一种进程系统调用重复子序列的压缩算法*)

黄梅荪　杨寿保　张　蕾　李宏伟

(中国科学技术大学计算机科学技术系　合肥 230026)

摘　要　以往入侵检测系统中采用的进程检测方法并未对进程系统调用序列中产生的重复子序列进行处理。本文提出了一种进程系统调用重复子序列的压缩算法,在系统调用序列收集过程中找出重复子序列,再将其作为一个整体参与模式的提取与检测。测试表明,对系统调用序列中的重复子序列进行压缩后能有效减少系统调用序列的长度,从而简化模式的学习和检测,提高进程检测的效率。

关键词　入侵检测,系统调用,变长模式,进程检测

A Method of Compressing Repeated Segments in System Call Sequences Executed by the Process

HUANG Mei-Sun　YANG Shou-Bao　ZHANG Lei　LI Hong-Wei

(Computer Science Dept., University of Science and Technology of China, Hefei 230026)

Abstract　The intrusion detection methods based on process don't pay enough attention to short repeated sequences of system calls executed by running processes. This paper presents a method of compressing repeated segments in system call sequences run by the process. When collecting the system calls, we mark the repeated sequences, which are treated as a whole to the process of pattern extraction. The data of experiment show that the length of the system call sequence is reduced after compressing the repeated sequences, to simplify the procedure of pattern study and detection and increase the efficiency.

Keywords　Intrusion detection, System call, Variable-length-patterns, Detection based on process

0　引言

入侵检测系统(Intrusion Detection System, IDS)是网络安全策略的重要组成部分。误用检测(misuse detection)和异常检测(anomaly detection)是 IDS 使用的两种入侵检测技术。基于进程执行过程中使用的系统调用序列的检测(以下简称为基于进程的检测或基于系统调用的检测)是一种异常检测方法。它是在这样一个前提下提出的:在正常情况下,进程执行的系统调用呈现出固有的特性;而异常情况下,进程执行的调用序列会偏离这种固有的特性。

可以将目前的进程检测方法大致分为定长模式检测和变长模式检测两种。二者的主要差异在于用以检测的片断长度上,定长模式检测依据定长的调用片断判断是否有入侵行为发生,而变长模式用于判断的系统调用序列片断长度各异。由于进程本身的特性,进程执行的系统调用序列中会出现重复子序列,这些子序列本身虽然很短,但循环出现后在整个系统调用序列中会占有一定的比例。如果将这些重复出现的子序列作为一个整体进行处理,势必能够简化基于进程检测的处理过程。本文提出一种方法首先对重复子序列进行压缩,在此基础上对调用序列模式特征进行提取和分析。

本文第 1 节介绍基于进程检测的基本原理、基本流程和定长模式检测的简单过程;第 2 节简要介绍变长模式提取方法;第 3 节分阐述了重复子序列压缩的基本原理和方法并实现了部分测试。最后对该方法进行了分析。

1　基于系统调用检测的基本原理

Forrest 等人提出利用进程执行过程中的系统调用区分正常与异常行为[1]。系统调用是内核向用户服务的唯一途径,进程在执行过程中产生系统调用序列,其固定的片断呈现出较稳定的特性。通过对这些特征片断的学习、统计和分析就能够识别出入侵行为。基于系统调用的入侵检测方法是一种异常检测技术,首先需要学习大量正常轨迹建立特征库,实际检测时,根据特征库描述的特征辨别正常与入侵行为。

基于系统调用的异常检测技术是建立在对正常序列学习的基础之上的,首先需要通过学习建立正常调用序列的模式库。为了避免异常行为的干扰,可以通过人为的方式创造一个安全的学习环境供 IDS 学习。图 1 所示的就是 IDS 的学习过程。首先,需要收集进程执行的系统调用,然后按照进程号区分不同进程执行的调用,再按执行顺序形成若干条调用序列(分别对应不同的进程),同时将不同的系统调用用不同的正整数表示。其间还可以忽略某些不涉及安全问题的系统调用。这部分工作由图 1 中的预处理模块执行。接下来就是对某一进程在不同环境下执行的多条调用序列进行模式提取、建立模式库。

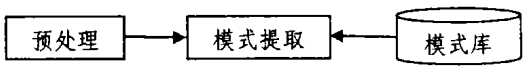

图 1　对正常调用序列进行学习建立正常调用模式

模式库中保存的是不同进程执行时的系统调用模式片断。建成模式库后,就可以对实际运行中的进程加以检测了。如图 2 所示,检测过程与学习过程类似。首先获取待查进程执行的系统调用序列、提取其模式特征、与模式库中保存的模式进行比较,根据特定的规则判断是否有入侵行为。

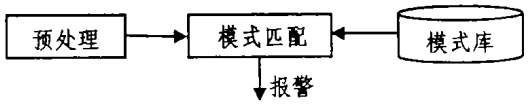

图 2　对实际运行中的进程进行检测

*)本文受国家自然科学基金项目(编号 90104030)和安徽省"十五"科技攻关项目(编号 01012013)支持。**黄梅荪**　硕士研究生,研究方向:网络安全。**杨寿保**　教授、博士生导师,研究方向:网格计算、密码学和网络安全。**张　蕾**　硕士研究生,研究方向:网络安全。**李宏伟**　硕士研究生,研究方向:网络安全。

基于进程的入侵检测方法大致可分为定长模式检测和变长模式检测方法。定长模式检测使用长度为 k 的窗口，在调用序列上滑动，依次选取长度为 k 的调用片断，作为描述调用轨迹的模式，并通过特定的方法形成模式库。利用模式库就能对实际运行的进程进行检测。再通过相同的方法获得待测进程对应的系统调用片断后，通过计算待测进程系统调用片断与模式库中模式片断的最小海明距离判断是否有入侵行为的发生[2]。

2 变长模式检测

定长模式检测能够较准确的区分正常与异常行为，但由于定长的原因，有时又无法完整刻画出系统调用序列的特征。入侵检测的能力与模式选取的长度息息相关。如果模式长度太短，则无法准确刻画出系统调用特征，即无法准确区分正常与异常行为[3,4]；如果模式选取的过长，则无疑会增加模式库存储的负担。目前模式长度的选取主要还是依赖于实际的经验数据。

一个进程在不同环境下的系统调用序列虽然存在差异，但是它们却拥有一段相同的起始子序列，且该序列通常较长。若将它们作为一个整体保存在模式库里，必定能在一定程度上简化模式提取的过程、减少检测时间。另外，还能发现有不少调用序列拥有某个相同的长子序列 S，它只是在不同的调用序列中处于不同的起始位置。序列 S 在模式库中也没有必要被拆分成为更小的子序列集合。

基于以上考虑，出现了变长模式的进程检测方法，该方法与定长模式检测方法基本类似，不同点就在于对模式描述的差异，它是通过特定算法主动地寻找出系统调用轨迹中的模式，而不是简单地通过滑动窗口盲目地标记特征。通过观察能够发现，进程执行的系统调用序列中经常出现一些重复子序列，无论是在定长模式检测还是在变长模式检测中，如果能够对这些重复子序列进行适当的处理，必然能减少检测序列的长度和检测的复杂度。以下提出对重复子序列的压缩方法和在变长模式提取中的使用。

3 重复子序列压缩

变长模式检测的关键问题就是如何从众多序列中找出表示序列特征的模式。目前存在不少算法可以从不同程度上获取序列的特征模式，例如基于 suffix tree[6]的变长模式生成算法、基于 Teiresias 的生成算法[5]等等。根据进程本身的特征，其执行的系统调用序列中往往会出现连续重复的子序列，如果在模式提取、分析中能将这些重复子序列作为一个整体看待，势必能够减少比对的时间、减少模式库的大小。本文提出一种重复子序列的压缩方法，使压缩后的重复子序列作为一个基本元素参与到模式提取及比对过程中去。以下讨论重复子序列提取的原理和方法。

3.1 重复子序列压缩的基本思想

首先观察 sendmail 进程对应的两个系统调用序列[7]：

"…4,50,27,2,5,105,104,104,106,105,104,104,106,105,104,104,106,105,104,104,106,56,19,155,83,19,4,93,94,5,112,19,93,19,100,50,128…"

"…4,50,27,2,5,105,104,104,106,105,104,104,106,105,104,104,106,105,104,104,106,105,104,104,106,56,19,155,83,19,4,93,94,5,112,19,93,19,100,50,128…"

可以发现，两条序列的差异仅仅在于"105,104,104,106"循环出现的次数不同。如果忽略"105,104,104,106"循环出现的次数，两个序列完全相同。如果使用 $(S)_l$ 表示序列 S 循环出现 l 次，则以上两个序列可以表示成"…4,50,27,2,5,$(105,104,104,106)_4$,56,19,155,83,19,4,93,94,5,112,19,93,19,100,50,128…"和"…4,50,27,2,5,$(105,104,104,106)_6$,56,19,155,83,19,4,93,94,5,112,19,93,19,100,50,128…"，这样本来可能因为"105,104,104,106"出现次数不同而将序列分拆的情况就不会发生，从而简化和方便了模式提取。

另外，对序列"85,5,5,5,5,…,5,4,59"（其中 5 出现 61 次），为了便于模式提取与比对可以将 61 个 5 作为一个整体进行处理，即 $(5)_{61}$ 表示 5 重复出现 61 次，从而将序列表示为"85,$(5)_{61}$,4,59"。这样就可以在模式提取时不受子序列重复出现的影响，同时也不丢失它们重复出现次数的信息。

3.2 改进的模式定义方法

定义 1 令 $\Sigma=\{1,2,\cdots,k\}$ 为系统调用集合，不同数字代表不同的系统调用，$|\Sigma|=k$ 为模式库中出现的不同的系统调用总数。

定义 2 定义集合 $\Sigma'=\{e_1,e_2,\cdots e_i\}$，其中 e_i 满足：

(1)$e_i\in\Sigma$ 为某个单独的系统调用；

(2)$(x)_l$；$x\in\Sigma$，l 为 x 重复出现的次数；称为循环体长为 1、循环次数为 l 的循环子模式；

(3)$(s)_l$；$s=f_1,f_2\cdots,f_j\in\Sigma$，$s$ 为系统调用序列，l 为 s 重复出现的次数；称为循环体长为 j、循环次数为 l 的循环子模式。

以下将使用 Σ' 中的元素为基本单位进行模式识别，这样可以不必对那些只包含某些子序列循环出现的调用序列进行拆分，而只需将其归为一种模式。例如可以将形如 $(x)_l$ 的序列作为一个整体进行识别。

定义 3 序列 p 为最大模式当且仅当满足以下任一条件：

(1)$p=f_1,f_2,\cdots,f_j\in\Sigma$，至少存在 p 的某个实例，它不是任何其它模式的子序列；

(2)p 中存在形如的 $(x)_l$ 或 $(s)_l$ 的序列，忽略 x 或 s 重复出现的次数，至少存在 p 的某个实例，它不是任何其它模式的子序列。

模式库中保存的将是满足定义 3 的最大模式。

3.3 重复子序列压缩的实现

对包含重复子序列的 S 的压缩分两种情况讨论：

(1)$S=x,x\in\Sigma$，对每条系统调用序列，在对系统调用进行收集的同时进行处理。设已经收集到序列：

$A_i=x_1,x_2,\cdots,x_k$ 其中 $x_i\in\Sigma,i=1,2,\cdots,k$（A 中尚未出现重复子模式），$x_{k+1}$ 为下一个待收集的系统调用。

若 $x_{k+1}\neq x_k$，则得到序列 $A_{i+1}=x_1,x_2,\cdots,x_k,x_{k+1}$；

若 $x_{k+1}=x_k$，得到序列 $A_{i+1}=x_1,x_2,\cdots,x_k.2$，其中 $x_k.2$ 表示 x_k 出现 2 次。

依次执行，若下一个系统调用 $x_{k+2}=x_k$，则得到 $A_{i+1}=x_1,x_2,\cdots,x_k.3$。例如有序列"85,5,5,5,5,…,5,4,59"（其中 5 出现 61 次），则将依次得到："85,5"、"85,5.2"、"85,5.3"、…、"85,5.61"、"85,5.61,4"、"85,5.61,4,59"。

(2)$S=x_1,\cdots,x_k,x_1,\cdots,x_k\in\Sigma$ 为子序列，由于标记子序列是为了减少模式之间非本质的差异，所以只要在最先收集的几条序列中找到重复出现的子序列是哪些。对于其余序列在收集的同时标记出这些子序列即可。另外，尽管序列中会出现循环子序列，但不同进程对应的循环子序列的数目一般只有少数的几个且长度较短。因此只需要在某个长度范围内寻找这种循环子模式。寻找到循环子序列后，在其余序列中收集这些子序列的花费也很小。例如，对于某个进程，首先获得 3 条完整的调用序列，然后在这 3 条完整序列中寻找长度≤6

的循环子序列，再在收集其余调用序列的同时标记出这些循环子序列即可。

首先考虑如何寻找循环体长为2的循环子序列。对序列 $A=a_1,a_2,\cdots,a_n$ 定义 x、y 分别为指向 a_1、a_3 的指针，

如果 $a_1,a_2=a_3,a_4$，则序列记为 $A=(a_1,a_2)2,a_3,\cdots,a_n$，表示 a_1、a_2 重复出现两次，修改 x、y 分别指向 a_1、a_5；

如果 $a_1,a_2\neq a_3,a_4$，则 x、y 分别指向 a_2、a_4；

如此循环，直至 y 指向 a_n。寻找的时间为 $O(n)$。

使用同样的方法寻找循环体长为3、4、5、6的循环子序列。总的花费为 $O(n)$。

收集到这些循环序列后，对于其他序列在收集的同时，标记出这些子序列即可，这种标记的代价也是很小的。

3.4 提取循环子模式后对模式提取的简化

模式提取包括如下过程[8]：

1. 将长度为 i 的模式扩展为长度为 $i+1$ 的模式；

2. 确定模式 i 是否为最大模式；

3. 根据观察确定是否有必要继续扩展模式，将没有必要进一步扩展的模式删除。

对包含形如 $(x)_l$ 或 $(s)_l$ 的序列该如何处理，首先让我们看下例，如图3所示。

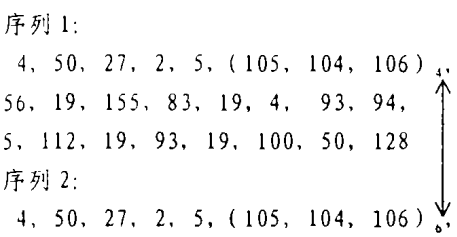

图3 提取两条序列的特征模式

寻找以4开头的特征模式时，如果不忽略“105,104,104,106”循环出现的次数，将会得到：

s_1=“4,50,27,2,5,(105,104,104,106)$_4$,56,19,155,83,19”

s_2=“4,50,27,2,5,(105,104,104,106)$_6$,56,19,155,83,19”

s_3=“4,93,94,5,112,19,93,19,100,50,128”

这三条序列。若采用本文地改进方法，就完全可以将 s_1、s_2 归为一个模式：

4,50,27,2,5,(105,104,104,106)$_{4|6}$,56,19,155,83,19

其中(105,104,104,106)$_{4|6}$表示(105,104,104,106)循环出现的次数可以是4次或6次。这样不但可以减少特征库中保存的特征片断，同时体现出(105,104,104,106)重复出现这一特征。

3.5 比较与分析

本文使用sendmail及ftp进程实现了部分测试：对收集的系统调用序列中的子模式进行处理。能够发现处理大大减少了调用序列的长度，同时发现标记出的调用子序列还可能出现在系统调用序列的其他部分、其本身就是一种模式。测试依次寻找系统调用序列中循环体长为1、2、3、4、5、6、7的循环子序列，但是对45,3,19,6,5,108,45,3.02,19,6,5,108，这样的序列，我们并不将其进行合并。下面是对两条序列处理后的结果：

对sendmail进程：简化44.62%

原系统调用序列长度：186

循环体长为1的子序列：15个，对其处理后序列长：107

循环体长为3的子序列：1个，对其处理后序列长：90

循环体长为4的子序列：1个，对其处理后序列长：83

对ftp进程：57.14%

原系统调用序列长度：399

循环体长为1的子序列：28个，对其处理后序列长：262

循环体长为3的子序列：1个，对其处理后序列长：257

循环体长为4的子序列：1个，对其处理后序列长：250

循环体长为5的子序列：1个，对其处理后序列长：241

循环体长为7的子序列：1个，对其处理后序列长：228

根据上述测试我们发现：循环子序列普遍存在于进程执行的系统调用序列中，其中以循环体长为1的子序列居多，对不同的进程，循环体长度大于1的子序列也普遍存在、但数目不多，也就是说在收集系统调用序列的同时，标记出这些循环子序列并不会给收集过程带来较重的负担、但却简化了后期的处理。同时，只要稍稍修改，就能使用已有的变长模式提取与检测方法对系统调用序列进行处理了。

增加对循环子序列的处理，还有以下优点：

(1)在对序列进行学习时，可以避免由于x循环出现次数不同而生成多个模式的情况，同时也能体现出x重复出现的次数；

(2)对于某 x、若 x 重复出现的次数为50或60，在模式中是将 $(x)_{50|60}$ 作为一个整体看待的，如果保存模式的数据结构是树则作为树的一个节点进行保存，简化了处理过程；

(3)x 在某位置上可能出现的次数被统一纪录在对应模式中，因而在实际检测过程中可以减少模式比对的工作量，尤其是当 x 重复出现在调用开始的地方。

结束语 基于进程的检测方法基于这样的假设，即在正常情况与异常情况下，进程执行的系统调用轨迹存在较大差异。它能较准确地发现入侵行为。在变长模式检测中，引入对循环子序列的处理能够简化模式的提取与检测过程。在接下来的研究中，我们将使用压缩后的系统调用序列进行模式提取和检测，进一步完善基于进程的入侵检测方法。

参 考 文 献

1 Forrest S, Hofmeyr S A, Somayaji A, Longstaff T A. A sense of self for Unix processes. Security and Privacy, 1996. In: Proc. 1996 IEEE Symposium on, 1996. 120～128

2 Hofmeyr S A, Forrest S, Somayaji A. Lightweight Intrusion Detection for Networked Operating Systems. http://www.cs.unm.edu/～immsec/publications/ids.pdf

3 Wespi A, Dacier M, Debar H. Intrusion Detection Using Variable-Length Audit Trail Patterns, Recent Advances in Intrusion Detection - Lecture Notes in Computer Science ed. by H. Debar, L. Mé, S. F. Wu., Berlin, Springer-Verlag, vol. 1907, 2000. 110～129

4 Debar H, et al. Fixed vs. variable-length patterns for detecting suspicious process. In J. J. Quisquater, Y. Deswarte, C. Meadows, D. Gollmann, eds. Proc. of the 1998 ESORICS Conference, number 1485 in LNCS, sep. 1998. 1～16

5 Wespi A, Dacier M, Debar H. An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm: [IBM Research Report]. 1999

6 Eskin E, et al. Modeling System Calls for Intrusion Detection with Dynamic Window Sizes. http://www1.cs.columbia.edu/ids/publications/smt-syscall-discex01.pdf

7 http://www.cs.unm.edu/～immsec/systemcalls.htm

8 Jiang N, et al. Exploiting Pattern Relationship for Intrusion Detection. In: Proc. of the 2003 Symposium on Applications and the Internet, Jan. 2003. 200～208