系统调用序列中关联规则的挖掘及其应用

Method and Usage of Mining Association Rules in System Call Serial

徐漫江（北京理工大学计算机科学工程系　100081）
顾　刚（江苏省电信实业集团公司　210006）

摘　要：本文首先介绍了利用程序运行时产生的系统调用序列进行入侵检测的原理，并提出一种挖掘系统调用序列中关联规则的方法，及利用挖掘出的规则进行检测的方法，最后用一个实例说明系统的运行过程。

关健词：入侵检测系统　关联规则　数据挖掘

1 前言

程序在正常状态下的运行流程具有一定的特征，利用这些特征可以检测针对程序漏洞进行的入侵。而这种入侵方法正是目前网络入侵中最常见、危害性最大的方法，如缓冲区溢出、IE浏览器的UNICODE攻击等。Forrest等人提出通过分析应用程序运行时系统调用进行检测，他们使用的主要方法是将系统调用序列分割为多个一定长度的“模式”，利用这些正常模式对比程序运行时动态生成的序列，从而得出程序是否正常运行。采用模式库检测的方法，具有两个明显的不足：由于其匹配算法过于严格，可能产生较高的误报；学习过程中产生的模式库规模与采用的模式长度有密切关系，当模式长度较大时，可能产生巨大的模式库，给检测工作带来不便，而模式长度的取值只是根据经验，缺乏科学依据。Wenke lee等针对以上不足，将数据挖掘算法引入对系统调用序列的分析。基于数据挖掘的入侵检测方法可以借用现有的KDD算法。如用于挖掘关联规则的Apriori 算法、用于挖掘序列模式的frequent episodes算法、用于分类的RIPPER 算法等。但是,这些算法在设计时未考虑任何专业知识。如果直接将这些算法用于入侵检测系统，效果往往并不理想。针对这种不足，本文介绍一种基于系统调用序列特征的关联规则挖掘算法，以及利用挖掘出的规则进行入侵检测的方法。

2 检测方法介绍

基于应用的入侵检测系统使用的基本思想是：当程序按正常流程运行时，不会产生错误的操作，这里的正常流程不光是指程序按设计时的流程进行运行，而且还包括程序在通常运行时可能并不包括流程的某些部分。因此可以通过监测应用程序的运行，及时发现异常的运行流程，从而发现攻击行为。

2.1 系统调用序列

刻画程序运行流程的方法有：基于程序源代码的静态分析[2]、基于标准库函数的系统调用记录[3]、基于系统调用序列等。通过分析可以发现，基于系统调用序列的分析与另外两种方法相比，具有明显的优势。

（1）无需源代码。由于很多软件是以二进制代码形式发布的，因此正常情况下很难合法得到其源代码，也就无法对其进行分析。

（2）与编程语言无关。不同的编程语言由于其特点不一样，因此很难采用统一的静态分析方法。

（3）分析对象数量适中。一般UNIX操作系统的系统调用数目在200左右，而库函数则有几千个甚至更多，而且不同的编程语言会使用不同的函数库。一方面对大量库函数进行修改工作量过大，另一方面所产生的模式库可能非常大，给分析工作带来不便。

（4）分析对象稳定。系统调用的版本间变化非常小，而库函数的版本间变化相对较大。

（5）入侵者无法绕过系统调用。系统调用是应用程序使用硬件设备的必经之路，这是由CPU硬件决定的，因此入侵者无法绕过系统调用进行攻击。而库函数则可以通过某些用户级操作进行替换，如重定义“PRELOAD”环境变量可以绕过C语言的库函数。

系统调用序列是指应用程序在执行过程中对系统调用的访问的顺序。由于一个系统调用的出现与其前面多个系统调用是相关的，如“读取文件”（read）之前一般会出

现“打开文件”（open）。因此通过分析多个系统调用之前的关系，可以分析出是否存在异常，如下例：

```
function foo() {
if(...) {
    f1();
    f2();
}
else {
    f3();
f4();
}
f5();
}
```

假设f1()至f5()分别系统调用了系统调用call1至call5，因此在此程序产生的系统调用序列中，会产生call1、call2、call5和call3、call4、call5两种序列。如果在实际运行时，检测到call1、call2、call3这样的序列，可以认为程序产生了异常。

2.2 数据预处理

在分析系统调用序列时可以发现，程序运行时可能会产生多个连续相同的系统调用。多数情况下，这些连续相同的系统调用是由程序中的循环语句产生的，如反复读取一个文件中多行内容会产生多个read系统调用。而且仅仅通过增加某一系统调用的数目，是无法达到攻击目的的。因此在分析时，可以忽略这个系统调用发生的次数，只将其视为一次系统调用。在某些情况下，这样的连续系统调用可能不是由循环产生的，但这样的处理过程并不会影响对异常的检测。

2.3 关联规则挖掘

设$I=\{I_1, I_2, \Lambda, I_m\}$是数据项集合。$D=\{T_1, T_2, \Lambda, T_n\}$是审计数据的集合，其中每条审计记录$T_i$是数据项的集合，$T_i \in I$。设A是I的一个子集，审计记录$T_i$包含A当且仅当$A \in T_i$。包含K个数据项的集合称为K-项集。关联规则的表现形式是“$A \rightarrow B, C=c, S=s$”，其中，A和B是同一条记录内的数据项集且AI B=Ø。支持度S是D中记录包含$AY B$的百分比；置信度C是一条记录包含A的情况下同时包含B的概率。挖掘关联规则就是对审计数据D，找出所有满足用户指定的最小支持度阈值S_{min}和最小置信度阈值C_{min}的关联规则。*Apriori*算法是一种最有影响的挖掘关联规则的算法，目前被广泛使用。

由程序的局部性特征可知，一个系统调用与其之前多个系统调用相关，但相关程度随着它们之间距离的增加而减小。因此在实际挖掘中，只对由C_i, $i<dist$产生的规则进行挖掘。具体方法是，首先用一固定大小（*dist*）的窗口在用于训练的系统调用序列上滑动，产生多条这一长度的样本数据。为了反应样本数据中系统调用的序列，将系统调用与位置映射到单一项上。如系统调用A位置为1，则对应的项为A1，因此同样的系统调用在不同的位置对应于不同的的项。然后使用Apriori算法对其进行挖掘，得到多条形如C_{K1}, C_{K2}, *A*, $C_{Kn} \rightarrow C_i$的规则，其中C_{Ki}被称为规则左项，C_i被称为规则右项。由于检测中需要的规则是表明当前系统调用前哪些系统调用与之相关，因此删除右项不为C_a的规则后，得出用于检测的规则集。

考虑到系统调用间的距离越大，则相关程度越小，因此不同的规则由于包含的系统调用距离不同，所具有的重要程度也不同，因此在挖掘出的规则中引入重要度。

设当前系统调用位置为0，其前方第i个系统调用的相关度为：

$$R_i=R(i)$$

相关度具有如下性质：

$$\begin{cases} R_i=0, i=0 \\ R_i=0, i>0 \end{cases}$$

$$\sum_{i>0} R_i=1$$

$$R_i<R_j>, \ \forall i>j$$

因此，规则集中的每条规则都具有重要度：

$$I=\sum(R_k \mid C_k \in \text{规则})$$

2.4 使用关联规则进行入侵检测

利用上节方法所挖掘出的规则对应用程序实际运行时动态生成的系统调用序列进行检测，可能出现三种结果：

1符合

当前系统调用序列与规则集中某条规则完全吻合。

2 不符合

当前系统调用序列与规则集中某条规则左项吻合，但不与右项吻合。

3 无规则

当前系统调用与规则集中任何一条规则的左项都不吻合。

针对这三种情况，所产生的异常值为：

$$Err=\begin{cases} Err, \text{ 完全符合} \\ Err+I \text{ 不符合} \\ Err+a \text{ 无规则} \end{cases}$$

其中I为所使用规则的重要度，a为某一固定值。

设系统调用序列中系统调用和个数为Num，当异常值Err/Num大于某一阈值时，认为出现入侵。

3 实际应用

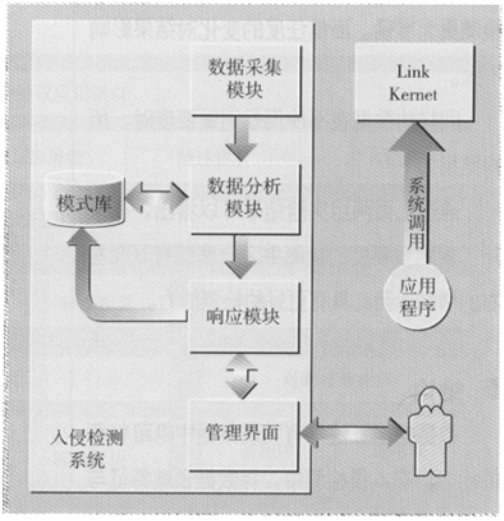

图 1 系统结构

基于上述检测方法设计的入侵检测系统，其原型系统运行于Intel平台的Linux操作系统，由数据采集模块、数据分析模块、模式库及响应模块组成。

每次当被检测程序调用系统调用时，系统转入内核运行。数据采集模块从内核中得到当前系统调用号，并将得到的数据传送至数据分析模块。数据分析模块用规则集中的规则检测系统调用序列，如果发现异常，立即通知响应模块。响应模块得到异常数据后，通过管理界面向管理人员通报异常情况，由管理人员对异常情况进行进一步处理。

4 实验结果

实验采用运行于Linux系统的wu-ftpd2.4[1]环境。

首先让系统在安全环境中运行，并对生成的系统调用序列进行挖掘。将这些规则分别对正常环境及入侵环境下的系统调用序列进行检测时，得出如下结果：

其中，为了反应在入侵环境及正常环境中异常值的差异程度，定义区分度：

D=入侵环境中的异常值/正常环境中的异常值

可以看出，正常环境与入侵环境下的异常值有明显的差异，可以准确地进行检测。支持度比较低时，所产生的规则数量很大，检测更为准确。而信任度的变化对结果影响不大。

同样的数据在不使用规则重要度时，所得结果如下：

通过上面两组实验结果可以看出，在采用了规则重要度的检测中区分度都有10%左右的增加，因此具有更好的检测能力。

5 结论

数据挖掘算法具有发现数据中隐藏特征的能力。在入侵检测中，将数据领域特征与数据挖掘获得的知识相结合，可以得到更有效的检测能力。

表1

支持度(%)	信任度(%)	规则数	正常环境中的异常值(%)	入侵环境中的异常值(%)	区分度
0.1	70	3804	1.4	46.5	33.2
0.5	70	846	5.4	60.2	11.1
1	70	233	8.5	66.9	7.9
0.1	80	3764	1.4	48.0	34.3
0.5	80	837	5.9	61.1	10.4
1	80	233	8.5	66.9	7.9
0.1	90	3714	1.6	56.5	35.3
0.5	90	812	6.6	63.6	9.6
1	90	231	8.6	68.4	8.0

表2

支持度(%)	信任度(%)	规则数	正常环境中的异常值(%)	入侵环境中的异常值(%)	区分度
0.1	70	3804	2.4	68.1	28.4
0.5	70	846	7.0	68.5	9.8
1	70	233	10.0	72.9	7.3
0.1	80	3764	2.3	67.8	29.5
0.5	80	837	7.4	69.1	9.3
1	80	233	10.0	72.9	7.3
0.1	90	3714	2.3	68.4	29.7
0.5	90	812	8.2	70.1	8.5
1	90	231	10.0	72.8	7.3

参考文献

1 Forrest S, Hofmeyr S, Somayaji A, et al. A sense of self for UNIX process[Z]. IEEE symposium on security and Privacy, Los Alamitos, USA, 1996.

2 Curry T. Profiling and tracing dynamic library usage via interposition[Z]. USENIX Summer 1994 Technical Conference, Boston, USA, 1994.

3 Wagner D, Dean D. Intrusion detection via static analysis[Z]. IEEE Symposium on Security and Privacy, Oakland, USA, 2001.