ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.11, November 2007, pp.2841−2850
DOI: 10.1360/jos182841

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于控制流的程序行为扩展模型*

陆 炜[1,2], 曾庆凯[1,2+]

[1](南京大学 计算机软件新技术国家重点实验室),江苏 南京 210093)

[2](南京大学 计算机科学与技术系,江苏 南京 210093)

A Control Flow Based Program Behavior Extended Model

LU Wei[1,2], ZENG Qing-Kai[1,2+]

[1](State Key Laboratory for Novel Software Technology, Nanjing University), Nanjing, 210093, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

+ Corresponding author: Phn: +86-25-83686700, E-mail: zqk@nju.edu.cn; http://cs.nju.edu.cn

Lu W, Zeng QK. A control flow based program behavior extended model. *Journal of Software*, 2007,18(11): 2841−2850. http://www.jos.org.cn/1000-9825/18/2841.htm

Abstract: This paper presents a control-flow-based program behavior extended model EMPDA (extended model based on push down automaton) by adding invariance constraints to control flow model, which can describe some invariance properties while a program is running safely, and enhance the ability of intrusion detection. By distinguishing the importance of system calls according to practical applications, this paper divides the program behavior model into core model and secondary model to reduce the workload of the model and improve the learning efficiency. Experimental results show that the extended model has better performances in many aspects, such as coverage speed, false positive rate and the capability of intrusion detection.

Key words: program behavior model; anomaly detection; control flow; invariance constraint; system call

摘 要: 提出一种基于控制流的程序行为扩展模型 EMPDA(extended model based on push down automaton).对控制流模型加入不变性约束扩展,该模型能够表达程序正常运行时所应保持的不变性质约束,增强了模型的监控能力;通过以实际应用区分系统调用重要性,将模型划分为核心模型和辅助模型,以降低模型整体消耗,提高模型学习效率.实验结果表明,该扩展模型较之原模型有更好的覆盖速度、误报率以及检测能力.

关键词: 程序行为模型;异常检测;控制流;不变性约束;系统调用

中图法分类号: TP393 **文献标识码**: A

随着软件的复杂性不断提高、软件规模不断扩大,针对软件漏洞的攻击也越来越多.对系统攻击的检测与防护方法的研究日益受到重视.入侵检测通过对行为、日志、审计数据或其他信息进行分析,检测对系统的攻击[1],通常采用特征检测和异常检测方法[2].特征检测方法对异常行为建模,通过特征模型匹配来检测可疑行为;

* Supported by the National Natural Science Foundation of China under Grant Nos.60473053, 60773170 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2004AA147070, 2006AA01Z432 (国家高技术研究发展计划(863)); the Natural Science Foundation of Jiangsu Province of China under Grant No.BK2005074 (江苏省自然科学基金)

Received 2006-04-26; Accepted 2006-07-05

异常检测方法对正常行为建模,违背模型的行为被视为可疑行为[3].与特征检测相比,异常检测的优势是具有对未知攻击的检测能力.相对于网络活动、用户行为而言,程序行为更稳定、更准确[4].因此,异常检测研究近年来较为集中于程序行为的建模和监控方法.

程序行为建模有多种方法,如形式化方法[5,6]、短序列模型[7]、数据挖掘方法[8]、神经网络方法[9]、隐马尔可夫模型[10]和控制流模型[4,11−16]等等.形式化方法要求有准确的安全规格说明,通过检查主客体信息流[5]或者程序的执行流程[6]是否满足规格说明以判断入侵行为的发生,是一种理想化的方法.实际上,即使是一个不安全的程序,也不一定会对系统造成很大的伤害,除非它和底层的操作系统交互,而这些交互是可以被监控的[17].因此,许多程序行为模型都是基于系统调用进行建模.短序列模型方法是基于系统调用方法的先驱,通过一定长度的系统调用序列建立正常行为库.数据挖掘方法、神经网络方法、隐马尔可夫模型等是从系统调用集中抽取特征,生成正常行为模型或满足的规则.

控制流模型方法基于系统调用点生成程序行为模型,约束程序的实际执行流程与程序的源代码保持一致[11],能够直接反映程序的要求.静态分析方法通过分析程序代码生成控制流模型[11],动态分析方法通过动态分析程序运行时刻信息生成模型[14].而在模型中加入函数调用信息,则扩展了控制流模型[13,15].研究表明,将动态和静态分析方法相结合可以生成更准确的模型[4].但是,现有的控制流模型方法对不改变程序控制流的入侵行为不能有效监控.此外,模型学习的覆盖速度不够快,监控误报率较高.本文提出一个基于控制流扩展程序正常运行约束要求的模型EMPDA(extended model based on push down automaton),利用约束模型扩展控制流模型的监控能力.约束模型在系统调用或者函数调用点上检测程序正常运行要求的不变性质是否满足.此外,针对实际应用中系统调用对于程序安全的重要程度不同,提出对系统调用的重要性进行区分,以提高模型的表达能力和效率.

本文第 1 节介绍相关研究工作.第 2 节描述提出的模型.第 3 节讨论模型的实现问题.第 4 节是模型性能分析和实验.第 5 节是本文总结.

1 相关工作

Forrest[7]提出了基于系统调用序列的N-gram的方法.该方法预先给定一个n,正常行为是通过列举所有训练数据中出现的具有长度n的唯一的、邻近的序列获得.检测时,使用长度为n的滑动窗口,顺序地对被检测的系统调用序列进行正确性匹配.这种方法训练速度快,但却无法辨别长度大于n的系统调用序列关系.而且只能识别在训练过程中遇到的序列,缺乏一般性.

很多研究者从静态分析程序源代码、二进制代码[11−13]和动态分析程序运行时刻信息[14−16]等方向对Forrest的方法进行了扩展.Wagner[11]通过静态分析源代码产生程序行为模型,产生一个非确定有穷状态自动机(non-deterministic finite automaton,简称NDFA),其状态隐含着程序的指令指针信息,一个状态转换与一个系统调用或者空动作相联系.在检测过程中,有穷状态机接受一个系统调用,从当前状态移动到该系统调用所关联的另一个状态.如果没有相应的转换,就表示发生了异常.因为没有对函数调用作处理,所以存在不可能路径问题(impossible path problem)[15].为了解决这个问题,提出了一个复杂的下推自动机(pushdown automaton,简称PDA)模型,即抽象栈模型.因为下推自动机的复杂性和不确定性,模型对很多程序的监控效率太低.Giffin[12]利用了上述模型的思想,对二进制代码进行静态分析.方法不依赖于任何语言,而是依赖于平台.该方法要求重写可执行文件,通过在程序调用之前和之后插入空调用来减少自动机的不确定性,解决了不可能路径问题,但不适用于一般的基于主机的异常检测环境.Feng和Giffin[13]提出了一个确定的下推自动机模型VPStatic.该模型利用了函数之间的调用信息,消除模型不确定性,并且是通过静态分析二进制代码获得的.他们还指出了消除自动机的不确定性的两种方法:一种是获得额外的信息,如函数调用信息;另一种是在程序内加入额外的代码,如在函数进口处加入额外的系统调用进而获得程序内部状态.

动态分析方法的研究也在不断进步.Sekar[14]通过动态的学习程序的审计数据,捕获程序的控制流.模型自动机没有不确定性问题.每个系统调用的指令指针表示一个状态,系统调用则作为状态转换的标签.因为每个转

换都是确定的,所以效率很高,并且不会由于不确定性而无法识别某些异常.因为没有捕获程序内函数之间的调用关系,所以存在不可能路径问题.Feng[15]扩展了文献[14]中的方法,在两个系统调用之间建立VtPath.VtPath是指两次连续的系统调用发生时调用栈内的不同,可以表示两个系统调用之间的函数调用关系.该方法使用一个哈希表记录以字符串的形式表示的所有出现的VtPath,实验表明,它具有更高的探测能力,在一定程度上解决了不可能路径问题.但是因为仅仅记录出现的VtPath,因而不具有一般性,而且不够简洁.Gao[16]的方法提取程序执行时的函数调用信息,将一个系统调用发生时的调用栈序列称为观察(observation),将程序一次执行产生的观察序列称为执行(execution).再通过观察和执行来构造执行图(execution graph).这个执行图是被归纳定义的,归纳能够获得可能存在的其他执行路径.该方法比Feng的方法具有一般性,并且证明了执行图对给定的训练数据与程序控制流具有最大的一致性.

Liu[4]提出的HPDA(hybrid push down automata)模型也利用了函数调用信息.该模型扩展了Wagner的PDA模型,将函数调用关系和系统调用作为输入构建模型,可归纳出可能存在的其他执行路径.该模型可以通过静态分析程序代码、动态分析程序运行信息或者动态和静态结合的方法生成.实验表明,用动态生成模型作为静态生成模型的补充,生成的模型比单独使用静态方法具有更高的探测能力,比单独使用动态方法具有更低的误报率.动态和静态方法的目标都是通过控制流适当地对程序行为建模,在保证准确性的同时兼顾模型的一般化程度,使得通过更少量的信息获得更全面的行为模型.本文也是基于这样的目标,对HPDA模型进行扩展,提出了改进模型.

2 扩展约束行为的模型 EMPDA

2.1 模型描述

本文提出了扩展约束行为的模型EMPDA.根据Gao[16]的证明,利用函数调用信息,并且归纳出所有可能执行路径的模型与程序控制流有很大的一致性,因此,EMPDA对HPDA进行扩展.主要在以下两个方面:

1. 控制流模型能够监控缓冲区溢出[18,19]、格式化字符串[19,20]等改变程序路径的攻击,然而对竞争条件[21]等不改变程序执行路径的攻击方法无能为力.为了扩展模型的监控能力,通过让行为模型学习恰当设定的约束模型,使得模型能够检测上述攻击,扩展模型监控能力.
2. 考虑到实际应用中不同系统调用对系统安全的影响程度不同[22]以及不同环境中所需监控的侧重点不同,将模型分为核心模型和辅助模型.核心模型保证精确性,辅助模型提高泛化程度,用辅助模型一定的监控精确性来换取整体模型的覆盖速度和误报率.

2.1.1 核心模型

一个核心模型 M 是一个五元组:(S,Σ,T,s,A).其中:

S 是有穷状态集,S 的元素是一个两元组:$(id,check\_action)$.id 是该状态的标识,$check\_action$ 表示到达该状态时模型需要进行的监控动作.

Σ是有穷输入集,Σ的元素为$(Y\times Addr)$.每个状态转换的输入包含两部分,Y 和 $Addr$,结合 Y 和 $Addr$ 能够唯一地标识程序的一个语句.输入调用 Y 可以是进入某一函数,退出某一函数或者是一个核心系统调用信息,$Y=\{Entry,Exit,SC\}$,$SC=Scall\_ID$,即系统调用标识.$Addr$ 是 Y 发生时对应的可执行文件或者共享库的标识 img_id 以及 Y 在该模块中的相对位置 $addr$,$Addr=(addr,img\_id)$.

$s\in S$,是开始状态.

A 是终止状态集.

$T:S\times\Sigma\rightarrow S$ 是状态之间的转换规则集.

核心模型相对于 HPDA 模型对状态作了新定义,每一个状态实质上是指监控模型接受到一个特定输入后进行的一系列检测动作的集合.核心监控模型的监控能力由两部分组成:一部分是控制流所监控的程序流程;另一部分是由状态集合监控的约束不变性质.

2.1.2 辅助模型

辅助模型 L 是若干序列组成的集合.

每个序列是一个三元组:(Σi,Li,Σi+1).Σi 和Σi+1 是发生在 Li 前后的重要系统调用,也可以是表示模型开始和结束的调用.Li 是非重要系统调用集合,其中元素之间无先后顺序且不重复.

2.1.3 约束模型

约束模型 CM 由若干行为组 $ActG$ 组成,每一个行为组 $ActG$ 表示程序必须满足的某一个约束要求,是应用要求的不变性或者程序安全运行要求的不变性.每个 $ActG$ 由存在逻辑关系的若干行为-响应对组成.每一个行为-响应对表示当给定行为发生时,保证程序安全运行所需要进行的检测动作.

约束模型定义如下:

CM={$ActG$},为包含若干行为组 $ActG$ 的约束模型.

$ActG$={($ActP$,$ResP$)},为一组 $ActP$ 及其相应 $ResP$ 的集合,表示不变性约束的逻辑关系.

$ActP\in\Sigma$,表示模型中特定的系统调用或者函数调用点,定义了相应的 $ResP$ 将要发生的地方.

$ResP$:定义了在 $ActP$ 处,约束模型具体的检测动作.

可根据程序安全运行要求的不变性构造约束模型.例如,针对文件系统的文件名绑定竞争条件漏洞,Lhee[21]提出监控任一文件的有效检查操作和有效使用操作处的一致性,模型的组成部分如图 1 所示.用一个文件对照表记录访问的文件以及其唯一的标识(*inode number*),对有效的文件检查操作(*echeck*),要求对文件一致性对照表内容进行更新(*uptable*);对有效的文件使用操作(*euse*),要求对访问的文件和文件对照表的内容进行一致性检查(*chetable*).即有行为组$ActG$={(*echeck*,*uptable*),(*euse*,*chetable*)},表示对任一文件的有效检查操作和使用操作需要有一致性约束.

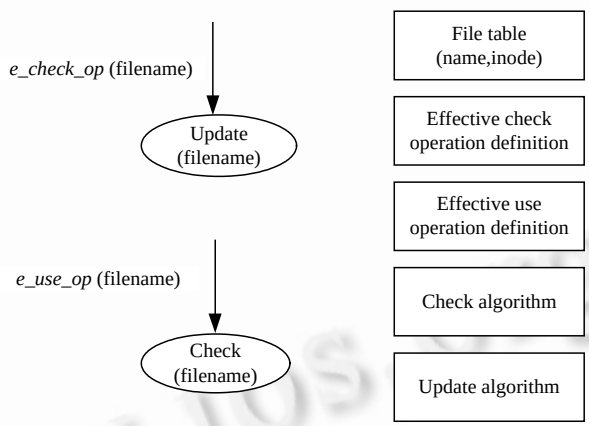

Fig.1 File-Based race conditions detection model

图 1 文件系统竞争条件检测模型

也可根据应用要求的不变性构造约束模型.例如,针对应用中的核心文件,在程序的进、出口处要检查核心文件的完整性,保证应用系统的安全性.这样的应用要求不变性也很容易用约束模型来表示.

2.1.4 模型表示

EMPDA 模型由核心模型和辅助模型两部分组成.核心模型是记录重要系统调用的有穷状态机,包含了约束模型的监控能力.辅助模型是核心模型的补充部分,针对非重要系统调用的模型.如图 2 所示的程序,其生成的行为模型包含重要调用形成的核心模型以及非重要调用形成的辅助模型.在核心模型中,creat,chown,chmod 和 rename 是文件名绑定竞争条件约束模型关注的系统调用,其中,creat 调用是有效检查操作,chown,chmod 和 rename 是有效使用操作,存在文件名绑定竞争条件的漏洞.需要约束程序对文件访问的一致性,在 creat 调用之后的状态中对文件对照表进行更新操作,在 chown,chmod 和 rename 调用之后的状态进行检查操作.

```
1   char filename[20]="/tmp/data";
    int pre_create(int index,time_t * tim){
2       if(index){
3           ...;
4           ...;
5       }
6       time(tim);
7       return creat(filename,O_RDWR);
    }
    void create_file(char file[50]){
8       uid_t uid = getuid();
9       gid_t gid = getgid();
10      time_t tim;
11      int handle;
12      if(uid == 0){
13          handle = pre_create(0,&tim);
14          char filepath[100] = "/root/";
15          chown(filename,0,0);
16          chmod(filename,...);
17          rename(filename,filepath);
18      }else{
19          handle = pre_create(1,&tim);
20          char filepath[100] = "/home/";
21          chown(filename,uid,gid);
22          chmod(filename,...);
23          rename(filename,filepath);
24      }
25      gethostname();
26      read(handle,...);
27      close(handle);
    }
```

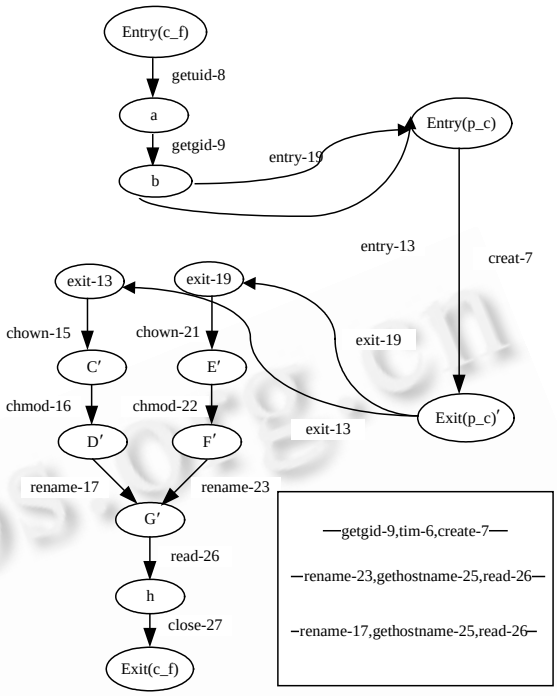

Fig.2 Example code and the corresponding behavior model

图 2 实例代码及其对应的行为模型

2.2 两阶段学习

EMPDA 的学习过程分为两个阶段:控制流的学习和约束模型的学习.控制流学习的目标是生成程序行为模型的核心模型框架和辅助模型.约束模型学习的目标是完成核心模型,在核心模型的状态中设置检测点.

2.2.1 控制流学习

程序控制流可以通过分析程序源代码或者分析程序运行时刻信息获得.由于现代程序设计的很多机制,比如动态连接、异常处理,静态分析程序控制流比较困难[23],并且对代码的分析依赖于语言或者平台;分析程序运行时刻信息方法通过给定一个测试案例集,在程序系统调用时分析程序控制流,获得的程序行为模型的完备性依赖于测试案例集的完备性.

本文模型学习采取分析程序运行时刻方法,不考虑测试案例集完备性问题.在程序内重要系统调用发生时,使用当前的调用栈和上一次调用栈的不同序列产生学习输入.

使用与Feng[15]类似的符号,CS_{last}表示上一次的系统调用栈信息,$CS_{current}$表示当前系统调用栈信息:

$$CS_{last}=(a_0,a_1,\ldots,a_l,a_{l+1},\ldots,a_m),\ CS_{current}=(b_0,b_1,\ldots,b_l,b_{l+1},\ldots,b_n),$$

其中,$a_0=b_0,a_1=b_1,a_l=b_l,a_{l+1}\neq b_{l+1}$.$a_m$和$b_n$是这两次系统调用的返回地址.

SyscallID 表示当前的重要系统调用标识.这样,核心行为模型的学习序列就是

$$(Exit-a_{m-1},\ldots,Exit-a_{l+1},Entry-b_{l+1},\ldots,Entry-b_{n-1},SyscallID-b_n).$$

设核心模型当前状态为 *S*,待学习辅助模型元素为 *l*,当前学习输入为 *ai*,学习算法如下:

Step 1. 若 *ai* 属于重要系统调用,则执行 Step 3.

Step 2. 将 *ai* 添加到 *l* 的 *Li* 中,结束对 *ai* 的学习,学习下一个输入.

Step 3. 若 *l* 的 *Li* 为空,则执行 Step 7.

Step 4. 将 *l* 的Σi+1 设置为 *ai*,在辅助模型中查找$\Sigma I,\Sigma i$+1 和 *l* 中相等的元素,若存在,则执行 Step 6.

Step 5. 添加 *l* 到辅助模型中,执行 Step 7.

Step 6. 将 *l* 的 *Li* 内非重要系统调用添加到 Step 4 元素的 *Li* 中.

Step 7. 清空 *l*,将 *l* 的*Σi* 设置为 *ai*;判断核心模型中当前状态 *S* 是否存在等于 *ai* 的出边,若不存在,则执行 Step 9.

Step 8. 将核心模型当前状态移动到 *ai* 所指向的状态,结束对 *ai* 的学习,学习下一个输入.

Step 9. 判断核心模型中是否存在等于 *ai* 的边,若存在,则执行 Step 11.

Step 10. 在核心模型中新建状态 *S′*,连接 *S* 和 *S′*,并标记其转换为 *ai*,将核心模型的当前状态移动到 *S′*,结束对 *ai* 的学习,学习下一个输入.

Step 11. 判断当前状态 *S* 是否存在出边,若不存在,则执行 Step 13;

Step 12. 连接 *S* 和 *ai* 指向的状态,并标记为 *ai*,将核心模型的当前状态移动到 *ai* 所指向的状态,结束对 *ai* 的学习,学习下一个输入;

Step 13. 将 *ai* 的出发点与 *S* 合并,将核心模型的当前状态移动到 *ai* 所指向的状态,结束对 *ai* 的学习,学习下一个输入.

2.2.2 约束模型学习

对约束模型的学习可以单独作为模型学习的第 2 个阶段,也可以在控制流学习过程中同时进行.在学习过程中,扫描核心模型中的每个输入调用*Σ*及其目标状态 *S*,根据*Σ*在约束模型中查找是否需要设置相应的监控动作标识 *check_action*,将需要的 *check_action* 添加到该输入调用的目标状态 *S* 中.

2.3 运行时刻监控

EMPDA 中的异常包含以下几种:

1. 核心异常是指核心模型中的检测异常,即与主要程序结构不相符合的程序执行.分为系统调用异常与函数调用异常.
2. 辅助异常是指辅助模型中的检测异常,即非重要系统调用集的匹配异常.
3. 约束模型异常是指核心模型状态的监控动作检测出的异常,即不满足约束模型要求的异常.
4. 核心异常体现控制流约束,是检测能力的基础;辅助异常体现低精度监控能力,是降低误报率、提高模型适用性的关键;约束异常体现约束模型能力,是增强监控能力的因素.3 种异常从不同的角度和精度互相补充,有利于对程序行为的监控.

设核心模型当前状态为 *S*,待监控的辅助模型元素为 *l*,当前监控输入为 *ai*,监控算法如下:

Step 1. 若 *ai* 为重要系统调用,则执行 Step 3.

Step 2. 将 *ai* 添加到 *l* 的 *Li* 中,结束对 *ai* 的监控,监控下一个输入.

Step 3. 判断核心模型当前状态 *S* 是否存在等于 *ai* 的出边,若存在,则执行 Step 6.

Step 4. 产生核心异常,判断核心模型中是否存在等于 *ai* 的边,若存在,则将核心模型当前状态移动到 *ai* 的目标状态.

Step 5. 清空 *l*,将 *l* 的*Σi* 设置为 *ai*,结束对 *ai* 的监控,监控下一个输入.

Step 6. 将核心模型当前状态移动到 *ai* 的目标状态,若 *l* 中 *Li* 为空,则执行 Step 9.

Step 7. 将 *l* 的*Σi*+1 设置为 *ai*,在辅助模型中查找*ΣI*,*Σi*+1 和 *l* 中相等的元素,若不存在,则产生辅助异常,执行 Step 9.

Step 8. 判断 *l* 中 *Li* 和 Step 7 的元素中 *Li* 的匹配程度,若匹配不满足,则产生辅助异常.

Step 9. 清空 *l*,将 *l* 的*Σi* 设置为 *ai*.

Step 10. 根据 *ai* 目标状态中 *check_action* 进行相应的监控动作,若约束要求不满足,则产生约束模型异常.

Step 11. 结束对 *ai* 的监控,监控下一个输入.

不同的异常有不同的权重,不同权重的异常也有不同的响应方式,如异常累计、发出警报、暂停程序等等.辅助异常相对于核心异常和约束模型而言,异常权值小,核心异常和约束模型异常的比重以及对不同权重异常的响应方式则根据实际应用来分析.本文的实验对辅助异常设定阈值,对核心异常、约束模型异常以及超过阈值的辅助异常发出警报.

3 模型实现问题

由于存在着各种编程手段,模型在实现阶段有一些问题需要进行相应的处理,如果处理不当会对模型的正确性有很大影响.这些问题包括动态链接库、信号处理函数、调用栈分析以及系统调用重要性分析等.其中,前两个问题在一些文章[4,14,15]中已有分析,并提出了较好的解决办法.调用栈和系统调用重要性的分析还需要结合实际情况来进行.

3.1 动态链接库

动态链接库可能在不同的时候被加载到不同的内存区域,导致系统调用发生时的指令指针信息不能标识该系统调用.因此,本文用发生系统调用的执行模块标识及其在模块内的相对地址来唯一地标识该系统调用.在Linux中,当系统调用发生时,根据其发生的虚拟地址找到该地址所处内存块映射的文件,用该文件的inode标识这个可执行模块,虚拟地址相对于内存块首地址的数值标识了该系统调用在模块内的相对地址.这样,两个数值唯一确定了该系统调用.

3.2 信号处理函数

信号处理函数只有在特定的信号产生时才被执行,执行完以后,继续执行信号处理前被挂起的程序.信号处理函数不能被当作一般的程序函数来对待,因为它可能在程序执行的任何时刻被调用,它不是程序顺序执行流程的一部分.处理这类函数的办法是在模型生成时,将其作为一个独立的模型进行构造;在监控过程中,当信号处理函数的第 1 个系统调用发生时,首先保存正在使用的监控模型的当前状态,然后跳转到此信号处理模型进行监控,在离开此模型后,恢复原先的监控模型状态继续执行.在 Linux 中,当在调用栈中发现被插入的新Sigreturn系统调用时,则表示进入了信号处理函数.当执行到Sigreturn系统调用时,表示信号处理函数运行结束返回.因此,它可以用来解决信号处理函数问题.

3.3 调用栈分析

对调用栈分析的目的是获得函数调用关系.分析的方法有两种[24]:

1) 在满足一定编译条件下,可以通过栈的内部框架链获得函数调用关系,这种方法快速而准确,但却不能保证栈框架链的完整性,因为有些时候为了提高程序执行效率,程序不提供栈框架链.
2) 通过直接分析栈的每32位内容,判断其是否为一个返回地址,进而获得函数调用关系.

3.4 系统调用重要性

系统调用是操作系统提供给用户的一系列功能强大的服务函数,也是程序与操作系统内核交互的方法.系统调用按照功能逻辑大致可分为“进程控制”、“文件系统控制”、“系统控制”、“内存管理”、“网络管理”、“Socket控制”、“用户管理”、“进程间通信”.程序行为建模的目的是监控程序执行,监控其对系统的机密性、完整性和可用性等安全属性的影响.因此,分析系统调用重要性以其对系统安全性的影响为依据.

对于“进程控制”、“文件系统控制”、“系统控制”、“网络管理”、“用户管理”这 5 个功能类的系统调用,除了单纯提供功能的系统调用以外,还可以按信息读取、信息修改以及行为控制等类型进行划分,其中“文件系统控制”、“网络管理”和“用户管理”不包括行为控制类型.上述类型分别与系统机密性、完整性以及可用性关系密切.“内存管理”、“Socket控制”和“进程间通信”等功能类的系统调用是操作系统提供的服务机制,其中部分与机密性、完整性和可用性均相关,部分为单纯提供的功能.因此,系统调用从影响安全的角度可以分为影响机密性类、影响完整性类、影响可用性类、综合影响类和单纯功能类.

一个系统调用的重要性一般需要考虑以下几个决定因素:

(1) 环境要求的程序监控侧重点,需根据实际应用分析对机密性、完整性和可用性的侧重.
(2) 系统调用所属类型与侧重安全属性的关联程度.
(3) 系统调用在该类型中的重要程度.约束模型是更重要的决定因素,其中要求检测的系统调用均作为重要系统调用来处理.

4 性能分析及实验

4.1 模型性能分析

约束模型扩展了控制流模型的异常监控能力.约束模型的关注点,即系统调用以及函数调用点,是程序行为模型的重要结构,因此,约束模型在描述程序约束不变性上具有一定的一般性.一方面,它能够作为控制流模型与其他特定缺陷模型之间的桥梁,通过分析缺陷模型体现的不变性安全约束,将其监控能力添加到控制流模型中;另一方面,应用要求的不变性质可以通过约束模型来表示,使行为模型更适合应用.

利用系统调用的重要性对模型进行分解,非重要系统调用的细节被忽略.对使用非重要系统调用漏洞的程序会有些漏报,但如果攻击者仅利用非重要系统调用,对系统及应用的危害有限,而且当攻击者使用重要系统调用或者不符合要求的非重要系统调用时,将被核心模型或辅助模型所检测.误报率过高是影响异常检测实际应用的主要问题,降低对非重要系统调用的监控能够换取模型的整体适用性.同时,模型分解可提高模型覆盖速度,使生成模型需要的数据集和模型生成的时间减少.

4.2 实验结果

原型系统主要包括两个组成部分:一个可装载内核模块和一个用户态的学习监控模块.内核模块用来截获进程运行时刻的信息,包括系统调用相关信息以及相应的函数调用信息;用户态的学习监控模块可对内核模块所采集的数据进行学习和分析,对异常作出相应的处理.此系统在 1.6G P4 CPU,512MB 内存的机器上,在 VMware5.5 上安装的内核版本 2.4.18 的 Red Hat8.0 系统中实现.

本文将 EMPDA 模型与动态生成的 HPDA 模型进行测试和比较.测试的主要指标为覆盖速度、误报率和检测能力等,并对 EMPDA 模型的监控时间消耗进行了测量.主要的测试对象为 ftp 服务器 vsftpd,通过一些脚本模拟用户的访问行为生成测试数据.

由于 ftp 服务器与系统安全性的关系非常密切,在 EMPDA 的模型学习中,按照完整性、可用性、机密性顺序对安全属性进行重要性排列.将系统调用划分为 3 个重要性层次,能够直接修改系统中的文件、进程、用户属性等,重要性划分为 1 级;读取文件系统内容,改变进程和系统行为等,重要性划分为 2 级;读取系统和进程信息,部分使用系统功能的调用等,重要性划分为 3 级.在实验中,将重要性为 3 级的系统调用作为非重要系统调用来处理.EMPDA 模型中还学习了针对文件名绑定竞争条件生成的约束模型.

4.2.1 覆盖速度

覆盖速度反映了模型在给定的训练数据集上的生成速度,速度越快,表示生成模型所需要的时间越短,所需要的训练数据集越小.所比较的两个模型都将函数之间的调用关系作为模型学习输入,图 3 中的 x 轴表示的是模型接受的输入数,包括系统调用和函数调用关系信息;y 轴表示导致模型学习的模型输入数.结果显示,EMPDA 模型比 HPDA 模型具有更快的覆盖速度.这是由于对系统调用分类,HPDA 模型中的非重要系统调用部分被简化处理了.

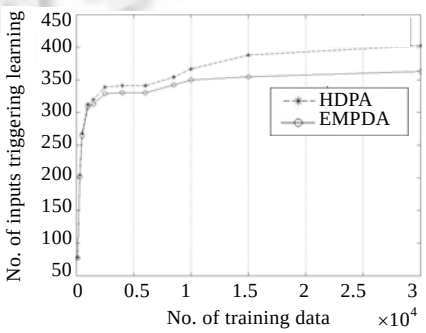

Fig.3 Coverage speed of vsftpd

图 3 vsftpd 模型覆盖速度

4.2.2 误报率

为了反映实际应用中学习阶段和测试阶段必然存在的差异[14],通过选择一些训练中出现过的正常行为数据以及一小部分在训练中未出现过的正常行为数据对模型的误报率进行测试.训练数据集也是以模型输入为单位.测试结果如图 4 所示.结果表明,EMPDA在同样学习数据的情况下能获得更低的误报率.然而由于vsftpd中非重要系统调用所占的比重比较小,EMPDA模型的优势并不十分明显,只有在使用大量的训练数据以后,模型的误报率才能降低到足够的程度.

4.2.3 时间耗费

时间消耗的测试利用了 gzip,在不监控和监控情况下分别对 gzip 压缩固定大小的文件时间进行了比较.图 5 中的结果是利用 time 命令获得压缩大小为 8M 的文件所需要的平均时间,共执行 10 次.

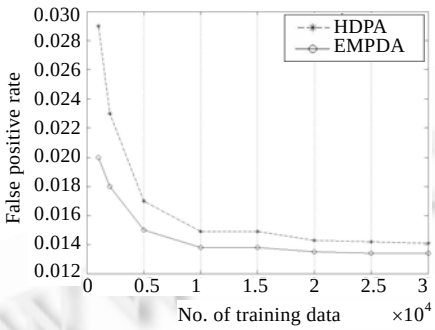

Fig.4 False positive rate of vsftpd

图 4 vsftpd 模型误报率

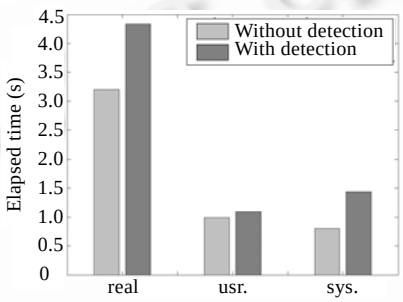

Fig.5 System overload

图 5 系统监控时间消耗

4.2.4 检测能力

针对 wuftpd 2.6.0 版本的远程溢出漏洞,分别采用 EMPDA 模型和动态生成的 HPDA 模型进行了测试,结果是两个模型都能成功地检测并报警.同时,对具有文件名绑定竞争条件漏洞的程序也进行了测试,试图利用该程序修改越权文件并对其进行了检测.测试结果是,EMPDA 模型成功地检测到了异常,HPDA 模型无法检测到异常.

5 总 结

本文提出了程序控制流模型的改进方法,并通过实验证明了其有效性和可行性.通过恰当的设置,约束模型能够表达程序正常运行的不变性质约束.由于加入不变性约束扩展,控制流模型的检测能力得到增强;而对系统调用重要性进行了区分,模型获得了更好的覆盖速度和误报率.有时可能因为程序中包含的非重要系统调用不多,导致区分系统调用重要性模型的效率和不区分重要性时相差不大,但其总体上对模型提高效率是有益的.今后,我们将着重于对更多实际应用和程序缺陷进行研究,获得更明确的、具有一般性的约束模型.同时,进一步研究在实际应用中系统调用重要性的区分规则和方法,以完善模型.

References:

[1] Denning DE. An intrusion-detection model. IEEE Trans. on Software Engineering, 1987,13(2):222−232.

[2] Sherif JS, Dearmond TG. Intrusion detection: Systems and models. In: Proc. of the IEEE Int'l Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises. Pittsburgh: IEEE Computer Society, 2002. 115−136.

[3] Verwoerd T, Hunt R. Intrusion detection techniques and approaches. Computer Communications, 2002,25(15):1356−1365.

[4] Liu Z, Bridges SM, Vaughn RB. Combining static analysis and dynamic learning to build accurate intrusion detection models. In: Proc. of the 3rd IEEE Int'l Workshop on Information Assurance. College Park: IEEE Computer Society, 2005. 164−177.

[5] Zimmermann J, Me L, Bidan C. An improved reference flow control model for policy-based intrusion detection. In: Snekkenes E, Gollmann D, eds. Proc. of the 8th European Symp. on Research in Computer Security. Gjøvik: Springer-Verlag, 2003. 291−308.

[6] Fink G, Levitt K. Property-Based testing of privileged programs. In: Proc. of the 10th Annual Computer Security Applications Conf. Orlando: IEEE Computer Society, 1994. 154−163.

[7] Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA. A sense of self for UNIX processes. In: Proc. of the IEEE Symp. on Security and Privacy. Oakland: IEEE Computer Society, 1996. 120−128.

[8] Lee W, Stolfo SJ. Data mining approaches for intrusion detection. In: Proc. of the 7th USENIX Security Symp. San Antonio: USENIX, 1998. 79−94.

[9] Ghosh AK, Schwartzbard A, Schatz M. Learning program behavior profiles for intrusion detection. In: Proc. of the Workshop on Intrusion Detection and Network Monitoring. Santa Clara: USENIX, 1999. 51−62.

[10] Warrender C, Forrest S, Pearlmutter BA. Detecting intrusions using system calls: Alternative data models. In: Proc. of the IEEE Symp. on Security and Privacy. Los Alamitos: IEEE Computer Society, 1999. 133−145.

[11] Wagner D, Dean D. Intrusion detection via static analysis. In: Proc. of the IEEE Symp. on Security and Privacy. Oakland: IEEE Computer Society, 2001. 156−169.

[12] Giffin JT, Jha S, Miller BP. Detecting manipulated remote call streams. In: Boneh D, ed. Proc. of the 11th USENIX Security Symp. San Francisco: USENIX, 2002. 61−79.

[13] Feng HH, Giffin JT, Huang Y, Jha S, Lee W, Miller BP. Formalizing sensitivity in static analysis for intrusion detection. In: Proc. of the IEEE Symp. on Security and Privacy. Berkeley: IEEE Computer Society, 2004. 194−210.

[14] Sekar R, Bendre M, Dhurjati D, Bollineni P. A fast automaton-based method for detecting anomalous program behaviors. In: Proc. of the IEEE Symp. on Security and Privacy. Oakland: IEEE Computer Society, 2001. 144−155.

[15] Feng HH, Kolesnikov OM, Fogla P, Lee W, Gong W. Anomaly detection using call stack information. In: Proc. of the IEEE Symp. on Security and Privacy. Berkeley: IEEE Computer Society, 2003. 62−75.

[16] Gao D, Reiter MK, Song D. Gray-Box extraction of execution graphs for anomaly detection. In: Atluri V, Pfitzmann B, McDaniel PD, eds. Proc. of the 11th ACM Conf. on Computer and Communications Security. Washingtion: ACM, 2004. 318−329.

[17] Goldberg I, Wagner D, Thomas R, Brewer EA. A secure environment for untrusted helper applications: Confining the wily hacker. In: Proc. of the 6th USENIX Security Symp. San Jose: USENIX, 1996. 1−13.

[18] Haugh E, Bishop M. Testing C Programs for buffer overflow vulnerabilities. In: Proc. of the Network and Distributed System Security Symp. San Diego: Internet Society, 2003.

[19] Jiwnani K, Zelkowitz MV. Maintaining software with a security perspective. In: Proc. of the Int'l Conf. on Software Maintenance. Montreal: IEEE Computer Society, 2002. 194−203.

[20] Shankar U, Talwar K, Foster JS, Wagner D. Detecting format string vulnerabilities with type qualifiers. In: Proc. of the 10th USENIX Security Symp. Washington: USENIX, 2001. 201−218.

[21] Lhee K, Chapin SJ. Detection of file-based race condition. Int'l Journal of information Security, 2005,4(1-2):105−119.

[22] Xu M, Chen C, Ying J. Anomaly detection based on system call classification. Journal of Software, 2004,15(3):391−403 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/391.htm

[23] Gao D, Reiter MK, Song DX. On gray-box program tracking for anomaly detection. In: Proc. of the 13th USENIX Security Symp. San Diego: USENIX, 2004. 103−118.

[24] Stack tracing methods. http://help.madshi.net/TraceMethods.htm

附中文参考文献:

[22] 徐明,陈纯,应晶.基于系统调用分类的异常检测.软件学报,2004,15(3):391−403. http://www.jos.org.cn/1000-9825/15/391.htm

陆炜(1981－),男,江苏镇江人,硕士生,主要研究领域为信息安全.

曾庆凯(1963－),男,博士,教授,博士生导师,主要研究领域为信息安全,分布计算.