基于开源JVM的安全策略强制实施

魏 达[1], 金 英[1], 张 晶[1], 郑晓娟[2], 李 卓[1]
(1. 吉林大学计算机科学与技术学院, 吉林长春 130012; 2. 东北师范大学软件学院, 吉林长春 130117)

摘 要: 非信任代码的安全执行是移动代码安全的重要问题之一. 携带模型代码(Model Carrying Code)方法同时考虑了移动代码生产者和使用者对安全性的支持和需求, 建立了以模型为中心的安全执行非信任代码的理论框架, 其中安全策略的定义和强制实施是MCC方法的重要组成部分之一. 本文针对已被广泛使用的Java移动代码, 以开源JVM Kaffe和Linux操作系统为研究载体, 提出了基于开源JVM的安全策略实施模型, 并实现从安全策略定义到实施的整个过程. 本文在安全策略规范描述, 可强制实施的扩展有限自动机(EFSA)模型和进程级监视以捕获系统调用等方面都做了有益的尝试, 为完善MCC方法和实现安全策略的强制实施提供很好的方案.

关键词: 携带模型代码; 安全策略描述和实施; 开源JVM; 进程级监视
中图分类号: TP311 文献标识码: A 文章编号: 0372-2112 (2009) 4A-036-06

Enforcing Security Policies in Open Source JVM

WEI Da[1], JIN Ying[1], ZHANG Jing[1], ZHENG Xiao-juan[2], LI Zhuo[1]
(1. *College of Computer Science and Technology, Jilin University, Changchun, Jilin* 130012, *China*;
2. *Software College, Northeast Normal University, Changchun, Jilin* 130117, *China*)

Abstract: Safe execution of untrusted mobile code is one of the most important problems in mobile code security. Model Carrying Code(MCC) provides a model-centered framework for safe execution of untrusted mobile code by taking both code producers and code consumers into consideration. Security policy specification and enforcement are important parts of MCC. An approach to formal specification and runtime enforcement of security policy has been proposed and implemented for Java mobile code running on open source JVM Kaffe and Linux operating system. Some useful works have been done on specification of security policies, the enforceable Extend Finite State Automata(EFSA) model and the method of monitoring on process level to catch system call information. It is indicated that our approach present a effective way to support MCC and implement security policy enforcement.

Key words: model carrying code; specification and enforcement of security policies; open source JVM; process level monitoring

1 引言

随着Internet的迅猛发展, Java等一些移动代码的广泛应用使得对其安全性的要求越来越高, 因此, 研究移动代码的安全性有着很重要的现实意义. 携带模型代码(Model Carrying Code)方法[1]是由Stony Brook大学的R Sekar等提出, 其主要思想是: 移动代码的生产者提供代码安全相关的行为信息(模型), 移动代码的使用者可以形式地对该信息进行推理, 检查是否与自己的安全策略相符合, 如果符合, 就执行代码, 如果有冲突, 则代码使用者可以精化自己的安全策略, 特别地, 还支持运行时对安全策略的强制实施[2].

MCC框架包括以下几个部分, 如图1所示:

(1) 程序安全相关的行为信息(模型)的描述及获取;

(2) 安全策略的规范描述;

(3) 模型的形式化验证;

(4) 运行时的安全策略强制实施.

在MCC方法中提到实施的内容有两种: 一种是按模型强制实施; 另一种是实施安全策略. 模型验证部分即是考察安全行为模型是否满足安全策略的过程, 即判断如下关系:

$$B[M] \subseteq B[P]$$

其中 M 表示安全行为模型, P 表示安全策略, B 是从模型(或策略)到满足该模型(或策略)的行为序列的集合的映射. 因此实施安全策略比实施模型应用范围更广; 实施模型将比实施安全策略更严格, 另一方面, 由于本文针对Java, 模型所代表的行为序列即为Java方法序

收稿日期: 2008-09-19; 修回日期: 2008-12-15
基金项目: 国家自然科学基金项目青年基金(No. 60603031)

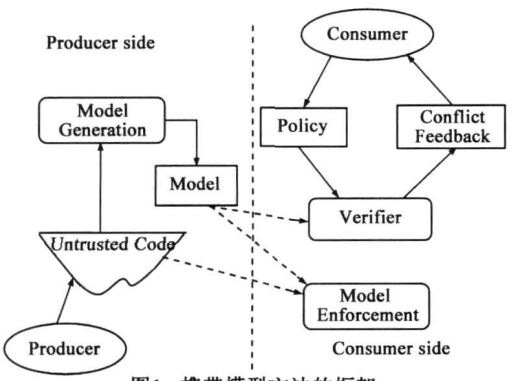

图1　携带模型方法的框架

列, 安全策略多由系统调用序列描述, 然而无论是模型还是安全策略, 他们的组织形式是一样的, 实施的方法和过程也都是一样的, 所以本文将以实施安全策略为例进行说明.

对程序的运行时监控能够强制实施安全策略[3], 本文给出了一种基于开源 Java 虚拟机 Kaffe 的强制实施安全策略的方法, 并且实现了相关的运行环境, 包括安全策略定义和 Java 代码的执行监控程序. 本文主要贡献包括:

(1) 通过修改虚拟机源码实现 Java 代码的执行监控. 通常实现执行监控的方法有两种: 一种是通过重写 Java 字节码来获得方法调用信息, 这种方法不仅需要修改字节码, 而且因为字节码多数经过加密, 因此必然会增加类的载入时间; 另一种方法是通过封装运行环境完成运行监控. 本文采用的是在虚拟机的级别上实施安全策略, 不仅能够更有效的控制 Java 的执行情况, 而且并不会带来很多效率上的代价;

(2) Linux 操作系统上的进程级监视技术是用于解决 Java 代码在进行本地方法调用时无法监控的一项关键技术, 弥补了虚拟机监视在 Java 本地方法调用时的不足;

(3) 基于规范的入侵检测技术[4]在描述和实施安全策略方面起着指导性作用, 通过引入集合类型的变量, 大大提高了变量的描述能力.

2　Kaffe 介绍

Kaffe 是一种开放源码的 Java 虚拟机, 通过阅读有关文档和源码(版本 kaffe-1.1.8)[5], 可了解其在解释执行 Java 字节码的主要过程如下(见图 2): 如果某 Java 方法是本地方法, 那么 Kaffe 将其代码映射到本地机器上执行; 否则, Kaffe 依次取出其操作码进行解释(在 Kaffe 函数 runVirtualMachine 中). Java 的指令集包含约有 203 种操作码, 本文主要关注其中的方法调用和方法返回指令. Kaffe 采用的是用一种间接递归的方式处理 Java 方法调用和返回, 因此, 若要窥视某 Java 程序在 Kaffe 虚拟机上的执行情况, 只需在 Kaffe 函数 virtualMachine() 中调用 runVirtualMachine() 的地方插入监视代码.

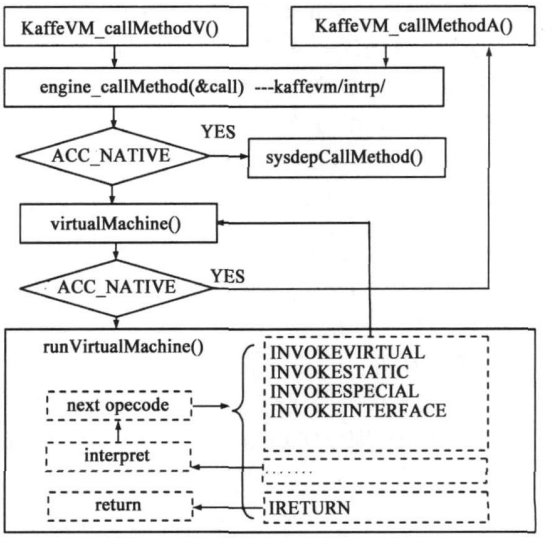

图2　Kaffe解释指令流程（部分）

安全策略形式定义

本文把安全策略看做是监控一个程序的执行使其行为不会破坏系统安全的准则. 从执行的角度看, Java 程序的行为可以粗略的看成是方法调用序列, 然而只有那些调用系统调用的方法才构成系统的潜在威胁. 因为系统调用是操作系统内核为程序提供服务的接口函数, 程序只有通过系统调用改变系统状态. 为了消除一些不正当的系统调用序列可能给系统带来的威胁, 一个基于模式的检测方法被应用到系统安全领域当中[4], 这一方法的核心是如何定义和实施安全策略.

如果将系统调用抽象为事件, 那么系统调用参数就成了事件参数, 在系统调用序列中, 一些系统调用的规律性出现将被理解成一种模式. 反过来, 这样的模式就能够匹配许许多多系统调用序列, 以检查这些系统调用序列是否安全. 这便是基于模式的检测方法的核心思想.

本文在 BMSL 语言[4]的基础上提出安全策略形式定义.

.1　事件与变量

在 UNIX/Linux 上, 系统调用是通过软中断实现的. 在系统调用前, 程序将系统调用号和参数放入到 CPU 寄存器中(以 X86 体系结构为例), 待中断返回, 程序从 CPU 寄存器中读取系统调用的返回值.

在 Kaffe 虚拟机上, Java 的方法调用是通过调用虚拟机函数 virtualMachine() 实现的(见图 2), 方法调用返回就是 virtualMachine() 函数的退出. 因此, 无论是系统调用还是 Java 方法调用, 都是可以获得其调用名和参数, 将系统调用和方法调用抽象为事件也是可以的.

变量主要用于记录事件及其参数值等信息, 也就是安全策略的实施环境. 变量在作用域上分为全局变量和局部变量, 全局变量的作用域是整个安全策略, 局部变量的作用域是某一模式. 正因为变量的作用, 模式一方面表现了事件出现的次序关系, 另一方面表现了事件参数之间的数据流关系.

变量有三种类型: 分别是整数型(int) 、实数型(real) 和字符串型(string) . 此外, 本文根据实际情况, 允许变量声明为集合形式, 集合的元素可以是上述四种类型中某种类型的数据, 从后面例中可以看到集合变量的应用是很广的.

.2 模式与规则

鉴于模式是用来描述程序的执行表现在系统调用的规律性出现的, 因此可以用定义在事件上的正则表达式(REE) 形式化的描述模式[4]. 具体的说, 模式就是事件用符号序列(•) , 并列(‖) , 闭包(*) 和括号连接在一起的表达式, 当然单个事件也是模式. 当程序的执行匹配了为其定制的某一模式, 监控程序将采取包括终止其执行在内的手段来保护系统安全, 这便是规则. 安全策略是由一系列规则组成的, 规则的左边是模式, 右边是监控程序所执行的动作. 下面是安全策略语言模式部分的文法:

```
Rule ::= PatternExp → Action
PattemExp ::= PrimPat
            | (PatternExp)
            | PatternExp*
            | PatternExp1. PattemExp2
            | PatternExp1 | | PattemExp2
PrimPat ::= ! PrimPat | PrimPat
PrimPat::= EventName EventArgs MatchCond StmBody
EventName ::= @id| id
EventArgs ::= ε| (ArgsList)
MatchCond ::= ε| | CondExp
StmBody ::= ε| [StmList]
Action ::= $ id(ExpList)
```

“@ ev(retval, …) ”表示需要取得事件 ev 的返回值, 存入 retval 变量中.

“ev (…) | CondExp” 表示只有当表达式 CondExp 的值为真时, 事件 ev 才能匹配, 称为事件参数条件, 或事件条件.

“ev(…) [StmList] ”表示当事件 ev 匹配模式成功后, StmList 语句序列将被处理.

“Action” 表示模式被成功匹配后, 监控程序需要做的一些事情, 通常执行预定义的一些函数, 包括写入监控记录, 查看系统数据, 反馈信息给用户等, 最后可能终止被监控程序的执行.

“$ func(…)”表示执行预定义函数 func( ) , 这些函数是监控程序给出的, 安全策略通常会用到的函数. 包括字符串操作函数; 集合操作函数; 紧急处理函数等, 其中有一个重要的紧急处理函数就是_ term( ) , 终止被监控程序的运行.

有关文件的创建与删除的安全策略的例子见图 3.

```
string wrDirs[ ] = {“/ home/ user/ work/” }; / / 允许写文件目录列表
string crFiles[ ] = {}; / / 程序创建的文件列表
/ / 规则一: 在不允许写的目录的文件中写入, 终止.
@ open( fd, file) | ! $ set_ exist( wrDirs, $ str_ getDir( file) ) . write( fd)
        → $ _ term( );
/ / 规则二: 将程序创建的文件添加到 crFiles 集和中
( open( file, f) | f = = fl O_ CREAT && fl = = fl O_ TRUNC[ ] ‖ creat( file) )
        → $ set_ add( crFiles, file);
/ / 规则三: 删除或清空不是自己创建的文件, 终止.
umlink( file) | $ set_ exist( crFiles, file) → $ _ term( );
```

图 3 访问控制策略的例子

根据实际情况, 定义如下三条隐式规则:

(1) 在将系统调用参数赋值给变量前要将变量值备份, 如果检查事件条件不满足, 则要恢复变量值;

(2) 如果规则中没有明确指示事件返回值的条件, 那么在事件(系统调用) 成功执行的情况下匹配;

(3) 考虑序列运算符(•) 后面出现的事件, 如果没有明确指示参数条件, 那么条件为变量当前值是否等于备份值.

4 安全策略的实施方法和实现过程

本文通过修改 Kaffe 源码实现 Java 代码的执行监控, 并且给出安全策略实施框架: 安全策略首先用一种基于事件的正则表达式形式定义出来, 然后经过编译转换成扩展自动机 EFSA 的中间表示, 最后在监控程序的辅助下强制实施. 进程级的监视对于在 Java 层面所看不到的 Linux 系统调用部分仍然是有效的.

整个安全策略的实施过程如图 4 所示: 监视进程(父进程) 首先启动, 在读取安全策略之后把它编译成扩展有限自动机(EFSA) 的形式, 利用 fork( ) 和 exec( ) 系统调用启动经过修改的 Kaffe. 此时, 监视进程和 Kaffe 进程协同工作: Kaffe 的作用是解释 Java 字节码, 处理可

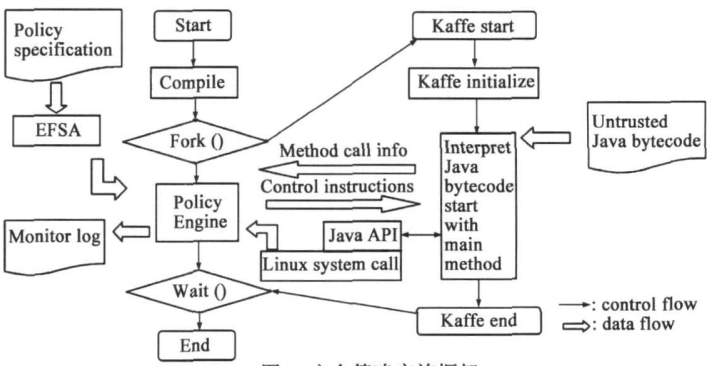

图4 安全策略实施框架

能来自用户的输入和输出; 父进程的作用是监视子进程的运行, 实施安全策略, 可能对子进程发出强制措施的指令, 并写入监控记录, 直到子进程终止, 父进程退出. 父子进程互通消息是通过共享内存和信号实现的.

4.1 EFSA

扩展有限状态自动机(EFSA) 是与 REE 等价的一种表示方法[4], 用于安全策略的强制实施. EFSA 定义为一个 7 元组$(Q, q0, F, V, Env, \Sigma, \delta)$:

- Q 有限状态集合
- $q0$ 初始状态
- F 终止状态集合
- V 状态变量有限集
- Env 所有可能环境的集合, 每个环境为状态变量可能的取值情况, 变量值改变意味着环境更新
- Σ 输入符集, 这里代表事件
- δ 转换函数: $Q \times \Sigma \times Env \rightarrow 2^{(Q \times Env)}$

定义中说明了 EFSA 是一个不确定的状态机, 而事实上也不得不这样, 因为模式的基本组成是事件, 事件不仅包含事件名及参数, 而且包含了匹配条件, 所以同一事件名可以在不同的模式中出现, 由于其匹配条件不同我们将其看成不同的事件, 至于是否要匹配该事件, 是需要动态获得相关参数后才能确定的.

将安全策略由正则表达式形式转化为自动机形式的理由有二: 一是为了提高实施效率, 如果用正则表达式形式, 监控程序将动态确定预测符集, 效率低下; 二是为了简化问题, 安全策略转化为自动机后, 监控程序只需要几次查表便可以确定实施安全策略情况.

本文定义的 EFSA 的内部表示有以下一些表结构:

(1) 变量表: 包括全局变量和局部变量, 解决变量命名冲突之后, 用统一编号标识;

(2) 事件表: 所有在策略中出现的事件列表, 包括事件名, 参数, 事件触发条件和事件匹配后所要执行的语句;

(3) 终止状态表: 每个终止状态关联到一个用于处理当模式匹配成功时的动作函数;

(4) 当前状态表: 由于自动机的不确定性, 需要一个集合表示当前状态, (3) 和 (4) 的交集如果不空, 说明某一模式匹配成功, 当前状态表初始化为仅包含初始状态;

(5) 状态转换矩阵: 提供输入符和当前状态到下一状态(也可能是几个状态) 的映射;

安全策略的编译过程是将安全策略(经过词法和语法检查) 转换为内部表示(各种表结构) 的过程, 其中最主要的是创建状态转换矩阵的过程. 在这个过程中用到从正则表达式到 NFA 的算法, 主要用到的技术是创建动态大小的表和表合并技术. 算法的思想是用递归的方式为模式表达式建立状态转换矩阵, 根据模式表达式间的不同连接符号和优先级, 用不同的规则合并这些矩阵. EFSA 为安全策略的中间表示提供了理论依据, 也为安全策略的高效实施提供现实可行的办法.

图 3 中的例子转换为EFSA 的形式如表 1~ 4 所示:

表 **1**　变量表

NO	NAME	TYPE	VALUE
G1	wrDirs	S[ ]	{" / home/ user/ work/" }
G2	crFiles	S[ ]	{ null}
R1_ 1	fd	I	0
R1_ 2	file	S	null
R2_ 1	file	S	null
R2_ 2	f	I	0
R3_ 1	file	S	null

表 **2**　事件表

NO	NAME	ARGS	COND
E1	open	R1_ 1, R1_ 2	! $ set_ exist(G1, $ str_ getDir(R1_ 2))
E2	write	R1_ 2	null
E3	open	R2_ 1, R2_ 2	R2_ 2= = R2_ 2| O_ CREAT&&R2 2! = R2_ 2| O_ TRUNC
E4	creat	R2_ 1	null
E5	unlink	R3_ 1	$ set_ exist(G2, R3_ 1)

表　状态转换矩阵(S_0^0 为初始状态)

	E1	E2	E3	E4	E5
S_0^0	S_1		S_3^*	S_3^*	S_4^*
S_1		S_2^*			

表 **4**　终止状态表

FIN_ STATE	ACTION
S_2^*, S_4^*	$ _ term( )
S_3^*	$ set_ add(G2, R2_ 1)

4.2 监控程序的实现

本文最终要实现的是一个监控程序, 是基于 Linux 操作系统上的用户级的一个进程监视另一个进程. 监控程序的实现分三步:

第一步, 修改 Kaffe 源码, 重新编译 Kaffe. 修改的地方有二: 一是在 Kaffe 执行 Java 主方法之前和之后分别发送信号给监视进程, 使监视行为能在适当的时候开始和停止; 二是分别在 Kaffe 解释 Java 方法调用指令和返回指令的地方插入代码, 以获得方法信息并能够发送给监视进程.

第二步, 用 ptrace( ) 系统调用建立进程监视机制. 目的是监视 Java 进行系统调用的情况, 使 Java 在进行本地方法调用时依然处于被监视状态.

第三步, 实现事件的匹配, 自动机转变的具体细节.

5 实验分析

采用执行监控方案的安全策略的强制实施虽然消除了一定的安全隐患, 但对于执行效率是有影响的. 首先程序产生的每一个系统调用都会产生两次用户级和内核级的进程上下文切换, 导致处理器对进程的重新调度, 但这种代价从长远来看是一个定值, 设为 Ts, 不会因系统调用的复杂程度而改变; 其次是有些被安全策略关注的系统调用会涉及到自动机的跳转, 设为 Tm. 假设被监控的程序产生 N 次系统调用, 其中有 M 次系统调用是被关注的, 那么监控程序所带来的时间成本是:

$$T = N \cdot Ts + M \cdot Tm$$

本文在 Intel Pentium 4, 2.80GHz 的硬件环境下, 操作系统内核为 Linux 2.6.24-1-686 的机器上进行研究测试, 结果表明 Ts 的值分布在 13~14 微秒之间, 所以把性能改进的重点放在 Tm 上. Tm 值由几部分组成: 首先, 从当前状态表出发, 搜索每一条边, 找出相匹配的事件; 其次, 匹配事件参数, 检查事件条件是否满足; 最后更新当前状态表.

这里面有一个事实是: 每一个系统调用都是在调用时截获一次, 在返回时截获一次, 调用与返回之间不会再产生其它的系统调用, 所以维护一个临时边(即 EFSA 中的跳转) 表, 在系统调用发生时将需要取得返回值的边放入临时边表中, 待系统调用返回, 依次取走所有的边. 这样做可以避免重复搜索, 也可以使事件的条件只检查一次.

针对图 3 给出的安全策略的例, 对若干自己编写的 Java 程序进行效率分析, 对比 Java 代码中系统调用的执行时间比例与监视成本的变化情况, 其中系统调用比重是人为控制的, 监视成本的计算方法为:

$$C = \frac{Tu - To}{To}$$

Tu 为程序在被监视下的执行时间, To 为没有被监视的程序执行时间. 统计结果如表 5 所示. 在程序中系统调用比重不是很大情况下, 监视的效率是很高的.

表 5 系统调用比重和监视成本对照表

系统调用比重	80% 以上	50%	30%	10%
监视成本	33%	29.5%	15.3%	5%

在实用性和有效性方面, 通过各种针对策略的 Java 代码的监控测试表明, 监控程序完全能够监测出程序违反策略的情况, 而对于没有违反策略的情形, 监控程序也没有错误的提示. 然而, 实验还有一定的片面性, 实验程序是针对具体的安全策略而写的, 本文以后任务将对一些实际的攻击做出安全策略方面的研究.

6 相关工作

在安全策略的规范定义方面: 基于可信任系统(TOS) 上整合安全策略的强制实施[6] 提出安全策略不仅需要关注某单一操作的访问控制, 而应该把重点放到攻击者的行为语义控制上, 也就是安全相关的操作序列, 这和本文的思想十分接近. 同样基于角色的访问控制也是安全策略的另一重要组成部分, 所以称之为整合安全策略. 整合安全策略定义的实体很多: 包括用户、角色、客体、域、类型、事件等, 策略也分两种: open 和 closed. 与其复杂形式相比, 本文更趋向于从语言(语法和语义) 的角度描述安全策略.

在安全策略的实施方面: MCC 方法[1] 对于强制实施模型部分采用操作系统内核级的监控措施, 在内核模块的层面上劫持系统调用, 达到强制实施的目的. 整合安全策略的强制实施[6] 也应用同样的方法.

在信息流控制系统上实施策略[7] 是从跟踪和控制信息流的角度监控系统安全执行的, Trishul 是在开源 Java 虚拟机 Kaffe 上实现的信息流控制系统, 解决了跟踪隐式信息流的难题, 对于面向数据的策略实施框架提供强有力的支持. 这与本文所进行的事件参数信息跟踪和检查是一致的, 不同的是本文把事件行为序列作为主要关注方面.

在移动设备上的策略实施框架[8] 主要介绍了在手持设备(如 PDA) 上的身份认证, 信任连接等安全机制, 实现了对设备自身和对网络服务的安全访问. 该框架采用 XML 语言描述访问控制策略和在无线网络环境下的安全控制, 对本文的未来工作有借鉴意义.

7 总结

本文以开源 Java 虚拟机 Kaffe 为载体, 采用了 Linux 上一个进程监视另一个进程的技术, 在 MCC 框架下, 借鉴了基于规范的入侵检测理论, 完成了安全策略的强制实施. 在安全策略的形式定义中引入集合类型的变量, 使环境变量的功能更强大, 给出安全策略的形式语法定义和语义说明, 使安全策略的描述工作更容易. 在安全策略的编译和实施过程中应用表处理技术, 使实施过程更简洁、高效. 通过实际例子验证本文方法可行有效.

本文也有一些不足之处, 首先, 基于 JVM 的安全策略实施程序是依赖具体 JVM 的, JVM 版本的变化将导致实施程序的需要调整; 其次, 对于 Java 的其它指令, 如线程同步指令, 异常指令, 没有给出监视措施. 今后, 将针对多线程和异常机制完善本文方法, 并且应用到更多实例中.

参考文献:

[1] R Sekar, V Venkatakrishnan, S Basu, S Bhatkar, D DuVarney. Model-carrying code: A practical approach for safe execution of untrusted applications[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 15- 28.

[2] 李泽鹏, 金英, 张晶, 郑晓娟. 基于 Java 平台实现安全行为模型验证[J]. 计算机工程与科学, 2007, 29(10): 7- 10.

[3] Lujo Bauer, Jarred Ligatti, David Walker. More enforceable security policies[R]. Princeton, New Jersey, USA: Princeton University, 2002.

[4] Prem Uppuluri. Intrusion detection/prevention using behavior specifications[D]. NY, USA: State University of New York at Stony Brook, 2003.

[5] Kaffe. org[Z]. URL: http://www. kaffe. org/documentation, 2008.

[6] HC Kim, RS Ramakrishna, W Shin, K Sakurai. Enforcement of integrated security policy in trusted operating systems[M/CD]. Nara, Japan: Springer Berlin/Heidelberg, 2007. 214- 229.

[7] SK Nair, PND Simpson, B Crispo, AS Tanenbaum. A virtual machine based information flow control system for policy enforcement[J]. Electronic Notes in Theoretical Computer Science 2008, 197(1): 3- 16.

[8] Jansen W, Karygiannis T, Korolev V, et al. Policy expression and enforcement for handheld devices[R]. Gaithersburg, Maryland, USA: Computer Security Division Information Technology Laboratory NIST, 2003.

作者简介:

魏　达　男, 1984 年 10 月生于吉林省榆树市, 现于吉林大学攻读计算机软件与理论硕士学位, 研究方向为软件形式化、移动代码安全. E-mail: weidadavid@gmail.com.

金　英　女, 1971 年生于吉林省长春市, 博士, 副教授, 主要从事软件形式化开发方法、软件工程、移动代码安全、程序设计语言及其实现等领域的研究工作. E-mail: jinying@jlu.edu.cn.

张　晶　女, 1975 年生于吉林省长春市, 博士, 讲师, 主要从事软件形式化、软件工程、需求工程、程序分析技术、程序设计语言等领域的科研工作.

郑晓娟　女, 博士, 副教授, 研究方向为软件形式化、程序设计语言及其实现技术、网络安全等.

李　卓　女, 硕士, 研究方向为计算机安全、计算机动漫.