FreeBSD的地址空间随机化

罗宇翔 梁洪亮
(中国科学院软件研究所 北京 100080)

摘 要 缓冲区溢出是一种最为常见的安全漏洞形式,在远程网络攻击中所占比重最大。地址空间随机化是一种针对缓冲区溢出攻击的有效防护技术。分别从栈、堆、动态库和可执行映像四个方面描述了地址空间随机化在 FreeBSD 6.0中的设计和实现,并对其防护果进行了评估。

关键词 地址空间随机化 栈随机化 缓冲区溢出

ADDRESS SPACE RANDOMIZATION IN FREEBSD

Luo Yuxiang Liang Hongliang
(Institute of Software, Chinese Academy of Science, Beijing 100080, China)

Abstract Buffer overflow is one of most known security holes, and most of the remote network attacks are originated from it. Address space randomization is an effective defending technique against buffer overflow attacks. In this paper it describes the design and implementation of the address space randomization in FreeBSD 6.0 from four aspects: stack, heap, dynamic library and executable image. The effectiveness evaluation of defence is also given.

Keywords Address space randomization Stack randomization Buffer overflow

0 引 言

缓冲区溢出一直是一种重要的安全漏洞。据统计,在所有的软件安全漏洞中,缓冲区溢出漏洞大约占一半[1]。缓冲区溢出漏洞主要被利用来发起远程网络攻击。为应对这种攻击,各种网络服务器通常是在网络层提供各种安全加固和防护措施,在某种缓冲区溢出漏洞被发现之后再打补丁。但这些方法并不能非常有效地阻止缓冲区溢出攻击。在国家相关项目支持下,我们在研制符合 GB17859-1999第三级的安全操作系统时,对如何在操作系统层次上增强系统的抗缓冲区溢出攻击能力进行了研究。

本文给出了我们将地址空间随机化方法引入 FreeBSD 6.0的设计和实现。

1 相关工作

缓冲区溢出攻击主要分为以下几类:栈溢出、堆溢出、格式化字符串漏洞和整型变量溢出等。针对这些攻击的方法归纳起来有:编译保护、libwrapper,缓冲区不可执行、指针保护、数组边界检查、指令集随机化、地址空间随机化、安全操作系统内核和硬件加固等等。

编译保护是通过扩展编译器以防止函数的返回地址被修改来实现堆栈保护。通常是在返回地址前放置一个标志(canary)字或者建立一个影子堆栈,在函数返回前查看是否有溢出存在。目前已有不少补丁可供使用,如 StackGuard[2]和 StackShield[3]。但这种方法只局限于对栈的保护。

缓冲区不可执行包括栈不可执行和数据段不可执行。攻击者通常是将恶意代码植入溢出的缓冲区中,只要让这些可写的缓冲区不可执行,就能防止恶意代码接管程序控制。尽管这种方法能极大地提升攻击的难度,但它无法阻止像 return-to-libc这种无须注入代码的攻击。采用这种技术的典型例子有 Linux下的 ExecShield[4], PAX[5]。

指令集随机化是指编译时对可执行程序的机器码进行加密,在指令执行前,对每条加密的指令进行解码。通过这个过程,使得攻击者植入的恶意代码不能完成预定的功能,绝大多数的情况下会导致程序崩溃。但这种方法有很多缺点[6]。

地址空间随机化是通过对操作系统内核或者 C库的修改,使进程加载到内存的地址随机化,从而降低攻击成功的概率。黑客如果想控制系统,要么需要修改返回地址为指定的值(如栈溢出和 ret-into-libc攻击),要么需要修改指定地址为指定的值(如 ret-into-libc攻击和.plt/.got覆盖)。地址随机化之后,这些指定地址或者指定值都将无法事先确定。相对上述方法而言,这种方法更为一般化,它试图阻止几乎所有类型的内存攻击。PAX实现了 Linux下的地址空间随机映射,OpenBSD[7]也做了某些相关工作。

2 FreeBSD 6.0中地址空间随机化的设计和实现

2.1 FreeBSD可执行程序的装载过程简介

FreeBSD内核内存管理子系统为可执行程序设置好相应的内存区域(栈、代码、数据、BSS),通过输入/输出(I/O)子系统将

收稿日期:2006-10-24。"十五"国家科技攻关计划项目支持(2005BA113A02)。罗宇翔,硕士生,主研领域:系统软件和信息安全。

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

可执行程序的代码段和数据段与内存区域进行关联。如果是动态链接的可执行程序,内核会随后从可执行文件中获取解释器的配置,然后将解释器映射到可执行程序的内存空间,最后由解释器将所有依赖的动态库通过 mmap装入可执行程序的内存空间。具体的过程如图 1所示。

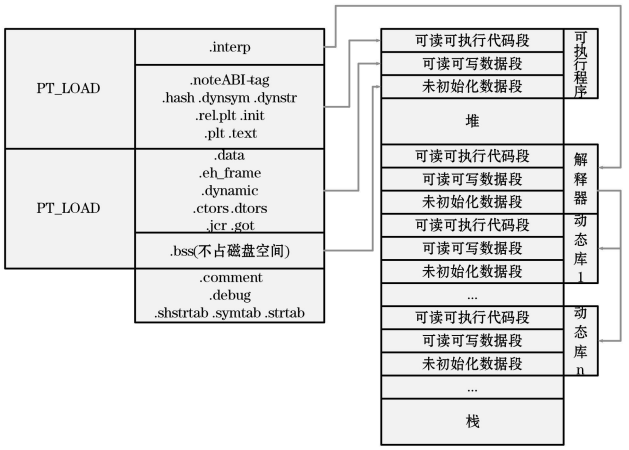

图 1 可执行文件的装载

2.2 栈的随机化

栈是操作系统内核为程序建立的。在 exec_copyout_strings函数里,有以下代码片断(/usr/src/sys/kern/kern_exec.c):

```
1017  arginfo=(struct ps_strings *)p->p_sysent->sv_psstrings;
1018  if (p->p_sysent->sv_szsigcode!=NULL)
1019    szsigcode=*(p->p_sysent->sv_szsigcode);
1020  destp=(caddr_t)arginfo-szsigcode-SPARE_USRSPACE-
1021    roundup((ARG_MAX-imgp->stringspace), sizeof(char *));
```

其中,szsigcode表示用于存储所有信号处理例程指针的内存大小,ARG_MAX - imgp->stringspace表示存放参数和环境的内存大小。这个片断之后的代码从 destp开始依次向高地址复制这些字符串和信号处理例程指针。假设初始栈指针为 vectp,在 vectp和 destp之间还会依次存放每个参数和环境字符串的指针。当控制权由内核交给用户程序时,局部变量的地址就从初始栈地址 vectp开始向低地址延伸。因此,我们只要紧接着将这个 destp减小一个随机值,或者在该函数返回前将 vectp减小一个随机值,就能实现我们的目标。考虑到效率,随机地址最好能按 Cacheline对齐。

2.3 堆的随机化

堆是由库函数 malloc分配管理的。我们不关心 malloc的内存管理算法,而只关心 malloc是如何初始化堆的。进程在首次调用 malloc时,malloc会通过 sbrk(0)来获得堆的起始地址。但是 sbrk的实现并不是从内核中获得的(Linux是采用这种做法),而是在 libc中,并且有汇编语言和 C语言两个版本,汇编语言版本才是系统使用的。它通过两个全局变量.minbrk和.curbrk来确定堆的范围,其中.minbrk表示当前堆的初始地址,.curbrk表示当前堆的结束地址。这两个值都在程序编译连接的过程中初始化为 BSS的结束地址(见图 1)。

因此,我们只需要把.minbrk和.curbrk的初始值向后挪一个随机值,就能实现堆初始位置的随机化。虽然.minbrk和.curbrk都是全局变量,但在 C语言中无法直接访问。通过编写一个汇编程序对它们进行修改,并在 malloc的初始化代码中调用该函数,将随机数传递给它,可以达到随机增加这两个变量的目的。

2.4 动态库映射的随机化

动态库的装载由解释器实施,解释器通过分析可执行程序获得其依赖的动态库列表之后,依次打开这些动态库,并通过 mmap将动态库的代码段和数据段映射到相应的内存空间中。由于动态库本身就是位置无关代码(PIC),解释器通过 mmap加载时将指定 addr参数为 0,交给内核来确定加载地址。因此,在内核 mmap函数中,我们关注的是以下逻辑流程:

```
272     if (addr==0 ||
273       (addr>=round_page((vm_offset_t)vms->vm_taddr) &&
274       addr<round_page((vm_offset_t)vms->vm_daddr+
275     lim_max(td->td_proc, RLIMIT_DATA))))
276       addr=round_page((vm_offset_t)vms->vm_daddr+
277         lim_max(td->td_proc, RLIMIT_DATA));
...
360   error=vm_mmap(&vms->vm_map, &addr, size, prot, maxprot,
361     flags, handle, pos);
```

如果 addr=0,那么先令 addr=数据段开始+数据段长度的最大值,然后传给 vm_mmap作为 mmap开始地址的最小可能值。如果把这个最小可能值随机加大,那么 mmap的初始地址也会随机加大。再进一步,为了防止 ret-to-libc攻击,我们将 mmap的地址尽量放在内存的前 16M,使地址字节中包含 0。

2.5 可执行映像的随机化

程序自身的随机化,又称位置无关可执行程序(PIE)。在 Linux系统中,实现 PIE需要内核、GCC、连接器和 C库四个部分的共同协作:

1) 内核支持直接运行共享代码;

2) GCC需要支持 -fpie选项,编译的时候指导 ld选择正确的 C库初始化目标代码;

3) 连接器 LD需要支持 -pie选项,连接的时候生成 PIE;

4) Libc需要有 PIE相关的 C库初始目标代码。

相应地,在 FreeBSD中,我们需要作如下处理:

1) FreeBSD内核不支持直接运行共享代码,因此需要在 elf32_imgact中添加对 ET_DYN类型的支持。装载 PIE时需要先确定基址,我们采用的方法是固定地址加上随机偏移量。在其后对程序每节的重定位时也要加上这个基地址。所有这些计算出的值都被保存在此次运行的 augargs中,解释器在加载动态库以及确定程序入口点时,需要用到该辅助向量中的信息。

2) FreeBSD的 GCC使用的不是官方的 GCC,对 -fpie选项支持不完全,在指导 ld的时候,选择了不同的 CRT初始化目标代码。为了指导 ld在有 fpie选项时能选择正确的 C库初始化代码,需要修改 GCC的 spec文件。

3) FreeBSD的标准连接器 ld已经支持 -pie选项。

4) 由于 FreeBSD本身不支持 PIE,因此没有 Linux下连接 PIE需要的 Scrt1.o crtbeginS.o crtendS.o。我们需要用 PIC的方式编译 crt1得到 Scrt1.o,使用新版本的 GCC得到 crtbeginS.o和 crtendS.o。

实现了以上的四个部分的工作,FreeBSD就可以实现 PIE了。

2.6 其它问题

还有两个问题需要解决:如何在用户态控制地址随机化的开启和随机范围的指定?如何获得随机数?我们在内核里实现了 SYSCTL的一个子项,其下再设几个调整项:apply_exec_random

(下转第 13页)

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

算法 1　基于案例查询处理策略的实现算法

```
For Peer P (querying Peer)
void InitiateQuery( query )
{
  results += QueryInLocalPeer( query );
  SendQueryToNeibs( query );
  results += WaitForResults( timeslot );
  RankingResults( results );
  DisplayResultsToUser( results );
}
For Peer X(queried Peer):
void RecieveQuery( query )
{
  if( isThereAnyRelatedResults( query ))
{
  results= ComputeSimilarity( query );
    }
    ReSendQueryToNeibs( query );
    for( resultA in results )
    {
      if(SR(resultA, query) > threshold )
        {
          SendResultToQueryInitiat
    or( resultA );
        }
    }
}
```

4 结　论

本文为 P2P信息检索系统提出了一种基于案例查询的处理策略，能够更精确地满足用户信息需求，同时更有效地利用系统计算资源和带宽资源。深入研究了该策略的理论机制，进行了相关的理论推导，阐述了该策略的处理过程及实现算法。这些成果为进一步研究 P2P信息检索夯实了基础。接下来将进行实验分析，进一步揭示 P2P信息检索的机理，并增强系统功能。

参 考 文 献

[1] Wee Siong Ng, Beng Chin Ooi, KianLee Tan, PeerDB: A P2P based system for distributed data sharing. In Proceedings of ICDE, 2003: 633-644.

[2] 凌波，陆治国，黄维雄，钱卫宁，周傲英. PeerIS:基于对等计算的信息检索系统.软件学报，2004, 15(9): 1375-1384.

[3] PlanetP project homepage. http://www.panic-lab.rutgers.edu/Research/planetp.

[4] Tang C, Xu Z, Mahalingam M. Psearch: Information retrieval in structured overlays. In Proceedings of the 1st ACM HotNets-I, 2002.

[5] Raghavan V V, Wong S K M. A Critical Analysis of Vector Space Model for Information Retrieval. Journal of the Amerivan Society for Information Science, 375(3): 279-287.

(上接第 2页)

为总开关，random_stack_range、random_heap_range、random_so_range代表地址随机化的最大范围，random_so_base代表动态库加载的搜寻起始地址。对于随机数，我们使用 /dev/random的实现函数 random_yarrow_read (/usr/src/sys/dev/random/yarrow.c)来获得。

3 效果评估

由于程序装载的位置不再固定，因此使用缓冲区溢出攻击程序成功的概率将大大降低。攻击成功概率计算如下：

(1)栈溢出成功的概率为：

$p=(L(buf)-L(shellcode))/(L(random\_stack\_range)-L(shellcode))$

若 L(buf)=256, L(shellcode)=50, L(random_stack_range)=1M，则 p=0.01%。

(2)堆溢出.plt/.got成功的概率为：

$p=L(int)*4096/L(random\_range)$

若 L(int)=4, L(random_range)=1M，则 p=1.5625%。

(3) ret-into-lib成功的概率为：

$p=(L(int)*4096/L(random\_range))*(L(int)/L(random\_range))$

若 L(int)=4, L(random_range)=64K，则 p=0.001%。

其中，L(a)代表 a的长度。

另外，我们对实施地址随机化后的操作系统进行了攻击实验，无一例攻击成功。尽管如此，地址空间的随机化并不能完全阻止缓冲区溢出攻击。

4 结　论

本文描述了地址空间随机化方法在 FreeBSD6.0中的设计和实现，并对其防护效果进行了理论估算和实际测试。结果表明，地址空间随机化技术是一种一般化的、试图阻止各种内存攻击的有效技术。同时，尽管它能有效降低缓冲区溢出攻击成功的概率，但不能完全阻止。如果采用 64位体系结构，能使攻击成功的概率变得非常低。我们下一步的工作将研究如何把地址随机化与其他安全机制有效结合，构建一套完整的缓冲区溢出防御体系。

参 考 文 献

[1] Cowan C, Wagle P, Pu C, et al. Buffer Overflows: Attacks and Defenses for the Vulnerabili-ty of the Decade[EB/OL]. http://www.cse.ogi.edu/DISC/projects/immunix/discex00.pdf, 2000-1.

[2] Perry Wagle, Crispin Cowan. StackGuard: Simple Stack Smash Protection for GCC. http://gcc.fyxm.net/summit/2003/Stackguard.pdf.

[3] Vendicator. Stack Shield. http://www.angelfire.com/sk/stackshield/. January 7, 2000.

[4] ExecShield. http://people.redhat.com/mingo/exec-shield/.

[5] PAX. http://pax.grsecurity.net/.

[6] Gaurav S Kc, Angelos D Keromytis, Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. Proceedings of the 10th ACM conference on Computer and communications security. October 27-30, 2003, Washington D.C., USA.

[7] OpenBSD. http://www.openbsd.org.

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net