一种基于审计的入侵检测模型及其实现机制

刘海峰, 卿斯汉, 蒙 杨, 刘文清
(中国科学院软件研究所, 北京 100080; 中国科学院信息安全技术工程研究中心, 北京 100080)

摘 要: 文中对基于系统调用序列的入侵检测进行了深入的研究, 提出了一种新的基于审计事件向量的入侵检测模型(AUDIDS). 这一模型除了具有系统调用序列入侵检测模型的优点外, 比之已有的模型具有更丰富的语义及更高的效率. 针对此模型, 文中还给出了此模型在 linux 上的实现机制, 实现了审计事件的定义、收集和存储, 并对正常库的存储及匹配方法进行了改进.

关键词: 审计; 入侵检测; linux; 计算机安全
中图分类号: TP309 文献标识码: A 文章编号: 0372-2112 (2002) 08-1167-05

A New Audit-Based Intrusion Detection Model and Its Implement Mechanism

LIU Hai-feng, QING Si-han, Meng Yang, LIU Wen-qing
(*Institute of Software Chinese Academy of Sciences, Beijing 100080, China; Engineering Research Center for Information Security Technology Chinese Academy of Sciences, Beijing 100080, China*)

Abstract: Intrusion detection models based on system call sequence are discussed, and a new intrusion detection model based on audit event vector which is named "AUDIDS" is presented. This model has not only the merits of the previous IDS model but also richer semantics and higher efficiency. We describe its implementation mechanism on Linux which defines, collects and stores audit event and improves the storage and matching of normal database.

Key words: audit; intrusion detection; linux; computer security

1 引言

随着 Internet 的飞速发展, 信息资源的共享程度进一步加强, 随之而来的信息安全问题日益突出. 信息系统的安全一般采用标识与鉴别、访问控制、加密技术等安全机制来加以保护, 内部网络一般采用防火墙技术来保护. 但是现在的系统越来越复杂, 系统中总是存在着各种各样的漏洞[1,2], 以及一些人为因素导致的漏洞(比如: 没有合理配置防护墙规则, 口令比较弱, 系统的安全配置没有随系统的改变而相应地改变等), 这些都有可能被黑客利用[2], 入侵到系统中去. 并且, 即使有了防火墙也不可能完全保护内部网的安全[3]. 因此, 除了传统的安全保护之外, 对入侵的实时报警也是非常必要的.

本文在基于系统调用序列的入侵检测模型的基础上提出了一种新的基于审计事件向量的入侵检测模型(AUDIDS). 此模型通过在系统调用和应用程序的基础上, 通过定义审计事件以作为检测入侵的基本单位, 以提高入侵检测的准确及效率; 在实现方面, 通过改进匹配顺序, 以简化模型的复杂度.

2 基于系统调用序列的入侵检测模型

在 1996 年, Stephanie Forrest 提出了一种通过监视特权进程的系统调用序列的入侵检测模型[4]. 他们的工作表明每一个程序的正常行为都可以由该程序正常运行时产生的系统调用序列来描述, 把这些正常的序列存放在库中, 称之为正常库. 建立起正常库后, 监视该程序的运行状态, 当该程序的产生的系统调用序列偏离了正常库时, 则可能产生了安全威胁. 1997 年, Steven A. Hofmeyr 在 Forrest 的基础上对正常库的结构进行了改进, 使库的结构更为紧凑, 并且改进了两个序列串匹配的方法, 使效率得以提高[5].

在过去的几年里, 基于统计的学习方法得到发展, 有基于频率统计的方法、数据挖掘的方法、有限自动机的方法(隐蔽马尔可夫图的方法)[6,7], 这些方法在模型的描述看上去更为准确一些, 但是除了基于隐蔽马尔可夫图的模型(计算代价比较高) 比 Forrest 的模型好一些外, 其它的模型均与其差不多, 这足以说明由于系统调用序列内在的规律性, 简单的模型一样工作的很好[8].

Forrest 的模型由于只关注系统调用本身而不记录系统调用的参数, 这样必然导致少记录很多有用的信息, 但是, 如果把系统调用的参数也记录下来, 则正常库将庞大无比、模型也将变得非常复杂; 此外, 对于应用程序特别是象 sendmail 这样大的程序在运行时产生大量的系统调用, 这也会使正常库变得非常大, 实时检测的效率降低.

收稿日期: 2001-07-20; 修回日期: 2002-04-20
基金项目: 国家自然科学基金(No. 60083007); 国家重点研究发展规划 973 资助项目(No. G1999035810)

3 一种基于审计事件向量的入侵检测模型——AUDIDS

3.1 AUDIDS 模型的建立

在 AUDIDS 模型中, 通过实时地收集审计事件序列来实现对入侵的检测. 模型共分为两部分. 首先通过大小为 k 的滑动窗口监控系统正常状态下产生的审计事件序列, 把这些审计事件序列以 k 个为一组存放在正常库中, 从而得到系统的正常状态, 这可以称为系统的学习状态, 即描述系统正常状态的过程; 然后进入第二步, 检测入侵的阶段, 以正常库为标准用滑动窗口对系统产生的审计事件序列进行实时监控, 如果发现偏离正常库的审计事件序列, 则发现异常. 整个模型的描述可以用图 1 表示:

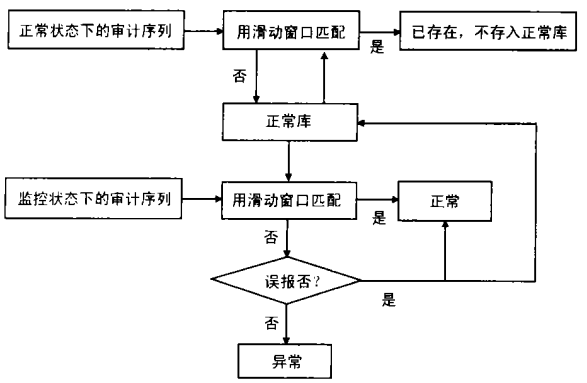

图 1 AUDIDS 模型框架

对于 AUDIDS 模型, 定义审计事件、审计事件向量、正常库、匹配和异常, 以更精确的描述此模型.

定义 1(审计事件): 定义系统中所有与安全有关的活动为审计事件. 用 $E_i(0 \leqslant i \leqslant 194)$ 表示一个审计事件, 在 linux 实现中我们共定义了 195 个审计事件.

定义 2(审计事件向量): 设 k 为一滑动窗口的大小, E_0, E_1, …, $E_m(m \geqslant k)$ 为任意的审计事件序列, 用滑动窗口在审计事件序列上滑动, 则 $f(k; E_0, E_1, \cdots, E_m)$ 表示所有在滑动窗口内的审计事件向量, 即:

$$f(k; E_0, E_1, \cdots, E_m) = \{(E_0, E_1, \cdots, E_{k-1}), (E_1, E_2, \cdots, E_k), \cdots, (E_{m-k+1}, E_{m-k+2}, \cdots, E_m)\}$$

定义 3(正常审计事件向量): 定义不会产生异常情况的审计事件向量为正常审计事件向量, 记为 $\boldsymbol{R}$.

定义 4(正常库): 设 $\boldsymbol{R}_i$ 为一个正常审计事件向量, 则正常库定义为 $\boldsymbol{NR} = \{\boldsymbol{R}_0, \boldsymbol{R}_1, \cdots, \boldsymbol{R}_n\}$

定义 5(匹配): 设获得审计事件序列 $E_0, E_1, \cdots, E_m$, 对任意向量 $\boldsymbol{u} \in f(k; E_0, E_1, \cdots, E_m)$, 若 $u \in \boldsymbol{NR}$ 则定义此审计事件序列为匹配; 若存在 $v \in f(k; E_0, E_1, \cdots, E_m)$, 但是 $v \in \boldsymbol{NR}$, 则定义此审计事件序列为不匹配.

3.2 对 AUDIDS 模型的讨论

AUDIDS 模型沿用了基于系统调用序列的入侵检测的基本思想, 即, 都是通过监控系统的特征序列来实现入侵检测的. 但是比基于系统调用序列的入侵检测, AUDIDS 除了具有前者的优点外, 还因为定义了审计事件, 使得 AUDIDS 模型具有更丰富的语义和更高的效率.

(1) 更丰富的语义 以往基于系统调用序列的入侵检测只关注系统调用本身而不记录系统调用的参数、时间、用户、成功与否等信息, 这样必然导致用以分析入侵的数据少了一些有用的信息. 单纯的系统调用号组成的序列在语义上比较简单, 使得以往的模型效率比较低, 当滑动窗口比较小的时候, 系统的误报率比较高.

在 AUDIDS 模型中, 根据系统调用号、系统调用参数、用户的安全级别和客体的安全级别等与安全有关的信息定义了审计事件, 具体的实现见 4.1. 这样审计事件比单纯的系统调用号具有更丰富的语义, 同样长度的审计事件序列比系统调用号序列更能精确地描述事件发生的本质. 因此, 在滑动窗口同样大小的情况下, 系统的误报率也大大降低了.

(2) 更高的效率 对于大的应用程序, 在运行的过程中产生大量的系统调用, 以往基于系统调用序列的入侵检测对产生的所有系统调用都进行记录、匹配, 这将大大降低了整个系统的运行效率.

在 AUDIDS 模型中, 针对一些常用的应用程序、特权命令, 比如 sendmail, 为这些程序的特殊状态、特殊情况专门定义了审计事件, 每个审计事件都表示程序的一种特殊状态或特殊情况, 在监控这些程序的时候只审计到为这些程序定义的审计事件, 而审计不到为整个系统定义的审计事件. 这样在简化审计事件序列的同时, 使审计事件序列的语义更明确; 由于简化了审计事件序列, 使得整个系统的效率得以进一步提高.

4 新模型 AUDIDS 在 linux 上的实现

Linux 系统运行态分为用户态和核心态两种. 用户程序只能通过系统调用陷入核心, 才能存取系统资源(文件、目录、设备等), 运行完系统调用, 又返回用户态. 如果在此处(称作审计点)增加审计控制, 就可以成功地审计系统调用, 也就可以成功地对所有安全有关的事件进行监控.

4.1 审计事件的定义

本入侵检测系统采用系统审计采集来的数据, 系统审计的基本单位就是审计事件, 在本系统中共定义了 195 个审计事件, 其中一部分是在关键的系统调用的基础上进行了扩充; 另外一部分, 是在一些特权命令和关键的程序的基础上建立的.

用户活动按各自的性质不同, 被视为不同的审计事件. 常见的审计事件中有象从 open、exec 这样的系统调用中演化而来. 举例来说, 对于 open 系统调用, 根据系统调用的参数不同定义了 ADT_CREATE、ADT_OPEN_RD 和 ADT_OPEN_WR 三个审计事件. 由于审计了 open 系统调用, 则不审计 read 和 write 系统调用.

还有一些与系统安全关系密切的系统命令和特权命令(如 login, su, passwd 等), 如果入侵检测模块对其调用的每一个基于系统调用的审计事件均进行监控, 则审计到的信息过于琐碎, 这将影响整个入侵检测系统的效率和准确性. 所以应该针对系统命令或特权命令中特殊的情况定义审计事件. 例如, 对于 login 程序, 根据程序中的一些关键点来定义 ADT_

BAD_AUTH(错误的认证)、ADT_BAD_LVL(错误的安全级别)、ADT_LOGIN(成功的登录)等. 这样监控到的信息比直接监控该程序所调用的系统调用要更简洁、语义更明确.

4.2 审计点的处理

审计点分为两种,一种是在系统调用总入口处;另一种是分布在特权程序中的审计点. 由前面可知,通过监控系统调用的总入口就可以知道特权程序作了那些事情. 可以在系统调用总入口处插入系统调用分析函数(void adt_auditchk(int scall, void * ap)),其中 scall 是系统调用号, ap 为系统调用参数),通过分析系统调用参数,判断用户实际对应的审计事件. adt_auditchk 的基本流程如下:

(1)根据 scall 知道程序具体调用的系统调用,则调用具体的系统调用分析函数. 比如 open 系统调用具体对应的分析函数是 int adt_openchk(void * ap, int evt).

(2)进入具体的系统调用分析函数,则根据传进来的系统调用参数决定到底选用哪一个审计事件. 比如 open 系统调用,则根据系统调用参数决定与 ADT_CREATE、ADT_OPEN_RD 和 ADT_OPEN_WR 哪一个审计事件对应.

(3)决定了对应的审计事件后,判断该进程是否需要审计,如果需要则通过后台审计进程 audit 并发地写到硬盘中去. 并把该审计事件挂到内存中对应的审计事件序列中. 当审计事件串达到系统配置规定长度的时候,则唤醒睡眠在后台的进程 alertd 判断系统是否是处在学习状态,如果是(建立正常库的时候),则写入该程序对应的正常库中;否则与对应的正常库中的数据进行匹配,如果发生异常则报警.

对于在特权程序的审计点,在特权程序中各个关键点插入新增的系统调用 auditdmp,通过它将采集到的审计事件送到核内,以后的处理也同在系统调用总入口采集的数据一样.

4.3 正常库的定义

建立系统正常库的过程就是对系统正常状态的一个学习过程,就是在系统正常状态下对特权程序进行监控,以收集正常状态下的审计事件序列. 为了对相应的特权命令建立正常库,我们用一个大小为 k 的滑动窗口在相应审计事件序列上取值,记录下在滑动窗口内各审计事件的先后顺序. 例如:取 $k=4$,正常审计事件序列如下: ADT_EXEC, ADT_BRK, ADT_OPEN_RD, ADT_FSTAT, ADT_MMAP, ADT_CLOSE, ADT_OPEN_RD, ADT_MMAP, ADT_MUNMAP

表 1　向前看匹配的正常库

	位置三	位置二	位置一	现在的位置
				ADT_EXEC
			ADT_EXEC	ADT_BRK
		ADT_EXEC	ADT_BRK	ADT_OPEN_RD
R_0	ADT_EXEC	ADT_BRK	ADT_OPEN_RD	ADT_FSTAT
R_1	ADT_BRK	ADT_OPEN_RD	ADT_FSTAT	ADT_MMAP
R_3	ADT_OPEN_RD	ADT_FSTAT	ADT_MMAP	ADT_CLOSE
R_4	ADT_FSTAT	ADT_MMAP	ADT_CLOSE	ADT_OPEN_RD
R_5	ADT_MMAP	ADT_CLOSE	ADT_OPEN_RD	ADT_MMAP
R_6	ADT_CLOSE	ADT_OPEN_RD	ADT_MMAP	ADT_MUNMAP

在正常审计事件序列上滑动窗口,记录下每一个审计事件后面第一个、第二个,直到第 k 个. 对于上面审计事件序列,得到如下的审计向量:

即:

f(4; ADT_EXEC, ADT_BRK, ADT_OPEN_RD, ADT_FSTAT, ADT_MMAP, ADT_CLOSE, ADT_OPEN_RD, ADT_MMAP, ADT_MUNMAP)

= {(ADT_EXEC, ADT_BRK, ADT_OPEN_RD, ADT_FSTAT),
(ADT_BRK, ADT_OPEN_RD, ADT_FSTAT, ADT_MMAP),
(ADT_OPEN_RD, ADT_FSTAT, ADT_MMAP, ADT_CLOSE),
(ADT_FSTAT, ADT_MMAP, ADT_CLOSE, ADT_OPEN_RD),
(ADT_MMAP, ADT_CLOSE, ADT_OPEN_RD, ADT_MMAP),
(ADT_CLOSE, ADT_OPEN_RD, ADT_MMAP, ADT_MUNMAP)}

在 Forrest 的系统中由于是离线地对审计数据进行分析,所以采用对系统调用序列向前看进行匹配的方法[4,5],就是对采集到的系统调用序列按发生的先后顺序 k 个为一组存放在库中. 在 AUDIDS 实现中,由于要进行实时地审计分析,所以用向后看的审计事件向量代替了向前看,就是对采集到的审计事件按发生顺序的逆序 k 个为一组存放在库中,这样可以提高匹配时的效率,并且简化了匹配的实现. 当一个审计事件发生多次,它后面可以对应不同的序列. 它们压缩的表示如下:

表 2　向后看匹配的正常库

	现在的位置	位置一	位置二	位置三
	ADT_EXEC			
	ADT_BRK	ADT_EXEC		
R_0	ADT_OPEN_RD	ADT_BRK	ADT_EXEC	ADT_FSTAT
R_1		ADT_CLOSE	ADT_MMAP	
R_2	ADT_FSTAT	ADT_OPEN_RD	ADT_BRK	ADT_EXEC
R_4	ADT_MMAP	ADT_FSTAT	ADT_OPEN_RD	ADT_BRK
R_5		ADT_OPEN_RD	ADT_CLOSE	ADT_MMAP
R_6	ADT_CLOSE	ADT_MMAP	ADT_FSTAT	ADT_OPEN_RD
R_7	ADT_MUNMAP	ADT_MMAP	ADT_OPEN_RD	ADT_CLOSE

相应地,向后看的审计向量数学表示如下:

f(4; ADT_EXEC, ADT_BRK, ADT_OPEN_RD, ADT_FSTAT, ADT_MMAP, ADT_CLOSE, ADT_OPEN_RD, ADT_MMAP, ADT_MUNMAP)

= {(ADT_FSTAT, ADT_OPEN_RD, ADT_BRK, ADT_EXEC),
(ADT_MMAP, ADT_FSTAT, ADT_OPEN_RD, ADT_BRK),
(ADT_CLOSE, ADT_MMAP, ADT_FSTAT, ADT_OPEN_RD),
(ADT_OPEN_RD, ADT_CLOSE, ADT_MMAP, ADT_FSTAT),
(ADT_MMAP, ADT_OPEN_RD, ADT_CLOSE, ADT_MMAP),
(ADT_MUNMAP, ADT_MMAP, ADT_OPEN_RD, ADT_CLOSE)}

对于审计事件可以用 8 个 unsigned long 表示(共 256bit,前 195 位每一位表示一个审计事件,因为在实时监测时进行位匹配效率较高),这样一个最大的正常库可用一个(195 * 8 * 4) * k 个字节的矩阵表示.

4.4 检测入侵行为

有了正常库后,就可以对系统进行实时报警了. 我们用同样的方法检查系统中新产生的审计事件序列,在新的审计事件序列上滑动大小为 k 的窗口,检查此审计事件序列是否有别于正常库中的正常审计向量,任何审计事件序列(现在监测到的审计事件和窗口内以前的审计事件)的审计向量不出现在正常库中,则称之为不匹配. 任何不匹配的审计事件序列有可能是入侵,也有可能是未列入正常库中的正常审计向量,对于后者在产生多次误报后,可把此正常审计向量加入正常库中,以后就不会产生误报了.

向后看的匹配规则如下:

(1) 将滑动窗口内新出现的审计事件与正常库中审计向量的第一个审计事件比较,如果找不到,则此窗口内的审计向量产生异常.

(2) 如果找到,则将滑动窗口内的已匹配的审计事件的前一个审计事件与该正常库的审计向量的下一个审计事件比较. 如果不匹配,则此窗口内的审计向量产生异常;如果匹配,则执行第 2 步直到该窗口内的审计向量匹配完.

(3) 匹配完该滑动窗口内的审计向量,则将滑动窗口在审计事件序列上向前滑动一个审计事件,则执行(1),直到该审计事件序列匹配完.

例如,我们用上面的正常库,检测如下新的审计事件序列:

ADT_ EXEC, ADT_ BRK, ADT_ OPEN_ RD, ADT_ OPEN_ RD, ADT_ MMAP, ADT_ CLOSE, ADT_ OPEN_ RD, ADT_ MMAP, ADT_ MUNMAP

如果用向前看的方法,则将在正常库中的第一条审计向量 $\boldsymbol{R}_0$ 的第四位发现不匹配.

如果用向后看的方法,则将在正常库中的第一条审计向量 $\boldsymbol{R}_0$ 的第二位发现不匹配. 用向后看的方法在实现上有如下优点:

检测到新的审计事件,只要把此审计事件先和正常库中审计向量的头一个审计事件比较,如果找到则将审计事件序列的倒数第一个与该向量的第二位匹配,如果匹配则再比较后面的一个,如果以上有一处不匹配则产生异常. 向后看的匹配方法效率较高,实现起来也比较容易. 而向前看的方法需要将新审计事件和前 $k-1$ 个审计事件整体地同正常库中的向量比较,实现起来比较繁琐,从而影响了效率.

5 实验结果

在 IBM PC 上基于 redhat7.0 实现了本文提出的 AUDIDS 模型. 实验首先测试了当 AUDIDS 运行时对整个系统的影响,通过对 *Ls*、*Ps* 和 *Date* 命令进行了测试,证明整个系统的效率下降不超过 10%,实验结果如表 3,单位为微秒.

表 3 系统效率测试结果

	Ls	*Ps*	*Date*
AUDIDS 未运行	25	208	14
AUDIDS 运行时	27	230	15

为了测试AUDIDS 对入侵的检测,测试了基于 linux 能力(capability) 漏洞的 sendmail 攻击的检测. linux 能力(capability) 漏洞允许一个非特权程序阻止一个特权程序降低它的能力,可以用 sendmail 利用这个漏洞使一般的用户获得系统的所有特权,这个漏洞的详细表述见www.securityfocus.com 的邮件列表[9]. 首先在 IBM PC 上,按 sendmail 的正常用法收集审计事件,以生成正常库. 通过模拟运行,取滑动窗口的大小为 6,收集了 267735 个审计事件,在正常库中产生了 1672 个审计向量. 在测试入侵的时候我们在正常使用 sendmail 的同时进行了 10 次 sendmail 攻击. 结果比较明显: AUDIDS 报警 12 次,其中对 10 次 sendmail 攻击都检测到了,但是有两次误报,总体误报率为 16.6%.

又通过收集不同大小的正常库,看对误报率的影响. 我们共对五个不同大小的正常库进行了测试,对误报率的影响见图 2.

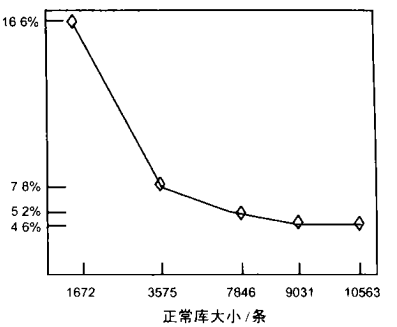

图 2 正常库对误报率的影响

从图中可以看出随着正常库的增大,系统对入侵的误报率有所降低,但是当正常库增大到一定程度的时候,在本系统中为 9031 以后,正常库的增大对误报率的降低作用不大了.

6 结束语

本文提出了一种基于审计事件向量的实时入侵检测模型. 在该方法中,正常库是由一些通过运行特权程序得到的审计事件串组成,由于审计事件比无参数的系统调用具有更多的语义,所以该方法对入侵的描述更为准确. 此外,通过改进审计事件序列与正常库的匹配顺序,从而降低了该方法的复杂性,提高了效率. 该方法对入侵检测的成功率很高,但是在学习阶段如果没有收集到足够的正常数据的话,则实时报警是误报率较高,所以以后应该对如何学习系统的正常状态进行进一步研究.

参考文献:

[1] Barton P Miller, David Koski, Cjin Pheow Lee, etc. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services [R]. Technical report, CS-TR-95-1268, Computer Sciences Department, University of Wisconsin, 1995.

[2] W Lee, S J Stolfo. Data mining approaches for intrusion detection [A]. In Proceedings of the 7th USENIX Security Symposium [C]. San Antonio, Texas, USA: 1998.

[3] John P Wack, Lisa J Carnahan. Keeping your site comfortably secure: An Introduction to Internet Firewalls [DB/OL]. NIST Special Publication 800-10, U. S. DEPARTMENT OF COMMERCE. <URL: http://csrc.nist.gov/publications/nistpubs/800-10/>.

[4] S Forrest, S A Hofmeyr, A Somayaji, T A Longstaff. A sense of self for UNIX processes [A]. In Proceedings of the 1996 IEEE Symposium on Security and Privacy [C]. Los Alamitos, CA: 1996. 120- 128.

[5] S A Hofmeyr, S Forrest, A Somayaji. Intrusion detection using sequences of system calls [J]. Journal of Computer Security, 1998, 6: 151 - 180.

[6] W Lee, S J Stolfo, P K Chan. Learning patterns from UNIX process execution traces for intrusion detection [A]. AAAI Workshop on AI Approaches to Fraud Detection and Risk Management [C]. AAAI Press, 1997. 7: 50- 56.

[7] G G Helmer, J S K Wong, V Honavar, L Miller. Intelligent agents for intrusion detection [A]. In Proceedings IEEE Information Technology Conference [C]. Syracuse, NY: 1998. 121- 124.

[8] C Warrender, S Forrest, B Pearlmutter. Detecting intrusions using system calls: alternative data models [A]. In Proceedings of the 1999 IEEE Symposium on Security and Privacy [C]. Los Alamitos, CA: IEEE Computer Society, 1999. 133- 145.

[9] Wojciech Purczynski. Sendmail & procmail local root exloits on Linux kernel up to 2. 2. 16pre5 [DB/OL]. BUGTRAQ Mailing list (bugtraq @ securityfocus. com), 2000- 06- 09.

作者简介:

刘海峰　男, 1975 年生于山东泰安, 中国科学院信息安全技术工程研究中心博士生, 主要研究方向为信息系统安全理论与技术.

卿斯汉　男, 1939 年生于湖南隆日, 中国科学院信息安全技术工程研究中心主任、研究员、博士导师, 主要研究方向为信息系统安全理论与技术.

蒙　杨　男, 1972 年生于甘肃天水, 博士, 在中国科学院信息安全技术工程研究中心工作, 主要研究方向防火墙.

刘文清　男, 1967 年生于河南虞城, 中国科学院信息安全技术工程研究中心博士生, 主要研究方向为安全操作系统.