地址空间随机化技术研究

戈 载 徐良华 史 洪
(江南计算技术研究所 江苏 无锡 214083)

摘 要 地址空间随机化 ASR(address space randomization)技术的目的是使进程空间不可预测,从而保护系统不受攻击。研究了几乎所有主流的 ASR系统和技术的原理;分析了其优缺点;研究了对部分 ASR系统进行攻击的方法;最后提出了实现实用的 ASR系统需要注意的一些问题。

关键词 地址空间随机化 缓冲区溢出 安全漏洞 网络安全

RESEARCH ON ADDRESS SPACE RANDOMIZATION TECHNIQUES

Ge Ji Xu Lianghua Shi Hong
(Jiangnan Institute of Computing Technology, Wuxi 214083, Jiangsu, China)

Abstract The goal of Address Space Randomization technology is to make the address space of a process unpredictable and protect the system from attacks. Almost all main Address Space Randomization systems and technologies are studied, their advantages and shortcomings are analyzed as well, the way of attacking against some of these systems are studied, and several issues on the realization of ASR are presented in the end of the paper.

Keywords Address space randomization(ASR) Buffer overflow Vulnerability Network security

0 引 言

1988年 11月 2日,莫里斯利用 Unix操作系统 fingerd软件中缓冲区溢出安全漏洞编写了莫里斯蠕虫,莫里斯蠕虫感染了数万台计算机,造成了巨大的损失,这也是缓冲区溢出安全漏洞被首次利用。从那以后,越来越多的缓冲区溢出安全漏洞被发现,利用缓冲区溢出安全漏洞的攻击已经成为网络安全的最大威胁,根据 US-CERT(美国计算机应急处理小组)报导,60%的针对计算机系统的攻击行为属于这一类。

按照修改程序执行流程的方式来划分,基于缓冲区溢出安全漏洞的攻击包括栈溢出[1]、堆溢出[2]、整形溢出[3]、格式化字符串攻击[4]和双重 free[5]攻击等。按照攻击代码的来源可以分为需要植入代码的攻击和 return-into-lib(c)[6]攻击。

尽管如 StackGuard[7]、FormatGuard[8]之类的系统可以阻止栈溢出和格式化字符串攻击,PaX[9]中的可写页不可执行技术(NOEXEC)和 Windows XP SP2之后的操作系统中的数据执行保护技术(DEP)可以有效地阻止植入代码的攻击,但是每个系统或技术只能防御某一特定方式的攻击,要将这些系统或技术集成在一起却很难,而新的攻击方式还在不断出现。另外,这当中某些技术存在额外开销过大的问题。为了避免这些不足,研究人员提出了从另一个角度来对系统进行保护的技术,即 ASR技术。ASR的基本出发点是:基于缓冲区溢出安全漏洞的攻击需要事先熟悉进程的地址空间,从而将程序的执行流程跳转到攻击代码的位置。为此,ASR技术随机修改进程的地址空间分布,使得攻击者无法将程序的执行流程跳转到预期的位置,从而阻止进程执行攻击者预先设置好的攻击代码。分析和实验表明,ASR技术能够以比较小的额外开销有效地降低攻击者成功的概率。

1 ASR技术

ASR技术主要对进程的栈、堆、主程序代码段、静态数据段、共享库(Shared Library)、GOT(Global Offset Table)等所在的地址进行随机化。图 1是 ASR技术的示意(以基本的栈溢出为例),不采用 ASR技术的进程,因其地址空间固定,攻击者可预先得出图中攻击代码的地址,将其覆盖到某函数返回地址的位置,使函数返回时跳转到攻击代码的位置执行;采用了 ASR技术后,因为图中攻击代码的地址不固定,攻击者只能猜测该地址,所以不能保证能正确地跳转到攻击代码的位置。ASR的最终效果在于:① 降低攻击者成功的概率;② 使针对一台计算机的攻击方法不能简单地移植到另一台计算机上。在 Windows Vista之前,ASR技术已经被提出并在一些开源系统中实现,本节按照各种 ASR技术的成熟度对它们进行介绍。

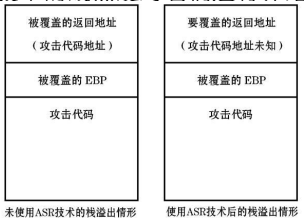

图 1 ASR技术原理示意

收稿日期:2008-03-06。国家 863高技术研究发展计划基金项目(2006AA01Z431)。戈载,硕士生,主研领域:信息安全。

1.1 早期的 ASR

S Forrest最早提出了 ASR的概念[10],指出计算机系统的相似性会带来安全隐患——对一台计算机的攻击方法可以轻易移植到其他计算机上,带来攻击的大众化和病毒的快速传播,并首次提出通过进程地址空间随机化的方法来实现多样化的计算机系统。S Forrest提出了一些地址空间随机化的方法,如修改栈基地址、为局部变量分配冗余空间、随机化静态变量地址等方法,并在原形系统中实现了其中的一种:修改 gcc编译器,如果一个局部变量需要的栈空间超过 16字节,则在分配空间时随机增加 8、16、24、…、64个字节。

M. Chew等人提出了将操作系统随机化来抵御缓冲区溢出攻击的方法[11],指出了多种实现技术,其中 ASR技术是修改操作系统内核,随机化进程栈基址。实现方法是:进程初始化时,先在进程栈顶放入 0-32K个字节的数据,将栈顶指针减去放入的字节数,然后开始进程。这样,进程实际使用的栈基地址就被随机化了。

Homebrew[12]是在 FreeBSD系统下实现 ASR的一种方案,可以通过修改 gcc在编译期间实现,也可以修改内核在进程加载时实现。实现方法是:修改进程栈基址 (将栈顶指针减少 0-1M),并且填充随机数量 (0-16K)的字节到每个函数的栈空间。这样不仅进程栈空间发生了变化,每个函数的栈空间也发生了变化,增加了要得到栈中某个变量的地址的难度。

这些早期的实现方案只对进程的栈空间进行了一定程度的随机化,使攻击者无法轻易得到栈中某个变量的地址 (攻击代码的起始地址),因此不能确定在函数的返回地址位置写入的数值,增加了攻击的难度。但这些早期的简单的方案无法抵御多种类型的攻击 (如堆溢出等),比较容易被绕过。而且,由于随机空间太小,容易被暴力破解。但这些方案提出的通过随机化进程地址空间来抵御缓冲区溢出攻击的思路具有重要意义。

1.2 成熟的 ASR

PaX[9]中的 ASR技术被认为是目前实现得最成功的,通过为 Linux内核打补丁,实现了对栈基地址、主程序基地址以及共享库的加载地址的随机化。实现方法是,在进程加载时,对栈基地址的 4—27位共 24位进行随机化,对包括主程序映象、静态数据区、堆这一连续区域的基地址的 12—27位共 16位进行随机化,对共享库加载地址的 12—27位共 16位进行随机化。这种 ASR技术加上可写页不可执行技术,PaX构成一个完整的实用的系统防护方案,可写页不可执行技术迫使攻击者必须使用进程空间中现有的代码 (如共享库中的函数)进行攻击,ASR使得这些现有的代码的位置不确定。而且,如果攻击者使用的共享库函数需要参数,攻击者就需要知道放入栈中 (或堆中)的参数的位置,而这也是不确定的。通过这些技术,PaX能大大降低攻击成功的概率,在 linux系统中得到广泛的应用。

TRR[13]通过修改进程加载器来实现 ASR,不需要修改系统内核。采用的随机方法与 PaX类似,在进程加载时修改了栈基址、堆基址以及共享库的加载地址。与 PaX相比,TRR增加了对 GOT的随机化,以阻止修改 GOT函数指针指向恶意代码的攻击。GOT保存在程序的静态数据区,在动态链接的程序中,动态库中的函数并不包含到主程序代码中,在编译时编译器将这些函数的地址保存在 GOT中,并将 GOT的地址保存在代码段中的 PLT(procedural linkage table)中,主程序对动态库中函数的调用通过 PLT和 GOT协作来实现。攻击者常常修改 GOT中的函数指针,指向预先安排好的攻击代码,因此 TRR对 GOT的位置进行了随机化。要对 GOT进行随机化,不仅需要将 GOT放在随机的位置,还需要修改 PLT中对 GOT的索引,使其正确指向 GOT。TRR修改了进程加载器,在进程加载时将 GOT移动到新的位置,并用二进制代码修改工具修改代码段中的 PLT,从而实现了 GOT的随机化。

这一类 ASR实现方案对栈、堆、主程序代码段以及共享库等在进程空间的基地址进行了随机化,粒度较粗、实现简单、额外开销较小,目前得到了广泛的应用,OpenBSD和 Windows Vista也采用了类似的方法。它们的缺陷主要在于只随机化了基地址以及使用的随机空间不大,Tyler Durden和 H. Shacham分别从相对地址攻击和暴力破解的角度分析了 ASR的缺陷[14,15],将在下面第二节进行介绍。

1.3 高级的 ASR

S Bhatkar先提出了称为 Address Obfuscation的实现方案[16],通过代码修改工具修改二进制代码:① 修改进程中所有单元的绝对地址;② 在栈中每个函数的参数和局部变量间、堆中 malloc分配的缓冲区之间随机增加空隙;③ 在静态变量间、函数间增加空隙并增加 jump指令跳过;④ 修改变量、函数的相互顺序。其中③和④没有实现。这种方案不仅修改了进程中各单元的绝对地址,而且修改了各个单元内部的相对地址,能抵御基于相对地址的攻击。其缺陷在于,因为需要在进程加载前修改二进制代码来确定随机参数,实现较为复杂,而且为了阻止对参数的猜测,需要经常修改,这样不仅开销很大,而且与某些具有程序完整性检查功能的入侵检测系统冲突。同时,由于实现难度很大,③和④没有实现,这使得系统抵御攻击的能力有所降低。

因为 Address Obfuscation的缺陷,S Bhatkar随后提出了另一种随机化相对地址的方案[17],通过修改 C源代码,可以生成相同的可执行代码,在进程加载时通过可执行代码修改地址空间,这样无须修改操作系统内核,实现简单,与 Address Obfuscation相比可行性更高,并且提出了将缓冲区类型的变量 (如数组)分开存储的思路。这种方案的缺点是需要修改程序源代码,采用自动化工具可能不能正确地修改一些比较复杂的程序代码。

Code Islands[18]主要针对 chained return-into-lib(c)之类需要知道多个库函数地址的攻击,指出只修改基地址不能有效地阻止攻击者进行与文献[15]中类似的暴力破解。该方案在程序或者动态库加载时将函数划分到不同的块中 (称为 code islands),并加载到随机的位置。这样猜测得到一个库函数的地址后,不能根据相对位置来计算其他库函数的地址,需要重新猜测。如果在猜测的过程中发生了再随机操作,那么猜测的难度将会大大增加。分析表明,在猜测的过程中进行再随机操作,将使得猜测难度以指数级增加。

这一类的 ASR技术从更细的粒度对进程空间进行了随机化,对栈、堆等模块的内部空间进行随机操作,并可能修改二进制代码中各个模块的相对位置。如果能够可靠地实现,这一类的 ASR将使得类似文献[14,15]的攻击变得非常困难甚至不可行。但由于其实现困难,目前没有得到广泛应用。

1.4 Vista中的 ASR

Windows Vista从 Beta2版本开始加入了 ASR功能 (ASLR address space layout randomization),Ollie Whitehouse M. Howard等人分别对

Vista的不同版本中的 ASR进行了研究[19—23]。Vista中的 ASR技术本来属于 1.2节中的一类,但由于 Vista的重要地位,这里单独分一节介绍。

从 Beta2版本开始,如果可执行文件被打上了 ASR标记,则进程的代码段基地址、栈基地址、堆基地址、PEB(process environment block)的位置等都在进程加载时被随机化了。进程初始化时,可执行文件(exe或者 dll)被随机加载到 256个地址之一;栈基址被随机选在 32个地址之一,然后栈顶指针随机向下移动 1—512个双字(4字节),共进行了 13位的随机化;PEB的随机化在 XP SP2和 2003 SP1中已经默认启用,被随机放在进程空间中 256个位置的中一个,且不需要对程序做任何设置;Ollie Whitehouse认为堆地址随机分配到 32个位置中的一个[19],但是实验结果表明远远多于 32个。其中,可执行程序和动态库的加载地址每次计算机重新启动就变化一次,其他是每次程序执行变化一次。

Vista中系统自带的程序和 DLL已经默认启用了 ASR功能,这些程序或者 DLL被加载时将会加载到随机的位置。自定义的程序要开启该功能,需要使用 Visual Studio 2005 SP1以上的编译器编译,并加上 /dynamicbase这一链接选项,编译器将会在生成的二进制程序中打上标记,Vista加载这些程序时将采用 ASR技术。同时,任何使用了包含标记的 DLL的程序,不管本身是否打过标记,使用的 DLL都将包含 ASR特征。

2 对 ASR的攻击

目前对 ASR技术的攻击主要针对 1.2中的实现方案,可能是因为这一类实现只随机化了基地址,且得到了广泛应用的缘故。

Tyler Durden演示了一种针对 PaX的攻击方法[14],因为 PaX只对主程序代码地址中 12—27位进行了随机修改,所以代码段中特定代码的页内地址与没有采用 PaX的系统中的页内地址相同,可以覆盖存在溢出漏洞的函数的返回地址的最后一个字节(页内地址的低 8位),使其跳转到本页内 call printf之类的指令,再通过格式化字符串攻击的方法,逐步获取共享库加载的基地址,从而实现 return-into-lib(c)攻击。这种攻击方法利用了 PaX只修改了基地址而没有修改主程序代码中语句的相对地址的漏洞,虽然只能在很特别的环境下才能实现,但是确实表明了 1.2节中的 ASR技术存在漏洞。

H. Shacham采用了暴力猜测 PaX中共享库的加载基地址的方法进行攻击[15],只要获得了该基地址,就可以由相对地址计算得到所有被加载的动态链接库中的函数的地址。因为 PaX只对该基地址进行了 16位的随机化,所以最多存在 65536种可能,平均进行 32768次猜测就可以猜出该地址,H. Shacham通过实验,用平均 216秒的时间就实现了 return-into-lib(c)攻击,得出以下结论:① 32位系统下 PaX中的 ASR技术只能降低蠕虫的传播速度,不能有效地抵御暴力破解,64位系统下该技术才可以使暴力破解变得不可能;② 猜测共享库中单目标函数地址时,实现再随机化最多使猜测时间增加一倍;③ 与运行时再随机化相比,在编译期间对共享库进行更细粒度的随机化更加有效,但是实现难度很大;④ 在某些重要的服务器系统中,即使检测到类似这种暴力攻击,也没有好的方法解决。这种攻击具有普遍意义,对于只需要事先确定单一目标地址的攻击,只有用增大随机空间的方法来解决,但是对于需要确定多个目标地址的攻击,在猜测的过程中如果进行再随机化,这种攻击将很难实现。另一方面,如果被攻击系统具有有效的攻击检测和处理机制,这种攻击就很容易被检测到,因为每次失败的猜测都将使程序跳转到不可预测的位置,而这可能会引起进程崩溃。

Ollie Whitehouse对 Windows Vista RTM版本中的 ASR技术进行了实验[19],统计了进程每次启动时的栈地址、代码加载地址、PEB地址和三个堆的地址,对结果进行分析后认为 PEB的位置和堆基地址的分布不是均匀的,使得猜测的难度大大降低,损害了 ASR的有效性。

同时,Windows Vista中的 ASR还存在其他技术缺陷:① 8位的随机空间比 PaX中的还要小很多,使得暴力破解更为简单;② 只有用 Visual Studio 2005 SP1以上编译器编译的程序才能被打上标记启用随机化,而不是所有程序默认启用;③ 主程序代码和动态库加载的基地址每次系统重新启动才变化,这样所有进程的主程序代码和动态库加载的基地址都是一样的,这可能带来安全问题。

3 实现 ASR系统的考虑

根据前面对现有 ASR技术和已知攻击方法的分析,这里提出一些实现高效可用的 ASR系统需要注意的一些问题。

• 实现的难度和系统开销

一种 ASR系统的实现难度不应过大,修改源代码或二进制代码的措施往往因为实现难度太大而得不到广泛应用。相对于其他抵御缓冲区溢出的技术,ASR带来的系统开销较小,但是如果频繁地调整进程的空间布局,也会让用户难以接受。

• 随机空间应该足够大

在没有其他漏洞的情况下,攻击者将会采用暴力猜测的方法对 ASR系统进行攻击,分析表明,若要抵御这种攻击,随机的位数应该在 20位以上。

• 随机化的粒度

随机化的粒度应尽可能细,以抵御基于相对地址的攻击,由前面的分析可知,只随机化进程各个模块加载的基地址是不够的,需要对各个模块内部元素间的间隔进行进一步的随机化,同时需要考虑实现的难度。

• 结合其他防护措施

ASR系统应和其他安全防护系统如入侵检测系统结合使用,当攻击者猜测随机值错误时,能及时发现攻击并作出响应。

4 结束语

ASR是一种新的抵御已知的和未知的基于缓冲区溢出安全漏洞的攻击的方法,它能有效地降低攻击者成功的概率。随着 Windows Vista操作系统的推出,目前主流的操作系统都使用了 ASR技术来增强安全性。但是目前得到广泛应用的 ASR技术的实现还不全面,还存在被暴力破解和被基于相对地址的攻击攻破的可能,更高级的对相对地址进行随机化的方法由于实现难度高尚没有得到推广。找到易于实现的细粒度随机化方法可能是 ASR技术未来的研究重点。而对 ASR技术进行攻击的重点将会集中在对 Windows Vista中的 ASR技术的分析和破解上。

参 考 文 献

[1] Aleph One. Smashing the stack for fun and profit[J/OL]. Phrack Magazine, 1996, 7(49).

[2] Michel Kaempf. Vudo malloc tricks[J/OL]. Phrack Magazine, 2001, 11(57).

[3] Snort(tm) advisory: Integer overflow in stream4[EB/OL]. 2003—04. http://www.kb.cert.org/vuls/id/JPLA-5LPR9S.
[4] Scut. Exploting format string vulnerabilities[EB/OL]. 2001—03. http://www.team-teso.net/articles/formatstring.
[5] Anonymous. Once upon a free()[J/OL]. Phrack Magazine, 2001, 9(57).
[6] Nergal. The advanced return-into-lib(c) exploits[J/OL]. Phrack Magazine, 2001, 11(58).
[7] Cowan C, Pu C, Maier D, et al. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks[C]//In USENIX Security Symposium, San Antonio, Texas, 1998: 63-78.
[8] Cowan C, Barringer M, Beattie S, et al. FormatGuard: Automatic protection from printf format string vulnerabilities[C]//In USENIX Security Symposium, Washington, DC, 2001.
[9] PaX Team. PaX[EB/OL]. 2001. http://pax.grsecurity.net.
[10] Forrest S, Somayaji A, Ackley D H. Buildling diverse computer systems[C]//In 6th Workshop on Hot Topics in Operating Systems. Los Alamitos: IEEE Computer Society, 1997: 67-72.
[11] Chew M, Song D. Mitigating Buffer Overflows by Operating System Randomization[R]. Technical Report CMU-CS-02-197, Carnegie Mellon University, 2002.
[12] Alexander S. Improving Security With Homebrew System Modifications[J]. USENIX, 2004, 29(6): 26-32.
[13] Jun Xu, Zbigniew Kalbarczyk, Ravishankar K Iyer. Transparent runtime randomization for security[R]. Technical Report UILU-ENG-03-2207, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, 2003.
[14] Tyler Durden. Bypassing pax aslr protection[J/OL]. Phrack Magazine, 2002, 11(59).
[15] Shacham H, Page M, Pfaff B, et al. On the effectiveness of address-space randomization[C]//In ACM conference on Computer and Communications Security (CCS), Washington, DC, 2004: 298-307.
[16] Bhatkar S, DuVarney D C, Sekar R. Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits[C]//In Proceedings of the 12th USENIX Security Symposium, Washington D. C, 2003.
[17] Bhatkar S, Sekar R, DuVarney D C. Efficient techniques for comprehensive protection from memory error exploits[C]//In Proceedings of the 14th USENIX Security Symposium, Baltimore MD, 2005.
[18] Haizhi Xu, Steve J. Chapin. Improving Address Space Randomization with a Dynamic Offset Randomization Technique[C]. In SAC '06 April 2327, 2006, Dijon, France.
[19] Ollie Whitehouse. An Analysis of Address Space Layout Randomization on Windows Vista[R/OL]. http://www.symantec.com/avcenter/reference/Address_Space_Layout_Randomization.pdf.
[20] Howard M. Address Space Layout Randomization for Windows[EB/OL]. 2005. http://blogs.msdn.com/michael_howard/archive/2005/09/30/475763.aspx.
[21] Howard M. Address Space Layout Randomization in Windows Vista[EB/OL]. 2006. http://blogs.msdn.com/michael_howard/archive/2006/05/26/608315.aspx.
[22] Howard M. Alleged Bugs in Windows Vista's ASLR Implementation[EB/OL]. 2006. http://blogs.msdn.com/michael_howard/archive/2006/10/04/Alleged-Bugs-in-Windows-Vista_1920_s-ASLR-Implementation.aspx.
[23] Symantec Advanced Threat Research. Security Implications of Microsoft Windows Vista[R/OL]. http://www.symantec.com/avcenter/reference/Security_Implications_of_Windows_Vista.pdf.

(上接第 6页)

的防护策略的实际意义与此类似,由 Nash均衡的意义[6]知:攻击者或者管理员单方面地改变各自的策略都不可能获得更好的收益,所以攻防双方都会理性地坚持自己的 Nash策略。与 NEAG相比,攻击图的分析方法给出的结果却是①、②、③,这是因为攻击图忽略了攻击者会综合考虑攻击成功的获益以及攻击失败的后果。从状态②到状态③的攻击如果失败,攻击者需要付出比较大的代价,在这种情况下,攻击者会理性地选择从状态②到状态④再到状态⑤的攻击。可见,NEAG算法给出的结果更加符合现实状况。

4 结　论

当前安全分析算法未考虑管理员对自身网络设备的重要性评定,及其在此评定基础上的防护行为对网络安全状况的影响,针对该问题提出了一种基于随机博弈模型的网络安全量化评估算法 NEAG。在假定攻防双方均是理性的前提下,NEAG算法能够根据攻防双方对收益和代价的综合考虑,给出攻防双方的 Nash策略。博弈论指出,攻防双方为获得最佳收益,会理性地坚持自己的 Nash策略,而传统的网络安全评估算法则假定攻击者会选择攻击成功概率最大的行为,忽略了管理员对网络的防护行为,以及攻击者在攻击成功的收益与攻击代价之间的权衡,导致其给出的结果与现实网络状况有出入。NEAG算法根据所获得的网络的具体信息,对网络可能的状态进行分析,给出攻防双方的最佳策略,指导管理员的防护工作,因此,与传统方法相比,NEAG算法具有更好的通用性和准确性。

参 考 文 献

[1] Phillips C, Swiler L P. Graph-based A. System for Network-Vulnerability Analysis[C]//Proceedings of the 1998th workshop on New Security Paradigms. Charlottesville, Virginia, United States, 1998. New York, NY, USA: ACM, 1998: 71-79.
[2] 夏阳,陆余良.计算机主机及网络脆弱性量化评估研究[J].计算机科学, 2007, 34(10): 74-79.
[3] Lye K, Wing J. Game Strategies in Network Security. International Journal of Information Security, 2005, 4(1-2): 71-86.
[4] Karin Sallhammar, Bjarne E Helvik, Svein J Knapskog. Towards a Stochastic Model for Integrated Security and Dependability Evalution[C]//Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06), Vienna, Austria, Apr 20—22 2006. Piscataway, NJ 08855—1331, United States: Institute of Electrical and Electronics Engineers Computer Society, 2006: 156-165.
[5] Karin Sallhammar, Bjarne E. Helvik, Svein J. Knapskog. A Game-theoretic Approach to Stochastic Security and Dependability Evalution[C]//Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC'06), Indianapolis IN, United States, Sep 29-Oct 1 2006. Piscataway, NJ 08855-1331, United States: Institute of Electrical and Electronics Engineers Computer Society, 2006: 61-68.
[6] Owen G. Game Theory[M]. 2nd ed. Academic Press, 1982.
[7] 哈罗德·W·库恩.博弈论经典[M].中国人民大学出版社, 2002.
[8] http://www.nessus.org.
[9] http://cve.mitre.org.