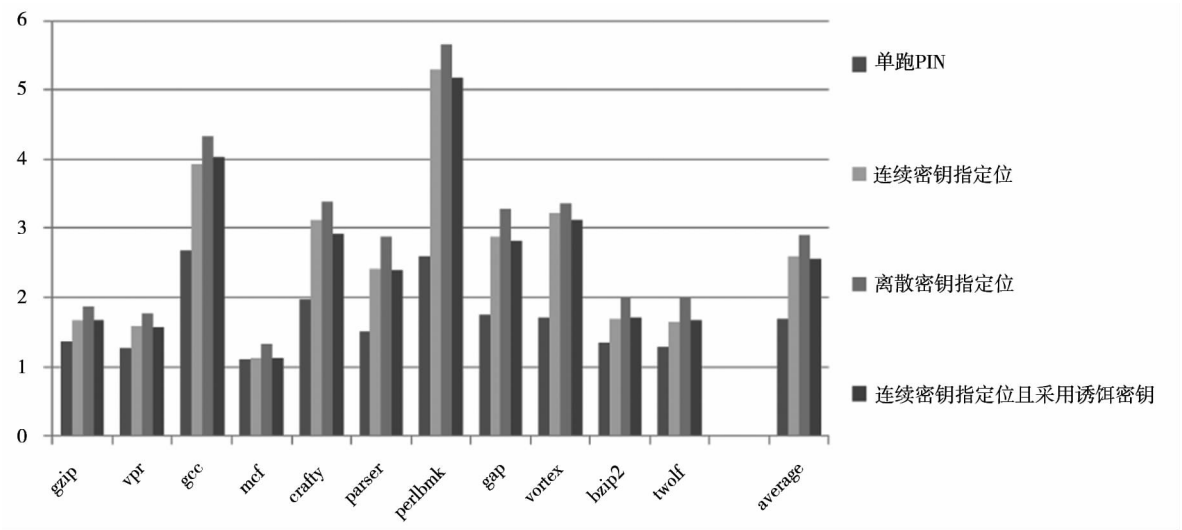

图 9 性能开销图

在图 9 中,“PIN without instrumentation”是指在 PIN 上直接运行基准程序,不添加任何插桩代码;“PIN + consecutive Multi-key”是指在 PIN 上实现的采用连续密钥指定位的多密钥防御方法,其中密钥集 R 中共有 32 个密钥,密钥指定位 $n = 8$,密钥迷惑位 $m_{\min} = 24$;“PIN + discrete Multi-key”是指在 PIN

上实现的采用离散的密钥指定位的多密钥防御方法,其中密钥集 R 中共有 32 个密钥,密钥指定位也是 $n=8$,但密钥迷惑位 $m_{\min}=16$;“PIN + consecutive Multi-key + cheating key”是指在“PIN + consecutive Multi-key”基础上增加了诱饵密钥,其中所增加的诱饵密钥个数为 224 个,总的密钥数 $L=256$。

从图 9 中可以看出,在 PIN 上不添加任何插桩代码,直接运行基准程序的性能平均开销为 1.686 倍。采用连续密钥指定位的多密钥防御方法的性能平均开销 2.595 倍,其性能增加的比例如图 10 所示。

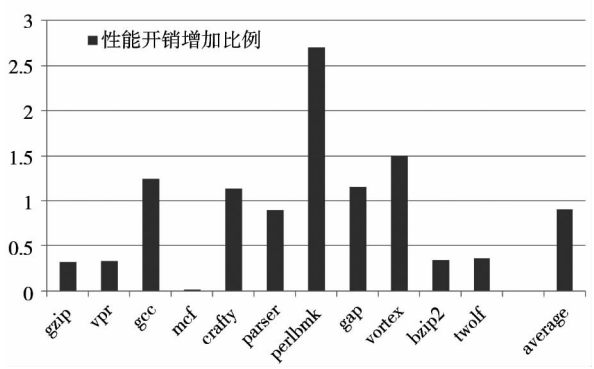

图 10　连续密钥指定位的性能开销增量图

性能开销增加的原因主要是在程序执行时,需要在每条 call 指令和 ret 指令执行前实施 call 操作和 ret 操作,建立返回地址与密钥的匹配关系,并对返回地址加密和解密。实验发现,在采用连续密钥指定位的多密钥方法中,获得密钥指定位仅需要简单的移位计算。且哈希运算较为简单,并且访问密钥集中单个密钥的开销时间也很小,因此在匹配密钥时所需时间较低,性能开销主要是由实现时的 call 操作和 ret 操作所造成。因此性能开销的增量与当前程序所需访问的 call 指令和 ret 指令的频率成正比,如图 11 所示。频率越高则意味着需要执行的加解密操作越频繁,而相应的性能开销也将越大。

“PIN + discrete Multi-key”的性能开销比“PIN + consecutive Multi-key”有所增加,主要原因是因为离散密钥指定位方法首先需要取得相应的密钥指定位,而密钥指定位离散地分布于地址中,需要执行多次移位运算和算术运算。密钥指定位越离散,所需要的计算指令越多,性能开销也会增加。尽管离散的密钥指定位带来了性能上的开销,其平均性能开销为 2.89 倍,与连续密钥位的平均性能开销相差不大。

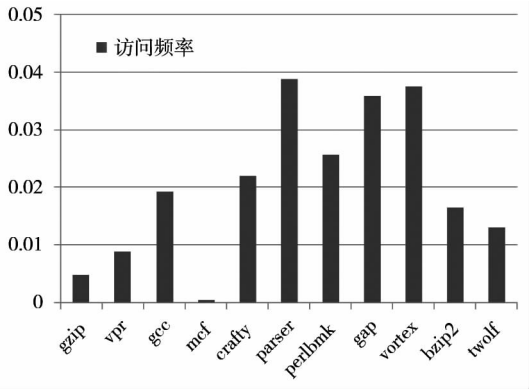

图 11　call 指令和 ret 指令的访问频率

在增加诱饵密钥后,即“PIN + consecutive Multi-key + cheating key”中全部程序相比“PIN + consecutive Multi-key”的性能开销几乎为零。这是由于在实现时,密钥集中所增加的诱饵密钥数量仅为 224 个,总数不多,且增加的密钥也不会被访问,因此并不会因为密钥集中诱饵数量的增多而导致访问时间增加。

3.3　局限

本文所述的多密钥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(jump-oriented programming,JOP)的攻击[20]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。由于本文所述防御方法只对 ret 指令使用的返回地址值加解密,因此该防御方法不能有效防御 JOP 攻击。在下一步的研究中,拟考虑将所有的间接跳转目的地址作为保护对象,实现控制流异常转移的有效检测,以防御多种代码复用类攻击。

攻击者可以通过 Non-control-data 攻击[21],不篡改控制流数据(如函数返回地址值)就可以达到攻击目的,如提升访问权限。尽管这类攻击已经不属于本文所讨论范畴,但是攻击者仍然可以篡改某些条件转移的标志值,以达到控制流非法转移,复用程序中已有的正常代码。本文所述的多密钥防御方法不能有效防御 Non-control-data 攻击,在后续研究中将扩大检测对象,不再局限于返回地址值的检测,并且可以考虑将程序的执行路径(条件转移的信息)拟合到检测对象中。

4 相关研究

针对代码复用类攻击的特性的防御方法可以分为加固含有恶意代码的程序和加固程序运行环境两大类。由于程序的现存代码可被重新组装成恶意代码,因此,替换与消除恶意代码可以在源头上抵御复用类攻击。加固程序运行环境则是根据代码复用类攻击的特性,在攻击实施后,利用其某一特性(如输入数据中含有大量地址值或 ret 指令执行频率高等)检测代码复用类攻击。

4.1 加固含有恶意代码的程序

在返回库(RILC)攻击中需要返回到程序代码段或加载库中有用的函数,因此可以移除其中不常用的函数如 system( ),以避免 RILC 攻击利用此类函数实施攻击,如 LibSafe[7] 通过检测和替换一些特定的库函数来防御 RILC 攻击。然而这种方法却不能有效防御 ROP 攻击和 JOP 攻击。

对于利用代码段甚至代码碎片作为恶意代码的面向返回编程(ROP)攻击及面向跳转指令编程(JOP)攻击,现有的策略则是通过消除程序中可能存在的恶意代码,用具有相同功能的但无法用做恶意代码的其他指令序列替代。

由于 ROP 攻击中需要以 ret 指令作为链接,将多个 gadgets 组装使用,因此 Li 等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[8]。在编译时消除 ret 指令的方法还有 G-free[9],它不仅在编译时采用寄存器重分配、指令变换、立即数混淆等多种方法消除含有 0xc3 字节的指令,并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,但由于需要对程序重新编译,并且若程序中的部分库函数不适用此类编译器,则不能全面有效地防御 ROP 攻击。并且基于编译器的指令替换方法需要源代码,不适用于部分库函数。而本文所述的多密钥加密防御方法则不需要源代码及其他附件信息。

防御攻击的 IPR 方法[10] 是针对二进制可执行文件的一种有效的指令替换方法,它首先利用反汇编工具提取二进制代码中的 gadget,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,消除程序中存在的恶意代码的影响。尽管有大约 77% 的 gadget 可以通过 IPR 消除或者部分消除,但是大型程序中至少存在数以万计的 gadget,IPR 防御方法的覆盖范围显然不够。而本文所实现的原型系统则可以检测所有 ret 指令的运行。

在 ROP 攻击和 JOP 攻击中,所使用的恶意代码多数为无意代码。造成无意代码的原因是由于指令集过于密集,任意组合的字符串均有极大概率被解释为有效的指令。因此可以通过指令集的对齐以消除无意代码的执行,如 McCamant 等在文献[5]及 Yee 等在文献[6]中所述。首先禁止所有的指令都必须以 n 字节对齐,对于达不到 n 字节大小的指令添加空指令对齐。其次所有的间接控制流的转移必须以 n 字节对齐,即不能跳转到指令中。最后所有的间接控制流转移的目的指令也必须以 n 字节对齐。其中 n 可以是 2 的倍数,一般取 $n=32$。

指令对齐方法本质上是消除程序中无意代码的威胁,但是不能消除程序中正常的以 ret 指令结尾的正常指令序列的威胁,攻击者有可能利用正常的指令序列实现攻击。尽管采用指令对齐方法尽管让攻击者所使用的恶意代码颗粒度有所增加,从字节级提升到函数级,但是已有研究表明,以函数作为恶意代码粒度的 RILC 攻击已被证明具有图灵完备运算的能力[22]。

4.2 加固程序运行环境

一次成功的代码复用类攻击不仅需要程序中含有内存漏洞和可用的恶意代码,更需要由攻击者精心设计的恶意数据输入,以改变控制流使其指向恶意代码处。因此可以利用代码复用类攻击的特性,在程序运行环境中引入防御机制,实时检测代码复用类攻击。

由于 ROP 攻击所采用的 gadget 的长度小,且其 ret 指令执行的频率高,根据这一特性,可通过检测 ret 指令出现的频率用以检测 ROP 攻击。检测 ROP 恶意代码的方法 DROP[23] 通过对动态插桩二进制代码在程序运行时查找以 ret 指令结尾的短序列的个数,判断 ret 指令出现的频率,以此检测 ROP 攻击。类似的方法还有 DynIMA[24]。但是攻击者可以通过增加指令的条数绕过此类防御方法,并且其性能开销较大,如 DROP 的平均性能开销高达 5 倍。

考虑到 ROP 攻击需要利用到栈中数据,特别是控制栈中返回地址值,ROPdenfender[11] 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 2 倍以上。而且二进制代码动态翻译工具可被攻击者直接攻击,影子栈中保存的返回地址值容易被篡改。而本文所开发的原型系统能在被直接攻击的情况下,有效防御 ROP 攻击,缩小了可信计算基的范围。

ILR 方法[25]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。同样,ILR 方法也不能完全覆盖程序中所有的 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。控制流完整性(CFI)[12]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。类似的方法有程序领引(program shepherding)[13]、控制流锁定(contol flow locking, CFL)[14]和控制流缓慢检查(control flow lazy checking, CFLC)[15]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。

5 结 论

针对代码复用类攻击特别是 ROP 攻击的特性,本文提出了一种利用多密钥加密返回地址值来防御攻击的方法。为防止攻击者利用程序中的漏洞获取密钥后绕过防御方法,在多密钥中引入了密钥迷惑位。分析针对连续密钥指定位及离散密钥指定位的攻击者成功攻击的概率,确定了最佳的密钥迷惑位的大小。并且利用 PIN 开发了原型系统,实现了多密钥加密方法。在实现时,为了避免攻击者直接攻击原型系统获取密钥,引入了诱饵密钥,增加了攻击的难度。实验表明,在不需要其他信息(如源码、调试信息等)条件下,多密钥加密防御方法能在适当的性能开销下,有效防御 ROP 攻击,并且其误报率极低,所提出的诱饵密钥缩小了可信计算基的范围。

参考文献

[1] Solar Designer. Getting around non-executable stack (and fix). http://seclists.org/bugtraq/1997/Aug/63: Bugtraq, 2012

[2] NERGAL. The Advanced Return-into-libc(c) Exploits: PaX Case Study. http://phrack.org/issues/58/4.html: Phrack, 2013

[3] Shacham H. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In: Proceedings of 14th ACM Conference on Computer and Communications Security, Alexandria, USA, 2007. 552-561

[4] Stojanovski N, Gusev M, Gligoroski D, et al. Bypassing Data Execution Prevention on Microsoft Windows XP SP2. In: Proceedings of the Second International Conference on Availability, Reliability and Security, Vienna, Austria, 2007. 1222-1226

[5] McCamant S, Morrisett G. Efficient, verifiable binary sandboxing for a CISC architecture. MIT-CSAIL-TR-2005-030, Massachusetts: MIT, 2005

[6] Yee B, Sehr D, Dardyk G, et al. Native client: a sandbox for portable, untrusted x86 native code. *Communications of the ACM*, 2010, 53(1): 91-99

[7] Tsai T, Singh N. Libsafe: transparent system-wide protection against buffer overflow attacks. In: Proceedings of the International Conference on Dependable Systems and Networks, Washington, D. C., USA, 2002. 541-551

[8] Li J, Wang Z, Jiang X, et al. Defeating return-oriented rootkits with return-less kernels. In: Proceedings of the 5th European conference on Computer systems, Paris, France, 2010. 195-208

[9] Onarlioglu K, Bilge L, Lanzi A, et al. G-Free: defeating return-oriented programming through gadget-less binaries. In: Proceedings of 26th Annual Computer Security Applications Conference, Austin, USA, 2010. 49-58

[10] Pappas V, Polychronakis M, Keromytis A D. Smashing the gadgets: hindering return-oriented programming using in-place code randomization. In: Proceedings of IEEE Symposium on Security and Privacy, Oakland, USA, 2012. 601-615

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

[12] Abadi M, Budiu M, Erilingsson U, et al. Control-flow integrity: principles, implementations, and applications. *ACM Transactions on Information and System Security*, 2009, 13(1): 4

[13] Kiriansky V, Bruening D, Amarasinghe S. Secure execu-

tion via program shepherding. In: Proceedings of the 11th USENIX Security Symposium, San Francisco, USA, 2002. 191-206

[14] Bletsch T, Jiang X X, Freeh V. Mitigating code-reuse attacks with control-flow locking. In: Proceedings of the 27th Annual Computer Security Applications Conference, Orlando, USA, 2011. 353-362

[15] Chen L B, Jiang J H, Zhang D Q. Code reuse prevention through control flow lazily check. In: Proceedings of the 2012 IEEE 18th Pacific Rim International Symposium on Dependable Computing, Niigata, Japan, 2012. 51-60

[16] Chi-Keung L, Robert C, Robert M, et al. Pin: Building customized program analysis tools with dynamic instrumentation. In: Proceedings of 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, Chicago, USA, 2005. 190-200

[17] Metasploit. Adobe CoolType SING Table 'uniqueName' Stack Buffer Overflow. http://www. exploit-db. com/exploits/16619/: Offensive Security, 2013

[18] Node. Integard Pro 2. 2. 0. 9026 (Win7 ROP-Code Metasploit Module). http://www. exploit-db. com/exploits/15016/: Offensive Security, 2013

[19] Nate_M. MPlayer (r33064 Lite) Buffer Overflow + ROP exploit. http://www. exploit-db. com/exploits/17124/: Offensive Security, 2013

[20] Checkoway S, Davi L, Dmitrienko A. Return-oriented programming without returns. In: Proceedings of ACM Conference on Computer and Communications Security, Chicago, USA, 2010. 559-572

[21] Chen S, Xu J, Sezer E C, et al. Non-control-data attacks are realistic threats. In: Proceedings of the 14th USENIX Security Symposium, Baltimore, USA, 2005. 77-192

[22] Tran M, Etheridge M, Bletsch T, et al. On the expressiveness of return-into-libc attacks. Recent Advances in Intrusion Detection Lecture Notes in Computer Science Volume 6961, 2011. 121-141

[23] Chen P, Xiao H, Shen X, et al. Drop: detecting return-oriented programming malicious code. In: Proceedings of the 5th International Conference on Information Systems Security, Kolkata, India, 2009. 163 -177

[24] Davi L, Sadeghi A, Winandy M. Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks. In: Proceedings of the 2009 ACM Workshop on Scalable Trusted Computing, Chicago, USA, 2009. 49-54

[25] Hiser J, Nguyen-Tuong A, Co M, et al. ILR: where' d my gadgets go? In: Proceedings of 2012 IEEE Symposium on Security and Privacy, San Francisco, USA, 2012. 571-585

Prevention of return-oriented programming attacks using multi-key protection

Chen Linbo, Jiang Jianhui, Zhang Danqing

(School of Software Engineering, Tongji University, Shanghai 201804)

Abstract

The multi-key protection method was applied to the study, and a novel technique was presented to prevent return-oriented programming (ROP) attacks based on the analysis of ROP attack properties. This new technique, which benefits from protection of return address by multi-key, introduces obfuscated bits to prevent exploits targeted key sets. With the help of binary dynamic translator of the Intel PIN, a proof-of-concept prototype system for Linux platform was implemented. In order to prevent the direct attacks targeted prototype, the cheating keys were proposed to obfuscate attackers as well as decrease the trusted computing base. The effectiveness analysis and the experimental results demonstrate that the multi-key protection method could effectively prevent ROP attack with a modest performance penalty and low false positive rate.

Keywords: code reuse attacks, return-oriented programming (ROP) attacks, multi-key protection, obfuscated bits, cheating keys