基于系统调用的软件行为模型

陶 芬[1,2] 尹芷仪[1] 傅建明[1,2,3]
(武汉大学计算机学院 武汉 430072)[1]
(武汉大学空天信息安全与可信计算教育部重点实验室 武汉 430072)[2]
(武汉大学软件工程国家重点实验室 武汉 430072)[3]

摘 要 由于系统调用信息可以在一定程度上反映程序的行为特性,因此利用系统调用来对程序行为进行建模是目前入侵检测领域的研究热点。以静态建模、动态建模和混合建模这3种不同的建模方式为切入点,按照时间顺序将基于系统调用的软件行为模型的发展划分为3个阶段:初期阶段、发展阶段和综合发展阶段。然后剖析了各阶段内的模型的发展轨迹以及它们之间的内在联系,并对它们做了横向对比分析。研究表明,基于系统调用的软件行为建模技术的发展趋势应是结合静态和动态建模技术以及结合系统调用的控制流信息和数据流信息,并综合考虑其他实时信息,如环境变量和上下文信息等,开发出检测能力更强、完备性更高以及实际可行性高的软件行为模型。

关键词 行为模型,入侵检测,系统调用
中图法分类号 TP309 **文献标识码** A

Software Behavior Model Based on System Calls

TAO Fen[1,2] YIN Zhi-yi[1] FU Jian-ming[1,2,3]
(School of Computer, Wuhan University, Wuhan 430072, China)[1]
(State Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry Education, Wuhan University, Wuhan 430072, China)[2]
(State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)[3]

Abstract Modeling program behavior based on system call has become the hot topic in intrusion detection since system call can reflect the program behavior in some degree. This paper studied three different types of modeling methods that are dynamically modeling, statically modeling and hybridly modeling as the breakthrough point, and concluded that the development process of behavior models can be divided into three stages: initial stage, developmental stage and synthetical stage. The evaluation and comparison experiments were done to find the inherent relations and development track of some typical models in different stages. The whole analysis in this paper indicates that the future trend of behavior modeling methods is to develop a software behavior model with high detection capability, completeness, and actual feasibility through the combination consideration of the static techniques with dynamic techniques, the control flow with data flow, and the other real-time information such as environment variables and context information.

Keywords Behavior model, Intrusion detection, System call

软件的行为是指软件运行表现形态和状态演变的过程。我国学者屈延文将软件行为[1]定义为软件运行时作为主体,依靠其自身的功能对客体的施用、操作或者动作。该定义指出软件行为是分层次的,从底层的二进制指令到高层的程序语句、系统调用、函数等都属于不同层次的软件行为。因为系统调用是操作系统提供给应用程序访问系统资源的接口,系统调用状况在一定程度上能够反映程序的行为特征,因此目前在国内外,很多学者都从系统调用这一层次来研究软件的行为特征[2-23]。

基于系统调用的软件行为模型就是指依赖于软件的系统调用相关信息和某种建模方法而建立的用于表征软件的正常行为特征的行为模型。由于程序受到入侵后将在所执行的系统调用状况中有所体现,因此基于系统调用的软件行为模型主要应用于异常入侵检测领域。传统的异常入侵检测是指建立系统的正常模式轮廓,若实时获得的系统或用户的轮廓值与正常值的差异超出指定的阈值,就进行入侵报警[2]。其一般是通过对系统或用户的行为进行建模的,直到20世纪90年代入侵检测技术才迎来了一个新的发展阶段。在1996年,Forrest[2]提出利用系统调用序列对程序的正常行为进行建模,随后提出基于系统调用的入侵检测系统[3],代表着基于系统调用的软件行为建模技术正式地应用于入侵检测领域。随后,国内外学者在此基础上,提出了一系列利用系统调用建立软件行为模型的入侵检测方法[2-21],使得该技术领域得到更深入、长远的发展。纵观整个基于系统调用的异常入侵检测技术的发展,其核心就在于如何构建更好的软件行为模型。因为行为模型的完备性、精确性等都会直接关系到整个入侵

到稿日期:2009-05-25 返修日期:2009-08-13 本文受863国家重点基金项目(2007AA01Z411),国家自然科学基金(90718005)资助。
陶 芬(1986—),女,硕士生,主要研究方向为网络安全,E-mail:aaronpeach@163.com;尹芷仪(1982—),女,博士生,主要研究方向为入侵检测、污点数据分析等;傅建明(1969—),男,教授,主要研究方向为软件行为、恶意代码分析、网络安全等。

检测系统的性能，所以很多学者都将研究重点放在了如何构建更加完备和精确的行为模型上。在这一发展过程中，提出了一系列典型的模型，例如 Forrest 提出的 N-gram 模型[3]，之后对其进行改进的 Var gram 模型[25]以及后续的 FSA 模型[32]、Vt-path 模型[33]、Abstract Stack 模型[31]、Execution Graph 模型[37]、Dyck 模型[38]、VPStatic 模型[39]和 HFA 模型[41]等，这些模型都应用于入侵检测系统，用来表示程序的正常行为。这些研究成果都在该领域内引起了广泛的关注。

本文以在入侵检测技术的发展过程中提出的典型的软件行为模型为研究对象，以 3 种不同的建模方式为切入点，按照时间顺序将整个软件行为模型的发展划分为 3 个阶段，并对这 3 个阶段内各自典型的行为模型进行深入分析并给出相应的模型实例。在整个分析过程中剖析了各个阶段模型之间发展的内在联系和发展轨迹，并对模型进行了横向对比分析，最后总结了该领域亟待解决的主要问题和未来发展趋势。

1 基于系统调用的软件行为模型分类

根据建模时收集软件系统调用信息方式的不同，可以将基于系统调用的行为建模主要分为 3 种方式：动态建模、静态建模和混合建模，相对应的行为模型即为动态模型、静态模型和混合模型。图 1 是对目前典型的基于系统调用的软件行为模型的分类。

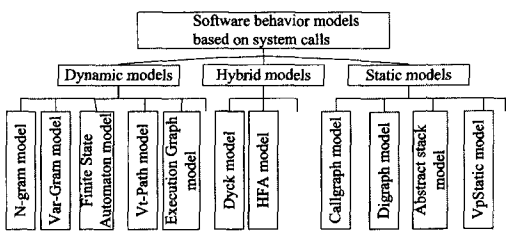

图 1 基于系统调用的软件行为模型分类

动态建模方式主要利用动态训练的方法来建立程序的正常行为模型。这种方法需要给定充足的训练数据，对程序进行大量良性执行、监控并记录下程序大量执行情况下的系统调用信息，从而生成行为模型，其依赖于外部的输入和运行环境。与之相反，静态建模方式并不直接运行程序，而是直接对程序的源代码或二进制代码进行静态分析，提取出程序的系统调用等相关信息，从而建立相应的行为模型。而混合建模方式是结合静态分析和动态分析方法，建立更加精确的、混合的行为模型，是目前研究的热点。这 3 种建模方式的比较如表 1 所列。当然，这 3 类建模方式之间并不是孤立存在的，而是相互继承和发展的。本文的下一节将深入剖析各个模型之间是如何继承和发展的。

表 1 3 种建模方式的比较

收集信息的方式		优点	缺点
动态建模	动态训练	模型表示的是程序的实际执行信息	依赖于训练时的外部输入、执行环境以及训练数据的充足性；无法捕获到程序所有可能的执行路径；误报率较高
静态建模	静态分析	可以提取程序的所有可能的执行路径，“零误报”	并不是总是可行，源代码往往不可得；二进制代码静态分析依赖于平台，较复杂；无法对自修改的程序进行静态分析；受反逆向分析技术如代码混淆技术影响
混合建模	以静态分析为主；动态分析为辅	结合静态分析和动态分析的优点，通过静态分析方式建立行为模型，然后利用动态分析技术对行为模型进行辅助修正	仍然是以静态分析为主，所以仍受到静态分析方法的限制，建模过程分为两个阶段，更复杂

2 基于系统调用的软件行为模型

总的来说，按照时间顺序可以将整个基于系统调用的软件行为建模技术的发展过程划分为 3 个阶段。

初期阶段：这一阶段的成果大多是对短序列模型的改进，都是采用动态建模方式建立行为模型。典型的模型有 Forrest[4]提出的 N-gram 模型和 Wepsi[25]提出的 Var-gram 模型。

N-gram 模型：Forrest[4]在 1998 年提出的 N-gram 模型是第一个基于系统调用的程序行为模型。N-gram 是指长度为 N 的系统调用短序列，即以长度为 N 的系统调用短序列来表示程序的正常行为。短序列的生成采用滑动窗口机制，建立一个大小为 n 的滑动窗口，记录下窗口内系统调用之间的跟随关系。

例如图 2 中记录的程序的系统调用序列为：open，read，mmap，mmap，open，getrlimit，mmap，close，开启一个大小为 $n=4$ 的滑动窗口来记录系统调用之间的跟随关系，就可以获得图 2 所示的长度为 $n=4$ 的短序列。

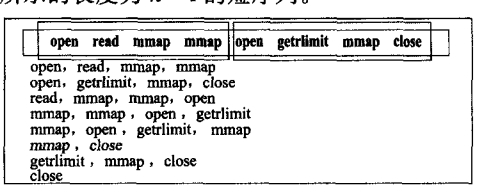

图 2 $n=4$ 滑动窗口示例

对于一个进程，一旦其正常的系统调用短序列数据库被构造，则可用它来监视进程的执行行为。异常检测的方法是计数不匹配的系统调用序列对的数目。该模型具有简单易懂、实现方便、利于实时监测的优点，是首个利用系统调用的跟随关系来检测攻击的模型，不过该模型的检测能力非常有限。随后，为了克服定长缺陷，Wepsi[25]提出了不定长序列模型 Var-gram。

Var-gram 模型：由于 Forrest 提出的 N-gram 是采取定长截取系统调用得到短序列集的，因此它的最大缺陷就是短序列定长，而实际程序的不同功能片断不一定是对应同样长度的，也就是说很难找到一个合适的长度。所以需要挖掘出程序中每个功能片段所对应的不定长短序列。这里将一个不定长短序列称为一个不定长模式。Var-gram[25]引入生物信息中寻找 DNA 序列不定长模式的 TERESIAS 算法[26]来发现系统调用序列的基本模式，然后精简这些基本模式，以找到能够覆盖原来序列的最简模式集。较之 N-gram 而言，其提取得到的行为模式是不定长的，比较符合程序的实际情况。同时，该模型的检测能力比 N-gram 有所提高，并且空间和时间复杂度比 N-gram 均有所提高。

在这个阶段，其他的行为模型本质上都是以序列模型为基础而进行的一系列改进，如引入系统调用的频率特性[27]、引入数据挖掘理论[28]和隐式马尔可夫链(HMM)[29,30]理论等。整体来说，这一阶段的软件行为建模都采用动态建模方

式来动态捕获程序执行时的系统调用序列。而捕获的信息也仅限于系统调用本身，根据系统调用的跟随关系建立短序列模型。由于其建模时依赖的系统调用信息非常有限，使得这些模型的检测能力都非常有限，难以抵抗复杂的攻击，如模仿攻击、数据流攻击等。

发展阶段：在发展阶段中，除了动态训练建立行为模型方式之外，Wagner[31]提出了静态建立行为模型的思想。这一阶段由于序列模型存在误报率高并且不能表示程序的分支和循环结构，因此研究者们要寻求比短序列能更好地表现程序行为的方式。由于 FSA(Finite State Automaton，有限状态自动机)能很好地表现程序的分支和循环结构，因此自动机便成为这一时期行为模型的主要表现形式。这一时期典型的、动态建立的行为模型主要是 Sekar[32] 提出的 FSA 模型以及 Feng[33] 提出的 Vt-Path 模型。

FSA 模型：是由 Sekar 针对 N-gram 模型的缺陷而提出的一种新的行为模型，采用有限状态自动机的方式来表示程序行为。Sekar 总结了 N-gram 模型主要存在如下缺陷：1)短序列的数目是随 N 呈指数增长，空间复杂度较大；2)不能捕获系统调用序列之间长跨度的关联关系；3)若程序出现正常的、细微的改变而出现与原来短序列数据库不相匹配的短序列，则可能会认为异常；4)仅从系统调用短序列上不能很直观地反映程序的循环、分支、递归等结构。针对上述缺陷，作者提出了 FSA 模型。作者采用动态训练方式，结合系统调用的 PC 值构造出程序的执行路径的有限状态自动机，自动机上结点为系统调用对应的 PC 值，而状态转换为系统调用。图 3 为一个简单的 FSA 模型实例。不过对于该方法而言，最大的问题就是存在不可能路径，所以后续研究者们提出了对该方法的改进模型。

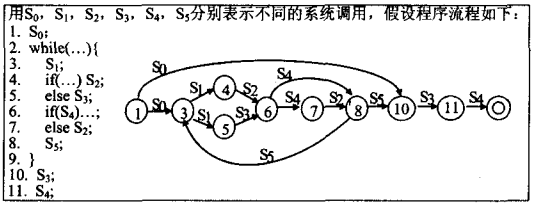

图 3　FSA 模型实例

Vt-path 模型：不管是动态建立的 FSA 模型还是静态建立的 Abstract stack 模型[31]，都需要建立自动机，因此存在时间、空间消耗较大以及不可能路径的问题，所以 Feng[33] 提出了一种新的模型，称为 Vt-Path(Virtual Path，虚拟路径)模型。这种方法采用动态训练方式建立程序的正常行为模型，利用程序执行时的返回地址、程序计数器值以及系统调用信息构造两个哈希表：一个用于存储返回地址列表(RA)，一个用于存储两个系统调用之间的虚拟路径(VP)。利用这两个表可以检测出很多攻击并且避免了构造自动机和不可能路径。图 4 就是一个简单的 Vt-path 模型实例。Vt-Path 模型与 FSA 模型相比，虽然两个模型的收敛时间差不多，但是 Vt-Path 模型比 FSA 模型的状态转换要少些，从而提高了精度和效率，同时其处理动态链接库更为有效，误报率相对较低。Vt-Path 模型与这一时期的静态提取方式建立的 Abstract stack[31] 模型相比，不需要构造下推自动机，时间和空间效率要高些。当然，其最重要的优点是解决了不可能路径问题，使得检测执行速度更快、检测能力更高。但由于需要构造两个哈希表，使得空间复杂度也比较高。

假设 $A=\{a_0,a_1,\cdots,a_{n-1}\}$ 和 $B=\{b_0,b_1,\cdots,b_{m-1}\}$ 分别表示当前系统调用虚拟栈列表和上次系统调用的虚拟栈列表，那么它们之间的虚拟路径为：

$P=b_m\rightarrow Exit;\cdots;b_{i+1}\rightarrow Exit;b_i\rightarrow a_i;$

$Entry\rightarrow a_{i+1};\cdots;Entry\rightarrow a_n$

Entry 和 Exit 分别代表函数的入口和退出点的 PC 值。

例如程序源代码如下：

```
int main(int argc,char* argv[]){
1: int a,b;
2: a=1;b=2;
3: f(a);
4: g();
5: f(b);}
void f(int x){
1: sys_call(5);
2: if(x==1)
3:  sys_call(3);
4: else if (x==2)
5:  sys_call(4);}
void g(){
1: sys_call(2);}
```

则可以得到

sys_call(2)的虚拟栈列表为 $A_2=\{main.4,g.1\}$

sys_call(3)的虚拟栈列表为 $A_3=\{main.3,f.3\}$

sys_call(4)的虚拟栈列表为 $A_4=\{main.5,f.5\}$

sys_call(5)的虚拟栈列表为 $A_5=\{main.3,f.1\}$和$\{main.5,f.1\}$

例如 $A_2=\{main.4,g.1\}$；$A_3=\{main.3,f.3\}$；则它们之间的虚拟路径为：$P=f.3\rightarrow Exit;main.3\rightarrow main.4;Entry\rightarrow g.1;$

图 4　VtPath 模型实例

除了继续沿着动态训练方式建立行为模型之外，一些学者也在积极研究不执行程序，而直接通过静态分析程序的源代码或二进制代码就能抽象出程序的行为模型的方法。Wagner[31] 最早提出通过静态分析源代码建立 3 种程序行为模型：callgraph 模型[31]、digraph 模型[31] 和 Abstract stack 模型[31]。callgraph 模型是基于不确定有限自动机(NFA)的，虽然高效但却可能存在“不可能路径”，攻击者有可能利用这些路径躲避检测。Abstract stack 模型是基于下推自动机(PDA)的，该上下文相关的 PDA 模型包含了对进程的函数调用栈的抽象，不过运行代价过高。所以 Wanger 最后提出一种 digraph 模型作为折衷的方案。

Abstract stack 模型：主要是针对 FSA 模型[32] 的缺陷而提出的一个静态分析模型。在该模型中引入了静态分析的思想，基于对程序源代码的静态控制流分析，先获取程序的 call graph model(调用图模型)，然后再由 call graph model 抽象得到 abstract stack model。最后的 abstract stack model 不但对 PC 值建模，而且对 call stack 的状态进行建模，采用上下文无关文法，建立起 NDPDA(非确定性下推自动机)模型。从本质上来说，Wanger 提出的 callgraph model 和 Sekar 提出的 FSA 模型都是根据系统调用序列建立起的一个 NDFA(非确定性有限自动机)模型，也就是说二者都存在不可能路径。如图 5 中的实例，因为 v 和 w 都可以转换到 $Entry(f)$，那么此 NDFA 模型中存在不可能路径：$v\rightarrow Entry(f)\rightarrow\cdots\rightarrow Exit(f)\rightarrow w'$。出现不可能路径的主要原因是没有考虑返回地址，所

以 abstract stack model 不但对 PC 值建模，而且对 call stack 的状态进行建模，采用上下文无关文法，建立起 NDPDA(非确定性下推自动机)模型。图 6 为 abstract stack 模型的一个简单实例。

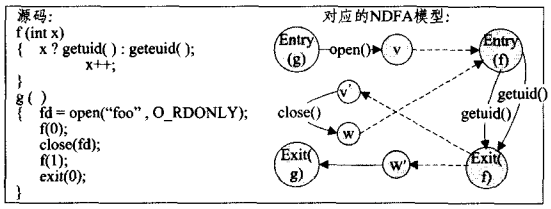

图 5 NDFA 模型实例

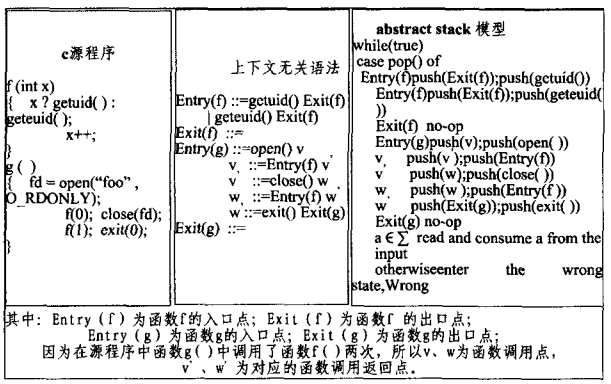

图 6 Abstract stack 模型实例

随后为了克服对源代码的依赖这一缺陷，Giffin[14] 提出了针对二进制代码建立的 NFA 和 PDA 方法。总的来说，这一阶段由于静态提取建立行为模型这一方式的提出使得基于系统调用的软件行为建模出现了两个主要的发展方向：动态建模和静态建模。这两种建模方式有各自的特点，并直接决定了其应用于入侵检测系统时的整体性能。本文在上一节中对这两种方法进行了详细比较。总的来说，二者各有利弊，相互补充。

同时在这一阶段，Wanger[35] 也提出了针对基于系统调用序列建模的入侵检测系统的各种攻击思想，其中最重要的一种攻击方式就是模仿攻击。攻击和检测是相对的，只有从攻击和防御两个角度来研究入侵检测系统，才能从根本上提高入侵检测系统的抗攻击能力。相继提出的很多基于系统调用的入侵检测方法，由于其在行为建模方面的共性，往往一种攻击方式可以同时对多种行为模型攻击成功。如 N-gram 方法和 Var-gram 方法都只考虑了系统调用的时序关系，因此都不能对抗那些不对系统调用时序序列造成更改的攻击，如改变系统调用参数等。从 Wanger 提出了模仿攻击之后，相继有很多学者也在这方面做了许多工作，使得研究模仿攻击也逐渐成为基于系统调用的异常入侵检测系统这一领域的一个研究重点。该文献的另一个重要意义就是深入讨论了系统调用的参数问题，使得系统调用参数分析逐渐被学者们重视起来。随后 Kenegel[36] 提出针对系统调用的参数特征(如参数的字符串长度等特征)进行建模，这一方向后来发展为基于系统调用的数据流分析的入侵检测系统。

综合发展阶段：综合发展阶段中，提出一些静态、动态混合的行为模型，也预示着将动态、静态方式相互结合建立更加精确的行为模型是基于系统调用的软件行为建模技术的发展趋势。

这一时期动态模型方面的焦点是灰盒模型，其典型的行为模型是 Gao[37] 提出的 Execution graph 模型。Gao 将基于系统调用进行程序正常行为建模的方法分为 3 类：白盒法、黑盒法和灰盒法(Gray-Box)，其分类的依据是建立正常行为模型所使用的信息。黑盒方法和灰盒方法都是通过动态监控程序样本的执行来建立系统调用行为模型的。其中，黑盒方法仅仅使用了系统调用号(一个系统调用号 system call number 代表着一个指定的系统调用)，当然有些黑盒方法也引入了系统调用参数分析。而灰盒方法除了系统调用号之外还需要提取更多的、附加的实时信息，如查看进程执行时的内存信息(如调用堆栈的值、程序计数器的值等)。按照这种思想，前面的 FSA 模型[32]、VtPath 模型[33] 都应该属于灰盒模型。白盒方法就是通过静态分析程序的源代码或二进制文件来建立行为模型，即前面提到的静态模型。

灰盒思想的提出标志着在利用系统调用进行程序行为建模时，单纯依赖系统调用本身已经不可行，还必须考虑其他执行信息，以使模型更加精确、完备，才能更好地抵抗各种类型的攻击。

Execution Graph 模型：Gao 提出的 Execution graphs(执行图)[37] 对动态分析技术的一个“标志性”改进是仅仅通过动态的灰盒分析技术就可以达到白盒分析的目标。其模型最大的创新点在于提出了一种思想，就是用动态分析的方法去模拟静态分析的结果，用 execution graphs 去模拟控制流程图的功能，可以在不需要对程序静态分析的情况下使提取的 execution graphs 的结果符合静态提取得到的控制流程图。模型的建立使用灰盒分析技术，收集的行为信息除了系统调用信息之外还有程序运行时的调用堆栈(call stack)信息，使得模型较为精确。图 7 是一个简单的 Execution graph 模型的实例。

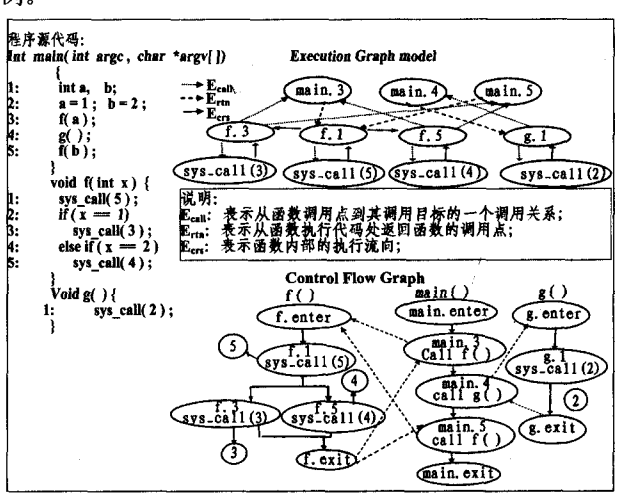

图 7 Execution graph model 的实例

不过，Gao 的 Execution graph 模型虽然是灰盒方法，但在进行行为建模时并没有考虑系统调用参数，不能检测针对系统调用参数的攻击。同时，因为作者提出的执行图是与程序的控制流程图对应的，其建立依赖于调用堆栈信息。如果攻击者能伪造调用堆栈信息，或者不对程序的控制流程进行更改而可以直接达到攻击目的，则检测系统将检测不出攻击。该模型的提出可以看出学者们已经在尝试将动态建模和静态建模这两种方式关联起来，因此在后续相继提出了一些动态、静态结合的模型。

Dyck 模型：Giffin 首次提出了一个具体的动态分析和静态分析相结合建立的模型，即 Dyck 模型[38]。由于前面的模型都没有考虑到程序上下文相关、系统调用参数，因此很难检测出模仿攻击和对非控制数据攻击，所以 Giffin 提出了 Dyck 模型。Dyck 模型是一个将静态技术与动态技术相结合来建立的行为模型。初期的 Dyck 模型是通过对二进制代码进行静态分析来建立的，然后再采用动态的“压制”技术来删除多余的 null calls 操作对和没有系统调用的子路径，得到更新精简后的 Dyck 模型。该模型是第一个考虑程序上下文相关的模型。整个异常入侵检测系统会首先对二进制代码按照一定的规则重写，使其能更完备和有效地生成模型。然后再对已重写过的二进制可执行文件进行实时检测，监控该文件实时的行为轨迹，检测实时行为是否与已建立的模型一致。除了动态分析和静态分析相结合建立模型之外，另一个显著特点是进行了数据流分析，数据流分析的加入可以提高模型的精确性和抗模仿攻击能力。

Giffin[39]随后也提出对该模型的改进，即在 Dyck 模型基础上考虑环境变量，使得检测攻击的能力更强。虽然程序上下文和数据流分析的加入使得 Dyck 模型能更为精确地描述程序的行为，并且完全剔除了不可能路径，但该模型也存在考虑因素较多、时间复杂度和空间复杂度比较大、实用性不高的缺陷。

VPStatic 模型：在动态建模和静态建模之间寻求平衡的另一个尝试就是 Feng 提出的 VPStatic 模型[40]。该模型是 VtPath 模型的免训练版本，即静态分析版本。作者的另一个重要贡献就是提出对于通过静态方式建立的自动机形式的行为模型，其精确性本质上是和自动机的确定性相关的。也就是说，自动机的确定性越高，表示模型越精确，越能准确地表现程序行为。Dyck 模型和 VtStatic 模型本质上都是 sDPDA（栈确定下推自动机）模型，可以证明 sDPDA 比单独的 PDA（下推自动机）的精确性更好。

HFA 模型：我国学者李闻等[41]在此基础上提出了一个混杂模型 HFA，它本质上是一个 DPDA（确定性下推自动机）模型。该模型采用静态分析加上动态绑定的方式，将静态分析中无法解决的问题推迟，在运行过程中利用对函数局部自动机的动态链接和运行时修正消除这种不确定性，比同类自动机模型更接近 DPDA，从而获得更高的精确性和效率。

总的来说，这一阶段的模型都体现着将动态分析和静态分析相互结合的趋势，也是未来基于系统调用的软件行为建模技术的发展趋势。与这一阶段模型发展紧密相关的一个方向就是基于系统调用的数据流分析。可以说，前面介绍的所有模型都是以系统调用的控制流分析为主的。为了增强模型的精确性，有些模型也进行了一些系统调用的参数和返回值分析。基于系统调用进行行为建模可以从系统调用的控制流层面分析，也可以从系统调用的数据流层面分析。

系统调用的数据流分析是指分析各个系统调用的参数和返回值之间的数据传播关系。基于系统调用的控制流分析的行为模型仅能检测出那些改变程序控制流的入侵，例如代码注入类型攻击，因为代码注入类型攻击将会通过添加一些额外的系统调用来改变程序的控制流向。监控系统调用的参数的优点是可以检测出一些更隐蔽的攻击，这些攻击没有改变程序的控制流而仅仅改变了程序的参数，如典型的缓冲区溢出攻击。数据流分析的提出是为了辅助控制流分析，它建立在控制流分析基础上，用于提高控制流分析模型的精确性。模型的精确性表示模型的检测攻击的能力，模型越精确，对程序正常行为的表示能力越强，则检测攻击的能力越强。

前面已经提到在 2002 年 Wanger[35]提出针对序列模型的模仿攻击方法，使得模型的检测攻击能力成为模型研究的热点。之后提出的模型都在检测能力上逐渐改进提高。在建模时考虑系统调用的参数约束，可以有效地提高模型的检测能力，在这个方向上 Kruegel[36]，Sufatrio[42]，Tandon[43]先后研究了如何针对单个系统调用的参数特征进行建模以及通过学习机制建立系统调用参数的值集规则。在 2006 年，Bhatkar[44]进一步扩展了这些研究，提出了基于数据流的异常检测，提出在一个控制流上下文中通过数据流分析来提取系统调用参数之间的关系规则。不过，这里的控制流上下文信息仅是指程序计数器的值，用以区分不同位置的同名系统调用。2008 年，Li Peng[45]对此进行了改进，结合控制流分析先获取系统调用模式集。每个模式是指一个具有特定功能的系统调用子序列，然后通过关系挖掘技术提取出更完备、更有用的规则。

总之，数据流分析是建立在基于控制流分析建立的行为模型的发展基础之上的，其目的是提高基于控制流建立的行为模型的精确性。结合动态分析和静态分析，以及控制流分析和数据流分析，从而创建更加精确、完备、有效的行为模型，是基于系统调用的入侵检测系统的未来趋势。

3 各种模型的评估

基于系统调用的软件行为模型有两个重要的评估指标：精确性、完备性。模型的精确性体现为模型检测攻击的能力。精确性越高，模型的检测攻击的能力越强，因为模型越精确，攻击者越难躲避检测。模型的完备性体现在模型是否能完整地表示软件行为，即不会将正常的软件行为误报为入侵。

本文按照 3 个阶段和 3 种不同的模型类型，从建模对象、检测攻击的能力（精确性）、模型的完备性以及整体优缺点等几个方面对前面介绍的典型行为模型进行了对比分析，结果如表 2 所列。

表 2 基于系统调用的软件行为模型比较

阶段	模型	检测攻击能力（精确性）	完备性	整体优点	整体缺点	
初期阶段	动态模型为主	N-gram	仅依赖系统调用的跟随关系建模，检测攻击的能力非常有限	模型的完备性依赖于训练环境和训练数据的完备性	模型简单易懂，实现方便，利于实时监测；	动态训练方式建模导致模型完备性难以保证；建模时所依赖的信息过于简单，使得检测能力非常有限
		Var-gram	同样仅依赖系统调用的跟随关系建模，检测攻击的能力非常有限	模型的完备性依赖于训练环境和训练数据的完备性	采用不定长模式，使得检测能力有所提高，同 N-gram 一样模型较简单，易于实现	同 N-gram 一样动态训练方式建模导致模型完备性难以保证；检测能力也非常有限

发展阶段	动态模型	FSA	依赖于系统调用的 PC 值来建立 FSA 模型，比短序列方式的检测能力有所提高	模型的完备性依赖于训练环境和训练数据的完备性	自动机方式比短序列方式能更完备的表示程序行为，使得其检测能力、误报率、学习时间、匹配速度等都有所提高	存在不可能路径问题，攻击者可以利用不可能路径躲避检测；同样建模时考虑的信息有限，使其无法检测复杂的攻击
		Vt-Path	较 FSA 方法因为解决了不可能路径，所以检测能力有所增强	模型的完备性依赖于训练环境和训练数据的完备性	完全解决了不可能路径，执行速度更快，检测能力提高	仅仅考虑返回地址、系统调用序列和 PC 值，所以还是无法检测出一些智能攻击比如模仿攻击，数据流攻击等等
	静态模型	Abstract stack	存在不可能路径问题，使得攻击者可以通过这些不可能路径来躲避检测	静态模型，包含程序所有可能的执行路径	首次研究静态建模，因为考虑栈的返回地址，可以剔除一部分 FSA 中的不可能路径，并且保证路径不会被忽略；	由于 NDPDA 也具有不确定性，所以它也可能引起一些不可能路径；仅仅考虑了栈的返回地址，还是无法检测出复杂的攻击，如模仿攻击，数据流攻击等
综合发展阶段	动态模型	Execution Graph	不能检测针对系统调用参数的攻击；如果攻击者能对调用堆栈信息进行伪造、或者不对程序的控制流程进行更改而可以直接达到攻击目的，则检测系统将检测不出攻击	模型的完备性依赖于训练环境和训练数据的完备性	用动态分析的方法去模拟静态分析的结果，用执行图去模拟控制流程图的功能。可以不需要对程序静态分析而获取的结果可以符合静态提取得到的控制流程图	仅考虑了程序运行时的调用堆栈信息，没有考虑系统调用参数等其他信息，因此还是无法检测出复杂的攻击，如模仿攻击，数据流攻击等
	静态模型	VPStatic	基于静态分析的上下文敏感模型，能够解决不可能路径问题和防止模仿攻击	静态模型，包含程序所有可能的执行路径	是 VtPath 模型的免训练版本，上下文敏感	本质上是栈确定下推自动机 sDPDA 模型，所以模型精确性较高
	混合模型	Dyck	考虑上下文、数据流和环境变量，检测能力是所以方法中最强的	静态模型，包含程序所有可能的执行路径	第一个混合模型，有效结合静态分析和动态分析，数据流分析、上下文和环境变量，使得模型更完备，检测能力更强	模型较复杂，实用性不高
		HFA	没有进行数据流分析，在建立模型时没有考虑系统调用的参数，所以不能检测数据流攻击、模仿攻击。	静态模型，包含程序所有可能的执行路径	本质上是一个 DPDA，比其他静态模型相比确定性最高，模型更精确	没有考虑数据流分析，是对 Dyck 模型的改进，因此模型也比较复杂，实用性不高

结束语 纵观整个基于系统调用的软件行为建模技术的发展过程，可以将其按照时间顺序划分为 3 个阶段：初期阶段、发展阶段、综合发展阶段。这 3 个阶段对应的主流建模方法依次为动态建模、静态建模和混合建模，而建模所依赖的系统调用信息也从最初的简单的系统调用时序信息，发展到系统调用参数信息、附加的实时信息（如调用堆栈的值、程序计数器的值）以及后来的上下文信息、环境变量和数据流信息等。研究者们正是按照这一发展轨迹，从提高模型的检测攻击能力和完备性这两个角度出发研究出一系列的基于系统调用的入侵检测方法。

不过，目前基于系统调用的软件行为建模技术面临的主要问题就是如何在性能和实时开销上达到平衡。一般在提高模型的完备性和精确性的同时会增加模型的复杂性和实时检测的难度，这直接影响着检测方法的实时开销。只有在性能和开销上做到有效平衡的方法才是可行的，才能实际运用于入侵检测领域。因此，对于研究者们所提出的任何模型，一个最终的评估指标就是该模型是否是实际可行的，即达到可行性要求。

在基于系统调用的软件行为建模技术发展的初期阶段提出的 N-gram 和 Var-gram 等模型，虽然较简单易懂，实现方便，可以应用于实时检测，但这些模型毕竟过于简单，考虑的因素太少，导致误报率非常高并且检测能力非常有限，所以并不适合应用到实际的入侵检测系统中。当然，正是基于其在检测性能方面的显著缺陷，才导致了后续大量对其进行改进的研究成果的出现。在后续两个阶段的研究方法主要将重点放在了如何提高检测性能即提高检测能力的同时降低误报率上面，如 Abstract stack，Vt-Path，Execution graph，Dyck，Vp-Static 和 HFA 方法等。在这些方法的发展过程中，模型考虑的因素越来越多，导致模型的复杂性越来越高，从而直接提高了实时检测的难度和开销，以至于这些方法的可行性都不高。因此，如何在提高方法的完备性和精确性的同时，尽可能低地降低实时检测的开销，是未来该领域亟需解决的主要问题。

总的来说，基于系统调用的软件行为建模技术的发展趋势就是结合静态建模和动态建模技术，以及结合系统调用的控制流信息和数据流信息，同时综合考虑其他实时信息，如环境变量和上下文信息等，开发出检测能力更强、更完备以及实际可行性高的软件行为模型。

参 考 文 献

[1] 屈延文. 软件行为学[M]. 北京：电子工业出版社，2004

[2] Forrest S. A sense of self for UNIX processes[C]// Proc. of the IEEE Symp. on Security and Privacy. Oakland: IEEE Press, 1996: 120-128. http://www. cs. unm. edu/~ forrest/publications/ieee-sp-96-unix. pdf

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

[4] Ramkumar C, van den B E. A Fast Static Analysis Approach to Detect Exploit Code Inside Network Flows[C]// Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID). Berlin: Springer Verlag, 2006, 3858: 284-308

[5] Liu Zhen, Bridges S M, Vaughn R B. Combining Static Analysis and Dynamic Learning to Build Accurate Intrusion Detection Models[C]// IEEE International Information Assurance Workshop 2005. Washington D C: IEEE Computer Society Press, 2005: 164-177

[6] Giffin J T, Jha S, Miller B P. Detecting manipulated remote call

streams[C]// Proc. of the 11th USENIX Security Symp. San Francisco: USENIX, 2002: 61-79

[7] Gao Debin, Reiter M K, Song Dawn. Behavioral Distance for Intrusion Detection[C]// Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005). Seattle, WA, USA, September 2005

[8] Gao Debin, Reiter M K, Song Dawn. Behavioral Distance Measurement Using Hidden Markov Models[C]// Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID 2006). Hamburg, Germany, September 2006

[9] Gao Debin, Reiter M K, Song Dawn. Beyond Output Voting: Detecting Compromised Replicas Using Behavioral Distance[R]. CMU-CYLAB-06-019. December 2006

[10] Gao D, Reiter M K, Song D. On gray-box program tracking for anomaly detection[C]// USENIX Security Symposium. SanDiego, California, August 2004

[11] 姚立红，酋小超，黄皓，等. 基于系统调用特征的入侵检测研究[J]. 电子学报，2003(08)：1134-1137

[12] 闫巧，谢维信，宋歌，等. 基于 HMM 的系统调用异常检测[J]. 电子学报，2003(10)：1486-1490

[13] 谭小彬，王卫平，奚宏生，等. 计算机系统入侵检测的隐马尔可夫模型[J]. 计算机研究与发展，2003(02)：245-250

[14] 贾春福，钟安鸣，周霞，等. 基于系统调用的 Linux 系统入侵检测技术研究[J]. 计算机应用研究，2007，24(04)：147-150

[15] 冯力，管晓宏，郭三刚，等. 采用规划识别理论预测系统调用序列中的入侵企图[J]. 计算机学报，2004，27(08)：1083-1091

[16] 苏璞睿，杨轶. 基于可执行文件静态分析的入侵检测模型[J]. 计算机学报，2006，29(9)：1572-1578

[17] 徐明，陈纯，应晶. 一个两层马尔可夫链异常入侵检测模型[J]. 软件学报，2005，16(02)：276-285

[18] 尹清波，张汝波，李雪耀，等. 基于动态马尔科夫模型的入侵检测技术研究[J]. 软件学报，2004，32(11)：1785-1788

[19] 徐明，陈纯，应晶. 基于系统调用分类的异常检测[J]. 软件学报，2004，15(03)：391-403

[20] 张诚，彭勤科. 基于系统调用与进程堆栈信息的入侵检测方法[J]. 计算机工程，2007，33(7)：139-142，148

[21] 张义荣，鲜明，肖顺平，等. 一种基于神经网络和系统调用的异常入侵检测方法[J]. 计算机应用研究，2006，23(9)：119-121

[22] Peng Guojun, Pan Xuanchen, Fu Jianming, et al. Static Extracting Method of Software Intended Behavior Based on API Functions Invoking[J]. Wuhan University Journal of Natural Sciences, 2008, 13(5): 615-620

[23] Peng Guojun, Pan Xuanchen, Zhang Huanguo, et al. Dynamic Trustiness Authentication Framework Based on Software's Behavior Integrity[C]// Proceedings of The 9th International Conference for Young Computer Scientists (ICYCS 2008). Hunan, China, 2008

[24] 卿斯汉，蒋建春，马恒太，等. 入侵检测技术研究综述[J]. 通信学报，2004，25(7)：19-29

[25] Wepsi A, Dacier M, Debar H. Intrusion detection using variable-length audit trail patterns[C]// Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection. London, U K: Springer Verlag, 2000: 110-129

[26] Rigoutsos I, Floratos A. Combinatorial pattern discovery in biological sequences[J]. Bioinformatics, 1998, 14(1): 55-67

[27] Helman P, Bhangoo J. A statistically based system for prioritizing information exploration under uncertainty[J]. IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans, 1997, 27(4): 449-466

[28] Lee W, Stolfo S J. Data mining approaches for intrusion detection[C]// Proc. of the 7th USENIX Security Symp. San Antonio, 1998: 26-40. http://www.usenix.org/publications/library/proceedings/sec98/full_papers/lee/lee.pdf

[29] Lee W, Stolfo S J, Chan P K. Learning patterns from UNIX process execution traces for intrusion detection[C]// AAAI Workshop on AI Approaches to Fraud Detection and Risk Management. AAAI Press, 1997: 50-56. http://www.cc.gatech.edu/~wenke/papers/osid_paper.ps

[30] Gao D, Reiter M K, Song D. Behavioral distance measurement using hidden Markov models[C]// Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID 2006). 2006

[31] Wagner D, Dean D. Intrusion detection via static analysis[C]// Proc. of the IEEE Symp. on Security and Privacy. Oakland: IEEE Press, 2001: 156-168. http://www.csl.sri.com/users/ddean/papers/oakland01.pdf

[32] Sekar R, Bendre M, Bollineni P, et al. A fast Automaton-Based method for detecting anomalous program behaviors[C]// IEEE Symp. on Security and Privacy. Oakland: IEEE Press, 2001: 144-155. http://www.cc.gatech.edu/~wenke/ids-readings/automaton.pdf

[33] Feng H H, Kolesnikov O M, Fogla P, et al. Anomaly detection using call stack information[C]// Proc. of the 2003 IEEE Symp. on Security and Privacy. Oakland: IEEE Press, 2003: 62-75. http://www-unix.ecs.umass.edu/~gong/papers/ok_idpc.pdf

[34] Giffin J, Jha S, Miller B. Detecting manipulated remote call streams[C]// Proc. of the 11th USENIX Security Symp. San Francisco, 2002: 61-79. http://www.cs.wisc.edu/wisa/papers/security02/gjm02.pdf

[35] Wagner D, Soto P. Mimicry attacks on host based intrusion detection systems[C]// 9th ACM Conference on Computer and Communications Security. Washington, DC, Nov. 2002

[36] Kruegel C, Mutz D, Valeur F, et al. On the detection of anomalous system call arguments[C]// Proceeding of ESORICS. 2003

[37] Gao D, Reiter M K, Song D. Gray-box extraction of execution graphs for anomaly detectio[C]// ACM Conference on Computer and Communications Security (CCS). Washington, DC, October 2004: 318-329

[38] Giffin J T, Jha S, Miller B P. Efficient context-sensitive intrusion detection[C]// 11th Network and Distributed Systems Security Symposium. San Diego, CA, Feb. 2004

[39] Giffin J T, Dagon D, Jha S, et al. Environment-sensitive intrusion detection[M]. Recent Advances in Intrusion Detection (RAID). September 2005

[40] Feng H H, Giffin J T, Huang Y, et al. Formalizing sensitivity in static analysis for intrusion detection[C]// IEEE Symp. on Security and Privacy. IEEE Press, 2004: 194-208

[41] 李闻，戴英侠，连一峰，等. 基于混杂模型的上下文相关主机入侵检测系统[J]. 软件学报，2009，20(1)：138-151

[42] Sufatrio, Yap R H C. Improving host-based ids with argument abstraction to prevent mimicry attacks[C]// Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection. 2005

[43] Tandon G, Chan P. Learning rules from system calls arguments and sequences for anomaly detection[C]// ICDM Workshop on Data Mining for Computer Security (DMSEC). Melbourne, FL, 2003

[44] Bhatkar S, Chaturvedi A, Sekar R. Dataflow anomaly detection[C]// Proceedings of the 2006 IEEE Symposium on Security and Privacy. 2006

[45] Li Peng, Park Hyundo, Gao Debin, et al. Bridging the Gap Between Data-flow and Control-flow Analysis for Anomaly Detection[C]// Proceedings of the 2008 Annual Computer Security Applications Conference. 2008