

内核入侵隐藏技术的研究与实现

李 鹏, 王汝传

(南京邮电大学 计算机学院, 江苏 南京 210003)

摘 要:对不支持可加载模块的系统内核入侵代码隐藏技术进行了研究。比较了内核支持可加载模块和内核不支持可加载模块的内核入侵的方法区别,阐述了内核入侵在求解系统调用表的地址、kmalloc 函数的地址、编写函数分配内核空间内存、编写入侵代码、汇编代码处理、提取代码段及重定位信息、分配内核空间的内存、代码写入分配的内存等八个主要流程。在总结入侵代码隐藏技术原理的基础上,给出了入侵代码隐藏文件信息、进程信息和网络连接技术的详细设计实现。

关键词:内核入侵;信息隐藏;进程隐藏;网络安全

中图分类号:TP393.08

文献标识码:A

文章编号:1673-629X(2011)03-0170-04

Research and Realization of Concealment Technology Based on Kernel Hacking

LI Peng, WANG Ru-chuan

(College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: Makes research on code concealment technology based on kernel hacking module without loadable kernel. It firstly analyzes the difference in kernel hacking method between module with loadable kernel and module without loadable kernel. Then, it expatiates the eight main steps of kernel hacking, which includes computing the address of system call table, calculating the address of kmalloc function, programming kernel space memory assignment function, implementing invasion codes, processing assembly code, extracting code segments and relocation information, assigning kernel space memory, and writing assigned memory. Based on summarizing the concealment technical principles of kernel hacking codes, finally designs and implements the kernel hacking technologies, which includes concealing file information, process information and network connections.

Key words: kernel concealment; information hide; process hide; network security

0 引 言

黑客在利用系统漏洞成功侵入一台主机并取得特权用户权限后,一般会在主机上留下一些恶意代码,如监听程序、木马程序等,为了使这些代码不被系统管理员发现,它们往往要对当前运行的内核做一些手脚,即内核入侵^[1]。对内核入侵技术的研究,可以更好地防御该类型的攻击。

最早的是 Netcat^[2] 的 UNIX 版本,这个工具仍然是最流行的 UNIX 系统后门,尽管它有很多合法的用途,但是 Netcat 也常作为一个后门被滥用。Creed 发布的内核级 RootKit^[3-5] 工具 Knark^[6],是基于 Linux 系统内核控制的,Knark 包含一个用于修改 Linux 内核的完整工具包,攻击者可以非常有效地隐藏文件、进程和网络行为。John Heasman 发表论文详细阐述了一种 PCI RootKit^[7]。入侵隐藏技术演进的发展趋势:从最开始简单的后门,到内核级 RootKit,新的入侵隐藏技术越来越复杂,越来越隐蔽;开始的入侵隐藏技术,大多数都围绕着后门和木马进行,在最近几年,主要集中在内核级的系统开发上。在不同的内核下,内核入侵的难易程度不一样,现对不同情形下内核入侵的方法做一个简要说明。

(1) 内核支持可加载模块。

在这种情形下,用户自己编写的代码在编译成目标文件并使用 insmod 命令插入内核后,代码运行在内核空间,可访问内核空间的内存,在这种情形下程序的

收稿日期:2010-08-18;修回日期:2010-11-10

基金项目:国家自然科学基金项目(60973139,60773041,61003039,61003236);江苏省科技支撑计划(工业)项目(BE2010197, BE2010198);江苏省现代服务业发展专项资金;国家和江苏省博士后基金(0801019C,20090451240,20090451241,20100471353,20100471355);江苏高校科技创新计划项目(CX09B_153Z,CX10B-196Z,CX10B-197Z,CX10B-198Z,CX10B-199Z,CX10B-200Z);江苏省六大高峰人才项目(2008118);江苏省计算机信息处理技术重点实验室基金(2010)

作者简介:李 鹏(1979-),男,讲师,博士生,研究方向为计算机网络和信息安全;王汝传,教授,博士生导师,研究方向为计算机软件、计算机网络、信息安全、移动代理和虚拟现实技术。

装入由系统提供。而且,在内核支持可加载模块的系统中,一般还会导出符号表,如果自己要用的某个系统函数或变量正好在导出的符号表中,在自己的代码中就可直接使用此函数或变量,此时系统实际上完成了类似编译器的连接器功能。

(2) 内核不支持可加载模块。

内核不支持可加载模块,说明系统不能将自己编写的程序装入内核空间执行,这意味着将自己的代码装入内核空间这个步骤必须自己编程实现。在将可执行代码装入内存的过程中,必不可少的步骤是代码的重定位。而对一般的用户程序而言,代码的装入由系统完成,代码编写者根本就不用关心重定位的问题。

不难看出,在内核支持可加载模块的情况下,进行内核入侵难度较小。正因为如此,系统管理员可能会基于安全性的考虑而使用不支持可加载模块的内核。文中的重点放在后一种情况,该攻击方法对支持可加载模块的内核同样适用。

1 内核入侵流程原理

本节详细分析入侵流程中各步骤的实现原理。某些步骤中会涉及到读写内存的操作,这些操作是通过读写设备文件/dev/kmem来实现的,在Linux中,文件/dev/kmem是系统虚拟内存的镜像,通过修改这个文件,就能达到修改系统内存的目的,一般情况下,这个文件的权限是root用户可读写。

1.1 求出系统调用表的地址

在Linux 2.4.18以前的内核中,系统调用表^[8]地址是导出的,程序员编写的模块程序可直接使用系统调用表地址sys_call_table变量,只需声明它为外部变量即可。在插入模块的过程中,操作系统的连接器ld会自动解析sys_call_table变量,赋给它正确的地址值。因此在实现时只需将sys_call_table变量的值赋给自定义变量syscall_table即可。

参考文献[9,10]中提出了通过模式匹配的方法在内存中搜索系统调用表的地址的方案。其思想是搜索Linux系统system_call函数的前100个字节,找到call指令,call指令后的四个字节就是系统调用表地址。system_call函数的地址可通过如下操作得到,首先通过汇编指令“sidt”得到中断描述符寄存器的值,中断描述符寄存器中保存有中断描述符表的首地址;然后根据中断描述符首地址得到0x80号中断描述符的地址;接着读取内存0x80号中断描述符,这个中断描述符中就保存有system_call函数的地址。

1.2 求出kmallocc函数的地址

Silvio提出了求kmallocc函数地址的方法,但它不能保证100%的成功率^[10]。文中提出了另外一种实

现方案,在多种操作系统的内核上均可取得正确的结果。

在内核中调用kmallocc需要两个参数,kmallocc函数原型为:

```
void * kmalloc (size_t size, int flags);
```

要在整个内核中进行模式匹配搜索无疑是不现实的,既然已经知道了系统调用表的地址,就知道了所有系统调用处理函数的地址。要在系统调用处理函数的二进制代码中寻找kmallocc函数地址,首先必须选择一个系统调用处理函数,可选择源代码目录中,在fs/select.c文件中定义的sys_poll函数。实现流程如下:

(1)根据系统调用表,求出sys_poll函数的地址(即syscall_table[__NR_poll])。

(2)读出内存中从sys_poll函数地址开始的800个字节。

(3)在读出的800个字节中搜索0xc7*****f0010000,其中*标识任意16进制数字。

(4)在接下来的32个字节中查找0xe8。

(5)0xe8后的四个字节即为kmallocc函数的相对偏移,对这四个字节进行验证,若计算得到的绝对地址大于0xc0000000,则地址为kmallocc函数的地址,结束;否则跳转到(4)。

1.3 编写函数get_kmm()在用户态程序中分配内核空间内存

找到kmallocc函数的地址后,接下来就可实现在内核中申请一块内存。必须注意,在用户态程序中是不能直接调用kmallocc函数的。

(1)编写函数get_kmm(),这个函数通过调用kmallocc来分配内核空间的内存,编写的这个函数的二进制代码执行时不需要重定位。

(2)将这个函数编译成目标代码,并用这段目标代码中的代码段部分来覆盖某个不常用的系统调用,例如olduname,覆盖操作通过写/dev/kmem文件实现。

(3)在用户空间进行olduname系统调用,此时执行的就是内核空间中完成分配内存任务的那段代码了。

(4)得到内核空间的内存后,恢复原有系统调用olduname代码。

1.4 编写入侵代码

这个步骤需要用到syscall_table值。编写的入侵代码主要是系统调用替换代码,为了让这些函数能替代系统调用,即用户空间的程序进行系统调用时,执行的是本程序的替代函数,需要在系统进入系统调用函数的过程中的某个步骤做修改。下面分析系统如何由用户态切换到系统态并执行系统调用代码。

Linux的系统调用都通过入口system_call进入内

核,使用中断向量(0x80)实现系统调用,当用户态进程执行一条 int 0x80 汇编指令时,CPU 就切换到内核态,并开始执行 system_call 函数,再通过系统调用表来取得相应系统调用的地址执行。分析上述过程,可得出三种方案:

(1)新建一个系统调用表,在保留原有系统调用表大部分内容的前提下,以被替换系统调用的编号为索引加入自己编写的替换函数的地址。再修改 system_call 函数的二进制代码,使其使用新的系统调用表。

(2)修改系统调用表,修改要替换的系统调用对应的表项,指向自己编写的替换函数。

(3)修改原有系统调用函数的二进制代码,在代码开头加上跳转指令,跳转地址为自己编写的替换函数的地址。

上述三种方案的性能比较,如表 1 所示。

表 1 三种方案的比较

比较方案	实现难度	对原调用影响	攻击隐蔽程度
方案一	容易	不影响	一般
方案二	一般	影响	较好
方案三	复杂	影响	最好

1.5 对汇编代码进行处理,将处理后的代码进行汇编

入侵代码一般会很复杂,编译后的汇编代码往往由好几个段构成,必须将代码中的几个段合并为一个代码段。为了保存入侵代码的相关信息供接下来的步骤使用,还要对汇编代码进行一定的修改。修改完成后,加入一定辅助信息,并进行汇编,得到目标代码。

1.6 对目标代码进行分析,提取出其中的代码段及重定位信息

要提取出目标代码中的代码段,需要知道代码段的偏移和大小,有两种方案:

(1)自己编程对目标代码文件进行解析,这要求编程者熟悉 ELF 文件格式。

(2)使用相应参数的 objdump 命令来读取段信息。

编译之后的目标代码中存在需要重定位的地址,对重定位的解决一般有两种方法。其原理阐述如下:

1) 入侵代码自身完成重定位。

入侵代码中有一段初始化代码,初始化代码首先执行一个 call 指令,将下一条指令地址压栈,这时栈顶保存的就是当前代码执行时的一个内存地址了。因此,只需先将入侵代码编译得到的汇编代码,在汇编代码中对重定位变量的访问方式进行修改,就可使入侵代码自身具有重定位能力。

2) 载入程序完成重定位。

方法 2 直接对目标代码进行操作。Linux 系统中目标代码的文件格式为 ELF,目标代码中保存有重定

位表,载入程序可以根据目标代码中保存的重定位表对代码段进行重定位。与获取目标代码中代码段信息的方法类似,获取重定位表可以编程对目标代码文件进行解析,也可以使用 Linux 下的带相应参数的 objdump 命令来读取重定位表。

1.7 调用 get_kmm() 分配内核空间的内存和代码重定位

调用函数 get_kmm() 本质上是执行函数 get_kmm() 的代码,这是通过先替换一个系统调用 sys_olduname() 执行代码,再在用户空间调用系统调用 olduname() 来实现的。

计算出目标代码中代码段的长度,根据这个长度信息决定分配内存空间的大小。调用 get_kmm() 分配内存,得到内存的首地址(大于 0xc0000000),载入程序根据内存首地址以及上一步骤得到的重定位信息对代码段中的二进制代码进行重定位。

1.8 将重定位后的代码写入分配的内存

通过写设备文件/dev/kmem 实现,可自己编写单独程序,配合对应的 make 文件来实现。

2 入侵代码隐藏技术分析

通过分析,可得出入侵代码实际需要隐藏三个方面的内容:攻击过程的文件信息、进程信息,以及网络连接^[11,12]。

2.1 隐藏文件信息的实现分析

Linux 系统中用来查询文件或目录信息的系统调用是 sys_getdents() 或 sys_getdents64(),后者用于较新的内核,通过 strace 命令可以观察 ls 是通过 sys_getdents() 或 sys_getdents64() 来执行操作的。因此,如果用自己的程序替换系统的 sys_getdents() 和 sys_getdents64(),自己的程序在实现原有系统调用功能的同时,对显示的信息进行过滤,就可达到隐藏文件和目录信息的目的。

2.2 隐藏进程信息的实现分析

对于 Linux 系统,虽然其没有直接查询进程的系统调用,但是在/proc 目录下有以数字做为目录名的目录,这些数字代表的含义是进程号,每一个系统当前运行的进程都会在这里出现和它对应的目录,该目录下的各个文件就包含了该进程的各项信息。分析 ps 的源代码可看出,ps 命令的代码也是调用 sys_getdents() 或 sys_getdents64() 对/proc 目录下的目录和文件进行遍历,找出其中进程对应的目录,并输出各进程对应目录中的信息。因此,要隐藏进程信息,只需在上述替换系统 sys_getdents() 和 sys_getdents64() 函数的代码中加入相应的判断和过滤代码即可。

对/proc 下以系统当前进程号为目录名的目录来

说,这些目录的 inode 号的低 16 位为 2,高 16 位为进程号,从而保证不同的进程有不同的 /proc 文件系统的 inode 与之对应。这就是对 /proc 下目录进行判断的原理。

2.3 隐藏网络连接的分析

Linux 环境中 netstat 命令可用来查看网络连接的有关信息,这个命令会读取目录 /proc/net/ 下的几个文件,如 /proc/net/tcp, /proc/net/udp, /proc/net/raw, 来获得网络连接的信息。与 ps 和 ls 命令不同的是,此命令会读特定的文件,不会调用 sys_getdents() 或 sys_getdents64(), 因此隐藏文件和进程的方法就失效了。为实现网络连接信息的隐藏,需要替换系统原有的 sys_open(), sys_read() 和 sys_close() 系统调用。

3 入侵代码隐藏技术的实现

3.1 隐藏文件和进程信息的实现

重新定义两个系统调用 new_getdents() 和 new_getdents64(), 用来替换原有系统调用 sys_getdents() 和 sys_getdents64(), 两个函数的实现步骤几乎一样, 这里详述 new_getdents() 的流程。

sys_getdents() 的原型如下:

```
int sys_getdents(unsigned int fd, struct dirent *
dirp, unsigned int count);
```

自定义的 new_getdents() 函数与 sys_getdents() 的原型一样。其实现流程如图 1 所示。

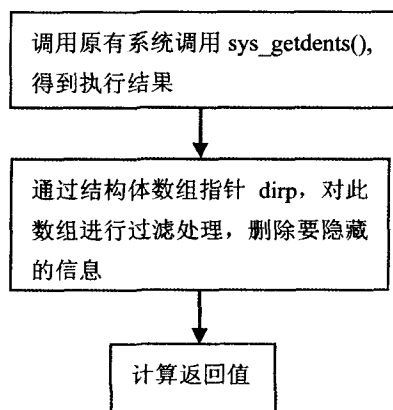


图 1 自定义系统调用 getdents 流程图

3.2 隐藏网络连接的实现

当某个进程甲,如 netstat 命令,调用 open() 来打开这三个文件中的某一个时,就对该文件的信息进行过滤,过滤后的信息就保存在该结构体中。接下来,如果进程需调用 read() 读取文件中的内容时,直接读取 net_struct 结构体中的信息,返回的内容是已经过滤的,这样就达到了隐藏的目的。

程序中必须重新编写 open(), read() 两个系统调

用,为了进行相关资源的释放工作,还需重新编写 close(), 修改的系统调用命名为 new_open(), new_read() 和 new_close(), 以 new_open() 和 new_read() 为例,其执行流程如图 2、图 3 所示。

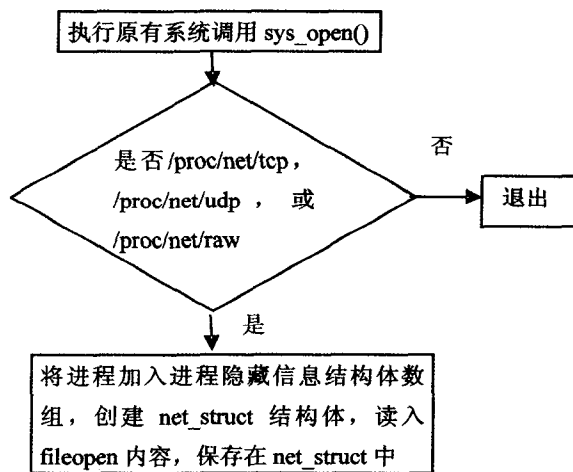


图 2 new_open() 的执行流程

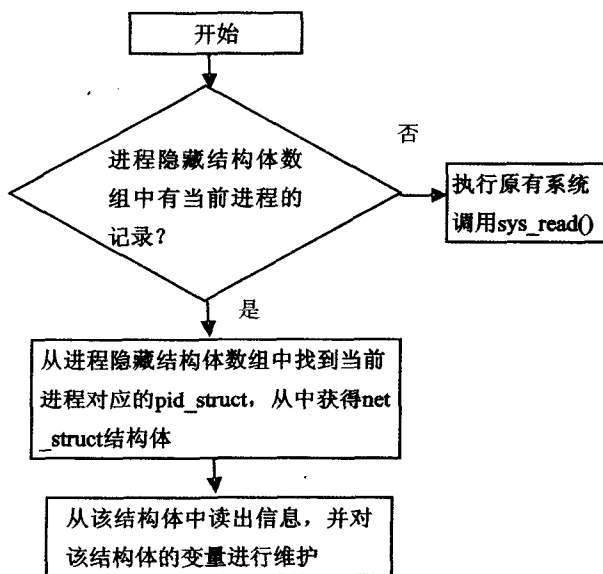


图 3 new_read() 的执行流程图

4 结束语

文中深入研究和分析了基于内核层次的内核隐藏技术。详细阐述了常见的控制系统内核的方法,并就基于内核层次的文件信息隐藏、进程信息隐藏、网络连接信息隐藏进行了实现原理和实现方法的分析。为进一步对内核入侵攻击进行防御打下了技术基础。

参考文献:

- [1] McClure S, Scambray J, Kurtz G. Hacking Exposed: Network (下转第 177 页)

该向量,这样则可以进一步去除光照变化的影响。

3 基于 SIFT 的水印方案

由上面的 SIFT 原理的分析和实践步骤可以看出, SIFT 基于尺度空间理论保证了缩放不变性,而基于特征点的特征梯度方向保证了旋转不变性,又因为特征点的局部性,可以保证平移的不变性,因此,基于 SIFT 的数字水印方案可以很好的抵抗 RST 攻击,此外,由于数字使用的具体应用,可以省去 SIFT 的一些步骤,如特征向量归一化,具体方案^[10~13]如下:

水印嵌入:查找原始图像的 SIFT 特征点,并对特征点进行筛选(如采用去除特征尺度较小和过大点的圆形特征区域方法)以获得适量的局部特征区域,在局部特征区域内进行水印的空域或变换域水印嵌入操作。

水印检测:查找带检测图像的 SIFT 特征点,依据 SIFT 特征点的尺度等信息,进行水印同步,即获得与嵌入时相同的特征区域,然后在各个特征区域内,逐一检测水印的存在性,取各个区域检测存在可能的最大值,如果超过预先设定的阈值,则认为水印存在,否则水印不存在。

4 结束语

详细分析了 SIFT 的原理及特点,指出了其基于尺度空间原理抵抗缩放变换,基于特征点特征梯度而抵抗旋转变换的本质,并用 Matlab 进行了优化实现,最后给出了可行的抗 RST 攻击水印方案。

参考文献:

- [1] 刘九芬,黄达人,黄继武. 图像水印抗几何攻击研究综述[J]. 电子与信息学报,2004,26(9):1495-1503.
- [2] 邓峰森,王炳锡. 基于特征点的抗几何失真数字图像水印[J]. 信号处理,2005,21(1):12-16.
- [3] 王向阳,邬俊,侯丽敏. 基于图像特征的数字水印算法研究[J]. 中国图象图形学报,2006,11(11):1562-1565.
- [4] 王春桃,倪江群,黄继武,等. 结合 Zernike 矩和模板具有 RST 不变性的 DWT-HMM 鲁棒水印算法[J]. 中国图象图形学报,2008,13(7):1250-1257.
- [5] 吴健珍,谢剑英. 基于支持向量机同步的自适应水印检测方法[J]. 上海交通大学学报,2006,40(3):480-484.
- [6] 李海峰,宋巍巍,王树勋. 基于 Contourlet 变换的稳健性图像水印算法[J]. 通信学报,2006,27(4):87-94.
- [7] Lowe D G. Distinctive image features from scaleinvariant keypoints[J]. International Journal of Computer Vision, 2004,60(2):91-110.
- [8] Lindeberg T. Scale-Space Theory: A Basic Tool for Analysing Structures at Different Scales[J]. Journal of Applied Statistics,1994,21(2):224-270.
- [9] Vedaldi A, Fulkerson B. VLFeat: An open and portable library of computer vision algorithms[EB/OL]. 2008. <http://www.vlfeat.org/>.
- [10] Dong P, Brankov J G, Galatsanos N P, et al. Digital watermarking robust to geometric distortions[J]. IEEE Transactions on Image Processing,2005,14(2):2140-2150.
- [11] Tang C W, Hang H M. A Feature-based Robust Digital Image Watermarking Scheme[J]. IEEE Transactions on Signal Processing,2003,51(4):950-959.
- [12] Lee H Y, Kim J T, Lee H Y, et al. Content-Based Synchronization Using the Local Invariant Feature for Robust Watermarking[J]. Lecture Notes in Computer Science,2004,3325:122-134.
- [13] Seo J S, Yoo C D. Localized Image Watermarking Based on Feature Points of Scale-Space Representation[J]. Pattern Recognition,2004,37(7):1365-1375.
- [1] 刘九芬,黄达人,黄继武. 图像水印抗几何攻击研究综述[J]. 电子与信息学报,2004,26(9):1495-1503.
- [2] Heasman J. Implementing and Detecting a PCI Rootkit[R]. [s. l.]: An NGSSoftware Insight Security Research (NISR) Publication,2006:1-15.
- [3] Kanclirz J. Netcat Power Tools [M]. USA:SYNGRESS,2008.
- [4] 张登银,高德华,李鹏. 一种新的注册表隐藏 Rootkit 检测方案[J]. 江苏大学学报(自然科学版),2010,31(3):328-333.
- [5] 左黎明,蒋兆峰,汤鹏志. Windows Rootkit 隐藏技术与综合检测方法[J]. 计算机工程,2009,35(10):118-120.
- [6] 石晶翔,陈蜀宇,黄哈辉. 基于 Linux 系统调用的内核级 Rootkit 技术研究[J]. 计算机技术与发展,2010,20(4):175-178.
- [7] Knark [EB/OL]. 2010-07-30. <http://en.wikipedia.org/wiki/Knark>.
- [8] 张步忠,金海平. Linux 内核系统调用扩展研究[J]. 计算机技术与发展,2007,17(5):163-165.
- [9] Sd, Devik. Linux on-the-fly kernel patching without KM[J]. Phrack, 2001,11(58):1-16.
- [10] Silvio C. Runtime kernel kmem patching[EB/OL]. 1998-09. <http://vx.netlux.org/lib/vsc07.html>.
- [11] 文伟平. 恶意代码机理与防范技术研究[D]. 北京:中国科学院软件研究所,2004:40-53.
- [12] 霍格兰德. ROOTKITS——Windows 内核的安全防护[M]. 北京:清华大学出版社,2007.

(上接第 173 页)

- [1] Security Secrets and Solutions [M]. Fourth Edition. [s. l.]: McGraw-Hill Osborne Media,2003:338-342.
- [2] Kanclirz J. Netcat Power Tools [M]. USA:SYNGRESS,2008.
- [3] 张登银,高德华,李鹏. 一种新的注册表隐藏 Rootkit 检测方案[J]. 江苏大学学报(自然科学版),2010,31(3):328-333.
- [4] 左黎明,蒋兆峰,汤鹏志. Windows Rootkit 隐藏技术与综合检测方法[J]. 计算机工程,2009,35(10):118-120.
- [5] 石晶翔,陈蜀宇,黄哈辉. 基于 Linux 系统调用的内核级 Rootkit 技术研究[J]. 计算机技术与发展,2010,20(4):175-178.
- [6] Knark [EB/OL]. 2010-07-30. <http://en.wikipedia.org/wiki/Knark>.