第 27卷 第 5期 南京邮电大学学报(自然科学版) Vol 27 No. 5
2007年 10月 Journal of Nanjing University of Posts and Telecommunications(Natural Science) Oct 2007

文章编号: 1673-5439(2007)05-0084-06

格式化字符串攻击检测与防范研究

李 鹏,王汝传,王绍棣
南京邮电大学 计算机学院,江苏 南京 210003

摘 要:从攻击原理、攻击检测与攻击防范角度研究格式化字符串攻击。文中首先比较了格式化字符串攻击和缓冲区溢出攻击的联系与区别,总结了 * printf( )系列函数与格式化字符串攻击相关的 3条特殊性质以及 6种格式化字符串攻击的方法。采用基于源代码的检测方法对格式化字符串漏洞进行检测,并阐述了漏洞检测方法的原理和关键代码。最后对格式化字符串攻击的 3种防范技术 FormatGuard、Libsafe和 White-Listing进行了比较研究,特别对 Linux下的基于动态链接库的保护方法 Libsafe的原理及其对格式化字符串攻击的防范策略采取的 3个步骤进行了详细分析。

关键词:格式化字符串攻击;溢出攻击;溢出检测;溢出防范
中图分类号: TP393. 08 **文献标识码**: B

Detection and Protection Research on Format String Vulnerability Exploits

LI Peng, WANG Ru-chuan, WANG Shao-di
College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China

Abstract: This paper focuses on format string vulnerability exploits in terms of principle, detection and protection. Firstly, it draws a comparison between format string vulnerability exploits and common buffer overflow attacks. At the same time, it summarizes three special properties of * printf( ) serial functions related to format string vulnerability exploits and six ways of exploiting format string vulnerabilities. Then it detectes format string vulnerabilities based on the examination of the source code, and expounds the principle and crucial codes of the vulnerable examination method. Finally, it makes research on three technologies for preventing format string attacks which namely FormatGuard, Libsafe and White-Listing by comparison, and analyzes the principle of Libsafe, which protects against format string vulnerability exploits based on the dynamic linked library under Linux operation system, and three steps used to protect against format string vulnerability exploits.

Key words: Format string vulnerability exploits; Overflow attack; Overflow detection; Overflow protection

1 引 言

格式化字符串攻击是缓冲区溢出攻击的其中一种,普通的缓冲区溢出就是利用了堆栈生长方向和数据存储方向相反的特点,用后存入的数据覆盖先前压栈的数据,一般是覆盖返回地址,从而改变程序的流程,这样函数返回时就跳转到了攻击者指定的

收稿日期: 2007-01-15
基金项目:国家自然科学基金(60573141、70271050)、江苏省自然科学基金(BK2005146)、江苏省高技术研究计划(BG2004004、BG2005037、BG2006001)、国家高技术研究发展计划(863计划)(2006AA01Z439)、南京市高科技项目(2006软资 105)、现代通信国家重点实验室基金(9140C1101010603)、江苏省计算机信息处理技术重点实验室基金(kjs050001、kjs06006)资助项目
通讯作者:王汝传 电话: (025) 83806001
E-mail: wangrc@njupt. edu. cn

地址,就可以按照攻击者的意愿做任何事情了[1]。

格式化字符串漏洞和普通的缓冲溢出有相似之处,但又有所不同,它们都是利用了程序员的疏忽大意来改变程序运行的正常流程[2]。

格式化字符串漏洞的历史要比缓冲区溢出短得多,而且一般被认为是程序员的编程错误。但是格式化字符串漏洞攻击可以往任意地址写任意内容,它的危害也是非常致命的[3]。

2　格式化字符串攻击

2.1　*printf()系列函数的3条特殊性质

所谓格式化字符串,就是在*printf()系列函数中按照一定的格式对数据进行输出,可以输出到标准输出,即printf(),也可以输出到文件句柄、字符串等。对应的函数有fprintf、sprintf、snprintf、vprintf、vfprintf、vsprintf等,能被黑客利用的地方也就出在这一系列的*printf()函数中。在正常情况下这些函数不会造成什么问题,但是*printf()系列函数有3条特殊的性质,这些特殊性质如果被攻击者结合起来利用,就会形成漏洞。可以被攻击者利用的*printf()系列函数的3个特性是:

(1) *printf()系列函数参数个数不固定造成访问越界数据。

因为*printf()系列函数的参数的个数是不固定的,如果其第一个参数即格式化字符串是由用户来提供的话,那么用户就可以访问到格式化字符串前面的堆栈里的任何内容了。之所以会出现格式化串漏洞,就是因为程序员把printf()的第一个参数,即格式化字符串,交给用户来提供,如果用户提供特定数量的%x,就可以访问到特定地址的堆栈内容。

(2) 利用%n格式符写入跳转地址。

上一步中只是显示内存的内容而没有改变它,利用*printf()的一个特殊格式符%n,就可以向内存中写入内容。%n是一个在编程中不经常用到的格式符,它的作用是把前面已经打印的长度写入某个内存地址。在实际利用某个漏洞时,并不是直接把跳转地址写入函数的返回地址,而是通过地址来间接的改写返回地址。现在已经可以利用提交格式化字符串访问格式化字符串前面堆栈里的内容,并且利用%n可以向一个内存单元中的地址去写入一个值。既然可以访问到提交的字符串,就可以在提交的字符串当中放上某个函数的返回地址的地址,这样就可以利用%n来改写这个返回地址。当然,%n向内存中写入的值并不是随意的,它只能写入前面打印过的字符数量,而攻击者需要的是写入存放shellcode的地址,就象普通的缓冲区溢出攻击那样。

(3)利用附加格式符控制跳转地址的值。

*printf()系列函数有个性质是程序员可以定义打印字符的宽度。就是在格式符的中间加上一个整数,*printf()就会把这个数值作为输出宽度,如果输出的实际大于指定宽度则仍按实际宽度输出;如果小于指定宽度,则按指定宽度输出。可以利用这个特性,用很少的格式符来输出一个很大的数值到%n,而且这个数值可以由攻击者来指定。通过附加格式符来控制向函数返回地址中写入的值,一般是利用%n前面的最后一个格式符来控制这个数值,这通常需要一些计算,计算方法一般是用shellcode的地址减去最后一个格式化字符串前的所有格式化字符串的打印长度。这样%n写入的数值就恰好是shellcode的地址了。

2.2　格式化字符串攻击的常用方法

格式化字符串攻击的常用方法[3],现归纳总结如下:

(1) 覆盖函数返回地址。

覆盖当前执行函数的返回地址,当这个函数执行完毕返回的时候,就可以按照攻击者的意愿改变程序的流程了。使用这种技术的时候需要知道以下两个信息:堆栈中存储函数的返回地址的那个存储单元的地址以及shellcode的首地址。

格式化字符串攻击覆盖函数返回地址通常有两种选择:①覆盖邻近的一个函数(调用该函数的函数)的返回地址,这是与普通缓冲区溢出攻击相类似的方法。②覆盖*printf()系列函数自身的返回地址。较前面的利用方法而言,该方法具有更高的精确度,即使是在条件较为苛刻的情况下也可以使用。

(2) 覆盖.dtors列表(析构函数指针)。

ELF文件格式的.dtors列表即析构函数指针,.dtors的作用一句话就可以表述出来,也就是该表中的内容将在main函数返回的时候被执行。利用格式化字符串漏洞进行攻击覆写的是内存映像的.dtors,默认编译的ELF文件都是有这个字段的。

这种攻击方式的思路是利用格式化字符串漏洞"往任意地址写任意内容"的特点,直接把shellcode的地址写入.dtors中,shellcode的地址可以是一个有效的范围,使用的shellcode位于其中,然后用

NOP填充满,这样可以提高精确度;或者可以考虑把 shellcode放入环境变量,这样精度更高,而且限制更少。

这种格式化字符串攻击技术的特点:①如果攻击者可以看到目标的二进制代码,那么很容易确定覆写的地址,只要分析 ELF文件镜像以及.dtors的位置就可以了。②其缺点是目标程序应当被编译,并且与 GNU工具链接;在程序执行 exit函数之前,很难找到可以存储 shellcode的地方[4]。

(3) 覆盖 GOT表(全局偏移表)。

在 ELF文件的进程空间中,全局偏移表(Global Offset Table, GOT)能够把位置无关的地址定位到绝对地址。程序使用到的每个库函数在 GOT表中都有一个函数地址的入口。在程序第一次使用函数之前,入口包含运行连接器(run-time-linker, rtl)的地址。当程序调用函数时,控制权将传递给 rtl,函数实地址被解析并插入到 GOT表中。此后,每次调用此函数就直接传递控制权给该函数,不再调用 rtl了[5]。

在利用格式化字符串漏洞之后,通过覆写程序即将使用到的函数 GOT表入口,攻击者可以夺取程序的控制权,并且跳转到任意可执行的地址。执行返回地址检查的堆栈保护策略对这种格式化字符串攻击方式不起作用。

通过覆写 GOT表的入口方法获得程序的控制权具有以下的优点:①独立于环境变量(如堆栈)和动态内存分配(如堆)。GOT表入口地址由二进制代码确定,如果两个系统运行相同的二进制代码,则其 GOT表的入口总是一样的。②易于操作,运行 objdump命令就可以获取到需要覆写的地址。

(4) Return into LibC(返回库函数)。

Return into LibC的原理阐述(见图 1):这里的 function_in_lib本来应该是函数的返回地址,现在变成系统库中的地址或者是 PLT(过程链接表)中的地址。程序用到基址寄存器 EBP, buffer fill-up (*)应该正确的处理这个 function_in_lib函数。arg_1, arg_2, …是 function_in_lib函数的参数。当 function_in_lib函数返回时,会把 dummy_int32作为返回地址 EIP,继续执行[6]。

←堆栈增长方向　　内存地址增长方向→

buffer fill-up (*)　function_in_lib　dummy_int32　arg_1 arg_2 …

图 1 Return into LibC的情况

攻击者的利用程序在调用 system()函数之前必须调用一系列的函数来获得特权。必须解决两个问题:①把一系列的调用都串联起来,同时要保证使用的是正确的参数。目前有两种方法把一系列函数调用串联起来:"esp lifting"方法,该方法适用于-fomit-frame-pointer编译的程序;frame faking方法,该方法是为编译时没有带上-fomit-frame-pointer编译选项设计准备的。②函数和参数所有的数据都不能包含'\0'。

利用 Return into LibC技术进行格式化字符串攻击的特点:①需要一个固定地址的缓冲区,也就是说缓冲区或者是静态的,或者是 malloc产生的,并且该缓冲区的数据是可由用户控制的。②必须精确得到一些重要的数据,如需要在利用程序开头定义的一些常量的数值。

(5) 利用 atexit结构。

这种格式化字符串攻击方式主要针对 Linux环境下静态链接二进制代码,利用的是 atexit结构的处理器,atexit()函数用于注册一个给定的函数,该函数在程序 exit时候被调用。注册的函数是反序被调用的,至少可以注册 32个函数,当然,只要有足够分配的内存,更多的函数也是允许的。这种机制允许程序设置多个处理器,在退出时用于释放资源。

在静态编译的二进制中,libc被保存在程序的堆区域,因此,__atexit的位置在程序中的静态数组附近。在静态的字符缓冲区后面构造攻击者的 atexit结构,覆盖 __atexit变量,可以使 exit()函数执行在内存中的任何地方,比如执行攻击者设置好的 shellcode[7]。

(6) 覆盖 jmpbuf's。

覆写 jmpbuf的技术最初是用于堆溢出漏洞利用的。由于格式化字符串 jmpbuf的特性与函数指针类似,而且格式化字符串可以往内存任意地址写入任何内容,不受缓冲区中 jmpbuf相对位置的限制[8]。

格式化字符串攻击可以使用 jmpbufs(setjmp/longjmp)进行漏洞利用。jmpbuf中保存的是栈帧,在后面执行时跳转到这里。如果攻击者有机会覆盖在 setjmp()和 longjmp()函数之间的缓冲区,就可以进行漏洞利用了。

3 格式化字符串攻击检测

本文的格式化字符串攻击原理部分提到了 *printf()系列函数存在参数个数不固定造成访问越界数据、可以在提交的字符串当中放上某个函数

的返回地址的地址以及通过附加格式符来控制向函数返回地址中写入的值等 3条特殊的性质,这些特殊性质如果被攻击者结合起来利用,就可以组织格式化字符串攻击。针对格式化字符串攻击的这 3条特殊性质,提出对格式化字符串攻击的检测思路,检测程序的流程图如图 2所示。

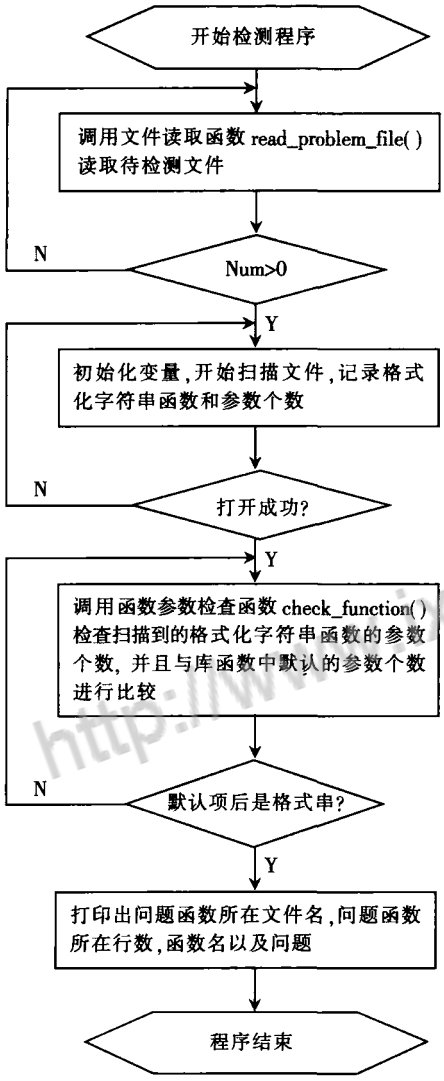


图 2　格式化字符串攻击检测程序流程图

采用的是基于源代码的检测方法,对使用格式化字符串函数的源文件进行扫描,统计出各个格式化字符串函数的参数个数,并与提供这些函数的库函数中默认的参数个数进行比较,如果缺少参数或者多出参数,则提示可能存在格式化字符串漏洞;如果符合参数个数的要求则不报错;同时对于一些固定的参数将认为是安全的,也不予报错。检测程序的关键代码分析如下:

(1) 程序首先定义库函数提供的默认的格式化字符串函数参数个数表,用于检测函数使用。定义的参数个数表的结构体如下:

```
typedef struct problem_t {
    const char * function;    /*格式化字符串函数名称
    int fmt_arg;              /*格式化字符串函数参数的默认个数
} problem_t;
```

(2) 读取文件函数 read_problem_file( ),这个函数的作用是读取文件,并将文件一行一行的进行扫描读取,然后将使用的格式化字符串函数以及所在行数放入自定义的数据结构中,准备检查使用。文件的数据结构如下所示:

```
void read_problem_file(const char * file) {
    FILE * fp;                /*创建文件指针
    char buffer[1024];        /*存放文件的缓冲区
    char name[1024];          /*存放文件名的数组
    int num, offset;          /*记录格式化字符串函数数目和函数偏移
    int line;                 /*记录函数所在行数
}
```

(3) 参数检查函数 check_function( ),这个函数的作用是检查扫描文件得到的格式化字符串函数,与上面定义的库函数比较参数的个数,这个函数是格式化字符串攻击的检测的核心部分,其核心代码如下:

```
if ((state→problem→fmt_arg == state→args) &&
(state→constant_string != state→problem→fmt_arg)) /*参数个数比较
{
  printf("%s: %d SECURITY: %s call should have \"%%s\" as argument %d\n",
        filename,                 /*文件名
        state→line,               /*函数所在的行号
        state→problem→function,   /*问题函数名称
        state→problem→fmt_arg);   /*默认的函数参数个数
        total_errors++;           /*统计存在问题的函数个数
}
```

上述程序中,state→problem→fmt_arg是自定义的库函数参数表中的参数个数,state→args是扫描得到的格式化函数的参数的个数,两者进行比较,如果函数参数相等,但最后一个参数是一个格式化字符串的话,表明存在问题,于是打印出问题函数的所在文件名,问题函数所在的行数,问题函数的名字,以及存在的问题等信息。

4　格式化字符串攻击防范

4.1　格式化字符串攻击防范技术

针对格式化字符串攻击,目前主要有 Format-

Guard、Libsafe和 White-Listing等防范技术[9-10]。这些技术对于已知的格式化字符串攻击具有很好的防御效果。这 3种格式化字符串攻击防范技术的基本原理总结如下:

(1) FormatGuard。

FormatGuard提供针对格式化字符串漏洞的实时保护。为了防止格式化字符串攻击, FormatGuard比较传递给格式化函数的实际参数个数和所需要的参数个数,如果函数需要的参数比实际参数个数要多,则认为是攻击, FormatGuard记录下攻击企图并终止程序。

(2) Libsafe。

动态防范工具 Libsafe,是 Linux下的基于动态链接库的保护方法。目前的 Linux系统中,程序链接时所使用的大多数是动态链接库。动态链接库本身就具有很多优点,比如在库升级之后,系统中原有的程序既不需要重新编译也不需要重新链接就可以使用升级后的动态链接库继续运行。除此之外, Linux还为动态链接库的使用提供了很多灵活的手段,而预载机制就是其中之一。在 Linux下,预载机制是通过环境变量 LD_PRELOAD的设置提供的。简单来说,如果系统中有多个不同的动态链接库都实现了同一个函数,那么在链接时优先使用环境变量 LD_PRELOAD中设置的动态链接库。这样一来,就可以利用 Linux提供的预载机制将存在安全隐患的函数替换掉,而 Libsafe正是基于这一思想实现的。

(3) White-Listing。

White-Listing是控制格式化函数修改内存的技术,显式保存允许修改的内存范围列表,这个列表被称为 White表。在程序运行期间,从 White表中动态插入或者删除地址范围。因此,格式化函数可以检查 White表以验证写入的指针是否有效。通过简单调整 White表中允许地址范围, White-Listing可以提供不同的安全策略。

这 3种格式化字符串攻击防范技术的比较如表 1所示。

表 1 格式化字符串攻击防范技术的比较表

比较方面	FormatGuard	Libsafe	White-Listing
需要程序源代码	是	否	是
兼容性问题	有	没有	没有
增加开销	很小	较小	较小
总体防范效果	较好	好	一般

4.2 Libsafe技术的防范步骤[11]

从表 1的比较可以看出, Libsafe对格式化字符串攻击的总体防范效果最好。 Libsafe对格式化字符串攻击的防范策略采取以下 3个步骤:

(1) 拦截: Libsafe对漏洞函数执行相对应的替换函数。

Libsafe首先检查确认函数在现有参数情况下是否能够安全执行,如果安全检查通过, Libsafe要么执行程序原来的函数,要么执行其相对应的替换代码。否则, Libsafe给出安全警告。

由于 Libsafe是基于共享库的实现,在标准函数库之前装载到内存中,所以 Libsafe能够拦截函数。对 Linux操作系统而言, ld.so(运行装载器)负责将程序代码和函数库载入内存。如果程序需要调用系统标准函数库, ld.so将其装载到内存,并且为其链接好所有相关的库函数。如果系统中运行了 Libsafe, ld.so会在标准函数库之前将 Libsafe函数库装载到内存。因为 Libsafe中的相应替换函数与标准函数库中的原函数名称一样,所以 ld.so可以使用 Libsafe中的函数替换标准函数库中相对应的函数。

(2) 安全检查: Libsafe判定程序的函数是否可以安全执行。

Libsafe对不同函数采取不同的安全检查方法。如 _D_vfprintf()函数, Libsafe执行以下两个检查操作: ①返回地址和栈帧指针检查,对每个 %n, Libsafe都会检查相关联的指针参数。 ②栈帧范围检查,函数的参数列表应当位于单个栈帧中。

为了执行这两个检查, Libsafe需确定堆栈中帧的位置和大小。图 3是进程在栈中的结构示意图。

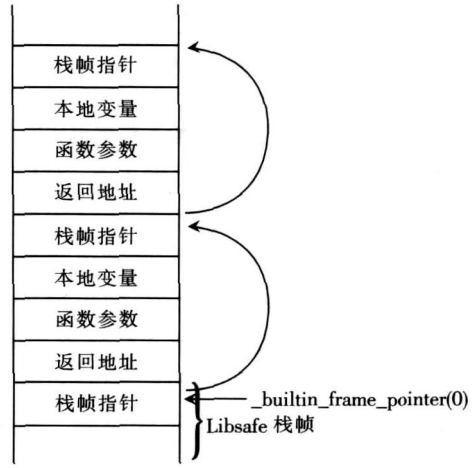


图 3 进程在栈中的结构示意图

每个栈帧的开头由指向前一个栈帧的栈帧指针所指示。 Libsafe从最顶端的栈帧开始,顺着栈帧指

针的指向找到 main函数的栈帧。最顶端的栈帧对应于 Libsafe函数本身。在这个 Libsafe函数中,栈帧指针通过 gcc函数 __builtin_frame_pointer(0)找到。函数返回地址就位于每个栈帧指针之前。

(3) 漏洞处理:如果函数不能安全执行,Libsafe给出警告,终止程序。

如果在对函数进行安全检查发现存在漏洞,Libsafe采取的措施是终止程序的运行。因为在无法保证漏洞函数之后的数据完整性,所以 Libsafe认为最安全的措施就是终止整个程序的运行。当然,对于返回地址和栈帧指针检查的漏洞,Libsafe可以选择继续执行程序,这是基于大部分的程序并不会有意图使用 %n覆写返回地址或者栈帧指针的假设。实际上,大多数格式化字符串攻击是由于处理包含 %n的用户输入而引起的,所以,只要包含 %n的用户输入不允许覆写内存,Libsafe就可以允许程序继续执行。

5　结束语

在对格式化字符串攻击的原理详细的分析的基础上,本文对格式化字符串攻击的常用攻击方法及其特点进行了归纳总结。由于格式化字符串漏洞被认为是程序员的编程错误所造成的,所以本文试图从源代码的角度对格式化字符串漏洞进行检测,从而在代码层次上消除格式化字符串漏洞的安全隐患。最后,对格式化字符串攻击的动态防范方法进行了比较研究,并且对动态防范工具 Libsafe的防范原理以及针对格式化字符串攻击采取的防范步骤进行了详细的分析,从而使得系统在程序运行过程中也能够防范格式化字符串攻击。

参考文献:

[1] WILANDER J, KAMKAR M. Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention [C]//10th Network and Distributed System Security Symposium. 2003: 149 - 162.

[2] NEWSHAM T. Format String Attacks [EB/OL]. http://www.lava.net/～newsham/format-string-attacks.pdf.

[3] TESO S T. Exploiting Format String Vulnerabilities version 1.2 [EB/OL]. http://www.crhc.uiuc.edu/～grier/projects/overflows/formatstring-1.2.pdf.

[4] RIVAS J B. Overwriting the. dtors section [EB/OL]. http://doc.bughunter.net/buffer-overflow/dtors.html.

[5] BULBA K. Bypassing StackGuard and StackShield [J]. Phrack Magazine, 2000, 10(38): 1 - 8.

[6] GERA R. Advances in format string exploiting [J]. Phrack Magazine, 2002, 11(59): 1 - 25.

[7] BOUCHAREINE P. Memory Bugs: _atexit [EB/OL]. http://doc.bughunter.net/buffer-overflow/atexit.html.

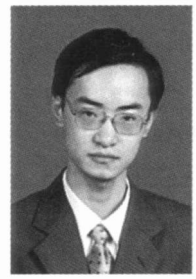
[8] CONOVER M. w00w00 Security Team. w00w00 on Heap Overflows [EB/OL]. http://www.w00w00.org/files/articles/heaptut.txt.

[9] RINGENBURG M, GROSSMAN D. Preventing format-string attacks via automatic and efficient dynamic checking [C]//Proceedings of the 12th ACM conference on Computer and communications security. 2005: 354 - 363.

[10] SINNADURAI S. A comparison of techniques to prevent Format String Attacks [EB/OL]. http://www.comp.nus.edu.sg/～saravan1/web/A comparison of Techniques to prevent Format String Attacks.pdf.

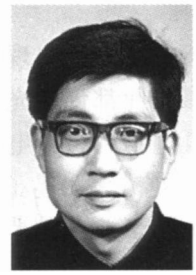
[11] TSAI T, SINGH N. Libsafe 2.0: Protection Critical Elements of Stacks [R]. USA: Basking Ridge, NJ07920, 2001: 1 - 16.

作者简介:



李　鹏 (1979 -),男,福建长汀人。南京邮电大学计算机学院讲师。2005年在南京邮电大学计算机应用技术专业获工学硕士学位。目前主要研究方向是计算机网络和信息安全。

王汝传 (1943 -),男,安徽合肥人。南京邮电大学计算机学院教授,博士生导师。(见本刊 2007年第 1期第 75页)



王绍棣 (1942 -),男,浙江宁波人。南京邮电大学计算机学院教授,博士生导师。1964年毕业于南京邮电学院无线电工程系。1987年以来先后在美国 University of Folorida信息研究中心作访问学者和在英国 Anglia Polytechnic讲学。目前研究方向是计算机在通信中的应用和数字信号处理。

(本文责任编辑:胡长贵)

阅读此文的还阅读了：

26. 僵尸网络检测和防范研究

27. SQL注入攻击及其防范检测技术的研究

28. 高压开关柜故障检测及防范措施

29. 基于信令的电话诈骗行为检测及防范研究

30. 配电线路故障检测与防范

31. ATP荧光检测技术在PIVAS感染控制中的风险防范研究

32. 配电线路故障检测与防范

33. SYN Flood(洪泛)攻击的检测与防范

34. 格式化字符串攻击检测算法

35. 网络安全问题的检测与防范

36. SQL注入的检测与防范策略研究

37. 配电线路故障检测与防范

38. 论人文社会科学研究的防范检测机制

39. 浅谈拒绝服务攻击、检测与防范

40. DDoS攻击检测与防范策略研究

41. 电气火灾的防范及安全检测(下)

42. 网络攻击的防范与检测技术研究

43. 安全技术防范工程检测流程与方法研究

44. 浅析Sniffer的检测方法和防范对策

45. 检测实验室风险评估与防范

46. 网络监听检测及防范技术研究

47. SQL注入攻击及漏洞检测防范技术

48. ARP攻击的检测识别与防范

49. 文字复制比检测的误区及其防范

50. 高压开关柜故障检测及防范措施