防御缓冲区溢出攻击的数据随机化方法

严 芬[1,2] 袁赋超[1] 沈晓斌[1] 殷新春[1,2] 茅 兵[2]
(扬州大学信息工程学院 扬州 225009)[1] (南京大学软件新技术国家重点实验室 南京 210093)[2]

摘 要 代码注入式攻击方法已经成为针对内存攻击的典型代表,缓冲区溢出攻击是其中最常用的一种代码注入攻击方法。它依靠修改程序的控制流指向,使程序转向预先注入的恶意代码区,以取得系统权限。提出了一种基于数据保护的随机化方法,即通过保护程序内的指针和数组来有效地防御缓冲区溢出攻击的方法。

关键词 缓冲区溢出,数组保护,指针保护,数据随机化
中图法分类号 TP309 **文献标识码** A

Data Randomization to Defend Buffer Overflow Attacks

YAN Fen[1,2] YUAN Fu-chao[1] SHEN Xiao-bing[1] YIN Xin-chun[1,2] MAO Bing[2]
(College of Information and Engineering, Yangzhou University, Yangzhou 225009, China)[1]
(State Key Laboratory for Novel Software Technology, Najing University, Nanjing 210093, China)[2]

Abstract Code injection attack has become a typical representative of the attacks against memory, in which buffer overflow attacks are the most commonly used. It relies on the change of control-flow, lets the program point to the malicious code in order to obtain the root rights. This paper presented a method using randomization based on data protection, wiich can defend buffer overflow attacks effectively, through the protection of pointers and arrays.

Keywords Buffer overflow, Array protection, PointGuard, Data randomization

1 引言

缓冲区溢出攻击一直是一种安全危害很大的计算机攻击,由缓冲区漏洞导致的攻击占了各式软件与操作系统漏洞的大部分[1]。目前,已经提出了多种针对缓冲区溢出攻击的解决方法,以从不同角度和层面解决缓冲区溢出漏洞。但是,除了边界检查[2]外,并没有一种方法可以有效地解决所有缓冲区溢出漏洞攻击。大多数方法都是通过防范某一方面来抵御某一些的漏洞[3-6],攻击者仍能通过绕过设置的保护来实施攻击,如 StackGuard[13,14] 只能保护直接栈溢出攻击,对非栈溢出的攻击无能为力。本文通过对程序空间的数组和指针变量使用随机化方法来保护缓冲区、指针数据和返回地址,实现在源码级上对缓冲区溢出漏洞实施防护。

2 缓冲区溢出攻击目标

缓冲区溢出漏洞攻击的产生都是基于对程序空间内存分布的先验知识,其根本原因还是在于 C 和 C++语言的不安全性,出于性能考虑而不进行内存操作的边界检查。一般而言,缓冲区溢出攻击的目的是为了加载攻击者自己精心设计的恶意代码,以获取系统权限,而要使程序控制流顺利跳转到恶意代码,让攻击顺利发生,就需要修改当前执行的程序指令的地址,改变执行的流程。这些能够通过溢出方式来改变程序控制流指向的目标就是缓冲区溢出攻击的目标,可以分为以下 3 种。

(1) 函数返回地址

当程序内函数被调用时,调用者先将当前程序的执行地址入栈(并做为返回地址),而通过相邻函数变量来覆盖返回地址,当函数调用结束返回后,程序就将跳转到被覆盖的返回地址所指向的代码段(即恶意代码区)。函数返回地址是程序攻击的首选目标。

(2) 函数指针

与函数返回地址类似,函数指针也会直接影响到程序的控制流。通过函数指针可以控制程序控制流的转换,例如程序内一个函数的类型为 void,则它声明了一个返回值是 void 类型的函数指针变量,当通过该指针附近的缓冲区来达到溢出时,溢出的是恶意代码区的地址,则下一次该函数指针被调用时就能直接跳转到恶意代码区。函数指针是程序攻击的重要目标。

(3) PLT/GOT 表

PLT 和 GOT 联合实现程序的动态重定位,从本质上来说,GOT 也可以认为是函数指针。发生函数调用时,如果被调用的函数需要重定位,则首先跳转到相应的 PLT 表项,

到稿日期:2010-02-05 返修日期:2010-04-27 本文受国家 863 高技术计划项目(2007AA01Z448 和 2007AA01Z409),江苏省科技支撑计划基金项目(BE2008124)资助。

严 芬(1978—),女,博士,讲师,主要研究方向为网络与信息安全,E-mail:yanfen@yzu.edu.cn;袁赋超(1985—),男,硕士生,主要研究方向为信息安全;沈晓斌(1985—),男,硕士生,主要研究方向为信息安全;殷新春(1962—),男,教授,博士生导师,主要研究方向为密码学与信息安全;茅 兵(1967—),男,教授,博士生导师,主要研究方向为包括安全操作系统、软件安全、分布式系统安全等。

PLT 表项的第一条指令跳转到其相应的 GOT 表项所指向的地址执行。简单来说，如果 GOT 表项通过其相邻的缓冲区被溢出，则 GOT 表项所指向的地址就可能是恶意代码区的地址。PLT/GOT 表也是程序攻击的重要目标。

3 数据随机化方法

3.1 数据随机化的思想

通过对缓冲区溢出目标的了解，通常发生溢出的都是数组类型的缓冲区，溢出发生后覆盖的对象都是指针类型的返回地址、函数指针或者 GOT 表项。由此可知，如果通过对数组类型和指针类型的数据进行保护，原理上可以有效地防止缓冲区溢出攻击。本文给出的数据随机化方法的基本思想是：对数组和指针变量进行随机化(异或加密)防护，并不阻止溢出的发生，而是在溢出发生之前，对数组和指针变量进行异或加密(加密使用的值 mask 是一个 32 位的随机数)，使得溢出发生后，溢出覆盖的目标(如返回地址、函数指针或者 PLT/GOT 表项)是个加密后的值，不能转到预期的恶意代码段，取得防御缓冲区溢出攻击的效果。本文的数据随机化方法用于在 C 语言编写的开源软件上防御缓冲区溢出攻击。在程序初始化时，对于任意一个数组变量 A，在内存空间中另外开辟一个变量 MA，其中存储一个随机数 mask$A$，当对这个变量 A 进行赋值操作时，首先是赋值，然后对 A 中的值进行异或加密：$A=A\oplus \text{mask}A$。此时在内存中存放变量 A 的地方实际存放的值是 $A\oplus \text{mask}A$，在变量 A 以后使用时再进行解密，即再进行一次异或运算：$A=A\oplus \text{mask}A$，从而把 A 的值还原成原来的值；对于任意一个指针变量 B，也在内存空间内开辟一个变量 MB，其中同样存储一个随机数 mask$B$，由于指针变量 B 内放的是指针所指空间的地址 B'，所以，在程序对这个地址值 B' 进行赋值操作时，先对 B' 赋值，然后对 B' 的值进行异或加密：$B'=B'\oplus \text{mask}B$。而在指针变量 B 以后使用时再对 B' 进行解密，同样地再进行一次异或：$B'=B'\oplus \text{mask}B$。通过这种方法在程序运行的时候，就能对数组和指针进行有效的保护。通过例 1 分析数据随机化方法是如何防御缓冲区溢出攻击的。

例 1
```
…
copy(char * msg)
{
  char buffer[512];
  strcpy(buffer,msg);
  }
main(int argc,char * argv[])
{
  if(argc>1)
    copy(arg[1]);
  }
```

例 1 程序的内存分配情况如图 1 所示。当调用函数 copy()后，参数、返回地址、前一个栈的指针、本地变量被依次压入栈中。函数 strcpy()将输入复制到 buffer 中。由于 strcpy()不检查输入变量的边界，当它向 buffer 中复制的字符多于 512 个时，输入的多余字符就可覆盖栈中的返回地址，使其指向攻击代码。通过使用数据随机化方法，在将输入复制到 buffer 时，输入的数据先使用随机数 mask$A$ 进行加密，此时，溢出的字符经过了一层加密。由于返回地址也属于指针变量，也使用随机数 mask$B$ 进行了一次加密，因此返回地址对应内存的值是溢出字符经过两层加密后得到的值，即“字符值 $\oplus$ mask$A$ $\oplus$ mask$B$”，在 copy()返回到 main()时候，返回地址直接指向“字符值 $\oplus$ mask$A$ $\oplus$ mask$B$”。由于 mask$A$ 和 mask$B$ 都是 32 位的随机数，而 mask$A$ 与 mask$B$ 相等的概率极低(见 4.2 节分析)，因此返回地址并没有指向攻击代码，攻击并没有成功。

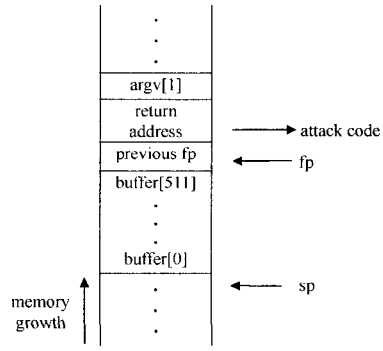

图 1 内存分配图

3.2 指针变量混淆

按照数据随机化的思想执行程序时，对数组变量进行加解密并没有问题，但指针变量会发生一种混淆的情况。当程序中发生例 2 的情况时，指针变量会发生混淆，就不能确定到底使用 a 还是 b 来对指针变量 c 进行解密。

例 2
```
Int a,b,c, * ptr
…
ptr=&a;
…
ptr=&b;
…
c= * ptr;
```

该例中 c 内存储的加密后的值是 mask$c$ $\oplus$(mask$a$ $\oplus$ a)还是 mask$c$ $\oplus$(mask$b$ $\oplus$ b)，使用数据随机化思想的加密方法，并不能分清楚，因此本文使用静态分析的方法来解决这一问题。

对于程序中控制流的流向所用的指针，可以通过使用相同的标记值 mask 来进行异或加密，因此，静态分析就可以分为以下两部分进行：指针分析和标记值确定。

3.3 指针静态分析

由于产生指针混淆情况，因此需要用到指针分析[7]的技术来确定指针的指向。通过分析指针的指向，将程序控制流流向的所有指针进行分类，打上同一标记 mask 值，而这种指针的指向流包含的数目多少取决于指针分析的准确性。由于程序对象可能是大型的，代码量比较大，因此在寻求指针分析方法时，首先考虑的是准确度，其次还要考虑性能损耗，所以使用 Steensgaard[8,9]的指针分析方法。Steensgaard 是一种基于合并的指针分析方法，即通过对指针的分析，解析指针之间的对应关系，然后划分等价类，对指针对象进行合并，得出程序的指针指向图(point-to graph)。通过例 3 分析 Steensgaard 方法是如何执行指针分析的。

例 3
```
void steensgarrd(int * * a1){…}
a2=&a4;
a3=&a5;
a3=&a6;
```

```
...
steensgarrd(&a2);
steensgarrd(&a3);
```

上例是一段程序，其中包含了一些指针指向关系，使用 Steensgaard 方法分析指针指向关系，并且将指针划分为不同的等价类，由这些等价类来表示分析后得到的指针指向图，如图 2 所示。

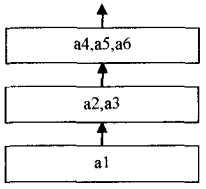

图 2　指针指向图

Steensgaard 方法就是对指针的指向关系进行分类，上例中将程序的所有指针分成了 3 个类，那么指针变量所对应的标记值 mask 也只会产生 3 个不同的 mask 值相应地，我们可以得到{$a1$}，{$*a1, a2, a3$}，{$**a1, *a2, *a3, a4, a5, a6$}这 3 个集合内的结点所对应的 mask 都是相同的。

3.4　标记值的确定

得到指针指向表后，明确了指针关系，就可以对各个指针变量进行标记值的确定。需要根据指针指向表，从指针指向的方向确定标记值。对于例 3 程序中的指针，就是从 $a1$ 开始打标记，$a1$ 打上标记 mask$a1$ 以后，再找 $a1$ 的指向 $*a1$，发现{$*a1, a2, a3$}同属于一级，则把 $a2$ 和 $a3$ 的 mask 值打上相同的 mask$a2$，然后再找 $a2, a3$ 的指向，发现{$**a1, *a2, *a3, a4, a5, a6$}同属于一级，再把 $a4, a5, a6$ 打上相同的 mask$a4$。

由于 mask 值是一个随机数，存放在对应的静态变量 MASK 中，因此只需要在指针分析完成后，根据分析得到需要进行加密对象的多少来分配相应个数的静态变量 MASK。通过指针分析清楚需要使用的 mask 后，就可进行数据随机化的初始化。mask 的值最大为 2^{32}，即随机数 mask 的值在 32 位的二进制数表示范围内。

4　实验过程及结果分析

4.1　实验方法及环境

在 32 位 X86 体系的 Linux 开源系统下，使用 CIL[10] 和 Objective Caml 语言实现我们的数据随机化方法。首先，通过 CIL 工具在源码级上进行静态分析，对 C 语言写的程序源代码分析产生指针指向图(point-to graph)，如图 3 所示，再给指针变量和数组变量打上标记，实现程序源代码的转换，然后，利用 Objective Caml 语言编写的程序实现数据随机化(异或加密)。下面通过实例介绍数据随机化过程的主要工作。

(1)假设有如下 C 程序源码

```
int Array[512];
int *A1, *A2, **A3, **A4, node;
int main()
{
  Array[512]={1}
  A1=&node;
  A3=&A1;
  A4=A3;
  A2=*A3;
  ...=&A3;
}
```

(2)静态分析指针间指向关系

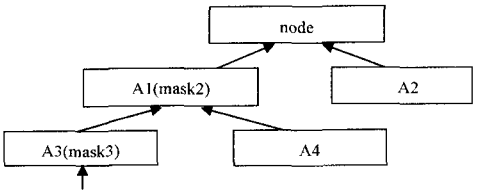

图 3　C 源程序的指针指向图

(3)将 C 程序源码用数据随机化方法转换以后的代码

```
static unsigned int mask1, mask2, mask3, mask4;
int Array[512];
int **A1_1, *A2, ***A3_1, **A4, *node_1;
int main()
{
  Array[512]={1^mask1}
  (*A1_1)=node_1;
  A3_1=A1_1;
  (*A4_1)=A3_1;
  (*A3_1)=(int **)(unsigned int)(*A3_1^mask2);
  A2=(int *)((unsigned int)(*((int **)((unsigned int)(*A3_
      1^mask3)))^mask2);
}
```

4.2　缓冲区溢出攻击测试

表 1 介绍了 Wilander 的攻击测试平台中的 18 种缓冲区溢出攻击。从攻击方式来看，可以将这些缓冲区溢出攻击分为两类，一类是直接的溢出攻击，即直接通过缓冲区溢出覆盖攻击目标对象的值，从而达到攻击的目的，前面 8 种都是直接溢出攻击；另外一类是间接攻击，即先通过缓冲区溢出覆盖某个指针，然后通过程序内部本来的对该指针的赋值来改变攻击目标的值，从而达到漏洞攻击的目的，后 10 种都是间接溢出攻击。从攻击目标的存储位置上又可以分为：针对栈的攻击、针对数据段和堆的攻击。使用本文的数据随机化方法对这 18 种攻击分别进行了测试，测试结果如表 1 所列。

分析：

(1) 针对栈的攻击：传统的缓冲区溢出攻击一般都是发生在栈上的溢出，比如说覆盖返回地址、函数指针等，这些攻击是可以被有效防御的。本文的随机化方法有效地保护了指针和数组变量，使得针对栈的攻击所预想发生的溢出的位置并不是原来所希望溢出的位置。实验结果表明，本文的方法可以完全抵御针对栈的攻击。从原理上分析，所要溢出覆盖的指针变量(返回地址、函数指针、PLT/GOT 表项)使用的 mask 值和溢出数组所使用的 mask 值一样时，还是有极低的被攻击概率的。由于考虑 mask 最大是 32 位的二进制值，因此具有相同 mask 值的概率就为 2^{-32}，即被攻击概率为 2^{-32}。

(2) 针对堆的攻击：堆溢出的位置一般都是目标堆块的尾或者是相临堆块的头，覆盖的对象都是堆控制数据相关的指针，如表 1 所列。当使用本文随机化方法后，这些指针与数组都得到了有效的保护，与针对栈的攻击一样，溢出仍然发生，但溢出的位置并不是所预想的位置，并不能继续流向恶意代码区，所以攻击没有顺利执行。同样，由于加密由 mask 值异或得到，只有在具有相同 mask 值的情况下才有被攻击的

可能，因此被攻击的概率很低，与针对栈攻击一样，也是 2^{-32}。但是，与针对栈的攻击不同的是，如果程序遇到针对堆的攻击时，溢出覆盖了堆控制数据相关的指针，那么会导致程序的崩溃，而栈攻击只会显示程序出错。

表 1　Wilander 基准测试集的测试结果

攻击技术	攻击类型	攻击目标	攻击描述	能否抵御
缓冲区溢出攻击	栈空间直接溢出攻击	(1)返回地址	缓冲区溢出直接覆盖函数返回地址	√
		(2)栈指针	缓冲区溢出直接覆盖函数旧的栈指针	√
		(3)局部函数指针	缓冲区溢出直接覆盖函数指针变量	√
		(4)函数指针参数	缓冲区溢出直接覆盖传入的函数指针参数	√
		(5)局部 Longjmp 缓冲区	缓冲区溢出直接覆盖局部 Longjmp 缓冲区	√
		(6)Longjmp 缓冲区参数	缓冲区溢出直接覆盖传入的 Longjmp 缓冲区参数	√
	堆空间直接溢出攻击	(7)函数指针	缓冲区溢出直接覆盖数据段/堆上的函数指针变量	√
		(8)Longjmp 缓冲区	缓冲区溢出直接覆盖数据段/堆上的 longjmp 缓冲区	√
	栈空间间接溢出攻击	(9)返回地址	缓冲区溢出覆盖栈上指针，通过指针间接修改返回地址	√
		(10)栈指针	缓冲区溢出覆盖栈上指针，通过指针间接修改栈指针	√
		(11)局部函数指针	缓冲区溢出覆盖栈上指针，通过指针间接修改局部函数指针	√
		(12)函数指针参数	缓冲区溢出覆盖栈上指针，通过指针间接修改传入函数指针参数	√
		(13)局部 Longjmp 缓冲区	缓冲区溢出覆盖栈上指针，通过指针间接修改 Longjmp 缓冲区	√
		(14)Longjmp 缓冲区参数	缓冲区溢出覆盖栈上指针，通过指针间接修改传入的 Longjmp 缓冲区参数	√
	堆空间间接溢出攻击	(15)返回地址	缓冲区溢出覆盖数据段/堆上指针，通过指针间接修改返回地址	√
		(16)栈指针	缓冲区溢出覆盖数据段/堆上指针，通过指针间接修改栈指针	√
		(17)函数指针	缓冲区溢出覆盖数据段/堆上指针，通过指针见解修改函数指针	√
		(18)Longjmp 缓冲区	缓冲区溢出覆盖数据段/堆上指针，通过指针间接修改 Longjmp 缓冲区	√

4.3　性能分析

我们在打标记过程中首先要运行数据随机化方法的初始化(即随机数 mask 值的产生)，才可以先生成各个不同的随机数 mask，并且加密过程是在 C 程序初始化的时候，解密是在数组和指针变量引用的时候。这样可以有效地保护返回地址、函数指针和 PLT/GOT 表，表 2 和表 3 是本文数据随机化方法在 SPEC CPU 2000 benchmark suite[11] 上 7 个程序运行时的性能损耗和内存空间损耗(程序用 gcc-3.2.2 RedHat Linux 9.0 系统在 Intel 酷睿双核 1.6GHz 处理器和 1G 内存的机器上运行)。

表 2　CPU 性能损耗表

程序	Cpu 性能损耗
Gzip	18%
Vpr	15%
Mcf	7%
Crafty	25%
Parser	11%
Bzip2	10%
Twolf	6%
平均	13%

表 3　内存空间损耗表

程序	内存空间损耗
Gzip	0.1%
Vpr	0.3%
Mcf	0.2%
Crafty	3.1%
Parser	0.8%
Bzip2	0.1%
Twolf	2.0%
平均	0.9%

对于数据随机化而言，运行时的 CPU 性能损耗主要还是在为数组和指针变量打标记和解密时标记值的使用，而内存空间的损耗则主要是为标记值开辟空间的损耗。根据实验结果可以得到在运行这 7 个程序时，最高性能损耗是 25%，平均性能损耗是 13%，最高内存空间损耗为 3.1%，平均内存空间损耗为 0.9%。

(1)其中，输入输出密集型(I/O bound)的程序就有比较低的性能损耗，如 mcf 和 twolf，而计算密集型(CPU bound)的程序则损耗较高，如 crafty。

(2)另外，程序中数组和指针变量的数量多少对于内存空间损耗影响最大。可以看出：程序比较大、代码行数多时，使用数据随机化以后对内存空间损耗较多，如 crafty 和 twolf。

相比地址空间随机化技术(ASR)[16 18]，本文方法在性能损耗上稍高(以平均性能损耗 13%来看)，但是，防范强度很高，被攻击概率极低；相比指令随机化技术(ISR)[12,15]，本文方法在性能损耗和空间损耗上都有所降低，同样都是针对代码的注入攻击，本文方法有很好的防御效果。

结束语　本文提出了一种基于数据保护的随机化方法来防御缓冲区溢出攻击漏洞。从缓冲区溢出攻击的目标着手进行保护，并不阻止溢出的发生，只利用加密的方法保护了数组和指针，使攻击位置发生改变，攻击不能顺利发生。相比于其它防御恶意代码注入攻击的随机化方法，本文提出的方法具有较低的性能损耗与内存空间损耗，且防御强度高，能够完全防御缓冲区溢出攻击。这种数据随机化是通过加密来实现的，因此从性能上考虑只是采用了最简单的异或加密，因为是在 32 位机上做的保护，加密时所使用的标记值 mask 也是个 32 位的随机数，所以，从原理上来说，还是存在着 2^{-32} 的被攻击概率，当然这个概率已经很低了。在以后的研究中可以根据程序运行的需要，在性能损耗可以承受的情况下，适当加强加密的强度，使攻击更难以发生。

参 考 文 献

[1] National Institute of Standards and Technology. ICAT Metabase [OL]. http://icat.nist.gov/

[2] SANS Institute. The twenty most critical internet security vulnerabiliti-es[OL]. http://www.sans.org/tip20/

[3] Cowan C, Wagle P, Pu C, et al. BufferOverflows: Attacks and Defenses for the Vulnerability of the Decade[OL]. http://www.csc.ogi.edu/DISC/Pro-jects/immunix

[4] Cowan C, Beattie S, Johansen J, et al. PointGuard: Protecting pointersfrom buffer overflow vulnerabilities[C]// USENIX Se-

curity Symposium. Washington, DC, August 2003

[5] Baratloo A, Singh N, Tsai T. Transparent RunTime Defense AgainstStack-Smashing Attacks [C] // 9th USENIX Security Symposium. August 2000

[6] Zhang Q. The Synmthetix MemGuard Kernel Programmerps Interface [OL]. http://www. cse. ogi. edu/DISC/projects/synthetix/toolkit/MemGuard/memg-urad. html

[7] Ramalingam G. The undecidability of aliasing [J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1994, 16(5): 1467-1471

[8] Steensgaard B. Points-to analysis by type inference of programs with structures and unions[C] // Conference on Compiler Construction, LN-CS 1060. April 1996: 136-150

[9] Steensgaard B. Points-to analysis in almost linear time [C] // ACM Sym-posium on Principles of Programming Languages (POPL). January 1996: 32-41

[10] McPeak S, Necula G C, Rahul S P, et al. CIL: Intermediate language andtools for C program analysis and transformation[C] // Conference on Compiler Construction. 2002

[11] SPEC. Spec Benchmarks[OL]. http://www. spec. org

[12] Barrantes E G, Ackley D H, Forrest S, et al. Randomized instruction setemulation to disrupt binary code injection attacks [C] // ACM Conferen-ce on Computer and Communications Security (CCS). Washington, DC, October 2003

[13] Cowan C, Beattie S, Day R, et al. Protecting Systems from Stack Sm-ashing Attacks with StackGuad [Z]. Linux-Expo, Raleigh, NC, May 1999

[14] Cowan C, Pu C, Maier D, et al. StackGuard: Automatic Adaptive Det-ection and Prevention of Buffer-Overflow Attacks[C] // 7th USENIX S-ecurity Conference. San Antonio, TX, January 1998: 63-77

[15] Kc G S, Keromytis A D, Prevelakis V. Countering code-injection at-tacks with instruction-set randomization [C] // ACM Conference on Com-puter and Communications Security (CCS). Washington, DC, October 2003: 272-280

[16] Bhatkar S, DuVarney D C, Sekar R. Address obfuscation: an effcie-nt approach to combat a board range of memory error exploits[C] // S-SYM'03: Proceedings of the 12thconference on USENIX Security Symposium. Berkeley, CA, USA, USENIX Association, 2003: 8-8

[17] Xu J, Kalbarczyk Z, Iyer R K. Transparent runtime randomization for security[C] // Proc. of 22nd Symposium on Reliable Distributed Systems (22nd SRDS'03). Florence, Italy, IE-EE Computer Society, October 2003: 260

[18] PaxTeam. Pax address space layout randomization (aslr) [OL]. http://pax. grse-curity. net/docs/aslr. txt, 2001

(上接第 50 页)

参 考 文 献

[1] Zeng Qing-an, Agrawal D P. An Analytical Modeling of Handoff for Integrated Voice/Data Wireless Networks with Priority Reservation and Preemptive Priority Procedures [C] // Proceeding of the Workshop on Wireless Networks and Mobile Computing in Conjunction with the International Conference on Parallel Processing (ICPP). 2000: 523-529

[2] Zheng Li, Zhang Liren. Modeling and Performance Analysis for IP Traffic with Multi-class QoS in VPN [C] // Proceedings 21st Century Military Communications Conference Volume 1. 2000: 330-334

[3] Wells L, Christensen S, Kristensen L M, et al. Simulation Based Performance Analysis of Web Servers [C] // Proceedings of 9th International Workshop on Petri Nets and Performance Models. 2000: 59-68

[4] Gruen R, Kubota T. A Neural Network Approach to System Performance Analysis [C] // Proceedings of IEEE. 2002: 349-354

[5] Turkboylari M, Madisetti V K. Effec of Handoff Delay on the System Performance of TDMA Cellular System [C] // Proceedings of 4th International Workshop on Mobile and Wireless Communications Network. 2002: 411-415

[6] Zeng Qing-an, Agrawal D P. Modeling and Efficient Handling of Handoffs in Integrated Wireless Mobile Networks [J]. IEEE Transactions on Vehicular Technology, 2002, 51(6): 1469-1478

[7] Kounev S, Buchmann A. Performance Modelling of Distributed E-BusinessApplications Using Queuing Petri Nets [C] // Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2003). Austin, Texas, March 2003: 143-155

[8] Camera D, Guitart J, Torres J, et al. Complete Instrumentation Requirements for Performance Analysis of Web Based Technologies [C] // 2003 IEEE International Symposium on Performance Analysis of Systems and Software. 2003: 166-175

[9] Xiao Zhengjin, He Qinming, Chen Qi. A Method of Workflow Model Performance Analysis Based on the Mechanism of Ant Colony Found Food [C] // Proceedings of the 6th World Congress on Intelligent Control and Automation. Dalian, China June 2006: 6983-6987

[10] Ma Xiaomin, Cao Yonghuan, Liu Yun, et al. Modeling and Performance Analysis for Soft Handoff Schemes in CDMA Cellular Systems [J]. IEEE Transactions on Vehicular Technqlogy, 2006, 55(2): 670-680

[11] El-hadidi M T, Hegazi N H, Aalan H K. Performance evaluation of a new hybrid encryption protocol for authentication and key distribution [C] // Proceedings of the 4th IEEE International Symposium on Computers and Communications. Los Alamitos, Calif: IEEE Computer Society Press, 1999: 16-22

[12] Zhang Yan, Fujise M. An Improvement for Authentication Protocol in Third-generation Wireless Network [J]. IEEE Transaction on Wireless Communications, 2006, 5(9): 2348-2352

[13] Harbitter A, Menasce D A. A Methodology for Analyzing the Performance of Authentication Protocols [J]. ACM Transactions on Information and System Security, 2002, 5(4): 458-491

[14] Liang Wei, Wang Wenye. An Analytical Study on the Impact of Authentication in Wireless Local Area Network [C] // Proceedings of 13th International Conference on Computer Communications and Networks. 2004: 361-366