一种基于图的异常入侵检测新算法*)

朱莺嘤[1] 叶 茂[1] 刘乃琦[1] 吴 康[2] 郑凯元[1]

(电子科技大学计算机科学与工程学院 成都 610054)[1] (南京科瑞达电子装备有限公司 南京 211100)[2]

摘 要 根据主机系统异常入侵过程中 Windows native API 序列的瞬时高频性,提出一种基于图的异常入侵检测算法。该算法首先将每个 Native API 映射成为图中的一个点,并以其为起点的子序列作为该点的路径、Native API 的前后调用关系为边;其次,对图中每个点记录各自的路径,在这些点的路径中找到若干个圈,圈定义为因对同一个 Native API 重复调用而在图中出现的回路;然后,对组成圈的所有边权值根据一定规则更新;最后利用图的边与其邻边权值差计算出异常指数,判断该序列是否异常。实验结果表明该算法在 Windows 平台下能实时有效地检测出异常入侵和病毒的 Native API 序列。

关键词 异常入侵检测,Windows Native API,图

Novel Intrusion Detection Algorithm Based on Graph

ZHU Ying-ying[1] YE Mao[1] LIU Nai-qi[1] WU Kang[2] ZHENG Kai-yuan[1]

(School of Computer Science and Engineering, UESTC, Chengdu 610045, China)[1]

(Nanjiing Corad Electronic Equipment CO. LTD, Nanjing 211100, China)[2]

Abstract Because of the high instantaneous frequency of the intrusion windows native API sequence, a novel method based on graph theory was proposed to solve the problem of the host-based intrusion detection. In our algorithm, each native API is considered as a node in the graph and the transfer of two native API is the edge between these two nodes. The route for a node is defined as the subsequence beginning from itself. The repetition of the same native API in the graph is referred to as a loop. The route of each node is recorded so that the loop can be found. When the loops are found, the weights of all edges in the loops are updated accordingly. The abnormal degree of a native API sequence can be computed using the weight difference of the neighborhood edges in the graph. Experiments on the windows platform illustrate that our method can detect the unknown intrusion and virus native API sequence efficiently.

Keywords Intrusion detection, Windows native API, Graph

1 引言

随着计算机应用领域日益广泛,计算机的安全问题受到人们越来越多的关注。入侵检测技术作为保护计算机安全的一种有效手段,开始发展并兴起。入侵检测技术主要分为误用检测(misuse detection)和异常检测(anomaly detection)两类。其中,误用检测是根据已知的攻击特征建立一个特征库,然后将采集的数据与特征库中特征进行匹配,若存在匹配的特征,则表明是一个入侵行为。而异常检测是将用户正常的行为特征存储在特征数据库中,然后将用户当前行为与特征库中的特征进行比较。若偏离达到了一定程度,则说明发生了异常。这两种技术各有优缺点,误用检测能够准确检测到已知攻击事例,但对新型攻击行为却无能为力;异常检测可以检测到新型攻击,其误检率却比较高,且不能描述入侵行为的类别。

传统的特征码检测技术不能有效检测出病毒的变种、加密病毒和新病毒等,但异常行为检测可以通过对系统行为特征分析,判断系统是否被病毒入侵,检测出未知攻击和病毒,因而人们针对系统异常行为的检测进行了很多研究。Forrest 等在 1996 年提出了基于 UNIX 系统调用入侵检测的算法[4],认为进程的系统调用序列反映了系统的行为,通过判断进程的系统调用序列是否异常来判断系统是否被入侵;之后 Schonlau 等人提出了基于贝叶斯网络的 UNIX 系统调用入侵检测[14];人工免疫[12,13]、神经网络[6,10]、隐马尔可夫链[2,5]、数据挖掘[1,7]、有限自动机[8,11]等的一些算法都分别应用在系统调用的入侵检测上,取得了一定的效果。

这些算法主要面临两个问题:一是对系统调用进行数值化时,对每个系统调用进行数字编号是人为的,如果用信号处理或者神经网络的方法进行处理,系统调用的数字编号对结果的影响会很大;二是对数据的分类中,神经网络或者人工免疫的方法存在收敛缓慢及容易达到局部最优导致算法结束的问题,从而使得对于异常和正常调用序列的分类效果不尽人意。

针对上述两个问题,本文提出了一种新的异常入侵检测的方法并在 Windows 平台下针对 Native API 序列进行检测。该方法将序列表示为图,每个 Native API 映射为图中的一个点,Native API 的前后调用关系为连接其图中对应点的边,每个点在图里是对等的,这样消除了对数据进行数值化时的人

*)基金项目:国家自然科学基金(60702071),四川省科技厅应用基础研究基金:数字免疫网络研究(2006J13~065),教育部新世纪优秀人才支持计划(NCET-06-0811)。朱莺嘤 硕士生,从事信息安全的研究;叶 茂 副教授,从事智能信息处理、信息安全的研究;刘乃琦 教授,从事信息安全、网络与多媒体技术的研究。

为影响因素，更符合客观特性。序列中 Native API 的重复出现在图中会产生回路，定义为圈，圈出现次数多少反映出序列重复频率的高低。通过对每个点记录以其为起点的子序列，也就是该点的路径，可以直接找出圈，并对组成圈的所有边权值更新，最后根据生成图的边权值差异来判断序列是否异常。

目前关于序列图的异常检测研究还很少，Diane J. Cook 也曾经提出过一种基于图的异常检测方法[15]，该方法挖掘序列图的正常特征子结构，与之偏离的子结构就认为是异常的结构。本文的方法是根据异常序列的瞬时高频率的特性来进行检测。该特性反映到序列生成图的权值差异，该差异被利用来判断异常和正常。在 Windows 平台下，通过截获内核级的 Windows Native API[3,9]，得到正常和异常的序列。利用本算法对正常和异常序列分类，证明其能够有效地对异常入侵和病毒行为进行检测。

文中首先介绍 Windows Native API，然后提出基于图的入侵检测算法，最后是对该算法的实验结果分析及总结。

2 Windows Native API

2.1 Windows Native API 介绍

Windows 平台上的入侵检测现在主要采取检测网络数据包的方式，并没有依据入侵行为的本质特征，深入到 Windows 系统的内核，因而不能从根本上判断系统是否发生异常入侵。而基于 Linux 系统的许多这方面研究都已深入到系统的内核。通过分析特权进程的系统调用序列，判断系统是否被入侵。Windows 系统在这方面的研究还很少，本文的算法主要针对 Windows 下的 NATIVE API 序列进行分析。

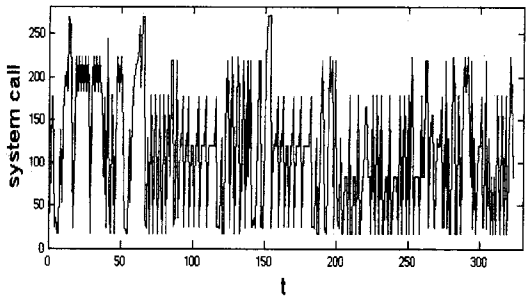
Windows 中存在用户和内核两种模式。用户应用程序在用户模式下运行，而系统程序在内核模式下运行。两种模式的重要区别是其处理文件、调用内存和使用 CPU 的优先级上不同，内核模式比用户模式拥有更高的优先级。即使用户应用程序出现了严重的错误，也不会对整个系统造成太大影响，保证了操作系统的正常运行。

API 是 Windows 操作系统在动态连接库中给用户提供系统服务的接口函数，运行在用户模式下或内核模式下。其中在内核模式下运行的 API 就是 Native API，是动态连接库中的内核级系统服务的接口函数。Native API 与用户模式下的 API 有很大的区别，其调用序列能够在内核级层次上反映应用程序的特征，因此能够用来作为异常检测的数据源。

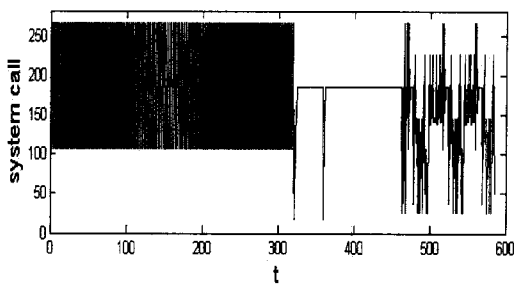
在 Win32 系统中，以下 4 个动态连接库都提供 API：User32. dll（用户接口 API），Gdi32. dll（图形接口 API），Kernel32. dll（系统管理接口 API），Adapi32. dll（高级系统管理接口 API）。其中 Kernel32. dll 提供内核模式的 API，即 Native API。在 Windows XP 系统中，大约有 949 个 Native API，Windows XP 系统中共有 949 个 Native API，其中在 NTdll. dll 中的 284 个 Native API 是系统最为关键的 Native API。Win32 API 中的所有调用最终都转向了 NTdll. dll，内核模式的驱动大部分时间调用这个模块，如果请求系统服务，NTdll. dll 的主要作用就是让内核函数的特定子集可以被用户模式下运行的程序调用。因此在实验中，我们主要跟踪截获这 284 个关键的 Native API。我们认为它类似于 Linux 的系统调用，可以基于其序列模式进行异常检测。

2.2 Windows Native API 序列分析

图 1 为实验中截获到的 Windows 系统关键进程 svchost 的正常 Native API 和异常 Native API 序列。按照横轴为时间 t、纵轴为 Native API 的数字编号作图，可以看出以下特点和问题：



(a)正常 Native API 短序列时序图



(b)异常 Native API 短序列时序图

图 1 Native API 短序列时序图

首先，数据的数字编号（在这里是指 Native API 的数字编号）由人为定义，因此对同一个序列，如果数字编号发生变化，时序图就会发生很大的改变。如果采取神经网络和信号处理的方法对时序图进行分析的话，原始数据的数字编号对结果的影响很大。

其次，正常数据和异常数据之间典型的特征是，正常数据的 API 短序列调用是一种准随机状态，即在一段时间内对每个短序列的调用频率基本相等；而对异常数据来说，短时间内对某几个短序列的调用则非常频繁，即异常数据的瞬时频率更高。因为异常入侵和病毒倾向于短时间内做出大量的破坏系统和对自身复制传播的行为，所以这也与异常入侵和病毒的本身作用机理相符合。

3 基于图的异常入侵检测算法

3.1 序列预处理

Native API 序列是在计算机运行过程中实时截取的。因为异常序列比正常序列的瞬时频率高，相同长度的正常序列和异常序列的图会表现出不同的特点，所以先采用滑动窗口方法截取等长度的序列对其进行预处理。

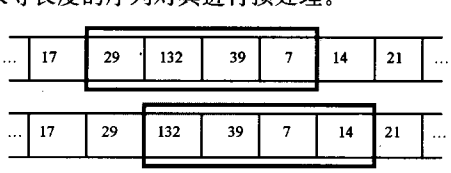


图 2 窗口滑动示意图

滑动窗口长度为 m，这里 m 的取值对于检测的准确度非常重要。m 过大，异常序列的瞬时高频性在比较长的数据中就不能凸显出来；反之，又不能把异常瞬时高频序列全部包括。在实验中，测定 m 的取值为 500 效果最佳。窗口滑动的

距离是自适应的，滑动窗口滑动到上一次数据出现最高权值的边对应的点处。这样滑动既能够包括高频率的信息，又保证了数据滑动的效率。窗口滑动方法如图 2 所示。

3.2 基本概念

在详细介绍算法之前，首先给出相关概念和定义。

定义 1 时间序列 S 是其元素按照时间排列的数据集。$S=\{s_0,s_1,\cdots,s_{m-1}\}$，$m$ 为时间序列 S 的长度，等于滑动窗口的长度。

定义 2 时间序列图 TSG 是时间序列 S 的关系图，它是一个四元组，即 $TSG=\langle V,R,E,W\rangle$，其中：

(1) V 是图中非空有穷结点集，$V=\{0,1,2,\cdots,n-1\}$，V 中的元素与 Native API 一一对应。

(2) R 是 V 中每个元素路径信息的集合。$R=\{(r_0,L_0),(r_1,L_1),\cdots,(r_{n-1},L_{n-1})\}$，$r_i$ 是结点 i 的路径，即时间序列 S 中以 i 为起点的子序列，L_i 是结点 i 的路径长度，即该子序列的长度，$i\in(0,1,2,3,\cdots,n-1)$。

(3) E 是图中有向边集。S 中的任两个相邻元素之间的前后关系都与 TSG 中相应结点的有向边一一对应。e_{ij} 为结点 i 到 j 的边，$i,j\in(0,1,2,3,\cdots,n-1)$。

(4) W 是图中有向边权值集。E 中每条边与 W 中的相应的权值一一对应。w_{ij} 为结点 i 到 j 的边 e_{ij} 的权值，$i,j\in(0,1,2,3,\cdots,n-1)$。

定义 3 圈为路径信息中出现的回路，即从某点出发又回到某点，则在图中形成圈。例如：序列 $a\ b\ c\ d\ e\ f\ a$，从 a 出发回到 a 形成了回路，即圈。

定义 4 X 为集合 R 的元素路径长度的门限值，初始值为 X_0。

定义 5 最大边权值 Max_W 是时序图 TSG 中所有边权值的最大值。

定义 6 边异常指数 $C_{ij}\in(0,1)$，为时序图 TSG 中边 e_{ij} 的异常指数，表示该边的异常程度。

定义 7 图的异常指数 $C\in(0,1)$，为时序图 TSG 的异常指数，表示该图的异常程度。

3.3 时间序列图构造算法

时序图 TSG 生成过程中，每个点记录以自身开始的路径信息，也就是以自身为起点的子时间序列。一旦某点的路径信息中出现回路，则出现圈，首先对图中组成圈的所有边权值加 1，然后清空出现圈的点的路径信息，路径长度为 0。如果某点记录的路径信息很长都未出现圈，因为超过一定的时间频率已经很低，则无论其会不会出现圈都不需要对其路径加权。因此设置路长阈值 X，当结点 i 的路长 L_i 超过门限 X，清空该点的路径信息 r_i，使路长 L_i 为 0，重新开始计算路径信息和路长。路长阈值 X 根据未形成圈路长的增加动态增大，初始值为 X_0，X_0 实验测定为 30 时效果最佳。时序图生成算法如下：

输入：时间序列 S

输出：时间序列图 TSG

construct_TSG(S)

1)集合 S 的第 k 位数据 s_k 输入。

2)若 i 与 s_k 不相等，则转到 3)。如果相等则判断结点 i 的路径信息 r_i 是否为空，如果不为空则转到 2)，如果为空则转到 3)。

3)对路径信息 r_i 记录的路径经过的所有边的权值加 1，然后清空 r_i，$L_i=0$，$X=X_0$。

4)若 L_i 大于 X，则 r_i 清空，$L_i=0$，$X=X+1$；若 L_i 小于 X，将 s_i 加入到 r_i 最末，$L_i=L_i+1$。

5)对集合 D 的每个元素重复以上 2)，3)，4)步骤。

6)对集合 S 的每个元素依次重复以上 1)，2)，3)，4)，5)步骤。

图的构造算法将每个 Native API 映射为图中的点，其相互调用关系为图中对应点之间的边。图中的点是不排序的，因此消除了人为数据编号的影响。每个点记录以该点路径信息和路径长度，一旦某点的路径中出现回路，则可找到圈，并将圈经过的边权值加 1，然后将该点的路径信息清空，路径长度为 0。每个点分布式地记录各自的路径信息，能够很简单地找到圈，避免了在整个图中找圈的复杂过程。该算法保留圈的信息于权值，然后清空圈或者过长路径的信息，避免过多的冗余路径信息增加图的复杂度、算法的时间复杂度和空间复杂度，又避免了对同一个圈的重复加权。最后生成图的权值反映了圈的出现情况，可以用于以后判断数据是否异常。

3.4 时间序列图的检测算法

长度相等序列生成的图中，正常序列图各个边的权值差异不大，而异常的图的某些边与周围的边权值差异很大。检测函数 F 用来量化边异常程度，它是计算边 e_{ij} 相对于其某一邻边的异常指数。该函数输入 e_{ij} 与其某一邻边权值差的绝对值 $|e_{ij}-e_{ik}|$ 或者 $|e_{ij}-e_{kj}|$，输出其对应的异常指数。$k=0,1,2,\cdots,n$。

函数 F 的表达式：

$$F(|e_{ij}-e_{ik}|)=(\tanh(|e_{ij}-e_{ik}|-\varepsilon)/t+1)/2 \quad (1)$$

最初权值差绝对值很小，异常程度小，函数的输出值随着权值差绝对值的增加增长速度缓慢；然后权值差绝对值较大，异常程度变大，函数的输出随着权值差绝对值的增加增长速度变快；最后权值差绝对值很大，异常程度很大，输出随着权值差异的增加增长速度变慢，输出值逐渐收敛到极限值 1。函数的整个变化过程充分表现出权值差绝对值变化对输出异常指数的影响。为了使 F 对正常和异常的序列有比较好的区分度，$F(|e_{ij}-e_{ik}|)\in(0,0.3)$ 时，增长速度缓慢；$F(|e_{ij}-e_{ik}|)\in(0.3,0.7)$ 时，函数 F 有较大的增长速度；$F(|e_{ij}-e_{ik}|)\in(0.7,1)$，增长速度又变为缓慢。这里 ε 是一个门限值，当 $|e_{ij}-e_{ik}|<\varepsilon$ 时候，函数的输出值 $F(|e_{ij}-e_{ik}|)<0.5$，表示正常；当 $|e_{ij}-e_{ik}|>\varepsilon$ 时候输出值 $F(|e_{ij}-e_{ik}|)>0.5$，表示异常。实验得出，ε 取值为 150，t 为 75，最满足以上条件。函数的图形如图 3 所示。

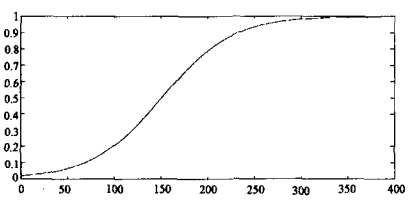


图 3 检测函数图像

异常序列瞬时频率高，因此，对于同样长度的一段数据，异常序列存在某些数据点或者数据段重复率高于正常序列。数据出现了重复，映射在图上，就产生了回路(即圈)。因此对于相同长度的序列，异常的序列出现圈的次数会明显高于正常的序列，在图中可以根据圈出现的次数来判定数据是否异常。边的权值反映了圈的出现次数，因此通过边的权值差异来判断是否异常。当某边的权值高于周围所有边的权值很

多，图中出现类似山峰的形状，肯定是异常；但是也会出现一些边，其周围边权值有大于它的，也有小于它的，不能直接判断出该边是否异常。这里用函数 F 来量化边与其所有邻边的权值差，输出其异常指数，然后求均值，能够对边的异常程度有比较精确的表示。

时序图检测算法如下：

输入：时间序列图 TSG

输出：异常指数 C

Detection_TSG(TSG)

1)若 e_{ij} 的边权值大于 Max_w/2，则对图 TSG 中与 e_{ij} 相邻的所有边计算其与边 e_{ij} 的权值差的绝对值，并输入函数 F 得到其对应的异常指数，求其均值得到边 e_{ij} 的异常指数 C_{ij}。

2)对图中所有满足边权值大于 Max_w/2 的边，重复 1)，图的异常指数 C 为这些边的异常指数的最大值。

异常的数据的瞬时频率高，其时序图中出现许多权值较大且与周围边权值差绝对值较大的边，因此该算法只选取图中权值较大的边用来检测，提高了算法效率。这里认为权值大于 Max_w/2 的边是权值较大的边。对于这些权值较大的边计算其与周围所有边的权值差的绝对值，然后输入 F，得到该权值差绝对值对应的异常指数，最后对这些异常指数求均值，得到该边的异常指数。这样可以准确地计算出边异常的指数，对于某些周围边的权值有的大于它、有的小于它的边也能够判断其是否异常。最后某序列生成的整个时序图中所有较大权值边异常指数的最大值为该序列的异常指数。根据序列生成图的最大异常边来获得该序列的异常指数，能够最大可能地检测到该序列的异常情况，判断其是否出现异常。

4 试验结果分析

本文在 Windows XP SP1 操作系统平台下收集 5 种不同漏洞攻击序列以及威金等 3 个病毒的 Native API 序列和正常系统的 Native API 序列。实验结果如下所示。

各图中的 X，Y 轴分别代表时序图的结点，Z 轴代表两点之间边的权值，采用滑动窗口取出序列的一些子序列作图。对于系统关键进程，svchost 的正常序列的多个图异常指数计算结果都小于 0.3，属于正常范围。图 4 是其中两个子序列的图，可以看出正常图比较平整，边的权值差异不大，数据分布比较均匀。

对系统攻击，会集中执行攻击相关的 Native API 序列，即攻击序列的瞬间频率高，所以异常序列图存在一些边的权值高于周围的边很多。异常序列图有一些突起的峰值点，峰值点的权值比周围的权值高出很多，如图 5 所示。

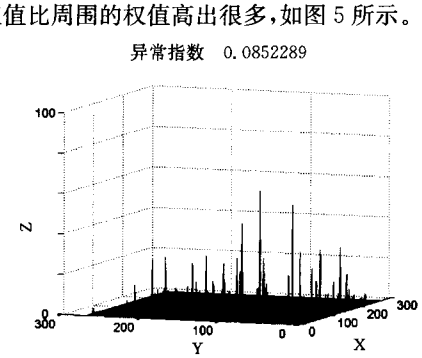


图 4 svchost 进程正常序列图

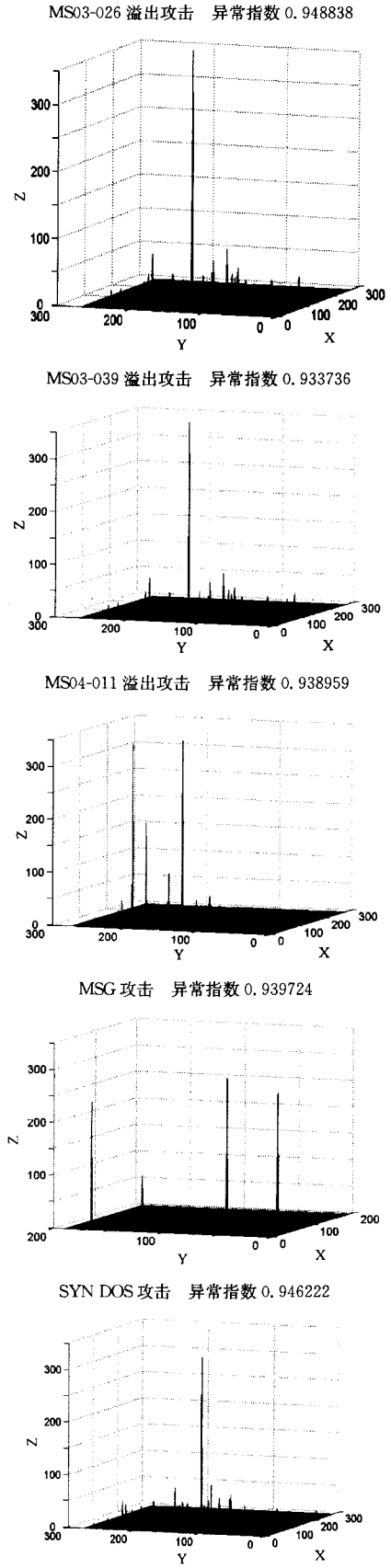


图 5 对 svchost 进程的攻击

病毒的序列也具有瞬时高频率特性，该特性跟病毒的作用机理符合。感染病毒，系统会迅速做一些危害系统的行为，占用大量系统资源进行复制传播病毒和破坏系统的行为。因此，病毒的序列比正常的序列频率更高，能够被该算法检测出

来。图6是系统正常序列图，可以看出各边的权值比较平均，差异比较小，整个图比较平整。图7是几个病毒序列图，可以看到一些突兀的峰值点，这些点对应边的权值远远大于周围边的权值。

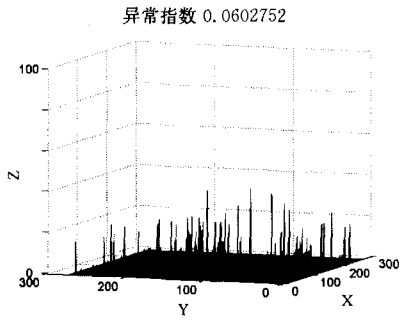


图6 系统正常情况下的调用序列图

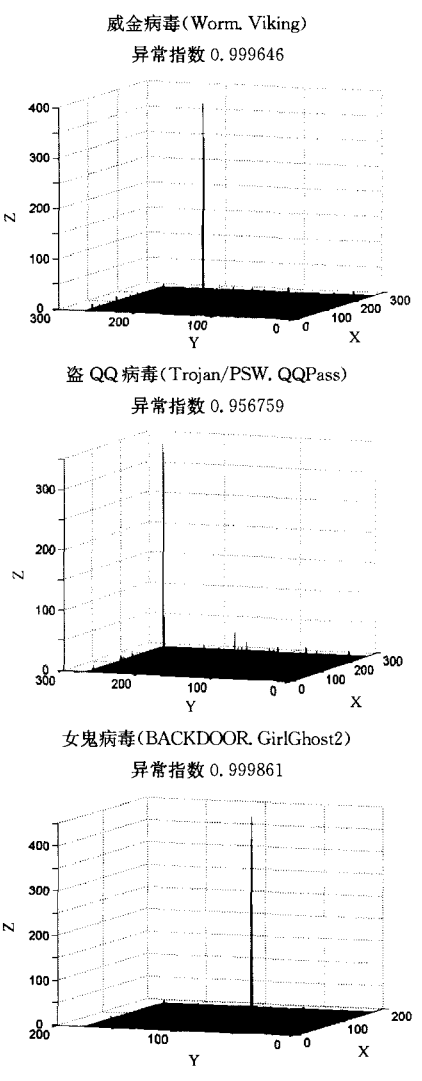


图7 病毒序列图

漏洞攻击和病毒序列图都有一些很高权值并且与周围边权值差异很大的边，映射到图中为突兀峰值点。本文算法将序列映射为时序图，通过图中各边权值差异反映出异常和正常序列的差异，进而量化该差异的程度，从而判定序列正常与否。从以上各图能够直观地看出正常序列和异常序列的差异。实验证明，本文算法能够快速准确地检测到瞬间的异常序列，并输出这些序列的异常程度指数。

结束语 本文针对异常进程活动的瞬时高频性提出异常入侵检测算法。本算法采用图的结构消除了数据数值化时人为造成的误差，利用滑动窗口取出子序列进行统计分析。采用局部点分布式地快速生成图，并从图的整体关系判定是否异常。本文算法中窗口的滑动以及检测阈值都是自适应的，由数据本身决定。在 Windows 操作系统平台上，基于 Native API 序列做的一些实验证明了本方法的可行性。

下一步的研究，将考虑把图中边的定义扩展到多维边，对应数据的多种特性，使我们能够更加准确地分析数据。通过多维边的定义，其适用范围也将大幅度提高。对于权值的变化条件或者是多维边信息变化的条件，可以扩展为重复性以外的特性来展开，将使本模型更具有普遍的意义，应用到更多的数据处理领域中。

参 考 文 献

[1] Lee W, Stolfo S J. Data mining approaches for intrusion detection// Proc. of the 7th USENIX Security Symp. San Antonio: USE-NIX, 1998: 6-9

[2] Ye N. A markov chains model of temporal behavior for anomaly detection// Proc. of the 2000 IEEE Workshop on Information As Assurance and Security. United States Military Academy, West Point: IEEE Press, 2000: 171-174

[3] Battistoni R, Gabrielli E, Mancini L V. A host intrusion prevention system for Windo-ws operating systems // 9th European Symposium on Research in Computer Security. Sop-hia Antipolice, France, 2004: 134-142

[4] Forrest S, Hofmeyr S A, Somayaji A, et al. A sense of self for UNIX proce-sses// Proceedings of the 1996 IEEE Symposium on Security and Privacy. May 1996: 120-128

[5] Lane T. Hidden Markov models for human/ computer interface modeling// Int'l AI Society, ed. Proc. of the IJCAI-99 Workshop on Learning About Users. Stockh-olm: International AI Society, 1999: 35-44

[6] Han S-J, Cho S-B. Evolution-ary Neural Networks for Anomaly Detection Based on the Behavior of a Program. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, 2006, 36(3): 559-570

[7] Michael C, Ghosh A. Simple, state-bas-ed approaches to pro-gram-based anomaly de-tection. ACM Transactions on Information and System Security, 2002, 5(3): 203-237

[8] Seka R, Bendre M, Dhurjati D, et al. A fast automaton-based method for detecting anomalous program behaviors[C]. Oakland: IEEE Computer Society, 2000: 144-155

[9] Nebbett G. Windows NT/2000 Native API Reference. Macmillan Technical Publishing (MTP), February 2000

[10] Zhang C L, Jiang J, Kamel M. Intrusion detection using hierarchical neural networks[J]. Pattern Recognition Letters, 2005, 26(6): 779-791

[11] Yu Zhenwei, Tsai J J P, Weigert T. An Automatically Tuning Intrusion Detection System. IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, 2007, 37(2): 373-384

[12] Hofmeyr S, Forrest S. Architecture for an artificial immune system. Evolutionary Computation, 2000, 8(4): 443-473

[13] Aickelin U, Greensmith J, Twycross J. Immune System Approaches to Intrusion Detection - A Review// Proceedings Inter-national Conference on Artificial Immune Systems [C]. Catania, Italy, 2004: 316-329

[14] Schonlau M, DuMouchel W, Ju Wen-hua, et al. Computer Intrusion: Masque-rades. Statistical Science, 2001, 16(1): 58-74

[15] Noble C C, Cook D J. Graph-bascd anomaly detection[C]. USA: SIGKDD Electronic edition, 2003: 631-636