文章编号:1001 - 9081(2006)09 - 2137 - 03

基于系统调用的入侵检测系统设计与实现

张 军[1],苏璞睿[2],冯登国[2]
(1. 中国科学技术大学 电子工程与信息科学系,安徽 合肥 230027;
2. 中国科学院 软件研究所,北京 100049)
(xiaonanyue@hotmail.com)

摘 要:介绍了一个基于系统调用的灵活加载的入侵检测系统。该系统改进了常用的数据采集方法,采用虚拟设备驱动来获取系统调用。这种数据采集方法对系统影响小,可以灵活装卸,并提供标准的接口。数据分析融合了异常检测和误用检测两种方法,提出了相应的检测模型,并引入了滤噪函数。

关键词:入侵检测; 系统调用; 虚拟设备驱动; 滤噪函数; 非层次聚类
中图分类号: TP309.5 **文献标识码**:A

Design and implementation of intrusion detection system based on system-call

ZHANG Jun[1], SU Pu-rui[2], FENG Deng-guo[2]
(1. *Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei Anhui* 230027, *China*;
2. *Institute of Software, Chinese Academy of Sciences, Beijing* 100049, *China*)

Abstract: The technology of Intrusion Detection is one of the important measures to protect the networks. Host-based intrusion detection is used to protect the key hosts. A flexible loading intrusion detection based on system-call was introduced in this paper. This system improved the common data collection method, and adopted virtual equipment driversto acquire system call. This method brings small influence on system, is easy to load and unload, and provides the standard interface. The data analysis integrates the two detection methods: anomaly and misuse, which provides corresponding detection models and introduces the noise filtering function.

Key words: intrusion detection; system call; virtual equipment drives; noise filtering function; nonhierarchical clustering

0 引言

随着网络技术的发展,黑客攻击方法也越来越全面,仅仅依赖于防火墙等访问控制设备并不能完全抵御网络攻击,入侵检测技术就是有效的弥补措施之一。目前入侵检测系统的数据源也分为两大类:基于主机的数据源和基于网络的数据源。但随着网络技术的发展,网络入侵检测的发展也遇到了许多问题,例如网络带宽瓶颈,误报率高,检测准确度不高,无法在加密环境中使用等等。主机入侵检测在一定程度上可以与网络入侵检测系统形成互补。它具有检测准确性高,不受带宽影响等特点,现已成为新的研究热点。

1996 年,Forrest 提出利用进程的系统调用相关数据作为入侵检测的数据源,分析了系统调用对于进程行为的重要性,并提出了第一个基于系统调用的入侵检测模型。后来在其启发下,出现了多种更优秀,更成熟的检测模型,提出了各种异常检测和误用检测方法,引入了各种系统调用相关属性作为检测数据源[2,3,7,8]。本文在参考 CIDF(Common Intrusion Detection Framework)模型[9]的基础上,设计并实现了一个基于系统调用的灵活加载的入侵检测系统。本系统专门设计了虚拟设备驱动,用于采集系统调用数据; 融合了误用检测和异常检测两种技术,提出了相应的检测模型,并引入了滤噪函数。

1 系统总体结构

FLID(Flexible Loading Intrusion Detection using system-call)系统是一个基于系统调用的灵活加载的入侵检测系统,它容易扩展,安装卸载方便,具有标准接口。系统结构如图 1 所示。

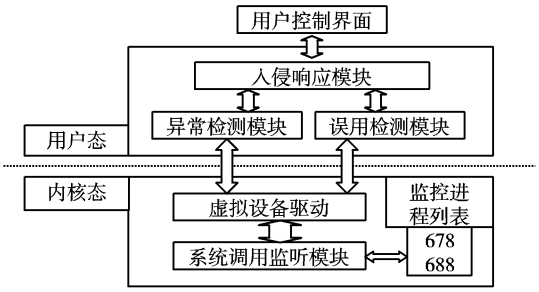


图 1 系统框架

整个系统分成 5 大部分,分别为:

(1) 数据采集模块负责搜集各个被监控进程触发系统调用序列(包括系统调用类型、系统调用参数、系统调用返回值等),该模块在操作系统内核空间运行,通过截获每一个系统调用截获各个进程执行的系统调用。该模块被设计成为一个虚拟设备,它将每一个被监控的应用程序视为一个设备,用户

收稿日期:2006 - 03 - 10; **修订日期**:2006 - 05 - 24
基金项目:国家 973 规划资助项目(G1999035802); 国家杰出青年基金项目(60025205)
作者简介: 张军(1982 -),男,安徽霍山人,硕士研究生,主要研究方向:入侵检测; 苏璞睿,男,博士,主要研究方向:入侵检测与评估; 冯登国,男,研究员,博士生导师,主要研究方向:信息安全.

可以通过 Linux 操作系统提供的标准设备操作接口对其进行操作控制,比如读、写、装载、卸载等等。

(2) 误用检测模块根据对已知攻击特征定义实施检测,首先对攻击有一个深刻、正确的认识。使用类似于 if - then 语法的规则输入攻击特征描述规则。攻击特征描述是响应动作的条件(即 if 响应条件),当满足这些条件时,完成相应的动作(即 then 响应动作)。

(3) 异常检测模块根据正常行为轮廓实施检测,超出正常行为轮廓范围的被认为是异常行为。通过对比分析实际行为和轮廓的差异来实施检测。

(4) 入侵响应模块根据检测到的攻击,采取相应的措施,目前有三种响应方法:终止进程运行,延迟系统调用执行和封禁用户。

(5) 用户控制界面主要用于用户管理整个系统,包括增加监控的应用程序,增加异常特征,调整系统参数,修改系统配置等。

2 数据采集

2.1 常用的系统调用截获方法的改进

搜集系统调用数据目前主要有三种方法:Strace 工具,修改内核,LKM 方法。

(1) Strace 工具是 Linux 等操作系统最初为用户提供的一个跟踪进程使用的系统调用和信号的工具,目前也开发出了 Windows 版本。其主要用于跟踪调试和分析应用程序,它可以提供进程触发系统调用和运行过程中信号处理的详细信息,包括系统调用的名称、参数、返回值、执行时间、触发地址以及信号的名称等等。利用 strace 收集数据简单易实现,它已经形成了标准的输出格式,不用再进行底层开发,但由于其主要为调试而设计,处理效率低,在进程监控过程中严重影响进程自身的运行效率,并且对于部分复杂程序的处理不稳定,易造成异常退出。这些问题使得它无法作为入侵检测系统实时检测的数据采集工具,但在实验期间需要采集数据离线分析时,是一个很好的辅助工具。

(2) 对于 Linux 操作系统,可以通过修改内核,增加采集系统调用数据的接口。采用该方法的最大优点是效率高,监控程序可以从内核直接获取数据,可以优化处理流程,增加数据采集效率。但采用该方法需要经过周密的设计,且对编码要求较高,对内核的修改特别是对系统调用部分的修改,直接影响到系统平台中运行的各个进程,因此,采用该方法可能会造成对某些应用程序的不兼容。

(3) 为了扩展内核的功能,Linux 提供了 LKM 机制,用户不需要重新编译内核的情况下,可以通过动态向内核载入模块来增加新的功能。模块直接通过 insmod 和 rmmod 增加和卸载。利用 LKM 机制可以向内核中装载新的模块,替换内核中系统调用表(System Call Table),直接从内核中获取进程执行的系统调用相关数据。但是 LKM 机制缺乏标准接口,无法有效的获取每个进程的系统调用。

通过分析上面三种系统调用截取方法的优点和缺点,我们提出了通过虚拟设备驱动来获取系统调用的方法,该方法主要具有以下特点:

(1) 虚拟设备驱动在核心态运行,数据搜集能力强,对系统运行影响小。设备驱动均在核心态运行,直接保证了搜集系统调用所有相关数据的能力,包括调用进程号、应用程序名称、参数、返回值等等;

(2) 具有标准的接口,容易扩展。和其他系统设备一样,它有标准的接口,包括 open( ),release( ),read( ),write( )等。各种应用可以如同其他设备一样对其操作,随时可以根据需要替换相应的数据分析部分,保证了系统良好的可扩展性和可移植性;

(3) 系统容易安装卸载,随时可以根据需要进行调整。与 LIDS 等系统相比,该方法不需要重新编译内核,并可根据需要随时安装卸载,操作简单,特别适合于作为基于系统调用的入侵检测研究的研究平台。

现在简单介绍一下该方法的原理和实现方法。

2.2 FLID 系统采集设备的设计与实现

(1) 主设备号和次设备号

主设备号用于标示设备类型,对于操作系统来说主要用于标识设备对应的驱动程序,主设备号的取值范围在0 ~ 256,但通常 0 和 255 保留。本设备的主设备号采用动态分配,即由操作系统动态分配一个空闲主设备号,通常情况下为 254。

次设备号只是由那些主设备号已经确定的驱动程序使用,内核的其他部分不会用到。它主要用于区分多个同类设备。本设备只可能有一个主设备号,但却可能有多个次设备号。本设备中 1 ~ 255 对应一个不同的被监控应用程序,0 号设备为控制设备,用于控制 1 ~ 255 的所有设备。0 号设备的具体控制功能将在 ioctl( )接口函数中介绍。

(2) 截获系统调用

为了获取所有系统调用相关数据,修改系统调用表,来截获所有系统调用。在本系统中也采用了该方法。其中用到了三个主要的数据结构,这三个数据主要用于记录被监控应用程序名称、进程 PID 及对应记录设备相关属性。

对于 Linux 中的任意一个系统调用,比如 sys_open (const char * filename,int mode),定义一个新的实现函数 my_open (const char * filename,int mode)替换系统调用表中的函数地址。新的实现函数只需检查进程状态和系统调用参数就可以确定对系统调用的处理方法。

但对于一些特殊系统调用,比如 sys_fork( ),sys_execve( ),sys_exit ( ) 等,还需要特殊处理。例如,新的实现函数 my_fock( )需要把新生成的进程 PID 加入到被监控进程列表中。新的实现函数 my_execve( )需要先检查文件名是否在应用程序列表中,如果是,将该进程加到被监控进程列表中。新的实现函数 my_exit( )需要先检查该进程是否在监控进程列表中,如果是,将该进程从列表中删除。

(3) 设备操作接口

该设备是一个标准的字符设备,实现了相应的接口函数,主要包括 open( ),release( ),read( ),write( ),ioctl( )等接口函数。其中 open( )、write( )、read( )、release( )与其他字符设备功能相当,执行打开或释放设备,发送或读取数据等功能。ioctl( )接口函数是专为管理、配置该虚拟设备而设计的控制函数。ioctl( )接口函数主要功能有:增加或删除被监控应用程序,查询被监控应用程序列表,暂停或启动设备运行。

当成功装载设备驱动程序后,设备默认状态为自动运行;当要完全终止设备运行时,应利用 rmmod 等命令卸载设备的驱动程序。

3　数据分析

数据分析是以数据采集模块预处理后的数据作为输入,分析进程行为是否出现异常。在本系统中融合异常检测和误用检测两种检测技术。

3.1　异常检测方法

采用了基于非层次聚类的无监督入侵检测模型(UNC)。聚类分析被广泛用于空间数据分析等应用,本模型引入了基于密度的非层次聚类分析算法生成行为轮廓,并对行为轮廓进行自动更新;通过分析被监控进程触发系统调用序列与行为轮廓的距离检测入侵,并在检测算法中引入了滤噪函数。

非层次聚类(Nonhierarchical Clustering)先给定一个粗糙的初始分类,然后按照某种原则进行修改,直至分类较为合理为止。行为轮廓(正常行为轮廓的简称)是对监控目标正常行为的描述,异常检测根据行为轮廓实施检测。在实施检测之前,我们需要根据训练数据生成行为轮廓。

在 UNC 模型中,我们引入了无监督的训练方法,无监督的训练方法基于以下假设:第一,正常行为的数量远大于入侵行为数量;第二,入侵行为和正常行为之间存在可区分的明显差异。它不需要对训练数据标识正常或异常,并且训练数据中允许少量的异常数据出现。采用无监督训练方法即可以直接利用系统实际运行的数据作为训练数据。

在统计分析过程中,有可能因为某种失误造成采集的实验数据不真实,这些失真的数据常离开数据群体而孤立出现,通常称为异常值(Outlier)。我们直接采用系统运行数据作为训练数据也存在异常值问题,在系统运行过程中可能发生入侵或用户误操作,相关数据均是异常值。

在非层次聚类中,为了进行分类,需要选择一批凝聚点(Condensation Point),让样本向凝聚点集中。所谓凝聚点就是一批有代表性的点,是形成类的中心。选择凝聚点常用均值法、密度法和 K-均值法等等。由于序列中的系统调用号代表的是系统调用类型,其均值没有任何意义,因此我们采用密度法选择凝聚点,生成行为轮廓。考虑到可能出现异常值,算法去除了部分密度过低的序列。

在异常检测过程中,通过分析实时序列和行为轮廓之间的差异来实施检测。首先,需要为每一个被监控的进程建立一个队列,并利用一个长度为 L(与正常行为轮廓中序列长度一致),步长为 1 的滑动窗口将实时数据截为定长片断 X_i。然后,通过计算 X_i 与行为轮廓 NP 的距离,分析监控进程是否存在异常。

系统调用片段 X 和行为轮廓 NP 中凝聚点的最短距离称之为 X 与 NP 的距离,记为 $P_{Dist}(X,NP)$,即:

$$P_{Dist}(X,NP) = \min\{Distance(X,C_j) \mid C_j \in NP\}。$$

异常检测基于以下假设:异常行为和正常行为存在可检测的差异。正常行为和异常行为在 $P_{Dist}(X_i,NP)$ 上的差异是否可检测直接决定了本模型检测算法的准确性。在各种实验中,由于设备、观测等偶然因素可能形成一些异常值,我们称之为噪声。在监控过程中,由于用户行为习惯的稍微变化或者训练数据的不完整也可能形成一些异常。但由攻击造成的异常则经常集中且连续出现,因此,在检测过程中,我们引入了滤噪函数 $F(X_i)$:

$$F(x_i) = \frac{1}{R}\sum_{j=i-R+1}^{i} P_{Dist}(X_i,NP)$$

其中,R 是滤噪序列长度,即取 R 个连续系统调用序列的 $P_{Dist}(X_i,NP)$ 进行滤噪处理。$R > 0$,是一个整数,其设置问题将在实验中进一步讨论。检测过程中,设置检测阈值 I,当 $F(X_i) > I$ 时,认为进程受到入侵。

3.2　误用检测方法

误用检测通过直接定义攻击过程特征来实施检测,在检测过程中,通过比较攻击过程特征与实时状态,确定系统是否受到攻击。

在基于进程行为的误用检测过程中,仍然采用针对每一个特定程序建立一个攻击行为特征库,针对每一个进程独立监控。

通过各种规则描述来定义攻击行为特征,每一条规则对应一种攻击方法,但各规则中关于系统调用节点(SysCallNode)特征定义则有可能出现重复。采用树形结构组织这些规则,有利于缩小存储空间,改善检测效率。

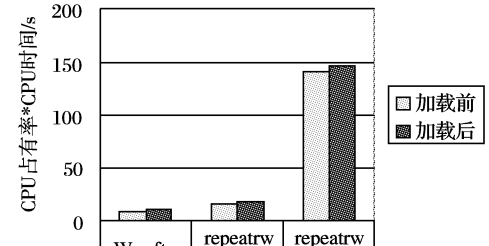
在规则树中,各个规则可能有不同数量的 SysCallNode 节点(至少要保证有一个),为了提高检测效率,还应该限定每一条规则中 SysCallNode 的最大数量。规则定义中 SysCallNode 允许的最大个数是 L,误用规则树的最大深度为 $L+2$,其中,另外还包含一个空的根节点和标示各种攻击的叶子节点。从根节点到任何一个叶子节点的路径即是一个攻击行为特征定义。

异常检测和误用检测的检测方法可以继续研究,以后可能会提出检测效率和检测能力更高的方法。本系统的数据采集模块采用了标准的接口,很容易和更好的检测方法结合。

4　实验数据

4.1　数据采集模块对系统影响

实验在 Redhat Linux 9 上进行,主要是对比加载数据采集模块前后,对系统的影响。这里通过 CPU 占有率和时间的乘积表示应用程序消耗的系统资源。实验使用了 2 个应用程序,一个为 Wu-ftpd,运行 10m;一个为自己编写的应用程序 repeatrw。Repeatrw 只是重复执行读写文件操作,可以设置读写次数,这里设置了 100 次和 1 000 次。实验结果如图 2 所示。



	Wu-ftp	repeatrw (100次)	repeatrw (1000次)
加载前	9.2	16.9	141
加载后	10.1	17.7	146.1

图 2　数据采集模块对系统影响

从图 2 中可以看出加载数据采集模块后对应用程序的影响很小,都小于 5%。

4.2　滤噪函数的效果

图 3 中虚线表示真实的系统调用序列的海明距离;实线表示通过滤噪函数处理后的海明距离。从图中可以发现经过滤噪函数处理后,可以有效地排除一些由于设备,观测等偶然因素引起的异常值。

(下转第 2144 页)

```
            #Step 4 依次改变路由表中路由条目的 metric 值,并将
            篡改后的路由条目发回路由器
            while { $i < $Num } {
              set RouteEntry [ RipngPreEntry $ GetRoute ( $ i )
              $RouteTag $GetLen( $i) $NewMetric]
              set EntryList [ RipngAddentry $EntryList $RouteEntry]
              set SendPck [ RipngSdu $Cmd $EntryList ]
              RIPng :: Send $SendSockRipng $SendPck
              incr i 1
            }
          }
        } else {
          # #其他情况下的处理方式,略
        }
      }
      #Step 5 接收路由器发来的路由更新报文,验证路由表是否已被
      篡改。同时检查是否收到来自路由器本应发往其他合法主机的
      报文。如果路由表没有改变,路由器没有受到影响,攻击失败。
      反之,攻击成功,将$result 设为 1,略
```

这个测试例的预期效果是攻击主机收到路由器发送回来的路由更新通告,其中包含了先前被篡改的路由信息。而且由于路由表被改动,攻击主机可截获路由器本应发往其他合法目的地主机的报文。

3.3 攻击测试结果

和上面的例子相似,可以实现并在实际的网络环境中运行其他攻击例,从而得到测试结果。利用开发的扩展命令和攻击测试系统,作者使用文章第一节列出的针对 RIP、RIPng、OSPF 和 BGP 协议的攻击方式对 Cisco 路由器进行了初步攻击测试。测试结果表明该系统能够有效的对上述几种路由协议的网络攻击进行模拟和测试,34 个攻击例中有 32 个的攻击测试例实现了攻击目的,没有成功的 2 个是关于 BGP 的序列号预测攻击的测试例,失败原因是被攻击的 Cisco 路由器未采用可预测序列号方案。

4 结语

文章在分析常见路由协议 RIP/RIPng、OSPF 和 BGP 协议特点的基础上介绍了一种新的网络攻击方法,这种方法利用协议本身的安全漏洞对网络进行攻击。文中着重介绍了作者设计的一个针对这种攻击方法的分布式网络攻击测试系统的开发及使用情况。试验表明,该系统能够有效的对基于路由协议的网络攻击进行模拟和测试,并具有一定的可扩展性。今后的工作方向主要集中在使用 Tcl 语言开发一个更完善的攻击测试系统,使其能对更多的网络层协议,如 ICMP、ARP 等协议,以及更多的下一代互联网中的路由协议,如 OSPFv3、BGP4 + 等协议进行模拟攻击测试,来更全面的检测出网络中可能存在的安全漏洞。

参考文献:

[1] 矫健,韩芳溪,毛忠东. 网络攻击手段及防御系统设计[J]. 计算机工程与应用, 2003,(33):168 - 170.

[2] HEDRICK C. Routing Information Protocol[S]. RFC1058, 1988.

[3] MALKIN G. RIP Version 2[S]. RFC2453, 1998.

[4] MALKIN G. RIPng for IPv6[S]. RFC 2080, 1997.

[5] MOY J. OSPF Version 2[S]. RFC 2328, 1997.

[6] REKHTER Y. Application of the Border Gateway Protocol in the Internet[S]. RFC 1268, 1991.

(上接第 2139 页)

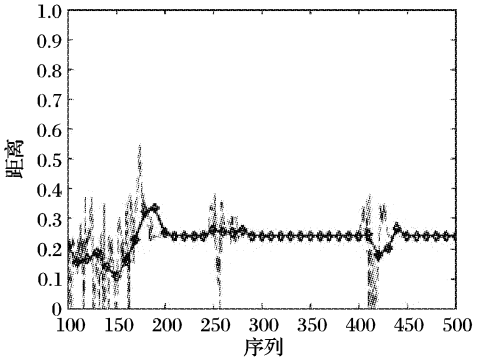


图 3 滤噪函数效果

5 结语

本文研究了基于主机的入侵检测,完成了一个基于系统调用的灵活加载的入侵检测系统(FLID)。该系统将数据采集模块设计为虚拟设备驱动,用户可以根据需要随时装载和卸载,和 LIDS 等系统相比,不需要编译内核,使用更灵活,更方便。数据分析模块设计中融合了异常检测和误用检测两种检测方式。异常检测中采用了非层次聚类的无监督入侵检测模型(UNC),并引入了滤噪函数;误用检测中提出了攻击树误用模型和攻击特征描述语言。

本系统数据采集模块使用标准的接口,可以继续改进异常检测算法,提高算法准确性,降低误报率和漏报率。研究关联分析技术,结合系统环境信息对异常检测结果和误用检测结果进一步关联分析,提高检测结果的准确性。

参考文献:

[1] PURUI SU. Research on Intrusion Detection Based on Privileged Process' Behaviors[D]. The Chinese Academy of Sciences, 2004.

[2] WARRENDER C, FORREST S, PEARLMUTTER B, Detecting intrusions using system calls: Alternative data models[A]. Proceedings IEEE Symposium on Security and Privacy[C], 1999. 133 - 145.

[3] JAIN K, SEKAR R. User-Level Infrastructure for System Call Interposition: A Platform for Intrusion Detection and Confinement[A]. Proceedings of the ISOC Symposium on Network and Distributed System Security[C], 2000.

[4] PROVOS N. Improving Host Security with System Call Policies[A]. Proceedings of 12th USENIX Security Symposium[C]. Washington DC, August 2003.

[5] CHARI S, CHENG PC. BlueBox: A Plolicy-Driven, Host-Based Intrusion Detection System[J]. ACM Transactions on Information and System Security, 2003, 6(2):173 - 200.

[6] SU PR, LI DQ, Feng DG, A Host-based Anomaly Intrusion Detection Model Based on Genetic Programming[J]. Journal of Software, 2003, 14(6).

[7] MARCEAU C. Characterizing the Behavior of a Program Using Multiple-Length N-grams[A]. Proceedings of the New Security Paradigms Workshop[C]. Cork, Ireland, 2000.

[8] LIAO YL, VEMURI VR. Use of K-Nearest Neighbor classifier for intrusion detection[J]. Computers & Security, 2002, 21(5):439 - 448.

[9] PORRAS P, SCHNACKENBERG D, STANIFORD-CHEN S, *et al*. The common intrusion detection framework architecture[EB/OL]. http://www.gidos.org/drafts/architecture.txt, 1998.