

一种基于随机匹配算法的堆溢出防范策略

彭 锋¹, 沈向洋²

(1. 中国科学院 计算技术研究所, 北京 100080; 2. 微软亚洲研究院, 北京 100080)

(PengFeng2000@gmail.com)

摘 要:对已有的溢出防范技术及其存在的问题进行分析, 提出了一种基于随机匹配算法的堆溢出防范策略。通过在内存管理系统中引入更多的随机性, 使攻击者无法准确预测将要覆盖的内容, 从而达到防范堆溢出攻击的目的。

关键词:缓冲区溢出; 内存管理; 随机匹配

中图分类号: TP309 **文献标识码:** A

New method to avoid heap overflow using random fit

PENG Feng¹, SHENG Xiang-yang²

(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China;

2. Microsoft Research Asia, Beijing 100080, China)

Abstract: Based on current methods of anti-buffer overflow and the analysis of existing problems, the random fit algorithm was presented. By introducing randomization into memory management algorithm, the likelihood that the attacker can predict the content to be overwritten was reduced.

Key words: buffer overflow; memory management; random fit

0 引言

缓冲区溢出漏洞在 Windows、Unix、Linux 等操作系统上都存在, 是众多病毒和蠕虫攻击的首选对象。缓冲区溢出按照溢出位置可分为栈溢出和堆溢出两种。随着栈探测^[3,4]等溢出防范技术的日趋成熟, 栈溢出的问题逐步得到解决。与此同时, 堆溢出的问题更显得严峻, 如果能有效防范堆溢出, 则很多安全问题可以得到缓解甚至消除。本文在现有堆溢出防范方法的基础上, 从内存分配算法的角度提出了一种新的溢出防范策略, 即通过随机分配堆内存, 使一个分配序列中前后内存块的相对位置变得不可预测, 从而达到防范堆溢出的目的。

1 堆溢出攻击原理和防范

1.1 堆溢出攻击原理

堆溢出是指向堆上缓冲区进行写操作时写入的数据量大于缓冲区长度, 从而覆盖掉其后紧邻区域的内容。攻击者可以利用这种覆盖来改写一些敏感数据, 以达到改变目标程序行为的目的。以具体覆盖对象来区分, 堆溢出攻击又可分为覆盖堆管理数据和覆盖紧邻内存块数据两种。

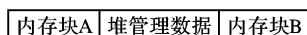


图 1 堆中内存块分布

图 1 是堆中一段内存分布示意图。在动态分配的内存块 A 和 B 间是一段堆管理数据。向内存块 A 中写数据时, 如果越过了它的边界就会覆盖掉堆管理数据的内容, 从而产生了溢出^[5]。如果进一步越过堆管理数据边界, 则可以覆盖掉紧邻的内存块 B 中的数据^[6]。堆溢出攻击利用应用程序编写中

的漏洞, 通过堆溢出直接改写敏感数据如函数指针等, 或先覆盖某个指针使其指向攻击者想修改的内存区域, 再通过该指针向这块内存区域写入恶意数据以达到攻击者的目的。

1.2 堆溢出攻击防范

溢出攻击防范可以从编译时静态分析和运行时动态防范两个方面入手, 常见的防范方法有如下几种:

1) 静态分析和边界检查

缓冲区溢出本质上是由程序自身缺陷引起的, 因此最好的防范方法就是杜绝程序的缺陷。一种常见的方法是替换不安全的 C 函数, 如把 strcpy, sprintf 等函数分别替换为 strncpy, snprintf 等。此外还有许多静态分析工具帮助程序员在编译阶段发现程序缺陷, 如 PCLint、Prefast 等。但时至今日, 这些工具还不够完善, 时有误报, 也不能发现系统中所有的漏洞^[7]。

2) 增强系统内核预防缓冲区溢出

系统内核并不知道应用程序会如何执行, 完成何种功能, 因此只能从运行环境上来预防。一般而言恶意代码是作为应用程序的输入注入到目标进程的。输入会存放在栈或堆上, 因此可以通过禁止执行栈和堆上的代码来提高系统的安全性。文献[8]、[9]的 Linux kernel patch 就是 Linux 下这样的系统补丁。但这种方法有如下缺点: 1) 有些情况下可执行代码必须放在栈中, 如 Linux 通过向进程栈填写代码然后引发中断来执行栈中的代码以便实现向进程发送信号。2) 攻击者可以将代码放在堆和栈之外的区域。例如文献[10]中提出了绕开“不可执行的栈”的攻击方法。3) 通过改写某些敏感数据如 system 函数的参数, 不用注入代码就可以达到攻击目的。而堆栈不可执行的防御方法对此种攻击手段无能为力。

收稿日期: 2006-04-14; 修订日期: 2006-06-08

作者简介: 彭锋 (1979-), 男, 湖北宜昌人, 硕士研究生, 主要研究方向: 计算机图形学、网络安全; 沈向洋 (1966-), 上海人, 教授, 博士, 主要研究方向: 计算机视觉与图形学。

力。

3) 随机化堆栈数据

溢出攻击的第一步是通过溢出达到对紧邻内存区域的覆盖。被覆盖的区域必须是可以预测且可用来实施某种攻击的。因此通过随机化堆栈上的数据分布,使攻击者无法预测其能覆盖的内容就可以达到预防溢出攻击的目的。文献[11]提出的地址混淆(Address Obfuscation, AO)就是这样一种方法。AO通过随机分布动态库、栈和堆数据等达到防范溢出攻击的目的。具体地讲,是通过改写 mmap、malloc 等系统调用实现对动态库、可执行程序以及堆、栈等基址的修改,从而实现对堆栈和程序数据的随机化。

AO对堆中内存块的随机化是通过挂接 malloc 函数,对用户请求的块长度随机增加 0%~25%来实现的。这种方法存在如下不足:首先,考虑到内存块的对齐要求,用户申请的内存块太小时,随机增加长度之后的内存块可能还在对齐范围以内,没有达到随机化的效果。其次,用户申请内存太大时会浪费大量内存。最后,很多情况下分配的内存块只是相对距离变远了,相对关系并未没变。攻击者通过向前一块内存填充大量重复数据,依然可以覆盖后续内存块,达到攻击目的。

2 基于随机匹配的堆溢出防范策略

2.1 内存管理算法概述

本文提出的方法也是通过随机化堆内存块分布达到防范溢出的目的。但不同于 AO 算法中简单修改 malloc 函数行为的方法,本文完整地实现了一个基于随机匹配(Random Fit)算法的内存管理系统,并在该系统引中引入更多的随机化因素,从而达到更好的随机化效果。

一般内存管理系统都是先分配一大块内存(chunk),保留这块内存起始部分用于放置管理数据,后面部分做为空闲块分配给用户使用。在管理数据中有一个空闲链表表头数组,数组中每一项元素都是一个双向链表的表头。该双向链表由若干特定大小的空闲块构成,空闲块大小随数组索引递增。此外管理数据中还维护一个大空闲块链表的表头,超过一定大小的空闲块均存放在这个链表中。

当用户向系统申请内存块时,系统先检查相应的空闲链表,按照一定的匹配规则寻找大小合适的内存块返回给用户,若未找到则从大空闲块链表中找到一个长度满足需求的空闲块,分割后一部分返回给用户,另一部分插入到相应的空闲块链表中。这里使用的分配规则都是确定的,对于一个给定的内存分配和释放序列,总是返回相同的匹配结果,这就给预测堆溢出可覆盖的内容提供了方便。随机匹配算法正是通过随机匹配来消除这种确定性以达到防范堆溢出攻击的目的。

2.2 随机匹配算法描述

如图 2 所示,随机匹配算法中使用的空闲链表表头数组包含 128 个元素,链表中能容纳的最大内存块长度为 1024 字节。算法对不同大小的内存块采用不同的随机化方法。

算法描述:

1) 检测用户请求的内存块长度 n ,长度不大于 1024 字节时向下执行,否则转 5);

2) 请求的块长度不大于 1024 字节时,检查对应的空闲链表。判断链表中空闲块数目是否大于预先设定的阈值。大于阈值则向下执行,否则转 4);

3) 从空闲块链表中随机取出一个空闲块返回给用户;

4) 从大空闲块链表上取一个长度不小于 $n + 512$ 字节的空闲块,随机将该空闲块切分为两部分。满足其中一部分长度不小于 n 字节并将其返回给用户,另一部分放入相应的空闲块链表;

5) 申请一个长度为 $n + 1024$ 字节的大内存块,在大内存块中随机取一个满足其后空间不小于 n 字节的位置返回给用户。

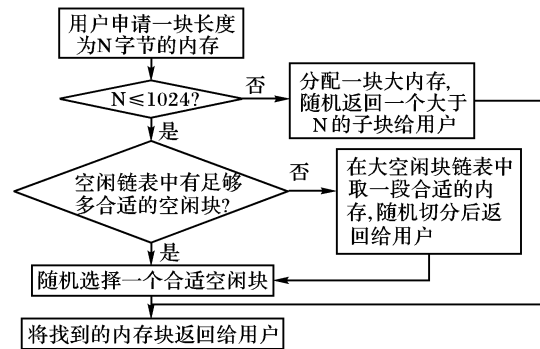


图2 随机匹配算法

在算法实现中需要注意,C库的 rand 函数产生的并不是真正意义上的随机函数,而是一个伪随机函数。如果没有用随机种子初始化,其产生的实际上是一个确定的伪随机数序列。为了得到更好的随机化函数,可以用时间、内存使用量、按键状态等参数作为随机数生成器的种子,以达到更好的随机化效果。

3 攻击测试及分析

3.1 攻击测试

为了检验随机匹配算法对堆溢出攻击的防范性能,我们进行了攻击实验。实验是在英文 Windows XP + SP2 系统上进行的,使用 Visual C++ 2005 作为开发工具。分别使用 C 运行时库中的 malloc 函数、经 AO 方法修改过的 malloc 函数和本文提出的随机匹配算法分配两块长度为 28 字节的内存块。攻击程序试图向第一块内存写入超长字符串,造成其溢出以达到将特定数据写入第二块内存的目的,如果能将数据写入第二块内存,则攻击成功。测试代码如下:

```
// 用宏开关把 kMALLOC, FREE 映射到要测试的函数上
```

```
#define MALLOC malloc
```

```
#define FREE free
```

```
int main()
```

```
{
```

```
char * p1 = (char *) MALLOC(28);
```

```
char * p2 = (char *) MALLOC(28);
```

```
p2[0] = 0;
```

```
sprintf(p1, "%s", "testtesttesttesttesttesttesttesthack");
```

```
// 向第一个缓冲区写入 32 + 8 + 4 = 44 个字节;
```

```
// 使用 8 字节对齐, 28 对齐到 32 字节;
```

```
// 随后的 8 字节为堆管理数据长度;
```

```
// 如 p1, p2 紧邻则 p2 前四个字节被改写;
```

```
if (strcmp(p2, "hack") == 0)
```

```
{
```

```
printf("I've been hacked");
```

```
}
```

```
FREE(p1);
```

```
FREE(p2);
```

```
return 0;
```

```
}
```

测试结果见表 1。

表1 攻击实验结果

内存系统	攻击次数	成功次数
C 运行时间库	10 000	10 000
地址混淆	10 000	7 150
随机匹配	10 000	25

3.2 实验结果分析

AO 算法在分配长度为 28 字节的内存块时,理论上随机增加的长度值在 0~6 之间分布。当该值为 0~4 时,相当于没有随机化,其概率:

$$P_{theory} = 5/7 = 71.43\%$$

实际概率:

$$P_{practise} = 7150/10000 = 71.50\%$$

理论值与实际值很吻合。

对随机匹配算法,由于随机化时没有 1/4 内存块大小的限制,且两个块之间的距离由两次生成的随机数决定,理论上可以取到整个 chunk 中任意两个大小满足要求且没有重叠的块。从长度为 N 字节的内存区间中取两个长度为 L 字节的内存块,设两个内存块所有可能的位置组合数目为 D ,相对距离有 R 种可能。当 $N > 3L$ 时有:

可能的位置组合数目:

$$D = \sum_{0 \leq i \leq L-1} (N-i-2L+1) + \sum_{L \leq i \leq N-2L} ((i-L+1) + (N-i-2L+1)) + \sum_{N-2L+1 \leq i \leq N-L} (i-L+1) \\ = (N-2L+2)(N-2L+1)$$

相对距离数:

$$R = 2(N-2L+1)$$

攻击者猜中两块相对距离则攻击成功。一次攻击猜中相对距离的平均概率:

$$P_{(avg)} = 1/R = 1/((N-2L+1) \times 2)$$

最大概率(两块紧邻):

$$P_{max} = (N-2L+1)/D = 1/(N-2L+2)$$

最小概率(两块分别在 chunk 首尾):

$$P_{min} = 1/D = 1/((N-2L+1)(N-2L+2))$$

考虑到内存分配的边界对齐因素,设分配的内存块按 A 字节对齐,以上公式中的 N, L 分别应该用 $N/A, L/A$ 替换。

在测试用例中 $N = 2512, L = 16, A = 8$, 解得 $P_{avg} = 0.16\%, P_{max} = 0.31\%$, 与实际数据 0.25% 较吻合。

由以上分析和测试结果可见随机匹配算法可以有效防范

堆溢出攻击。

4 结语

本文提出的随机匹配算法在分配小内存块时不会退化为确定性算法,分配大内存块时也没有太多额外的空间开销,并且随机化非常彻底,能有效降低堆溢出攻击成功概率。

由于攻击手段的多样性,只依赖单一的防范措施是远远不够的,必须综合应用多种防范手段,才能达到更好的安全性。就内存块随机化来讲,目前只实现了堆上数据的随机化,将来的工作中还包括针对栈、静态数据和动态链接库地址的随机化进行研究。

参考文献:

- [1] SALKEVER A. The Ever-Growing Virus Crisis [J/OL]. http://yahoo.businessweek.com/technology/content/aug2003/tc20030826_4386_tc047.htm, 2003.
- [2] GCC documents. Specifying How Stack Checking is Done [CP/OL]. <http://gcc.gnu.org/onlinedocs/gccint/Stack-Checking.html>, 2005.
- [3] MSDN. Visual C++ Compiler Options [CP/OL]. <http://msdn2.microsoft.com/en-us/library/9598wk25.aspx>, 2005.
- [4] CONOVER M. w00w00 on Heap Overflows [R/OL]. <http://www.w00w00.org/files/articles/heaptut.txt>, 1999.
- [5] CONOVER M, HOROVITZ O. Reliable Windows Heap Exploits [R/OL]. <http://www.cybertech.net/~sh0ksh0k/heap/CSW04%20-%20Reliable%20Windows%20Heap%20Exploits.ppt>, 2005.
- [6] HOARE T. 21 世纪的智能编译器 [R/OL]. http://research.microsoft.com/asia/dload_files/21century/3_tony_hoare_E.pdf, 2001.
- [7] Pax Documentation [CP/OL]. <http://pax.grsecurity.net/docs/pax.txt>, 2003.
- [8] DESIGNER S. Non-executable user stack [R/OL]. <http://www.openwall.com/>, 2000.
- [9] WOJTCZUK R. Defeating Solar Designer's Non-executable Stack Patch [R/OL]. http://www.insecure.org/spl0its/non-executable_stack.problems.html, 1998.
- [10] BHATKAR S, DUVARNEY D, SEKAR R. Address obfuscation: An efficient approach to combat a broad range of memory error exploits [A]. In Proceedings of the 12th USENIX Security Symposium [C]. 2003. 105-120.
- [1] LEE W. A Data Mining Framework for Building Intrusion Detection Model [J]. In IEEE Symposium on Security and Privacy, 1999, 7: 120-132.
- [2] SOURDIS I, PNEVMATIKATOS D. Fast, Large-Scale String Match for a 10Gbps FPGA-based Network Intrusion Detection System [A]. In 13th Conference on Field Programmable Logic and Applications [C]. Lisbon, Portugal, September Springer-Verlag, 2003.
- [3] 刘航, 戴冠中, 李晖晖, 等. 基于 FPGA 的高速网络入侵监测系统 [J]. 计算机应用, 2004, 24(5): 33-35.
- [4] 李建武. 基于 IPv6 的网络入侵检测系统研究和设计 [D]. 西安: 西北工业大学, 2005.
- [5] 王玲, 钱华林. IPv6 的安全机制及其对现有网络安全体系的影响 [DB/OL]. <http://www.77125.com/download/2003/11/27/160306.pdf>, 2003-11/2005-10.
- [6] ROESCH M. Snort: The open source network intrusion detection system [EB/OL]. <http://www.snort.org>, 2003-10.

(上接第 2343 页)

层实现的,而对于 IP 层以上的一些缺陷和应用程序的漏洞在 IPv6 中还不能提供全面的保护。通过分析 IPv6 协议的特点及其安全性能,设计了一种协议分析技术与模式匹配技术相结合的 IPv6 网络入侵检测系统,提出了一个基于可重配置硬件的网络入侵检测系统体系结构,将其中网络数据包的特征匹配用 FPGA 加以实现,使网络数据包的处理并行化。

由本文的分析可以看出, FPGA 等可重配置硬件将成为未来高速网络入侵检测系统以及网络安全系统开发和运行的理想硬件平台,值得我们在相关领域进行更深入的研究和探索。

参考文献: