

基于模型检测的程序恶意行为识别方法

张一弛, 庞建民, 范学斌, 姚鑫磊

(国家数字交换系统工程技术研究中心, 郑州 450002)

摘 要: 利用恶意代码所具有的共同或相似的行为特征, 提出一种基于模型检测技术的程序恶意行为识别方法。通过对二进制可执行文件进行反汇编, 构建程序控制流图, 使用 Kripke 结构对程序建模, 利用线性时序逻辑描述典型的恶意行为, 采用模型检测器识别程序是否具有恶意行为, 并在程序控制流图上对该恶意行为进行标注。实验结果表明, 与常用的杀毒软件相比, 该方法能更有效地发现程序中的恶意行为。

关键词: 模型检测; 恶意行为; 线性时序逻辑; 控制流图; 反汇编; Kripke 结构

Program Malicious Behavior Recognizing Method Based on Model Checking

ZHANG Yi-chi, PANG Jian-min, FAN Xue-bin, YAO Xin-lei

(National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450002, China)

【Abstract】 By using the same or similar behavior characteristics of malicious codes, this paper proposes a method based on model checking technology to recognize malicious behaviors in binary files. It extracts Control Flow Graph(CFG) from disassembled binary executable and builds program model with Kripke structure, then produces Linear Temporal Logic formula to describe malware specification. The model checker recognizes malicious behavior and denotes detected behavior in the CFG. Experimental result shows that compared with common antivirus software, the proposed method is more effectively in recognizing malicious behaviors.

【Key words】 model checking; malicious behavior; Linear Temporal Logic(LTL); Control Flow Graph(CFG); disassemble; Kripke structure

DOI: 10.3969/j.issn.1000-3428.2012.18.029

1 概述

近年来, 由恶意代码引发的信息安全事件造成了巨大的经济损失和无法估计的社会影响。安全分析人员和软件用户迫切希望知道二进制程序是否含有恶意行为。但是传统的反病毒软件仍然依靠特征码匹配的方法进行检测。这些产品只关注程序在二进制层面的特征, 无法识别程序中含有的恶意行为, 因此, 不能检测出新的病毒和病毒变种。

文献[1]提出一种利用程序的控制流图(Control Flow Graph, CFG)来识别程序中可疑行为的方法。该方法先消除 CFG 中部分无用信息, 形成原 CFG 的子图, 在其子图上搜索事先定义的系统调用, 从而完成恶意行为的识别。由于该方法只处理函数调用信息, 因此识别的行为必须是函数调用行为, 对其他行为没有识别能力。

文献[2]提出了一种基于指令语义的恶意软件检测方法。该方法针对恶意代码常用的混淆手段, 力图在语义层次消除代码混淆、变形给行为识别带来的影响, 通过定义语义模板, 完成恶意代码的检测工作。由于混淆、变形手

段的多样性, 很多方法可以改变原有程序的语义, 却实现相同的行为。因此, 该方法具有一定的局限性。

文献[3]也是针对变形、混淆的恶意程序, 提出了一种基于自动机的恶意代码方法。该方法认为变形、混淆是原有恶意代码经过添加无用的行为实现的, 能够有效识别垃圾数据插入和代码位置置换等常用混淆方法。由于其针对指令层进行分析, 分析粒度较小, 因此会造成检测效率低等问题。

本文提出一种基于模型检测的程序恶意行为识别方法, 首先通过反编译得到二进制程序的汇编代码并构建程序的控制流图, 使用 Kripke 结构描述程序模型; 之后使用线性时序逻辑(Linear Temporal Logic, LTL)公式在高层次上描述程序的恶意行为, 最后进行状态空间搜索, 完成程序中恶意行为的识别。

2 模型检测技术

软件安全性分析方法与技术逐渐发展, 主要有程序切

基金项目: 国家“863”计划基金资助项目(2006AA01Z408, 2009AA01Z434); 河南省重大科技攻关计划基金资助项目(092101210500)

作者简介: 张一弛(1983—), 男, 博士研究生, 主研方向: 信息安全, 逆向工程; 庞建民, 教授、博士生导师; 范学斌、姚鑫磊, 硕士研究生

收稿日期: 2011-09-08 **修回日期:** 2011-12-14 **E-mail:** each5595@sina.com

片技术、动态污点传播方法和模型检测方法。近些年来,模型检测被应用于计算机硬件、通信协议、控制系统、安全认证协议等方面的分析与验证中,取得了令人瞩目的成功。但模型检测应用于在软件安全分析方面仍处于起步阶段,如何将模型检测技术应用于软件的安全性分析已经成为各大厂商和科研院校研究的热点^[4-5]。

模型检测的研究始于20世纪80年代初。之后文献[6]提出了用于描述并发系统性质的CTL逻辑,给出了检测有穷状态系统是否满足给定CTL公式的算法,并设计了一个原型工具。模型检测是近20年来最成功的形式化自动验证技术之一,因自动化程度高、效率高等优点而被广泛应用于并发系统的分析与验证中。与定理证明等其他形式化验证方法相比,模型检测的优点主要在于:当断言被违反时,模型检测方法能够给出反例,以解释断言被违反的原因。

模型检测过程如图1所示。首先将系统抽象为状态转移图(系统模型),然后使用形式化语言(如时序逻辑表达式)来描述系统属性,最后将系统模型与系统规范输入模型检测器。模型检测器通过穷举搜索判断系统模型是否满足系统规范,如果不满足,则输出反例。

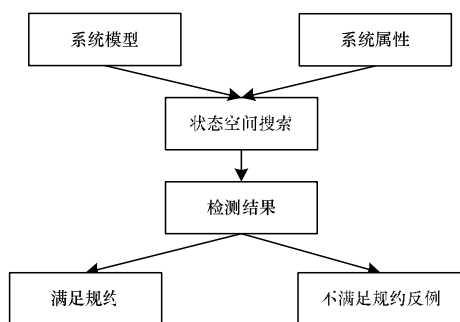


图1 模型检测的流程

将模型检测应用于程序行为识别的优势主要体现在以下3个方面:

- (1)模型检测方法能够对程序中的所有路径进行检测,不依赖程序的输入。
- (2)模型检测器自动执行,对人工操作依赖较少,检测周期短,能够及时对新型病毒做出反应。
- (3)在检测时不仅能够判断是否满足规约,在不满足时还可以给出反例,方便对恶意行为的定位与分析。

3 程序恶意行为识别

本文使用模型检测方法识别二进制程序中的行为,首先抽取二进制程序的模型,之后对恶意行为进行形式化描述,最后对程序进行形式化验证。系统框架如图2所示。具体工作原理如下:

- (1)二进制可执行文件送入模型抽取模块,判断文件是否加壳及其加壳类型,如果加壳,则使用专用的脱壳工具脱壳。
- (2)对完整的可执行程序进行反汇编,得到对应的机器代码,根据指令构建程序的控制流图。

(3)使用Kripke结构对构建的控制流图进行描述,抽取程序模型。

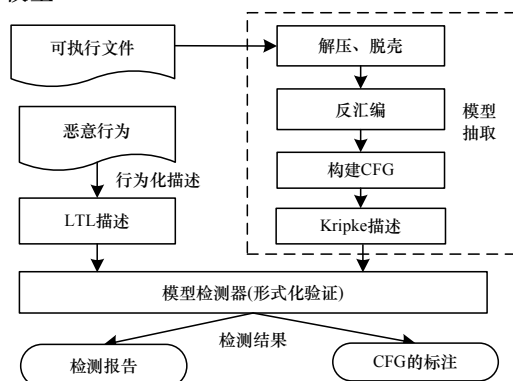


图2 基于模型检测的行为识别框架

在行为规约部分使用LTL公式描述定义的程序恶意行为,得到用于检测的恶意行为属性。模型检测器的输入分别是待检测程序对应的Kripke结构和用于恶意性分析的LTL公式,输出是程序恶意行为分析报告及标注了恶意行为存在位置的CFG。

3.1 程序模型抽取

3.1.1 反汇编

对二进制可执行程序进行分析,首先要对其进行反汇编,即将二进制可执行序列转化为对应平台的机器指令以了解程序功能。出于知识产权保护的需要,可执行程序保护方法层出不穷,其中,壳的使用最为广泛,类型也非常多,例如UPX和FSG。因此,在进行反汇编之前需要对程序的加壳情况进行判断,对于特定种类的壳使用相应的脱壳工具进行处理,得到原始的二进制代码。

RTL^[7]是一种描述基于寄存器指令中信息转换的中间语言,已经被用作很多工具的中间表示,例如链接时优化器OM、GNU编译器GCC以及编辑库EEL。因为RTL操作简单并且具有良好的平台无关性,所以本文选用它作为系统中间表示语言。

3.1.2 Kripke表示

为了保证行为识别的可靠性,需要对程序的状态迁移关系进行描述,本文选用Kripke结构对程序的控制流图进行描述。将待检测程序的控制流图表示为一组Kripke结构的集合。每个Kripke结构表示一条机器指令对应的若干条RTL语句,它由3个命题组成:(1)该汇编指令所在的位置Loc();(2)该指令包含的RTL语句Sta();(3)该指令的类型Typ()。图3是病毒重定位代码片段,图4是重定位代码对应的Kripke结构。

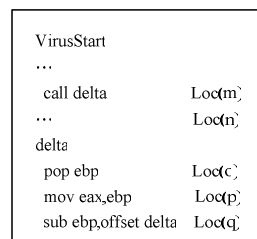


图3 病毒重定位代码片段

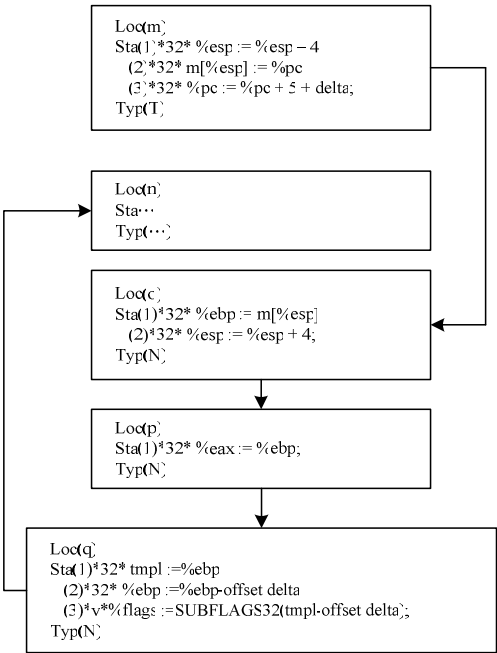


图 4 病毒重定位代码对应的 Kripke 结构

3.2 基于 LTL 的恶意行为描述

人们主观地按照程序是否具有恶意性, 将程序分为正常程序和恶意程序 2 类, 但程序在没有执行之前以二进制代码存放起来, 它本身并没有正常和恶意之分, 程序的安全性在程序执行的过程中, 通过程序的“行为”表现出来。可执行程序都由若干“行为”构成, 当执行完这些“行为”后, 也就实现了程序的功能。程序的“行为”是抽象的概念, 具体的“行为”需要根据实际情况而定, 它可以是程序的结构特征, 可以是一段二进制代码, 可以是指令序列, 可以是 API 函数调用关系, 也可以是上述这些“行为”的组合。

本文以典型的蠕虫 Worm.Japanize 为例, 描述其通过搜索注册表查找用户的 Outlook 地址簿文件路径的行为, 其部分汇编代码如下:

```
CODE:00401188  Push offset "Default Mail Account"
CODE:0040118D  push ds:hKey
CODE:00401193  call RegQueryValueExA
...
CODE:004011F6  push offset "SMTP Email Address"
CODE:004011FB  push ds:hkey
CODE:00401201  call RegQueryValueExA
...
CODE:0040121D  Push offset "Software\Microsoft\
WAB\WAB4\Wab File Name"
CODE:00401222  push 8000001h
CODE:00401227  Push RegOpenKeyExA
...
```

蠕虫首先在注册表中查找是否拥有名为 Default Mail Account 的值, 从而得到当前用户的外发邮件地址, 之后查询默认外发邮件的 SMTP Server 地址, 最后查找用户的

地址簿文件路径, 通过以上 3 个步骤完成 Outlook 地址簿文件地址的查找。对该行为的 LTL 描述如下:

```
CODE:00401188  Push offset "Default Mail Account"
CODE:0040118D  push ds:hKey
CODE:00401193  call RegQueryValueExA
...
CODE:004011F6  push offset "SMTP Email Address"
CODE:004011FB  push ds:hkey
CODE:00401201  call RegQueryValueExA
...
CODE:0040121D  Push offset "Software\Microsoft\
WAB\WAB4\Wab File Name"
CODE:00401222  push 8000001h
CODE:00401227  Push RegOpenKeyExA
...
```

3.3 恶意行为识别

恶意行为识别阶段主要是对程序做形式化验证, 由于 LTL 公式(性质取反)能够转化为一个表示无穷状态序列的 Buchi 自动机^[8], 使用 Kripke 结构描述的程序模型能够对应于有限状态自动机, 因此模型检测问题最终转化为乘积自动机的判空问题, 即检查乘积自动机所接受的语言是否为空, 该问题又可进一步归结为深度优先搜索算法检查有向图的强连通分支, 当存在强连通分支时, 表示程序不满足给定的属性规约。

在具体进行模型检测时, 可基于深度优先搜索算法思想, 将每个强连通分支看作是搜索树中的一棵子树, 搜索过程中把当前搜索树中未处理的节点加入一个堆栈, 回溯时可以判断栈顶到栈中节点是否为一个强连通分支。如存在该强连通分支, 则可验证目标二进制文件中包含该恶意行为。在完成行为检测的同时, 在控制流图上对检测到的行为进行标注, 以便进一步分析程序。

4 实验与结果分析

实验环境: 运行系统 Windows XP Professional SP3, 硬件: Intel Core(TM)i5 CPU 760 @ 2.8 GHz, 4 GB RAM。

本文通过分析典型的恶意程序代码, 抽取了它们在感染和传播时常用的行为, 如表 1 所示。

表 1 程序常用行为归类	
编号	行为描述
B1	目标搜索
B2	内存分配
B3	注册表操作
B4	加密、解密操作
B5	网络连接操作
B6	系统配置

实验中选择了 10 个二进制程序来测试原型系统, 包括 5 个正常程序和 5 个恶意程序。正常程序都来自新安装的 Windows XP 系统 Windows 目录下的 PE 文件; 恶意代码实验样本均来自 VX Heavens。表 2 给出了测试结果,

可以看出，从恶意程序中检测出的行为普遍较多，而正常程序里含有的行为较少甚至没有。

表 2 常用行为测试结果

正常程序	B1	B2	B3	B4	B5	恶意程序	B1	B2	B3	B4	B5	B6
ftp					✓	SpyBot					✓	
PDF						Matrix	✓		✓	✓		✓
calculator						Fighter	✓				✓	✓
regedit			✓			Aztec	✓	✓	✓			✓
winrar				✓		Hatred	✓	✓			✓	✓

下面对已知的恶意程序及其变种共 35 个进行检测，并与 3 款常用杀毒软件进行对比测试。使用的杀毒软件版本如下：卡巴斯基 KAV2012，版本号为 12.0.0.374；ClamAV，版本号为 0.97.2；瑞星 2011，版本号为 23.00。病毒库更新时间为 2011 年 11 月 16 日。在测试用例中，1 版表示插入垃圾代码，2 版表示修改数据段，3 版表示修改控制流，4 版表示相同语义指令替换。测试结果如表 3 所示。

表 3 混淆程序测试结果

名称	Kaspersky	ClamAV	瑞星	本文系统	名称	Kaspersky	ClamAV	瑞星	本文系统
AnalBeeds	✓	✓	✓	✓	demiurg3	✓			
AnalBeeds1				✓	demiurg4	✓			✓
AnalBeeds2	✓			✓	hume	✓	✓		✓
AnalBeeds3					hume1		✓		✓
AnalBeeds4				✓	hume2		✓		✓
BOLZANO	✓	✓	✓	✓	hume3		✓		
BOLZANO1		✓		✓	hume4		✓		✓
BOLZANO2	✓	✓			icebat	✓	✓		✓
BOLZANO3		✓		✓	icebat1	✓			✓
BOLZANO4		✓		✓	icebat2	✓			✓
cvaw	✓	✓	✓	✓	icebat3	✓			
cvaw1				✓	icebat4	✓			✓
cvaw2	✓	✓		✓	yildiz	✓	✓	✓	✓
cvaw3	✓	✓			yildiz1				✓
cvaw4	✓	✓		✓	yildiz2				✓
demiurg		✓	✓	✓	yildiz3	✓			
demiurg1	✓			✓	yildiz4				✓
demiurg2	✓		✓	✓					

其中，Kaspersky 的检测总数为 20；ClamAV 的检测为 18；瑞星为 6，本文系统为 28。从上述结果可以看出，常用杀毒软件对恶意程序的原始版本识别率较高，但是对经过变形的恶意程序识别率不高，而本文的原型系统由于能够有效分析可执行文件中所具有的恶意行为，因此对变形后的恶意程序识别率最高。

5 结束语

本文针对可执行程序中恶意行为的识别问题，提出了一种基于模型检测的识别方法，测试结果证明，本文方法不仅能够识别程序常用的行为，而且相比杀毒软件能够更有效地识别经过混淆的恶意程序中包含的行为。但是在识别程序行为之后，如何根据程序行为对程序的安全性做出有效的评估是下一步需要研究的内容。

参考文献

[1] Bergeron J, Debbabi M, Desharnais J, et al. Static Detection of Malicious Code in Executable Programs[C]//Proc. of Symposium on Requirements Engineering for Information Security. Indianapolis, USA: [s. n.], 2001.

[2] Christodorescu M, Jha S, Seshia S A, et al. Semantics-aware Malware Detection[C]//Proc. of IEEE Symposium on Security and Privacy. Oakland, USA: IEEE Press, 2005: 8-11.

[3] 陈超, 李俊, 孔德光. 模型检测迷惑二进制恶意代码[J]. 计算机工程与应用, 2008, 44(15): 61-63.

[4] Schmerl S, Vogel M, König H. Using Model Checking to Identify Errors in Intrusion Detection Signatures[J]. International Journal on Software Tools for Technology Transfer, 2011, 13(1): 89-106.

[5] 何恺铎, 顾明, 宋晓宇, 等. 面向源代码的软件模型检测及其实现[J]. 计算机科学, 2009, 36(1): 267-272.

[6] Clarke E M, Grumberg O, Peled D A. Model Checking[M]. Cambridge, USA: MIT Press, 1999.

[7] Cifuentes C, Emmerik M V. UQBT: Adaptable Binary Translation at Low Cost[J]. Computer, 2000, 33(3): 60-66.

[8] 董威. 并发和实时系统的模型检验技术[J]. 计算机研究与发展, 2001, 38(6): 698-705.