一种针对 JVM 运行时库安全策略的全自动检测方法

吴蓉晖,汪 宁,孙建华,陈 浩,陈志文
(湖南大学信息科学与工程学院,湖南长沙 410082)

摘 要: JVM 运行时库通过调用自身库函数的安全管理器类能够实现多种安全策略,其中非常重要的一条安全策略是保证程序在执行敏感操作之前必须进行相应的访问控制权限检查.传统上依赖于人工分析来确保 JVM 运行时库满足该安全策略,由于 Java 标准类库涵盖上千个类,上万个方法,且处于快速发展和演化过程中,人工分析费时费力,容易出错.本文提出一种全自动、高效、快速的模型检测方法评估 JVM 是否遵守这一安全策略,扫描 Java 标准类库字节码文件,将类的成员方法生成控制流图,通过定义检验模型,结合污点分析计算出方法摘要,自动检测出风险方法.

关键词: 安全策略; 控制流图; 污点分析; 方法摘要
中图分类号: TN309 **文献标识码:** A **文章编号:** 0372-2112 (2013)01-00161-05
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.01.028

A Full Automatic Detection Method for Security Policy of JVM Run-Time Library

WU Rong-hui, WANG Ning, SUN Jian-hua, CHEN Hao, CHEN Zhi-wen
(*School of Information Science and Engineering, Hunan University, Changsha, Hunan* 410082, *China*)

Abstract: JVM run-time library can implement various security policies by calling the library functions of its own, one of the extremely important security policy requires that sensitive operations must be performed after access control permissions checks. Traditionally it relies manual analysis to ensure that the JVM run-time library satisfy this security policy. Java standard library, covering thousands of classes, tens of thousands of methods, is in rapid development and high-rate evolution. It is time-consuming and error-prone to analyze the security policy artificially. This paper presents a full automatic, efficient and rapid model detection method for evaluating that whether the JVM in compliance with this security policy. Scanning the byte code files of Java standard class library, generating control flow graph of the member methods, our method can work out method summary by taint analysis after defining detecting model and automatically detect the risky methods.

Key words: security policy; control flow graph; taint analysis; method summary

1 引言

Java 平台在底层拥有自己的虚拟机(JVM),Java API 类库运行在 JVM 之上,其上可以运行各种容器和工具集,最上层运行 Java 代码编写的应用程序.如果 Java 程序不进行任何访问权限检查,允许用户随意访问系统资源,将会使本地资源遭遇风险,比如文件系统的破坏、私密数据的泄露.Java 平台利用运行时监控[1]机制,对正在访问的系统资源(如网络、文件系统等)实时监控,保证了其安全性.JVM 首先授权访问各种底层资源,然后执行一种特定的安全策略,保证在 Java 平台与操作系统交互时,敏感的系统资源被访问前都要经过访问控制权限的安全检查.

传统的检测方法往往依赖于手工添加程序断点进行验证.但是由于 Java 标准类库过于庞大,且更新较快.导致工作量大,容易出错.近年来出现了一些半自动的检测方法,能够把所有不遵守安全策略的类的成员方法标定为风险方法输出,但是输出结果存在很多误报,后期仍需借助大量人工分析.

本文提出了一种高效的检验模型,能够自动检验出未经安全检查就直接访问敏感操作的所有路径,通过使用静态分析技术[10]和污点分析[4]技术,标记出所有被访问到的风险本地方法和所有直接或间接调用了该本地方法的成员方法.最终自动实现了从 Java 字节码的

收稿日期:2011-08-03;修回日期:2012-07-26
基金项目:国家自然科学基金(No.60803130,No.61173166);湖南大学"青年教师成长计划"

扫描到输出待评估风险方法及其对应的敏感操作和本地方法的过程,减轻了人工分析的工作,更加精确地评估了 JVM 虚拟机的这种基于安全管理器的安全策略的可靠性.

2 安全策略检测的研究现状

JVM 通过一套安全策略保证 Java 程序在与操作系统交互的时候,即调用内核函数时,不会对系统及用户造成风险.规定在进行敏感操作之前,必须进行访问控制权限的安全检查.敏感操作的表现形式即本地方法,是一些与硬件平台和操作系统相关的特定代码,比如一些由 C/C ++ 语言书写并被 Java 调用的方法.对于系统底层资源,JVM 设置了默认的访问控制权限,程序员可以修改这些资源的访问控制权限.然后执行一种特定的安全策略,它是通过实现其内部的安全管理器[2]类对象实现的,即在调用本地方法等敏感操作前,需要执行访问控制[3]权限的安全检查,访问控制通常用于系统管理员控制用户对服务器、目录、文件等网络资源的访问.

计算机软件安全测试领域中常用的检测技术有:动态分析[13]、静态分析、污点分析等.目前有很多安全检测方法和检测工具(Nessus[14],CMV[6],MOPS[7])都融合了上述的一个或多个安全检测技术,其中一部分工具已经能够对 JVM 安全策略的可靠性进行检测分析.目前还没有一种检测方法能够完全实现自动化检测 JVM 运行时库安全策略的可靠性.本文在 CMV 等静态检测工具的基础上,结合污点分析技术排除误报,设计出一种针对 JVM 运行时库安全策略的全自动检测方法(A Full Automatic Detection Method for Security Policy of JVM Run-Time Library,简称为 FADM).

3 FADM 方法流程图

我们的目标是确保 Java 标准类库的是安全可信的,即安全管理器必须处理所有可能涉及敏感操作的方法.常见的快速模型检验工具 CMV 已经可以对 Java API 进行半自动分析,但该工具只能粗略地计算出所有的风险方法,仍然需要涉及大量人工辅助分析才能得出最终结论.因此,结合 CMV 等软件的优点,设计一种更加高效的全自动模型检测方法是十分有必要的.

FADM 能够全自动检测 JVM 安全策略的可靠性,通过读取 Java 标准类库的字节码文件,进行静态分析.然后根据自定义的检验模型,生成类中各成员方法的控制流图[5](简称 CFG),最后,把生成的所有方法的方法摘要以文本形式输出,对出现的风险方法和敏感操作加以标记,全自动检测部分完毕.

方法摘要是一个方法属性的集合,其中记录了判定是否风险方法的重要属性.所谓风险方法,即在方法体中调用了敏感操作,但是并没有进行安全检查,这违背了 JVM 的安全策略.因此,只要计算出方法摘要,就能判断该方法是否风险方法.如果检测出风险方法,说明 JVM 的安全策略并不可靠.反之,能够证明该安全策略的可靠性.

图 1 为我们提出的 FADM 检测方法流程图,首先输入需要检验的类名,程序会自动将该类的字节码文件读取,然后经过静态分析,可以按照检验模型生成 CFG 流图,在分析程序 CFG 流图的基础上,融合程序的数据流信息,完成相关污点标记和判定过程,充分利用了程序的控制流、数据流,计算出类中成员方法的方法摘要,最后输出检测结果.自定义检验模型和方法摘要的计算是本检测方法的核心,将在下文中进行详细阐述.

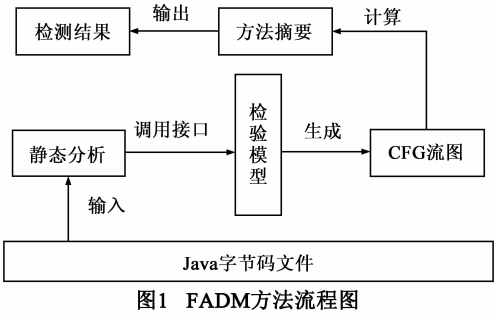

图1 FADM方法流程图

4 检验模型

4.1 增强型检验模型

CMV 通过方法建模,只达到了输出所有风险方法的目的,后期是需要人工分析排除误报的.FADM 意在通过机器分析排除所有误报,找出最终需要分析的风险方法,我们通过引入污点分析技术跟踪标记所有风险方法来解决这个问题.为了实现我们的技术,需要建立一个增强型的检验模型.

FADM 以类为单位对 Java 标准类库进行读取,对每个类对应的字节码文件进行扫描,开始词法、语法的静态分析,然后通过过程内分析,生成类中每个方法的 CFG,最终根据我们的定义检验模型寻找出现风险方法的所有路径.为防止状态空间爆炸,启用方法摘要存储单个方法的状态,确保每一个方法只计算一次.通过污点分析技术标记所有风险方法的传播路径,并计算出他们之间调用的层次关系,输出最终需要分析的风险方法.

4.2 检验模型的定义

FADM 检验模型是把方法的 CFG 流图的每一步抽象成一个节点,并定义节点类型,跟据不同的节点类型进行相应处理,最终能够算出方法的属性.

定义 1 在 CFG 中,把节点分为以下几类:入口节

点,返回节点,安全检查节点,敏感操作节点,方法调用节点,始源节点,中立节点.

定义 2　从方法的 CFG 入口节点到返回节点之间,如果没有安全检查节点,这条路径为不安全路径;否则这条路径成为安全路径.

定义 3　对于方法 M,如果敏感操作节点之前没有出现安全检查节点,把 M 称为风险方法;否则 M 为安全方法.

定义 4　对于每一个风险方法 M,都直接或间接的调用了一个本地方法,这个本地方法即为 M 的始源节点;安全方法的始源节点为空值.

定义 5　本文中出现的字符'*',代表未确定的值;字符'#'代表初始值.

定义 6　方法摘要 $summary<i,j,k>$ 是一个三元组,存储方法属性. i 对应不安全路径(0)/安全路径(1);j 对应风险方法(0)/安全方法(1);k 对应始源节点(*)/空值(null).

定义 7　方法调用节点 M_Node 调用方法 M,如果 $summary(M)=\{*,0,*\}$,则 M_Node 等价于敏感操作节点;如果 $summary(M)=\{1,1,\text{null}\}$,则 M_Node 等价于安全检查节点;如果 $summary(M)=\{0,*,*\}$,则 M_Node 等价于中立节点.

上述的 7 个定义中,定义 2,3 是 CMV 中定义的模型,定义 1,4,5,6,7 是对原模型的扩展.

5　FADM 检测算法的实现

5.1　方法摘要的初始化

FADM 是通过扫描 Java 类的字节码文件,分析出其中的风险方法及其始源节点.根据定义 6,方法摘要中存储了以上两个属性,所以只要计算出方法摘要,就能得到风险方法及其始源节点,就能进一步评估 JVM 运行时库安全策略的可靠性.因此本文提出的检测方法是以计算方法摘要为核心的,方法摘要算法的复杂度直接决定了检测方法效率的高低.为了尽量减少复杂度,首先计算风险方法,然后标定始源节点,需要的时候再计算不安全路径.为保证每一个方法的方法摘要只计算一遍,当计算出方法摘要时,把该方法标记为已被计算过,同时更新方法摘要库.当下次遇到该方法,只需直接读取其方法摘要.

方法摘要计算的准备步骤如下:首先假设所有的方法都是安全方法,对所有的方法摘要进行初始化,初始化方法摘要中的属性为安全路径、安全方法,没有始源节点,即 $summary(For\ All\ methods)=\{1,1,\text{null}\}$,然后访问一个类中的所有访问修饰符为 public 的方法 $M1$, $M2$, $M3$,…,将其 CFG 的入口节点加入到一个队列 Q 中.由于只有具有 public 属性的方法才能被其他类中的其他方法调用,因此我们只考虑计算访问修饰符为 public 的方法.

初始化后,队列 Q 中已经装有了被加入的初始化节点,此时需要从中依次取出节点并读取,按照节点类型进行下一步操作.队列计算方法摘要的算法描述如下:

定义 8　如果有节点 N,则节点所属的方法是 $N.method$.

定义 9　如果有方法调用节点 M,调用了方法 $M1$,如果要求 $summary(M)$,需要把 M 放入节点等待队列 $WQN(M,M1)$,等待求出 $summary(M1)$;如果要求 $summary(M.method)$,需要把 $M.method$ 放入方法等待队列 $WQM(M.method,M1)$.其中 $WQN(M1)$是方法 $M1$ 的等待节点列表,$WQM(M1)$是方法 $M1$ 的等待方法列表.

(1)如果 N 是中立节点,把 N 的继任节点集加入 Q;

(2)如果 N 是敏感操作节点,首先通过污点分析技术,标记出始源节点 R,并把该始源节点与 N 节点相关联.计算 $WQM(T.method,N.method)$和 $N.method$,标记所有涉及到的节点集 T 的 $T.method$ 为风险操作,如果 $N\in\{T\}$,则 $summary(T)=\{*,0,R\}$.否则标记为间接调用,即 $summary(T)=\{*,0,R+\text{“}indrect\text{”}\}$;

(3)如果 N 是方法调用节点,其调用的方法是 $N1$:

(a)如果 $summary(N1)=\{\#,\#,\#\}$,即所有属性尚未计算,执行 $WQM(N.method,N1)$和 $WQN(N,N1)$.

(b)如果 $summary(N1)=\{*,0,*\}$,即已经计算出风险方法属性后,记录下已标记的 $N1$ 的始源节点 $R1$,并更新为 $summary(N1)=\{*,0,R1+\text{“}indrect\text{”}\}$.

(c)如果 $summary(N1)=\{*,\#,*\}$,即风险方法属性尚未计算,执行 $WQM(N.method,N1)$.

(d)如果 $summary(N1)=\{\#,*,*\}$,即安全路径属性尚未计算,执行 $WQN(N,N1)$.

(e)如果 $summary(N1)=\{0,*,*\}$,即已经计算出不安全路径属性后,加入 N 的下一个节点集.

(4)若 N 是返回节点,则更新 $summary(N.method)=\{0,*,*\}$,把 $WQN(N.method)$中的所有方法加入到 Q 中;

(5)如果 N 是安全检查节点,跳过 N 继续读取 Q 中的下一个节点.

根据始源节点调用本地方法的方式,可以将其分为直接始源节点和间接始源节点两种.前者是风险方法直接调用了本地方法,后者是风险方法间接调用了本地方法.本文中的污染源是指类的成员方法中的风险方法和风险本地方法.通过对不信任的输入数据做标记,静态跟踪程序运行过程中污点数据的传播路径,

检测使用污点数据的不安全方式,当检测到风险函数时,利用污点分析可以追踪详细的污点传播过程,得出由于调用污点数据即始源节点所导致的风险函数.

污点标记的流程如图 2 所示,通过对类字节码文件进行词法、语法分析生成 CFG 流图.然后集中处理敏感操作节点和方法调用节点.由于敏感操作节点直接调用了本地方法,所以可以直接记录其始源节点.而方法调用节点往往是调用了其他风险方法,导致自身成为风险方法,所以间接记录其始源节点.然后追踪已被记录的始源节点,通过出入对列,反复标记直接或间接调用了始源节点的风险方法.调用始源节点最后添加所有风险方法的始源节点,并更新方法摘要.所有直接调用始源节点的风险方法就是需要进一步分析的对象.

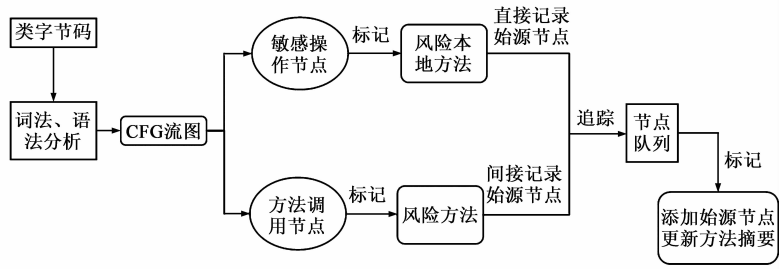

图2 污点标记流程图

5.2 算法性能分析

由于 FADM 是基于方法摘要的,且通过队列反复添加、读取节点进行分析,保证了一个方法的方法摘要只会计算一次,这避免了状态空间的无限增长,保证了算法的线性复杂度,通过使用污染追踪风险方法和风险本地方法的操作,静态标记了所有相关的直接始源节点和间接始源节点.这种全自动的标记检测方法,缩小了需要进一步分析的风险方法的范围,减少了某些半自动检测方法后期需要的人工分析工作,可以对任何版本任何平台的 Java 标准类库进行检测,具有良好的扩展性.

6 实验测试与分析

为检验评估 JVM 运行时库的安全策略的可靠性,搭建了如下测试平台:在 3.00GHz 的双核 CPU,1G 内存的 PC 上安装 Ubuntu9.10 操作系统,其中搭建常用的商用 HotSpot VM 的 JDK1.5 的 Java 平台.使用 soot[11,12]框架进行 Java 字节码的扫描分析工作,对 HotSpot VM 标准类库进行测试.分别使用半自动检测工具 CMV 与本文提出的全自动模型检测方法 FADM 进行测试,得出的实验测试结果如表 1 所示.

实验列举了 Java API 中最常用的 22 个类;所属类包,即这些测试类所在的包结构,涵盖了三大常用包结构(java.io.*,java.net.*,java.lang.Class);JVM 类中的方法,即需要进行检测的类中的所有成员方法;处理的新方法,即方法调用过程中,调用了其他类的成员方法.平均运行时间,即运行检测方法 10 次的平均时间.需要后续分析的方法,即自动检测出的风险方法,还需要进行人工分析排除.

CMV 和 FADM 的实验对比数据如表 2 所示,实验测试一共扫描了 22 个类,其中一共分析了 967 个方法,在所处理的 Java 类和方法完全相同的情况下,FADM 将需要后续分析的方法数量从 56 个锐减到 2 个,且运行时间与 CMV 相比差别不大(检测 22 个类的运行时间只相差 7s).最后联系 java.lang.Class 类中的两个需要后续分析的风险方法的始源节点,发现它们均为 < java.lang.Class forName0(java.lang.String,boolean,java.lang.ClassLoader) > .通过人工判断,发现方法 forname0 和 JVM 的初始化有关,并不是真的风险本地方法,而是一个误报,因此证明了 HotSpot VM 的运行时库的安全策略是安全可靠的.

表 1 实验测试结果

JVM 类	所属类包	JVM 类中的方法	处理的新方法	平均运行时间(s)		需要后续分析的方法	
				CMV	FADM	CMV	FADM
File	java.io	54	197	6	7	2	0
FileInputStream	java.io	17	10	4	4	0	0
FileOutputStream	java.io	18	12	3	4	0	0
ObjectInputStream	java.io	68	56	4	4	3	0
ObjectOutputStream	java.io	59	51	4	4	0	0
RandomAccessFile	java.io	47	36	4	4	0	0
Authenticator	java.net	15	19	4	4	0	0
CookieHandler	java.net	5	3	3	3	0	0
DatagramSocket	java.net	40	57	4	4	23	0
HttpURLConnection	java.net	19	22	4	4	0	0
InetAddress	java.net	42	44	4	4	0	0
MulticastSocket	java.net	18	18	4	4	14	0
NetworkInterface	java.net	19	16	4	5	0	0
ProxySelector	java.net	6	4	3	4	0	0
ResponseCache	java.net	5	3	3	3	0	0
ServerSocket	java.net	32	39	4	4	0	0
Socket	java.net	61	62	4	4	0	0
SocksSocketImpl	java.net	25	31	5	6	0	0
URLClassLoader	java.net	17	73	4	5	0	0
URLConnection	java.net	60	57	4	4	0	0
URL	java.net	36	33	4	5	6	0
Java.lang.Class	java.lang	112	124	4	4	8	2

如表 3 所示,通过对实验结果的分析,得出 FADM 在平均扫描每一个类的时间开销上只比 CMV 多了 0.32s,但是误报率却比 CMV 低了 5.584%.在几乎同样时间开销的条件下,自动分析的正确性远远高于传统的分析方法,且后期人工分析的工作量只有 2 个方法,

而以高效、快速著称的静态检测方法 CMV 需要后期人工分析 56 个方法.显而易见,FADM 在很大限度地减轻了人工分析的工作,避免了不必要的麻烦和错误,在风险方法自动化分析实现方面取得了很好的效果,且不失为一种高效,快速的检测方法.

表 2　实验结果对比

方法对比	JVM 类	分析的方法总数	运行时间(s)	风险方法		后续分析的风险方法
CMV	22	967	87	java.io.File 中 java.io.objectInputStream 中 java.net.DatagramSocket 中 java.net.MulticastSocket 中 java.net.URL 中 java.lang.Class 中	2 个 3 个 23 个 14 个 6 个 8 个	56
FADM	22	967	94	java.lang.Class 中	2 个	2

表 3　实验结果分析

方法对比	平均运行时间(s)	误报风险方法	误报率	实际风险方法
CMV	3.95	56	5.791%	0
FADM	4.27	2	0.207%	0

7　结束语

本文提出了一种高效、快速的全自动的模型检测方法 FADM,能够检查出 Java 标准类库是否满足程序在执行敏感操作之前必须进行相应的访问控制权限检查这一重要的安全策略.所得结果与传统的半自动人工分析方法比对,完全一致.证明了 JVM 运行时库安全策略的可靠性.我们今后的研究重点是扩展本自动化检测模型,可以实现自定义风险操作和安全检查,使该方法能够应用于其他编程语言运行时库的检查和其他程序代码安全性的检查.

参考文献

[1] Irem Aktug, Mads Dam, Dilian Gurov. Provably correct runtime monitoring [J]. The journal of logic and algebraic programming, 2009, 78(5): 262 – 277.

[2] Almut Herzog, et al. Performance of the Java security manager [J]. Computers & Security, 2005, 24(3): 192 – 207.

[3] Koufi V, et al. Context-aware access control for pervasive access to process-based healthcare systems [J]. Studies in Health Technology and Informatics, 2008, 136: 679 – 684.

[4] Zhang X, Edwards A, Jaeger T. Using CQUAL for static analysis of authorization hook placement [A]. Proceedings of the 11th Usenix Security Symposium [C]. San Francisco, CA, USA: USENIX Association, 2002. 33 – 48.

[5] 王学香,浦汉来,杨军.基于扩展控制流图的片上存储器分配策略[J].电子学报,2007,35(8):1558 – 1562.
Wang Xue-xiang, Pu Han-lai, Yang Jun. Performance oriented allocation scheme for scratch-pad memory [J]. Acta Electronica Sinica, 2007, 35(8): 1558 – 1562. (in Chinese)

[6] A P Sistla, et al. CMV: Automatic verification of complete mediation for Java Virtual Machine [A]. Proceedings of ASIACCS'08 [C]. New York, USA: ACM, 2008. 100 – 111.

[7] H Chen, D Wagner. MOPS: an infrastructure for examining security properties of software [A]. CCS'02 Proceedings of the 9th ACM Conference on Computer and Communications Security [C]. New York, USA: ACM, 2002. 235 – 244.

[8] Clause J, Li W, Orso A. Dytan: A generic dynamic taint analysis framework [A]. Proceedings of ISSTA'07 [C]. New York, USA: ACM, 2007. 196 – 206.

[9] Anand S, Pasareanu C S, Visser W. JPF-SE: A symbolic execution extension to Java Pathfinder [A]. Proceedings of TACAS'07 [C]. Heidelberg: Springer, 2007. 134 – 138.

[10] X Fu, X Lu, B Peltsverger, et al. A static analysis framework for detecting SQL injection vulnerabilities [A]. Proceedings of the 31st Annual International Computer Software and Applications Conference [C]. New York, USA: ACM, 2007. 87 – 96.

[11] Sable Research Group. Soot: A Java optimization framework [OL]. http://www.sable.mcgill.ca/soot/tutorial/index.htm, 2010-01-01.

[12] 严俊,郭涛等. JUTA:一个 Java 自动化单元测试工具[J].计算机研究与发展,2010,47(10):1840 – 1848.
Yan Jun, Guo Tao, et al. JUTA: An automated unit testing framework for Java [J]. Journal of Computer Research and Development, 2010, 47(10): 1840 – 1848. (in Chinese)

[13] 王祥根,等.基于代码覆盖的恶意代码多路径分析方法[J].电子学报,2009,37(4):701 – 705.
Wang Xiang-gen, et al. Exploring multiple execution paths for malware analysis based on coverage of codes [J]. Acta Electronica Sinica, 2009, 37(4): 701 – 705. (in Chinese)

[14] Ben H Thacker, David S Riha, et al. Probabilistic engineering analysis using the NESSUS software [J]. Structural Safety, 2006, 28(1/2): 83 – 107.

作者简介

吴蓉晖　女.1967 年生,河南太康人,湖南大学副教授,主要研究方向为网络安全,生物信息技术.
E-mail: wwrh@foxmail.com

汪　宁　男.1985 年生,河北邯郸人.2012 年毕业于湖南大学信息科学与工程学院,获得工学硕士学位.从事 web 网络安全研究.
E-mail: you20@126.com

孙建华(通讯作者)　女.1977 年生,河南焦作人.博士,湖南大学副教授.主要研究方向为网络安全.
E-mail: jhsun@aimlab.org