基于系统调用子集的入侵检测

张相锋, 孙玉芳, 赵庆松
(中科院软件所, 北京 100080)

摘 要: 入侵检测技术是入侵检测系统(IDS)的重要内容. 根据系统调用的作用效果对系统调用进行划分, 在此基础上提出基于系统调用的一个子集(W 子集)的入侵检测技术. 实验表明, 与基于系统调用全集的方法相比, 基于 W 子集的入侵检测技术具有较低的误报率, 且所需存储空间代价和计算代价都较小, 因而更加适合于实时入侵检测.

关键词: 信息安全; 入侵检测; 异常检测; 入侵检测技术; 系统调用
中图分类号: TP309 文献标识码: A 文章编号: 0372-2112 (2004) 08-1338-04

Intrusion Detection Based on Sub-Set of System Calls

ZHANG Xiang-feng, SUN Yu-fang, ZHAO Qing-song
(Institute of Software, Chinese Academy of Sciences, Beijing 100080, China)

Abstract: Intrusion detection (ID) techniques are important to intrusion detection systems (IDS). Based on the ID technique using system call sequences, a new detection technique was put forward that uses a subset of system calls (named W-subset). Experiments show that the new technique has low false positive rate, low storage-cost, high computing efficiency and hence is applicable to real-time intrusion detection.

Key words: information security; intrusion detection; anomaly detection; intrusion detection technique; system call

1 引言

没有缺陷的软件只是一个梦想[1], 在无法避免软件缺陷的情况下, 入侵检测系统(IDS)就成为提高系统安全性的重要工具之一. Dorothy Denning 在 1987 年最早提出 IDS 的通用结构模型[2], 1997 年 Teresa Lunt 等人在 IDS 已经得到充分发展的基础上又提出 IDS 的公共入侵检测框架(CIDF)[3], 标志着 IDS 在结构上已经很成熟. 但是 IDS 在实际应用中仍存在很多问题, 其中之一就是入侵检测技术不够完善. 误报率和漏报率是评价入侵检测技术的重要依据, 这两个值越低, 表明入侵检测技术的检测能力也就越强. 虽然入侵检测技术有很多种, 但都不外乎误用检测和异常检测这两类. 一般地, 误用检测技术具有较高的漏报率, 而异常检测技术具有较高的误报率. 由于二者各具优缺点, 实际的入侵检测系统往往把它们结合起来使用. 此外, 入侵检测应该是实时的, 以便能够及时发现入侵行为, 尽早采取应对措施, 这要求入侵检测技术不但具有很强的检测能力, 还要有较高的执行效率.

2 相关工作

系统调用是所有应用程序接受操作系统服务的公共途径, 是构建程序功能的基本元素, 通过对系统调用的监视可以追踪进程的行为. 基于这种思想, Stephanie Forrest 提出了一种异常检测技术[4~6], 它通过基于定长系统调用子序列构造的特征库可用来区分 UNIX 程序的正常行为和异常行为. 有很多其他的研究项目也借鉴了 Forrest 的入侵检测技术的思想, 如文献[7]提出通过监视中间件的使用来检测进程的异常行为, 而文献[8]提出了基于库函数调用序列的入侵检测技术.

Forrest 提出的入侵检测技术仍存在一定的限制, 即, 它使用了所有的系统调用, 因而对计算资源和存储资源的要求较高, 不利于实时地进行入侵检测. 文献[9]对该方法进行改进, 提出了基于变长系统调用子序列进行入侵检测的方法, 能够明显降低特征库的容量, 但这使得特征库的组织变得复杂, 而且相关算法的计算代价很高. 文献[10]通过基因规划手段来构造特征库, 但需要覆盖应用程序所有执行路径, 因而所需的计算量也很大. 减少所监视的系统调用, 从而缩短进程生成的系统调用序列, 可能会降低入侵检测的存储代价和计算代价, 但前提是受监视的系统调用能够反映应用程序的行为特征. 通过对系统调用集合的划分, 我们推断: 只需监视系统调用的一个具有/写0性质的子集, 就可以进行有效的入侵检测, 同时又能减少需要分析的数据量, 提高入侵检测的性能.

3 系统调用集合的划分

考察两个最常用的系统调用, read 和 write. 和文献[6]一样, 我们仍不考虑系统调用的参数. 系统调用 read 只会影响进程自身, 而对进程以外的其他实体不会有任何影响(除了消耗一些 CPU 时间外), 而系统调用 write 则不同, 它可以改变文

收稿日期: 2003-07-01; 修回日期: 2004-04-20
基金项目: 国家 863 高技术研究发展计划项目(No. 2002AA141080); 国家自然科学基金(No. 60073022); 中国科学院知识创新工程项目(No. KGCX1-09); 国家自然科学基金(No. 60373054)

件系统的数据或将数据通过网络传送到系统以外, 从而会影响到其他进程甚至远程系统的运行. 可见, 两个系统调用的性质是不同的. 与 read/ write 类似的系统调用还有很多. 我们可以把系统调用分为两类, 一类就象 read, 称为具有/ 读0性质的系统调用, 它们的执行不会对进程以外的实体造成直接的影响, 只会影响到进程本身的运行; 其他的则类似于 write, 称为具有/ 写0性质的系统调用, 它们的执行可能直接影响到进程的运行或各种资源的状态. 标准的 Linux 2.4.18 内核有237个系统调用, 其中具有/ 写0性质的系统调用有 105 个(其中某些系统调用只有在接受特定参数时才具有/ 写0作用, 如 open 接受O_CREAT 参数时. 因为我们不关心系统调用的参数, 这类系统调用也称为具有/ 写0性质). 我们将利用这个具有/ 写0性质的系统调用子集进行入侵检测.

为了方便, 我们把系统调用的全集记为 F 集, 把基于 F 集的系统调用序列称为F 序列, 基于 F 序列的入侵检测方法简称为F 检测; 而把具有/ 写0性质的系统调用子集称为 W 子集, 把基于 W 子集的系统调用序列称为W 序列, 基于 W序列的入侵检测方法称为 W检测. 从 F(W)序列中抽取的长度为 k 的子序列称为F(W) 子序列, 特征库中的F(W) 子序列称为F(W)特征序列, 在入侵检测时进程发出的 F(W)子序列称为F(W)检测序列. 由于 W 子集的基数小于全集F 的基数, W特征序列的总数总要小于F 特征序列的总数, 在 Linux 2.4.18中, 二者之比在特征序列长度 k= 10(文献[6] 推荐的特征序列长度)时约为 3432.34! 可见, 与基于 F 集的特征库相比, 基于 W 子集的特征库可能会小很多, 因而 W检测的效率也就可能高很多.

4　实验环境与流程

基于系统调用序列进行入侵检测主要有三个步骤: 系统调用收集、特征抽取(即学习或训练) 和异常检测. 为便于比较, 我们仍采用文献[6] 的方法来收集进程发出的系统调用、构造特征库、进行异常检测, 只不过在进行 W检测时增加了/ 系统调用过滤0这个环节. 实验系统的工作流程如图 1.

我们在 Linux 2.4.18 核心上进行实验. 首先为被监视程序设定/ 受监视0标志, 它执行时产生一个具有/ 受监视0标志的进程. 在核心的 ENTRY(system_call) 处截获/ 受监视0进程发出的所有系统调用, 将系统调用号和相关的进程号收集并保存到与受监视程序对应的磁盘文件中. 实验过程兼顾了 F 检测和W检测两种方法: 在核心中获取进程发出的所有系统调用, 即进程的 F 序列; 在进行有关 W 检测的实验时, 再根据具有/ 写0性质的系统调用的列表对 F 序列进行过滤, 以得到相应的 W 序列. 程序行为的特征抽取和入侵活动的检测这两个过程对于 F 检测与 W检测并没有什么区别, 因此实验中我们仍采用文献[6] 所使用的工具, 即 stide[11].

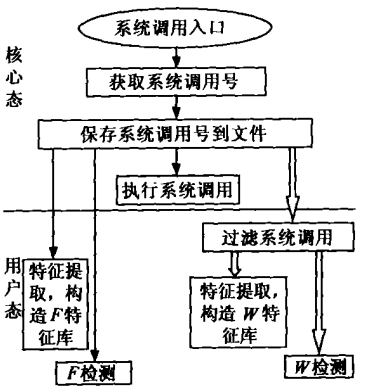

图1　实验系统的流程图

5　模式抽取

构建应用程序特征库的过程称为模式抽取, 即首先获得程序正常运行时发出的系统调用序列, 然后通过滑动窗口找出其中所有长度为 k 且不重复的系统调用子序列, 用以构成应用程序的特征模式库(model base), 即特征库, 记为 MB, k称为特征序列长度, 特征库中特征序列的个数称为特征库的容量, 记为 N. 应用程序基于 F 集(W子集) 的特征库称为 F(W)特征库. 随着学习时间的延长, 特征库也会越来越丰富.

例如, 取滑动窗口长度 k= 3. 某进程顺序发出以下系统调用: open, create, read, write, read, write, read, write, read, write, close, close, exit

这实际上就是进程的 F 序列, 相应的 F 特征库则含有以下F 特征序列:

open, create, read　create, read, write　read, write, read
write, read, write　read, write, close　write, close, close
close, close, exit

此 F 特征库的容量为 7. 与该F 序列对应的W序列为:

open, create, write, write, write, close, close, exit

相应的 W特征库则含有以下 6 个 W特征序列:

open, create, write　create, write, write　write, write, write
write, write, close　write, close, close　close, close, exit

6　入侵检测指标

特征库可以作为参照来检测应用程序的异常行为. 一个检测序列如果出现在特征库中则称为一次/ 匹配0(match), 该检测序列是一个正常序列(或匹配序列); 否则称为一次/ 失配0(mismatch), 该检测序列就是一个异常序列(或失配序列). 程序运行过程中出现的失配序列越多, 就越能表明其行为有异常, 当失配数量超过某阈值时, 我们就可认为应用程序受到了入侵. 通常, 失配序列数量会随入侵活动、应用程序及其运行环境的不同而有较大差异, 因此该阈值难以准确确定. 类似地, 失配序列与进程产生的所有检测序列的数量比(失配比) 也可以用来判定程序行为是否异常, 但相应的阈值也难以确定. 其他的检测指标同样存在阈值确定问题.

另一类异常检测指标是Forrest 采用的系统调用序列与特征库之间的/ 距离0. 共有四种/ 距离0, 这里分别称为 d1距离、d2距离、d3 距离和 d4 距离. 两个等长系统调用序列 X 和 Y之间的差异称为二者的d1 距离, 简记为 d1(X, Y), 其定义为:

$$d1(X, Y)=\sum_{i=1}^{k}\{f(Xi, Yi)\}$$

其中, X_i, Y_i 分别表示序列X 和Y 中的第i 个元素, k 为X和 Y 的长度, 函数 f 的定义为: 如果 x= y, 则 f(x, y) = 1; 否则 f(x, y) = 0.

对于一个检测序列 s, 它与特征库 MB 之间的距离就是它与MB 中所有特征序列之间差异的最小值, 称为 d2 距离,

其定义为: $d2(s, MB) = \min_{j \in MB}\{d(s, j)\}$

在检测阶段,应用程序的运行会产生一个很长的系统调用序列(LS),由它可以构造出多个长为 k 的检测序列.应用程序的行为与其特征库之间的差异,即这个长序列 LS 与特征库 MB(其序列长度为 k)之间的距离,称为 d3 距离,其定义如下: $d3(LS, MB) = \max_{s \in Q_k(LS)}\{d2(s, MB)\}$

其中 $Q_k(LS)$ 表示 LS 中所有长度为 k 的子序列构成的集合.d3 距离与特征序列长度 k 有关,而 k 是根据经验选择的,如果要消除特征序列长度 k 对 d3 距离的影响,可以将 d3 距离规格化,这样就得到 d4 距离: $d4 = d3/k$

d4 距离消除了特征序列长度带来的影响,可以作为一个较为客观的检测指标.因此,我们在实验中将主要采用 d4 距离进行入侵检测.以上所有检测指标都需要一个阈值作为参照才能进行入侵检测,由于很难找出准确的阈值,因此进行检测时会产生误报和漏报这两种错误.误报率和漏报率的高低可以说明入侵检测技术的检测能力.

7 测试结果

我们在同样的软硬件环境下同时测试 F 检测和 W 检测这两种方法,从而判断 W 检测是否比 F 检测更实用.

7.1 特征库的建立

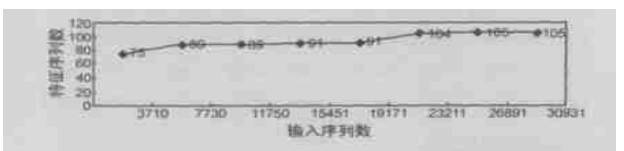

图 2 进程 ps 基于 W 子集的特征序列数量变化趋势图

图 2 显示了使用程序 ps 测试的 W 特征库的容量与输入序列(即在学习时收集到的 W 序列)数量之间的关系.可以看出,随着输入序列数量的增长,所得的 W 特征序列数量有所增长,但增幅不大,可见 W 特征库能够保持相对稳定,可以用来表征应用程序的行为特征.这符合 Forrest 的实验结果.在实际构造特征库时,若特征库增长的速度小于某给定阈值,就可认为已经收集了足够多的特征序列[12].

7.2 d4 距离与特征序列长度的关系

若以 d4 距离为主要检测指标,还要确定它是否能够稳定地反映异常行为与正常行为的差异程度.为此,我们在 login 程序中加入窃取用户口令的代码,构造出它的木马程序,然后通过它考察 d4 距离与特征序列长度的关系.由图 3 可以看出,特征序列长度在[6, 30]之间时,W 检测与 F 检测的表现很类似:随着特征序列长度的增长,d4 距离起伏不定,但变化幅度不大,且总的趋势是增长的.特征序列在长度超过 16 后对 d4 距离已经没有显著的影响,说明 d4 距离还是比较稳定的.

7.3 对典型入侵的检测

缓冲区溢出、木马、拒绝服务是类 Unix 系统上最常见的三种攻击方式,我们主要针对这三类入侵进行检测实验,具体是:针对 wu2.ftpd2.6.1218 的缓冲区溢出缺陷[13]进行的缓冲区溢出攻击、针对窃取用户口令的 login 木马攻击、内存过度消耗对 vi 的拒绝服务攻击[8].各实验所得的 F 检测的 d4 距离都大于 0.58,而 W 检测的 d4 距离也都大于 0.4,这说明 W 检测与 F 检测一样,可以作为一种有效的入侵检测方法.测试结果如下各表.

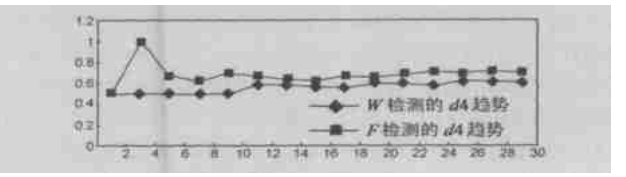

图 3 规格化距离 d4 与特征序列长度的关系

7.4 W 检测的误报率

降低误报率比降低漏报率显得更有意义[14].我们采用针对 sendmail2.8.11.2* 的缓冲区溢出攻击[15]来考查 W 检测的误报率,所使用的 d4 距离阈值为 0.5.实验中所有的攻击性测试均被发现.由图 4 可见,误报率随着特征序列长度的增加呈下降趋势,在特征序列的长度 k 超过 24 后,误报率的下降趋势变得很不明显.

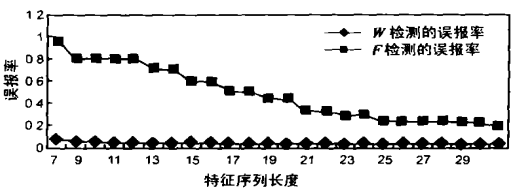

图 4 误报率与特征序列长度的关系

7.5 两种检测方法的比较

我们从检测效果和所花费的代价两个方面对 F 检测和 W 检测两种方法进行对比.与 F 序列相比,W 序列丢失很多关于应用程序运行的细节信息,所以 W 检测得到的 d3 距离、d4 距离要小于 F 检测的结果.但是所有实验中 W 检测所得的 d4 距离都高于 0.4,表明它可以用来进行入侵检测.在特征序列长度相同时,应用程序的 W 特征库较其 F 特征库要小很多(实验中前者要小于后者的一半),W 检测所需时间比 F 检测也少得多(实验数据中一般低于 1:5),而且误报率也更低.这表明与 F 检测相比,W 检测在保持较强检测能力的同时,

表 1 检测针对 wu2.ftpd 的缓冲区溢出攻击

缓冲区溢出攻击检测	特征序列长度	特征库容量	异常序列数	异常序列数/检测序列数* 100%	d3 距离	d4 距离	检测所需时间(秒)	特征库大小(byte)
F 检测	10	1144	1284	8.00349%	6	0.6	1.766	24639
F 检测	12	1288	1396	8.73209%	7	0.5833	2.766	33086
F 检测	14	1409	1508	9.46582%	8	0.6667	3.826	42154
F 检测	16	1511	1620	10.20470%	10	0.625	4.969	51745
W 检测	10	584	381	5.75964%	5	0.5	0.363	10720
W 检测	12	662	441	6.72359%	5	0.41667	0.544	14509
W 检测	14	725	502	7.71951%	6	0.42857	0.715	18515
W 检测	16	776	562	8.71723%	7	0.4375	0.903	22701

大大降低了对计算资源和存储资源的要求, 因而更适合于实时入侵检测.

8 结论和将来的工作

我们在对系统调用进行划分的基础上提出了基于系统调用子集进行入侵检测的技术, 并初步通过实验证明基于系统调用的 W 子集进行的 W 检测具有较好的检测能力和较低的存储代价和计算代价. 我们仍需在真实的应用环境中进一步验证这一结论, 同时还要研究实际实施该技术的方法和系统结构等问题.

W 子集的划分只是系统调用划分的一种. 我们还要对系统调用的分类做进一步的研究, 以确定一个更为合适的子集, 从而构造检测能力更强、运行效率更高的入侵检测系统.

表 2　对 login 木马攻击的检测

木马攻击检测	特征序列长度	特征库容量	异常序列数	异常序列数/检测序列数* 100%	d3 距离	d4 距离	检测所需时间(秒)	特征库大小(byte)
F 检测	10	671	162	4.31540%	7	0.7	0.210	15343
F 检测	12	733	198	5.28000%	8	0.667	0.300	20085
F 检测	14	781	228	6.08649%	9	0.643	0.383	25009
F 检测	16	825	256	6.84126%	10	0.625	0.510	30094
W 检测	10	307	66	2.53456%	5	0.5	0.107	6204
W 检测	12	338	74	2.84615%	7	0.5833	0.116	8160
W 检测	14	356	82	3.15871%	8	0.5714	0.126	10148
W 检测	16	372	90	3.47222%	9	0.5625	0.143	12187

表 3　检测针对 vi 的拒绝服务型攻击

拒绝服务攻击检测	特征序列长度	特征库容量	异常序列数	异常序列数/检测序列数* 100%	d3 距离	d4 距离	检测所需时间(秒)	特征库大小(byte)
F 检测	10	2258	24062	54.9700%	9	0.9	52.017	42362
F 检测	12	2733	25096	57.3426%	10	0.833	77.051	60759
F 检测	14	3191	26072	59.5836%	11	0.7857	107.883	82078
F 检测	16	3651	27107	61.9603%	12	0.75	144.573	106349
W 检测	10	847	284	2.13984%	7	0.7	0.507	13976
W 检测	12	979	357	2.69150%	9	0.75	0.622	19347
W 检测	14	1098	435	3.28153%	9	0.64286	0.960	25249
W 检测	16	1209	510	3.84964%	11	0.6875	1.422	31633

参考文献:

[1] Barton P Miller, et al. A Re2Examination of the Reli2ability of UNIX Utilities and Services[R]. Depart2ment of Computer science, university of Wisconsin, 1995.

[2] Dorothy E Denning. An intrusion2detection model[J]. IEEE Transactions on Software Engineering, 1987, 13(2): 222- 232.

[3] Kahn C, P porras, S Stanford2Chen, B Tung. A Com2mon Intrusion Detection Framework[Z]. Submitted to Journal of Com2puter Security, 2000.

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

[5] S Forrest, S Hofmeyr, A Somayaji. Computer immunology[J]. Commu2nications of the ACM, 1997, 40(10): 88- 96.

[6] Steven A Hofmeyr, Stephanie Forrest, Anil Somayaji. Intrusion detection using sequences of system calls[J]. Journal of Computer Security (6), 1998, 3: 151- 180.

[7] Matthew Stillerman, Carla Marceau, Mareen Stillman. Intrusion detec2tion for distributed applications[J]. Communications of the ACM, 1999, 42(7): 62- 69.

[8] Anita Jones, Yu Lin. Application intrusion detection using language li2brary calls[A]. 17th Annual Computer Security Applications Confer2ence[C]. New Orleans, Louisiana, USA: IEEE CS Press, December 10 - 14, 2001.

[9] Andreas Wespi, Marc Dacier, Herv Debar. Intrusion Detection Using Variable2Length Audit Trail Patterns[A]. Proceedings of the 3rd Sym2posium on Recent Advances in Intrusion Detection (RAID 2000)[C]. Toulouse, France: Springer LNCS 1907, RAID, 2000. 110- 129.

[10] 苏璞睿. 基于基因规划的主机异常入侵检测模型[J]. 软件学报, 2003, 14(6): 1120- 1126.

[11] Julie Rehmeyer. DRAFT: User documentation for the STIDE software package[CP/OL]. Available at www. cs. unm. edu/~ immsec/soft2ware/stide_ user_ doc. ps, 1998- 06.

[12] Christina Warrender, Stephanie Forrest, Barak Pearlmutter. Detecting intrusion using system calls: Alternative data models[J]. Proceedings of the 1999 IEEE Symposium on Security and Privacy, 1999: 133- 145.

[13] Wu2FTPd Remote Heap Overflow Exploit (In Java)[CP/OL]. Avail2able at http://www. securiteam. com/exploits/5KP0S2A7FY. html, 2002- 06- 28.

[14] Stefan Axelsson. The base2rate fallacy and its implications for the diff2culty of intrusion detection[A]. Proceedings of the 6th ACM Confer2ence on Computer and Communications Security[C]. Kent Ridge Digi2tal Labs, Singapore: ACM Press, November 1- 4, 1999. 1- 7.

[15] Alexander Yurchenko. Another sendmail exploit[CP/OL]. Available at http://cert. uni2stuttgart. de/archive/bugtraq/2001/08/msg00336. html, 2001- 08.

作者简介:

张相锋　男, 1971 年 4 月生于山东济南, 中国科学院软件研究所博士研究生, 主要研究方向为操作系统和信息系统安全. Email: mailofzxf@sina. com.